10 ANEXOS

10.1 LICENCIAS

La licencia que cubre la mayor parte de la distribución de las herramientas que nos ofrece la Fundación de Software Libre es la GNU GPL, Licencia Pública General de GNU (GNU es un acrónimo recursivo para “GNU No es Unix”).

Además existen otras muchas herramientas, no pertenecientes a la Fundación de Software Libre, que se distribuyen también bajo los términos de esta licencia. Existen otras licencias de Software Libre, algunas compatibles con la GPL y otras no.

El echo de ser compatibles o no, a veces implican que puedan tener copyright o algún tipo de patente. Sin embargo siguen cumpliendo los principios básicos del Software Libre en cuanto a ejecución, copia distribución, estudio, cambios y mejoras.

Existen también licencias no libres pero que permiten el uso gratuito de las herramientas aunque imponen fuertes restricciones a su distribución. Cuando no hemos encontrado una herramienta libre adecuada o lo suficientemente potente hemos optado por este tipo de licencias siempre y cuando la herramienta estuviese orientada a uso propio y no distribución.

Incluimos las licencias utilizadas en nuestro desarrollo. Se introduce el texto original, escrito en inglés. El lenguaje legal ha de ser muy exacto y en ocasiones no se ofrecen traducciones oficiales, ya que traducciones oficiales inexactas podrían poner en peligro los términos originales de distribución y uso.

10.1.1 Acuerdo de Licencia de Código Binario de Sun Microsystems, Inc. (Sun BCL) para J2SE 5.0

Licencia no libre. Esta licencia no impone muchas restrinciones para uso particular pero sí para la distribución del software. En nuestro desarrollo hacemos uso del J2SE bajo esta licencia.

specific legal rights. You may have others, which vary from state to state.

5. DISCLAIMER OF WARRANTY. UNLESS SPECIFIED IN THIS AGREEMENT, ALL EXPRESS OR IMPLIED CONDITIONS, REPRESENTATIONS AND WARRANTIES, INCLUDING ANY IMPLIED WARRANTY OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NON-INFRINGEMENT ARE DISCLAIMED, EXCEPT TO THE EXTENT THAT THESE DISCLAIMERS ARE HELD TO BE LEGALLY INVALID.

6. LIMITATION OF LIABILITY. TO THE EXTENT NOT PROHIBITED BY LAW, IN NO EVENT WILL SUN OR ITS LICENSORS BE LIABLE FOR ANY LOST REVENUE, PROFIT OR DATA, OR FOR SPECIAL, INDIRECT, CONSEQUENTIAL, INCIDENTAL OR PUNITIVE DAMAGES, HOWEVER CAUSED REGARDLESS OF THE THEORY OF LIABILITY, ARISING OUT OF OR RELATED TO THE USE OF OR INABILITY TO USE SOFTWARE, EVEN IF SUN HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. In no event will Sun's liability to you, whether in contract, tort (including negligence), or otherwise, exceed the amount paid by you for Software under this Agreement. The foregoing limitations will apply even if the above stated warranty fails of its essential purpose. Some states do not allow the exclusion of incidental or consequential damages, so some of the terms above may not be applicable to you.

7. TERMINATION. This Agreement is effective until terminated. You may terminate this Agreement at any time by destroying all copies of Software. This Agreement will terminate immediately without notice from Sun if you fail to comply with any provision of this Agreement. Either party may terminate this Agreement immediately should any Software become, or in either party's opinion be likely to become, the subject of a claim of infringement of any intellectual property right. Upon Termination, you must destroy all copies of Software.

8. EXPORT REGULATIONS. All Software and technical data delivered under this Agreement are subject to US export control laws and may be subject to export or import regulations in other countries. You agree to comply strictly with all such laws and regulations and acknowledge that you have the responsibility to obtain such licenses to export, re-export, or import as may be required after delivery to you.

9. TRADEMARKS AND LOGOS. You acknowledge and agree as between you and Sun that Sun owns the SUN, SOLARIS, JAVA, JINI, FORTE, and iPLANET trademarks and all SUN, SOLARIS, JAVA, JINI, FORTE, and iPLANET-related trademarks, service marks, logos and other brand designations ("Sun Marks"), and you agree to comply with the Sun Trademark and Logo Usage Requirements currently located at http://www.sun.com/policies/trademarks. Any use you make of the Sun Marks inures to Sun's benefit.

10. U.S. GOVERNMENT RESTRICTED RIGHTS. If Software is being acquired by or on behalf of the U.S. Government or by a U.S. Government prime contractor or subcontractor (at any tier), then the Government's rights in Software and accompanying documentation will be only as set forth in this Agreement; this is in accordance with 48 CFR 227.7201 through 227.7202-4 (for Department of Defense (DOD) acquisitions) and with 48 CFR 2.101 and 12.212 (for non-DOD acquisitions).

11. GOVERNING LAW. Any action related to this Agreement will be governed by California law and controlling U.S. federal law. No choice of law rules of any jurisdiction will apply.

12. SEVERABILITY. If any provision of this Agreement is held to be unenforceable, this Agreement will remain in effect with the provision omitted, unless omission would

frustrate the intent of the parties, in which case this Agreement will immediately terminate.

13. INTEGRATION. This Agreement is the entire agreement between you and Sun relating to its subject matter. It supersedes all prior or contemporaneous oral or written communications, proposals, representations and warranties and prevails over any conflicting or additional terms of any quote, order, acknowledgment, or other communication between the parties relating to its subject matter during the term of this Agreement. No modification of this Agreement will be binding, unless in writing and signed by an authorized representative of each party.

SUPPLEMENTAL LICENSE TERMS

These Supplemental License Terms add to or modify the terms of the Binary Code License Agreement. Capitalized terms not defined in these Supplemental Terms shall have the same meanings ascribed to them in the Binary Code License Agreement . These Supplemental Terms shall supersede any inconsistent or conflicting terms in the Binary Code License Agreement, or in any license contained within the Software.

A. Software Internal Use and Development License Grant. Subject to the terms and conditions of this Agreement and restrictions and exceptions set forth in the Software "README" file, including, but not limited to the Java Technology Restrictions of these Supplemental Terms, Sun grants you a non-exclusive, non-transferable, limited license without fees to reproduce internally and use internally the Software complete and unmodified for the purpose of designing, developing, and testing your Programs.

B. License to Distribute Software. Subject to the terms and conditions of this Agreement and restrictions and exceptions set forth in the Software README file, including, but not limited to the Java Technology Restrictions of these Supplemental Terms, Sun grants you a non-exclusive, non-transferable, limited license without fees to reproduce and distribute the Software, provided that (i) you distribute the Software complete and unmodified and only bundled as part of, and for the sole purpose of running, your Programs, (ii) the Programs add significant and primary functionality to the Software, (iii) you do not distribute additional software intended to replace any component(s) of the Software, (iv) you do not remove or alter any proprietary legends or notices contained in the Software, (v) you only distribute the Software subject to a license agreement that protects Sun's interests consistent with the terms contained in this Agreement, and (vi) you agree to defend and indemnify Sun and its licensors from and against any damages, costs, liabilities, settlement amounts and/or expenses (including attorneys' fees) incurred in connection with any claim, lawsuit or action by any third party that arises or results from the use or distribution of any and all Programs and/or Software.

C. License to Distribute Redistributables. Subject to the terms and conditions of this Agreement and restrictions and exceptions set forth in the Software README file, including but not limited to the Java Technology Restrictions of these Supplemental Terms, Sun grants you a non-exclusive, non-transferable, limited license without fees to reproduce and distribute those files specifically identified as redistributable in the Software "README" file ("Redistributables") provided that: (i) you distribute the Redistributables complete and unmodified, and only bundled as part of Programs, (ii) the Programs add significant and primary functionality to the Redistributables, (iii) you do not distribute additional software intended to supersede any component(s) of the Redistributables (unless otherwise specified in the applicable README file), (iv) you do not remove or alter any proprietary legends or notices contained in or on the Redistributables, (v) you only distribute the Redistributables pursuant to a license agreement that protects Sun's interests consistent with the terms contained in the Agreement, (vi) you agree to defend and indemnify Sun and its licensors from and against any damages, costs, liabilities, settlement amounts and/or expenses (including attorneys' fees) incurred in connection with any claim, lawsuit or action by any third party that arises or results from the use or distribution of any and all Programs and/or Software.

D. Java Technology Restrictions. You may not create, modify, or change the behavior of, or authorize your licensees to create, modify, or change the behavior of, classes, interfaces, or subpackages that are in any way identified as "java", "javax", "sun" or similar convention as specified by Sun in any naming convention designation.

E. Distribution by Publishers. This section pertains to your distribution of the Software with your printed book or magazine (as those terms are commonly used in the industry) relating to Java technology ("Publication"). Subject to and conditioned upon your compliance with the restrictions and obligations contained in the Agreement, in addition to the license granted in Paragraph 1 above, Sun hereby grants to you a non-exclusive, nontransferable limited right to reproduce complete and unmodified copies of the Software on electronic media (the "Media") for the sole purpose of inclusion and distribution with your Publication(s), subject to the following terms: (i) You may not distribute the Software on a stand-alone basis; it must be distributed with your Publication(s); (ii) You are responsible for downloading the Software from the applicable Sun web site; (iii) You must refer to the Software as JavaTM 2 Platform Standard Edition Development Kit 5.0; (iv) The Software must be reproduced in its entirety and without any modification whatsoever (including, without limitation, the

Binary Code License and Supplemental License Terms accompanying the Software and proprietary rights notices contained in the Software); (v) The Media label shall include the following information: Copyright 2004, Sun Microsystems, Inc. All rights reserved. Use is subject to license terms. Sun, Sun Microsystems, the Sun logo, Solaris, Java, the Java Coffee Cup logo, J2SE , and all trademarks and logos based on Java are trademarks or registered trademarks of Sun Microsystems, Inc. in the U.S. and other countries. This information must be placed on the Media label in such a manner as to only apply to the Sun Software; (vi) You must clearly identify the Software as Sun's product on the Media holder or Media label, and you may not state or imply that Sun is responsible for any third-party software contained on the Media; (vii) You may not include any third party software on the Media which is intended to be a replacement or substitute for the Software; (viii) You shall indemnify Sun for all damages arising from your failure to comply with the requirements of this Agreement. In addition, you shall defend, at your expense, any and all claims brought against Sun by third parties, and shall pay all damages awarded by a court of competent jurisdiction, or such settlement amount negotiated by you, arising out of or in connection with your use, reproduction or distribution of the Software and/or the Publication. Your obligation to provide indemnification under this section shall arise provided that Sun: (i) provides you prompt notice of the claim; (ii) gives you sole control of the defense and settlement of the claim; (iii) provides you, at your expense, with all available information, assistance and authority to defend; and (iv) has not compromised or settled such claim without your prior written consent; and (ix) You shall provide Sun with a written notice for each Publication; such notice shall include the following information: (1) title of Publication, (2) author(s), (3) date of Publication, and (4) ISBN or ISSN numbers. Such notice shall be sent to Sun Microsystems, Inc., 4150 Network Circle, M/S USCA12-110, Santa Clara, California 95054, U.S.A , Attention: Contracts Administration.

F. Source Code. Software may contain source code that, unless expressly licensed for other purposes, is provided solely for reference purposes pursuant to the terms of this Agreement. Source code may not be redistributed unless expressly provided for in this Agreement.

G. Third Party Code. Additional copyright notices and license terms applicable to portions of the Software are set forth in the THIRDPARTYLICENSEREADME.txt file. In addition to any terms and conditions of any third party opensource/freeware license identified in the THIRDPARTYLICENSEREADME.txt file, the disclaimer of warranty and limitation of liability provisions in paragraphs 5 and 6 of the Binary Code License Agreement shall apply to all Software in this distribution.

For inquiries please contact: Sun Microsystems, Inc., 4150 Network Circle, Santa Clara, California 95054, U.S.A.
(LFI#141623/Form ID#011801)

10.1.2 Licencia Pública de Sun (SPL) Versión 1.0

Licencia de Software Libre no compatible con GNU GPL. El entorno de desarrollo NetBeans se distribuye bajo los términos de esta licencia.

SUN PUBLIC LICENSE Version 1.0

1. Definitions.

1.0.1. "Commercial Use" means distribution or otherwise making the Covered Code available to a third party.

1.1. "Contributor" means each entity that creates or contributes to the creation of Modifications.

1.2. "Contributor Version" means the combination of the Original Code, prior Modifications used by a Contributor, and the Modifications made by that particular Contributor.

1.3. "Covered Code" means the Original Code or Modifications or the combination of the Original Code and Modifications, in each case including portions thereof and corresponding documentation released with the source code.

1.4. "Electronic Distribution Mechanism" means a mechanism generally accepted in the software development community for the electronic transfer of data.

1.5. "Executable" means Covered Code in any form other than Source Code.

1.6. "Initial Developer" means the individual or entity identified as the Initial Developer in the Source Code notice required by Exhibit A.

1.7. "Larger Work" means a work which combines Covered Code or portions thereof with code not governed by the terms of this License.

1.8. "License" means this document.

1.8.1. "Licensable" means having the right to grant, to the maximum extent possible, whether at the time of the initial grant or subsequently acquired, any and all of the rights conveyed herein.

1.9. "Modifications" means any addition to or deletion from the substance or structure of either the Original Code or any previous Modifications. When Covered Code is released as a series of files, a Modification is:

A. Any addition to or deletion from the contents of a file containing Original Code or previous Modifications.

B. Any new file that contains any part of the Original Code or previous Modifications.

1.10. "Original Code" means Source Code of computer software code which is described in the Source Code notice required by Exhibit A as Original Code, and which, at the time of its release under this License is not already Covered Code governed by this License.

1.10.1. "Patent Claims" means any patent claim(s), now owned or hereafter acquired, including without limitation, method, process, and apparatus claims, in any patent Licensable by grantor.

1.11. "Source Code" means the preferred form of the Covered Code for making modifications to it, including all modules it contains, plus any associated documentation, interface definition files, scripts used to control compilation and installation of an Executable, or source code differential comparisons against either the Original Code or another well known, available Covered Code of the Contributor's choice. The Source Code can be in a compressed or archival form, provided the appropriate decompression or de-archiving software is widely available for no charge.

1.12. "You" (or "Your") means an individual or a legal entity exercising rights under, and complying with all of the terms of, this License or a future version of this License issued under Section 6.1. For legal entities, "You" includes any entity which controls, is controlled by, or is under common control with You. For purposes of this definition, "control" means (a) the power, direct or indirect, to cause the direction or management of such entity, whether by contract or otherwise, or (b) ownership of more than fifty percent (50%) of the outstanding shares or beneficial ownership of such entity.

2. Source Code License.

2.1 The Initial Developer Grant.

The Initial Developer hereby grants You a world-wide, royalty-free, non-exclusive license, subject to third party intellectual property claims:

(a) under intellectual property rights (other than patent or trademark) Licensable by Initial Developer to use, reproduce, modify, display, perform, sublicense and distribute the Original Code (or portions thereof) with or without Modifications, and/or as part of a Larger Work; and

(b) under Patent Claims infringed by the making, using or selling of Original Code, to make, have made, use, practice, sell, and offer for sale, and/or otherwise dispose of the Original Code (or portions thereof).

(c) the licenses granted in this Section 2.1(a) and (b) are effective on the date Initial Developer first distributes Original Code under the terms of this License.

(d) Notwithstanding Section 2.1(b) above, no patent license is granted: 1) for code that You delete from the Original Code; 2) separate from the Original Code; or 3) for infringements caused by:

i) the modification of the Original Code or ii) the combination of the Original Code with other software or devices.

2.2. Contributor Grant.

Subject to third party intellectual property claims, each Contributor hereby grants You a world-wide, royalty-free, non-exclusive license

(a) under intellectual property rights (other than patent or trademark) Licensable by Contributor, to use, reproduce, modify, display, perform, sublicense and distribute the Modifications created by such Contributor (or portions thereof) either on an unmodified basis, with other Modifications, as Covered Code and/or as part of a Larger Work; and

(b) under Patent Claims infringed by the making, using, or selling of Modifications made by that Contributor either alone and/or in combination with its Contributor Version (or portions of such combination), to make, use, sell, offer for sale, have made, and/or otherwise dispose of: 1) Modifications made by that Contributor (or portions thereof); and 2) the combination of Modifications made by that Contributor with its Contributor Version (or portions of such combination).

(c) the licenses granted in Sections 2.2(a) and 2.2(b) are effective on the date Contributor first makes Commercial Use of the Covered Code.

(d) notwithstanding Section 2.2(b) above, no patent license is granted: 1) for any code that Contributor has deleted from the Contributor Version; 2) separate from the Contributor Version; 3) for infringements caused by: i) third party modifications of Contributor Version or ii) the combination of Modifications made by that Contributor with other software (except as part of the Contributor Version) or other devices; or 4) under Patent Claims infringed by Covered Code in the absence of Modifications made by that Contributor.

3. Distribution Obligations.

3.1. Application of License.

The Modifications which You create or to which You contribute are governed by the terms of this License, including without limitation Section 2.2. The Source Code version of Covered Code may be distributed only under the terms of this License or a future version of this License released under Section 6.1, and You must include a copy of this License with every copy of the Source Code You distribute. You may not offer or impose any terms on any Source Code version that alters or restricts the applicable version of this License or the recipients' rights hereunder. However, You may include an additional document offering the additional rights described in Section 3.5.

3.2. Availability of Source Code.

Any Modification which You create or to which You contribute must be made available in Source Code form under the terms of this License either on the same media as an Executable version or via an accepted Electronic Distribution Mechanism to anyone to whom you made an Executable version available; and if made available via Electronic Distribution Mechanism, must remain available for at least twelve (12) months after the date it initially became available, or at least six (6) months after a subsequent version of that particular Modification has been made available to such recipients. You are responsible for ensuring that the Source Code version remains available even if the Electronic Distribution Mechanism is maintained by a third party.

3.3. Description of Modifications.

You must cause all Covered Code to which You contribute to contain a file documenting the changes You made to create that Covered Code and the date of any change. You must include a prominent statement that the Modification is derived, directly or indirectly, from Original Code provided by the Initial Developer and including the name of the Initial Developer in (a) the Source Code, and (b) in any notice in an Executable version or related documentation in which You describe the origin or ownership of the Covered Code.

3.4. Intellectual Property Matters.

(a) Third Party Claims.

If Contributor has knowledge that a license under a third party's intellectual property rights is required to exercise the rights granted by such Contributor under Sections 2.1 or 2.2, Contributor must include a text file with the Source Code distribution titled "LEGAL" which describes the claim and the party making the claim in sufficient detail that a recipient will know whom to contact. If Contributor obtains such knowledge after the Modification is made available as described in Section 3.2, Contributor shall promptly modify the LEGAL file in all copies Contributor makes available thereafter and shall take other steps (such as notifying appropriate mailing lists or newsgroups) reasonably calculated to inform those who received the Covered Code that new knowledge has been obtained.

(b) Contributor APIs.

If Contributor's Modifications include an application programming interface ("API") and Contributor has knowledge of patent licenses which are reasonably necessary to implement that API, Contributor must also include this information in the LEGAL file.

(c) Representations.

Contributor represents that, except as disclosed pursuant to Section 3.4(a) above, Contributor believes that Contributor's Modifications are Contributor's original creation(s) and/or Contributor has sufficient rights to grant the rights conveyed by this License.

3.5. Required Notices.

You must duplicate the notice in Exhibit A in each file of the Source Code. If it is not possible to put such notice in a particular Source Code file due to its structure, then You must include such notice in a location (such as a relevant directory) where a user would be likely to look for such a notice. If You created one or more Modification(s) You may add your name as a Contributor to the notice described in Exhibit A. You must also duplicate this License in any documentation for the Source Code where You describe recipients' rights or ownership rights relating to Covered Code. You may choose to offer, and to charge a fee for, warranty, support, indemnity or liability obligations to one or more recipients of Covered Code. However, You may do so only on Your own behalf, and not on behalf of the Initial Developer or any Contributor. You must make it absolutely clear than any such warranty, support, indemnity or liability obligation is offered by You alone, and You hereby agree to indemnify the Initial Developer and every Contributor for any liability incurred by the Initial Developer or such Contributor as a result of warranty, support, indemnity or liability terms You offer.

3.6. Distribution of Executable Versions.

You may distribute Covered Code in Executable form only if the requirements of Section 3.1-3.5 have been met for that Covered Code, and if You include a notice stating that the Source Code version of the Covered Code is available under the terms of this License, including a description of how and where You have fulfilled the obligations of Section 3.2. The notice must be conspicuously included in any notice in an Executable version, related documentation or collateral in which You describe recipients' rights relating to the Covered Code. You may distribute the Executable version of Covered Code or ownership rights under a license of Your choice, which may contain terms different from this License, provided that You are in compliance with the terms of this License and that the license for the Executable version does not attempt to limit or alter the recipient's rights in the Source Code version from the rights set forth in this License. If You distribute the Executable version under a different license You must make it absolutely clear that any terms which differ from this License are offered by You alone, not by the Initial Developer or any Contributor. You hereby agree to indemnify the Initial Developer and every Contributor for any liability incurred by the Initial Developer or such Contributor as a result of any such terms You offer.

3.7. Larger Works.

You may create a Larger Work by combining Covered Code with other code not governed by the terms of this License and distribute he Larger Work as a single product. In such a case, You must make sure the requirements of this License are fulfilled for the Covered Code.

4. Inability to Comply Due to Statute or Regulation.

If it is impossible for You to comply with any of the terms of this License with respect to some or all of the

Covered Code due to statute, judicial order, or regulation then You must: (a) comply with the terms of this License to the maximum extent possible; and (b) describe the limitations and the code they affect. Such description must be included in the LEGAL file described in Section 3.4 and must be included with all distributions of the Source Code. Except to the extent prohibited by statute or regulation, such description must be sufficiently detailed for a recipient of ordinary skill to be able to understand it.

5. Application of this License.

This License applies to code to which the Initial Developer has attached the notice in Exhibit A and to related Covered Code.

6. Versions of the License.

6.1. New Versions.

Sun Microsystems, Inc. ("Sun") may publish revised and/or new versions of the License from time to time. Each version will be given distinguishing version number.

6.2. Effect of New Versions.

Once Covered Code has been published under a particular version of the License, You may always continue to use it under the terms of that version. You may also choose to use such Covered Code under the terms of any subsequent version of the License published by Sun. No one other than Sun has the right to modify the terms applicable to Covered Code created under this License.

6.3. Derivative Works.

If You create or use a modified version of this License (which you may only do in order to apply it to code which is not already Covered Code governed by this License), You must: (a) rename Your license so that the phrases "Sun," "Sun Public License," or "SPL" or any confusingly similar phrase do not appear in your license (except to note that your license differs from this License) and (b) otherwise make it clear that Your version of the license contains terms which differ from the Sun Public License. (Filling in the name of the Initial Developer, Original Code or Contributor in the notice described in Exhibit A shall not of themselves be deemed to be modifications of this License.)

7. DISCLAIMER OF WARRANTY.

COVERED CODE IS PROVIDED UNDER THIS LICENSE ON AN "AS IS'' BASIS, WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, WITHOUT LIMITATION, WARRANTIES THAT THE COVERED CODE IS FREE OF DEFECTS, MERCHANTABLE, FIT FOR A PARTICULAR PURPOSE OR NON-INFRINGING.

THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE COVERED CODE IS WITH YOU. SHOULD ANY COVERED CODE PROVE DEFECTIVE IN ANY RESPECT, YOU (NOT THE INITIAL DEVELOPER OR ANY OTHER CONTRIBUTOR) ASSUME THE COST OF ANY NECESSARY SERVICING, REPAIR OR CORRECTION. THIS DISCLAIMER OF WARRANTY CONSTITUTES AN ESSENTIAL PART OF THIS LICENSE. NO USE OF ANY COVERED CODE IS AUTHORIZED HEREUNDER EXCEPT UNDER THIS DISCLAIMER.

8. TERMINATION.

8.1. This License and the rights granted hereunder will terminate automatically if You fail to comply with terms herein and fail to cure such breach within 30 days of becoming aware of the breach. All sublicenses to the Covered Code which are properly granted shall survive any termination of this License. Provisions which, by their

nature, must remain in effect beyond the termination of this License shall survive.

8.2. If You initiate litigation by asserting a patent infringement claim (excluding declaratory judgment actions) against Initial Developer or a Contributor (the Initial Developer or Contributor against whom You file such action is referred to as "Participant") alleging that:

(a) such Participant's Contributor Version directly or indirectly infringes any patent, then any and all rights granted by such Participant to You under Sections 2.1 and/or 2.2 of this License shall, upon 60 days notice from Participant terminate prospectively, unless if within 60 days after receipt of notice You either: (i) agree in writing to pay Participant a mutually agreeable reasonable royalty for Your past and future use of Modifications made by such Participant, or (ii) withdraw Your litigation claim with respect to the Contributor Version against such Participant. If within 60 days of notice, a reasonable royalty and payment arrangement are not mutually agreed upon in writing by the parties or the litigation claim is not withdrawn, the rights granted by Participant to You under Sections 2.1 and/or 2.2 automatically terminate at the expiration of the 60 day notice period specified above.

(b) any software, hardware, or device, other than such Participant's Contributor Version, directly or indirectly infringes any patent, then any rights granted to You by such Participant under Sections 2.1(b) and 2.2(b) are revoked effective as of the date You first made, used, sold, distributed, or had made, Modifications made by that Participant.

8.3. If You assert a patent infringement claim against Participant alleging that such Participant's Contributor Version directly or indirectly infringes any patent where such claim is resolved (such as by license or settlement) prior to the initiation of patent infringement litigation, then the reasonable value of the licenses granted by such Participant under Sections 2.1 or 2.2 shall be taken into account in determining the amount or value of any payment or license.

8.4. In the event of termination under Sections 8.1 or 8.2 above, all end user license agreements (excluding distributors and resellers) which have been validly granted by You or any distributor hereunder prior to termination shall survive termination.

9. LIMITATION OF LIABILITY.

UNDER NO CIRCUMSTANCES AND UNDER NO LEGAL THEORY, WHETHER TORT (INCLUDING NEGLIGENCE), CONTRACT, OR OTHERWISE, SHALL YOU, THE INITIAL DEVELOPER, ANY OTHER CONTRIBUTOR, OR ANY DISTRIBUTOR OF COVERED CODE, OR ANY SUPPLIER OF ANY OF SUCH PARTIES, BE LIABLE TO ANY PERSON FOR ANY INDIRECT, SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES OF ANY CHARACTER INCLUDING, WITHOUT LIMITATION, DAMAGES FOR LOSS OF GOODWILL, WORK STOPPAGE, COMPUTER FAILURE OR MALFUNCTION, OR ANY AND ALL OTHER COMMERCIAL DAMAGES OR LOSSES, EVEN IF SUCH PARTY SHALL HAVE BEEN INFORMED OF THE POSSIBILITY OF SUCH DAMAGES. THIS LIMITATION OF LIABILITY SHALL NOT APPLY TO LIABILITY FOR DEATH OR PERSONAL INJURY RESULTING FROM SUCH PARTY'S NEGLIGENCE TO THE EXTENT APPLICABLE LAW PROHIBITS SUCH LIMITATION. SOME JURISDICTIONS DO NOT ALLOW THE EXCLUSION OR LIMITATION OF INCIDENTAL OR CONSEQUENTIAL DAMAGES, SO THIS EXCLUSION AND LIMITATION MAY NOT APPLY TO YOU.

10. U.S. GOVERNMENT END USERS.

The Covered Code is a "commercial item," as that term is defined in 48 C.F.R. 2.101 (Oct. 1995), consisting of "commercial computer software" and "commercial computer software documentation," as such terms are used in 48 C.F.R. 12.212 (Sept. 1995). Consistent with 48 C.F.R. 12.212 and 48 C.F.R. 227.7202-1 through 227.7202-4 (June

1995), all U.S. Government End Users acquire Covered Code with only those rights set forth herein.

11. MISCELLANEOUS.

This License represents the complete agreement concerning subject matter hereof. If any provision of this License is held to be unenforceable, such provision shall be reformed only to the extent necessary to make it enforceable. This License shall be governed by California law provisions (except to the extent applicable law, if any, provides otherwise), excluding its conflict-of-law provisions. With respect to disputes in which at least one party is a citizen of, or an entity chartered or registered to do business in the United States of America, any litigation relating to this License shall be subject to the jurisdiction of the Federal Courts of the Northern District of California, with venue lying in Santa Clara County, California, with the losing party responsible for costs, including without limitation, court costs and reasonable attorneys' fees and expenses. The application of the United Nations Convention on Contracts for the International Sale of Goods is expressly excluded. Any law or regulation which provides that the language of a contract shall be construed against the drafter shall not apply to this License.

12. RESPONSIBILITY FOR CLAIMS.

As between Initial Developer and the Contributors, each party is responsible for claims and damages arising, directly or indirectly, out of its utilization of rights under this License and You agree to work with Initial Developer and Contributors to distribute such responsibility on an equitable basis. Nothing herein is intended or shall be deemed to constitute any admission of liability.

13. MULTIPLE-LICENSED CODE.

Initial Developer may designate portions of the Covered Code as "Multiple-Licensed". "Multiple-Licensed" means that the Initial Developer permits you to utilize portions of the Covered Code under Your choice of the alternative licenses, if any, specified by the Initial Developer in the file described in Exhibit A.

Exhibit A -Sun Public License Notice.

The contents of this file are subject to the Sun Public License Version 1.0 (the "License"); you may not use this file except in compliance with the License. A copy of the License is available at http://www.sun.com/

The Original Code is _________________. The Initial Developer of the Original Code is ___________. Portions created by ______ are Copyright (C) _________. All Rights Reserved.

Contributor(s): ______________________________________.

Alternatively, the contents of this file may be used under the terms of the _____ license (the "[___] License"), in which case the provisions of [______] License are applicable instead of those above.

If you wish to allow use of your version of this file only under the terms of the [____] License and not to allow others to use your version of this file under the SPL, indicate your decision by deleting the provisions above and replace them with the notice and other provisions required by the [___] License. If you do not delete the provisions above, a recipient may use your version of this file under either the SPL or the [___] License."

[NOTE: The text of this Exhibit A may differ slightly from the text of the notices in the Source Code files of

10.1.3 Licencia de Apache (ASL) Versión 2.0

Licencia de Software Libre. En la lista de licencias de Software Libre que ofrece la Fundación de Software Libre, está clasificada como incompatible con GNU GPL, sin embargo, la Fundación de Software Apache defiende su compatibilidad con GNU GPL. Actualmente parece que se sigue intentando determinar si realmente son compatibles, o no, las licencias.

Utilizamos el contendor de Servlets y JSP Tomcat, el framework de Struts y las librerías de etiquetas Standard TagLibs bajo esta licencia.

"Legal Entity" shall mean the union of the acting entity and all other entities that control, are controlled by, or are under common control with that entity. For the purposes of this definition, "control" means (i) the power, direct or indirect, to cause the direction or management of such entity, whether by contract or otherwise, or (ii) ownership of fifty percent (50%) or more of the outstanding shares, or (iii) beneficial ownership of such entity.

"You" (or "Your") shall mean an individual or Legal Entity exercising permissions granted by this License.

"Source" form shall mean the preferred form for making modifications, including but not limited to software source code, documentation source, and configuration files.

"Object" form shall mean any form resulting from mechanical transformation or translation of a Source form, including but not limited to compiled object code, generated documentation, and conversions to other media types.

"Work" shall mean the work of authorship, whether in Source or Object form, made available under the License, as indicated by a copyright notice that is included in or attached to the work (an example is provided in the Appendix below).

"Derivative Works" shall mean any work, whether in Source or Object form, that is based on (or derived from) the Work and for which the editorial revisions, annotations, elaborations, or other modifications represent, as a whole, an original work of authorship. For the purposes of this License, Derivative Works shall not include works that remain separable from, or merely link (or bind by name) to the interfaces of, the Work and Derivative Works thereof.

where such license applies only to those patent claims licensable by such Contributor that are necessarily infringed by their Contribution(s) alone or by combination of their Contribution(s) with the Work to which such Contribution(s) was submitted. If You institute patent litigation against any entity (including a cross-claim or counterclaim in a lawsuit) alleging that the Work or a Contribution incorporated within the Work constitutes direct or contributory patent infringement, then any patent licenses granted to You under this License for that Work shall terminate as of the date such litigation is filed.

4. Redistribution. You may reproduce and distribute copies of the Work or Derivative Works thereof in any medium, with or without modifications, and in Source or Object form, provided that You meet the following conditions:

(a) You must give any other recipients of the Work or Derivative Works a copy of this License; and

(b) You must cause any modified files to carry prominent notices stating that You changed the files; and

(c) You must retain, in the Source form of any Derivative Works that You distribute, all copyright, patent, trademark, and attribution notices from the Source form of the Work, excluding those notices that do not pertain to any part of the Derivative Works; and

(d) If the Work includes a "NOTICE" text file as part of its distribution, then any Derivative Works that You distribute must include a readable copy of the attribution notices contained within such NOTICE file, excluding those notices that do not pertain to any part of the Derivative Works, in at least one of the following places: within a NOTICE text file distributed as part of the Derivative Works; within the Source form or documentation, if provided

10.1.4 Licencia de Software de Kannel Versión 1.0

Esta licencia de Software Libre es totalmente compatible con GNU GPL. Utilizamos el servidor de SMS Kannel bajo esta licencia.

10.1.5 Licencia Pública General de GNU (GPL) Versión 2

Licencia de Software Libre bajo la que se distribuyen la mayor parte de herramienta de la Fundación de Software Libre. Otras herramientas no pertenecientes a esta fundación también distribuyen su software bajo esta licencia. Utilizamos la base de datos MySQL, el driver MySQL Connector/J y el entorno Linux bajo Windows Cygwin, bajo los términos de esta licencia.

GNU General Public License
****

Version 2, June 1991

Preamble
========

The licenses for most software are designed to take away your freedom to share and change it. By contrast, the GNU General Public License is intended to guarantee your freedom to share and change free software--to make sure the software is free for all its users. This General Public License applies to most of the Free Software Foundation's software and to any other program whose authors commit to using it. (Some other Free Software Foundation software is covered by the GNU Library General Public License instead.) You can apply it to your programs, too.

When we speak of free software, we are referring to freedom, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for this service if you wish), that you receive source code or can get it if you want it, that you can change the software or use pieces of it in new free programs; and that you know you can do these things.

To protect your rights, we need to make restrictions that forbid anyone to deny you these rights or to ask you to surrender the rights. These restrictions translate to certain responsibilities for you if you distribute copies of the software, or if you modify it.

For example, if you distribute copies of such a program, whether gratis or for a fee, you must give the recipients all the rights that you have. You must make sure that they, too, receive or can get the source code. And you must show them these terms so they know their rights.

We protect your rights with two steps: (1) copyright the software, and (2) offer you this license which gives you legal permission to copy, distribute and/or modify the software.

Also, for each author's protection and ours, we want to make certain that everyone understands that there is no warranty for this free software. If the software is modified by someone else and passed on, we want its recipients to know that what they have is not the original, so that any problems introduced by others will not reflect on the original authors' reputations.

Finally, any free program is threatened constantly by software patents. We wish to avoid the danger that redistributors of a free program will individually obtain patent licenses, in effect making the program proprietary. To prevent this, we have made it clear that any patent must be licensed for everyone's free use or not licensed at all.

The precise terms and conditions for copying, distribution and modification follow.

GNU GENERAL PUBLIC LICENSE

TERMS AND CONDITIONS FOR COPYING, DISTRIBUTION AND MODIFICATION

0. This License applies to any program or other work which contains a notice placed by the copyright holder saying it may be distributed under the terms of this General Public License. The "Program", below, refers to any

such program or work, and a "work based on the Program" means either the Program or any derivative work under copyright law: that is to say, a work containing the Program or a portion of it, either verbatim or with modifications and/or translated into another language. (Hereinafter, translation is included without limitation in the term "modification".) Each licensee is addressed as "you".

Activities other than copying, distribution and modification are not covered by this License; they are outside its scope. The act of running the Program is not restricted, and the output from the Program is covered only if its contents constitute a work based on the Program (independent of having been made by running the Program). Whether that is true depends on what the Program does.

1. You may copy and distribute verbatim copies of the Program's source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice and disclaimer of warranty; keep intact all the notices that refer to this License and to the absence of any warranty; and give any other recipients of the Program a copy of this License along with the Program.

You may charge a fee for the physical act of transferring a copy, and you may at your option offer warranty protection in exchange for a fee.

2. You may modify your copy or copies of the Program or any portion of it, thus forming a work based on the Program, and copy and distribute such modifications or work under the terms of Section 1 above, provided that you also meet all of these conditions:

a. You must cause the modified files to carry prominent notices stating that you changed the files and the date of any change.

b. You must cause any work that you distribute or publish, that in whole or in part contains or is derived from the Program or any part thereof, to be licensed as a whole at no charge to all third parties under the terms of this License.

c. If the modified program normally reads commands interactively when run, you must cause it, when started running for such interactive use in the most ordinary way, to print or display an announcement including an appropriate copyright notice and a notice that there is no warranty (or else, saying that you provide a warranty) and that users may redistribute the program under these conditions, and telling the user how to view a copy of this License. (Exception: if the Program itself is interactive but does not normally print such an announcement, your work based on the Program is not required to print an announcement.)

These requirements apply to the modified work as a whole. If identifiable sections of that work are not derived from the Program, and can be reasonably considered independent and separate works in themselves, then this License, and its terms, do not apply to those sections when you distribute them as separate works. But when you distribute the same sections as part of a whole which is a work based on the Program, the distribution of the whole must be on the terms of this License, whose permissions for other licensees extend to the entire whole, and thus to each and every part regardless of who wrote it.

Thus, it is not the intent of this section to claim rights or contest your rights to work written entirely by you; rather, the intent is to exercise the right to control the distribution of derivative or collective works based on the Program.

In addition, mere aggregation of another work not based on the Program with the Program (or with a work based on the Program) on a volume of a storage or distribution medium does not bring the other work under the scope of this License.

3. You may copy and distribute the Program (or a work based on it, under Section 2) in object code or executable form under the terms of Sections 1 and 2 above provided that you also do one of the following:

a. Accompany it with the complete corresponding machine-readable source code, which must be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,

b. Accompany it with a written offer, valid for at least three years, to give any third-party, for a charge no more than your cost of physically performing source distribution, a complete machine-readable copy of the corresponding source code, to be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,

c. Accompany it with the information you received as to the offer to distribute corresponding source code. (This alternative is allowed only for noncommercial distribution and only if you received the program in object code or executable form with such an offer, in accord with Subsection b above.)

The source code for a work means the preferred form of the work for making modifications to it. For an executable work, complete source code means all the source code for all modules it contains, plus any associated interface definition files, plus the scripts used to control compilation and installation of the executable. However, as a special exception, the source code distributed need not include anything that is normally distributed (in either source or binary form) with the major components (compiler, kernel, and so on) of the operating system on which the executable runs, unless that component itself accompanies the executable.

If distribution of executable or object code is made by offering access to copy from a designated place, then offering equivalent access to copy the source code from the same place counts as distribution of the source code, even though third parties are not compelled to copy the source along with the object code.

4. You may not copy, modify, sublicense, or distribute the Program except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense or distribute the Program is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

5. You are not required to accept this License, since you have not signed it. However, nothing else grants you permission to modify or distribute the Program or its derivative works. These actions are prohibited by law if you do not accept this License. Therefore, by modifying or distributing the Program (or any work based on the Program), you indicate your acceptance of this License to do so, and all its terms and conditions for copying, distributing or modifying the Program or works based on it.

6. Each time you redistribute the Program (or any work based on the Program), the recipient automatically receives a license from the original licensor to copy, distribute or

modify the Program subject to these terms and conditions. You may not impose any further restrictions on the recipients' exercise of the rights granted herein. You are not responsible for enforcing compliance by third parties to this License.

7. If, as a consequence of a court judgment or allegation of patent infringement or for any other reason (not limited to patent issues), conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot distribute so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not distribute the Program at all. For example, if a patent license would not permit royalty-free redistribution of the Program by all those who receive copies directly or indirectly through you, then the only way you could satisfy both it and this License would be to refrain entirely from distribution of the Program.

If any portion of this section is held invalid or unenforceable under any particular circumstance, the balance of the section is intended to apply and the section as a whole is intended to apply in other circumstances.

It is not the purpose of this section to induce you to infringe any patents or other property right claims or to contest validity of any such claims; this section has the sole purpose of protecting the integrity of the free software distribution system, which is implemented by public license practices. Many people have made generous contributions to the wide range of software distributed through that system in reliance on consistent application of that system; it is up to the author/donor to decide if he or she is willing to distribute software through any other system and a licensee cannot impose that choice.

This section is intended to make thoroughly clear what is believed to be a consequence of the rest of this License.

8. If the distribution and/or use of the Program is restricted in certain countries either by patents or by copyrighted interfaces, the original copyright holder who places the Program under this License may add an explicit geographical distribution limitation excluding those countries, so that distribution is permitted only in or among countries not thus excluded. In such case, this License incorporates the limitation as if written in the body of this License.

9. The Free Software Foundation may publish revised and/or new versions of the General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

Each version is given a distinguishing version number. If the Program specifies a version number of this License which applies to it and "any later version", you have the option of following the terms and conditions either of that version or of any later version published by the Free Software Foundation. If the Program does not specify a version number of this License, you may choose any version ever published by the Free Software Foundation.

10. If you wish to incorporate parts of the Program into other free programs whose distribution conditions are different, write to the author to ask for permission. For software which is copyrighted by the Free Software Foundation, write to the Free Software Foundation; we sometimes make exceptions for this. Our decision will be guided by the two goals of preserving the free status of

10.2 APACHE SOFTWARE FOUNDATION

Apache, a través del proyecto Jakarta, ofrece una serie de soluciones Java de código abierto bajo una licencia de software abierto.

Entre los subproyectos de Jakarta, relacionados con J2EE destacamos:

Jakarta TagLibs:

Conjunto de etiquetas personalizadas JSP y herramientas para la creación de etiquetas personalizadas. Entre estas etiquetas incluye la implementación de referencia de la librería de etiquetas estándar de Java, JSTL (The Standard Tag Library), estandarizadas por la Comunidad Java, JSR-52.

Struts:
Framework de desarrollo MVC. Se inició como un subproyecto de Jakarta, pero debido a la importación que ha adquirido este proyecto, ya se ha establecido como un proyecto independiente del grupo Apache.

Tomcat:
Implementación de referencia de contenedor de Servlet y JSP. Al igual que el proyecto anterior, se ha independizado del proyecto Jakarta.

10.3 VERSIONES

10.3.1 NetBeans 4.0.

NetBeans es un entorno de programado en Java. Existen versiones compatibles con Windows, Linux, Solaris, Java Desktop, MAC, Open VMS y otros sistemas operativos. Las versiones más recientes son:

- NetBeans IDE 3.6, de Marzo de 2004.
- NetBeans IDE 4.0, de Diciembre de 2004.
- NetBeans IDE 4.1, de Mayo de 2005. Esta es la última versión estable actualmente aunque ya existe en versión beta NetBeans IDE 5.0.

Los requisitos de Hardware son mínimos. Estas versiones pueden trabajar con un Pentium III (o equivalente) de 500MHz, memoria de 318MB y disco libre disponible de 125MB. Como requisito software coinciden en el uso de J2SE desde 1.4.2, preferible JDK 5.0. Incorporan una versión de Tomcat 5.0.x (5.0.19 en 3.6 y 5.0.28 en 4.0).

Son compatibles entre otras tecnologías con Servlet 2.3 y 2.4, JSP 1.2 y 2.0, J2EE 1.3 y 1.4, y multitud de bases de datos (por ejemplo MySQL 4.x).

La versión 4.0 nos ofrece más beneficios y facilidad de uso, entre ellas mayor compatibilidad con J2ME aunque en principio no utilizará en nuestra aplicación. Aunque el desarrollo de nuestra aplicación se comenzó con la versión 3.6, pronto se migró a la versión 4.0.

La versión 4.1 es mucho más completa respecto a J2EE que las versiones anteriores, sin embargo, debido a lo avanzado del proyecto y a que nuestra aplicación se centraba en el uso de Servlets y JSP, y no de un servidor J2EE completo, se decidió mantener el proyecto en la versión 4.0.

10.3.2 Tomcat 5.0.28.

Tomcat es el contenedor de Servlets y JSP más utilizado. Además de ser gratuito y permitido su uso comercial, es la implementación de referencia de las especificaciones de Servlets y JSP. Además de poder utilizarse independientemente, muchas de las herramientas de desarrollo lo utilizan embebido e integrado con sus propias herramientas. Este último, es el caso del NetBeans.

Se trabaja con la versión Tomcat 5.0.28 que es la que trae por defecto NetBeans 4.0. Aunque se puede configurar otro contenedor se consideró éste una buena opción.

Las últimas versiones de Servlets son la 2.4 y la 2.3 aunque parece que en poco saldrá la 2.5. Las versiones más recientes de la especificación de JSP son las 1.2 y 2.0, aunque igualmente está en desarrollo la versión 2.1.

Servlets 2.3 y JSP 1.2, versiones estables desde Septiembre de 2001. Incluidas en las especificaciones de J2EE 1.3 e implementadas por Tomcat 4.x. Necesita un J2SE 1.3 o superior.
Servlets 2.4 y JSP 2.0, versiones estables desde Noviembre de 2003. Incluidas en las especificación de J2EE 1.4 e implementadas por Tomcat 5.x. Necesita J2SE 1.4 o superior.

Las versiones de Tomcat 5.x permitien trabajar con versiones anteriores de la especificación. En NetBeans configuramos un entorno de desarrollo de Servlets 2.4 y JSP 2.0, y utilizamos el contenedor Tomcat 5.0.18.

10.3.3 J2SE 5.0.

J2SE nos proporciona un entorno completo para el desarrollo de aplicaciones en Java. Aporta operaciones las bases de seguridad, conectividad a bases de datos y otras muchas características necesarias para nuestras aplicaciones.

J2SE Run Environment (JRE) proporciona librerías, máquina virtual java y otras herramientas necesarias para ejecutar aplicaciones desarrolladas en Java. J2SE Development Kit (JDK) incluye el JRE además de herramientas de desarrollo como compilador. Este último es el que utilizamos. Existen varias versiones:

- J2SE 1.4, primera versión estable en Mayo de 2002 y con posteriores actualizaciones 1.4.1 en Septiembre de 2002 y 1.4.2 en Junio de 2003.
- J2SE 1.5 (esta versión cambió el nombre a J2SE 5.0), primera versión estable en Septiembre 2004.

El desarrollo de la aplicación se inició utilizando versiones de J2SE 1.4 sin embargo en cuanto aparece la versión 5.0 migramos a ella. Puesto que J2SE es compatible hacia atrás, nuestra aplicación no se vió afectada negativamente y la nueva versión nos ofreció, entre otros beneficios, la conversión de tipos.

NetBeans utiliza J2SE no sólo para ejecutarse sino también para compilar las aplicaciones que desarrollemos por lo que el JRE no sería suficiente.

Además J2SE nos proporciona los paquetes básicos para serialización, uso de JavaBeans, seguridad, y jaxp (parser implementado en Java para trabajar con documentos XML). También está integrado con tecnologías como JNDI, JDBC, RMI e IDL, algunas de las cuales utilizamos.

Para nuestra aplicación se utiliza como contenedor de Servlets y JSP, Tomcat, integrado en NetBeans. Sin embargo, si hubiésemos utilizado alguna de sus versiones en stand-alone, también hubiese sido necesario el J2SE completo, ya que es éste el que nos proporciona algunas librerías necesarías para que nuestra aplicación funcionase, si no lo utilizáramos, tendríamos que proporcionarle cada una de las librerías necesarias de forma independiente.
J2SE 5.0 incluye JAXP 1.3, JDBC 3.0 e implementaciones de JNDI.

10.3.4 Struts 1.2.8.

Para facilitar el desarrollo de la aplicación utilizamos el Framework de Struts. Este framework implementará la capa controlador y nos ofrece librerías de etiquetas que nos ayudarán en la vista. Struts es un framework maduro ya que su primera versión apareció en Junio de 2001. Las posibles versiones a utilizar:

- Struts v.1.0 de Junio 2001 y diversas mejoras con v.1.0.1 en Enero de 2002, y v.1.0.2 en Febrero de 2002.
- Struts v.1.1 en Junio de 2003.
- Struts v.1.2.1 (versión beta) en Julio de 2004 y posteriores versiones v.1.2.2 en Agosto de 2004, v.1.2.2 en Septiembre 2004, v.1.2.7 en Mayo de 2005. Actualmente la última versión disponible es la v.1.2.8 desde Noviembre de 2005.

10.3.5 JSTL 1.1.

Si se trabajara con JSP 1.2, sería conveniente usar una de las versiones de JSTL 1.0.x. Si se trabaja con JSP 2.0 necesitamos una de las versiones JSTL 1.1.x.

- JSTL v.1.0, Junio de 2002.
- JSTL v.1.0.1 de Julio 2002, v.1.0.2 de Octubre de 2002, v1.0.3 de Febrero de 2003, v.1.0.4 de Septiembre 2003, v.1.0.5 de Enero de 2004 y v.1.0.6 de Julio de 2004.
- JSTL v.1.1.0 de Enero de 2004, v.1.1.1 de Julio de 2004 y v.1.1.2 de Octubre de 2004.

10.3.6 MySQL Database Server 4.1.9.

Se considera una de las base de datos más rápidas. Inicialmente, MySQL carecía de elementos considerados esenciales en las bases de datos relacionales, tales como integridad referencial y transacciones. Pero poco a poco los elementos faltantes en MySQL están siendo incorporados tanto por desarrollos internos, como por desarrolladores de software libre, por lo que existen multitud versiones. Entre las características disponibles en las últimas versiones se puede destacar la disponibilidad en gran cantidad de plataformas y sistemas, diferentes opciones de almacenamiento según si se desea velocidad en las operaciones o el mayor número de operaciones disponibles, transacciones y claves foráneas, conectividad segura, búsqueda e indexación de campos de texto, etc.

Esta base de datos ya ofrece versiones para multitud de plataformas: Linux, Windows, Mac, Solaris, FreeBSD, Open BSD, sistemas operativos de Novell, de HP, de IBM y muchas más. La primera versión para Windows apareció en Enero de 2000 con la 3.22.30. Desde entonces se han desarrollado multitud de versiones y actualizaciones de versiones existentes.

La versión 3.23.x comenzó a desarrollarse en 2001. Inicialmente se eligió una de estas versiones, la MySQL Database Server 3.23.52, desarrollada en Agosto 2002, durante este año y el siguiente se desarrollaron más versiones de 3.23.52.

En el 2001 ya comenzó a desarrollarse versiones de MySQL 4.0.x, hasta la última versión actualmente MySQL Database Server 4.0.26 de Septiembre de 2005.

Debido a agentes exteriores al desarrollo de la aplicación, fue necesario reinstalar la base de datos ya en proceso de desarrollo de la aplicación. Se decidió entonces actualizar la base de datos, y para ello se eligió una nueva versión, MySQL Database Server 4.1.9, desarrollada en Enero de 2005. La primera versión de 4.1.x comenzó a desarrollarse en Diciembre de 2003.

En un principio, uno de los motivos por los que se eligió MySQL como base de datos fue su licencia de software libre tanto para uso comercial o no comercial. Sin embargo, a partir de las versiones 4.0 se comenzó a distribuir este software con licencia dual, en las que las licencias para uso comercial no son gratuitas. Nuestra aplicación no es actualmente una aplicación comercial, se trata de una aplicación con uso sin ánimo de lucro, por lo que podemos utilizar este servidor de datos gratuitamente, incluso con versiones posteriores a la 4.0.

Desde finales de 2003, existen también versiones 5.0.

10.3.7 Connector/J 3.1.6.

Inicialmente se utilizó el driver Connector/J 3.0.7 de 2003. Esta versión funcionaba correctamente con la versión de MySQL Database Server 3.23.52. Sin embargo, cuando se actualizó a una versión de MySQL más moderna, comprobamos que daba lugar a fallos. Investigando se descubrió, que el este esta versión de MySQL provocaba errores con el driver 3.0.7, por ello se decidió cambiar también el driver a la versión Connector/J 3.1.6 de Diciembre de 2004, solucionando los fallos anteriores.

Actulmente ya existen versiones 3.2.x aunque estas son versiones alpha.

10.3.8 Kannel version 1.4.0.

Las pruebas con el servidor Kannel se comenzaron a realizar con la versión 1.2.0 de Noviembre 2002. Esta versión mostraba inicialmente algunas incompatibilidades con el módem GSM que pretendiamos utilizar, el módem interno de un teléfono alcatel OT535. Estos errores se corrigieron modificando el código en C de la pasarela adecuadamente.

Cuando se comenzaron las pruebas para conectar la aplicación con la pasarela GSM se actualizó la pasarela a la versión 1.4.0 de Noviembre de 2004. Esta versión ya era compatible con nuestro módem GSM.

10.4 INSTALACIÓN Y CONFIGURACIÓN BÁSICA DE HERRAMIENTAS

10.4.1 Instalación y configuración básica de MySQL Database Server 4.1.9

Descargar esta versión del software del portal de MySQL, http://www.mysql.com. Buscar en *"downloads → MySQL Community Edition -- Database server and client → older releases → MySQL Database Server 4.1"* y descargar la versión 4.1.9 para Microsoft Windows 32. (Probar enlace http://downloads.mysql.com/archives.php).

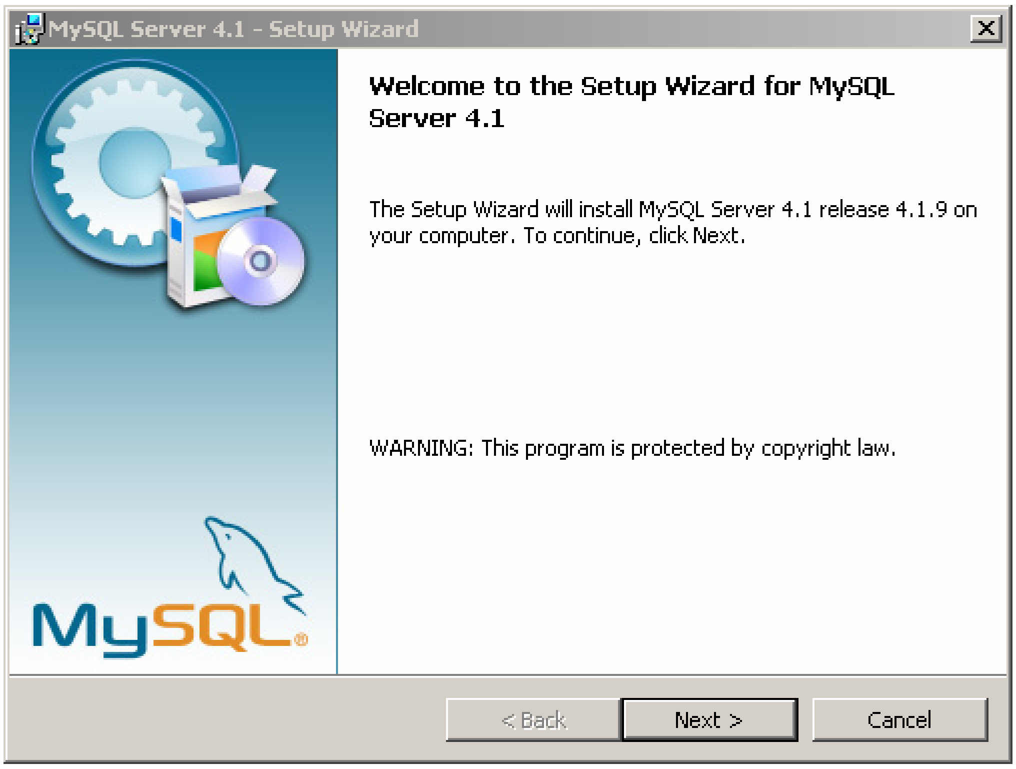
Si se tiene una versión de MySQL ya instalada, y quiere instalarse esta, es conveniente desinstalar la ya existente. Si la instalación se realizó desde un ejecutable o desde el panel de agregar programas de Windows, la desinstalación se realizará desde el mismo modo. Existen versiones anteriores en las que posiblemente sea necesario eliminar los archivos manualmente. En ambos casos es conveniente asegurarse que el fichero my.ini se ha eliminado y que el registro *"HKEY_LOCAL_MACHINE/SYSTEM/CurrentControlSet/Service"* también ha sido eliminado. Para esta última acción es posible que sea necesario ejecutar **`regedit`** desde la línea de comandos.

También es posible tener varios servidores MySQL en la misma máquina, sin embargo no es recomendable si no es por motivos de desarrollo y pruebas de compatibilidad, ya que requiere conocimientos más avanzados de configuración y si no se prepara correctamente puede ocasionar multitud de fallos. Algunas de las opciones de configuración que habrá que tener en cuenta son las siguientes: utilizar puertos distintos para los distintos servidores (por defecto se utiliza el 3306), utilizar nombres distintos de servicio si se ejecuta como servicio de windows (nombre por defecto MySQL, pero podría por ejemplo especificar la versión en el nombre), también se puede utilizar distinta ruta para la ubicación de datos y configuración. Puede que estas indicaciones no sean suficientes para mantener varios servidores de MySQL.

A continuación se muestra la instalación para *"mysql-essential-4.1.9-win32.msi"*. Este paquete es igual al *"mysql-4.1.9-win32.msi"* aunque ocupa menos ya que no incluye la documentación.

La instalación de ambos paquetes es idéntica y similar a la de otras versiones 4.x. Para las versiones 3.x la instalación se tendrá que hacer de modo manual ya que no ofrecen un paquete de auto instalación. Se muestra una instalación básica muy parecida a la instalación utilizada por defecto.

- Ejecutar archivo. Aparecerá la siguiente pantalla:


MySQL Server 4.1 - Setup Wizard

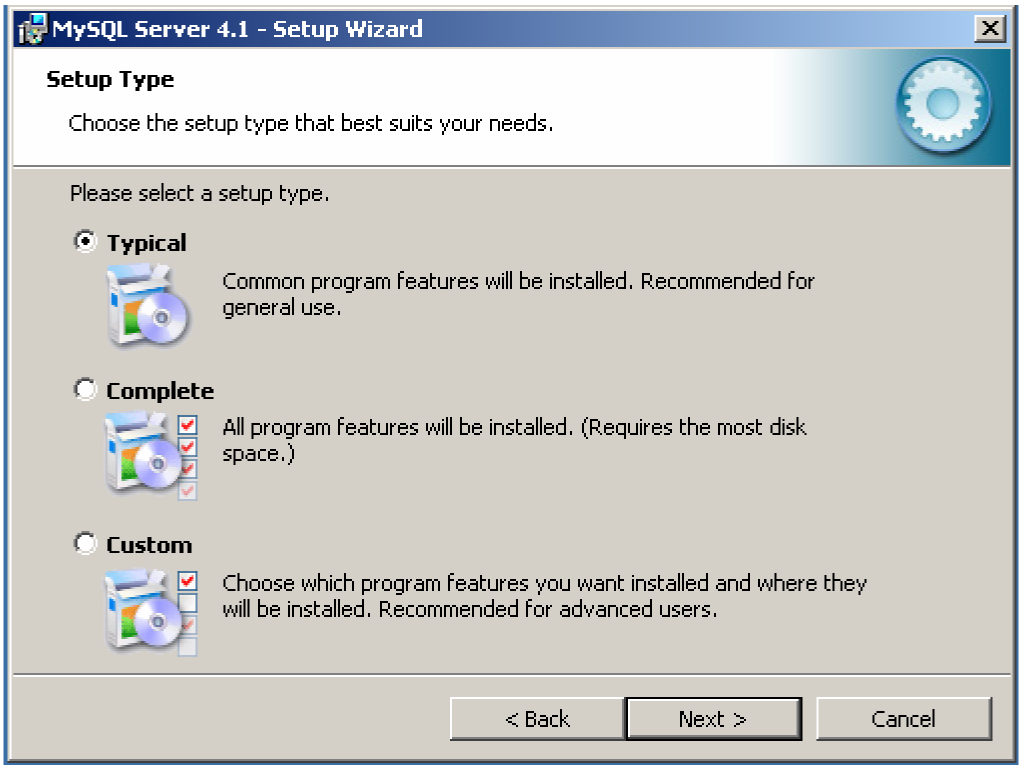
Welcome to the Setup Wizard for MySQL Server 4.1

The Setup Wizard will install MySQL Server 4.1 release 4.1.9 on your computer. To continue, click Next.

WARNING: This program is protected by copyright law.

F. 10-1: Instalación MySQL Database Server 4.1.9.

- Continuamos (Next).



MySQL Server 4.1 - Setup Wizard

Setup Type

Choose the setup type that best suits your needs.

Please select a setup type.

[✓ Typical]

Common program features will be installed. Recommended for general use.

Complete

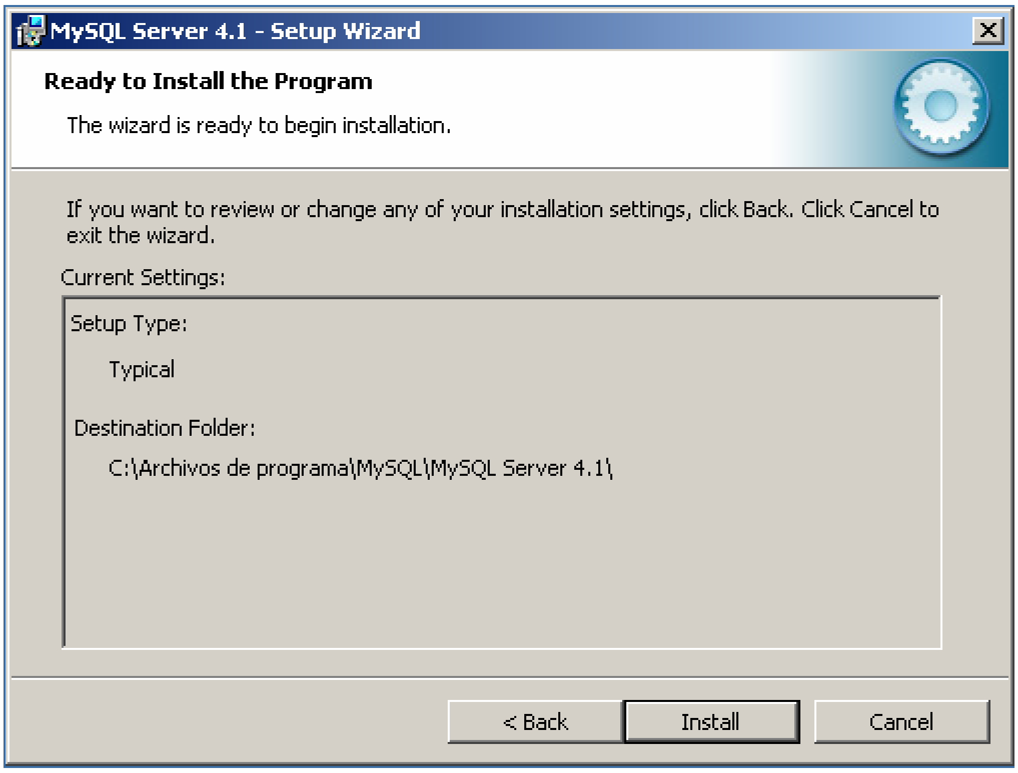
All program features will be installed. (Requires the most disk space.)

Custom

Choose which program features you want installed and where they will be installed. Recommended for advanced users.

F. 10-2: Instalación MySQL Database Server 4.1.9.

- Elegimos la configuración típica (Typical) y continuamos (Next). Aparecerá una pantalla con los datos de configuración para la instalación.


MySQL Server 4.1 - Setup Wizard

Ready to Install the Program

The wizard is ready to begin installation.

If you want to review or change any of your installation settings, click Back. Click Cancel to exit the wizard.

Current Settings:

Setup Type:

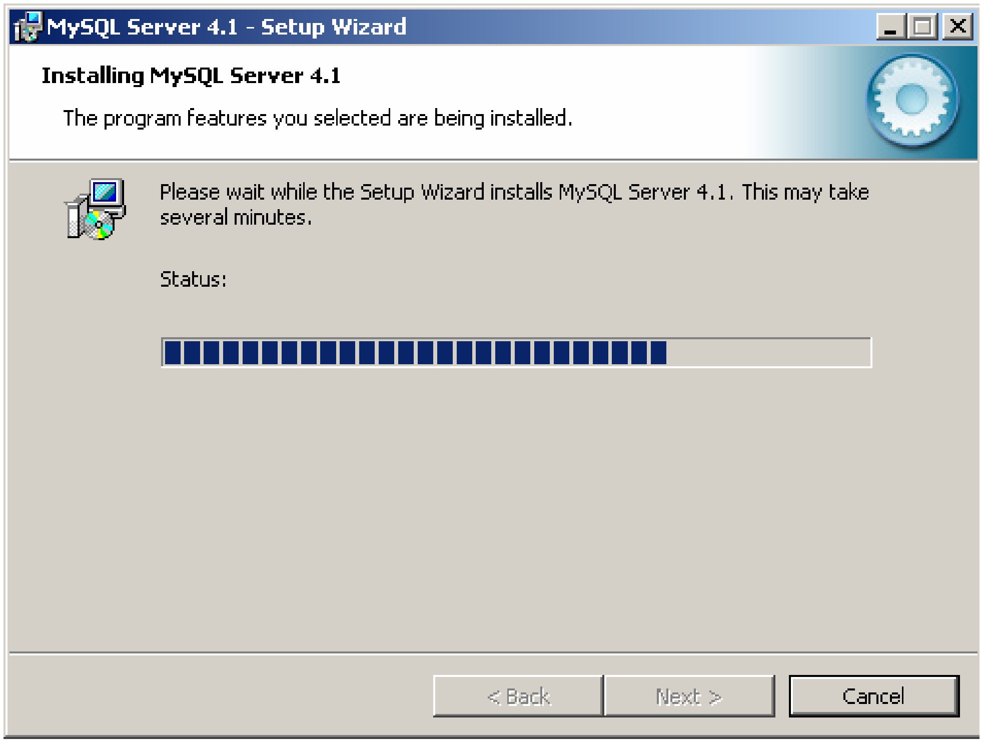
Typical

Destination Folder:

C:\Archivos de programa\MySQL\MySQL Server 4.1\

F. 10-3: Instalación MySQL Database Server 4.1.9.

- La instalación se realizará en *"C:\Archivos de programa\MySQL\MySQL Server 4.1\"*. No nos ofrece opción para elegir otra ubicación. Comenzamos la instalación(Install).



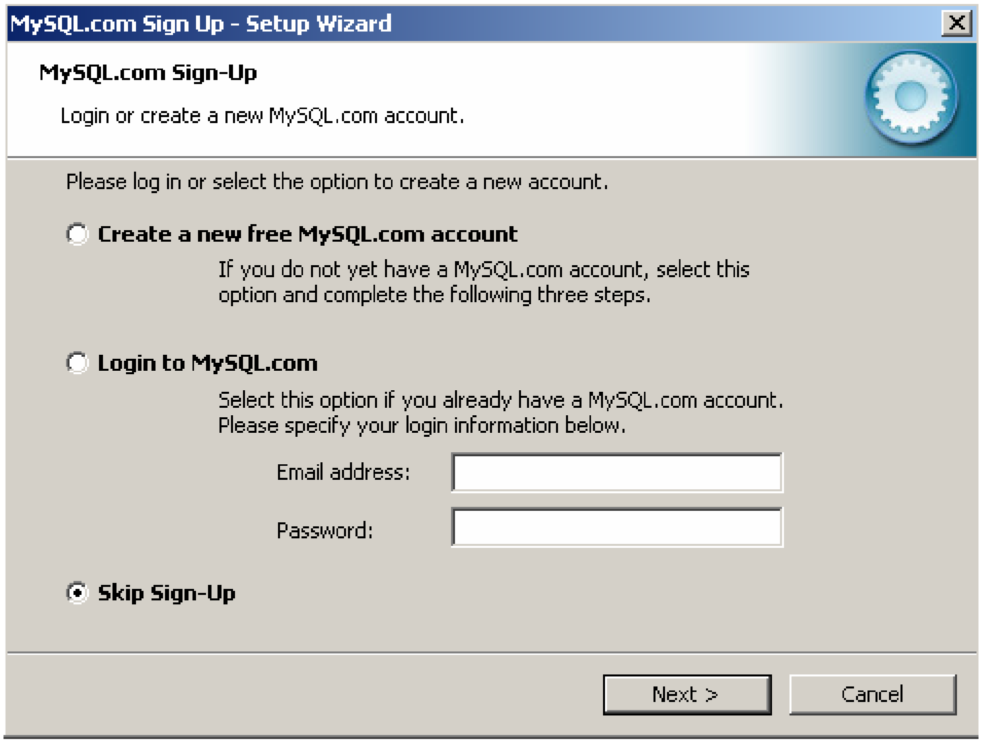
Installing MySQL Server 4.1

The program features you selected are being installed.

Please wait while the Setup Wizard installs MySQL Server 4.1. This may take several minutes.

Status:

F. 10-4: Instalación MySQL Database Server 4.1.9.



MySQL.com Sign-Up

Login or create a new MySQL.com account.

Please log in or select the option to create a new account.

○ **Create a new free MySQL.com account**

If you do not yet have a MySQL.com account, select this option and complete the following three steps.

○ **Login to MySQL.com**

Select this option if you already have a MySQL.com account. Please specify your login information below.

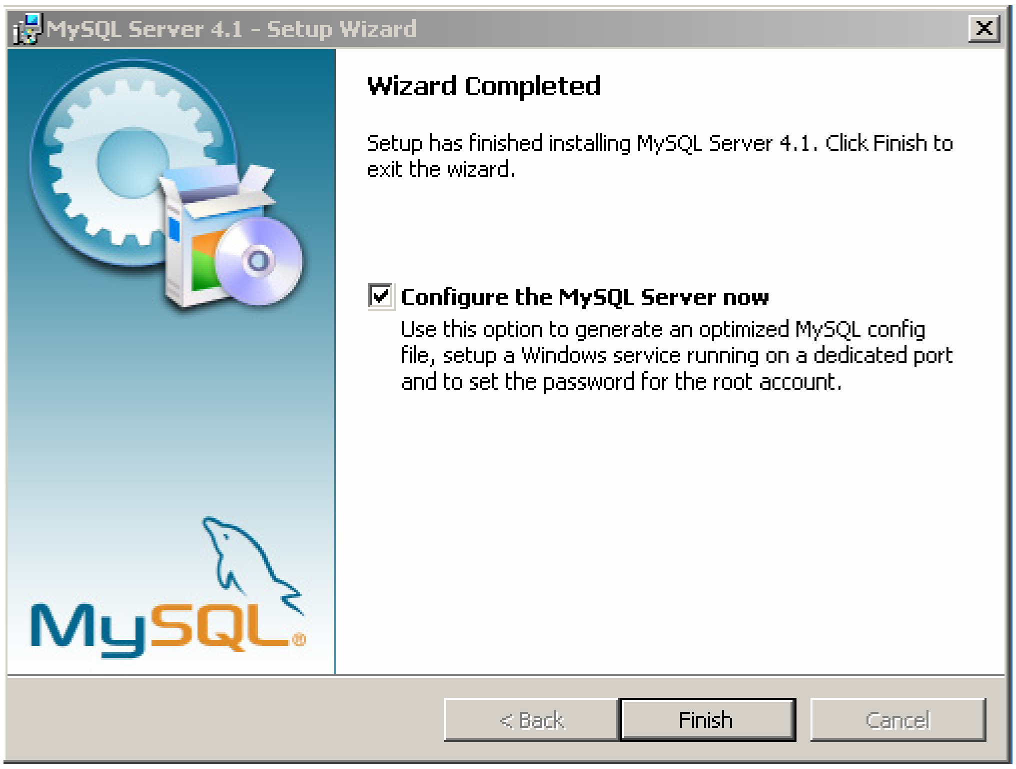
Email address:

Password:

[✓ **Skip Sign-Up**]

F. 10-5: Instalación MySQL Database Server 4.1.9.

- Aunque nos ofrecerá la posibilidad de registrarnos podemos marcar saltar este paso (Skip Sign-Up) y continuamos (Next).


MySQL Server 4.1 - Setup Wizard

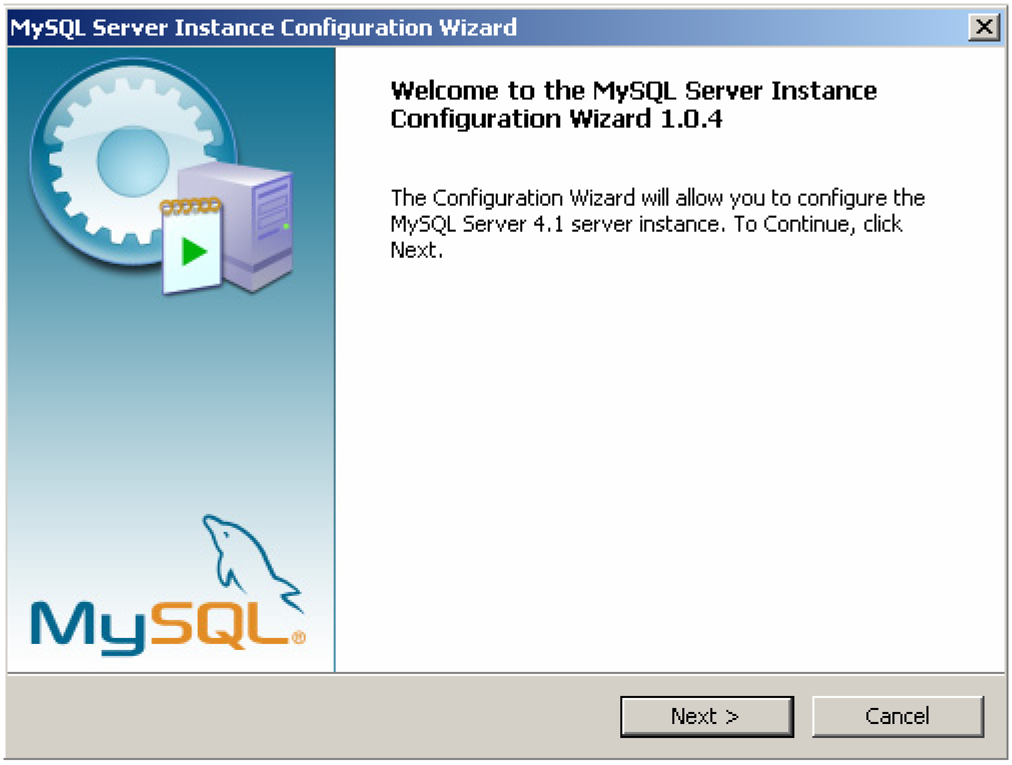
Wizard Completed

Setup has finished installing MySQL Server 4.1. Click Finish to exit the wizard.

[✓] **Configure the MySQL Server now**
Use this option to generate an optimized MySQL config file, setup a Windows service running on a dedicated port and to set the password for the root account.

F. 10-6: Instalación MySQL Database Server 4.1.9.

- Dejamos marcada la opción de configurar MySQL Server y finalizamos (Finish).

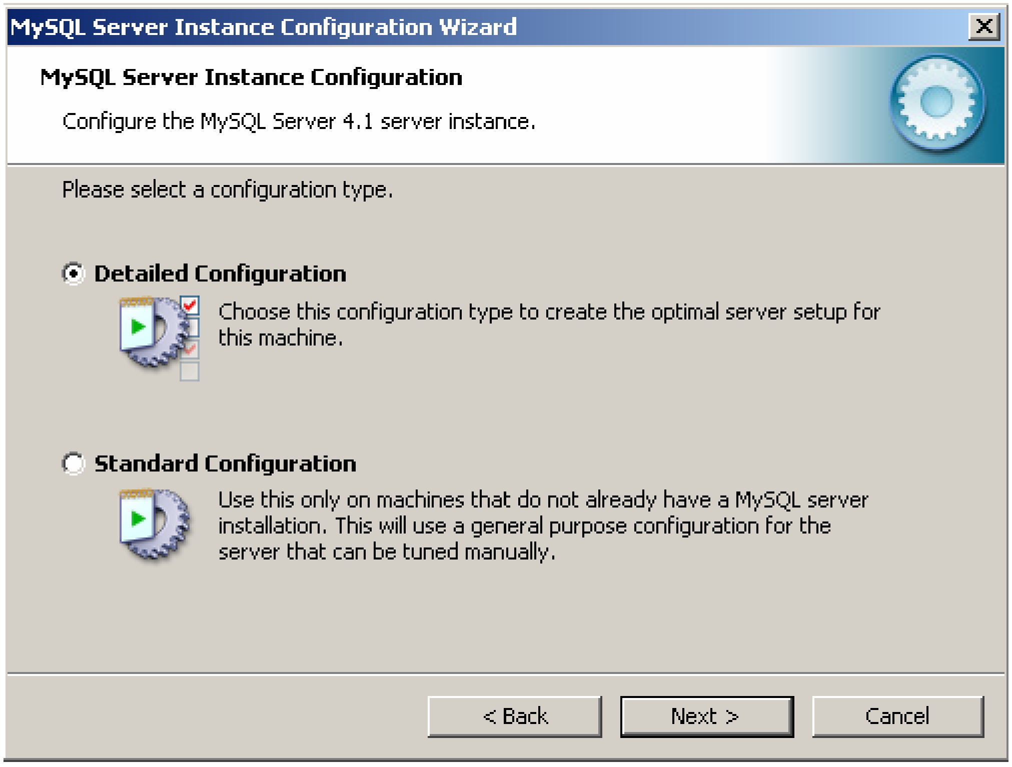

MySQL Server Instance Configuration Wizard

Welcome to the MySQL Server Instance Configuration Wizard 1.0.4

The Configuration Wizard will allow you to configure the MySQL Server 4.1 server instance. To Continue, click Next.

F. 10-7: Instalación MySQL Database Server 4.1.9.

- Comenzamos la configuración del servidor (Next).


MySQL Server Instance Configuration Wizard

MySQL Server Instance Configuration

Configure the MySQL Server 4.1 server instance.

Please select a configuration type.

[✓ Detailed Configuration]

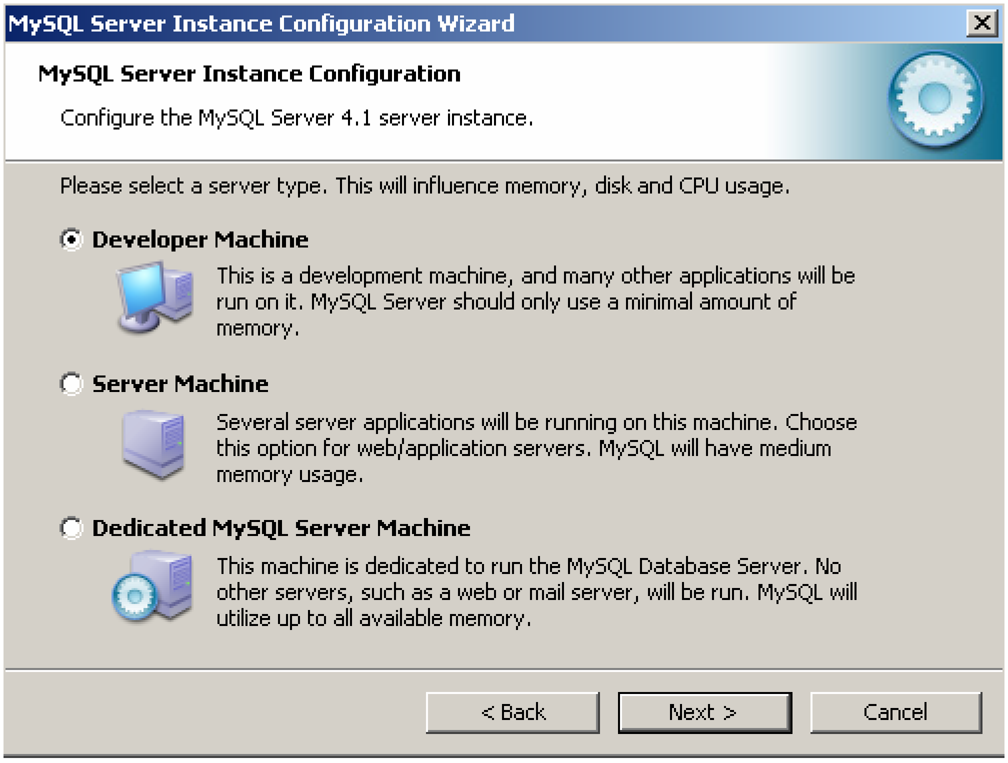
Choose this configuration type to create the optimal server setup for this machine.

Standard Configuration

Use this only on machines that do not already have a MySQL server installation. This will use a general purpose configuration for the server that can be tuned manually.

F. 10-8: Instalación MySQL Database Server 4.1.9.

- Elegimos la opción marcada, configuración detallada. Continuamos (Next).



MySQL Server Instance Configuration Wizard

MySQL Server Instance Configuration

Configure the MySQL Server 4.1 server instance.

Please select a server type. This will influence memory, disk and CPU usage.

[✓ **Developer Machine**]

This is a development machine, and many other applications will be run on it. MySQL Server should only use a minimal amount of memory.

Server Machine

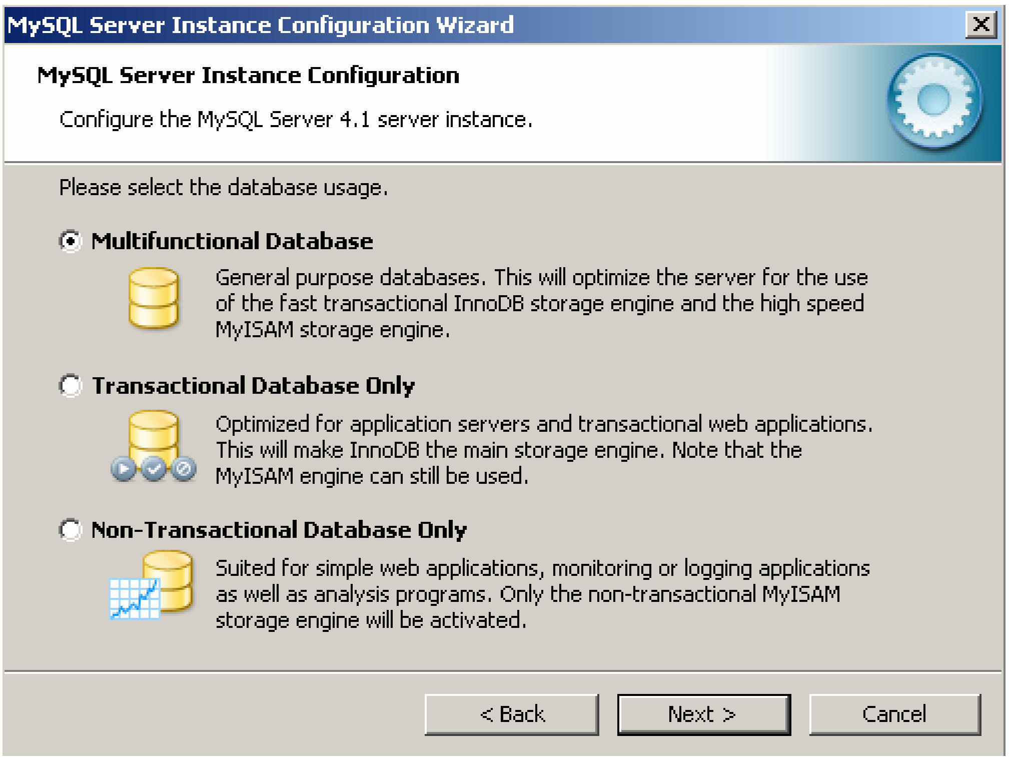
Several server applications will be running on this machine. Choose this option for web/application servers. MySQL will have medium memory usage.

Dedicated MySQL Server Machine

This machine is dedicated to run the MySQL Database Server. No other servers, such as a web or mail server, will be run. MySQL will utilize up to all available memory.

F. 10-9: Instalación MySQL Database Server 4.1.9.

- Con la opción marcada el uso de memoria es mínimo. Para nuestros propósitos de prueba esta opción es adecuada. Continuamos (Next).


MySQL Server Instance Configuration Wizard

MySQL Server Instance Configuration

Configure the MySQL Server 4.1 server instance.

Please select the database usage.

(●) **Multifunctional Database**

General purpose databases. This will optimize the server for the use of the fast transactional InnoDB storage engine and the high speed MyISAM storage engine.

( ) **Transactional Database Only**

Optimized for application servers and transactional web applications. This will make InnoDB the main storage engine. Note that the MyISAM engine can still be used.

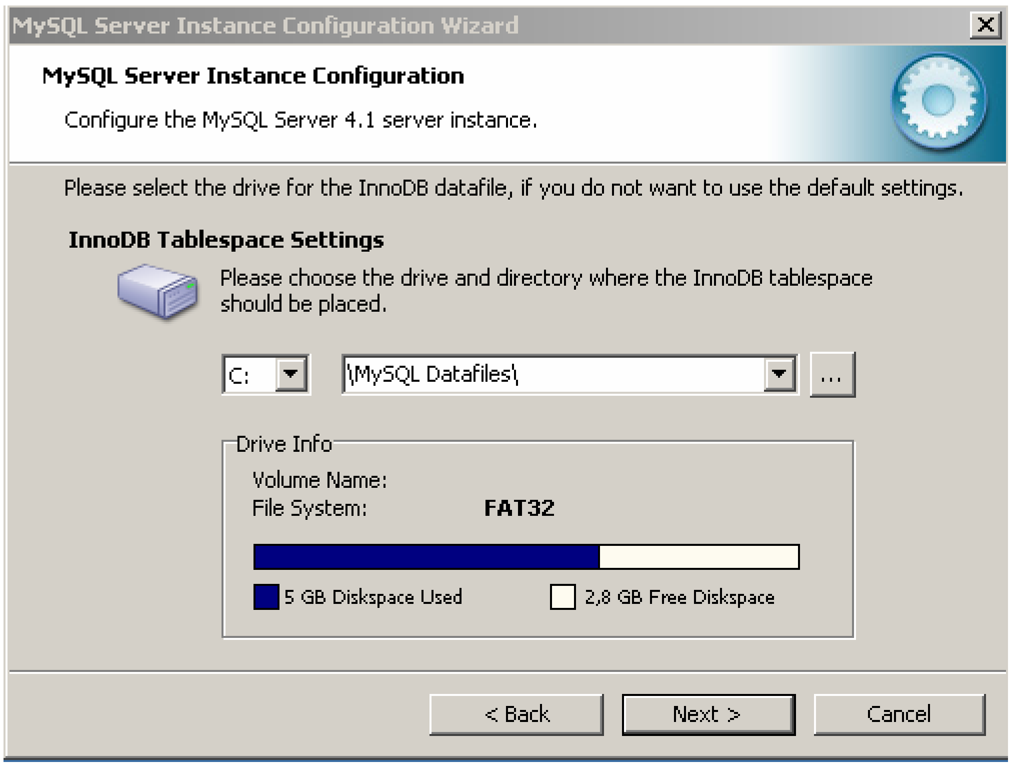
( ) **Non-Transactional Database Only**

Suited for simple web applications, monitoring or logging applications as well as analysis programs. Only the non-transactional MyISAM storage engine will be activated.

< Back | Next > | Cancel

F. 10-10: Instalación MySQL Database Server 4.1.9.

- Elegimos la configuración de una base de datos multifuncional. MySQL nos ofrece dos tipos de almacenamiento: transaccional (InnoBD) y almacenamiento no transaccional pero de alta velocidad (MyISAM). Para nuestra aplicación ejemplo sólo utilizaremos almacenamiento transaccional, sin embargo, configurarlo como multifuncional nos proporcional más flexibilidad para otros almacenamientos ajenos a nuestra aplicación. Continuamos (Next).


MySQL Server Instance Configuration Wizard

MySQL Server Instance Configuration

Configure the MySQL Server 4.1 server instance.

Please select the drive for the InnoDB datafile, if you do not want to use the default settings.

InnoDB Tablespace Settings

Please choose the drive and directory where the InnoDB tablespace should be placed.

C: | \MySQL Datafiles\ | ...

Drive Info

Volume Name:

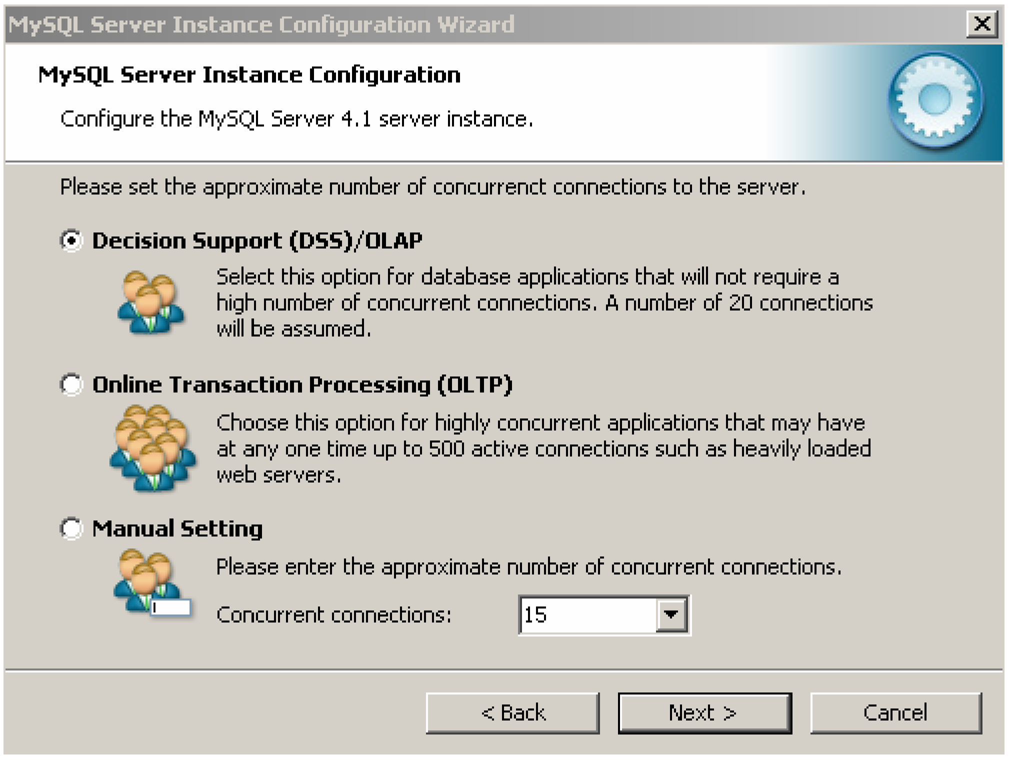
File System: **FAT32**

■ 5 GB Diskspace Used □ 2,8 GB Free Diskspace

< Back | Next > | Cancel

F. 10-11: Instalación MySQL Database Server 4.1.9.

- Podemos elegir la ubicación de nuestros archivos de datos. Continuamos (Next).


MySQL Server Instance Configuration Wizard

MySQL Server Instance Configuration

Configure the MySQL Server 4.1 server instance.

Please set the approximate number of concurrenct connections to the server.

[✓ **Decision Support (DSS)/OLAP**]

Select this option for database applications that will not require a high number of concurrent connections. A number of 20 connections will be assumed.

○ **Online Transaction Processing (OLTP)**

Choose this option for highly concurrent applications that may have at any one time up to 500 active connections such as heavily loaded web servers.

○ **Manual Setting**

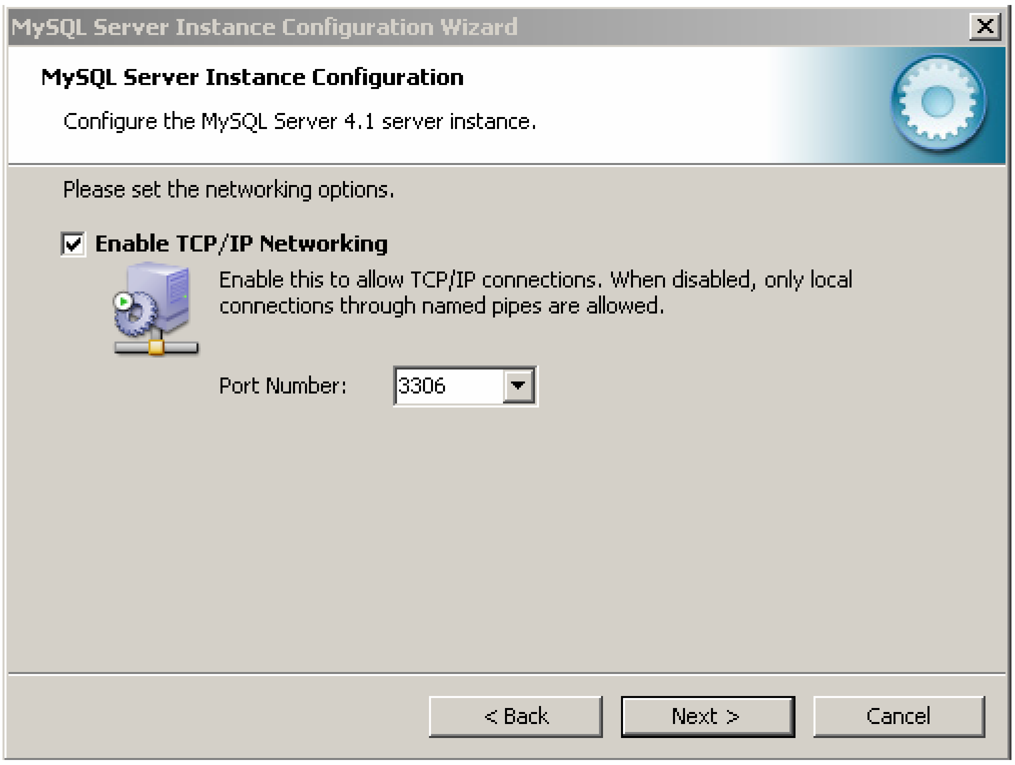
Please enter the approximate number of concurrent connections.

Concurrent connections: 15

< Back | Next > | Cancel

F. 10-12: Instalación MySQL Database Server 4.1.9.

- No esperamos muchas conexiones concurrentes a nuestra base de datos, por lo que dejamos marcada la opción por defecto. Continuamos (Next).



MySQL Server Instance Configuration Wizard

MySQL Server Instance Configuration

Configure the MySQL Server 4.1 server instance.

Please set the networking options.

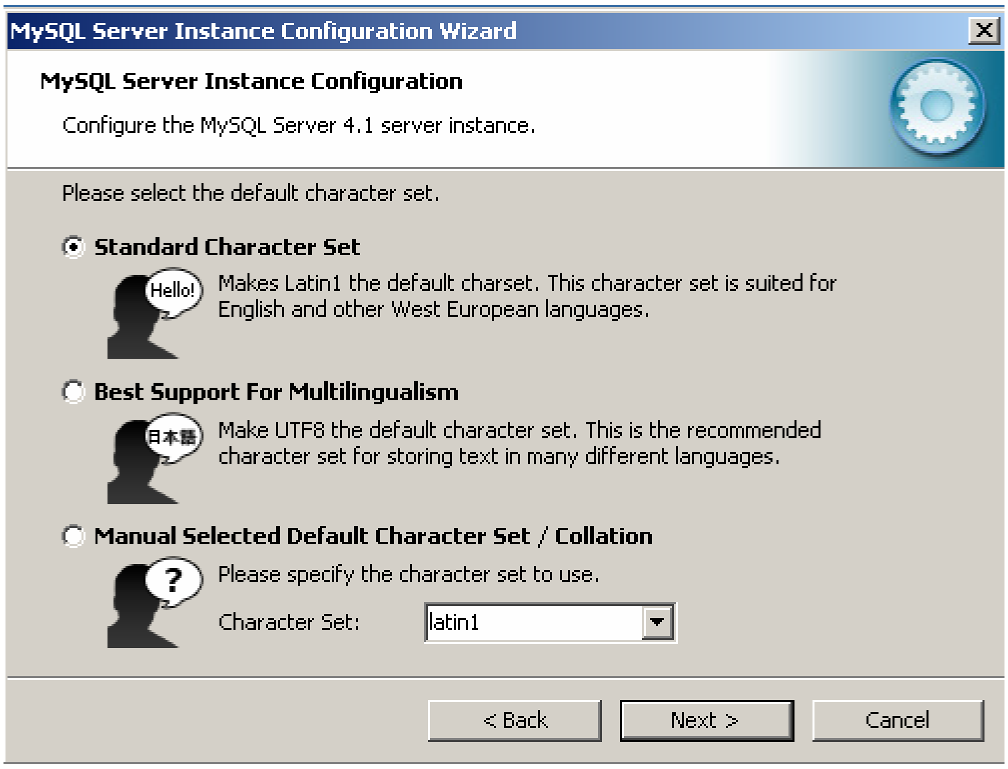
[✓] **Enable TCP/IP Networking**

Enable this to allow TCP/IP connections. When disabled, only local connections through named pipes are allowed.

Port Number: 3306

F. 10-13: Instalación MySQL Database Server 4.1.9.

- Seleccionamos el puerto para conexiones TCP/IP. Continuamos (Next). Si este puerto estuviese ocupado, nos mostraría un mensaje y podremos elegir otro puerto.


MySQL Server Instance Configuration Wizard

MySQL Server Instance Configuration

Configure the MySQL Server 4.1 server instance.

Please select the default character set.

[✓ Standard Character Set]

Makes Latin1 the default charset. This character set is suited for English and other West European languages.

Best Support For Multilingualism

Make UTF8 the default character set. This is the recommended character set for storing text in many different languages.

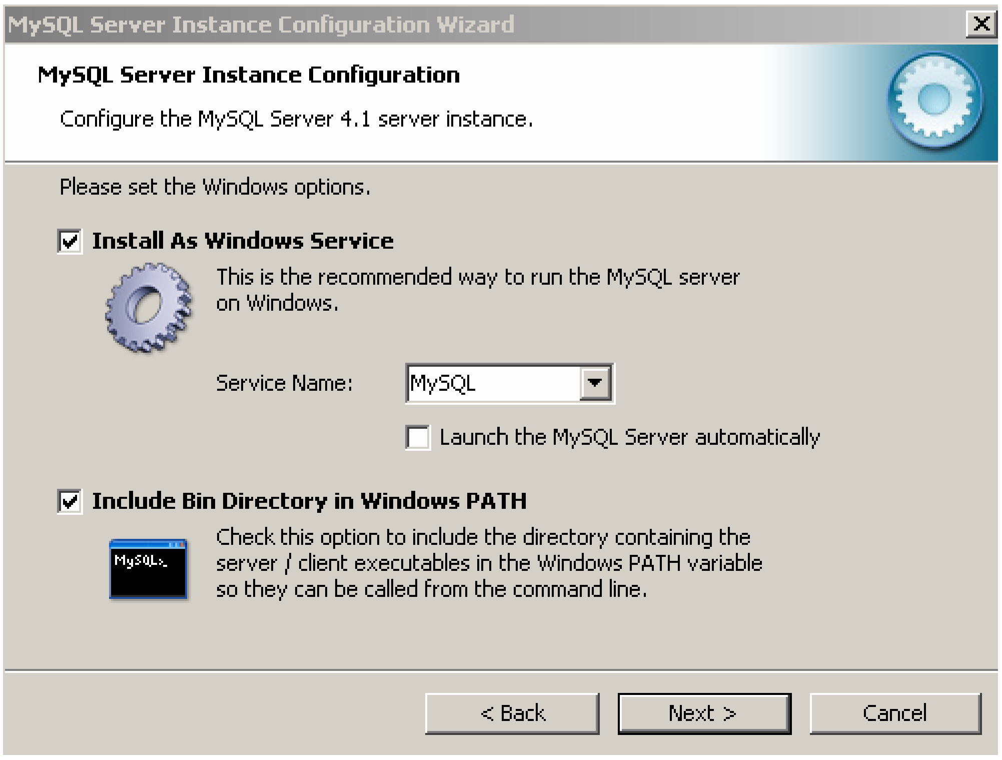
Manual Selected Default Character Set / Collation

Please specify the character set to use.

Character Set: latin1

F. 10-14: Instalación MySQL Database Server 4.1.9.

- También dejamos la opción de caracteres por defecto, que reconoce caracteres latinos. Continuamos (Next).



MySQL Server Instance Configuration

Configure the MySQL Server 4.1 server instance.

Please set the Windows options.

☑ Install As Windows Service

This is the recommended way to run the MySQL server on Windows.

Service Name: MySQL

☐ Launch the MySQL Server automatically

☑ Include Bin Directory in Windows PATH

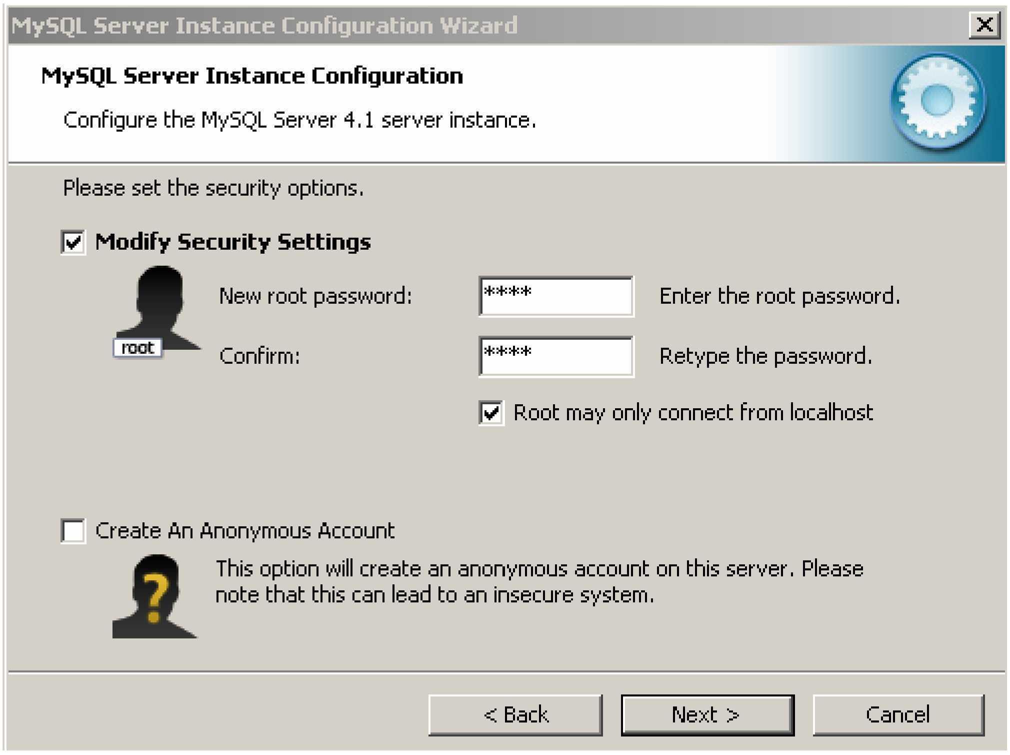
Check this option to include the directory containing the server / client executables in the Windows PATH variable so they can be called from the command line.

F. 10-15: Instalación MySQL Database Server 4.1.9.

- A continuación nos ofrece la posibilidad de conectar el servidor de MySQL automáticamente al iniciar Windows. Nosotros, aunque mantenemos el servidor como un servicio de Windows, deseleccionamos la opción de conectarse automáticamente. Cuando queramos utilizarlo tendremos que conectarlo nosotros desde la línea de comandos o con algunas de las herramientas que MySQL proporciona.

- Por defecto, la opción de incluir el directorio Bin en el path de Windows está deseleccionada. Nosotros hemos activado esta casilla para poder ejecutar el cliente o servidor de MySQL fácilmente desde la línea de comandos.

- Continuamos (Next).


MySQL Server Instance Configuration Wizard

MySQL Server Instance Configuration

Configure the MySQL Server 4.1 server instance.

Please set the security options.

[✓] **Modify Security Settings**

New root password: **** Enter the root password.

Confirm: **** Retype the password.

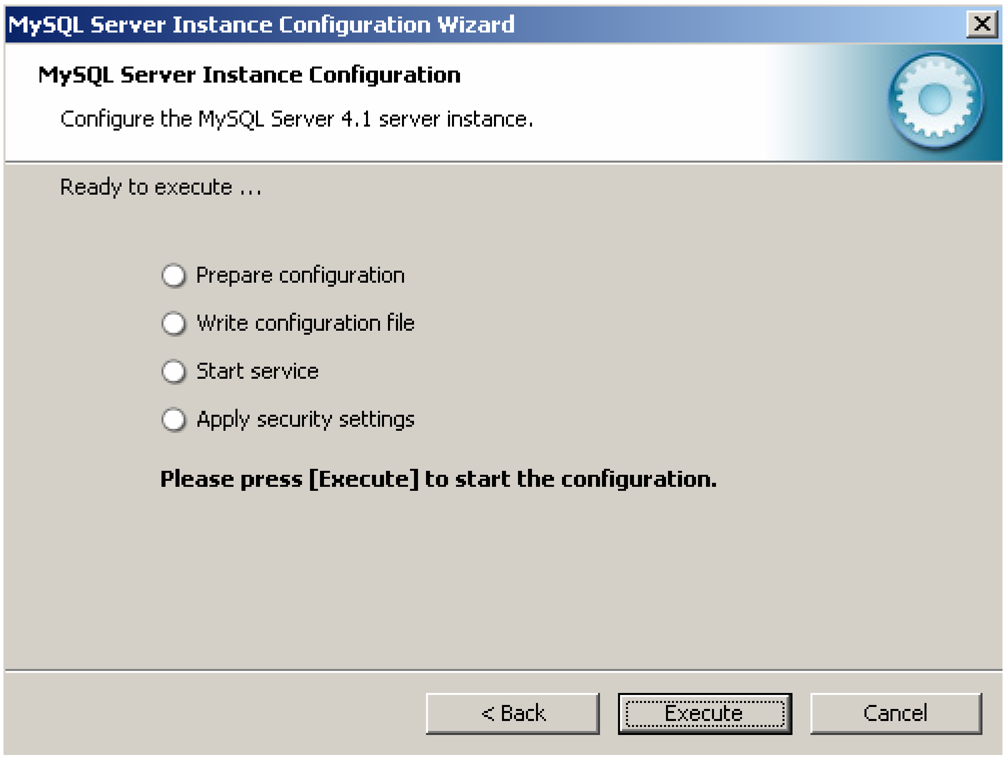
[✓] Root may only connect from localhost

[ ] Create An Anonymous Account

This option will create an anonymous account on this server. Please note that this can lead to an insecure system.

F. 10-16: Instalación MySQL Database Server 4.1.9.

- Introducimos la contraseña para el usuario root. El usuario root es aquel que tiene todos los permisos. Nosotros hemos marcado la opción de conectarse como root sólo de modo local. Continuamos (Next).


MySQL Server Instance Configuration Wizard

MySQL Server Instance Configuration

Configure the MySQL Server 4.1 server instance.

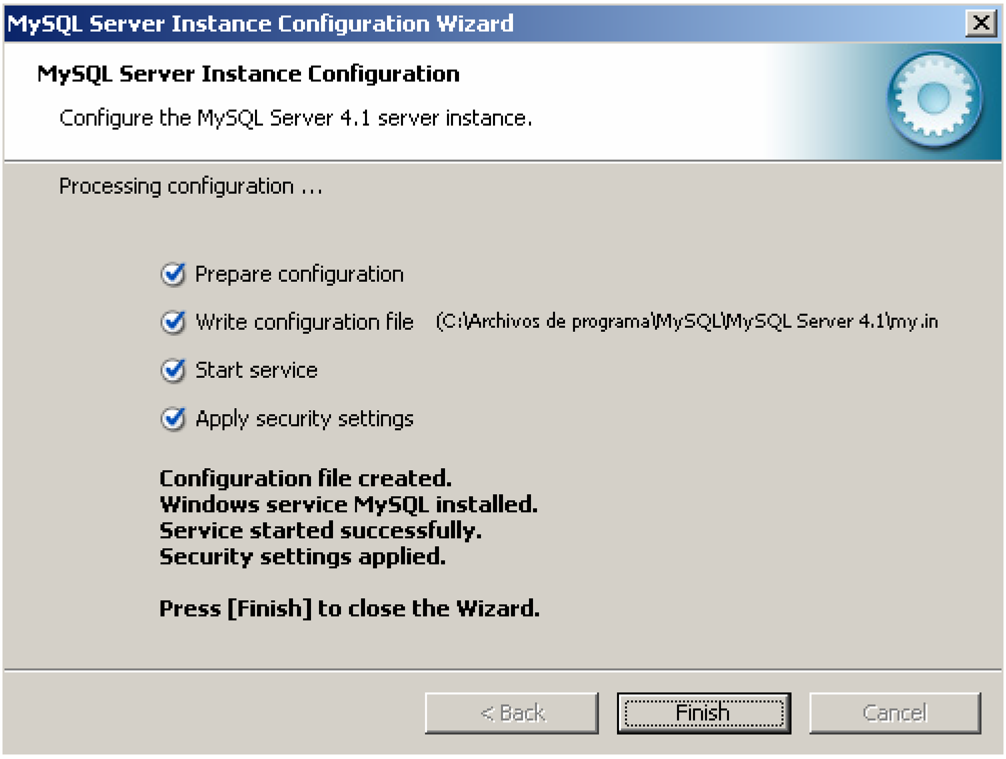
Ready to execute ...

- Prepare configuration
- Write configuration file
- Start service
- Apply security settings

Please press [Execute] to start the configuration.

F. 10-17: Instalación MySQL Database Server 4.1.9.

- Ejecutamos para comenzar la configuración (Execute).


MySQL Server Instance Configuration Wizard
MySQL Server Instance Configuration
< Back
Finish
Cancel

Configure the MySQL Server 4.1 server instance.

Processing configuration ...

✓ Prepare configuration
✓ Write configuration file (C:\Archivos de programa\MySQL\MySQL Server 4.1\my.in
✓ Start service
✓ Apply security settings

Configuration file created.
Windows service MySQL installed.
Service started successfully.
Security settings applied.

Press [Finish] to close the Wizard.

F. 10-18: Instalación MySQL Database Server 4.1.9.

- Finalizamos la configuración (Finish). Si no se eliminaron los datos de posibles instalaciones anteriores puede que la nueva configuración tenga efecto.

- Si se produce un error al intentar iniciar el servicio (Start service) posiblemente sea causado por instalaciones anteriores que no se han eliminado correctamente o que entran en conflicto con esta.

- Si se produce un error al intentar aplicar las configuraciones de seguridad (Apply security settings) puede estar originado por la existencia de configuraciones y datos de versiones anteriores.

- Si todo se realizado correctamente hemos configurado nuestro servidor del siguiente modo:

  Nombre del servicio: MySQL
  Nombre del servidor (Server Host): localhost
  Puerto: 3306

Usuario con todos los privilegios: root
(Clave de usuario root la introducida durante la configuración)

Para comprobar si el servicio MySQL (nombre que le hemos dado durante la configuración) se ha activado abrimos la consola de comandos de Windows y ejecutamos por ejemplo **`net start | more`**. Con esta opción podemos ver todos los servicios que Windows ha iniciado. Si acabamos de instalar nuestro servidor y configurado como se ha explicado anteriormente, en la lista aparecerá MySQL.

La opción **`more`** se utiliza simplemente para leer más fácilmente los datos en pantalla.

Para detener el servicio utilizamos **`net stop mysql`** y para volverlo a arrancar **`net start mysql`**. Estos comandos pueden introducirse en el menú de inicio de Windows en *"Ejecutar"*.

A continuación ya se puede trabajar con el servidor. MySQL incluye un cliente en línea de comandos. Cuando se ejecuta te pide una contraseña, la del usuario root. Este fue el usuario que configuramos inicialmente. Para que el cliente pueda conectarse al introducir la contraseña correctamente es necesario que el servicio de MySQl esté iniciado.

Si se quiere utilizar el cliente de comandos con otro usuario (creado previamente), se puede introducir en *"Ejecutar"* o bien en la consola de comandos de Windows **`mysql –u [nombre_usuario] –p`**. Se abrirá la consola de MySQL y pedirá una contraseña, esta vez la del usuario.

Para detener el cliente de MySQL basta con escribir **`quit`** o **`exit`** junto al shell de mysql. Esta acción no parará el servicio de MySQL, es decir, el servidor seguirá activo, lo único que hemos parado ha sido el cliente, hemos cerrado la conexión.

Si no se quiere trabajar con líneas de comando, MySQL AB proporciona herramientas de administración. Por ejemplo, MySQL Administrador 1.0 nos ofrece monitorización del servicio MySQL con facilidades de inicio y finalización, y paneles para la creación de usuarios, dotación de privilegios a estos usuarios, creación de bases de datos, de tablas, copias, etc. Estas herramientas son muy útiles cuando se maneja MySQL, sin embargo para nuestra aplicación no son imprescindibles. Se puede encontrar estas herramientas en el portal de MySQL en *"downloads → MySQL Tools"*.

10.4.2 Instalación de J2SE 5.0

Como se comenta en el capítulo **¡Error! No se encuentra el origen de la referencia. ¡Error! No se encuentra el origen de la referencia.**, en el apartado **¡Error! No se encuentra el origen de la referencia. ¡Error! No se encuentra el origen de la referencia.**, es necesario tener instalado en nuestro sistema un J2SE completo, no será suficiente con un JRE.

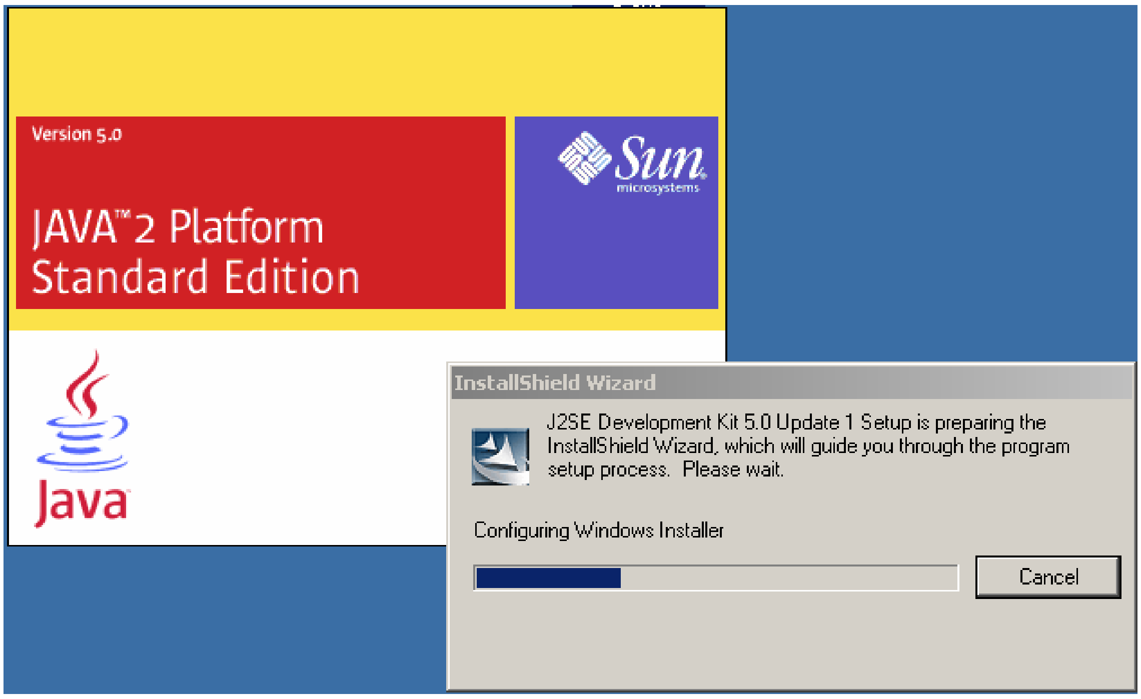
Puesto que nuestro código se ha desarrollado con la versión J2SE 5.0 Update 1, con versiones anteriores de J2SE pueden producirse errores de compilación. Con versiones posteriores no debe existir ningún problema, ya que los J2SE muestran compatibilidad hacia atrás.

Pueden existir varias máquinas virtuales en nuestro sistema, sin embargo, normalmente, esto es innecesario. Para desinstalar una versión existente de J2SE, JDK o JRE utilizar el panel de quitar programas de Windows. J2SE 5.0 incluye un JRE.

Podemos descargar el J2SE del portal Java de Sun Microsystems, http://java.sun.com. Buscar la versión requerida en http://java.sun.com/j2se/downloads/index.html.

Nosotros vamos hemos descargado el ejecutable "**jdk-1_5_0_01-windows-i586-p.exe**" que corresponde a la versión J2SE 5.0 Update 1.

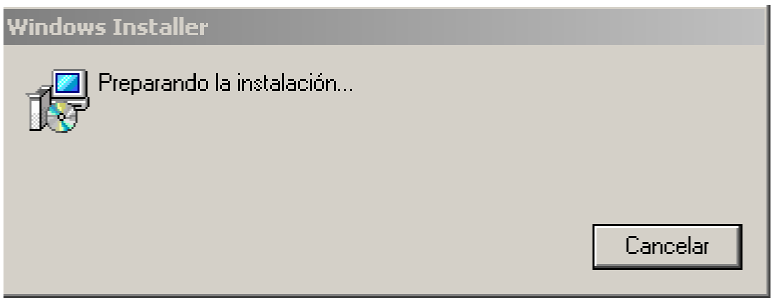
- Ejecutamos el archivo "**jdk-1_5_0_01-windows-i586-p.exe**". Aparecerán las siguientes ventanas:



J2SE Development Kit 5.0 Update 1 Setup is preparing the InstallShield Wizard, which will guide you through the program setup process. Please wait.

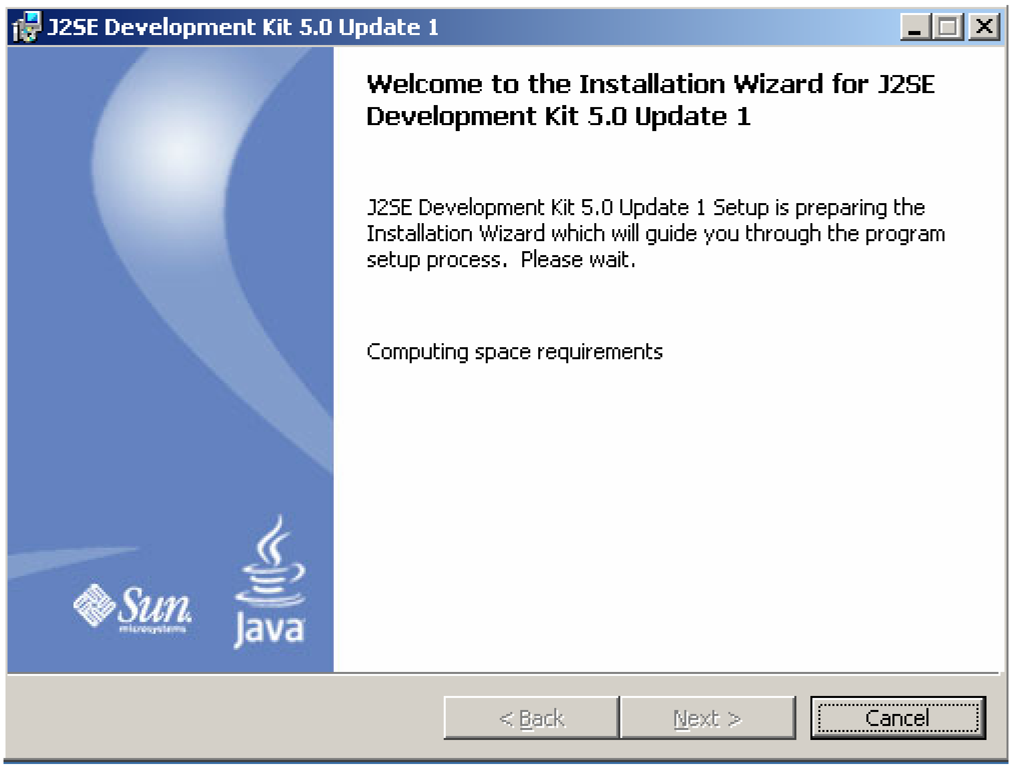
Configuring Windows Installer

F. 10-19: Instalación de J2SE 5.0.



Preparando la instalación...

F. 10-20: Instalación de J2SE 5.0.

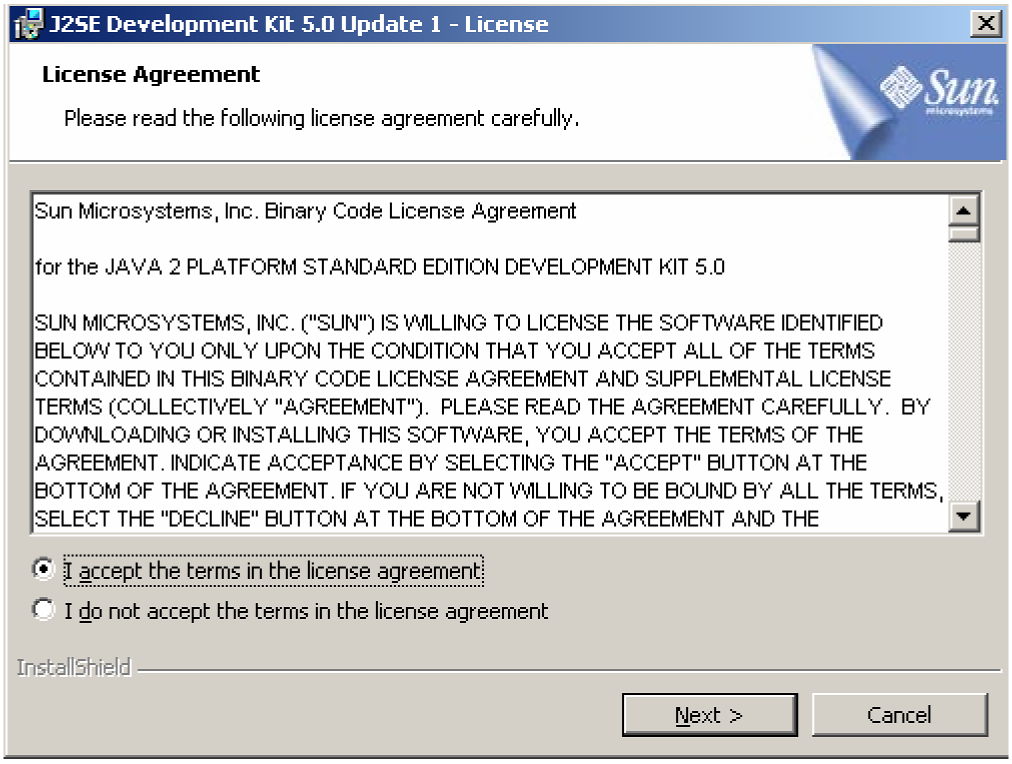


Welcome to the Installation Wizard for J2SE Development Kit 5.0 Update 1

J2SE Development Kit 5.0 Update 1 Setup is preparing the Installation Wizard which will guide you through the program setup process. Please wait.

Computing space requirements

F. 10-21: Instalación de J2SE 5.0.



License Agreement

Please read the following license agreement carefully.

Sun Microsystems, Inc. Binary Code License Agreement

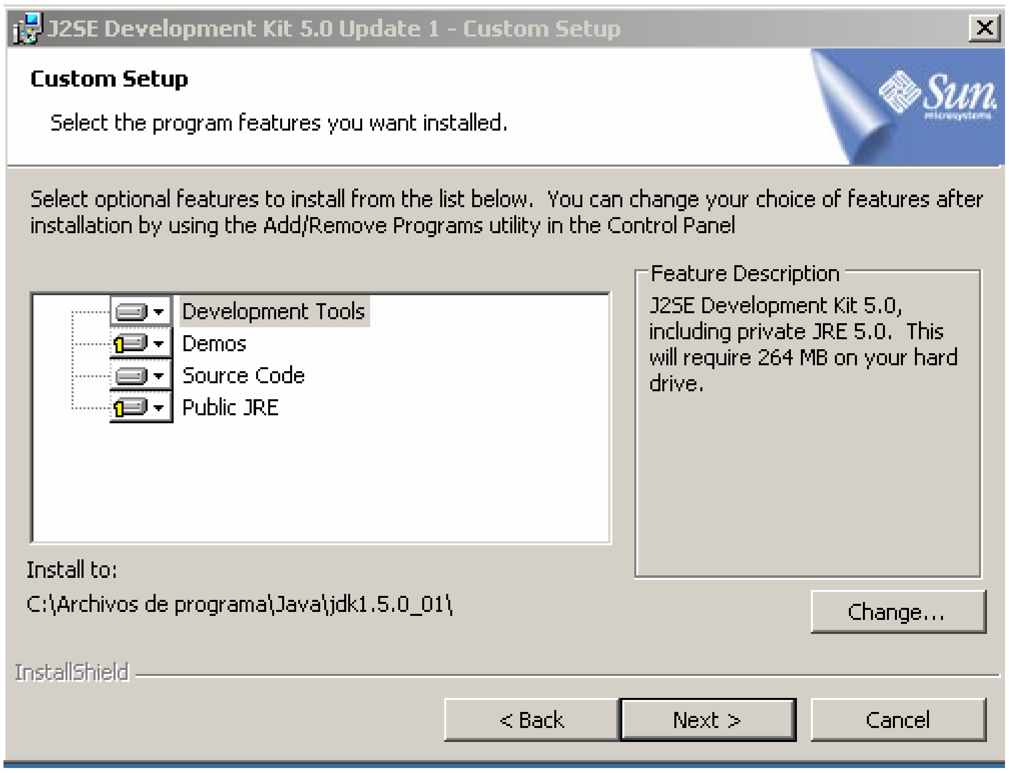
for the JAVA 2 PLATFORM STANDARD EDITION DEVELOPMENT KIT 5.0

SUN MICROSYSTEMS, INC. ("SUN") IS WILLING TO LICENSE THE SOFTWARE IDENTIFIED BELOW TO YOU ONLY UPON THE CONDITION THAT YOU ACCEPT ALL OF THE TERMS CONTAINED IN THIS BINARY CODE LICENSE AGREEMENT AND SUPPLEMENTAL LICENSE TERMS (COLLECTIVELY "AGREEMENT"). PLEASE READ THE AGREEMENT CAREFULLY. BY DOWNLOADING OR INSTALLING THIS SOFTWARE, YOU ACCEPT THE TERMS OF THE AGREEMENT. INDICATE ACCEPTANCE BY SELECTING THE "ACCEPT" BUTTON AT THE BOTTOM OF THE AGREEMENT. IF YOU ARE NOT WILLING TO BE BOUND BY ALL THE TERMS, SELECT THE "DECLINE" BUTTON AT THE BOTTOM OF THE AGREEMENT AND THE

[✓ I accept the terms in the license agreement] I do not accept the terms in the license agreement

F. 10-22: Instalación de J2SE 5.0.

- Si queremos instalar el J2SE debemos aceptar los términos de la licencia, que se corresponde al Acuerdo de Licencia de Código Binario de Sun Microsystems, Inc. Aceptamos (I accept the terms ...) y continuamos (Next).


J2SE Development Kit 5.0 Update 1 - Custom Setup

Custom Setup

Select the program features you want installed.

Select optional features to install from the list below. You can change your choice of features after installation by using the Add/Remove Programs utility in the Control Panel

- Development Tools
- Demos
- Source Code
- Public JRE

Feature Description

J2SE Development Kit 5.0, including private JRE 5.0. This will require 264 MB on your hard drive.

Install to:

C:\Archivos de programa\Java\jdk1.5.0_01\

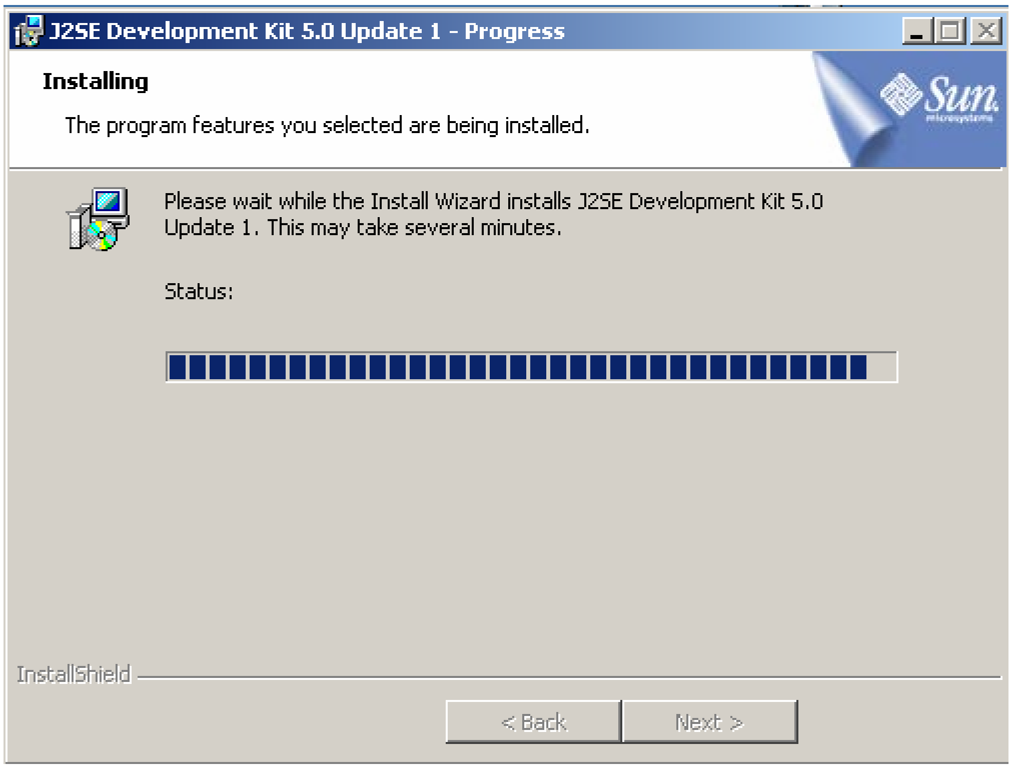
F. 10-23: Instalación de J2SE 5.0.

- Podemos elegir los componentes a instalar. Para el desarrollo de aplicaciones es imprescindible intalar las herramientas de desarrollo que incluyen el JRE (Development Tools). También es interesante instalar el código fuente (Source Code), esto nos ayudará al utilizar entornos de desarrollo ya que incluye información sobre paquetes, funciones y clases, lo que nos puede facilitar el desarrollo. Los ejemplos (Demos) y el JRE público (Public JRE) no son imprescindible para el desarrollo. Este último, puede ser útil para aplicaciones que utilice el Plug-in de Java, como lo hacen algunas aplicaciones Web, desde el navegador. Nosotros sólo vamos a instalar las herramientas necesarias para nuestra aplicación ejemplo.

- Además en esta ventana debemos elegir la ubicación los componentes que deseamos instalar. Por defecto, puede verse en la imagen que las herramientas de desarrollo

(Development Tools) se instalarán en C:\Archivos de programa\Java\jdk1.5.0_01, podemos cambiar la ubicación si queremos (Change...).

- Continuamos (Next).


J2SE Development Kit 5.0 Update 1 - Progress

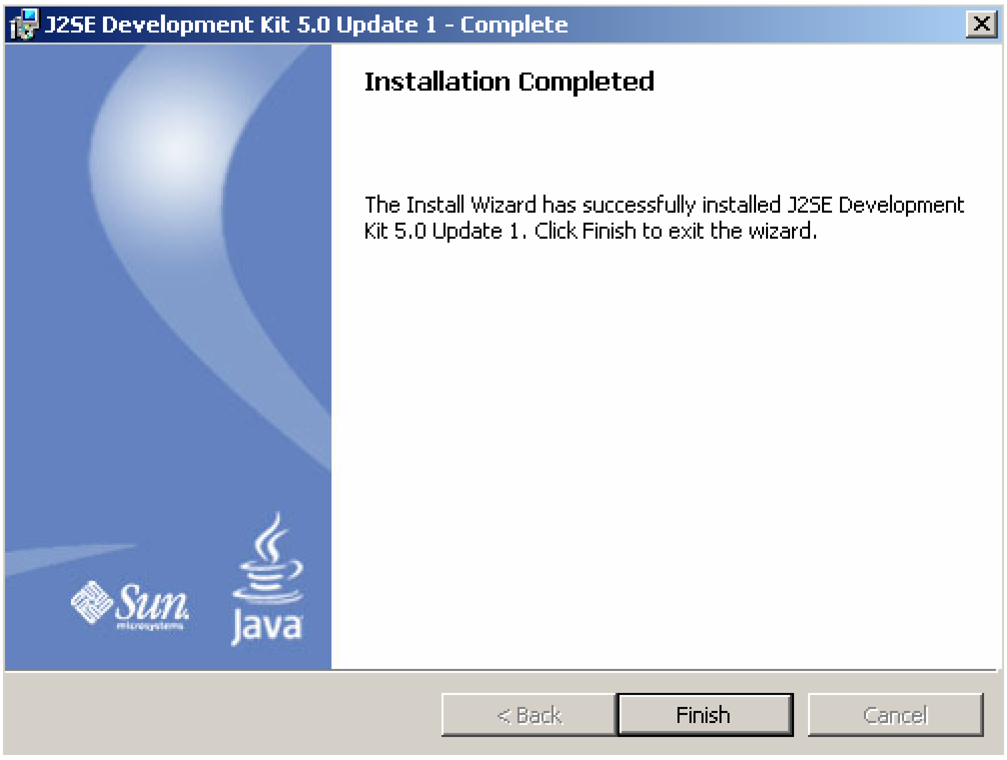
Installing

The program features you selected are being installed.

Please wait while the Install Wizard installs J2SE Development Kit 5.0 Update 1. This may take several minutes.

Status:

F. 10-24: Instalación de J2SE 5.0.



Installation Completed

The Install Wizard has successfully installed J2SE Development Kit 5.0 Update 1. Click Finish to exit the wizard.

F. 10-25: Instalación de J2SE 5.0.

- Si previamentes decidimos instalar el JRE público aparecerá una ventana en la que preguntará para cual o cuales navegadores se habilitará. En este caso bastará con seleccionar los navegadores y continuar.

- Finalizamos (Finish).

Puede que en algún otro momento quisiéramos instalar alguno de los componentes restantes, desinstalar alguno de los instalados previamente o desinstalar el J2SE completo. Si esto ocurriese se recomienda utilizar el panel de agregar o quitar programas de Windows.

10.4.3 Configuración de MySQL Database Server 4.1.9 y creación de la estructura de datos.

Una vez instalado y configurado el servidor de MySQL, necesitamos crear la base de datos que utilizará nuestra aplicación y las tablas necesarias.

Además sería conveniente la creación de un nuevo usuario con permisos limitados para conectarse a la base de datos. Este será el usuario que utilizará nuestra aplicación para conectarse al servidor.

Se ha creado un script **“application.sql”**. Este fichero realiza las siguientes acciones:

- Crea la base de datos “`apps`” si no existe.
- Se conecta a esta base de datos.
- Elimina, si existen las tablas “`holidays_request_registry`”, “`holidays_service_registry`” y “`user_profile_registry`”. Con esta acción elimina posibles datos incorrectos.
- Crea las nuevas tablas “`holidays_request_registry`”, “`holidays_service_registry`” y “`user_profile_registry`”.
- Inserta un usuario administrador con identificador de usuario “`adm`” y clave “`adm`” (aunque se inserta encriptada). En principio, se pretende que todo usuario que entre en el sistema se autentifique previamente por lo tanto es necesaria la existencia de un usuario administrador al ejecutar la aplicación. Este paso no es necesario ya que existen otras formas de insertar el primer dato en la aplicación.
- Se añaden claves primarias y foráneas a la aplicación. Este paso es posterior a la inserción de un usuario tan sólo por eficiencia. Las claves pueden ralentizar el sistema en la inserción y actualización de datos, aunque puede provocar mayor rapidez en las búsquedas.
- Crea un usuario con permisos limitados. Este usuario será el que la aplicación utilice para conectarse a la base de datos. Sólo tendrá acceso a la base de datos “`apps`”, podrá acceder a todas sus tablas pero sólo para operaciones de insertar, actualizar o buscar datos. No tendrá permiso para borrar. Este usuario se identificará con “`userapps`” y la clave que utilizará será también “`userapps`”.

Estos pasos se pueden realizar desde las herramientas de administración que MySQL proporciona creando la base de datos, las tablas, las claves y el usuario. Sin embargo, ya que se ha creado el script, es mucho más rápido utilizar el cliente de línea de comandos que se proporciona con MySQL Database Server 4.1.9.

Conectarse MySQL desde la línea de comandos como el usurio "`root`". Introducir el comando `source [ruta/]application.sql` donde ruta indica la ubicación del fichero "**application.sql**".

Se ejecutará el script realizando las operaciones previamente descritas. Finalmente, podemos cerrar la conexión.

10.4.4 Instalación de la aplicación ejemplo en Tomcat

La aplicación ejemplo se puede probar fácilmente instalando un contenedor Tomcat, y copiando el archivo `.war` de la aplicación, en la carpeta `webapps` del contenedor.

La instalación de Tomcat es muy sencilla. Basta con descargarse la versión ejecutable y seguir los pasos de la configuración por defecto.

Por defecto la dirección del contendor Tomcat será http://localhost:8080, aunque puede configurarse de otro modo.

La ruta de nuestra aplicación será pues, http://localhost:8080/Apps (versión sin SMS) o http://localhost:8080/Appssms.

10.4.5 Configuración de Kannel para la integración de servicio SMS.

La instalación de Kannel viene detallada en la documentación que se ofrece en http://www.kannel.org. También se enumeran los paquetes de Cygwin necesarios para su instalación.

Una vez instalada la pasarela es necesario arrancarla para la configuración deseada. Para nuestra ampliación utilizamos el fichero de configuración `appssms.config`.

Para esta configuración es necesario tener en cuenta:

- La configuracón se ha realizado para la SMSC Virtual utilizada. Ésta es el módem interno de un teléfono Alcatel OT535.
- Para la configuración de los servicios SMS se ha utilizado la dirección http://localhost:8084/AppsSms, por tanto, si se utiliza la configuración por defecto de Tomcat tendrá que sustituirse el puerto 8084, por el puerto 8080.

10.5 CÓDIGO

Para el desarrollo de la aplicación, han sido de gran ayuda documental el conjunto de ejemplos de la asignatura de Integración de Sistemas año 2003-2004, del Departamento de Tecnologías de la Información y las Comunicaciones (TIC), de la Universidad de A Coruña, impartida por Fernando Bellas Permuy y otros profesores.

Estos ejemplo se tomaron como base inicial, y de ellos se adoptaron y adaptaran algunas de las utilizadades y estructuras proporcionadas. La tecnología J2EE, gracias a su estructura de capas y modularidad, potencia esta reusabilidad.

10.5.1 Creación de la estructura de datos: SQL

Incluimos el código de creación de base de datos y tablas. Con este fichero se crea un usuario con los permisos de insertar, actualizar o buscar datos en la base de datos creada. Nuestra aplicación ejemplo se conectará a la base de datos utilizando este usuario. Esta información se encuentra el fichero *application.sql*.

application.sql

```
#Creating a new database if it doesn't exist.
CREATE DATABASE apps;

#Changing database in use.
USE apps;

#Deleting existing tables to evoid errors in new table creation.
DROP TABLE holidays_request_registry;
DROP TABLE holidays_service_registry;
DROP TABLE user_profile_registry;

#Creating tables.
#In MySQL ENGINE = InnoDB for transactions.

CREATE TABLE user_profile_registry (
loginName VARCHAR(9) NOT NULL COMMENT "user identifier",
enPassword VARCHAR(11) NOT NULL COMMENT "encrypted password",
firstName VARCHAR(30) COMMENT "user firstname",
surname VARCHAR(40) COMMENT "user surname",
document VARCHAR(9) COMMENT "user identity documment number",
email VARCHAR(40) COMMENT "user e-mail",
address VARCHAR(60) COMMENT "user address",
details VARCHAR(120) COMMENT "details about user",
phone VARCHAR(12) COMMENT "user telephone",
isAdministrator BOOLEAN default 0 COMMENT "user administrator role"
) ENGINE = InnoDB;

CREATE TABLE holidays_service_registry (
loginName VARCHAR(10) NOT NULL COMMENT "user identifier",
period YEAR(4) NOT NULL COMMENT "period of holidays service for the
      user",
days INTEGER NOT NULL default 0 COMMENT "days of holidays"
) ENGINE = InnoDB;

CREATE TABLE holidays_request_registry (
autoId INTEGER AUTO_INCREMENT PRIMARY KEY COMMENT "auto generated
value for the
        request",
loginName VARCHAR(10) NOT NULL COMMENT "user identifier",
period YEAR(4) NOT NULL COMMENT "period of holidays service for the
      user",
stateDate TIMESTAMP NOT NULL default 0 COMMENT "requesting or
      processing date",
startDate TIMESTAMP NOT NULL COMMENT "first day included resquested",
finalDate TIMESTAMP NOT NULL COMMENT "last day included requested",
state TINYINT NOT NULL default 0 COMMENT "processing state: 0-not
      processed, 1-accept, others-rejected or cancelled",
details VARCHAR(120) COMMENT "holidays request details"
) ENGINE = InnoDB;

#Inserting first data.
INSERT INTO user_profile_registry (loginName, enPassword, firstName,
      isAdministrator) VALUES ("adm", "YKC9TcSAQhw", "administrador",
1);

#Adding keys.
#This is done after inserting data because keys make the database
#slower in inserting or updating data.
#Primary keys are use for find and identify data.
#Foreing keys are use for database reference integrity.

ALTER TABLE user_profile_registry ADD PRIMARY KEY (loginName);

ALTER TABLE holidays_service_registry ADD PRIMARY KEY
(loginName,period),

ADD CONSTRAINT fk_holidays_service_loginName FOREIGN KEY fk_loginName
(loginName)
REFERENCES user_profile_registry (loginName) ON DELETE RESTRICT ON
UPDATE RESTRICT;
```

```
ALTER TABLE holidays_request_registry ADD CONSTRAINT
fk_holidays_request_period
FOREIGN KEY fk_period (loginName, period)
REFERENCES holidays_service_registry (loginName, period)
ON DELETE RESTRICT ON UPDATE RESTRICT;

GRANT INSERT, SELECT, UPDATE ON apps.* TO userapps IDENTIFIED BY
'userapps';
```

10.5.2 Estructura de directiorios

```
Apps/
|
|- index.jsp (+)
|--- CSS/ (+)
|--- HTML/ (+)
|--- META-INF/
|           |- context.xml (+)
|--- WEB-INF/
            |- web.xml (+)
            |--- StdTagLibs
            |           |- c.tld
            |           |- fmt.tld
            |--- Struts
            |           |- struts-config.xml (+)
            |           |- struts-html.tld
            |           |- struts-tiles.tld
            |           |- tiles-defs.xml (+)
            |           |- validation.xml (+)
            |           |- validator-rules.xml (+)
            |--- classes
            |           |- Messages.properties
            |           |--- application (+, classes)
            |--- lib
            |     |- antlr.jar
            |     |- commons-beanutils.jar
            |     |- commons-collections.jar
            |     |- commons-dbcp-1.2.1.jar
            |     |- commons-digister.jar
            |     |- commons-fileupload.jar
            |     |- commons-logging.jar
            |     |- commons-validator.jar
            |     |- jakarta-oro.jar
            |     |- jstl.jar
            |     |- mysql-connector-java-3.1.6-bin.jar (driver)
            |     |- standard.jar
            |     |- struts.jar
            |--- scripts
                        |- application.sql (+)
```

Páginas JSP

```
/
|- index.jsp
|--- CSS/
|     |- Default.css
|     |- Print.css
|--- HTML/
      |- Login.jsp
      |- MainPage.jsp
      |- RegisterUser.jsp
      |- UserProfileDetails.jsp
      |- UserProfileAccess.jsp
      |- ViewUserProfile.jsp
      |- UpdateUserProfile.jsp
      |- ChangePassword.jsp
      |- GivePassword.jsp
      |- LoginAsUser.jsp
      |- FindUser.jsp
      |- FindUserResult.jsp
      |- RegisterHolidaysService.jsp
      |- FindHolidaysServiceResult.jsp
      |- UpdateHolidaysService.jsp
      |- ChangePhone.jsp
      |- GivePhone.jsp
      |- FindHolidaysService.jsp
      |- RegisterHolidaysRequest.jsp
      |- HolidaysRequest.jsp
      |- ConfirmRegisterHolidaysRequest.jsp
      |- PreviewHolidaysRequest.jsp
      |- ViewHolidaysRequest.jsp
      |- FindHolidaysRequest.jsp
      |- FindHolidaysRequestResult.jsp
      |- ViewHolidaysProfile.jsp
      |- FindHolidaysProfile.jsp
      |- MainUserProfilePage.jsp
      |- MainHolidaysServicePage.jsp
      |- MainUtilitiesPage.jsp
      |- HolidaysRequestReason.jsp
      |--- /layout
      |     |- DefaultTemplate.jsp
      |     |- PrintTemplate.jsp
      |     |- PageLinksTemplate.jsp
      |     |- HLinksTemplate.jsp
      |     |- VLinksTemplate.jsp
      |     |- BasicTemplate.jsp
      |--- /common
            |- user_working.jsp
            |- FindResultPageLink.jsp
            |-ViewHolidaysRequestPageLink.jsp
```

Fuentes

application.http

```
|--- controller
            |--- actions
            |           |--- holidaysservice
            |           |           |- AdministratorFindHolidaysRequest
            |           |           |                  Action
            |           |           |- CancelHolidaysRequestAction
            |           |           |- EditHolidaysRequestAction
            |           |           |- EditHolidaysServiceAction
            |           |           |- EditUserHolidaysRequestAction
            |           |           |- FindHolidaysProfileAction
            |           |           |- FindHolidaysProfilesAction
            |           |           |- FindHolidaysRequestAction
            |           |           |- FindHolidaysServicesAction
            |           |           |- FindPeriodHolidaysProfileAction
            |           |           |- FindPeriodHolidaysServiceAction
            |           |           |- FindUserHolidaysServiceAction
            |           |           |- MapUpdatesHolidaysRequestAction
            |           |           |- RegisterHolidaysRequestAction
            |           |           |- RegisterHolidaysServiceAction
            |           |           |- UpdateHolidaysServiceAction
            |           |--- userprofile
            |                       |- AdministratorUpdateUserProfile
            |                             Action
            |                       |- ChangePasswordAction
            |                       |- ChangePhoneAction
            |                       |- EditPhoneAction
            |                       |- EditUserProfileAction
            |                       |- FindUserAction
            |                       |- GivePasswordAction
            |                       |- GivePhoneAction
            |                       |- LoginAction
            |                       |- LoginAsUserAction
            |                       |- LogoutAction
            |                       |- RegisterUserAction
            |                       |- UpdateUserProfileAction
            |                       |- ViewUserProfileDetailsAction
            |--- filtering
            |           |- AuthenticationPreProcessingFilter
            |           |- AdministratorPreProcessingFilter
            |--- session
                        |- SessionManager
```

```
application.model
|--- userprofileservice
|           |--- facade
|           |           |--- delegate
|           |           |           |- UserProfileServiceFacade
|           |           |           |- UserProfileServiceFacadeFactory
|           |           |           |- PlainUserProfileServiceFacade
|           |           |--- actions
|           |           |           |- LoginAction
|           |           |           |- RegisterUserAction
|           |           |           |- UpdateUserProfileDetailsAction
|           |           |           |- UpdateUserProfileAccessAction
|           |           |           |- ChangePasswordAction
|           |           |           |- GivePasswordAction
|           |           |           |- ChangePhoneAction
|           |           |           |- GivePhoneAction
|           |           |           |- FindUserByAction
|           |           |           |- FindUserInByAction
|           |           |           |- FindUserProfileAction
|           |           |--- vo
|           |                 |- UserFacadeVO
|           |--- operations
|           |           |- Jcrypt
|           |           |- PasswordEncrypter
|           |--- dao
|           |     |- SQLUserProfileDAO
|           |     |- SQLUserProfileDAOFactory
|           |     |- StandardSQLUserProfileDAO
|           |--- vo
|                 |- UserProfileVO
|                 |- UserProfileDetailsVO
|                 |- UserProfileAccessVO
|--- holidaysrequestservice
|           |--- facade
|           |     |--- delegate
|           |     |           |- HolidaysRequestServiceFacade
|           |     |           |- HolidaysRequestServiceFacadeFactory
|           |     |           |- PlainHolidaysRequestServiceFacade
|           |     |--- actions
|           |     |           |--- holidaysservice
|           |     |           |     |- FindHolidaysServiceAction
|           |     |           |     |- FindHolidaysServicesAction
|           |     |           |     |- FindPeriodHolidaysServicesAction
|           |     |           |     |- FindUserHolidaysServicesAction
|           |     |           |     |- RegisterHolidaysServiceAction
|           |     |           |     |- UpdateHolidaysServiceAction
|           |     |           |--- holidaysrequest
|           |     |                 |- FindHolidaysProfilesAction
|           |     |                 |- FindAdvanceAction
|           |     |                 |- FindIdHolidaysRequestAction
|           |     |                 |- FindUserHolidaysRequestByAction
|           |     |                 |- FindUserIdHolidaysRequestAction
|           |     |                 |- FindUserPeriodStateHolidays
|           |     |                 |                RequestByAction
|           |     |                 |- RegisterHolidaysRequestAction
|           |     |                 |- UpdateHolidaysRequestAction
|           |     |                 |- CancelHolidaysRequestAction
|           |     |--- vo
|           |           |- HolidaysProfileVO
|           |           |- HolidaysVO
|           |--- operations
|           |           |- Jworkingdays
|           |--- dao
|           |     |--- holidaysservice
|           |     |           |- SQLHolidaysServiceDAO
|           |     |           |- SQLHolidaysServiceDAOFactory
|           |     |           |- StandardSQLHolidaysServiceDAO
|           |     |--- holidaysrequest
|           |                 |- SQLHolidaysRequestDAO
|           |                 |- SQLHolidaysRequestDAOFactory
|           |                 |- StandardSQLHolidyasRequestDAO
|           |--- vo
|                 |--- holidaysservice
|                 |           |- HolidaysServiceVO
|                 |--- holidaysrequest
|                             |- HolidaysRequestVO
```

```
|--- util
      |- GlobalNames
      |- UserProfileTable
      |- HolidaysRequestTable
```

application.util

```
|--- exceptions
|           |- AdministratorRequiredException
|           |- DuplicateInstanceException
|           |- InstanceNotFoundException
|           |- IncorrectPasswordException
|           |- ModelException
|           |- InternalErrorException
|--- controller
|           |--- filter
|           |           |- DefaultFilter
|           |--- struts
|                       |- AdministratorDefaultAction
|                       |- DefaultAction
|--- view
|     |--- tiles
|           |- PlusTargetMenuItem
|--- jndi
|     |- ConfigurationParametersManager
|     |- MissingConfigurationParameterException
|--- sql
      |- GeneralOperations
      |- DatasourceLocator
      |- PlainActionProcessor
      |- PlainAction
      |- NonTransactionalPlainAction
      |- TransactionalPlainAction
```

10.5.3 Diseño de la lógica de negocio: Java

A continuación se incluye el código que implementa la lógica de negocio.

10.5.3.1 Objetos de datos

Los objetos de datos implementan el patrón de diseño Transfer Object, también conocido como Value Object (VO).

10.5.3.1.a Perfil de usuario

10.5.3.1.a.1 application.model.userprofileservice.vo

UserProfileVO

```
package application.model.userprofileservice.vo;

import java.io.Serializable;

import application.model.userprofileservice.vo.UserProfileAccessVO;
import application.model.userprofileservice.vo.UserProfileDetailsVO;

/**
 * This class is used for interacting with user profile datas.
 * <BR>
 * LoginName, encrypted password, user details, phone and access
 * permission details are user profile datas. Getter and setter
 * methods are used for accessing user profile properties.
 * <BR>
 * {@link UserProfileAccessVO} is used for user details properties.
 * <BR>
 * {@link UserProfileDetailsVO} is used for access permission
 * properties.
 */
public class UserProfileVO implements Serializable {

    /**
     * Hold values of properties.
     */
```

```
private String loginName;
private String enPassword;
private UserProfileDetailsVO userProfileDetailsVO;
private String phone;
private UserProfileAccessVO userProfileAccessVO;

/**
 * Construct VO with the given user profile details.
 * @param loginName user identifier.
 * @param enPassword encrypted password
 * @param userProfileDetailsVO more user details
 * @param phone
 * @param userProfileAccessVO
 */
public UserProfileVO(String loginName, String enPassword,
        UserProfileDetailsVO userProfileDetailsVO, String phone,
        UserProfileAccessVO userProfileAccessVO) {

    this.loginName = loginName;
    this.enPassword = enPassword;
    this.userProfileDetailsVO = userProfileDetailsVO;
    this.phone = phone;
    this.userProfileAccessVO = userProfileAccessVO;
}

/**
 * Returns user identifier.
 * @return loginName.
 */
public String getLoginName()  {
    return loginName;
}

/**
 * Returns encrypted password.
 * @return
 */
public String getEnPassword() {
    return enPassword;
}

/**
 * Set new encrypted password.
 * @param enPassword
 */
public void setEnPassword(String enPassword) {
    this.enPassword = enPassword;
}

/**
 * Returns user details.
 * @return user details in {@link UserProfileDetailsVO}.
 */
public UserProfileDetailsVO getUserProfileDetailsVO() {
    return userProfileDetailsVO;
}
```

```
/**
 * Set new user details.
 * @param userProfileDetailsVO user details as {@link
 * UserProfileDetailsVO}
 */
public void setUserProfileDetailsVO(UserProfileDetailsVO
  userProfileDetailsVO) {
    this.userProfileDetailsVO = userProfileDetailsVO;
}

/**
 * Get phone value.
 * @return <CODE>phone</CODE> property value as
 * <CODE>String</CODE>.
 */
public String getPhone() {
    return phone;
}

/**
 * Set a new phone number.
 * @param phone
 */
public void setPhone(String phone) {
    this.phone = phone;
}

/**
 * Returns access permission details.
 * @return access permission in {@link UserProfileAccessVO}
 */
public UserProfileAccessVO getUserProfileAccessVO() {
    return userProfileAccessVO;
}

/**
 * Set new access permission details.
 * @param userProfileAccessVO access permission details in {@link
 * UserProfileAccessVO}
 */
public void setUserProfileAccessVO(UserProfileAccessVO
  userProfileAccessVO){
    this.userProfileAccessVO = userProfileAccessVO;
}

/**
 * Returns a <CODE>String</CODE> object representing this VO's
 * value. Useful for debugging.
 * @return a string representation of this object.
 */
public String toString(){

    return new String("loginName = " + loginName + " | enPassword
  = " + enPassword + " | " + userProfileDetailsVO + " | phone
  = " + phone + " | " + userProfileAccessVO.toString());
```

```
    }

}
```

UserProfileAccessVO

```
package application.model.userprofileservice.vo;


import java.io.Serializable;

/**
 * This class is used for interacting with user access permissions
 * data.
 * <BR>
 * A <CODE>TRUE</CODE> value for the <CODE>isAdministrator</CODE>
 * property, represents that an user has administrator access
 * permission. Getter and setter methods are used for accessing
 * <CODE>isAdministrator</CODE> property.
 */
public class UserProfileAccessVO implements Serializable {

    /**
     * Holds the value of administrator access permission.
     */
    private Boolean isAdministrator;

    /**
     * Constructs VO with the given access permissions.
     * @param isAdministrator <CODE>TRUE</CODE> for an administrator
     * access permision.
     */
    public UserProfileAccessVO(Boolean isAdministrator) {
        this.isAdministrator = isAdministrator;
    }

     /**
     * Returns the <CODE>Boolean</CODE> value of the administrator
     * accesspermission.
     * @return <CODE>TRUE</CODE> represents an administrator access
     * permision.
     */
    public Boolean getIsAdministrator() {
        return isAdministrator;
    }

    /**
     * Returns a <CODE>String</CODE> object representing this VO's
     * value. Useful for debugging.
     * @return a string representation of this object.
     */
    public String toString(){
        return new String("isAdministrator = " + isAdministrator);
    }
```

UserProfileDetailsVO

```
package application.model.userprofileservice.vo;

import java.io.Serializable;

/**
 * This class is used for interacting with user details datas.
 * <BR>
 * Firstname, surname, identity document number, e-mail, address and
 * other details are user details datas. Getter and setter methods are
 * used for accessing user detail properties.
 */
public class UserProfileDetailsVO implements Serializable {

    /**
     * Hold values of user details properties.
     */
    private String firstName;
    private String surname;
    private String document;
    private String email;
    private String address;
    private String details;

    /**
     * Constructs VO with the given user details.
     * @param firstName Represents user firstname.
     * @param surname Represents user surname.
     * @param document Represents identity document number.
     * @param email Represents user email address.
     * @param address Represents user address.
     * @param details Represents other user details.
     */
    public UserProfileDetailsVO(String firstName, String surname,
            String document, String email, String address, String
      details) {

        this.firstName = firstName;
        this.surname = surname;
        this.document = document;
        this.email = email;
        this.address = address;
        this.details = details;
    }

    /**
     * Returns the firstName property value.
     * @return the <CODE>String</CODE> value of the
     * <CODE>firstName</CODE> property.
     */
    public String getFirstName() {
        return firstName;
    }

    /**
     * Returns the surname property value.
     * @return the <CODE>String</CODE> value of the
     * <CODE>surname</CODE> property.
     */
    public String getSurname() {
        return surname;
    }

    /**
     * Returns the <CODE>document</CODE> property value.
     * @return the identity document number as <CODE>String</CODE>.
     */
    public String getDocument() {
        return document;
    }

    /**
     * Returns the email address value.
     * @return the <CODE>String</CODE> value of <CODE>email</CODE>
     * property.
     */
    public String getEmail() {
        return email;
    }

    /**
     * Returns the <CODE>address</CODE> property value as
     * <CODE>String</CODE>.
     * @return the address value.
     */
    public String getAddress() {
        return address;
    }

    /**
     * Returns <CODE>details</CODE> property value.
     * @return
     */
    public String getDetails() {
        return details;
    }

    /**
     * Returns a <CODE>String</CODE> object representing this VO's
     * value. Useful for debugging.
     * @return a string representation of this object.
     */
    public String toString(){
        return new String("firstName = " + firstName + " | surname = "
            + surname + " | document = " + document + " | email = " +
      email + " | address = " + address + " | details = " +
      details);
    }

}
```

10.5.3.1.b Alta en servicio de solicitud de vacaciones

10.5.3.1.b.1 application.model.holidaysrequestservice.vo.holidaysservice

HolidaysServiceVO

```
package application.model.holidaysrequestservice.vo.holidaysservice;

import java.io.Serializable;

import java.util.Calendar;

/**
 * This class is used for interacting with holidays service registry
 * data.
 * <BR>
 * LoginName, period and days represents an user holidays service
 * registry profile. An user have different holidays service registry
 * profiles for different periods.
 */
public class HolidaysServiceVO implements Serializable {

    /**
     * Hold values of properties.
     */
    private String loginName;
    private Calendar period;
    private Integer days;

    /**
     * Construct VO with the given holidays service profile.
     * @param loginName
     * @param period
     * @param days
     */
    public HolidaysServiceVO(String loginName, Calendar period,
      Integer days) {

        this.loginName = loginName;
        this.period = period;
        this.days = days;
    }

    /**
     * Returns user identifier.
     * @return loginName.
     */
    public String getLoginName() {
        return loginName;
    }

    /**
     * Returns holidays service period as Calendar. Calendar type for
     * period property is used for easier use in operations with
     * dates. In fact, period is represented in Calendar.YEAR
     * property.
     * @return period
     */
    public Calendar getPeriod() {
        return period;
    }

    /**
     * Returns days of holidays in a holidays service profile.
     * @return days.
     */
    public Integer getDays() {
        return days;
    }

    /**
     * Sets new number of days of holidays in a holidays service
     * profile.
     * @param days
     */
    public void setDays(Integer days) {
        this.days = days;
    }

    /**
     * Returns a <CODE>String</CODE> object representing this VO's
     * value. Useful for debugging.
     * @return a string representation of this object.
     */
    public String toString(){

        return new String("loginName = " + loginName + " | period = "
      + period + " | days = " + days);
    }

}
```

10.5.3.1.c Solicitud de vacaciones

10.5.3.1.c.1 application.model.holidaysrequestservice.vo.holidaysrequest

HolidaysRequestVO

```java
package application.model.holidaysrequestservice.vo.holidaysrequest;

import java.io.Serializable;

import java.util.Calendar;

import
application.model.holidaysrequestservice.operations.Jworkingdays;

/**
 * This class is used for interacting with holidays request related
 * data.
 * <BR>
 * AutoId, LoginName, period, stateDate, startDate, finalDate, state
 * and details represents a holidays request.
 * <BR>
 * AutoId is used as unique identifier of the request and it is
 * automatically gereneted for a new request.
 * <BR>
 * State represents: "Not trammited yet" when 0, "Accepted" when 1,
 * "Rejected" when 2, and "Cancelled" when 3. Others values are not
 * considered.
 */
public class HolidaysRequestVO implements Serializable {

    /**
     * Hold values of properties.
     */
    private Integer autoId;
    private String loginName;
    private Calendar period;
    private Calendar stateDate;
    private Calendar startDate;
    private Calendar finalDate;
    private Short state;
    private String details;

    /**
     * Construct VO with the given holidays request.
     * @param autoId
     * @param loginName
     * @param period
     * @param stateDate
     * @param startDate
     * @param finalDate
     * @param state
     * @param details
     */
```

```
public HolidaysRequestVO(Integer autoId, String loginName,
  Calendar period, Calendar stateDate, Calendar startDate,
  Calendar finalDate, Short state, String details) {

    this.autoId = autoId;
    this.loginName = loginName;
    this.period = period;
    this.stateDate = stateDate;
    this.startDate = startDate;
    this.finalDate = finalDate;
    this.state = state;
    this.details = details;
}

/**
 * Returns holidays request identifier.
 * @return autoId.
 */
public Integer getAutoId() {
    return autoId;
}

/**
 * Returns user identifier.
 * @return loginName.
 */
public String getLoginName() {
    return loginName;
}

/**
 * Returns holidays service period of a holidays request as
 * Calendar. Calendar type for period property is used for easier
 * use in operations with dates. In fact, period is represented in
 * Calendar.YEAR property.
 * @return period
 */
public Calendar getPeriod() {
    return period;
}

/**
 * Returns last modifaction date of the holidays request state. It
 * represents last updating access or inserting access.
 * @return stateDate
 */
public Calendar getStateDate() {
    return stateDate;
}

/**
 * Returns first day include in the holidays request.
 * @return first day in holidays request.
 */
public Calendar getStartDate() {
    return startDate;
```

```
}

/**
 * Returns last day include in the holidays request.
 * @return last day in holidays request.
 */
public Calendar getFinalDate() {
    return finalDate;
}

/**
 * Returns holidays request state.
 * State represents: "Not trammited yet" when 0, "Accepted" when
 * 1, "Rejected" when 2, and "Cancelled" when 3. Others values are
 * not considered.
 * @return state
 */
public Short getState() {
    return state;
}

/**
 * Sets holidays request state.
 * State represents: "Not trammited yet" when 0, "Accepted" when
 * 1, "Rejected" when 2, and "Cancelled" when 3. Others values are
 * not considered.
 * @param state
 */
public void setState(Short state) {
    this.state = state;
}

/**
 * Returns holidays request related details.
 * @return details
 */
public String getDetails() {
    return details;
}

public Integer getWorkingDays() {

    return Jworkingdays.workingdays(startDate, finalDate);
}

/**
 * Returns a <CODE>String</CODE> object representing this VO's
 * value. Useful for debugging.
 * @return a string representation of this object.
 */
public String toString(){

    return new String("autoId = " + autoId + " | loginName = " +
  loginName + " | period = " + period + " | stateDate = " +
  stateDate + " | startDate = " + startDate + " | finalDate =
      " + finalDate + " | state = " + state + " | details = " +
      details + " | workingdays = " + getWorkingDays());
    }

}
```

10.5.3.2 Capa de acceso a datos (DAO)

10.5.3.2.a Perfil de usuario

10.5.3.2.a.1 application.model.userprofileservice.dao

SQLUserProfileDAO

```
package application.model.userprofileservice.dao;

import java.io.Serializable;

import java.util.Collection;

/* Uses jdbc. */
import java.sql.Connection;

import application.model.userprofileservice.vo.UserProfileVO;

import application.util.exceptions.DuplicateInstanceException;
import application.util.exceptions.InstanceNotFoundException;
import application.util.exceptions.InternalErrorException;

/**
 * This interface defines basic methods for interacting with user
 * profile datas in data access layer.
 * <BR>
 * Upper layers use this interface, not its implementation. With
 * <CODE>SQLUserProfileDAOFactory<CODE>, upper layers can determine
 * which implementation must be used. This makes code independent from
 * each data access implementation and datasource.
 */
public interface SQLUserProfileDAO extends Serializable {

    /**
     * Creates a new row with and user profile.
     * @param connection
     * @param UserProfileVO
     * @throws application.util.exceptions.DuplicateInstanceException
     * @throws application.util.exceptions.InternalErrorException
     */
    public void create(Connection connection, UserProfileVO
      UserProfileVO) throws DuplicateInstanceException,
      InternalErrorException;

    /**
     * Checks if a user profile already exists. Returns
     * <code>true</code> if loginName already exists and
     * <code>false</code> if not.
     * @param connection
     * @param loginName
     * @throws application.util.exceptions.InternalErrorException
     * @return if a user profile exists.
     */
    public Boolean exists(Connection connection, String loginName)
      throws InternalErrorException;

    /**
     * Tries to update an already existing user profile.
     * @param connection
     * @param userProfileVO
     * @throws application.util.exceptions.InstanceNotFoundException
     * @throws application.util.exceptions.InternalErrorException
     */
    public void update(Connection connection, UserProfileVO
      userProfileVO) throws InstanceNotFoundException,
      InternalErrorException;

    /**
     * Finds an user profile with a loginName given.
     * @param connection
     * @param loginName
     * @throws application.util.exceptions.InstanceNotFoundException
     * @throws application.util.exceptions.InternalErrorException
     * @return user profile
     */
    public UserProfileVO find(Connection connection, String loginName)
      throws InstanceNotFoundException, InternalErrorException;

    /**
     * Finds all users ordered by propertyName.
     * <BR>
     * Returns a collection of
     * <code>count</code> user profile. If count is <code>0</code>
     * returns all user profiles.
     * <BR>
     * StartIndex represents the first profile of the collection.
     * @param connection
     * @param propertyByName
     * @param startIndex
     * @param count
     * @throws application.util.exceptions.InternalErrorException
     * @return a collection of user profiles
```

```
     */
    public Collection findBy(Connection connection,String
      propertyByName, Integer startIndex, Integer count) throws
      InternalErrorException;

    /**
     * Finds user profiles whith a pattern given in the propertyInName
     * column ordered by propertyByName.
     * <BR>
     * Returns a collection of
     * <code>count</code> user profile. If count is <code>0</code>
     * returns all user profiles those satisfied the search.
     * <BR>
     * StartIndex represents the first profile of the collection.
     * @param connection
     * @param propertyInName column in which a pattern must be found
     * @param propertyInValue pattern to find
     * @param propertyByName shows results order by this property
     * @param startIndex
     * @param count
     * @throws application.util.exceptions.InternalErrorException
     * @return
     */
    public Collection findInBy(Connection connection,
            String propertyInName, String propertyInValue,
            String propertyByName, Integer startIndex, Integer count)
      throws InternalErrorException;

}
```

SQLUserProfileDAOFactory

```
package application.model.userprofileservice.dao;

/* Uses JNDI. */
import application.util.jndi.ConfigurationParametersManager;

import application.util.exceptions.InternalErrorException;

/**
 * This class creates an intance of the implementation of
 * <code>SQLUserProfileDAO</code>.
 * <BR>
 * Upper layers use this class to determine which implementation must
 * be used. This makes code independent from each data access
 * implementation and datasource.
 *<BR>
 *<code>SQLUserProfileDAOFactory/daoClassName</code> parameter has to
 * be defined using JNDI. This parameter can be defined in
 * <code>web.xml</code> with <code><env-entry></code> tags.
 *<BR>
 *This code provide an implementation of
 * <code>SQLUserProfileDAO</code> for MySQL in
 * <code>StandarSQLUserProfileDAO</code>.
 */
public final class SQLUserProfileDAOFactory {

    /* This parameter has to be define in a JNDI context. */
    private final static String DAO_CLASS_NAME_PARAMETER =
            "SQLUserProfileDAOFactory/daoClassName";

    private final static Class daoClass = getDAOClass();

    private SQLUserProfileDAOFactory() {}

    /* Gets the name of the implementation class of DAO. */
     private static Class getDAOClass() {

        Class theClass = null;
        try {
            String daoClassName =
            ConfigurationParametersManager.getParameter(
                    DAO_CLASS_NAME_PARAMETER);
            theClass = Class.forName(daoClassName);
        } catch (Exception e) {
            e.printStackTrace();
        }

        return theClass;
    }

    /**
     * Returns an instance of the implementation class of DAO.
     * @throws application.util.exceptions.InternalErrorException
     * @return implementation of data access object
     */
    public static SQLUserProfileDAO getDAO() throws
      InternalErrorException {
        try {
            return (SQLUserProfileDAO) daoClass.newInstance();
        } catch (Exception e) {
            throw new InternalErrorException(e);
        }
    }

}
```

StandardSQLUserProfileDAO

```
package application.model.userprofileservice.dao;

import java.util.ArrayList;
import java.util.Collection;

/* JDBC.*/
import java.sql.Connection;
import java.sql.SQLException;
import java.sql.PreparedStatement;
import java.sql.ResultSet;

import application.model.userprofileservice.dao.SQLUserProfileDAO;

import application.model.userprofileservice.vo.UserProfileVO;
import application.model.userprofileservice.vo.UserProfileDetailsVO;
import application.model.userprofileservice.vo.UserProfileAccessVO;

import application.util.exceptions.DuplicateInstanceException;
import application.util.exceptions.InstanceNotFoundException;
import application.util.exceptions.InternalErrorException;

import application.util.sql.GeneralOperations;

import application.model.util.UserProfileTable;

/**
 * This class provides an implementation of
 * <code>SQLUserProfileDAO</code> for interacting with user profile
 * datas in data access layer. This is an implementation for a
 * relational database, MySQL, using JDBC driver.
 */
public class StandardSQLUserProfileDAO implements SQLUserProfileDAO {

     /**
     * Creates a new row with and user profile. If a user with the
     * same loginName exists throws a
     * <CODE>DuplicateInstanceException</CODE>.
     * @param connection
     * @param UserProfileVO
     * @throws application.util.exceptions.DuplicateInstanceException
     * @throws application.util.exceptions.InternalErrorException
     */
    public void create(Connection connection, UserProfileVO
      userProfileVO) throws DuplicateInstanceException,
      InternalErrorException {

        /* Check if loginName already exists. */
        if (exists(connection, userProfileVO.getLoginName())) {
            throw new
      DuplicateInstanceException(userProfileVO.getLoginName());
        }

        PreparedStatement preparedStatement = null;

        try {

            /* Creates statement. */
            String queryString = "INSERT INTO user_profile_registry" +
                  " (loginName, enPassword, firstName, surname,
            document, " + "email, address, details, phone,
      isAdministrator) " + "VALUES (?, ?, ?, ?, ?, ?, ?, ?,
      ?, ?)";
```

```
        preparedStatement =
        connection.prepareStatement(queryString);

        /* Fills statement. */
        int i = 1;
        preparedStatement.setString(i++,
        userProfileVO.getLoginName());
        preparedStatement.setString(i++,
        userProfileVO.getEnPassword());
        preparedStatement.setString(i++,

  userProfileVO.getUserProfileDetailsVO().getFirstName());
        preparedStatement.setString(i++,
           userProfileVO.getUserProfileDetailsVO().getSurname());
        preparedStatement.setString(i++,
           userProfileVO.getUserProfileDetailsVO().getDocument());
        preparedStatement.setString(i++,
           userProfileVO.getUserProfileDetailsVO().getEmail());
        preparedStatement.setString(i++,
           userProfileVO.getUserProfileDetailsVO().getAddress());
        preparedStatement.setString(i++,
           userProfileVO.getUserProfileDetailsVO().getDetails());
        preparedStatement.setString(i++,
     userProfileVO.getPhone());
        preparedStatement.setBoolean(i++,
     userProfileVO.getUserProfileAccessVO().getIsAdministrator());

        /* Executes query. */
        int insertedRows = preparedStatement.executeUpdate();

        /* Checks errors. */
        if (insertedRows == 0) {
            throw new SQLException("Can not add row to table " +
                    "'user_profile_registry'");
        }

        if (insertedRows > 1) {
            throw new SQLException("Duplicate row for key = '" +
                    userProfileVO.getLoginName() +
                    "' in table 'user_profile_registry'");
        }

    } catch (SQLException e) {
        throw new InternalErrorException(e);
    } finally {
        GeneralOperations.closeStatement(preparedStatement);
    }

}

/**
 * Checks if a user profile already exists. Returns
 * <code>true</code> if loginName already exists and
 * <code>false</code> if not.
 * @param connection
 * @param loginName
 * @throws application.util.exceptions.InternalErrorException
 * @return if a user profile exists.
 */
public Boolean exists(Connection connection, String loginName)
  throws InternalErrorException {

    PreparedStatement preparedStatement = null;
    ResultSet resultSet = null;

    try {

        /* Creates statement. */
        String queryString = "SELECT loginName FROM
        user_profile_registry" + " WHERE loginName = ?";

        preparedStatement =
        connection.prepareStatement(queryString);

        /* Fills statement. */
        int i = 1;
        preparedStatement.setString(i++, loginName);

        /* Executes query. */
        resultSet = preparedStatement.executeQuery();

        return resultSet.next();

    } catch (SQLException e) {
        throw new InternalErrorException(e);
    } finally {
        GeneralOperations.closeResultSet(resultSet);
        GeneralOperations.closeStatement(preparedStatement);
    }

}

/**
 * Tries to update an already existing user profile. Throws
 * <code>InstanceNotFoundException</code> when user profile for
 * updating doesn't exist.
 * @param connection
 * @param userProfileVO
 * @throws application.util.exceptions.InstanceNotFoundException
 * @throws application.util.exceptions.InternalErrorException
 */
public void update(Connection connection, UserProfileVO
  userProfileVO) throws InstanceNotFoundException,
  InternalErrorException {

    PreparedStatement preparedStatement = null;

    try {

        /* Creates statement. */
        String queryString = "UPDATE user_profile_registry" +
                " SET enPassword = ?, firstName = ?, " +
                "surname = ?, document = ?, email = ?, address =
          ?, " + "details = ? , phone = ?, isAdministrator =
    ? " + " WHERE loginName = ?";

        preparedStatement =
        connection.prepareStatement(queryString);

        /* Fills statement. */
        int i = 1;
        preparedStatement.setString(i++,
        userProfileVO.getEnPassword());
        preparedStatement.setString(i++,
          userProfileVO.getUserProfileDetailsVO().getFirstName());
        preparedStatement.setString(i++,
          userProfileVO.getUserProfileDetailsVO().getSurname());
        preparedStatement.setString(i++,
          userProfileVO.getUserProfileDetailsVO().getDocument());
        preparedStatement.setString(i++,
          userProfileVO.getUserProfileDetailsVO().getEmail());
        preparedStatement.setString(i++,
          userProfileVO.getUserProfileDetailsVO().getAddress());
        preparedStatement.setString(i++,
          userProfileVO.getUserProfileDetailsVO().getDetails());
        preparedStatement.setString(i++,
    userProfileVO.getPhone());
        preparedStatement.setBoolean(i++,
     userProfileVO.getUserProfileAccessVO().getIsAdministrator());
        preparedStatement.setString(i++,
    userProfileVO.getLoginName());

        /* Executes query. */
        int updatedRows = preparedStatement.executeUpdate();

        /* Checks errors. */
        if (updatedRows == 0) {
            throw new InstanceNotFoundException(
                    userProfileVO.getLoginName());
        }

        if (updatedRows > 1) {
            throw new SQLException("Duplicate row for key = '" +
                    userProfileVO.getLoginName() +
                    "' in table 'user_profile_registry'");
        }

    } catch (SQLException e) {
        throw new InternalErrorException(e);
    } finally {
        GeneralOperations.closeStatement(preparedStatement);
    }

}

/**
 * Finds an user profile with a loginName given.
 * @param connection
```

```
  * @param loginName
  * @throws application.util.exceptions.InstanceNotFoundException
  * @throws application.util.exceptions.InternalErrorException
  * @return user profile
  */
public UserProfileVO find(Connection connection, String loginName)
  throws InstanceNotFoundException, InternalErrorException {

    PreparedStatement preparedStatement = null;
    ResultSet resultSet = null;

    try {

        /* Creates statement. */
        String queryString = "SELECT enPassword, firstName,
        surname, document, " + "email, address, details,
  phone, isAdministrator " + "FROM
  user_profile_registry WHERE loginName = ?";

        preparedStatement =
        connection.prepareStatement(queryString);

        /* Fills statement. */
        int i = 1;
        preparedStatement.setString(i++, loginName);

        /* Executes query. */
        resultSet = preparedStatement.executeQuery();

        /* Checks errors. */
        if (!resultSet.next()) {
            throw new InstanceNotFoundException(loginName);
        }

        /* Gets results. */
        i = 1;
        String enPassword = resultSet.getString(i++);
        String firstName = resultSet.getString(i++);
        String surname = resultSet.getString(i++);
        String document = resultSet.getString(i++);
        String email = resultSet.getString(i++);
        String address = resultSet.getString(i++);
        String details = resultSet.getString(i++);
        String phone = resultSet.getString(i++);
        Boolean isAdministrator = resultSet.getBoolean(i++);

        /* Return the value object. */
        return new UserProfileVO(loginName, enPassword,
                new UserProfileDetailsVO(firstName, surname,
          document, email, address, details), phone,
                new UserProfileAccessVO(isAdministrator));

    } catch (SQLException e) {
        throw new InternalErrorException(e);
    } finally {
        GeneralOperations.closeResultSet(resultSet);
```

```
            GeneralOperations.closeStatement(preparedStatement);
        }

    }

    /**
     * Finds all users ordered by propertyName.
     * <BR>
     * Returns a collection of <code>count</code> objects starting in
     * <code>startIndex</code> object. If <code>count</code> is
     * <code>0</code>
     * returns a collection of all objects.
     * @param connection
     * @param propertyByName
     * @param startIndex
     * @param count
     * @throws application.util.exceptions.InternalErrorException
     * @return a collection of user profiles
     */
    public Collection findBy(Connection connection,String
      propertyByName, Integer startIndex, Integer count) throws
      InternalErrorException {

        PreparedStatement preparedStatement = null;
        ResultSet resultSet = null;

        try {

            /* Creates statement. */
            String queryString = "SELECT loginName, firstName,
            surname, " + "document, email, address, details,
      phone, isAdministrator" + " FROM
      user_profile_registry ORDER BY ?";

            preparedStatement =
            connection.prepareStatement(queryString,
                        ResultSet.TYPE_SCROLL_INSENSITIVE,
                        ResultSet.CONCUR_READ_ONLY);

            /* Checks propertyByName validity and sets default value
             * if it isn´t correct. */
            int i = 1;
            if(UserProfileTable.validate(propertyByName)){
                preparedStatement.setString(i++,propertyByName);
            }else{
                preparedStatement.setString(i++,
            UserProfileTable.LOGINNAME);
            }

            /* Executes query. */
            resultSet = preparedStatement.executeQuery();

            /* Gets objects. Gets a collection of "count" objects
             * starting in "startIndex" object. If "count" is "0"
             * returns a collection of all objects.*/
            Collection<UserProfileVO> userProfileVOs =
                    new ArrayList<UserProfileVO>();

            int currentCount = 0;

            if ((startIndex >=1) && resultSet.absolute(startIndex)) {
                do {
                    i = 1;
                    String loginName = resultSet.getString(i++);
                    String firstName = resultSet.getString(i++);
                    String surname = resultSet.getString(i++);
                    String document = resultSet.getString(i++);
                    String email = resultSet.getString(i++);
                    String address = resultSet.getString(i++);
                    String details = resultSet.getString(i++);
                    String phone = resultSet.getString(i++);
                    Boolean isAdministrator =
                  resultSet.getBoolean(i++);

                    userProfileVOs.add(new UserProfileVO(loginName,
                             "Not avaliable", new
                       UserProfileDetailsVO(firstName,
                                surname, document, email, address,
                             details), phone, new
                   UserProfileAccessVO(isAdministrator)));

                    currentCount++;

                } while (resultSet.next() && ((currentCount < count)
                  || (count == 0)) ) ;
            }

            /* Returns value objects. */
            return userProfileVOs;

        } catch (SQLException e) {
            throw new InternalErrorException(e);
        } finally {
            GeneralOperations.closeResultSet(resultSet);
            GeneralOperations.closeStatement(preparedStatement);
        }

    }

    /**
     * Finds a pattern in an all users profile attribute ordered by
     * propertyName.
     * <BR>
     * Returns a collection of <code>count</code> objects starting in
     * <code>startIndex</code> object. If <code>count</code> is
     * <code>0</code>
     * returns a collection of all objects.
     * @param connection
     * @param propertyInName Attribute where tries to find a pattern.
     * @param propertyInValue Pattern to find.
     * @param propertyByName Attribute used for ordering result.
     * @param startIndex
```

```
  * @param count
  * @throws application.util.exceptions.InternalErrorException
  * @return
  */
 public Collection findInBy(Connection connection, String
   propertyInName, String propertyInValue, String propertyByName,
   Integer startIndex, Integer count) throws InternalErrorException
   {

     PreparedStatement preparedStatement = null;
     ResultSet resultSet = null;

     try {

         /* Creates statement. */
         String queryString = "SELECT loginName, firstName,
               surname, " + "document, email, address,
         details, phone, isAdministrator"
                     + " FROM user_profile_registry";

         /* Checks validity of propertyInName. */
         if(UserProfileTable.validate(propertyInName)){
             queryString = queryString + " WHERE " + propertyInName
                     + " LIKE ?";
         }

         /* Checks validity of propertyByName. */
         if(UserProfileTable.validate(propertyByName)){
             queryString = queryString + " ORDER BY " +
                     propertyByName;
         }

         preparedStatement =
         connection.prepareStatement(queryString,
                 ResultSet.TYPE_SCROLL_INSENSITIVE,
                 ResultSet.CONCUR_READ_ONLY);

         int i = 1;
         if(UserProfileTable.validate(propertyInName)){
             preparedStatement.setString(i++,propertyInValue);
         }

         /* Executes query. */
         resultSet = preparedStatement.executeQuery();

         /* Gets objects. Gets a collection of "count" objects
          * starting in "startIndex" object. If "count" is "0"
   * returns a collection of all objects.*/
         Collection<UserProfileVO> userProfileVOs =
                 new ArrayList<UserProfileVO>();

         int currentCount = 0;

         if ((startIndex >=1) && resultSet.absolute(startIndex)) {
             do {
                 i = 1;
```

```
                    String loginName = resultSet.getString(i++);
                    String firstName = resultSet.getString(i++);
                    String surname = resultSet.getString(i++);
                    String document = resultSet.getString(i++);
                    String email = resultSet.getString(i++);
                    String address = resultSet.getString(i++);
                    String details = resultSet.getString(i++);
                    String phone = resultSet.getString(i++);
                    Boolean isAdministrator =
                  resultSet.getBoolean(i++);

                    userProfileVOs.add(new UserProfileVO(loginName,
                             "Not avaliable", new
                       UserProfileDetailsVO(firstName,
                                   surname, document, email, address,
                             details), phone, new
                    UserProfileAccessVO(isAdministrator)));

                    currentCount++;

                } while (resultSet.next() && ((currentCount < count)
                  || (count == 0)) ) ;
            }

            /* Returns value objects. */
            return userProfileVOs;

        } catch (SQLException e) {
            throw new InternalErrorException(e);
        } finally {
            GeneralOperations.closeResultSet(resultSet);
            GeneralOperations.closeStatement(preparedStatement);
        }

    }

}
```

10.5.3.2.b Alta de usuarios en el servicio

10.5.3.2.b.1 application.model.holidaysrequestservice.dao.holidaysservice

SQLHolidaysServiceDAO

```
package application.model.holidaysrequestservice.dao.holidaysservice;
```

```java
import java.io.Serializable;

import java.util.Calendar;
import java.util.Collection;

/* Uses jdbc. */
import java.sql.Connection;

import application.model.holidaysrequestservice.vo.holidaysservice.
HolidaysServiceVO;

import application.util.exceptions.DuplicateInstanceException;
import application.util.exceptions.InstanceNotFoundException;
import application.util.exceptions.InternalErrorException;

/**
 * This interface defines basic methods for interacting with holidays
 * service registry datas in data access layer.
 * <BR>
 * Upper layers use this interface, not its implementation. With
 * <CODE>SQLHolidaysServiceDAOFactory<CODE>, upper layers can
 * determine which implementation must be used. This makes code
 * independent from each data access implementation and datasource.
 */
public interface SQLHolidaysServiceDAO extends Serializable {

    /**
     * Creates a new row with and user holidays service data.
     * @param connection
     * @param holidaysServiceVO
     * @throws application.util.exceptions.DuplicateInstanceException
     * @throws application.util.exceptions.InternalErrorException
     */
    public void create(Connection connection,
            HolidaysServiceVO holidaysServiceVO) throws
            DuplicateInstanceException, InternalErrorException;

    /**
     * Checks if a user holidays service already exists.
     * @param connection
     * @param loginName
     * @param period
     * @throws application.util.exceptions.InternalErrorException
     * @return
     */
    public Boolean exists(Connection connection, String loginName,
            Calendar period) throws
            InternalErrorException;

    /**
     * Tries to update an already existing user holidays service.
     * @param connection
     * @param holidaysServiceVO
     * @throws application.util.exceptions.InstanceNotFoundException
     * @throws application.util.exceptions.InternalErrorException
     */
```

```
public void update(Connection connection,
        HolidaysServiceVO holidaysServiceVO) throws
        InstanceNotFoundException, InternalErrorException;

/**
 * Finds an user holidays service with a loginName and period
 * given.
 * @param connection
 * @param loginName
 * @param period
 * @throws application.util.exceptions.InstanceNotFoundException
 * @throws application.util.exceptions.InternalErrorException
 * @return
 */
public HolidaysServiceVO find(Connection connection, String
  loginName, Calendar period) throws
        InstanceNotFoundException, InternalErrorException;

/**
 * Finds all user holidays services.
 * <BR>
 * Returns a collection of <code>count</code> user holidays
 * services. If count is <code>0</code> returns all user holidays
 * services.
 * <BR>
 * StartIndex represents the first holidays service of the
 * collection.
 * @return a collection of user holidays services
 * @param connection
 * @param loginName
 * @param startIndex
 * @param count
 * @throws application.util.exceptions.InternalErrorException
 */
public Collection findUserHolidaysServices(Connection connection,
  String loginName, Integer startIndex, Integer count) throws
  InternalErrorException;

/**
 * Finds all users registered in a given holidays service period.
 * <BR>
 * Returns a collection of <code>count</code> user holidays
 * services. If count is <code>0</code> returns all user
 * registered in the given period.
 * <BR>
 * StartIndex represents the first holidays service of the
 * collection.
 * @return a collection of user holidays services
 * @param connection
 * @param period
 * @param startIndex
 * @param count
 * @throws application.util.exceptions.InternalErrorException
 */
```

```
    public Collection findPeriodHolidaysService(Connection connection,
            Calendar period, Integer startIndex, Integer count) throws
            InternalErrorException;

    /**
     * Finds all existing holidays services.
     * <BR>
     * Returns a collection of <code>count</code> holidays services
     * registered period. If count is <code>0</code> returns all
     * period registered.
     * <BR>
     * StartIndex represents the first holidays service period of the
     * collection.
     * @return a collection of holidays services periods
     * @param connection
     * @param startIndex
     * @param count
     * @throws application.util.exceptions.InternalErrorException
     */
    public Collection findHolidaysServices(Connection connection,
      Integer startIndex, Integer count) throws
      InternalErrorException;

}
```

SQLHolidaysServiceDAOFactory

```
package application.model.holidaysrequestservice.dao.holidaysservice;

/* Uses JNDI. */
import application.util.jndi.ConfigurationParametersManager;

import application.util.exceptions.InternalErrorException;

/**
 * This class creates an intance of the implementation of
 * <code>SQLHolidaysServiceDAO</code>.
 * <BR>
 * Upper layers use this class to determine which implementation must
 * be used. This makes code independent from each data access
 * implementation and datasource.
 * <BR>
 * <code>SQLHolidaysServiceDAOFactory/daoClassName</code> parameter
 * has to be defined using JNDI. This parameter can be defined in
 * <code>web.xml</code> with <code><env-entry></code> tags.
 * <BR>
 * This code provide an implementation of
 * <code>SQLHolidaysServiceDAO</code> for
 * MySQL in <code>StandarSQLHolidaysServiceDAO</code>.
 */
public final class SQLHolidaysServiceDAOFactory {

    /* This parameter has to be define in a JNDI context. */
    private final static String DAO_CLASS_NAME_PARAMETER =
            "SQLHolidaysServiceDAOFactory/daoClassName";

    private final static Class daoClass = getDAOClass();

    private SQLHolidaysServiceDAOFactory() {}

    /* Gets the name of the implementation class of DAO. */
    private static Class getDAOClass() {

        Class theClass = null;
        try {
            String daoClassName =
            ConfigurationParametersManager.getParameter(
                    DAO_CLASS_NAME_PARAMETER);
            theClass = Class.forName(daoClassName);
        } catch (Exception e) {
            e.printStackTrace();
        }

        return theClass;
    }

    /**
     * Returns an instance of the implementation class of DAO.
     * @throws application.util.exceptions.InternalErrorException
     * @return implementation of data access object
     */
    public static SQLHolidaysServiceDAO getDAO() throws
      InternalErrorException {
        try {
            return (SQLHolidaysServiceDAO) daoClass.newInstance();
        } catch (Exception e) {
            throw new InternalErrorException(e);
        }
    }

}
```

<u>StandardSQLHolidaysServiceDAO</u>

```
package application.model.holidaysrequestservice.dao.holidaysservice;

import java.util.Calendar;
import java.util.ArrayList;
import java.util.Collection;

/* JDBC.*/
import java.sql.Connection;
import java.sql.SQLException;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
```

```
import
application.model.holidaysrequestservice.dao.holidaysservice.SQLHolida
ysServiceDAO;

import
application.model.holidaysrequestservice.vo.holidaysservice.HolidaysSe
rviceVO;

import application.util.exceptions.DuplicateInstanceException;
import application.util.exceptions.InstanceNotFoundException;
import application.util.exceptions.InternalErrorException;

import application.util.sql.GeneralOperations;

/**
 * This class provides an implementation of
 * <code>SQLHolidaysServiceDAO</code> for
 * interacting with user profile datas in data access layer. This is
 * an implementation for a relational database, MySQL, using JDBC
 * driver.
 */
public class StandardSQLHolidaysServiceDAO implements
SQLHolidaysServiceDAO {

    /**
     * Creates a new row with and user holidays service. If a user
     * with the same
     * loginName and period exists throws a
     * <CODE>DuplicateInstanceException</CODE>.
     * @param connection
     * @param holidaysServiceVO
     * @throws application.util.exceptions.DuplicateInstanceException
     * @throws application.util.exceptions.InternalErrorException
     */
    public void create(Connection connection,
            HolidaysServiceVO holidaysServiceVO) throws
            DuplicateInstanceException, InternalErrorException {

        /* Check if the service already exists. */
        if (exists(connection, holidaysServiceVO.getLoginName(),
                holidaysServiceVO.getPeriod())) {
            throw new DuplicateInstanceException(
                    holidaysServiceVO.getLoginName() +  "-"
                    +
              holidaysServiceVO.getPeriod().get(Calendar.YEAR));
        }

        PreparedStatement preparedStatement = null;

        try {

            /* Creates statement. */
            String queryString = "INSERT INTO
            holidays_service_registry" + " (loginName, period,
      days) VALUES (?, ?, ?)";
```

```
        preparedStatement =
        connection.prepareStatement(queryString);

        /* Fills statement. */
        int i = 1;
        preparedStatement.setString(i++,
        holidaysServiceVO.getLoginName());
        preparedStatement.setInt(i++,
              holidaysServiceVO.getPeriod().get(Calendar.YEAR));
        preparedStatement.setInt(i++,
        holidaysServiceVO.getDays().intValue());

        /* Executes query. */
        int insertedRows = preparedStatement.executeUpdate();

        /* Checks errors. */
        if (insertedRows == 0) {
            throw new SQLException("Can not add row to table
        'holidays_service_registry'");
        }

        if (insertedRows > 1) {
            throw new SQLException("Duplicate row for key = '" +
                    holidaysServiceVO.getLoginName() + "-" +

        holidaysServiceVO.getPeriod().get(Calendar.YEAR) +
              "' in table 'holidays_service_registry'");
        }

    } catch (SQLException e) {
        throw new InternalErrorException(e);
    } finally {
        GeneralOperations.closeStatement(preparedStatement);
    }
}

/**
 * Checks if a user holidays service already exists. Returns
 * <code>true</code> if a period for a loginName already exists
 * and <code>false</code> if not.
 * @return if a user holidays service exists or not.
 * @param connection
 * @param loginName
 * @param period
 * @throws application.util.exceptions.InternalErrorException
 */
public Boolean exists(Connection connection, String loginName,
        Calendar period) throws InternalErrorException {

    PreparedStatement preparedStatement = null;
    ResultSet resultSet = null;

    try {

        /* Create "preparedStatement". */
```

```
            String queryString = "SELECT loginName, period FROM
            holidays_service_registry" + " WHERE loginName = ?
      AND period = ?";

            preparedStatement =
            connection.prepareStatement(queryString);

            /* Fill "preparedStatement". */
            int i = 1;
            preparedStatement.setString(i++, loginName);
            preparedStatement.setInt(i++, period.get(Calendar.YEAR));

            /* Execute query. */
            resultSet = preparedStatement.executeQuery();

            return resultSet.next();

        } catch (SQLException e) {
            throw new InternalErrorException(e);
        } finally {
            GeneralOperations.closeResultSet(resultSet);
            GeneralOperations.closeStatement(preparedStatement);
        }

    }

    /**
     * Tries to update an already existing user holidays service.
     * @param connection
     * @param holidaysServiceVO
     * @throws application.util.exceptions.InstanceNotFoundException
     * @throws application.util.exceptions.InternalErrorException
     */
    public void update(Connection connection,
            HolidaysServiceVO holidaysServiceVO) throws
            InstanceNotFoundException, InternalErrorException {

        PreparedStatement preparedStatement = null;

        try {

            /* Creates statement. */
            String queryString = "UPDATE holidays_service_registry" +
                    " SET days = ? " +
                    " WHERE loginName = ? AND period = ?";

            preparedStatement =
            connection.prepareStatement(queryString);

            /* Fills statement. */
            int i = 1;
            preparedStatement.setInt(i++,
            holidaysServiceVO.getDays());
            preparedStatement.setString(i++,
            holidaysServiceVO.getLoginName());
            preparedStatement.setInt(i++,
                  holidaysServiceVO.getPeriod().get(Calendar.YEAR));

            /* Executes query. */
            int updatedRows = preparedStatement.executeUpdate();

            /* Checks errors. */
            if (updatedRows == 0) {
                throw new InstanceNotFoundException(
                        holidaysServiceVO.getLoginName() + "-" +
                    holidaysServiceVO.getPeriod().get(Calendar.YEAR));
            }

            if (updatedRows > 1) {
                throw new SQLException("Duplicate row for key = '" +
                        holidaysServiceVO.getLoginName() + "-" +
                      holidaysServiceVO.getPeriod().get(Calendar.YEAR)
                        + "' in table 'holidays_service_registry'");
            }

        /* Catches exceptions and close stament. */
        } catch (SQLException e) {
            throw new InternalErrorException(e);
        } finally {
            GeneralOperations.closeStatement(preparedStatement);
        }
    }

    /**
     * Finds an user holidays service with a loginName and period
     * given.
     * @return user holidays service
     * @param connection
     * @param loginName
     * @param period
     * @throws application.util.exceptions.InstanceNotFoundException
     * @throws application.util.exceptions.InternalErrorException
     */
    public HolidaysServiceVO find(Connection connection, String
      loginName, Calendar period) throws
            InstanceNotFoundException, InternalErrorException {

        PreparedStatement preparedStatement = null;
        ResultSet resultSet = null;

        try {

            /* Creates statement. */
            String queryString = "SELECT days FROM
            holidays_service_registry " + " WHERE loginName = ?
      AND period = ?";

            preparedStatement =
            connection.prepareStatement(queryString);

            /* Fills statement. */
            int i = 1;
```

```
            preparedStatement.setString(i++, loginName);
            preparedStatement.setInt(i++, period.get(Calendar.YEAR));

            /* Executes query. */
            resultSet = preparedStatement.executeQuery();

            /* Checks errors. */
            if (!resultSet.next()) {
                throw new InstanceNotFoundException(loginName + "-" +
                        period.get(Calendar.YEAR));
            }

            /* Gets results. */
            i = 1;
            Integer days = resultSet.getInt(i++);

            /* Returns value object. */
            return new HolidaysServiceVO(loginName, period, days);

        } catch (SQLException e) {
            throw new InternalErrorException(e);
        } finally {
            GeneralOperations.closeResultSet(resultSet);
            GeneralOperations.closeStatement(preparedStatement);
        }
    }

    /**
     * Finds all user holidays services.
     * <BR>
     * Returns a collection of <code>count</code> user holidays
     * services. If count is <code>0</code> returns all user holidays
     * services.
     * <BR>
     * StartIndex represents the first holidays service of the
     * collection.
     * @return a collection of user holidays services
     * @param connection
     * @param loginName
     * @param startIndex
     * @param count
     * @throws application.util.exceptions.InternalErrorException
     */
    public Collection findUserHolidaysServices(Connection connection,
      String loginName, Integer startIndex, Integer count) throws
      InternalErrorException {

        PreparedStatement preparedStatement = null;
        ResultSet resultSet = null;

        try {

            /* Creates statement. */
            String queryString = "SELECT period, days" +
                    " FROM holidays_service_registry WHERE loginName=
              ?" + " ORDER BY period DESC";
            preparedStatement =
            connection.prepareStatement(queryString,
                    ResultSet.TYPE_SCROLL_INSENSITIVE,
                    ResultSet.CONCUR_READ_ONLY);

            int i = 1;
            preparedStatement.setString(i++, loginName);

            /* Executes query. */
            resultSet = preparedStatement.executeQuery();

            /* Gets objects. Gets a collection of "count" objects
             * starting in "startIndex" object. If "count" is "0"
             * returns a collection of all objects.*/
            Collection<HolidaysServiceVO> holidaysServiceVOs =
                    new ArrayList<HolidaysServiceVO>();
            int currentCount = 0;

            if ((startIndex >=1) && resultSet.absolute(startIndex)) {
                do {
                    i = 1;
                    Integer year = resultSet.getInt(i++);
                    Integer days = resultSet.getInt(i++);

                    Calendar period = Calendar.getInstance();
                    period.clear();
                    period.set(Calendar.YEAR, year);

                    holidaysServiceVOs.add(new
                  HolidaysServiceVO(loginName, period, days));

                    currentCount++;

                } while (resultSet.next() && ((currentCount < count)
            || (count == 0)) ) ;
            }

            /* Returns value objects. */
            return holidaysServiceVOs;

        } catch (SQLException e) {
            throw new InternalErrorException(e);
        } finally {
            GeneralOperations.closeResultSet(resultSet);
            GeneralOperations.closeStatement(preparedStatement);
        }
    }

    /**
     * Finds all users registered in a given holidays service period.
     * <BR>
     * Returns a collection of <code>count</code> user holidays
     * services. If count is <code>0</code> returns all user
     * registered in the given period.
     * <BR>
     * StartIndex represents the first holidays service of the
     * collection.
     * @return a collection of user holidays services
     * @param connection
     * @param period
     * @param startIndex
     * @param count
     * @throws application.util.exceptions.InternalErrorException
     */
    public Collection findPeriodHolidaysService(
            Connection connection, Calendar period, Integer
      startIndex, Integer count) throws InternalErrorException {

        PreparedStatement preparedStatement = null;
        ResultSet resultSet = null;

        try {

            /* Creates statement. */
            String queryString = "SELECT loginName, days" +
                    " FROM holidays_service_registry WHERE period= ?"
              + " ORDER BY loginName";

            preparedStatement =
            connection.prepareStatement(queryString,
                    ResultSet.TYPE_SCROLL_INSENSITIVE,
                    ResultSet.CONCUR_READ_ONLY);

            int i = 1;
            preparedStatement.setInt(i++, period.get(Calendar.YEAR));

            /* Executes query. */
            resultSet = preparedStatement.executeQuery();

            /* Gets objects. Gets a collection of "count" objects
             * starting in "startIndex" object. If "count" is "0"
             * returns a collection of all objects.*/
            Collection<HolidaysServiceVO> holidaysServiceVOs =
                    new ArrayList<HolidaysServiceVO>();
            int currentCount = 0;

            if ((startIndex >=1) && resultSet.absolute(startIndex)) {
                do {
                    i = 1;
                    String loginName = resultSet.getString(i++);
                    Integer days = resultSet.getInt(i++);

                    holidaysServiceVOs.add(new
                  HolidaysServiceVO(loginName, period, days));

                    currentCount++;

                } while (resultSet.next() && ((currentCount < count)
            || (count == 0)) ) ;
            }
```

```
            /* Returns value objects. */
            return holidaysServiceVOs;

        } catch (SQLException e) {
            throw new InternalErrorException(e);
        } finally {
            GeneralOperations.closeResultSet(resultSet);
            GeneralOperations.closeStatement(preparedStatement);
        }
    }

    /**
     * Finds all existing holidays services.
     * <BR>
     * Returns a collection of <code>count</code> holidays services
     * registered period. If count is <code>0</code> returns all
     * period registered.
     * <BR>
     * StartIndex represents the first holidays service period of the
     * collection.
     * @return a collection of holidays services periods
     * @param connection
     * @param startIndex
     * @param count
     * @throws application.util.exceptions.InternalErrorException
     */
    public Collection findHolidaysServices(Connection connection,
            Integer startIndex, Integer count) throws
      InternalErrorException {

        PreparedStatement preparedStatement = null;
        ResultSet resultSet = null;

        try {

            /* Creates statement. */
            String queryString = "SELECT DISTINCT period" +
                    " FROM holidays_service_registry ORDER BY period
              DESC";

            preparedStatement =
            connection.prepareStatement(queryString,
                    ResultSet.TYPE_SCROLL_INSENSITIVE,
                    ResultSet.CONCUR_READ_ONLY);

            /* Executes query. */
            resultSet = preparedStatement.executeQuery();

            /* Gets objects. Gets a collection of "count" objects
             * starting in "startIndex" object. If "count" is "0"
             * returns a collection of all objects.*/
            Collection<Calendar> periods =
                    new ArrayList<Calendar>();

            int currentCount = 0;
            int i;

            if ((startIndex >=1) && resultSet.absolute(startIndex)) {
                do {

                    i = 1;
                    Integer year = resultSet.getInt(i++);

                    Calendar period = Calendar.getInstance();
                    period.clear();
                    period.set(Calendar.YEAR, year);

                    periods.add(period);

                    currentCount++;

                } while (resultSet.next() && ((currentCount < count)
            || (count == 0)) ) ;
            }

            /* Returns value objects. */
            return periods;

        } catch (SQLException e) {
            throw new InternalErrorException(e);
        } finally {
            GeneralOperations.closeResultSet(resultSet);
            GeneralOperations.closeStatement(preparedStatement);
        }
    }

}
```

10.5.3.2.c Solicitud de vacaciones

10.5.3.2.c.1 application.model.holidaysrequestservice.dao.holidaysrequest

SQLHolidaysRequestDAO

```
package application.model.holidaysrequestservice.dao.holidaysrequest;

import java.io.Serializable;

import java.util.Calendar;
import java.util.Collection;

/* Uses jdbc. */
import java.sql.Connection;
import application.model.holidaysrequestservice.vo.holidaysrequest.
HolidaysRequestVO;

import application.util.exceptions.DuplicateInstanceException;
import application.util.exceptions.InstanceNotFoundException;
import application.util.exceptions.InternalErrorException;

/**
 * This interface defines basic methods for interacting with holidays
 * request datas in data access layer.
 * <BR>
 * Upper layers use this interface, not its implementation. With
 * <CODE>SQLHolidaysRequestDAOFactory<CODE>, upper layers can
 * determine which implementation must be used. This makes code
 * independent from each data access implementation and datasource.
 */
public interface SQLHolidaysRequestDAO {

    /**
     * Creates a new row with and user holidays request data.
     * @param connection
     * @param holidaysRequestVO
     * @throws application.util.exceptions.DuplicateInstanceException
     * @throws application.util.exceptions.InternalErrorException
     * @return autoId identifier for the request
     */
    public Integer create(Connection connection,
            HolidaysRequestVO holidaysRequestVO) throws
            DuplicateInstanceException, InternalErrorException;

    /**
     * Checks if a user holidays request already exists.
     * @param connection
     * @param loginName
     * @param startDate
     * @param finalDate
     * @throws application.util.exceptions.InternalErrorException
     * @return
     */
    public Boolean exists(Connection connection, String loginName,
            Calendar startDate, Calendar finalDate) throws
            InternalErrorException;

    /**
     * Tries to update state of an already existing user holidays
     * request.
     * @param connection
     * @param holidaysRequestVO
     * @throws application.util.exceptions.InstanceNotFoundException
     * @throws application.util.exceptions.InternalErrorException
     */
    public void update(Connection connection,
            HolidaysRequestVO holidaysRequestVO) throws
            InstanceNotFoundException, InternalErrorException;
```

```
/**
 * Finds a holidays request with an autoId given.
 * @param connection
 * @param autoId
 * @throws application.util.exceptions.InstanceNotFoundException
 * @throws application.util.exceptions.InternalErrorException
 * @return
 */
public HolidaysRequestVO findId(Connection connection, Integer
  autoId) throws InstanceNotFoundException,
  InternalErrorException;

/**
 * Finds an user holidays request with an autoId given.
 * @param connection
 * @param autoId
 * @param loginName
 * @throws application.util.exceptions.InstanceNotFoundException
 * @throws application.util.exceptions.InternalErrorException
 * @return
 */
public HolidaysRequestVO findIdUser(Connection connection, Integer
        autoId, String loginName) throws InstanceNotFoundException,
        InternalErrorException;

/**
 * Finds all user holidays requests ordered by propertyName.
 * @param connection
 * @param loginName
 * @param propertyName
 * @param isDesc
 * @param startIndex
 * @param count
 * @throws application.util.exceptions.InternalErrorException
 * @return
 */
public Collection findUserBy(Connection connection, String
  loginName, String propertyName, Boolean isDesc, Integer
  startIndex, Integer count) throws InternalErrorException;

/**
 * Finds all user holidays requests in a given period with a given
 * state ordered by propertyName.
 * @param connection
 * @param loginName
 * @param period
 * @param state
 * @param propetyName
 * @param startIndex
 * @param count
 * @throws application.util.exceptions.InternalErrorException
 * @return
 */
public Collection findUserPeriodStateBy(Connection connection,
        String loginName, Calendar period, Short state,
            String propetyName, Integer startIndex, Integer count)
      throws InternalErrorException;

    /**
     * Carries out an advanced search based on loginName, autoId,
     * period, sinceDate, state, order by, ascendant or descendant.
     * @param connection
     * @param loginName
     * @param autoId
     * @param period
     * @param sinceDate
     * @param state
     * @param propertyName
     * @param isDesc
     * @param startIndex
     * @param count
     * @throws application.util.exceptions.InternalErrorException
     * @return
     */
    public Collection findAdvanced(Connection connection, String
      loginName, Integer autoId, Calendar period, Calendar
      sinceDate, Short state, String propertyName, Boolean
      isDesc, Integer startIndex, Integer count)
            throws InternalErrorException;

    /**
     * Returns all periods with at least a holidays request of any
     * user.
     * @param connection
     * @param startIndex
     * @param count
     * @throws application.util.exceptions.InternalErrorException
     * @return
     */
    public Collection findPeriods(Connection connection, Integer
      startIndex, Integer count) throws InternalErrorException;
}
```

<u>SQLHolidaysRequestDAOFactory</u>

```
package application.model.holidaysrequestservice.dao.holidaysrequest;

/* Uses JNDI. */
import application.util.jndi.ConfigurationParametersManager;

import application.util.exceptions.InternalErrorException;

/**
 * This class creates an intance of the implementation of
 * <code>SQLHolidaysRequestDAO</code>.
 * <BR>
 * Upper layers use this class to determine which implementation must
 * be used. This makes code independent from each data access
 * implementation and datasource.
 * <BR>
 * <code>SQLHolidaysRequestDAOFactory/daoClassName</code> parameter
 * has to be defined using JNDI. This parameter can be defined in
 * <code>web.xml</code> with <code><env-entry></code> tags.
 * <BR>
 * This code provide an implementation of
 * <code>SQLHolidaysRequestDAO</code> for
 * MySQL in <code>StandarSQLHolidaysRequestDAO</code>.
 */
public final class SQLHolidaysRequestDAOFactory {

    /* This parameter has to be define in a JNDI context. */
    private final static String DAO_CLASS_NAME_PARAMETER =
            "SQLHolidaysRequestDAOFactory/daoClassName";

    private final static Class daoClass = getDAOClass();

    private SQLHolidaysRequestDAOFactory() {}

    /* Gets the name of the implementation class of DAO. */
    private static Class getDAOClass() {

        Class theClass = null;
        try {
            String daoClassName =
            ConfigurationParametersManager.getParameter(
                    DAO_CLASS_NAME_PARAMETER);
            theClass = Class.forName(daoClassName);
        } catch (Exception e) {
            e.printStackTrace();
        }

        return theClass;
    }

    /**
     * Returns an instance of the implementation class of DAO.
     * @throws application.util.exceptions.InternalErrorException
     * @return implementation of data access object
     */
    public static SQLHolidaysRequestDAO getDAO() throws
      InternalErrorException {
        try {
            return (SQLHolidaysRequestDAO) daoClass.newInstance();
        } catch (Exception e) {
            throw new InternalErrorException(e);
        }
    }

}
```

StandardSQLHolidaysRequestDAO

```
package application.model.holidaysrequestservice.dao.holidaysrequest;

import java.util.ArrayList;
import java.util.Calendar;
import java.util.Collection;

/* JDBC.*/
import java.sql.Connection;
import java.sql.SQLException;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.Statement;
import java.sql.Timestamp;

import application.model.holidaysrequestservice.dao.holidaysrequest.
SQLHolidaysRequestDAO;

import application.model.holidaysrequestservice.vo.holidaysrequest.
HolidaysRequestVO;

import application.util.exceptions.DuplicateInstanceException;
import application.util.exceptions.InstanceNotFoundException;
import application.util.exceptions.InternalErrorException;

import application.util.sql.GeneralOperations;

import application.model.util.HolidaysRequestTable;

/**
 * This class provides an implementation of
 * <code>SQLHolidaysRequestDAO</code> for interacting with user
 * profile datas in data access layer. This is an implementation for a
 * relational database, MySQL, using JDBC driver.
 */
public class StandardSQLHolidaysRequestDAO implements
      SQLHolidaysRequestDAO {

    /**
     * Creates a new row with and user holidays request data.
     * @param connection
     * @param holidaysRequestVO
     * @throws application.util.exceptions.DuplicateInstanceException
     * @throws application.util.exceptions.InternalErrorException
     * @return autoId identifier for the request
     */
    public Integer create(Connection connection, HolidaysRequestVO
      holidaysRequestVO) throws DuplicateInstanceException,
      InternalErrorException {

        /* Checks if the request already exists. */
        if (exists(connection, holidaysRequestVO.getLoginName(),
                holidaysRequestVO.getStartDate(),
                holidaysRequestVO.getFinalDate())) {

            throw new DuplicateInstanceException(
                    holidaysRequestVO.getLoginName() + "-" +
                    holidaysRequestVO.getStartDate().get(
                    Calendar.DAY_OF_MONTH) + "." +
                    (holidaysRequestVO.getStartDate().get(
                    Calendar.MONTH)+1) + "." +
                   holidaysRequestVO.getStartDate().get(Calendar.YEAR)
                    + "-" + holidaysRequestVO.getFinalDate().get(
                    Calendar.DAY_OF_MONTH) + "." +
                    (holidaysRequestVO.getFinalDate().get(
                    Calendar.MONTH)+1) + "." +
                 holidaysRequestVO.getFinalDate().get(Calendar.YEAR));
        }

        PreparedStatement preparedStatement = null;
        ResultSet rs = null;

        try {

            /* Creates statement. */
            String queryString = "INSERT INTO
            holidays_request_registry" + " (loginName, period,
      stateDate, startDate, finalDate, state, " +
                  "details) VALUES (?, ?, ?, ?, ?, ?, ?)";

            preparedStatement =
            connection.prepareStatement(queryString,
            Statement.RETURN_GENERATED_KEYS);

            /* Fills statement. */
            int i = 1;
            preparedStatement.setString(i++,
            holidaysRequestVO.getLoginName());
            preparedStatement.setInt(i++,
                 holidaysRequestVO.getStartDate().get(Calendar.YEAR));
            preparedStatement.setTimestamp(i++, new Timestamp(
                 Calendar.getInstance().getTime().getTime()));
            preparedStatement.setTimestamp(i++, new Timestamp(
               holidaysRequestVO.getStartDate().getTime().getTime()));
            preparedStatement.setTimestamp(i++, new Timestamp(
               holidaysRequestVO.getFinalDate().getTime().getTime()));
            preparedStatement.setShort(i++,
            holidaysRequestVO.getState());

      preparedStatement.setString(i++,holidaysRequestVO.getDetails());

            /* Executes query. */
            int insertedRows = preparedStatement.executeUpdate();

            /* Checks errors. */
            if (insertedRows == 0) {
                throw new SQLException("Can not add row to table " +
                        "'holidays_request_registry'");
            }

            if (insertedRows > 1) {
                throw new SQLException("Duplicate row for key = '" +
                        holidaysRequestVO.getLoginName() + "-" +
                        holidaysRequestVO.getStartDate().get(
                        Calendar.DAY_OF_MONTH) + "." +
                        (holidaysRequestVO.getStartDate().get(
                        Calendar.MONTH)+1) + "." +
              holidaysRequestVO.getStartDate().get(Calendar.YEAR)
                  + "-" +
                        holidaysRequestVO.getFinalDate().get(
                        Calendar.DAY_OF_MONTH) + "." +
                        (holidaysRequestVO.getFinalDate().get(
                        Calendar.MONTH)+1) + "." +
              holidaysRequestVO.getFinalDate().get(Calendar.YEAR)
                  + "' in table 'holidays_request_registry'");
            }

            /* Gets autoId created. */
            rs = preparedStatement.getGeneratedKeys();

            /* Checks errors. */
            if(rs.next()) {
                return rs.getInt(1);
            } else {
                throw new SQLException("No autoId created");
            }

        } catch (SQLException e) {
            throw new InternalErrorException(e);
        } finally {
            GeneralOperations.closeResultSet(rs);
            GeneralOperations.closeStatement(preparedStatement);
        }
    }

    /**
     * Checks if a user holidays request already exists.
     * @param connection
     * @param loginName
     * @param startDate
     * @param finalDate
     * @throws application.util.exceptions.InternalErrorException
     * @return
     */
    public Boolean exists(Connection connection, String loginName,
            Calendar startDate, Calendar finalDate) throws
            InternalErrorException {

        PreparedStatement preparedStatement = null;
        ResultSet resultSet = null;

        try {

            /* Creates statement. */
            String queryString = "SELECT loginName, startDate,
            finalDate FROM" + " holidays_request_registry WHERE
      loginName = ? AND " + "startDate = ? AND finalDate =
      ?";
```

```
            preparedStatement =
            connection.prepareStatement(queryString);

            /* Fills statement. */
            int i = 1;
            preparedStatement.setString(i++, loginName);
            preparedStatement.setTimestamp(i++, new Timestamp(
                    startDate.getTime().getTime()));
            preparedStatement.setTimestamp(i++, new Timestamp(
                    finalDate.getTime().getTime()));

            /* Executes query. */
            resultSet = preparedStatement.executeQuery();

            return resultSet.next();

        } catch (SQLException e) {
            throw new InternalErrorException(e);
        } finally {
            GeneralOperations.closeResultSet(resultSet);
            GeneralOperations.closeStatement(preparedStatement);
        }
    }

    /**
     * Tries to update state of an already existing user holidays
     * request.
     * @param connection
     * @param holidaysRequestVO
     * @throws application.util.exceptions.InstanceNotFoundException
     * @throws application.util.exceptions.InternalErrorException
     */
    public void update(Connection connection,
            HolidaysRequestVO holidaysRequestVO) throws
            InstanceNotFoundException, InternalErrorException {

        PreparedStatement preparedStatement = null;

        try {

            /* Creates statement. */
            String queryString = "UPDATE holidays_request_registry" +
                    " SET state = ? WHERE loginName = ? AND " +
                    "startDate = ? AND finalDate = ?";

            preparedStatement =
            connection.prepareStatement(queryString);

            /* Fills statement. */
            int i = 1;
            preparedStatement.setShort(i++,
            holidaysRequestVO.getState());
            preparedStatement.setString(i++,
            holidaysRequestVO.getLoginName());
            preparedStatement.setTimestamp(i++, new Timestamp(
                holidaysRequestVO.getStartDate().getTime().getTime()));
            preparedStatement.setTimestamp(i++, new Timestamp(
               holidaysRequestVO.getFinalDate().getTime().getTime()));

            /* Execute query. */
            int updatedRows = preparedStatement.executeUpdate();

            /* Checks errors. */
            if (updatedRows == 0) {
                throw new InstanceNotFoundException(
                        holidaysRequestVO.getLoginName() + "-" +
                        holidaysRequestVO.getStartDate().get(
                        Calendar.DAY_OF_MONTH) + "." +
                        (holidaysRequestVO.getStartDate().get(
                        Calendar.MONTH)+1) + "." +
                    holidaysRequestVO.getStartDate().get(Calendar.YEAR)
                        + "-" +
                        holidaysRequestVO.getFinalDate().get(
                        Calendar.DAY_OF_MONTH) + "." +
                        (holidaysRequestVO.getFinalDate().get(
                        Calendar.MONTH)+1) + "." +
                 holidaysRequestVO.getFinalDate().get(Calendar.YEAR));
            }

            if (updatedRows > 1) {
                throw new SQLException("Duplicate row for key = '" +
                        holidaysRequestVO.getLoginName() + "-" +
                        holidaysRequestVO.getStartDate().get(
                        Calendar.DAY_OF_MONTH) + "." +
                        (holidaysRequestVO.getStartDate().get(
                        Calendar.MONTH)+1) + "." +
                   holidaysRequestVO.getStartDate().get(Calendar.YEAR)
                        + "-" +
                        holidaysRequestVO.getFinalDate().get(
                        Calendar.DAY_OF_MONTH) + "." +
                        (holidaysRequestVO.getFinalDate().get(
                        Calendar.MONTH)+1) + "." +
                   holidaysRequestVO.getFinalDate().get(Calendar.YEAR)
                        + "' in table 'holidays_request_registry'");
            }

        } catch (SQLException e) {
            throw new InternalErrorException(e);
        } finally {
            GeneralOperations.closeStatement(preparedStatement);
        }
    }

    /**
     * Finds a holidays request with a loginName, startDate and
     * finalDate given.
     * @param connection
     * @param loginName
     * @param startDate
     * @param finalDate
     * @throws application.util.exceptions.InstanceNotFoundException
     * @throws application.util.exceptions.InternalErrorException
```

```
     * @return
     */
    public HolidaysRequestVO find(Connection connection, String
      loginName, Calendar startDate, Calendar finalDate) throws
            InstanceNotFoundException, InternalErrorException {

        PreparedStatement preparedStatement = null;
        ResultSet resultSet = null;

        try {

            /* Creates statement. */
            String queryString = "SELECT autoId, period, stateDate,
            state, " + "details FROM holidays_request_registry
      WHERE " + "loginName = ? AND startDate = ? AND
      finalDate = ?";

            preparedStatement =
            connection.prepareStatement(queryString);

            /* Fills statement. */
            int i = 1;
            preparedStatement.setString(i++, loginName);
            preparedStatement.setTimestamp(i++, new Timestamp(
                    startDate.getTime().getTime()));
            preparedStatement.setTimestamp(i++, new Timestamp(
                    finalDate.getTime().getTime()));

            /* Executes query. */
            resultSet = preparedStatement.executeQuery();

            /* Checks errors. */
            if (!resultSet.next()) {
                throw new InstanceNotFoundException(loginName + "-" +
                        startDate.get(Calendar.DAY_OF_MONTH) + "." +
                        (startDate.get(Calendar.MONTH)+1) + "." +
                        startDate.get(Calendar.YEAR) + "-" +
                        finalDate.get(Calendar.DAY_OF_MONTH) + "." +
                        (finalDate.get(Calendar.MONTH)+1) + "." +
                        finalDate.get(Calendar.YEAR));
            }

            /* Gets results. */
            i = 1;
            Integer autoId = resultSet.getInt(i++);
            Integer year = resultSet.getInt(i++);
            Calendar period = Calendar.getInstance();
            period.clear();
            period.set(Calendar.YEAR, year);
            Calendar stateDate = Calendar.getInstance();
            stateDate.setTime(resultSet.getTimestamp(i++));
            Short state = resultSet.getShort(i++);
            String details = resultSet.getString(i++);

            /* Returns value object. */
            return new HolidaysRequestVO(autoId, loginName, period,
            stateDate, startDate, finalDate, state, details);

        } catch (SQLException e) {
            throw new InternalErrorException(e);
        } finally {
            GeneralOperations.closeResultSet(resultSet);
            GeneralOperations.closeStatement(preparedStatement);
        }

    }

    /**
     * Finds a holidays request with an autoId given.
     * @param connection
     * @param autoId
     * @throws application.util.exceptions.InstanceNotFoundException
     * @throws application.util.exceptions.InternalErrorException
     * @return
     */
    public HolidaysRequestVO findId(Connection connection, Integer
      autoId) throws InstanceNotFoundException,
      InternalErrorException{

        PreparedStatement preparedStatement = null;
        ResultSet resultSet = null;

        try {

            /* Creates statement. */
            String queryString = "SELECT loginName, period, stateDate,
                  " + "startDate, finalDate, state, details FROM " +
                  "holidays_request_registry WHERE autoId = ?";

            preparedStatement =
            connection.prepareStatement(queryString);

            /* Fills statement. */
            int i = 1;
            preparedStatement.setInt(i++, autoId);

            /* Executes query. */
            resultSet = preparedStatement.executeQuery();

            /* Checks errors. */
            if (!resultSet.next()) {
                throw new InstanceNotFoundException(autoId);
            }

            /* Gets results. */
            i = 1;
            String loginName = resultSet.getString(i++);
            Integer year = resultSet.getInt(i++);
            Calendar period = Calendar.getInstance();
            period.clear();
            period.set(Calendar.YEAR, year);
            Calendar stateDate = Calendar.getInstance();
            stateDate.setTime(resultSet.getTimestamp(i++));
            Calendar startDate = Calendar.getInstance();
            startDate.setTime(resultSet.getTimestamp(i++));
            Calendar finalDate = Calendar.getInstance();
            finalDate.setTime(resultSet.getTimestamp(i++));
            Short state = resultSet.getShort(i++);
            String details = resultSet.getString(i++);

            /* Returns value object. */
            return new HolidaysRequestVO(autoId, loginName, period,
            stateDate, startDate, finalDate, state, details);

        } catch (SQLException e) {
            throw new InternalErrorException(e);
        } finally {
            GeneralOperations.closeResultSet(resultSet);
            GeneralOperations.closeStatement(preparedStatement);
        }
    }

    /**
     * Finds an user holidays request with an autoId given.
     * @param connection
     * @param autoId
     * @param loginName
     * @throws application.util.exceptions.InstanceNotFoundException
     * @throws application.util.exceptions.InternalErrorException
     * @return
     */
    public HolidaysRequestVO findIdUser(Connection connection, Integer
            autoId, String loginName) throws InstanceNotFoundException,
            InternalErrorException{

        PreparedStatement preparedStatement = null;
        ResultSet resultSet = null;

        try {

            /* Creates statement. */
            String queryString = "SELECT period, stateDate, " +
                    "startDate, finalDate, state, details FROM " +
                    "holidays_request_registry WHERE autoId = ? AND "
              + "loginName LIKE ?";

            preparedStatement =
            connection.prepareStatement(queryString);

            /* Fills statement. */
            int i = 1;
            preparedStatement.setInt(i++, autoId);
            preparedStatement.setString(i++, loginName);

            /* Executes query. */
            resultSet = preparedStatement.executeQuery();
            /* Checks errors. */
            if (!resultSet.next()) {
                throw new InstanceNotFoundException(autoId);
            }

            /* Gets results. */
            i = 1;
            Integer year = resultSet.getInt(i++);
            Calendar period = Calendar.getInstance();
            period.clear();
            period.set(Calendar.YEAR, year);
            Calendar stateDate = Calendar.getInstance();
            stateDate.setTime(resultSet.getTimestamp(i++));
            Calendar startDate = Calendar.getInstance();
            startDate.setTime(resultSet.getTimestamp(i++));
            Calendar finalDate = Calendar.getInstance();
            finalDate.setTime(resultSet.getTimestamp(i++));
            Short state = resultSet.getShort(i++);
            String details = resultSet.getString(i++);

            /* Returns value object. */
            return new HolidaysRequestVO(autoId, loginName, period,
            stateDate, startDate, finalDate, state, details);

        } catch (SQLException e) {
            throw new InternalErrorException(e);
        } finally {
            GeneralOperations.closeResultSet(resultSet);
            GeneralOperations.closeStatement(preparedStatement);
        }
    }

    /**
     * Finds all user holidays requests ordered by propertyName.
     * @param connection
     * @param loginName
     * @param propertyName
     * @param isDesc
     * @param startIndex
     * @param count
     * @throws application.util.exceptions.InternalErrorException
     * @return
     */
    public Collection findUserBy(Connection connection, String
      loginName, String propertyName, Boolean isDesc, Integer
      startIndex, Integer count)throws InternalErrorException{

        PreparedStatement preparedStatement = null;
        ResultSet resultSet = null;

        try {

            /* Validates propertyName. */
            if(!HolidaysRequestTable.validate(propertyName)){
                propertyName = "autoId ";
            }
```

```
      /* Creates statement. */
      String queryString = "SELECT autoId, loginName, period,
      stateDate, " + "startDate, finalDate, state, details
FROM " + "holidays_request_registry WHERE loginName
LIKE \"" + loginName + "\" ORDER BY " + propertyName;

      if(isDesc) {
          queryString = queryString + " DESC";
      }

      preparedStatement =
      connection.prepareStatement(queryString,
              ResultSet.TYPE_SCROLL_INSENSITIVE,
              ResultSet.CONCUR_READ_ONLY);

      int i = 1;

      /* Executes query. */
      resultSet = preparedStatement.executeQuery();

      /* Gets objects. Gets a collection of "count" objects
       * starting in "startIndex" object. If "count" is "0"
       * returns a collection of all objects.*/
      Collection<HolidaysRequestVO> holidaysRequestVOs =
              new ArrayList<HolidaysRequestVO>();

      int currentCount = 0;

      if ((startIndex >=1) && resultSet.absolute(startIndex)) {
          do {
              i = 1;
              Integer autoId = resultSet.getInt(i++);
              loginName = resultSet.getString(i++);
              Integer year = resultSet.getInt(i++);
              Calendar period = Calendar.getInstance();
              period.clear();
              period.set(Calendar.YEAR, year);
              Calendar stateDate = Calendar.getInstance();
              stateDate.setTime(resultSet.getTimestamp(i++));
              Calendar startDate = Calendar.getInstance();
              startDate.setTime(resultSet.getTimestamp(i++));
              Calendar finalDate = Calendar.getInstance();
              finalDate.setTime(resultSet.getTimestamp(i++));
              Short state = resultSet.getShort(i++);
              String details = resultSet.getString(i++);

              holidaysRequestVOs.add(new
            HolidaysRequestVO(autoId,
                        loginName, period, stateDate, startDate,
                  finalDate, state, details));

              currentCount++;

          } while (resultSet.next() && ((currentCount < count)
      || (count == 0)) ) ;
```

```
        }

        /* Returns value objects. */
        return holidaysRequestVOs;

    } catch (SQLException e) {
        throw new InternalErrorException(e);
    } finally {
        GeneralOperations.closeResultSet(resultSet);
        GeneralOperations.closeStatement(preparedStatement);
    }
}

/**
 * Finds all user holidays requests in a given period with a given
 * state ordered by propertyName.
 * @param connection
 * @param loginName
 * @param period
 * @param state
 * @param propetyName
 * @param startIndex
 * @param count
 * @throws application.util.exceptions.InternalErrorException
 * @return
 */
public Collection findUserPeriodStateBy(Connection connection,
        String loginName, Calendar period, Short state, String
  propertyName, Integer startIndex, Integer count) throws
  InternalErrorException {

    PreparedStatement preparedStatement = null;
    ResultSet resultSet = null;

    try {

        /* Creates statement. */
        String queryString = "SELECT autoId, loginName, period,
        stateDate, " + "startDate, finalDate, state, details
  FROM " + "holidays_request_registry WHERE loginName
  LIKE ? AND " + "period = ?";

        if(state >= 4){
            queryString = queryString + " AND state < ?";
        }else{
            queryString = queryString + " AND state = ?";
        }
        if(!HolidaysRequestTable.validate(propertyName)) {
            propertyName = "autoId";
        }

        queryString = queryString + " ORDER BY " + propertyName +
        " DESC";

        preparedStatement =
        connection.prepareStatement(queryString,
                ResultSet.TYPE_SCROLL_INSENSITIVE,
                ResultSet.CONCUR_READ_ONLY);

        int i = 1;
        preparedStatement.setString(i++, loginName);
        preparedStatement.setInt(i++, period.get(Calendar.YEAR));
        preparedStatement.setShort(i++, state);

        /* Executes query. */
        resultSet = preparedStatement.executeQuery();

        /* Gets objects. Gets a collection of "count" objects
         * starting in "startIndex" object. If "count" is "0"
   * returns a collection of all objects.*/
        Collection<HolidaysRequestVO> holidaysRequestVOs =
                new ArrayList<HolidaysRequestVO>();

        int currentCount = 0;

        if ((startIndex >=1) && resultSet.absolute(startIndex)) {

              do {
                i = 1;
                Integer autoId = resultSet.getInt(i++);
                loginName = resultSet.getString(i++);
                period.set(Calendar.YEAR, resultSet.getInt(i++));
                Calendar stateDate = Calendar.getInstance();
                stateDate.setTime(resultSet.getTimestamp(i++));
                Calendar startDate = Calendar.getInstance();
                startDate.setTime(resultSet.getTimestamp(i++));
                Calendar finalDate = Calendar.getInstance();
                finalDate.setTime(resultSet.getTimestamp(i++));
                state = resultSet.getShort(i++);
                String details = resultSet.getString(i++);

                holidaysRequestVOs.add(new
              HolidaysRequestVO(autoId,
                          loginName, period, stateDate, startDate,
                    finalDate, state, details));

                currentCount++;

            } while (resultSet.next() && ((currentCount < count)
        || (count == 0)) ) ;
        }

        /* Returns value objects. */
        return holidaysRequestVOs;

    } catch (SQLException e) {
        throw new InternalErrorException(e);
    } finally {
        GeneralOperations.closeResultSet(resultSet);
        GeneralOperations.closeStatement(preparedStatement);
    }
}
```

```
/**
 * Carries out an advanced search based on loginName, autoId,
 * period, sinceDate, state, order by, ascendant or descendant.
 * @param connection
 * @param loginName
 * @param autoId
 * @param period
 * @param sinceDate
 * @param state
 * @param propertyName
 * @param isDesc
 * @param startIndex
 * @param count
 * @throws application.util.exceptions.InternalErrorException
 * @return
 */
public Collection findAdvanced(Connection connection, String
  loginName, Integer autoId, Calendar period, Calendar
  sinceDate, Short state, String propertyName, Boolean
  isDesc, Integer startIndex, Integer count) throws
  InternalErrorException {

    PreparedStatement preparedStatement = null;
    ResultSet resultSet = null;

    try {

        /* Creates statement. */
        String queryString = "SELECT autoId, loginName, period,
        stateDate, " + "startDate, finalDate, state, details
  FROM " + "holidays_request_registry WHERE loginName
  LIKE \"" + loginName + "\"";

        if(!(autoId == null)) {
            queryString = queryString + " AND autoID = " +autoId;
        }

        if((state != null) && (state >= 0) && (state <=3)) {
            queryString = queryString + " AND state = " + state;
        }

        if(!(period == null)) {
            queryString = queryString + " AND period = " +
              period.get(Calendar.YEAR);
        }

        if(!(sinceDate == null)) {
            queryString = queryString + " AND finalDate >= ?";
        }

        if(!HolidaysRequestTable.validate(propertyName)) {
            propertyName = "autoId";
        }

        queryString = queryString + " ORDER BY " + propertyName;
```

```
if(isDesc) {
    queryString = queryString + " DESC";
}

preparedStatement =
connection.prepareStatement(queryString,
        ResultSet.TYPE_SCROLL_INSENSITIVE,
        ResultSet.CONCUR_READ_ONLY);

int i = 1;
if(!(sinceDate == null)) {
preparedStatement.setTimestamp(i++, new
Timestamp(sinceDate.getTime().getTime()));
}

/* Executes query. */
resultSet = preparedStatement.executeQuery();

/* Gets objects. Gets a collection of "count" objects
 * starting in "startIndex" object. If "count" is "0"
 * returns a collection of all objects.*/
Collection<HolidaysRequestVO> holidaysRequestVOs =
        new ArrayList<HolidaysRequestVO>();

int currentCount = 0;

Calendar newPeriod = Calendar.getInstance();
newPeriod.clear();

if ((startIndex >=1) && resultSet.absolute(startIndex)) {

      do {
        i = 1;
        autoId = resultSet.getInt(i++);
        loginName = resultSet.getString(i++);
        newPeriod.set(Calendar.YEAR,
      resultSet.getInt(i++));
        Calendar stateDate = Calendar.getInstance();
        stateDate.setTime(resultSet.getTimestamp(i++));
        Calendar startDate = Calendar.getInstance();
        startDate.setTime(resultSet.getTimestamp(i++));
        Calendar finalDate = Calendar.getInstance();
        finalDate.setTime(resultSet.getTimestamp(i++));
        state = resultSet.getShort(i++);
        String details = resultSet.getString(i++);

        holidaysRequestVOs.add(new
      HolidaysRequestVO(autoId,
                  loginName, newPeriod, stateDate,
            startDate, finalDate, state, details));

        currentCount++;

    } while (resultSet.next() && ((currentCount < count)
|| (count == 0)) ) ;
```

```
            }

            /* Returns value objects. */
            return holidaysRequestVOs;

        } catch (SQLException e) {
            throw new InternalErrorException(e);
        } finally {
            GeneralOperations.closeResultSet(resultSet);
            GeneralOperations.closeStatement(preparedStatement);
        }
    }

    /**
     * Returns all periods with at least a holidays request of any
     * user.
     * @param connection
     * @param startIndex
     * @param count
     * @throws application.util.exceptions.InternalErrorException
     * @return
     */
    public Collection findPeriods(Connection connection, Integer
      startIndex, Integer count) throws InternalErrorException {

        PreparedStatement preparedStatement = null;
        ResultSet resultSet = null;

        try {

            /* Creates statement. */
            String queryString = "SELECT DISTINCT period" +
                  " FROM holidays_request_registry ORDER BY period
           DESC";

            preparedStatement =
           connection.prepareStatement(queryString,
                    ResultSet.TYPE_SCROLL_INSENSITIVE,
                    ResultSet.CONCUR_READ_ONLY);

            /* Executes query. */
            resultSet = preparedStatement.executeQuery();

            /* Gets objects. Gets a collection of "count" objects
             * starting in "startIndex" object. If "count" is "0"
       * returns a collection of all objects.*/
            Collection<Calendar> periods =
                    new ArrayList<Calendar>();
            int currentCount = 0;

            int i;

            if ((startIndex >=1) && resultSet.absolute(startIndex)) {
                do {

                    i = 1;
                    Integer year = resultSet.getInt(i++);

                    Calendar period = Calendar.getInstance();
                    period.clear();
                    period.set(Calendar.YEAR, year);

                    periods.add(period);

                    currentCount++;

                } while (resultSet.next() && ((currentCount < count)
            || (count == 0)) ) ;
            }

            /* Returns value objects. */
            return periods;

        } catch (SQLException e) {
            throw new InternalErrorException(e);
        } finally {
            GeneralOperations.closeResultSet(resultSet);
            GeneralOperations.closeStatement(preparedStatement);
        }
    }

}
```

10.5.3.3 Capa de fachadas de servicios

10.5.3.3.a Servicio de perfil de usuario

10.5.3.3.a.1 application.model.userprofileservice.facade.vo

UserFacadeVO

```
package application.model.userprofileservice.facade.vo;

import java.io.Serializable;

/**
 * This class is used for interacting with user profile service.
 * <BR>
 * LoginName and administrator access permission can be usefull values
 * from {@link UserProfileVO} for loading in user session after
 * logging.
 * Only getter methods are provided for accessing login result
 * properties.
 */
public class UserFacadeVO implements Serializable {

    /**
     * Hold values of properties.
     */
    private String loginName;
    private Boolean isAdministrator;

    /**
     * Construct VO with the given user login result.
     * @param loginName
     * @param isAdministrator
     */
    public UserFacadeVO(String loginName, Boolean isAdministrator) {

            this.loginName = loginName;
            this.isAdministrator = isAdministrator;
    }

    /**
     * Returns user identifier.
     * @return loginName
     */
    public String getLoginName() {
            return loginName;
    }

    /**
     * Returns administrator access permission.
     * @return administrator access permission.
     */
    public Boolean getIsAdministrator() {
            return isAdministrator;
    }

    /**
     * Returns a <CODE>String</CODE> object representing this VO's
     * value. Useful for debugging.
     * @return a string representation of this object.
     */
    public String toString() {
            return new String("loginName = " + loginName + " | "
                    + "isAdministrator = " + isAdministrator);
    }

}
```

10.5.3.3.a.2 application.model.userprofileservice.facade.actions

LoginAction

```
package application.model.userprofileservice.facade.actions;

/* JDBC. */
import java.sql.Connection;

import application.util.exceptions.IncorrectPasswordException;
import application.util.exceptions.InstanceNotFoundException;
import application.util.exceptions.InternalErrorException;

import application.util.sql.NonTransactionalPlainAction;

import application.model.userprofileservice.dao.SQLUserProfileDAO;
import
application.model.userprofileservice.dao.SQLUserProfileDAOFactory;

import application.model.userprofileservice.vo.UserProfileVO;

import application.model.userprofileservice.facade.vo.UserFacadeVO;

import
application.model.userprofileservice.operations.PasswordEncrypter;

/**
 * This non transactional action finds an user an checks if his
 * password is the password given.
 */
public class LoginAction implements NonTransactionalPlainAction {

    private String loginName;
    private String encryptedPassword;

    /**
     *
     * @param loginName
     * @param password
     * @param passwordIsEncrypted
     */
    public LoginAction(String loginName, String password,
            Boolean passwordIsEncrypted) {

        this.loginName = loginName;

        /* Gets password encrypted. */
        this.encryptedPassword =
                getPasswordEncrypted(password, passwordIsEncrypted);
    }

    /**
     *
     * @param connection
     * @throws application.util.exceptions.IncorrectPasswordException
     * @throws application.util.exceptions.InstanceNotFoundException
     * @throws application.util.exceptions.InternalErrorException
     * @return
     */
    public Object execute(Connection connection) throws
            IncorrectPasswordException, InstanceNotFoundException,
            InternalErrorException {

        SQLUserProfileDAO userProfileDAO = (SQLUserProfileDAO)
                SQLUserProfileDAOFactory.getDAO();

        /* Finds user profile. */
        UserProfileVO userProfileVO =
                userProfileDAO.find(connection, loginName);

        /* Checks password. */
        if (!userProfileVO.getEnPassword().equals(encryptedPassword))
{
            throw new IncorrectPasswordException(loginName);
        }

        return new UserFacadeVO(userProfileVO.getLoginName(),
         userProfileVO.getUserProfileAccessVO().getIsAdministrator());

    }

    private String getPasswordEncrypted(String password,
            Boolean passwordIsEncrypted) {

        if(passwordIsEncrypted){
            return password;
        }else{
            return PasswordEncrypter.crypt(password);
        }
    }

}
```

<u>RegisterUserAction</u>

```
package application.model.userprofileservice.facade.actions;

/* JDBC. */
import java.sql.Connection;

import application.util.exceptions.DuplicateInstanceException;
import application.util.exceptions.InternalErrorException;

import application.util.sql.TransactionalPlainAction;

import application.model.userprofileservice.dao.SQLUserProfileDAO;
import
application.model.userprofileservice.dao.SQLUserProfileDAOFactory;
```

```
import application.model.userprofileservice.vo.UserProfileVO;
import application.model.userprofileservice.vo.UserProfileDetailsVO;
import application.model.userprofileservice.vo.UserProfileAccessVO;

import
application.model.userprofileservice.operations.PasswordEncrypter;

/**
 * This transactional action creates a new user profile.
 */
public class RegisterUserAction implements TransactionalPlainAction {

    private String loginName;
    private String encryptedPassword;
    private UserProfileDetailsVO userProfileDetailsVO;
    private String phone;
    private UserProfileAccessVO userProfileAccessVO;

    /**
     *
     * @param loginName
     * @param clearPassword
     * @param userProfileDetailsVO
     * @param phone
     * @param userProfileAccessVO
     */
    public RegisterUserAction(String loginName, String clearPassword,
            UserProfileDetailsVO userProfileDetailsVO, String phone,
            UserProfileAccessVO userProfileAccessVO) {

        this.loginName = loginName;

        /* Gets password encrypted. */
        this.encryptedPassword =
      PasswordEncrypter.crypt(clearPassword);

        this.userProfileDetailsVO = userProfileDetailsVO;
        this.phone = phone;
        this.userProfileAccessVO = userProfileAccessVO;

    }

    /**
     *
     * @param connection
     * @throws application.util.exceptions.DuplicateInstanceException
     * @throws application.util.exceptions.InternalErrorException
     * @return
     */
    public Object execute(Connection connection) throws
            DuplicateInstanceException, InternalErrorException {

        SQLUserProfileDAO userProfileDAO = (SQLUserProfileDAO)
                SQLUserProfileDAOFactory.getDAO();

        UserProfileVO userProfileVO = new UserProfileVO(loginName,
                encryptedPassword, userProfileDetailsVO,  phone,
                userProfileAccessVO);

        /* Creates new user. */
        userProfileDAO.create(connection, userProfileVO);

        return null;
    }

}
```

UpdateUserProfileDetailsAction

```
package application.model.userprofileservice.facade.actions;

/* JDBC. */
import java.sql.Connection;

import application.util.exceptions.InternalErrorException;
import application.util.exceptions.InstanceNotFoundException;

import application.util.sql.TransactionalPlainAction;

import application.model.userprofileservice.dao.SQLUserProfileDAO;
import
application.model.userprofileservice.dao.SQLUserProfileDAOFactory;

import application.model.userprofileservice.vo.UserProfileVO;
import application.model.userprofileservice.vo.UserProfileDetailsVO;

/**
 * This transactional action updates an user profile details.
 */
public class UpdateUserProfileDetailsAction implements
      TransactionalPlainAction {

      private String loginName;
      private UserProfileDetailsVO userProfileDetailsVO;

      public UpdateUserProfileDetailsAction(String loginName,
            UserProfileDetailsVO userProfileDetailsVO) {

            this.loginName = loginName;
            this.userProfileDetailsVO = userProfileDetailsVO;
        }

      /**
       *
       * @param connection
       * @throws application.util.exceptions.InstanceNotFoundException
```

```
	 * @throws application.util.exceptions.InternalErrorException
	 * @return
	 */
	public Object execute(Connection connection) throws
			InstanceNotFoundException, InternalErrorException {

		SQLUserProfileDAO userProfileDAO = (SQLUserProfileDAO)
					SQLUserProfileDAOFactory.getDAO();

			/* Finds an user profile. */
		UserProfileVO userProfileVO =
		userProfileDAO.find(connection, loginName);

			/* Changes user profile data. */
	       userProfileVO.setUserProfileDetailsVO(userProfileDetailsVO);

			/* Updates user profile. */
		userProfileDAO.update(connection, userProfileVO);

		return null;

	}

}
```

UpdateUserProfileAccessAction

```
package application.model.userprofileservice.facade.actions;

/* JDBC. */
import java.sql.Connection;

import application.util.exceptions.InternalErrorException;
import application.util.exceptions.InstanceNotFoundException;

import application.util.sql.TransactionalPlainAction;

import application.model.userprofileservice.dao.SQLUserProfileDAO;
import
application.model.userprofileservice.dao.SQLUserProfileDAOFactory;

import application.model.userprofileservice.vo.UserProfileVO;
import application.model.userprofileservice.vo.UserProfileAccessVO;

/**
 * This transactional action updates an user profile access details.
 */
public class UpdateUserProfileAccessAction implements
	TransactionalPlainAction {

	private String loginName;
	private UserProfileAccessVO userProfileAccessVO;
```

```
      /**
       *
       * @param loginName
       * @param userProfileAccessVO
       */
      public UpdateUserProfileAccessAction(String loginName,
                UserProfileAccessVO userProfileAccessVO) {

            this.loginName = loginName;
            this.userProfileAccessVO = userProfileAccessVO;
        }

      /**
       *
       * @param connection
       * @throws application.util.exceptions.InstanceNotFoundException
       * @throws application.util.exceptions.InternalErrorException
       * @return
       */
      public Object execute(Connection connection) throws
                InstanceNotFoundException, InternalErrorException {

            SQLUserProfileDAO userProfileDAO = (SQLUserProfileDAO)
                        SQLUserProfileDAOFactory.getDAO();

                /* Finds an user profile. */
            UserProfileVO userProfileVO =
            userProfileDAO.find(connection, loginName);

                /* Changes user profile data. */
            userProfileVO.setUserProfileAccessVO(userProfileAccessVO);

                /* Updates user profile. */
            userProfileDAO.update(connection, userProfileVO);

            return null;

      }

}
```

ChangePasswordAction

```
package application.model.userprofileservice.facade.actions;

/* JDBC. */
import java.sql.Connection;

import application.util.exceptions.InternalErrorException;
import application.util.exceptions.InstanceNotFoundException;
import application.util.exceptions.IncorrectPasswordException;

import application.util.sql.TransactionalPlainAction;
import application.model.userprofileservice.dao.SQLUserProfileDAO;
import
application.model.userprofileservice.dao.SQLUserProfileDAOFactory;

import application.model.userprofileservice.vo.UserProfileVO;

import
application.model.userprofileservice.operations.PasswordEncrypter;

/**
 * This transactional actions changes user password. Old password is
 * needed.
 */
public class ChangePasswordAction implements TransactionalPlainAction
      {

    private String loginName;
    private String oldEncryptedPassword;
    private String newEncryptedPassword;

    /**
     *
     * @param loginName
     * @param oldPassword
     * @param oldPasswordIsEncrypted
     * @param newClearPassword
     */
    public ChangePasswordAction(String loginName, String oldPassword,
            Boolean oldPasswordIsEncrypted, String newClearPassword) {

        this.loginName = loginName;

        /* Gets passwords encrypted. */
        this.oldEncryptedPassword =
                getPasswordEncrypted(oldPassword,
            oldPasswordIsEncrypted);
        this.newEncryptedPassword =
                PasswordEncrypter.crypt(newClearPassword);

    }

    /**
     *
     * @param connection
     * @throws application.util.exceptions.InstanceNotFoundException
     * @throws application.util.exceptions.IncorrectPasswordException
     * @throws application.util.exceptions.InternalErrorException
     * @return
     */
    public Object execute(Connection connection) throws
            InstanceNotFoundException, IncorrectPasswordException,
            InternalErrorException {

            SQLUserProfileDAO userProfileDAO = (SQLUserProfileDAO)
                    SQLUserProfileDAOFactory.getDAO();
            /* Finds user profile. */
            UserProfileVO userProfileVO =
            userProfileDAO.find(connection,
                    loginName);

            /* Checks old password. */
      if(!userProfileVO.getEnPassword().equals(oldEncryptedPassword)){
                throw new IncorrectPasswordException(loginName);
            }

            /* Sets new password. */
            userProfileVO.setEnPassword(newEncryptedPassword);

            /* Updates user profile. */
            userProfileDAO.update(connection, userProfileVO);

            return null;

    }

    private String getPasswordEncrypted(String password,
        Boolean passwordIsEncrypted) {

        if(passwordIsEncrypted){
            return password;
        }else{
            return PasswordEncrypter.crypt(password);
        }
    }

}
```

GivePasswordAction

```
package application.model.userprofileservice.facade.actions;

/* JDBC. */
import java.sql.Connection;

import application.util.exceptions.InternalErrorException;
import application.util.exceptions.InstanceNotFoundException;
import application.util.exceptions.IncorrectPasswordException;

import application.util.sql.TransactionalPlainAction;

import application.model.userprofileservice.dao.SQLUserProfileDAO;
import
application.model.userprofileservice.dao.SQLUserProfileDAOFactory;

import application.model.userprofileservice.vo.UserProfileVO;

import
application.model.userprofileservice.operations.PasswordEncrypter;
```

```
/**
 * This transactional actions changes user password.
 */
public class GivePasswordAction implements TransactionalPlainAction {

    private String loginName;
    private String newEncryptedPassword;

    /**
     *
     * @param loginName
     * @param newPassword
     * @param passwordIsEncrypted
     */
    public GivePasswordAction(String loginName, String newPassword,
            Boolean passwordIsEncrypted) {

        this.loginName = loginName;

        /* Get passwords encrypted. */
        this.newEncryptedPassword =
                getPasswordEncrypted(newPassword,
            passwordIsEncrypted);

    }

    /**
     *
     * @param connection
     * @throws application.util.exceptions.InstanceNotFoundException
     * @throws application.util.exceptions.IncorrectPasswordException
     * @throws application.util.exceptions.InternalErrorException
     * @return
     */
    public Object execute(Connection connection) throws
            InstanceNotFoundException, IncorrectPasswordException,
            InternalErrorException {

            SQLUserProfileDAO userProfileDAO = (SQLUserProfileDAO)
                    SQLUserProfileDAOFactory.getDAO();

            /* Finds user profile. */
            UserProfileVO userProfileVO =
            userProfileDAO.find(connection,
                    loginName);

            /* Sets new password. */
            userProfileVO.setEnPassword(newEncryptedPassword);

            /* Updates user profile. */
            userProfileDAO.update(connection, userProfileVO);

            return null;

    }
    private String getPasswordEncrypted(String password,
        Boolean passwordIsEncrypted) {

        if(passwordIsEncrypted){
            return password;
        }else{
            return PasswordEncrypter.crypt(password);
        }
    }

}
```

ChangePhoneAction

```
package application.model.userprofileservice.facade.actions;

/* JDBC. */
import java.sql.Connection;

import application.util.exceptions.InternalErrorException;
import application.util.exceptions.InstanceNotFoundException;
import application.util.exceptions.IncorrectPasswordException;

import application.util.sql.TransactionalPlainAction;

import application.model.userprofileservice.dao.SQLUserProfileDAO;
import
application.model.userprofileservice.dao.SQLUserProfileDAOFactory;

import application.model.userprofileservice.vo.UserProfileVO;

import
application.model.userprofileservice.operations.PasswordEncrypter;

/**
 * This transactional actions changes user phone. As phone will be
 * used for important and individual notifications, password is need
 * to be provided.
 */
public class ChangePhoneAction implements TransactionalPlainAction {

    private String loginName;
    private String encryptedPassword;
    private String newPhone;

    /**
     *
     * @param loginName
     * @param password
     * @param passwordIsEncrypted
     * @param newPhone
     */
    public ChangePhoneAction(String loginName, String password,
            Boolean passwordIsEncrypted, String newPhone) {

        this.loginName = loginName;

        /* Gets password encrypted. */
        this.encryptedPassword =
                getPasswordEncrypted(password, passwordIsEncrypted);

        this.newPhone = newPhone;

    }

    /**
     *
     * @param connection
     * @throws application.util.exceptions.InstanceNotFoundException
     * @throws application.util.exceptions.IncorrectPasswordException
     * @throws application.util.exceptions.InternalErrorException
     * @return
     */
    public Object execute(Connection connection) throws
            InstanceNotFoundException, IncorrectPasswordException,
            InternalErrorException {

            SQLUserProfileDAO userProfileDAO = (SQLUserProfileDAO)
                    SQLUserProfileDAOFactory.getDAO();

            /* Finds user profile. */
            UserProfileVO userProfileVO =
            userProfileDAO.find(connection,
                    loginName);

            /* Checks password. */
         if(!userProfileVO.getEnPassword().equals(encryptedPassword)){
                throw new IncorrectPasswordException(loginName);
            }

            /* Changes phone. */
            userProfileVO.setPhone(newPhone);

             /* Updates user profile. */
            userProfileDAO.update(connection, userProfileVO);

            return null;

    }

    private String getPasswordEncrypted(String password,
        Boolean passwordIsEncrypted) {

        if(passwordIsEncrypted){
            return password;
        }else{
            return PasswordEncrypter.crypt(password);
        }
    }
```

```
}
```

GivePhoneAction

```
package application.model.userprofileservice.facade.actions;

/* JDBC. */
import java.sql.Connection;

import application.util.exceptions.InternalErrorException;
import application.util.exceptions.InstanceNotFoundException;

import application.util.sql.TransactionalPlainAction;

import application.model.userprofileservice.dao.SQLUserProfileDAO;
import
application.model.userprofileservice.dao.SQLUserProfileDAOFactory;

import application.model.userprofileservice.vo.UserProfileVO;

/**
 * This transactional actions changes user phone detail.
 */
public class GivePhoneAction implements TransactionalPlainAction {

    private String loginName;
    private String phone;

    /**
     *
     * @param loginName
     * @param newPhone
     */
    public GivePhoneAction(String loginName, String newPhone) {

        this.loginName = loginName;
        this.phone = newPhone;

    }

    /**
     *
     * @param connection
     * @throws application.util.exceptions.InstanceNotFoundException
     * @throws application.util.exceptions.InternalErrorException
     * @return
     */
    public Object execute(Connection connection) throws
            InstanceNotFoundException, InternalErrorException {

            SQLUserProfileDAO userProfileDAO = (SQLUserProfileDAO)
                    SQLUserProfileDAOFactory.getDAO();
            /* Finds user profile. */
            UserProfileVO userProfileVO =
            userProfileDAO.find(connection,
                    loginName);

            /* Sets new phone. */
            userProfileVO.setPhone(phone);

            /* Updates user profile. */
            userProfileDAO.update(connection, userProfileVO);

            return null;

    }
}
```

FindUserProfileAction

```
package application.model.userprofileservice.facade.actions;

/* JDBC.*/
import java.sql.Connection;

import application.util.exceptions.InternalErrorException;
import application.util.exceptions.InstanceNotFoundException;

import application.util.sql.NonTransactionalPlainAction;

import application.model.userprofileservice.dao.SQLUserProfileDAO;
import
application.model.userprofileservice.dao.SQLUserProfileDAOFactory;

import application.model.userprofileservice.vo.UserProfileVO;

/**
 * This non transactional action finds an user profile.
 */
public class FindUserProfileAction implements
      NonTransactionalPlainAction {

      private String loginName;

      /**
       *
       * @param loginName
       */
      public FindUserProfileAction(String loginName) {
            this.loginName = loginName;
      }

      /**
       *
       * @param connection
       * @throws application.util.exceptions.InstanceNotFoundException
       * @throws application.util.exceptions.InternalErrorException
       * @return
       */
      public Object execute(Connection connection) throws
                InstanceNotFoundException, InternalErrorException {

            SQLUserProfileDAO userProfileDAO = (SQLUserProfileDAO)
                    SQLUserProfileDAOFactory.getDAO();

            /* Finds user profile. */
            UserProfileVO userProfileVO =
                    userProfileDAO.find(connection, loginName);

            userProfileVO.setEnPassword("Not avaliable");

            return userProfileVO;

      }

}
```

<u>FindByAction</u>

```
package application.model.userprofileservice.facade.actions;

/* JDBC. */
import java.sql.Connection;

import application.util.exceptions.InternalErrorException;
import application.util.exceptions.InstanceNotFoundException;

import application.util.sql.NonTransactionalPlainAction;

import application.model.userprofileservice.dao.SQLUserProfileDAO;
import
application.model.userprofileservice.dao.SQLUserProfileDAOFactory;

/**
 * This non transactional action finds all user profiles order by
 * specified property. It uses page-by-page Iterator getting block of
 * data using an index for getting more data.
 */
public class FindByAction implements NonTransactionalPlainAction {

    private String propertyOrderName;
    private Integer startIndex;
    private Integer count;

    /**
     *
     * @param propertyOrderName
     * @param startIndex
     * @param count
     */
    public FindByAction(String propertyOrderName, Integer startIndex,
            Integer count) {

            this.propertyOrderName = propertyOrderName;
            this.startIndex= startIndex;
            this.count = count;
    }

    /**
     *
     * @param connection
     * @throws application.util.exceptions.InternalErrorException
     * @return
     */
    public Object execute(Connection connection) throws
            InternalErrorException {

        SQLUserProfileDAO userProfileDAO = (SQLUserProfileDAO)
                SQLUserProfileDAOFactory.getDAO();

        return userProfileDAO.findBy(connection, propertyOrderName,
                    startIndex,count);

    }

}
```

FindInByAction

```
package application.model.userprofileservice.facade.actions;

/* JDBC. */
import java.sql.Connection;

import application.util.exceptions.InternalErrorException;

import application.util.sql.NonTransactionalPlainAction;

import application.model.userprofileservice.dao.SQLUserProfileDAO;
import
application.model.userprofileservice.dao.SQLUserProfileDAOFactory;

/**
 * This non transactional action carries out an advanced search in
 * user profiles.
 */
public class FindInByAction implements NonTransactionalPlainAction {

    private String propertyInName;
    private String propertyInValue;
    private String propertyByName;
    private Integer startIndex;
    private Integer count;
    /**
     *
     * @param propertyInName
     * @param propertyInValue
     * @param propertyOrderName
     * @param startIndex
     * @param count
     */
    public FindInByAction(String propertyInName, String
      propertyInValue, String propertyOrderName, Integer
      startIndex, Integer count) {

            this.propertyInName = propertyInName;
            this.propertyInValue = propertyInValue;
            this.propertyByName = propertyOrderName;
            this.startIndex = startIndex;
            this.count = count;

    }

    /**
     *
     * @param connection
     * @throws application.util.exceptions.InternalErrorException
     * @return
     */
    public Object execute(Connection connection) throws
            InternalErrorException {

        SQLUserProfileDAO userProfileDAO = (SQLUserProfileDAO)
                SQLUserProfileDAOFactory.getDAO();

        return userProfileDAO.findInBy(connection, propertyInName,
                        propertyInValue, propertyByName, startIndex,
                  count);

    }

}
```

10.5.3.3.a.3 application.model.userprofileservice.facade.delegate

UserProfileServiceFacade

```
package application.model.userprofileservice.facade.delegate;
```

```
import java.io.Serializable;

import java.util.Collection;

import application.util.exceptions.AdministratorRequiredException;
import application.util.exceptions.DuplicateInstanceException;
import application.util.exceptions.InstanceNotFoundException;
import application.util.exceptions.IncorrectPasswordException;
import application.util.exceptions.InternalErrorException;

import application.model.userprofileservice.facade.vo.UserFacadeVO;

import application.model.userprofileservice.vo.UserProfileVO;
import application.model.userprofileservice.vo.UserProfileDetailsVO;
import application.model.userprofileservice.vo.UserProfileAccessVO;

/**
 * A facade to model the interaction of the user with the service.
 * There is some logical restrictions with regard to the order of
 * method calling.
 */
public interface UserProfileServiceFacade extends Serializable {

    /**
     * An user logs in and his facade is initializated.
     * @param loginName
     * @param password
     * @param passwordIsEncrypted
     * @throws application.util.exceptions.InstanceNotFoundException
     * @throws application.util.exceptions.IncorrectPasswordException
     * @throws application.util.exceptions.InternalErrorException
     * @return
     */
    public UserFacadeVO login(String loginName, String password,
            Boolean passwordIsEncrypted) throws
      InstanceNotFoundException, IncorrectPasswordException,
      InternalErrorException;

    /**
     * Updates user profile details of the user load in the facade.
     * @param userProfileDetailsVO
     * @throws application.util.exceptions.InternalErrorException
     */
    public void updateUserProfileDetails(
            UserProfileDetailsVO userProfileDetailsVO) throws
            InternalErrorException;

    /**
     * Changes the password of the user load in the facade. Old
     * password is needed for this action.
     * @param oldPassword
     * @param newClearPassword
     * @param oldPasswordIsEncrypted
     * @throws application.util.exceptions.IncorrectPasswordException
     * @throws application.util.exceptions.InternalErrorException
     */
    public void changePassword(String oldPassword, String
      newClearPassword, Boolean oldPasswordIsEncrypted) throws
      IncorrectPasswordException,
            InternalErrorException;

     /**
     * Changes phone details of the user loaded in the facade. As
     * phone will be used for important and individual notifications,
     * password is needed to be provided.
     * @param password
     * @param newPhone
     * @param passwordIsEncrypted
     * @throws application.util.exceptions.IncorrectPasswordException
     * @throws application.util.exceptions.InternalErrorException
     */
    public void changePhone(String password, String newPhone,
            Boolean passwordIsEncrypted) throws
      IncorrectPasswordException, InternalErrorException;


//FINDING ACTIONS

    /**
     * Finds the profile of the user loaded in the facade.
     * @throws application.util.exceptions.InternalErrorException
     * @return
     */
    public UserProfileVO findUserProfile() throws
      InternalErrorException;

    /**
     * Finds a pattern in an user profile and shows results ordered by
     * propertyName.
     * @param propertyInName
     * @param propertyInValue
     * @param propertyByName
     * @param startIndex
     * @param count
     * @throws application.util.exceptions.InternalErrorException
     * @return
     */
    public Collection findInBy(String propertyInName, String
      propertyInValue, String propertyByName, Integer startIndex,
      Integer count) throws InternalErrorException;

//ADMINISTRATOR ACTIONS

    /**
     * Register a new user. This user will access the system and
     * userprofile services. When a new user is registered his façade
     * is inicializiated.
     * @param userName
     * @param clearPassword
     * @param userProfileDetailsVO
     * @param phone
     * @param userProfileAccessVO
     * @throws
     * application.util.exceptions.AdministratorRequiredException
     * @throws application.util.exceptions.DuplicateInstanceException
     * @throws application.util.exceptions.InternalErrorException
     */
    public void registerUser(String userName, String clearPassword,
            UserProfileDetailsVO userProfileDetailsVO, String phone,
            UserProfileAccessVO userProfileAccessVO) throws
            AdministratorRequiredException,
      DuplicateInstanceException, InternalErrorException;

    /**
     * Updates the access profile of the user load in the facade.
     * @param userProfileAccessVO
     * @throws
     * application.util.exceptions.AdministratorRequiredException
     * @throws application.util.exceptions.InternalErrorException
     */
    public void updateUserProfileAccess(UserProfileAccessVO
      userProfileAccessVO) throws AdministratorRequiredException,
      InternalErrorException;

    /**
     *
     * Give a new user password. Administrator password is needed for
     * this action.
     * @param PasswordIsEncrypted
     * @param password
     * @param newClearPassword
     * @throws
     * application.util.exceptions.AdministratorRequiredException
     * @throws application.util.exceptions.IncorrectPasswordException
     * @throws application.util.exceptions.InternalErrorException
     */
    public void givePassword(String password, String newClearPassword,
            Boolean PasswordIsEncrypted) throws
      AdministratorRequiredException,
            IncorrectPasswordException, InternalErrorException;

    /**
     * Gives a new phone detail for the user loaded in facade. As
     * phone will be used for important and individual notifications,
     * password is need to be provided.
     * @param password
     * @param newPhone
     * @param passwordIsEncrypted
     * @throws
application.util.exceptions.AdministratorRequiredException
     * @throws application.util.exceptions.IncorrectPasswordException
     * @throws application.util.exceptions.InternalErrorException
     */
    public void givePhone(String password, String newPhone,
            Boolean passwordIsEncrypted) throws
      AdministratorRequiredException,
            IncorrectPasswordException, InternalErrorException;
    /**
     * Loads other user in facade.
     * @param password
     * @param loginName
     * @param passwordIsEncrypted
     * @throws
     * application.util.exceptions.AdministratorRequiredException
     * @throws application.util.exceptions.IncorrectPasswordException
     * @throws application.util.exceptions.InstanceNotFoundException
     * @throws application.util.exceptions.InternalErrorException
     */
    public void loginAsUser(String password, String loginName,
            Boolean passwordIsEncrypted) throws
      AdministratorRequiredException,
            IncorrectPasswordException, InstanceNotFoundException,
            InternalErrorException;

    /**
     * Finds an user profile.
     * @return
     * @param loginName
     * @throws
     * application.util.exceptions.AdministratorRequiredException
     * @throws application.util.exceptions.InstanceNotFoundException
     * @throws application.util.exceptions.InternalErrorException
     */
    public UserProfileVO findUserProfile(String loginName) throws
            AdministratorRequiredException, InstanceNotFoundException,
            InternalErrorException;

//UTILITIES

    /**
     * Returns the loginName of the main user loaded in the facade.
     * @return
     */
    public String getMainLoginName();

    /**
     * Returns the loginName of the secondary user loaded in the
     * facade.
     * @return
     */
    public String getLoginName();

    /**
     * Returns if the main user is administrator or not.
     * @return
     */
    public Boolean getIsAdministrator();

}
```

<u>UserProfileServiceFacadeFactory</u>

```
package application.model.userprofileservice.facade.delegate;

/* Uses JNDI. */
import application.util.jndi.ConfigurationParametersManager;

import application.util.exceptions.InternalErrorException;

/**
 * This class creates an intance of the implementation of
 * <code>UserProfileServiceFacade</code>.
 * <BR>
 * Upper layers use this class to determine which implementation must
 * be used. This makes code independent from each façade
 * implementation.
 * <BR>
 * <code>UserProfileServiceFacadeFactory/facadeClassName</code>
 * parameter has to be defined using JNDI. This parameter can be
 * defined in <code>web.xml</code> with <code><env-entry></code> tags.
 * <BR>
 * This code provide an implementation of
 * <code>UserProfileServiceFacade</code>
 * for not distributed, but plain application in
 * <code>PlainUserProfileServiceFacade</code>.
 */
public final class UserProfileServiceFacadeFactory {

    /* This parameter has to be define in a JNDI context. */
    private static String CLASS_NAME_PARAMETER =
            "UserProfileServiceFacadeFactory/facadeClassName";

    private static Class facadeClass = getFacadeClass();

    private UserProfileServiceFacadeFactory(){};

    /* Gets the name of the implementation class of Facade. */
    private static Class getFacadeClass() {

        Class theClass = null;
        try {
            String className =
            ConfigurationParametersManager.getParameter(
                    CLASS_NAME_PARAMETER);
            theClass = Class.forName(className);
        } catch (Exception e) {
            e.printStackTrace();
        }

        return theClass;
    }

    /**
     * Returns an instance of the implementation class of Facade.
     * @throws application.util.exceptions.InternalErrorException
     * @return implementation of facade
     */
    public static UserProfileServiceFacade getFacade() throws
            InternalErrorException {

        try {
            return (UserProfileServiceFacade)
            facadeClass.newInstance();
        } catch (Exception e) {
            throw new InternalErrorException(e);
        }
    }

}
```

PlainUserProfileServiceFacade

```
package application.model.userprofileservice.facade.delegate;

import java.util.Collection;

/* JDBC. */
import javax.sql.DataSource;

import application.util.exceptions.AdministratorRequiredException;
import application.util.exceptions.DuplicateInstanceException;
import application.util.exceptions.InstanceNotFoundException;
import application.util.exceptions.IncorrectPasswordException;
import application.util.exceptions.InternalErrorException;

import application.util.sql.DataSourceLocator;
import application.util.sql.PlainActionProcessor;

import application.model.util.GlobalNames;

import application.model.userprofileservice.facade.delegate.
UserProfileServiceFacade;

import
application.model.userprofileservice.facade.actions.LoginAction;
import application.model.userprofileservice.facade.actions.
RegisterUserAction;
import application.model.userprofileservice.facade.actions.
FindUserProfileAction;
import application.model.userprofileservice.facade.actions.
UpdateUserProfileDetailsAction;
import application.model.userprofileservice.facade.actions.
UpdateUserProfileAccessAction;
import application.model.userprofileservice.facade.actions.
ChangePasswordAction;
import application.model.userprofileservice.facade.actions.
GivePasswordAction;
import
application.model.userprofileservice.facade.actions.ChangePhoneAction;
```

```
import
application.model.userprofileservice.facade.actions.GivePhoneAction;
import
application.model.userprofileservice.facade.actions.FindByAction;
import
application.model.userprofileservice.facade.actions.FindInByAction;

import application.model.userprofileservice.facade.vo.UserFacadeVO;

import application.model.userprofileservice.vo.UserProfileVO;
import application.model.userprofileservice.vo.UserProfileDetailsVO;
import application.model.userprofileservice.vo.UserProfileAccessVO;

/**
 * This class provides an implementation of
 * <code>UserProfileServiceFacade</code> for interacting with user
 * profile service in facade layer. This implementation is for a not
 * EJB but JDBC application.
 */
public class PlainUserProfileServiceFacade implements
UserProfileServiceFacade {

    /* Holds the value of the main user loaded in facade. */
    private String mainLoginName;

    /* Holds value of the user load in facade. */
    private String loginName;

    /* Holds if the main user is administrator or not. */
    private Boolean isAdministrator;

    /**
     * Constructs Facade.
     */
    public PlainUserProfileServiceFacade() {
        mainLoginName = null;
        loginName = null;
        isAdministrator = false;
    }

    /**
     * An user logs in and his facade is initializated.
     * @param loginName
     * @param password
     * @param passwordIsEncrypted
     * @throws application.util.exceptions.InstanceNotFoundException
     * @throws application.util.exceptions.IncorrectPasswordException
     * @throws application.util.exceptions.InternalErrorException
     * @return
     */
    public UserFacadeVO login(String loginName, String password,
            Boolean passwordIsEncrypted) throws
      InstanceNotFoundException,
            IncorrectPasswordException, InternalErrorException {

        try {
            LoginAction action = new LoginAction(loginName, password,
                    passwordIsEncrypted);

            UserFacadeVO userFacadeVO = (UserFacadeVO)
                    PlainActionProcessor.process(getDataSource(),
                  action);

            this.mainLoginName = loginName;
            this.loginName = loginName;
            this.isAdministrator = userFacadeVO.getIsAdministrator();

            return userFacadeVO;

        } catch (IncorrectPasswordException e) {
            throw e;

        } catch (InstanceNotFoundException e) {
            throw e;

        } catch (InternalErrorException e) {
            throw e;

        } catch (Exception e) {
            throw new InternalErrorException(e);
        }

    }

    /**
     * Updates user profile details of the user load in the facade.
     * @param userProfileDetailsVO
     * @throws application.util.exceptions.InternalErrorException
     */
    public void updateUserProfileDetails(
            UserProfileDetailsVO userProfileDetailsVO) throws
            InternalErrorException {

    try {

        UpdateUserProfileDetailsAction action =
                new UpdateUserProfileDetailsAction(loginName,
                userProfileDetailsVO);

        PlainActionProcessor.process(getDataSource(), action);

        } catch (InstanceNotFoundException e) {
            throw new InternalErrorException(e);
        } catch (InternalErrorException e) {
            throw e;
        } catch (Exception e) {
            throw new InternalErrorException(e);
        }
    }

    /**
```

```
     * Changes the password of the user load in the facade. Old
     * password is needed for this action.
     * @param oldPassword
     * @param newClearPassword
     * @param oldPasswordIsEncrypted
     * @throws application.util.exceptions.IncorrectPasswordException
     * @throws application.util.exceptions.InternalErrorException
     */
    public void changePassword(String oldPassword, String
      newClearPassword, Boolean oldPasswordIsEncrypted) throws
      IncorrectPasswordException,
            InternalErrorException {

        try {

            ChangePasswordAction action = new ChangePasswordAction(
                    loginName, oldPassword, oldPasswordIsEncrypted,
                    newClearPassword);

            PlainActionProcessor.process(getDataSource(), action);

        } catch (InstanceNotFoundException e) {
            throw new InternalErrorException(e);
        } catch (IncorrectPasswordException e) {
            throw e;
        } catch (InternalErrorException e) {
            throw e;
        } catch (Exception e) {
            throw new InternalErrorException(e);
        }

    }

    /**
     * Changes phone details of the user loaded in the facade. As
phone will be
     * used for important and individual notifications, password is
needed to be
     * provided.
     * @param password
     * @param newPhone
     * @param passwordIsEncrypted
     * @throws application.util.exceptions.IncorrectPasswordException
     * @throws application.util.exceptions.InternalErrorException
     */
    public void changePhone(String password, String newPhone,
            Boolean passwordIsEncrypted) throws
      IncorrectPasswordException, InternalErrorException {

        try {

            ChangePhoneAction action = new
            ChangePhoneAction(loginName,
                    password, passwordIsEncrypted, newPhone);

            PlainActionProcessor.process(getDataSource(), action);
        } catch (InstanceNotFoundException e) {
            throw new InternalErrorException(e);
        } catch (IncorrectPasswordException e) {
            throw e;
        } catch (InternalErrorException e) {
            throw e;
        } catch (Exception e) {
            throw new InternalErrorException(e);
        }
    }

// FINDING ACTIONS

    /**
     * Finds the profile of the user loaded in the facade.
     * @throws application.util.exceptions.InternalErrorException
     * @return
     */
    public UserProfileVO findUserProfile() throws
      InternalErrorException {

        try {

            FindUserProfileAction action = new
            FindUserProfileAction(loginName);

            return (UserProfileVO)
            PlainActionProcessor.process(getDataSource(),
                    action);

        } catch (InstanceNotFoundException e) {
            throw new InternalErrorException(e);
        } catch (InternalErrorException e) {
            throw e;
        } catch (Exception e) {
            throw new InternalErrorException(e);
        }
    }

    /**
     * Finds a pattern in an user profile and shows results ordered by
     * propertyName.
     * @param propertyInName
     * @param propertyInValue
     * @param propertyByName
     * @param startIndex
     * @param count
     * @throws application.util.exceptions.InternalErrorException
     * @return
     */
    public Collection findInBy(String propertyInName,
            String propertyInValue, String propertyByName,
            Integer startIndex, Integer count) throws
            InternalErrorException {
```

```
        try {

            FindInByAction action =
                    new FindInByAction(propertyInName,
                  propertyInValue, propertyByName, startIndex,
            count);

                    return (Collection)
                  PlainActionProcessor.process(getDataSource(),
                            action);

            } catch (InternalErrorException e) {
                    throw e;
            } catch (Exception e) {
                    throw new InternalErrorException(e);
            }
    }

//ADMINISTRATOR ACTIONS

    /**
     * Register a new user. This user will access the system and
     * userprofile services. When a new user is registered is load in
     * the facade assecondary user.
     * @param loginName
     * @param clearPassword
     * @param userProfileDetailsVO
     * @param phone
     * @param userProfileAccessVO
     * @throws
     * application.util.exceptions.AdministratorRequiredException
     * @throws application.util.exceptions.DuplicateInstanceException
     * @throws application.util.exceptions.InternalErrorException
     */
    public void registerUser(String loginName, String clearPassword,
            UserProfileDetailsVO userProfileDetailsVO, String phone,
            UserProfileAccessVO userProfileAccessVO) throws
            AdministratorRequiredException,
      DuplicateInstanceException, InternalErrorException {

        try {

            if (isAdministrator) {

                RegisterUserAction action = new
            RegisterUserAction(loginName,
                        clearPassword, userProfileDetailsVO, phone,
                        userProfileAccessVO);

                PlainActionProcessor.process(getDataSource(), action);

                this.loginName = loginName;

            } else {
                throw new
                  AdministratorRequiredException(mainLoginName);
            }

        } catch (AdministratorRequiredException e) {
            throw e;
        } catch (DuplicateInstanceException e) {
            throw e;
        } catch (InternalErrorException e) {
            throw e;
        } catch (Exception e) {
            throw new InternalErrorException(e);
        }
    }

    /**
     * Updates the access profile of the user load in the facade.
     * @param userProfileAccessVO
     * @throws
     * application.util.exceptions.AdministratorRequiredException
     * @throws application.util.exceptions.InternalErrorException
     */
    public void updateUserProfileAccess(UserProfileAccessVO
      userProfileAccessVO) throws AdministratorRequiredException,
      InternalErrorException {

        try {

            if (isAdministrator) {
            UpdateUserProfileAccessAction action =
                    new UpdateUserProfileAccessAction(loginName,
                    userProfileAccessVO);

            PlainActionProcessor.process(getDataSource(), action);
            } else {
                throw new
                  AdministratorRequiredException(mainLoginName);
            }
        }catch (AdministratorRequiredException e) {
            throw e;
        } catch (InstanceNotFoundException e) {
            throw new InternalErrorException(e);
        } catch (InternalErrorException e) {
            throw e;
        } catch (Exception e) {
            throw new InternalErrorException(e);
        }
    }

    /**
     * Gives a new user password. Administrator password is needed for
     * this action.
     * @param password
     * @param newClearPassword
     * @param passwordIsEncrypted
     * @throws
     * application.util.exceptions.AdministratorRequiredException
     * @throws application.util.exceptions.IncorrectPasswordException
     * @throws application.util.exceptions.InternalErrorException
     */
    public void givePassword(String password, String newClearPassword,
            Boolean passwordIsEncrypted) throws
      AdministratorRequiredException,
            IncorrectPasswordException, InternalErrorException {

        try {

            if (isAdministrator) {
                checkAuthority(password, passwordIsEncrypted);

                GivePasswordAction action = new
            GivePasswordAction(loginName,
                        newClearPassword, false);

                PlainActionProcessor.process(getDataSource(), action);

            } else {
                throw new
            AdministratorRequiredException(mainLoginName);
            }
        } catch ( AdministratorRequiredException e) {
            throw e;
        } catch (InstanceNotFoundException e) {
            throw new InternalErrorException(e);
        } catch (IncorrectPasswordException e) {
            throw e;
        } catch (InternalErrorException e) {
            throw e;
        } catch (Exception e) {
            throw new InternalErrorException(e);
        }

    }

    /**
     * Gives a new phone detail for the user loaded in facade. As
     * phone will be used for important and individual notifications,
     * password is need to be provided.
     * @param password
     * @param newPhone
     * @param passwordIsEncrypted
     * @throws
     * application.util.exceptions.AdministratorRequiredException
     * @throws application.util.exceptions.IncorrectPasswordException
     * @throws application.util.exceptions.InternalErrorException
     */
    public void givePhone(String password, String newPhone,
            Boolean passwordIsEncrypted) throws
      AdministratorRequiredException,
            IncorrectPasswordException, InternalErrorException {

        try {

            if (isAdministrator) {
                checkAuthority(password, passwordIsEncrypted);

                GivePhoneAction action = new
                  GivePhoneAction(loginName, newPhone);

                PlainActionProcessor.process(getDataSource(), action);
            } else {
                throw new
            AdministratorRequiredException(mainLoginName);
            }
        } catch (AdministratorRequiredException e) {
            throw e;
        } catch (InstanceNotFoundException e) {
            throw new InternalErrorException(e);
        } catch (IncorrectPasswordException e) {
            throw e;
        } catch (InternalErrorException e) {
            throw e;
        } catch (Exception e) {
            throw new InternalErrorException(e);
        }

    }

    /**
     * An administrator logs in as another user.
     * @param password
     * @param loginName
     * @param passwordIsEncrypted
     * @throws
     * application.util.exceptions.AdministratorRequiredException
     * @throws application.util.exceptions.IncorrectPasswordException
     * @throws application.util.exceptions.InstanceNotFoundException
     * @throws application.util.exceptions.InternalErrorException
     */
    public void loginAsUser(String password, String loginName,
            Boolean passwordIsEncrypted) throws
      AdministratorRequiredException,
            IncorrectPasswordException, InstanceNotFoundException,
            InternalErrorException {

        try {

            if (isAdministrator) {
                checkAuthority(password, passwordIsEncrypted);

                FindUserProfileAction action = new
            FindUserProfileAction(loginName);

                PlainActionProcessor.process(getDataSource(), action);

                this.loginName = loginName;
            } else {
                throw new
            AdministratorRequiredException(mainLoginName);
            }
        } catch (AdministratorRequiredException e) {
            throw e;
        } catch (IncorrectPasswordException e) {
            throw e;
        } catch (InstanceNotFoundException e) {
            throw e;
        } catch (InternalErrorException e) {
            throw e;
        } catch (Exception e) {
            throw new InternalErrorException(e);
        }
    }

    /**
     * Finds an user profile.
     * @return
     * @param loginName
     * @throws
     * application.util.exceptions.AdministratorRequiredException
     * @throws application.util.exceptions.InstanceNotFoundException
     * @throws application.util.exceptions.InternalErrorException
     */
    public UserProfileVO findUserProfile(String loginName) throws
            AdministratorRequiredException, InstanceNotFoundException,
            InternalErrorException {

        try {

            if (isAdministrator) {
                FindUserProfileAction action =
                        new FindUserProfileAction(loginName);

                return (UserProfileVO)
                    PlainActionProcessor.process(getDataSource(),
                  action);
            } else {
                throw new
           AdministratorRequiredException(mainLoginName);
            }
        } catch (AdministratorRequiredException e) {
            throw e;
        } catch (InstanceNotFoundException e) {
            throw e;
        } catch (InternalErrorException e) {
            throw e;
        } catch (Exception e) {
            throw new InternalErrorException(e);
        }
    }

// UTILITIES

    /**
     * Returns the loginName of the main user loaded in the facade.
     * @return
     */
```

```
public String getMainLoginName() {
    return mainLoginName;
}

/**
 * Returns the loginName of the secondary user loaded in the
 * facade.
 * @return
 */
public String getLoginName() {
    return loginName;
}
/**
 * Returns if the main user is administrator or not.
 * @return
 */
public Boolean getIsAdministrator() {
    return isAdministrator;
}

/**
 * Checks the authority of an user for doing an action.
 * @param userName
 * @param password
 * @param passwordIsEncrypted
 * @throws application.util.exceptions.InstanceNotFoundException
 * @throws application.util.exceptions.IncorrectPasswordException
 * @throws application.util.exceptions.InternalErrorException
 * @return
 */
private UserFacadeVO checkAuthority(String password,
        Boolean passwordIsEncrypted) throws
  IncorrectPasswordException,
        InternalErrorException {

    try {

        LoginAction action = new LoginAction(mainLoginName,
        password, passwordIsEncrypted);

        UserFacadeVO loginResultVO = (UserFacadeVO)
                PlainActionProcessor.process(getDataSource(),
              action);

        return loginResultVO;

    } catch (InstanceNotFoundException e) {
        throw new InternalErrorException(e);
    } catch (IncorrectPasswordException e) {
        throw e;
    } catch (InternalErrorException e) {
        throw e;
    } catch (Exception e) {
        throw new InternalErrorException(e);
    }
    }

    /**
     * Gets datasource using JDNI.
     */
    private DataSource getDataSource() throws InternalErrorException {
          return DataSourceLocator.getDataSource(
                  GlobalNames.APPLICATION_DATA_SOURCE);
      }
}
```

10.5.3.3.b Servicio de solicitud de vacaciones

10.5.3.3.b.1 application.model.holidaysrequestservice.facade.vo

HolidaysProfileVO

```
package application.model.holidaysrequestservice.facade.vo;

import java.io.Serializable;

import java.util.Calendar;

/**
 * This class is used for interacting with user interface and holidays
 * profile.
 * <BR>
 * LoginName, period, allDaysOfHolidays and holidaysLeft resume
 * holidays service profile. Only getter methods are available for
 * accessing user holidays profile properties.
 * <BR>
 * LoginName represents an user.
 * Period represents the year in which a holidays service is valid.
 * AllDaysOfHolidays represents the number of days that the user can
 * take for holidays in a period.
 * HolidaysLeft represents the number of days which are still
 * available for taking holidays.
 */
public class HolidaysProfileVO implements Serializable {

    private String loginName;
    private Calendar period;
    private Integer allDaysOfHolidays;
    private Integer holidaysLeft;

    /**
     * Constructs VO.
     * @param loginName
     * @param period
     * @param allDaysOfHolidays
     * @param holidaysLeft
     */
    public HolidaysProfileVO(String loginName, Calendar period,
            Integer allDaysOfHolidays, Integer holidaysLeft) {

        this.loginName = loginName;
        this.period = period;
        this.allDaysOfHolidays = allDaysOfHolidays;
        this.holidaysLeft = holidaysLeft;

    }

    /**
     * Gets loginName.
     * @return
     */
    public String getLoginName()  {

        return loginName;
    }

    /**
     * Gets period.
     * @return
     */
    public Calendar getPeriod() {

        return this.period;
    }

    /**
     * Gets all days of holidays in a period.
     * @return
     */
    public Integer getAllDaysOfHolidays()   {

        return this.allDaysOfHolidays;
    }

    /**
     * Gets days of holidays still left.
     * @return
     */
    public Integer getHolidaysLeft()  {

        return this.holidaysLeft;
    }

    /**
     * Useful for debugging.
     * @return
     */
    public String toString() {
        return new String("loginName = " + loginName + " | period = "
            + period + " | allDaysOfHolidays = " +
      allDaysOfHolidays + " | holidaysLeft = " +
      holidaysLeft);
    }
}
```

HolidaysVO

```
package application.model.holidaysrequestservice.facade.vo;

import java.io.Serializable;

import java.util.Calendar;

import application.model.holidaysrequestservice.vo.holidaysrequest.
HolidaysRequestVO;
import
application.model.holidaysrequestservice.facade.vo.HolidaysProfileVO;

/**
 * This class is used for interacting with user interface and holidays
 * profile.
 * <BR>
 * It represents a holidays request and an user holidays profile.
 */
public class HolidaysVO implements Serializable {

    private HolidaysRequestVO holidaysRequestVO;
    private HolidaysProfileVO holidaysProfileVO;

    /**
     * Constructs VO
     * @param holidaysRequestVO
     * @param holidaysProfileVO
     */
    public HolidaysVO(HolidaysRequestVO holidaysRequestVO,
            HolidaysProfileVO holidaysProfileVO) {

        this.holidaysRequestVO = holidaysRequestVO;
        this.holidaysProfileVO = holidaysProfileVO;

    }

    /**
     * Gets holidays request.
     * @return
     */
    public HolidaysRequestVO getHolidaysRequestVO()  {

        return this.holidaysRequestVO;
    }
    /**
     * Gets user holidays profile.
     * @return
     */
    public HolidaysProfileVO getHolidaysProfileVO() {

        return this.holidaysProfileVO;
    }

    /**
     * Useful for debugging.
     * @return
     */
    public String toString() {

        return new String(holidaysRequestVO + " | " +
      holidaysProfileVO);
    }
}
```

10.5.3.3.b.2 application.model.holidaysrequestservice.facade.actions.holidaysservice

FindHolidaysServiceAction

```
package application.model.holidaysrequestservice.facade.actions.
holidaysservice;

import java.util.Calendar;

/* JDBC.*/
import java.sql.Connection;

import application.util.exceptions.InternalErrorException;
import application.util.exceptions.InstanceNotFoundException;

import application.util.sql.NonTransactionalPlainAction;

import application.model.holidaysrequestservice.dao.holidaysservice.
SQLHolidaysServiceDAO;
import application.model.holidaysrequestservice.dao.holidaysservice.
SQLHolidaysServiceDAOFactory;

/**
 * This non transactional action finds a holidays service with user
 * and perido given.
 */
public class FindHolidaysServiceAction implements
NonTransactionalPlainAction {

    private String loginName;
    private Calendar period;
    public FindHolidaysServiceAction(String loginName, Calendar
      period) {

        this.loginName = loginName;
        this.period = period;
    }

    public Object execute(Connection connection) throws
            InstanceNotFoundException, InternalErrorException {

        SQLHolidaysServiceDAO holidaysServiceDAO =
                (SQLHolidaysServiceDAO)
                SQLHolidaysServiceDAOFactory.getDAO();

        return holidaysServiceDAO.find(connection, loginName, period)

    }

}
```

FindHolidaysServicesAction

```
package application.model.holidaysrequestservice.facade.actions.
holidaysservice;

import java.util.Calendar;

/* JDBC.*/
import java.sql.Connection;

import application.util.exceptions.InternalErrorException;

import application.util.sql.NonTransactionalPlainAction;

import application.model.holidaysrequestservice.dao.holidaysservice.
SQLHolidaysServiceDAO;
import application.model.holidaysrequestservice.dao.holidaysservice.
SQLHolidaysServiceDAOFactory;

/**
 * This non transactional action finds alll holidays services.
 */
public class FindHolidaysServicesAction implements
        NonTransactionalPlainAction {

    private Integer startIndex;
    private Integer count;

    public FindHolidaysServicesAction(Integer startIndex,
            Integer count) {
```

```
        this.startIndex = startIndex;
        this.count = count;
    }

    public Object execute(Connection connection) throws
      InternalErrorException {

        SQLHolidaysServiceDAO holidaysServiceDAO =
                (SQLHolidaysServiceDAO)
                SQLHolidaysServiceDAOFactory.getDAO();

        return holidaysServiceDAO.findHolidaysServices(connection,
      startIndex, count);
    }

}
```

FindPeriodHolidaysServicesAction

```
package application.model.holidaysrequestservice.facade.actions.
holidaysservice;

import java.util.Calendar;

/* JDBC.*/
import java.sql.Connection;

import application.util.exceptions.InternalErrorException;
import application.util.exceptions.InstanceNotFoundException;

import application.util.sql.NonTransactionalPlainAction;

import application.model.holidaysrequestservice.dao.holidaysservice.
SQLHolidaysServiceDAO;
import application.model.holidaysrequestservice.dao.holidaysservice.
SQLHolidaysServiceDAOFactory;

/**
 * This non transactional action returns users holidays request
 * services in a period given.
 */
public class FindPeriodHolidaysServicesAction implements
        NonTransactionalPlainAction {

    private Calendar period;
    private Integer startIndex;
    private Integer count;

    public FindPeriodHolidaysServicesAction(Calendar period, Integer
      startIndex, Integer count) {
        this.period = period;
        this.startIndex = startIndex;
        this.count = count;
    }

    public Object execute(Connection connection) throws
            InstanceNotFoundException, InternalErrorException {

        SQLHolidaysServiceDAO holidaysServiceDAO =
                (SQLHolidaysServiceDAO)
                SQLHolidaysServiceDAOFactory.getDAO();

        return
holidaysServiceDAO.findUsersRegisteredInHolidaysService(connection,
      period, startIndex, count);

    }

}
```

FindUserHolidaysServicesAction

```
package application.model.holidaysrequestservice.facade.actions.
holidaysservice;

import java.util.Calendar;
import java.util.Collection;

/* JDBC.*/
import java.sql.Connection;

import application.util.exceptions.InstanceNotFoundException;
import application.util.exceptions.InternalErrorException;

import application.util.sql.NonTransactionalPlainAction;

import application.model.holidaysrequestservice.dao.holidaysservice.
SQLHolidaysServiceDAO;
import application.model.holidaysrequestservice.dao.holidaysservice.
SQLHolidaysServiceDAOFactory;

/**
 * This non transactional action returns all user holidays service.
 */
public class FindUserHolidaysServicesAction implements
        NonTransactionalPlainAction {

    private String loginName;
    private Integer startIndex;
    private Integer count;
    public FindUserHolidaysServicesAction(String loginName, Integer
      startIndex, Integer count) {

        this.loginName = loginName;
        this.startIndex = startIndex;
        this.count = count;
    }

    public Object execute(Connection connection) throws
            InstanceNotFoundException, InternalErrorException {

        SQLHolidaysServiceDAO holidaysServiceDAO =
                (SQLHolidaysServiceDAO)
                SQLHolidaysServiceDAOFactory.getDAO();

        return holidaysServiceDAO.findUserHolidaysServices(connection,
            loginName, startIndex, count);

    }

}
```

<u>RegisterHolidaysServiceAction</u>

```
package application.model.holidaysrequestservice.facade.actions.
holidaysservice;

import java.util.Calendar;

/* JDBC.*/
import java.sql.Connection;

import application.util.exceptions.DuplicateInstanceException;
import application.util.exceptions.InternalErrorException;

import application.util.sql.TransactionalPlainAction;

import application.model.holidaysrequestservice.dao.holidaysservice.
SQLHolidaysServiceDAO;
import application.model.holidaysrequestservice.dao.holidaysservice.
SQLHolidaysServiceDAOFactory;

import application.model.holidaysrequestservice.vo.holidaysservice.
HolidaysServiceVO;

/**
 * This transactional action insert a new holidays service.
 */
public class RegisterHolidaysServiceAction implements
      TransactionalPlainAction {
```

```
    private String loginName;
    private Calendar period;
    private Integer days;

    public RegisterHolidaysServiceAction(String loginName, Calendar
      period, Integer days) {

        this.loginName = loginName;
        this.period = period;
        this.days = days;

    }

    public Object execute(Connection connection) throws
            DuplicateInstanceException, InternalErrorException {

        SQLHolidaysServiceDAO holidaysServiceDAO =
      (SQLHolidaysServiceDAO)
                SQLHolidaysServiceDAOFactory.getDAO();
        HolidaysServiceVO holidaysServiceVO =
                new HolidaysServiceVO(loginName, period, days);

        holidaysServiceDAO.create(connection, holidaysServiceVO);

        return null;

    }

}
```

UpdateHolidaysServiceAction

```
package application.model.holidaysrequestservice.facade.actions.
holidaysservice;

import java.util.Calendar;

/* JDBC.*/
import java.sql.Connection;

import application.util.exceptions.InstanceNotFoundException;
import application.util.exceptions.InternalErrorException;

import application.util.sql.TransactionalPlainAction;

import application.model.holidaysrequestservice.dao.holidaysservice.
SQLHolidaysServiceDAO;
import application.model.holidaysrequestservice.dao.holidaysservice.
SQLHolidaysServiceDAOFactory;

import application.model.holidaysrequestservice.vo.holidaysservice.
```

```
HolidaysServiceVO;

/**
 * This transactional action updates a holidays service.
 */
public class UpdateHolidaysServiceAction implements
      TransactionalPlainAction {

    private String loginName;
    private Calendar period;
    private Integer days;

    public UpdateHolidaysServiceAction(String loginName, Calendar
      period, Integer days) {

        this.loginName = loginName;
        this.period = period;
        this.days = days;

    }

    public Object execute(Connection connection) throws
            InstanceNotFoundException, InternalErrorException {

        SQLHolidaysServiceDAO holidaysServiceDAO =
      (SQLHolidaysServiceDAO)
                SQLHolidaysServiceDAOFactory.getDAO();

        HolidaysServiceVO holidaysServiceVO =
                holidaysServiceDAO.find(connection, loginName,
            period);

        holidaysServiceVO.setDays(days);

        holidaysServiceDAO.update(connection, holidaysServiceVO);

        return null;

    }
}
```

10.5.3.3.b.3 application.model.holidaysrequestservice.facade.actions.holidaysrequest

FindAdvancedAction

```
package application.model.holidaysrequestservice.facade.actions.
holidaysrequest;
```

```
import java.util.Calendar;

/* JDBC. */
import java.sql.Connection;

import application.util.exceptions.InternalErrorException;

import application.util.sql.NonTransactionalPlainAction;

import application.model.holidaysrequestservice.vo.holidaysrequest.
HolidaysRequestVO;

import application.model.holidaysrequestservice.dao.holidaysrequest.
SQLHolidaysRequestDAO;
import application.model.holidaysrequestservice.dao.holidaysrequest.
SQLHolidaysRequestDAOFactory;

/**
 * This non transactional action carries out an advanced search.
 */
public class FindAdvancedAction implements
      NonTransactionalPlainAction{

    private String loginName;
    private Integer autoId;
    private Calendar period;
    private Calendar sinceDate;
    private Short state;
    private String propertyName;
    private Boolean isDesc;
    private Integer startIndex;
    private Integer count;

    public FindAdvanceAction(String loginName, Integer autoId,
      Calendar period, Calendar sinceDate, Short state, String
      propertyName, Boolean isDesc, Integer startIndex,
            Integer count) {

        this.loginName = loginName;
        this.autoId = autoId;
        this.period = period;
        this.sinceDate = sinceDate;
        this.state = state;
        this.propertyName = propertyName;
        this.isDesc = isDesc;
        this.startIndex = startIndex;
        this.count = count;
    }

    public Object execute(Connection connection) throws
      InternalErrorException {

        SQLHolidaysRequestDAO holidaysRequestDAO =
      (SQLHolidaysRequestDAO)
                SQLHolidaysRequestDAOFactory.getDAO();
        return holidaysRequestDAO.findAdvanced(connection, loginName,
      autoId, period, sinceDate, state, propertyName, isDesc,
      startIndex, count);

    }
}
```

FindHolidaysProfiles

```
package application.model.holidaysrequestservice.facade.actions.
holidaysrequest;

import java.util.Calendar;

/* JDBC.*/
import java.sql.Connection;

import application.util.exceptions.InternalErrorException;

import application.util.sql.NonTransactionalPlainAction;

import application.model.holidaysrequestservice.dao.holidaysrequest.
SQLHolidaysRequestDAO;
import application.model.holidaysrequestservice.dao.holidaysrequest.
SQLHolidaysRequestDAOFactory;

/**
 * This non transactional action returns all holidays profiles.
 */
public class FindHolidaysProfilesAction implements
        NonTransactionalPlainAction {

    private Integer startIndex;
    private Integer count;

    public FindHolidaysProfilesAction(Integer startIndex,
            Integer count) {

        this.startIndex = startIndex;
        this.count = count;
    }

    public Object execute(Connection connection) throws
      InternalErrorException {

        SQLHolidaysRequestDAO holidaysRequestDAO =
                (SQLHolidaysRequestDAO)
                     SQLHolidaysRequestDAOFactory.getDAO();

        return holidaysRequestDAO.findPeriods(connection, startIndex,
                count);

    }
}
```

FindIdHolidaysRequestAction

```
package application.model.holidaysrequestservice.facade.actions.
holidaysrequest;

/* JDBC. */
import java.sql.Connection;

import application.util.exceptions.InternalErrorException;
import application.util.exceptions.InstanceNotFoundException;

import application.util.sql.NonTransactionalPlainAction;

import application.model.holidaysrequestservice.vo.holidaysrequest.
HolidaysRequestVO;

import application.model.holidaysrequestservice.dao.holidaysrequest.
SQLHolidaysRequestDAO;
import application.model.holidaysrequestservice.dao.holidaysrequest.
SQLHolidaysRequestDAOFactory;

/**
 * This non transactional action finds a holidays request by
 * identifier.
 */
public class FindIdHolidaysRequestAction implements
      NonTransactionalPlainAction{

    private Integer autoId;

    public FindIdHolidaysRequestAction(Integer autoId) {

        this.autoId = autoId;
    }

    public Object execute(Connection connection) throws
            InstanceNotFoundException, InternalErrorException {

        SQLHolidaysRequestDAO holidaysRequestDAO =
      (SQLHolidaysRequestDAO)
                SQLHolidaysRequestDAOFactory.getDAO();

       return holidaysRequestDAO.findId(connection, autoId);

    }
}
```

FindUserIdHolidaysRequestAction

```
package application.model.holidaysrequestservice.facade.actions.
holidaysrequest;

/* JDBC. */
import java.sql.Connection;

import application.util.exceptions.InternalErrorException;
import application.util.exceptions.InstanceNotFoundException;

import application.util.sql.NonTransactionalPlainAction;

import application.model.holidaysrequestservice.vo.holidaysrequest.
HolidaysRequestVO;

import application.model.holidaysrequestservice.dao.holidaysrequest.
SQLHolidaysRequestDAO;
import application.model.holidaysrequestservice.dao.holidaysrequest.
SQLHolidaysRequestDAOFactory;

/**
 * This non transactional action finds a holidays request by
 * identifier.
 */
public class FindUserIdHolidaysRequestAction implements
      NonTransactionalPlainAction{

    private Integer autoId;
    private String loginName;

    /**
     *
     * @param autoId
     * @param loginName
     */
    public FindUserIdHolidaysRequestAction(Integer autoId, String
      loginName) {

        this.autoId = autoId;
        this.loginName = loginName;
    }

    /**
     *
     * @param connection
     * @throws application.util.exceptions.InstanceNotFoundException
     * @throws application.util.exceptions.InternalErrorException
     * @return
     */
    public Object execute(Connection connection) throws
            InstanceNotFoundException, InternalErrorException {

        SQLHolidaysRequestDAO holidaysRequestDAO =
      (SQLHolidaysRequestDAO)
```

```
                SQLHolidaysRequestDAOFactory.getDAO();

       return holidaysRequestDAO.findIdUser(connection, autoId,
      loginName);

    }

}
```

FindUserHolidaysRequestByAction

```
package application.model.holidaysrequestservice.facade.actions.
holidaysrequest;

/* JDBC. */
import java.sql.Connection;

import application.util.exceptions.InternalErrorException;

import application.util.sql.NonTransactionalPlainAction;

import application.model.holidaysrequestservice.vo.holidaysrequest.
HolidaysRequestVO;

import application.model.holidaysrequestservice.dao.holidaysrequest.
SQLHolidaysRequestDAO;
import application.model.holidaysrequestservice.dao.holidaysrequest.
SQLHolidaysRequestDAOFactory;

/**
 * This non transactional action finds all user holidays request.
 */
public class FindUserHolidaysRequestByAction implements
      NonTransactionalPlainAction{

    private String loginName;
    private String propertyName;
    private Boolean isDesc;
    private Integer startIndex;
    private Integer count;

    public FindUserHolidaysRequestByAction(String loginName,
            String propertyName, Boolean isDesc, Integer startIndex,
      Integer count) {

        this.loginName = loginName;
        this.propertyName = propertyName;
        this.isDesc = isDesc;
        this.startIndex = startIndex;
        this.count = count;
    }
    public Object execute(Connection connection) throws
      InternalErrorException {

        SQLHolidaysRequestDAO holidaysRequestDAO =
      (SQLHolidaysRequestDAO)
                SQLHolidaysRequestDAOFactory.getDAO();

       return holidaysRequestDAO.findUserBy(connection, loginName,
      propertyName, isDesc, startIndex, count);

    }
}
```

FindUserPeriodStateHolidaysRequestByAction

```
package application.model.holidaysrequestservice.facade.actions.
holidaysrequest;

import java.util.Calendar;

/* JDBC. */
import java.sql.Connection;

import application.util.exceptions.InternalErrorException;

import application.util.sql.NonTransactionalPlainAction;

import application.model.holidaysrequestservice.vo.holidaysrequest.
HolidaysRequestVO;

import application.model.holidaysrequestservice.dao.holidaysrequest.
SQLHolidaysRequestDAO;
import application.model.holidaysrequestservice.dao.holidaysrequest.
SQLHolidaysRequestDAOFactory;

/**
 * This non transactional action finds all user holidays service in a
 * period with a state given.
 */
public class FindUserPeriodStateHolidaysRequestByAction implements
      NonTransactionalPlainAction{

    private String loginName;
    private Calendar period;
    private Short state;
    private String propertyName;
    private Integer startIndex;
    private Integer count;
    public FindUserPeriodStateHolidaysRequestByAction(String
      loginName, Calendar period, Short state, String
      propertyName, Integer startIndex, Integer count) {

        this.loginName = loginName;
        this.period = period;
        this.state = state;
        this.propertyName = propertyName;
        this.startIndex = startIndex;
        this.count = count;
    }

    public Object execute(Connection connection) throws
      InternalErrorException {

        SQLHolidaysRequestDAO holidaysRequestDAO =
      (SQLHolidaysRequestDAO)
                SQLHolidaysRequestDAOFactory.getDAO();

       return holidaysRequestDAO.findUserPeriodStateBy(connection,
      loginName, period, state, propertyName, startIndex, count);

    }

}
```

<u>RegisterHolidaysRequestAction</u>

```
package application.model.holidaysrequestservice.facade.actions.
holidaysrequest;

import java.util.Calendar;

/* JDBC. */
import java.sql.Connection;

import application.util.exceptions.InternalErrorException;
import application.util.exceptions.DuplicateInstanceException;

import application.util.sql.TransactionalPlainAction;

import application.model.holidaysrequestservice.vo.holidaysrequest.
HolidaysRequestVO;

import application.model.holidaysrequestservice.dao.holidaysrequest.
SQLHolidaysRequestDAO;
import application.model.holidaysrequestservice.dao.holidaysrequest.
SQLHolidaysRequestDAOFactory;

/**
 * This transactional action register a new holidays request.
 */
public class RegisterHolidaysRequestAction implements
      TransactionalPlainAction {

    private String loginName;
    private Calendar startDate;
    private Calendar finalDate;
    private String details;

    public RegisterHolidaysRequestAction(String loginName, Calendar
      startDate, Calendar finalDate, String details) {

        this.loginName = loginName;
        this.startDate = startDate;
        this.finalDate = finalDate;
        this.details = details;
    }

    public Object execute(Connection connection) throws
            DuplicateInstanceException, InternalErrorException {

        SQLHolidaysRequestDAO holidaysRequestDAO =
      (SQLHolidaysRequestDAO)
                SQLHolidaysRequestDAOFactory.getDAO();

        HolidaysRequestVO holidaysRequestVO = new HolidaysRequestVO(0,
                  loginName, Calendar.getInstance(),
             Calendar.getInstance(), startDate, finalDate,
      Short.valueOf("0"), details);

       return holidaysRequestDAO.create(connection,
      holidaysRequestVO);

    }
}
```

UpdateHolidaysRequestAction

```
package application.model.holidaysrequestservice.facade.actions.
holidaysrequest;

/* JDBC. */
import java.sql.Connection;

import application.util.exceptions.InternalErrorException;
import application.util.exceptions.InstanceNotFoundException;
import application.util.exceptions.OperationNotAllowedException;

import application.util.sql.TransactionalPlainAction;

import application.model.holidaysrequestservice.vo.holidaysrequest.
```

```
HolidaysRequestVO;

import application.model.holidaysrequestservice.dao.holidaysrequest.
SQLHolidaysRequestDAO;
import application.model.holidaysrequestservice.dao.holidaysrequest.
SQLHolidaysRequestDAOFactory;

/**
 * This transactional actions updates a holidays request state.
 */
public class UpdateHolidaysRequestAction implements
TransactionalPlainAction {

    private Integer autoId;
    private Short state;

    /**
     *
     * @param autoId
     * @param state
     */
    public UpdateHolidaysRequestAction(Integer autoId, Short state) {

        this.autoId = autoId;
        this.state = state;
    }

    /**
     *
     * @param connection
     * @throws application.util.exceptions.InstanceNotFoundException
     * @throws application.util.exceptions.InternalErrorException
     * @throws
     * application.util.exceptions.OperationNotAllowedException
     * @return
     */
    public Object execute(Connection connection) throws
            InstanceNotFoundException,
      InternalErrorException,OperationNotAllowedException {

        SQLHolidaysRequestDAO holidaysRequestDAO =
      (SQLHolidaysRequestDAO)
                SQLHolidaysRequestDAOFactory.getDAO();

        HolidaysRequestVO holidaysRequestVO =
                holidaysRequestDAO.findId(connection, autoId);

        if (holidaysRequestVO.getState().equals(Short.valueOf("0"))) {

            holidaysRequestVO.setState(state);

            holidaysRequestDAO.update(connection, holidaysRequestVO);

        } else {
           throw new OperationNotAllowedException("cancel request-
            "+autoId);
        }

       return null;

    }

}
```

CancelHolidaysRequestAction

```
package application.model.holidaysrequestservice.facade.actions.
holidaysrequest;

/* JDBC. */
import java.sql.Connection;

import application.util.exceptions.InternalErrorException;
import application.util.exceptions.InstanceNotFoundException;
import application.util.exceptions.OperationNotAllowedException;

import application.util.sql.TransactionalPlainAction;

import application.model.holidaysrequestservice.vo.holidaysrequest.
HolidaysRequestVO;

import application.model.holidaysrequestservice.dao.holidaysrequest.
SQLHolidaysRequestDAO;
import application.model.holidaysrequestservice.dao.holidaysrequest.
SQLHolidaysRequestDAOFactory;

/**
 * This transactional actions cancels a holidays request.
 */
public class CancelHolidaysRequestAction implements
      TransactionalPlainAction {

    private String loginName;
    private Integer autoId;
    private Short state;

    /**
     *
     * @param loginName
     * @param autoId
     */
    public CancelHolidaysRequestAction(String loginName, Integer
      autoId) {

        this.loginName = loginName;
        this.autoId = autoId;
        this.state = state;
    }
    /**
     *
     * @param connection
     * @throws
     * application.util.exceptions.OperationNotAllowedException
     * @throws application.util.exceptions.InstanceNotFoundException
     * @throws application.util.exceptions.InternalErrorException
     * @return
     */
    public Object execute(Connection connection) throws
            InstanceNotFoundException, InternalErrorException,
      OperationNotAllowedException {

        SQLHolidaysRequestDAO holidaysRequestDAO =
      (SQLHolidaysRequestDAO)
                SQLHolidaysRequestDAOFactory.getDAO();

        HolidaysRequestVO holidaysRequestVO =
                holidaysRequestDAO.findIdUser(connection, autoId,
            loginName);

        if (holidaysRequestVO.getState().equals(Short.valueOf("0"))) {

            holidaysRequestVO.setState(Short.valueOf("3"));

            holidaysRequestDAO.update(connection, holidaysRequestVO);

        } else {
           throw new OperationNotAllowedException("cancel request-
            "+autoId);
        }

       return null;

    }

}
```

10.5.3.3.b.4 application.model.holidaysrequestservice.facade.delegate

HolidaysRequestServiceFacade

```
package application.model.holidaysrequestservice.facade.delegate;

import java.io.Serializable;

import java.util.Calendar;
import java.util.Collection;
import application.util.exceptions.AdministratorRequiredException;
import application.util.exceptions.DuplicateInstanceException;
import application.util.exceptions.InstanceNotFoundException;
import application.util.exceptions.InternalErrorException;
import application.util.exceptions.OperationNotAllowedException;

import
application.model.holidaysrequestservice.facade.vo.HolidaysProfileVO;

import application.model.holidaysrequestservice.vo.holidaysservice.
HolidaysServiceVO;
import application.model.holidaysrequestservice.vo.holidaysrequest.
HolidaysRequestVO;

import application.model.userprofileservice.facade.delegate.
UserProfileServiceFacade;

/**
 * A facade to model the interaction of the user with the service.
 * There is some logical restrictions with regard to the order of
 * method calling.
 * In particular, each specific holidays service related action needs
 * the user previously being authenticated.
 */
public interface HolidaysRequestServiceFacade extends Serializable {

//
// HOLIDAYS SERVICE REGISTRY
//--------------------------

// FINDING ACTIONS

    /**
     * Finds an existing holidays service for the user.
     * @param period
     * @throws application.util.exceptions.InstanceNotFoundException
     * @throws application.util.exceptions.InternalErrorException
     * @return
     */
    public HolidaysServiceVO findHolidaysService(Calendar period)
      throws
            InstanceNotFoundException, InternalErrorException;

    /**
     * Finds all user holidays services.
     * @param startIndex
     * @param count
     * @throws application.util.exceptions.InternalErrorException
     * @return
     */
    public Collection findUserHolidaysServices(Integer startIndex,
            Integer count) throws InternalErrorException;


// ADMINISTRATOR ACTIONS
```

```
    /**
     * Register new user holidays service.
     * @param period
     * @param days
     * @throws
     * application.util.exceptions.AdministratorRequiredException
     * @throws application.util.exceptions.DuplicateInstanceException
     * @throws application.util.exceptions.InternalErrorException
     */
    public void registerHolidaysService(Calendar period, Integer days)
            throws AdministratorRequiredException,
      DuplicateInstanceException, InternalErrorException;

    /**
     * Updates an user holidays service.
     * @param period
     * @param days
     * @throws
     * application.util.exceptions.AdministratorRequiredException
     * @throws application.util.exceptions.InstanceNotFoundException
     * @throws application.util.exceptions.InternalErrorException
     */
    public void updateHolidaysService(Calendar period, Integer days)
      throws AdministratorRequiredException,
      InstanceNotFoundException, InternalErrorException;

    /**
     * Finds all holidays services in a period.
     * @return
     * @param period
     * @param startIndex
     * @param count
     * @throws
     * application.util.exceptions.AdministratorRequiredException
     * @throws application.util.exceptions.InternalErrorException
     */
    public Collection findPeriodHolidaysServices(Calendar period,
            Integer startIndex, Integer count) throws
            AdministratorRequiredException, InternalErrorException;

    /**
     * Finds all period with at least one holidays service.
     * @return
     * @param startIndex
     * @param count
     * @throws
     * application.util.exceptions.AdministratorRequiredException
     * @throws application.util.exceptions.InternalErrorException
     */
    public Collection findHolidaysServices(Integer startIndex, Integer
            count) throws AdministratorRequiredException,
      InternalErrorException;

//
// HOLIDAYS REQUEST REGISTRY
//--------------------------
```

```
    /**
     * Register new holidays service and returns auto gerenerated id.
     * @param startDate
     * @param finalDate
     * @param details
     * @throws application.util.exceptions.InstanceNotFoundException
     * @throws application.util.exceptions.DuplicateInstanceException
     * @throws application.util.exceptions.InternalErrorException
     * @return auto generated id for the request
     */
    public Integer registerHolidaysRequest(Calendar startDate,
            Calendar finalDate, String details)throws
      InstanceNotFoundException,
            DuplicateInstanceException, InternalErrorException;

    /**
     * Finds an user holidays request by its identificator.
     * @return
     * @param autoId
     * @throws application.util.exceptions.InstanceNotFoundException
     * @throws application.util.exceptions.InternalErrorException
     */
    public HolidaysRequestVO findUserIdHolidaysRequest(Integer autoId)
            throws InstanceNotFoundException, InternalErrorException;

    /**
     *
     * Cancel a holidays request if the request is not tramitted.
     * @param autoId
     * @throws
     * application.util.exceptions.OperationNotAllowedException
     * @throws application.util.exceptions.InstanceNotFoundException
     * @throws application.util.exceptions.InternalErrorException
     */
    public void cancelHolidaysRequest(Integer autoId) throws
            InstanceNotFoundException, InternalErrorException,
      OperationNotAllowedException;

// FINDING ACTIONS

    /**
     * Carries out an advanced search.
     * @return
     * @param autoId
     * @param period
     * @param sinceDate
     * @param state
     * @param propertyName
     * @param isDesc
     * @param startIndex
     * @param count
     * @throws application.util.exceptions.InternalErrorException
     */
    public Collection findAdvancedUser(Integer autoId, Calendar
      period, Calendar sinceDate, Short state, String
      propertyName, Boolean isDesc, Integer startIndex, Integer
      count) throws InternalErrorException;

// ADMINISTRATOR ACTIONS
    /**
     * Updates state of a holidays request.
     * @param autoId
     * @param state
     * @throws
     * application.util.exceptions.OperationNotAllowedException
     * @throws
     * application.util.exceptions.AdministratorRequiredException
     * @throws application.util.exceptions.InstanceNotFoundException
     * @throws application.util.exceptions.InternalErrorException
     */
    public void updateHolidaysRequest(Integer autoId, Short state)
      throws AdministratorRequiredException,
      InstanceNotFoundException,
            InternalErrorException, OperationNotAllowedException;

    /**
     * Finds a holidays request by its identificator.
     * @return
     * @param autoId
     * @throws
     * application.util.exceptions.AdministratorRequiredException
     * @throws application.util.exceptions.InstanceNotFoundException
     * @throws application.util.exceptions.InternalErrorException
     */
    public HolidaysRequestVO findIdHolidaysRequest(Integer autoId)
      throws AdministratorRequiredException,
      InstanceNotFoundException, InternalErrorException;

    /**
     * Carries out an advanced search.
     * @return
     * @param isUserLoaded
     * @param autoId
     * @param period
     * @param sinceDate
     * @param state
     * @param propertyName
     * @param isDesc
     * @param startIndex
     * @param count
     * @throws
     * application.util.exceptions.AdministratorRequiredException
     * @throws application.util.exceptions.InternalErrorException
     */
    public Collection findAdvanced(Boolean isUserLoaded, Integer
      autoId, Calendar period, Calendar sinceDate, Short state,
            String propertyName, Boolean isDesc, Integer startIndex,
      Integer count)
            throws AdministratorRequiredException,
      InternalErrorException;
```

```
//
// HOLIDAYS PROFILE
//-----------------

    /**
     * Returns all user holidays profiles.
     * @param startIndex
     * @param count
     * @throws application.util.exceptions.InternalErrorException
     * @return
     */
    public Collection findHolidaysProfile(Integer startIndex, Integer
      count) throws InternalErrorException;

    /**
     *
     * @param period
     * @throws application.util.exceptions.InstanceNotFoundException
     * @throws application.util.exceptions.InternalErrorException
     * @return
     */
    public HolidaysProfileVO findHolidaysProfilePeriod(Calendar
      period) throws
            InstanceNotFoundException, InternalErrorException;

    /**
     * Retunrs holidays service profile in a period.
     * @return
     * @param userName
     * @param period
     * @throws
     * application.util.exceptions.AdministratorRequiredException
     * @throws application.util.exceptions.InstanceNotFoundException
     * @throws application.util.exceptions.InternalErrorException
     */
    public HolidaysProfileVO findUserHolidaysProfilePeriod(String
      userName, Calendar period) throws
      AdministratorRequiredException,
            InstanceNotFoundException, InternalErrorException;

    /**
     * Finds periods with holidays profiles.
     * @return
     * @param startIndex
     * @param count
     * @throws
     * application.util.exceptions.AdministratorRequiredException
     * @throws application.util.exceptions.InternalErrorException
     */
    public Collection findHolidaysProfiles(Integer startIndex, Integer
            count) throws AdministratorRequiredException,
      InternalErrorException;

    /**
     * Retunrs all holidays profiles in a period.
     * @return
     * @param period
     * @param startIndex
     * @param count
     * @throws
     * application.util.exceptions.AdministratorRequiredException
     * @throws application.util.exceptions.InternalErrorException
     */
    public Collection findPeriodHolidaysProfile(Calendar period,
            Integer startIndex, Integer count) throws
            AdministratorRequiredException, InternalErrorException;

// UTILITIES

    /**
     * Initiates an user holidays request service from an
     * <code>UserProfileServiceFacade</code>.
     * @param userFacadeDelegate
     */
    public void initHolidaysRequestService(UserProfileServiceFacade
            userFacadeDelegate);

}
```

HolidaysRequestServiceFacadeFactory

```
package application.model.holidaysrequestservice.facade.delegate;

/* Uses JNDI. */
import application.util.jndi.ConfigurationParametersManager;

import application.util.exceptions.InternalErrorException;

/**
 * This class creates an intance of the implementation of
 * <code>HolidaysRequestServiceFacade</code>.
 * <BR>
 * Upper layers use this class to determine which implementation must
 * be used. This makes code independent from each façade
 * implementation.
 * <BR>
 * <code>HolidaysRequestServiceFacadeFactory/facadeClassName</code>
 * parameter has
 * to be defined using JNDI. This parameter can be defined in
 * <code>web.xml</code> with <code><env-entry></code> tags.
 * <BR>
 * This code provide an implementation of
 * <code>UserProfileServiceFacade</code> for not distributed, but
 * plain application in <code>PlainUserProfileServiceFacade</code>.
 */
public final class HolidaysRequestServiceFacadeFactory {

    /* This parameter has to be define in a JNDI context. */
    private final static String CLASS_NAME_PARAMETER =
            "HolidaysRequestServiceFacadeFactory/facadeClassName";

    private final static Class facadeClass = getFacadeClass();

    private HolidaysRequestServiceFacadeFactory(){};

    /* Gets the name of the implementation class of Facade. */
    private final static Class getFacadeClass() {

        Class theClass = null;
        try {
            String className =
            ConfigurationParametersManager.getParameter(
                    CLASS_NAME_PARAMETER);
            theClass = Class.forName(className);
        } catch (Exception e) {
            e.printStackTrace();
        }

        return theClass;
    }

    /**
     * Returns an instance of the implementation class of Facade.
     * @throws application.util.exceptions.InternalErrorException
     * @return implementation of facade
     */
    public final static HolidaysRequestServiceFacade getFacade()
      throws InternalErrorException {

        try {
            return (HolidaysRequestServiceFacade)
            facadeClass.newInstance();
        } catch (Exception e) {
            throw new InternalErrorException(e);
        }
    }
}
```

PlainHolidaysRequestServiceFacade

```
package application.model.holidaysrequestservice.facade.delegate;

import java.util.ArrayList;
import java.util.Calendar;
import java.util.Collection;
import java.util.Iterator;

import application.model.util.HolidaysRequestTable;

/* JDBC.*/
import javax.sql.DataSource;
import application.util.sql.DataSourceLocator;
import application.util.sql.PlainActionProcessor;

import application.util.exceptions.AdministratorRequiredException;
import application.util.exceptions.DuplicateInstanceException;
import application.util.exceptions.InstanceNotFoundException;
import application.util.exceptions.InternalErrorException;
import application.util.exceptions.OperationNotAllowedException;

import application.model.util.GlobalNames;

import application.model.holidaysrequestservice.vo.holidaysservice.
HolidaysServiceVO;
import application.model.holidaysrequestservice.vo.holidaysrequest.
HolidaysRequestVO;

import
application.model.holidaysrequestservice.facade.vo.HolidaysProfileVO;

import application.model.userprofileservice.facade.delegate.
UserProfileServiceFacade;

import application.model.holidaysrequestservice.facade.actions.
holidaysservice.RegisterHolidaysServiceAction;
import application.model.holidaysrequestservice.facade.actions.
holidaysservice.UpdateHolidaysServiceAction;
import application.model.holidaysrequestservice.facade.actions.
holidaysservice.FindHolidaysServiceAction;
import application.model.holidaysrequestservice.facade.actions.
holidaysservice.FindPeriodHolidaysServicesAction;
import application.model.holidaysrequestservice.facade.actions.
holidaysservice.FindUserHolidaysServicesAction;
import application.model.holidaysrequestservice.facade.actions.
holidaysservice.FindHolidaysServicesAction;
import application.model.holidaysrequestservice.facade.actions.
holidaysrequest.RegisterHolidaysRequestAction;
import application.model.holidaysrequestservice.facade.actions.
holidaysrequest.FindAdvancedAction;
import application.model.holidaysrequestservice.facade.actions.
holidaysrequest.FindUserIdHolidaysRequestAction;
import application.model.holidaysrequestservice.facade.actions.
holidaysrequest.FindIdHolidaysRequestAction;
import application.model.holidaysrequestservice.facade.actions.
holidaysrequest.FindUserHolidaysRequestByAction;
import application.model.holidaysrequestservice.facade.actions.
holidaysrequest.FindUserPeriodStateHolidaysRequestByAction;
import application.model.holidaysrequestservice.facade.actions.
holidaysrequest.UpdateHolidaysRequestAction;
import application.model.holidaysrequestservice.facade.actions.
holidaysrequest.CancelHolidaysRequestAction;
import application.model.holidaysrequestservice.facade.actions.
holidaysrequest.FindHolidaysProfilesAction;

import
application.model.holidaysrequestservice.operations.Jworkingdays;
```

```
/**
 * This class provides an implementation of
 * <code>HolidaysRequestServiceFacade</code> for interacting with user
 * holidays request service in facade layer. This implementation is
 * for a not EJB but
 * JDBC application.
 */
public class PlainHolidaysRequestServiceFacade implements
        HolidaysRequestServiceFacade {

    /* Holds the value of the main user loaded in facade. */
    private String mainLoginName;

    /* Holds value of the user load in facade. */
    private String loginName;

    /* Holds is the main user is administrator or not. */
    private Boolean isAdministrator;

    /**
     * Constructs Facade.
     */
    public PlainHolidaysRequestServiceFacade() {
        mainLoginName = null;
        loginName = null;
        isAdministrator = false;
    }

//
// HOLIDAYS SERVICE REGISTRY
//--------------------------


//FINDING ACTIONS

    /**
     * Finds an existing holidays service for the user.
     * @param period
     * @throws application.util.exceptions.InstanceNotFoundException
     * @throws application.util.exceptions.InternalErrorException
     * @return
     */
    public HolidaysServiceVO findHolidaysService (Calendar period)
      throws
            InstanceNotFoundException, InternalErrorException {

        try {

            FindHolidaysServiceAction action =
                    new FindHolidaysServiceAction(loginName, period);

            return (HolidaysServiceVO) PlainActionProcessor.process(
                    getDataSource(), action);

        } catch (InstanceNotFoundException e) {
            throw e;
        } catch (InternalErrorException e) {
            throw e;
        } catch (Exception e) {
            throw new InternalErrorException(e);
        }
    }

    /**
     * Finds all user holidays services.
     * @param startIndex
     * @param count
     * @throws application.util.exceptions.InternalErrorException
     * @return
     */
    public Collection findUserHolidaysServices(Integer startIndex,
            Integer count) throws InternalErrorException{

        try {

            FindUserHolidaysServicesAction action =
                    new FindUserHolidaysServicesAction(loginName,
                  startIndex, count);

            return (Collection)
            PlainActionProcessor.process(getDataSource(),
                    action);

        } catch (InternalErrorException e) {
            throw e;
        } catch (Exception e) {
            throw new InternalErrorException(e);
        }
    }


//ADMINISTRATOR ACTIONS

    /**
     * Register new user holidays service.
     * @param period
     * @param days
     * @throws
     * application.util.exceptions.AdministratorRequiredException
     * @throws application.util.exceptions.DuplicateInstanceException
     * @throws application.util.exceptions.InternalErrorException
     */
    public void registerHolidaysService(Calendar period, Integer days)
            throws AdministratorRequiredException,
      DuplicateInstanceException, InternalErrorException {

        try {

            if (isAdministrator) {

            RegisterHolidaysServiceAction action =
                    new RegisterHolidaysServiceAction(loginName,
                  period, days);

            PlainActionProcessor.process(getDataSource(), action);

            } else {
                throw new
                   AdministratorRequiredException(mainLoginName);
            }

        } catch (AdministratorRequiredException e) {
            throw e;
        } catch (DuplicateInstanceException e) {
            throw e;
        } catch (InternalErrorException e) {
            throw e;
        } catch (Exception e) {
            throw new InternalErrorException(e);
        }
    }

    /**
     * Updates an user holidays service.
     * @param period
     * @param days
     * @throws
     * application.util.exceptions.AdministratorRequiredException
     * @throws application.util.exceptions.InstanceNotFoundException
     * @throws application.util.exceptions.InternalErrorException
     */
    public void updateHolidaysService(Calendar period, Integer days)
      throws AdministratorRequiredException,
      InstanceNotFoundException, InternalErrorException {

        try {

            if (isAdministrator) {

            UpdateHolidaysServiceAction action =
                    new UpdateHolidaysServiceAction(loginName, period,
                        days);

            PlainActionProcessor.process(getDataSource(), action);

            } else {
                throw new
                   AdministratorRequiredException(mainLoginName);
            }
        } catch (AdministratorRequiredException e) {
            throw e;
        } catch (InstanceNotFoundException e) {
            throw e;
        } catch (InternalErrorException e) {
            throw e;
        } catch (Exception e) {
            throw new InternalErrorException(e);
        }

    }

    /**
     * Finds all holidays services in a period.
     * @return
     * @param period
     * @param startIndex
     * @param count
     * @throws
     * application.util.exceptions.AdministratorRequiredException
     * @throws application.util.exceptions.InternalErrorException
     */
    public Collection findPeriodHolidaysServices(Calendar period,
            Integer startIndex, Integer count) throws
            AdministratorRequiredException, InternalErrorException{

        try {

            if (isAdministrator) {


            FindPeriodHolidaysServicesAction action =
                    new FindPeriodHolidaysServicesAction(period,
                  startIndex, count);

            return (Collection)
            PlainActionProcessor.process(getDataSource(),
                    action);
            } else {
                throw new
                   AdministratorRequiredException(mainLoginName);
            }

        } catch (AdministratorRequiredException e) {
            throw e;
        } catch (InternalErrorException e) {
            throw e;
        } catch (Exception e) {
            throw new InternalErrorException(e);
        }
    }

    /**
     * Finds all period with at least one holidays service.
     * @return
     * @param startIndex
     * @param count
     * @throws
     * application.util.exceptions.AdministratorRequiredException
     * @throws application.util.exceptions.InternalErrorException
     */
    public Collection findHolidaysServices(Integer startIndex, Integer
            count) throws AdministratorRequiredException,
      InternalErrorException {
        try {

            if (isAdministrator) {

            FindHolidaysServicesAction action =
                    new FindHolidaysServicesAction(startIndex, count);

            return (Collection)
            PlainActionProcessor.process(getDataSource(),
                    action);
            } else {
                throw new
            AdministratorRequiredException(mainLoginName);
            }

        } catch (AdministratorRequiredException e) {
            throw e;
        } catch (InternalErrorException e) {
            throw e;
        } catch (Exception e) {
            throw new InternalErrorException(e);
        }
    }


//
// HOLIDAYS REQUEST REGISTRY
//---------------------------


    /**
     * Register new holidays request and returns auto gerenerated id.
     * @param startDate
     * @param finalDate
     * @param details
     * @throws application.util.exceptions.InstanceNotFoundException
     * @throws application.util.exceptions.DuplicateInstanceException
     * @throws application.util.exceptions.InternalErrorException
     * @return auto generated id for the request
     */
    public Integer registerHolidaysRequest(Calendar startDate,
            Calendar finalDate, String details)throws
      InstanceNotFoundException,
            DuplicateInstanceException, InternalErrorException {

        try{

            FindHolidaysServiceAction findAction =
                    new FindHolidaysServiceAction(loginName,
                  startDate);

            PlainActionProcessor.process(getDataSource(), findAction);

            RegisterHolidaysRequestAction action =
                    new RegisterHolidaysRequestAction(loginName,
                  startDate, finalDate, details);

            return (Integer) PlainActionProcessor.process(
                    getDataSource(), action);

        } catch (InstanceNotFoundException e) {
            throw e;
        } catch (DuplicateInstanceException e) {
            throw e;
        } catch (Exception e) {
            throw new InternalErrorException(e);
        }

    }

    /**
     * Finds an user holidays request by its identificator.
     * @return
     * @param autoId
     * @throws application.util.exceptions.InstanceNotFoundException
     * @throws application.util.exceptions.InternalErrorException
     */
    public HolidaysRequestVO findUserIdHolidaysRequest(Integer autoId)
            throws
            InstanceNotFoundException, InternalErrorException{

        try {

            FindUserIdHolidaysRequestAction action =
                    new FindUserIdHolidaysRequestAction(autoId,
                  loginName);

            return (HolidaysRequestVO)
            PlainActionProcessor.process(getDataSource(), action);

        } catch (InstanceNotFoundException e) {
            throw e;
        } catch (Exception e) {
            throw new InternalErrorException(e);
        }
    }

    /**
     * Cancel a holidays request if the request is not tramitted.
     * @param autoId
     * @throws
     * application.util.exceptions.OperationNotAllowedException
     * @throws application.util.exceptions.InstanceNotFoundException
     * @throws application.util.exceptions.InternalErrorException
     */
    public void cancelHolidaysRequest(Integer autoId) throws
            InstanceNotFoundException, InternalErrorException,
      OperationNotAllowedException {

        try{
            CancelHolidaysRequestAction action =
                    new CancelHolidaysRequestAction(loginName,
                  autoId);

            HolidaysRequestVO holidaysRequestVO = (HolidaysRequestVO)
                    PlainActionProcessor.process(getDataSource(),
                  action);

        } catch (InstanceNotFoundException e) {
            throw e;
        } catch (OperationNotAllowedException e) {
            throw e;
        } catch (Exception e) {
            throw new InternalErrorException(e);
        }

    }

    /**
     * Carries out an advanced search.
     * @return
     * @param autoId
     * @param period
     * @param sinceDate
     * @param state
     * @param propertyName
     * @param isDesc
     * @param startIndex
     * @param count
     * @throws application.util.exceptions.InternalErrorException
     */
    public Collection findAdvancedUser(Integer autoId,
            Calendar period, Calendar sinceDate, Short state,
            String propertyName, Boolean isDesc, Integer startIndex,
      Integer count) throws InternalErrorException {

        try{

            FindAdvancedAction  action =
                    new FindAdvancedAction(loginName, autoId, period,
                  sinceDate, state, propertyName, isDesc,
            startIndex, count);

            return (Collection)
                    PlainActionProcessor.process(getDataSource(),
                  action);

        } catch (Exception e) {
            throw new InternalErrorException(e);
        }
    }

// ADMINISTRATOR ACTIONS

    /**
```

```
 * Updates state of a holidays request.
 * @param autoId
 * @param state
 * @throws
 * application.util.exceptions.OperationNotAllowedException
 * @throws
 * application.util.exceptions.AdministratorRequiredException
 * @throws application.util.exceptions.InstanceNotFoundException
 * @throws application.util.exceptions.InternalErrorException
 */
public void updateHolidaysRequest(Integer autoId, Short state)
  throws
        AdministratorRequiredException, InstanceNotFoundException,
        InternalErrorException, OperationNotAllowedException {

    try{

        if (isAdministrator) {

        UpdateHolidaysRequestAction action =
                new UpdateHolidaysRequestAction(autoId, state);

        HolidaysRequestVO holidaysRequestVO = (HolidaysRequestVO)
                PlainActionProcessor.process(getDataSource(),
              action);
        } else {
            throw new
              AdministratorRequiredException(mainLoginName);
        }
    } catch (AdministratorRequiredException e) {
        throw e;
    }  catch (InstanceNotFoundException e) {
        throw e;
    } catch (OperationNotAllowedException e) {
        throw e;
    } catch (Exception e) {
        throw new InternalErrorException(e);
    }

}

/**
 * Finds a holidays request by its identificator.
 * @return
 * @param autoId
 * @throws
 * application.util.exceptions.AdministratorRequiredException
 * @throws application.util.exceptions.InstanceNotFoundException
 * @throws application.util.exceptions.InternalErrorException
 */
public HolidaysRequestVO findIdHolidaysRequest(Integer autoId)
  throws AdministratorRequiredException,
  InstanceNotFoundException, InternalErrorException {

    try {
        if (isAdministrator) {

        FindIdHolidaysRequestAction action =
                new FindIdHolidaysRequestAction(autoId);

        return (HolidaysRequestVO)
        PlainActionProcessor.process(getDataSource(), action);
        } else {
            throw new
               AdministratorRequiredException(mainLoginName);
        }
    } catch (AdministratorRequiredException e) {
        throw e;
    } catch (InstanceNotFoundException e) {
        throw e;
    } catch (Exception e) {
        throw new InternalErrorException(e);
    }
}

/**
 * Carries out an advanced search.
 * @return
 * @param isUserLoaded
 * @param autoId
 * @param period
 * @param sinceDate
 * @param state
 * @param propertyName
 * @param isDesc
 * @param startIndex
 * @param count
 * @throws
 * application.util.exceptions.AdministratorRequiredException
 * @throws application.util.exceptions.InternalErrorException
 */
public Collection findAdvanced(Boolean isUserLoaded, Integer
  autoId, Calendar period, Calendar sinceDate, Short state,
        String propertyName, Boolean isDesc, Integer startIndex,
  Integer count) throws AdministratorRequiredException,
  InternalErrorException {

    try{

        if (isAdministrator) {
              String user="%";

        if(isUserLoaded) {
            user=this.loginName;
        }

        FindAdvancedAction  action =
                new FindAdvancedAction(user, autoId, period,
              sinceDate, state, propertyName, isDesc,
        startIndex, count);
```

```
            return (Collection)
                    PlainActionProcessor.process(getDataSource(),
                  action);
            } else {
                throw new
                   AdministratorRequiredException(mainLoginName);
            }
        } catch (AdministratorRequiredException e) {
            throw e;
        } catch (Exception e) {
            throw new InternalErrorException(e);
        }
    }

//
// HOLIDAYS PROFILE
//-----------------

    /**
     * Returns all user holidays profiles.
     * @param startIndex
     * @param count
     * @throws application.util.exceptions.InternalErrorException
     * @return
     */
    public Collection findHolidaysProfile(Integer startIndex, Integer
      count) throws InternalErrorException {

        try {

            /* Finds all user holidays services. */
            FindUserHolidaysServicesAction findAction =
                    new FindUserHolidaysServicesAction(loginName,
                  startIndex, count);

            Collection holidaysServiceVOs =
                    (Collection)
                  PlainActionProcessor.process(getDataSource(),
                              findAction);

            /* Creates holidays profiles for diferent periods, */
            Collection<HolidaysProfileVO> holidaysProfileVOs = new
            ArrayList<HolidaysProfileVO>();


            if(holidaysServiceVOs.size()>0){

                /* Finds holidays profile for each period. */
                for(Iterator iterator =
                        holidaysServiceVOs.iterator();
          iterator.hasNext();) {

                    HolidaysServiceVO holidaysServiceVO =
                            (HolidaysServiceVO) iterator.next();

                    Calendar period = holidaysServiceVO.getPeriod();
                    Integer allDaysOfHolidays =
                  holidaysServiceVO.getDays();

                    /* Finds all accepted user holidays request for
                     * each period. */
                    FindUserPeriodStateHolidaysRequestByAction  action
                       = new
                      FindUserPeriodStateHolidaysRequestByAction(
                             loginName, period, Short.valueOf("1"),
                         HolidaysRequestTable.FINALDATE, 1,
                             0);

                    Collection holidaysRequestVOs = (Collection)
                         PlainActionProcessor.process(getDataSource(),
                            action);

                    /* Calculates workingDays taken as holidays days.
               */
                    int workingDays = 0;
                    int holidaysTaken = 0;

                    if(holidaysRequestVOs.size()>0) {

                        Calendar previousStartDate = null;
                        Calendar previousFinalDate = null;

                        for (Iterator iterator1 =
                       holidaysRequestVOs.iterator();
                              iterator1.hasNext(); ) {

                            HolidaysRequestVO holidaysRequestVO =
                                    (HolidaysRequestVO)
                                   iterator1.next();

                            Calendar startDate =
                                    holidaysRequestVO.getStartDate();
                            Calendar finalDate =
                                    holidaysRequestVO.getFinalDate();

                            if(previousStartDate != null ||
                                    previousFinalDate != null) {

                              if(!startDate.before(previousStartDate))
                              {
                                    workingDays = 0;
                                } else {

                         if(!finalDate.before(previousStartDate))
                              {
                                        previousFinalDate =
                                   previousStartDate;
                                    } else {
                                        previousFinalDate = finalDate;
                                    }

                                    previousStartDate = startDate;
```

```
                              workingDays =
                             Jworkingdays.workingdays(
                                     previousStartDate,
                                     previousFinalDate);

                         }
                     } else {

                         previousStartDate = startDate;
                         previousFinalDate = finalDate;
                         workingDays =
                       Jworkingdays.workingdays(
                          previousStartDate,previousFinalDate);
                     }

                     holidaysTaken = holidaysTaken +
                 workingDays;
                  }
               }

               /* Calculates holidays left. */
               Integer holidaysLeft = allDaysOfHolidays -
             holidaysTaken;

               /* Add holidays profile. */
               holidaysProfileVOs.add(new
             HolidaysProfileVO(loginName,
                     period, allDaysOfHolidays,holidaysLeft));
           }
       }

       return holidaysProfileVOs;

   }catch (InternalErrorException e) {
       throw e;
   } catch (Exception e) {
       throw new InternalErrorException(e);
   }
}

/**
 *
 * @param period
 * @throws application.util.exceptions.InstanceNotFoundException
 * @throws application.util.exceptions.InternalErrorException
 * @return
 */
public HolidaysProfileVO findHolidaysProfilePeriod(Calendar
  period)
        throws InstanceNotFoundException, InternalErrorException {

    try {


        /* Finds user holidays profile. */
        FindHolidaysServiceAction findAction =
               new FindHolidaysServiceAction(loginName, period);

       HolidaysServiceVO holidaysServiceVO = (HolidaysServiceVO)
               PlainActionProcessor.process(getDataSource(),
             findAction);

       Integer allDaysOfHolidays = holidaysServiceVO.getDays();

       /* Finds all accepted user holidays request in this
  * period. */
       FindUserPeriodStateHolidaysRequestByAction  action =
               new
            FindUserPeriodStateHolidaysRequestByAction(loginName,
                 period, Short.valueOf("1"),
             HolidaysRequestTable.FINALDATE, 1, 0);

       Collection holidaysRequestVOs = (Collection)
               PlainActionProcessor.process(getDataSource(),
             action);

       /* Calculates working days taken as holidays days. */
       int workingDays = 0;
       int holidaysTaken = 0;

       if(holidaysRequestVOs.size()>0) {

           Calendar previousStartDate = null;
           Calendar previousFinalDate = null;

           for (Iterator iterator1 =
                  holidaysRequestVOs.iterator();
        iterator1.hasNext(); ) {

               HolidaysRequestVO holidaysRequestVO =
                     (HolidaysRequestVO) iterator1.next();

               Calendar startDate =
             holidaysRequestVO.getStartDate();
               Calendar finalDate =
             holidaysRequestVO.getFinalDate();

               if(previousStartDate != null || previousFinalDate
             != null){

                  if(!startDate.before(previousStartDate)) {
                     workingDays = 0;
                  } else {

                     if(!finalDate.before(previousStartDate)) {
                         previousFinalDate = previousStartDate;
                     } else {
                         previousFinalDate = finalDate;
                     }

                     previousStartDate = startDate;
                     workingDays = Jworkingdays.workingdays(
                                 previousStartDate,
                         previousFinalDate);
                   }
               } else {

                   previousStartDate = startDate;
                   previousFinalDate = finalDate;
                   workingDays = Jworkingdays.workingdays(
                           previousStartDate, previousFinalDate);
               }

               holidaysTaken = holidaysTaken + workingDays;
             }
         }

         /* Calculates holidays left days. */
         int holidaysLeft = allDaysOfHolidays - holidaysTaken;
         HolidaysProfileVO holidaysProfileVO = new
         HolidaysProfileVO(
                    loginName, period, allDaysOfHolidays,
               holidaysLeft);

         /* Returns holidays profile. */
         return holidaysProfileVO;



     }  catch (InstanceNotFoundException e) {
         throw e;
     }catch (InternalErrorException e) {
         throw e;
     } catch (Exception e) {
         throw new InternalErrorException(e);
     }

 }

 /**
  * Retunrs holidays service profile in a period.
  * @return
  * @param userName
  * @param period
  * @throws
  * application.util.exceptions.AdministratorRequiredException
  * @throws application.util.exceptions.InstanceNotFoundException
  * @throws application.util.exceptions.InternalErrorException
  */
 public HolidaysProfileVO findUserHolidaysProfilePeriod(String
   userName, Calendar period) throws
   AdministratorRequiredException,
         InstanceNotFoundException, InternalErrorException {

     try {

         if (isAdministrator) {
           /* Finds user holidays profile. */
           FindHolidaysServiceAction findAction =
                  new FindHolidaysServiceAction(userName, period);

           HolidaysServiceVO holidaysServiceVO = (HolidaysServiceVO)
                  PlainActionProcessor.process(getDataSource(),
               findAction);

           Integer allDaysOfHolidays = holidaysServiceVO.getDays();

           /* Finds all accepted user holidays request in this
* period. */
           FindUserPeriodStateHolidaysRequestByAction  action =
                  new
               FindUserPeriodStateHolidaysRequestByAction(userName,
                     period, Short.valueOf("1"),
               HolidaysRequestTable.FINALDATE, 1, 0);

           Collection holidaysRequestVOs = (Collection)
                  PlainActionProcessor.process(getDataSource(),
               action);

           /* Calculates working days taken as holidays days. */
           int workingDays = 0;
           int holidaysTaken = 0;

           if(holidaysRequestVOs.size()>0) {

               Calendar previousStartDate = null;
               Calendar previousFinalDate = null;

               for (Iterator iterator1 =
                     holidaysRequestVOs.iterator();
               iterator1.hasNext(); ) {

                  HolidaysRequestVO holidaysRequestVO =
                         (HolidaysRequestVO) iterator1.next();

                  Calendar startDate =
               holidaysRequestVO.getStartDate();
                  Calendar finalDate =
               holidaysRequestVO.getFinalDate();

                  if(previousStartDate != null || previousFinalDate
               != null){

                     if(!startDate.before(previousStartDate)) {
                         workingDays = 0;
                     } else {

                         if(!finalDate.before(previousStartDate)) {
                             previousFinalDate = previousStartDate;
                         } else {
                             previousFinalDate = finalDate;
                         }
                        previousStartDate = startDate;
                        workingDays = Jworkingdays.workingdays(
                                previousStartDate,
                         previousFinalDate);
                    }
                } else {

                    previousStartDate = startDate;
                    previousFinalDate = finalDate;
                    workingDays = Jworkingdays.workingdays(
                            previousStartDate, previousFinalDate);
                }

                holidaysTaken = holidaysTaken + workingDays;
            }
        }

        /* Calculates holidays left days. */
        int holidaysLeft = allDaysOfHolidays - holidaysTaken;
        HolidaysProfileVO holidaysProfileVO = new
        HolidaysProfileVO(
                    userName, period, allDaysOfHolidays,
              holidaysLeft);

        /* Returns holidays profile. */
        return holidaysProfileVO;

        } else {
            throw new
              AdministratorRequiredException(mainLoginName);
        }

    } catch (AdministratorRequiredException e) {
        throw e;
    } catch (InstanceNotFoundException e) {
        throw e;
    }catch (InternalErrorException e) {
        throw e;
    } catch (Exception e) {
        throw new InternalErrorException(e);
    }

}

/**
 * Finds periods with holidays profiles.
 * @return
 * @param startIndex
 * @param count
 * @throws
 * application.util.exceptions.AdministratorRequiredException
 * @throws application.util.exceptions.InternalErrorException
 */
public Collection findHolidaysProfiles(Integer startIndex, Integer
        count) throws AdministratorRequiredException,
  InternalErrorException{
    try {

        if (isAdministrator) {
        FindHolidaysProfilesAction action =
                new FindHolidaysProfilesAction(startIndex, count);

        return (Collection)
        PlainActionProcessor.process(getDataSource(),
                action);
        } else {
            throw new
              AdministratorRequiredException(mainLoginName);
        }

    } catch (AdministratorRequiredException e) {
        throw e;
    } catch (InternalErrorException e) {
        throw e;
    } catch (Exception e) {
        throw new InternalErrorException(e);
    }
}

/**
 * Retunrs all holidays profiles in a period.
 * @return
 * @param period
 * @param startIndex
 * @param count
 * @throws
 * application.util.exceptions.AdministratorRequiredException
 * @throws application.util.exceptions.InternalErrorException
 */
public Collection findPeriodHolidaysProfile(Calendar period,
        Integer startIndex, Integer count) throws
        AdministratorRequiredException, InternalErrorException {

    try {

        if (isAdministrator) {

        /* Finds all holidays services in a period. */
        FindPeriodHolidaysServicesAction findAction =
                new FindPeriodHolidaysServicesAction(period,
              startIndex, count);

        Collection holidaysServiceVOs =
                (Collection)
              PlainActionProcessor.process(getDataSource(),
                        findAction);

        /* Creates holidays profiles. */
        Collection<HolidaysProfileVO> holidaysProfileVOs = new
        ArrayList<HolidaysProfileVO>();
```

```
if(holidaysServiceVOs.size()>0){

    /* Finds holidays profiles for each holidays service.
*/
    for(Iterator iterator =
            holidaysServiceVOs.iterator();
      iterator.hasNext();) {

        HolidaysServiceVO holidaysServiceVO =
                (HolidaysServiceVO) iterator.next();

        String userName =
      holidaysServiceVO.getLoginName();
        Integer allDaysOfHolidays =
      holidaysServiceVO.getDays();

        /* Finds all accepted user holidays request for
         * each holidays service. */
        FindUserPeriodStateHolidaysRequestByAction  action
            = new
         FindUserPeriodStateHolidaysRequestByAction(
                userName, period, Short.valueOf("1"),
                HolidaysRequestTable.FINALDATE, 1, 0);

        Collection holidaysRequestVOs = (Collection)
             PlainActionProcessor.process(getDataSource(),
                action);

        /* Calculates working days taken as holidays days.
          */
        int workingDays = 0;
        int holidaysTaken = 0;

        if(holidaysRequestVOs.size()>0) {

            Calendar previousStartDate = null;
            Calendar previousFinalDate = null;

            for (Iterator iterator1 =
           holidaysRequestVOs.iterator();
                  iterator1.hasNext(); ) {

                HolidaysRequestVO holidaysRequestVO =
                        (HolidaysRequestVO)
                       iterator1.next();

                Calendar startDate =
                        holidaysRequestVO.getStartDate();
                Calendar finalDate =
                        holidaysRequestVO.getFinalDate();

                if(previousStartDate != null ||
                        previousFinalDate != null) {

                  if(!startDate.before(previousStartDate))
                 {
                        workingDays = 0;
                    } else {

                if(!finalDate.before(previousStartDate))
          {
                            previousFinalDate =
                       previousStartDate;
                        } else {
                            previousFinalDate = finalDate;
                        }

                        previousStartDate = startDate;
                        workingDays =
                       Jworkingdays.workingdays(
                                previousStartDate,
                                previousFinalDate);

                    }
                } else {

                    previousStartDate = startDate;
                    previousFinalDate = finalDate;
                    workingDays =
                  Jworkingdays.workingdays(
                              previousStartDate,
                              previousFinalDate);
                }

                holidaysTaken = holidaysTaken +
                  workingDays;
            }
        }

        /* Calculates holidays left days. */
        Integer holidaysLeft = allDaysOfHolidays -
      holidaysTaken;

        /* Adds holidays profile. */
        holidaysProfileVOs.add(new
      HolidaysProfileVO(userName,
                period, allDaysOfHolidays,holidaysLeft));
      }
    }

    return holidaysProfileVOs;

    } else {
        throw new
          AdministratorRequiredException(mainLoginName);
    }
} catch (AdministratorRequiredException e) {
    throw e;
}catch (InternalErrorException e) {
    throw e;
} catch (Exception e) {
    throw new InternalErrorException(e);
```

```
        }

        }

//UTILITIES

    /**
     * Initiates an user holidays request service from an
     * <code>UserProfileServiceFacade</code>.
     * @param userFacadeDelegate
     */
    public void initHolidaysRequestService(UserProfileServiceFacade
            userFacadeDelegate) {

        this.mainLoginName= userFacadeDelegate.getMainLoginName();
        this.loginName = userFacadeDelegate.getLoginName();
        this.isAdministrator =
      userFacadeDelegate.getIsAdministrator();
    }

    /**
     * Gets datasource using JDNI.
     * @throws application.util.exceptions.InternalErrorException
     * @return
     */
    private DataSource getDataSource() throws InternalErrorException {
            return DataSourceLocator.getDataSource(
                  GlobalNames.APPLICATION_DATA_SOURCE);
    }

}
```

10.5.3.4 Operaciones

10.5.3.4.a Servicio de perfil de usuario : encriptador de claves

10.5.3.4.a.1 application.model.userprofileservice.operations

PasswordEncrypter

```
package application.model.userprofileservice.operations;
```

```
public class PasswordEncrypter {

      private PasswordEncrypter() {}

      public final static String crypt(String clearPassword) {

            /*
             * "jcrypt" only considers the first 8 characters of the
             * clear text, and generates an encrypted text that is
       * always 11 characters in length. The first two characters
       * of the encrypted text are always the first two
* characters of the salt (the first parameter of               *
"jcrypt"). If the length of the salt is lower than 2,
             * the salt is completed with a default character ('A').
       * This salt is used to perturb the algorithm in one of
* 4096 different ways. Check
             * "man crypt" on a Unix machine and the implementation of
             * "jcrypt" for further details.
             *
             * The implementation of this method uses the clear
       * password as the salt. The first two characters of the
* string returned by "jcrypt" are removed (they are the           *
first two characters of the clear password).
             */
            String encryptedPasswordWithSalt =
            Jcrypt.crypt(clearPassword, clearPassword);
            String encryptedPassword =
            encryptedPasswordWithSalt.substring(2);

            return encryptedPassword;

      }

}
```

Jcrypt

```
package application.model.userprofileservice.operations;

/**
 * This utility class is a implementation of the C source code written
 * by Eric Young (eay@psych.uq.oz.au) of the UNIX crypt command.
 * <p>
 */
public final class Jcrypt
{
   private Jcrypt() {}

   private static final int ITERATIONS = 16;

   private static final int con_salt[] =
   {
      0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
      0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
      0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
      0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
      0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
      0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x01,
      0x02, 0x03, 0x04, 0x05, 0x06, 0x07, 0x08, 0x09,
      0x0A, 0x0B, 0x05, 0x06, 0x07, 0x08, 0x09, 0x0A,
      0x0B, 0x0C, 0x0D, 0x0E, 0x0F, 0x10, 0x11, 0x12,
      0x13, 0x14, 0x15, 0x16, 0x17, 0x18, 0x19, 0x1A,
      0x1B, 0x1C, 0x1D, 0x1E, 0x1F, 0x20, 0x21, 0x22,
      0x23, 0x24, 0x25, 0x20, 0x21, 0x22, 0x23, 0x24,
      0x25, 0x26, 0x27, 0x28, 0x29, 0x2A, 0x2B, 0x2C,
      0x2D, 0x2E, 0x2F, 0x30, 0x31, 0x32, 0x33, 0x34,
      0x35, 0x36, 0x37, 0x38, 0x39, 0x3A, 0x3B, 0x3C,
      0x3D, 0x3E, 0x3F, 0x00, 0x00, 0x00, 0x00, 0x00,
   };

   private static final boolean shifts2[] =
   {
      false, false, true, true, true, true, true, true,
      false, true,  true, true, true, true, true, false
   };

   private static final int skb[][] =
   {
      {
         /* for C bits (numbered as per FIPS 46) 1 2 3 4 5 6 */
         0x00000000, 0x00000010, 0x20000000, 0x20000010,
         0x00010000, 0x00010010, 0x20010000, 0x20010010,
         0x00000800, 0x00000810, 0x20000800, 0x20000810,
         0x00010800, 0x00010810, 0x20010800, 0x20010810,
         0x00000020, 0x00000030, 0x20000020, 0x20000030,
         0x00010020, 0x00010030, 0x20010020, 0x20010030,
         0x00000820, 0x00000830, 0x20000820, 0x20000830,
         0x00010820, 0x00010830, 0x20010820, 0x20010830,
         0x00080000, 0x00080010, 0x20080000, 0x20080010,
         0x00090000, 0x00090010, 0x20090000, 0x20090010,
         0x00080800, 0x00080810, 0x20080800, 0x20080810,
         0x00090800, 0x00090810, 0x20090800, 0x20090810,
         0x00080020, 0x00080030, 0x20080020, 0x20080030,
         0x00090020, 0x00090030, 0x20090020, 0x20090030,
         0x00080820, 0x00080830, 0x20080820, 0x20080830,
         0x00090820, 0x00090830, 0x20090820, 0x20090830,
      },
      {
         /* for C bits (numbered as per FIPS 46) 7 8 10 11 12 13 */
         0x00000000, 0x02000000, 0x00002000, 0x02002000,
         0x00200000, 0x02200000, 0x00202000, 0x02202000,
         0x00000004, 0x02000004, 0x00002004, 0x02002004,
         0x00200004, 0x02200004, 0x00202004, 0x02202004,
         0x00000400, 0x02000400, 0x00002400, 0x02002400,
         0x00200400, 0x02200400, 0x00202400, 0x02202400,
         0x00000404, 0x02000404, 0x00002404, 0x02002404,
         0x00200404, 0x02200404, 0x00202404, 0x02202404,
         0x10000000, 0x12000000, 0x10002000, 0x12002000,
         0x10200000, 0x12200000, 0x10202000, 0x12202000,
         0x10000004, 0x12000004, 0x10002004, 0x12002004,
         0x10200004, 0x12200004, 0x10202004, 0x12202004,
         0x10000400, 0x12000400, 0x10002400, 0x12002400,
         0x10200400, 0x12200400, 0x10202400, 0x12202400,
         0x10000404, 0x12000404, 0x10002404, 0x12002404,
         0x10200404, 0x12200404, 0x10202404, 0x12202404,
      },
      {
         /* for C bits (numbered as per FIPS 46) 14 15 16 17 19 20 */
         0x00000000, 0x00000001, 0x00040000, 0x00040001,
         0x01000000, 0x01000001, 0x01040000, 0x01040001,
         0x00000002, 0x00000003, 0x00040002, 0x00040003,
         0x01000002, 0x01000003, 0x01040002, 0x01040003,
         0x00000200, 0x00000201, 0x00040200, 0x00040201,
         0x01000200, 0x01000201, 0x01040200, 0x01040201,
         0x00000202, 0x00000203, 0x00040202, 0x00040203,
         0x01000202, 0x01000203, 0x01040202, 0x01040203,
         0x08000000, 0x08000001, 0x08040000, 0x08040001,
         0x09000000, 0x09000001, 0x09040000, 0x09040001,
         0x08000002, 0x08000003, 0x08040002, 0x08040003,
         0x09000002, 0x09000003, 0x09040002, 0x09040003,
         0x08000200, 0x08000201, 0x08040200, 0x08040201,
         0x09000200, 0x09000201, 0x09040200, 0x09040201,
         0x08000202, 0x08000203, 0x08040202, 0x08040203,
         0x09000202, 0x09000203, 0x09040202, 0x09040203,
      },
      {
         /* for C bits (numbered as per FIPS 46) 21 23 24 26 27 28 */
         0x00000000, 0x00100000, 0x00000100, 0x00100100,
         0x00000008, 0x00100008, 0x00000108, 0x00100108,
         0x00001000, 0x00101000, 0x00001100, 0x00101100,
         0x00001008, 0x00101008, 0x00001108, 0x00101108,
         0x04000000, 0x04100000, 0x04000100, 0x04100100,
         0x04000008, 0x04100008, 0x04000108, 0x04100108,
         0x04001000, 0x04101000, 0x04001100, 0x04101100,
         0x04001008, 0x04101008, 0x04001108, 0x04101108,
         0x00020000, 0x00120000, 0x00020100, 0x00120100,
         0x00020008, 0x00120008, 0x00020108, 0x00120108,
         0x00021000, 0x00121000, 0x00021100, 0x00121100,
         0x00021008, 0x00121008, 0x00021108, 0x00121108,
         0x04020000, 0x04120000, 0x04020100, 0x04120100,
         0x04020008, 0x04120008, 0x04020108, 0x04120108,
         0x04021000, 0x04121000, 0x04021100, 0x04121100,
         0x04021008, 0x04121008, 0x04021108, 0x04121108,
      },
      {
         /* for D bits (numbered as per FIPS 46) 1 2 3 4 5 6 */
         0x00000000, 0x10000000, 0x00010000, 0x10010000,
         0x00000004, 0x10000004, 0x00010004, 0x10010004,
         0x20000000, 0x30000000, 0x20010000, 0x30010000,
         0x20000004, 0x30000004, 0x20010004, 0x30010004,
         0x00100000, 0x10100000, 0x00110000, 0x10110000,
         0x00100004, 0x10100004, 0x00110004, 0x10110004,
         0x20100000, 0x30100000, 0x20110000, 0x30110000,
         0x20100004, 0x30100004, 0x20110004, 0x30110004,
         0x00001000, 0x10001000, 0x00011000, 0x10011000,
         0x00001004, 0x10001004, 0x00011004, 0x10011004,
         0x20001000, 0x30001000, 0x20011000, 0x30011000,
         0x20001004, 0x30001004, 0x20011004, 0x30011004,
         0x00101000, 0x10101000, 0x00111000, 0x10111000,
         0x00101004, 0x10101004, 0x00111004, 0x10111004,
         0x20101000, 0x30101000, 0x20111000, 0x30111000,
         0x20101004, 0x30101004, 0x20111004, 0x30111004,
      },
      {
         /* for D bits (numbered as per FIPS 46) 8 9 11 12 13 14 */
         0x00000000, 0x08000000, 0x00000008, 0x08000008,
         0x00000400, 0x08000400, 0x00000408, 0x08000408,
         0x00020000, 0x08020000, 0x00020008, 0x08020008,
         0x00020400, 0x08020400, 0x00020408, 0x08020408,
         0x00000001, 0x08000001, 0x00000009, 0x08000009,
         0x00000401, 0x08000401, 0x00000409, 0x08000409,
         0x00020001, 0x08020001, 0x00020009, 0x08020009,
         0x00020401, 0x08020401, 0x00020409, 0x08020409,
         0x02000000, 0x0A000000, 0x02000008, 0x0A000008,
         0x02000400, 0x0A000400, 0x02000408, 0x0A000408,
         0x02020000, 0x0A020000, 0x02020008, 0x0A020008,
         0x02020400, 0x0A020400, 0x02020408, 0x0A020408,
         0x02000001, 0x0A000001, 0x02000009, 0x0A000009,
         0x02000401, 0x0A000401, 0x02000409, 0x0A000409,
         0x02020001, 0x0A020001, 0x02020009, 0x0A020009,
         0x02020401, 0x0A020401, 0x02020409, 0x0A020409,
      },
      {
         /* for D bits (numbered as per FIPS 46) 16 17 18 19 20 21 */
         0x00000000, 0x00000100, 0x00080000, 0x00080100,
         0x01000000, 0x01000100, 0x01080000, 0x01080100,
         0x00000010, 0x00000110, 0x00080010, 0x00080110,
         0x01000010, 0x01000110, 0x01080010, 0x01080110,
         0x00200000, 0x00200100, 0x00280000, 0x00280100,
         0x01200000, 0x01200100, 0x01280000, 0x01280100,
         0x00200010, 0x00200110, 0x00280010, 0x00280110,
         0x01200010, 0x01200110, 0x01280010, 0x01280110,
         0x00000200, 0x00000300, 0x00080200, 0x00080300,
         0x01000200, 0x01000300, 0x01080200, 0x01080300,
         0x00000210, 0x00000310, 0x00080210, 0x00080310,
         0x01000210, 0x01000310, 0x01080210, 0x01080310,
         0x00200200, 0x00200300, 0x00280200, 0x00280300,
         0x01200200, 0x01200300, 0x01280200, 0x01280300,
         0x00200210, 0x00200310, 0x00280210, 0x00280310,
         0x01200210, 0x01200310, 0x01280210, 0x01280310,
      },
      {
         /* for D bits (numbered as per FIPS 46) 22 23 24 25 27 28 */
         0x00000000, 0x04000000, 0x00040000, 0x04040000,
         0x00000002, 0x04000002, 0x00040002, 0x04040002,
         0x00002000, 0x04002000, 0x00042000, 0x04042000,
         0x00002002, 0x04002002, 0x00042002, 0x04042002,
         0x00000020, 0x04000020, 0x00040020, 0x04040020,
         0x00000022, 0x04000022, 0x00040022, 0x04040022,
         0x00002020, 0x04002020, 0x00042020, 0x04042020,
         0x00002022, 0x04002022, 0x00042022, 0x04042022,
         0x00000800, 0x04000800, 0x00040800, 0x04040800,
         0x00000802, 0x04000802, 0x00040802, 0x04040802,
         0x00002800, 0x04002800, 0x00042800, 0x04042800,
         0x00002802, 0x04002802, 0x00042802, 0x04042802,
         0x00000820, 0x04000820, 0x00040820, 0x04040820,
         0x00000822, 0x04000822, 0x00040822, 0x04040822,
         0x00002820, 0x04002820, 0x00042820, 0x04042820,
         0x00002822, 0x04002822, 0x00042822, 0x04042822,
      },
   };

   private static final int SPtrans[][] =
   {
      {
         /* nibble 0 */
         0x00820200, 0x00020000, 0x80800000, 0x80820200,
         0x00800000, 0x80020200, 0x80020000, 0x80800000,
         0x80020200, 0x00820200, 0x00820000, 0x80000200,
         0x80800200, 0x00800000, 0x00000000, 0x80020000,
         0x00020000, 0x80000000, 0x00800200, 0x00020200,
         0x80820200, 0x00820000, 0x80000200, 0x00800200,
         0x80000000, 0x00000200, 0x00020200, 0x80820000,
         0x00000200, 0x80800200, 0x80820000, 0x00000000,
         0x00000000, 0x80820200, 0x00800200, 0x80020000,
         0x00820200, 0x00020000, 0x80000200, 0x00800200,
         0x80820000, 0x00000200, 0x00020200, 0x80800000,
         0x80020200, 0x80000000, 0x80800000, 0x00820000,
         0x80820200, 0x00020200, 0x00820000, 0x80800200,
         0x00800000, 0x80000200, 0x80020000, 0x00000000,
         0x00020000, 0x00800000, 0x80800200, 0x00820200,
         0x80000000, 0x80820000, 0x00000200, 0x80020200,
      },
      {
         /* nibble 1 */
         0x10042004, 0x00000000, 0x00042000, 0x10040000,
         0x10000004, 0x00002004, 0x10002000, 0x00042000,
         0x00002000, 0x10040004, 0x00000004, 0x10002000,
         0x00040004, 0x10042000, 0x10040000, 0x00000004,
         0x00040000, 0x10002004, 0x10040004, 0x00002000,
         0x00042004, 0x10000000, 0x00000000, 0x00040004,
         0x10002004, 0x00042004, 0x10042000, 0x10000004,
         0x10000000, 0x00040000, 0x00002004, 0x10042004,
         0x00040004, 0x10042000, 0x10002000, 0x00042004,
         0x10042004, 0x00040004, 0x10000004, 0x00000000,
         0x10000000, 0x00002004, 0x00040000, 0x10040004,
         0x00002000, 0x10000000, 0x00042004, 0x10002004,
         0x10042000, 0x00002000, 0x00000000, 0x10000004,
         0x00000004, 0x10042004, 0x00042000, 0x10040000,
         0x10040004, 0x00040000, 0x00002004, 0x10002000,
         0x10002004, 0x00000004, 0x10040000, 0x00042000,
      },
      {
         /* nibble 2 */
         0x41000000, 0x01010040, 0x00000040, 0x41000040,
         0x40010000, 0x01000000, 0x41000040, 0x00010040,
         0x01000040, 0x00010000, 0x01010000, 0x40000000,
         0x41010040, 0x40000040, 0x40000000, 0x41010000,
         0x00000000, 0x40010000, 0x01010040, 0x00000040,
         0x40000040, 0x41010040, 0x00010000, 0x41000000,
         0x41010000, 0x01000040, 0x40010040, 0x01010000,
         0x00010040, 0x00000000, 0x01000000, 0x40010040,
         0x01010040, 0x00000040, 0x40000000, 0x00010000,
         0x40000040, 0x40010000, 0x01010000, 0x41000040,
         0x00000000, 0x01010040, 0x00010040, 0x41010000,
         0x40010000, 0x01000000, 0x41010040, 0x40000000,
         0x40010040, 0x41000000, 0x01000000, 0x41010040,
         0x00010000, 0x01000040, 0x41000040, 0x00010040,
         0x01000040, 0x00000000, 0x41010000, 0x40000040,
         0x41000000, 0x40010040, 0x00000040, 0x01010000,
      },
      {
         /* nibble 3 */
         0x00100402, 0x04000400, 0x00000002, 0x04100402,
         0x00000000, 0x04100000, 0x04000402, 0x00100002,
         0x04100400, 0x04000002, 0x04000000, 0x00000402,
         0x04000002, 0x00100402, 0x00100000, 0x04000000,
         0x04100002, 0x00100400, 0x00000400, 0x00000002,
         0x00100400, 0x04000402, 0x04100000, 0x00000400,
         0x00000402, 0x00000000, 0x00100002, 0x04100400,
         0x04000400, 0x04100002, 0x04100402, 0x00100000,
         0x04100002, 0x00000402, 0x00100000, 0x04000002,
         0x00100400, 0x04000400, 0x00000002, 0x04100000,
         0x04000402, 0x00000000, 0x00000400, 0x00100002,
         0x00000000, 0x04100002, 0x04100400, 0x00000400,
         0x04000000, 0x04100402, 0x00100402, 0x00100000,
         0x04100402, 0x00000002, 0x04000400, 0x00100402,
         0x00100002, 0x00100400, 0x04100000, 0x04000402,
         0x00000402, 0x04000000, 0x04000002, 0x04100400,
      },
      {
         /* nibble 4 */
         0x02000000, 0x00004000, 0x00000100, 0x02004108,
         0x02004008, 0x02000100, 0x00004108, 0x02004000,
         0x00004000, 0x00000008, 0x02000008, 0x00004100,
         0x02000108, 0x02004008, 0x02004100, 0x00000000,
         0x00004100, 0x02000000, 0x00004008, 0x00000108,
         0x02000100, 0x00004108, 0x00000000, 0x02000008,
         0x00000008, 0x02000108, 0x02004108, 0x00004008,
         0x02004000, 0x00000100, 0x00000108, 0x02004100,
         0x02004100, 0x02000108, 0x00004008, 0x02004000,
         0x00004000, 0x00000008, 0x02000008, 0x02000100,
         0x02000000, 0x00004100, 0x02004108, 0x00000000,
         0x00004108, 0x02000000, 0x00000100, 0x00004008,
         0x02000108, 0x00000100, 0x00000000, 0x02004108,
         0x02004008, 0x02004100, 0x00000108, 0x00004000,
         0x00004100, 0x02004008, 0x02000100, 0x00000108,
         0x00000008, 0x00004108, 0x02004000, 0x02000008,
      },
      {
         /* nibble 5 */
         0x20000010, 0x00080010, 0x00000000, 0x20080800,
         0x00080010, 0x00000800, 0x20000810, 0x00080000,
         0x00000810, 0x20080810, 0x00080800, 0x20000000,
         0x20000800, 0x20000010, 0x20080000, 0x00080810,
         0x00080000, 0x20000810, 0x20080010, 0x00000000,
         0x00000800, 0x00000010, 0x20080800, 0x20080010,
         0x20080810, 0x20080000, 0x20000000, 0x00000810,
         0x00000010, 0x00080800, 0x00080810, 0x20000800,
         0x00000810, 0x20000000, 0x20000800, 0x00080810,
         0x20080800, 0x00080010, 0x00000000, 0x20000800,
         0x20000000, 0x00000800, 0x20080010, 0x00080000,
         0x00080010, 0x20080810, 0x00080800, 0x00000010,
         0x20080810, 0x00080800, 0x00080000, 0x20000810,
         0x20000010, 0x20080000, 0x00080810, 0x00000000,
         0x00000800, 0x20000010, 0x20000810, 0x20080800,
         0x20080000, 0x00000810, 0x00000010, 0x20080010,
      },
      {
         /* nibble 6 */
         0x00001000, 0x00000080, 0x00400080, 0x00400001,
         0x00401081, 0x00001001, 0x00001080, 0x00000000,
         0x00400000, 0x00400081, 0x00000081, 0x00401000,
         0x00000001, 0x00401080, 0x00401000, 0x00000081,
         0x00400081, 0x00001000, 0x00001001, 0x00401081,
         0x00000000, 0x00400080, 0x00400001, 0x00001080,
         0x00401001, 0x00001081, 0x00401080, 0x00000001,
         0x00001081, 0x00401001, 0x00000080, 0x00400000,
         0x00001081, 0x00401000, 0x00401001, 0x00000081,
         0x00001000, 0x00000080, 0x00400000, 0x00401001,
         0x00400081, 0x00001081, 0x00001080, 0x00000000,
         0x00000080, 0x00400001, 0x00000001, 0x00400080,
         0x00000000, 0x00400081, 0x00400080, 0x00001080,
         0x00000081, 0x00001000, 0x00401081, 0x00400000,
         0x00401080, 0x00000001, 0x00001001, 0x00401081,
         0x00400001, 0x00401080, 0x00401000, 0x00001001,
      },
      {
         /* nibble 7 */
         0x08200020, 0x08208000, 0x00008020, 0x00000000,
         0x08008000, 0x00200020, 0x08200000, 0x08208020,
         0x00000020, 0x08000000, 0x00208000, 0x00008020,
         0x00208020, 0x08008020, 0x08000020, 0x08200000,
         0x00008000, 0x00208020, 0x00200020, 0x08008000,
         0x08208020, 0x08000020, 0x00000000, 0x00208000,
         0x08000000, 0x00200000, 0x08008020, 0x08200020,
         0x00200000, 0x00008000, 0x08208000, 0x00000020,
         0x00200000, 0x00008000, 0x08000020, 0x08208020,
         0x00008020, 0x08000000, 0x00000000, 0x00208000,
         0x08200020, 0x08008020, 0x08008000, 0x00200020,
         0x08208000, 0x00000020, 0x00200020, 0x08008000,
         0x08208020, 0x00200000, 0x08200000, 0x08000020,
         0x00208000, 0x00008020, 0x08008020, 0x08200000,
         0x00000020, 0x08208000, 0x00208020, 0x00000000,
         0x08000000, 0x08200020, 0x00008000, 0x00208020
      }
   };
```

```
private static final int cov_2char[] =
{
   0x2E, 0x2F, 0x30, 0x31, 0x32, 0x33, 0x34, 0x35,
   0x36, 0x37, 0x38, 0x39, 0x41, 0x42, 0x43, 0x44,
   0x45, 0x46, 0x47, 0x48, 0x49, 0x4A, 0x4B, 0x4C,
   0x4D, 0x4E, 0x4F, 0x50, 0x51, 0x52, 0x53, 0x54,
   0x55, 0x56, 0x57, 0x58, 0x59, 0x5A, 0x61, 0x62,
   0x63, 0x64, 0x65, 0x66, 0x67, 0x68, 0x69, 0x6A,
   0x6B, 0x6C, 0x6D, 0x6E, 0x6F, 0x70, 0x71, 0x72,
   0x73, 0x74, 0x75, 0x76, 0x77, 0x78, 0x79, 0x7A
};

private static final int byteToUnsigned(byte b)
{
   int value = (int)b;

   return(value >= 0 ? value : value + 256);
}

private static int fourBytesToInt(byte b[], int offset)
{
   int value;

   value  =  byteToUnsigned(b[offset++]);
   value |= (byteToUnsigned(b[offset++]) <<  8);
   value |= (byteToUnsigned(b[offset++]) << 16);
   value |= (byteToUnsigned(b[offset++]) << 24);

   return(value);
}

private static final void intToFourBytes(int iValue, byte b[], int
   offset) {

   b[offset++] = (byte)((iValue)        & 0xff);
   b[offset++] = (byte)((iValue >>> 8 ) & 0xff);
   b[offset++] = (byte)((iValue >>> 16) & 0xff);
   b[offset++] = (byte)((iValue >>> 24) & 0xff);
}

private static final void PERM_OP(int a, int b, int n, int m, int
   results[]) {

   int t;

   t = ((a >>> n) ^ b) & m;
   a ^= t << n;
   b ^= t;

   results[0] = a;
   results[1] = b;
}

private static final int HPERM_OP(int a, int n, int m)
{
   int t;
```

```
      t = ((a << (16 - n)) ^ a) & m;
      a = a ^ t ^ (t >>> (16 - n));

      return(a);
   }

   private static int [] des_set_key(byte key[])
   {
      int schedule[] = new int[ITERATIONS * 2];

      int c = fourBytesToInt(key, 0);
      int d = fourBytesToInt(key, 4);

      int results[] = new int[2];

      PERM_OP(d, c, 4, 0x0f0f0f0f, results);
      d = results[0]; c = results[1];

      c = HPERM_OP(c, -2, 0xcccc0000);
      d = HPERM_OP(d, -2, 0xcccc0000);

      PERM_OP(d, c, 1, 0x55555555, results);
      d = results[0]; c = results[1];

      PERM_OP(c, d, 8, 0x00ff00ff, results);
      c = results[0]; d = results[1];

      PERM_OP(d, c, 1, 0x55555555, results);
      d = results[0]; c = results[1];

      d = (((d & 0x000000ff) <<  16) |  (d & 0x0000ff00)     |
           ((d & 0x00ff0000) >>> 16) | ((c & 0xf0000000) >>> 4));
      c &= 0x0fffffff;

      int s, t;
      int j = 0;

      for(int i = 0; i < ITERATIONS; i ++)
      {
         if(shifts2[i])
         {
            c = (c >>> 2) | (c << 26);
            d = (d >>> 2) | (d << 26);
         }
         else
         {
            c = (c >>> 1) | (c << 27);
            d = (d >>> 1) | (d << 27);
         }

         c &= 0x0fffffff;
         d &= 0x0fffffff;

         s = skb[0][ (c       ) & 0x3f                       ]|
             skb[1][((c >>>  6) & 0x03) | ((c >>>  7) & 0x3c)]|
             skb[2][((c >>> 13) & 0x0f) | ((c >>> 14) & 0x30)]|
             skb[3][((c >>> 20) & 0x01) | ((c >>> 21) & 0x06) |
                                          ((c >>> 22) & 0x38)];

         t = skb[4][ (d     )  & 0x3f                       ]|
             skb[5][((d >>> 7) & 0x03) | ((d >>>  8) & 0x3c)]|
             skb[6][ (d >>>15) & 0x3f                       ]|
             skb[7][((d >>>21) & 0x0f) | ((d >>> 22) & 0x30)];

         schedule[j++] = ((t <<  16) | (s & 0x0000ffff)) & 0xffffffff;
         s             = ((s >>> 16) | (t & 0xffff0000));

         s             = (s << 4) | (s >>> 28);
         schedule[j++] = s & 0xffffffff;
      }
      return(schedule);
   }

   private static final int D_ENCRYPT
   (int L, int R, int S, int E0, int E1, int s[])
   {
      int t, u, v;

      v = R ^ (R >>> 16);
      u = v & E0;
      v = v & E1;
      u = (u ^ (u << 16)) ^ R ^ s[S];
      t = (v ^ (v << 16)) ^ R ^ s[S + 1];
      t = (t >>> 4) | (t << 28);

      L ^= SPtrans[1][(t       ) & 0x3f] |
           SPtrans[3][(t >>>  8) & 0x3f] |
           SPtrans[5][(t >>> 16) & 0x3f] |
           SPtrans[7][(t >>> 24) & 0x3f] |
           SPtrans[0][(u       ) & 0x3f] |
           SPtrans[2][(u >>>  8) & 0x3f] |
           SPtrans[4][(u >>> 16) & 0x3f] |
           SPtrans[6][(u >>> 24) & 0x3f];

      return(L);
   }

   private static final int [] body(int schedule[], int Eswap0, int
      Eswap1) {

      int left = 0;
      int right = 0;
      int t     = 0;

      for(int j = 0; j < 25; j ++)
      {
         for(int i = 0; i < ITERATIONS * 2; i += 4)
         {
            left  = D_ENCRYPT(left,  right, i,     Eswap0, Eswap1,
            schedule);
```

```
         right = D_ENCRYPT(right, left,  i + 2, Eswap0, Eswap1,
         schedule);
      }
      t     = left;
      left  = right;
      right = t;
   }

   t = right;

   right = (left >>> 1) | (left << 31);
   left  = (t    >>> 1) | (t    << 31);

   left  &= 0xffffffff;
   right &= 0xffffffff;

   int results[] = new int[2];

   PERM_OP(right, left, 1, 0x55555555, results);
   right = results[0]; left = results[1];

   PERM_OP(left, right, 8, 0x00ff00ff, results);
   left = results[0]; right = results[1];

   PERM_OP(right, left, 2, 0x33333333, results);
   right = results[0]; left = results[1];

   PERM_OP(left, right, 16, 0x0000ffff, results);
   left = results[0]; right = results[1];

   PERM_OP(right, left, 4, 0x0f0f0f0f, results);
   right = results[0]; left = results[1];

   int out[] = new int[2];

   out[0] = left; out[1] = right;

   return(out);
}

public static final String crypt(String salt, String original)
{
   while(salt.length() < 2)
      salt += "A";

   StringBuffer buffer = new StringBuffer("             ");

   char charZero = salt.charAt(0);
   char charOne  = salt.charAt(1);

   buffer.setCharAt(0, charZero);
   buffer.setCharAt(1, charOne);

   int Eswap0 = con_salt[(int)charZero];
   int Eswap1 = con_salt[(int)charOne] << 4;
```

```
      byte key[] = new byte[8];

      for(int i = 0; i < key.length; i ++)
         key[i] = (byte)0;

      for(int i = 0; i < key.length && i < original.length(); i ++)
      {
         int iChar = (int)original.charAt(i);

         key[i] = (byte)(iChar << 1);
      }

      int schedule[] = des_set_key(key);
      int out[]      = body(schedule, Eswap0, Eswap1);

      byte b[] = new byte[9];

      intToFourBytes(out[0], b, 0);
      intToFourBytes(out[1], b, 4);
      b[8] = 0;

      for(int i = 2, y = 0, u = 0x80; i < 13; i ++)
      {
         for(int j = 0, c = 0; j < 6; j ++)
         {
            c <<= 1;

            if(((int)b[y] & u) != 0)
               c |= 1;

            u >>>= 1;

            if(u == 0)
            {
               y++;
               u = 0x80;
            }
            buffer.setCharAt(i, (char)cov_2char[c]);
         }
      }
      return(buffer.toString());
   }

   public static void main(String args[])
   {
      if(args.length >= 2)
      {
         System.out.println
         (
            "[" + args[0] + "] [" + args[1] + "] => [" +
            Jcrypt.crypt(args[0], args[1]) + "]"
         );
      }
   }
}
```

10.5.3.4.b Servicio de solicitud de vacaciones : Días laborables

10.5.3.4.b.1 application.model.holidaysrequestservice.operations

Jworkingdays

```
package application.model.holidaysrequestservice.operations;

import java.util.Calendar;

public final class Jworkingdays{

    private Jworkingdays() {}

    public static final Integer workingdays (Calendar startDate,
            Calendar finalDate){

        int i = 0;

        for ( Calendar temp = (Calendar) startDate.clone();
                temp.get(Calendar.DAY_OF_YEAR) <=
                finalDate.get(Calendar.DAY_OF_YEAR);
                temp.add(Calendar.DAY_OF_YEAR, 1)) {

            if(!(temp.get(Calendar.DAY_OF_WEEK) == Calendar.SATURDAY
            || temp.get(Calendar.DAY_OF_WEEK) ==
      Calendar.SUNDAY)) {

                i++;
            }
        }

        return new Integer(i);
    }

}
```

10.5.3.5 Util

10.5.3.5.a Utilidades del modelo

10.5.3.5.a.1 application.model.util

GlobalNames

```java
package application.model.util;

public final class GlobalNames {

      public static final String APPLICATION_DATA_SOURCE =
            "ApplicationDataSource";

      private GlobalNames () {}

}
```

UserProfileTable

```java
package application.model.util;

import java.util.Collection;
import java.util.Arrays;

public final class UserProfileTable{

    public final static String LOGINNAME = "loginName";
    public final static String FIRSTNAME = "firstName";
    public final static String SURNAME = "surname";
    public final static String DOCUMENT = "document";
    public final static String EMAIL = "email";
    public final static String ADDRESS = "address";
    public final static String PHONE = "phone";

    private final static Collection COLUMNS = Arrays.asList(
            new String[] {LOGINNAME, FIRSTNAME, SURNAME, DOCUMENT,
                  EMAIL, ADDRESS, PHONE});

    private UserProfileTable(){}

    public final static Boolean validate(String propertyName) {

        Boolean isValidValue = true;

        if (!COLUMNS.contains(propertyName)) {
            isValidValue = false;
        }

        return isValidValue;
    }

}
```

HolidaysRequestTable

```
package application.model.util;

import java.util.Collection;
import java.util.Arrays;

public final class HolidaysRequestTable{

    public final static String AUTOID = "autoId";
    public final static String LOGINNAME = "loginName";
    public final static String PERIOD = "period";
    public final static String STATEDATE = "stateDate";
    public final static String STARTDATE = "startDate";
    public final static String FINALDATE = "finalDate";
    public final static String STATE = "state";
    public final static String DETAILS = "details";

    private final static Collection COLUMNS = Arrays.asList(
            new String[] {AUTOID, LOGINNAME, PERIOD, STATEDATE,
                  STARTDATE, FINALDATE, STATE, DETAILS});

    private HolidaysRequestTable(){}

    public final static Boolean validate(String propertyName) {

        Boolean isValidValue = true;

        if (!COLUMNS.contains(propertyName)) {
            isValidValue = false;
        }

        return isValidValue;
    }

}
```

10.5.4 Controlador: Java

10.5.4.1 Manejo de peticiones y sessión

10.5.4.1.a Session Façade

10.5.4.1.a.1 application.http.controller.session

Acceso al modelo cuando existe una conexión establecida por el usuario.

SessionManager

```
package application.http.controller.session;

import java.util.Calendar;
import java.util.Collection;
import java.util.Map;

import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpSession;

import application.model.userprofileservice.facade.delegate.
UserProfileServiceFacade;
import application.model.userprofileservice.facade.delegate.
UserProfileServiceFacadeFactory;
import application.model.userprofileservice.facade.vo.UserFacadeVO;

import application.model.holidaysrequestservice.facade.delegate.
HolidaysRequestServiceFacade;
import application.model.holidaysrequestservice.facade.delegate.
HolidaysRequestServiceFacadeFactory;
import
application.model.holidaysrequestservice.facade.vo.HolidaysProfileVO;

import application.model.userprofileservice.vo.UserProfileVO;
import application.model.userprofileservice.vo.UserProfileDetailsVO;
import application.model.userprofileservice.vo.UserProfileAccessVO;
import application.model.holidaysrequestservice.vo.holidaysservice.
HolidaysServiceVO;

import application.model.holidaysrequestservice.vo.holidaysrequest.
HolidaysRequestVO;

import application.util.exceptions.AdministratorRequiredException;
import application.util.exceptions.DuplicateInstanceException;
import application.util.exceptions.IncorrectPasswordException;
import application.util.exceptions.InstanceNotFoundException;
import application.util.exceptions.InternalErrorException;
import application.util.exceptions.OperationNotAllowedException;

/**
 * This class is an implementation of the Session Façcade Pattern.
 * This façade hides the business services façades providing an unique
 * access point to the model.
 * <BR>
 * Implementing this pattern allows more indenpendency because client
 * interface only is dependent of this class not the rest of model.
 * This class represents a controller layer, hides complexity model
 * and shows a simpler interface to clients.
 */
public final class SessionManager {

// SESSION ATTRIBUTES

    /**
     * ADMINISTRATOR_NAME_SESSION_ATTRIBUTE represents the user who
     * iniciated the session when this user is administrator.
     */
    public final static String ADMINISTRATOR_NAME_SESSION_ATTRIBUTE =
            "administratorName";

    /**
     * LOGIN_NAME_SESSION_ATTRIBUTE representes the user load in
     * session. It means that if the user who iniciated the session is
     * not administrator,
     * LOGIN_NAME_SESSION_ATTRIBUTE represents this user. If the user
     * who iniciated the session is administrator, he could load other
     * user profile in session. The user loaded is represented in this
     * session attribute.
     */
    public final static String LOGIN_NAME_SESSION_ATTRIBUTE =
      "loginName";

    /**
     * USER_FACADE_DELEGATE_SESSION_ATTRIBUTE represents the user
     * facade load in session.
     */
    private final static String USER_FACADE_SESSION_ATTRIBUTE =
            "userFacade";

    /**
     * HOLIDAYS_SERVICE_FACADE_DELEGATE_SESSION_ATTRIBUTE represents
     * the user holidays service facade load in session.
     */
    private final static String
            HOLIDAYS_SERVICE_FACADE_SESSION_ATTRIBUTE =
                  "holidaysFacade";

    /**
     * REPEAT_FIND_HOLIDAYS_REQUEST_SESSION_ATTRIBUTE allows the user
     * repeat last holidays request search with the same last
     * criterias.
     */
    public final static String
      REPEAT_FIND_HOLIDAYS_REQUEST_SESSION_ATTRIBUTE =
                  "repeatFindHolidaysRequest";

//CONSTRUCTOR

    /**
     * Creates a new instance of SessionManager.
     */
    private SessionManager() {}

//FILTERING

    /**
     * Guarantees that the session will have the necessary objects if
     * the user has been authenticated.
     * @param request
     * @return
     */
    public final static Boolean isUserAuthenticated(HttpServletRequest
      request){

        HttpSession session = request.getSession(false);
        if (session == null) {
            return false;
        } else {
            return session.getAttribute(LOGIN_NAME_SESSION_ATTRIBUTE)
                  != null;
        }
    }

    /**
     * Guarantees that the session will have the necessary objects if
     * the user has been authenticated as administrator.
     * @param request
     * @return
     */
    public final static Boolean isUserAdministrator(HttpServletRequest
      request){

        HttpSession session = request.getSession(false);
        if (session == null) {
            return false;
        } else {
            return
session.getAttribute(ADMINISTRATOR_NAME_SESSION_ATTRIBUTE)!=
                    null;
        }
    }

//STORED FINDINGS

    /**
     * Stores last holidays request search in session.
     * @param request
     * @param findHolidaysRequestParameters
     */
    public final static void
      storeLastFindHolidaysRequest(HttpServletRequest request,
            Map<String, Object> findHolidaysRequestParameters) {

        HttpSession session = request.getSession(false);
        if(!(session == null)) {

session.setAttribute(REPEAT_FIND_HOLIDAYS_REQUEST_SESSION_ATTRIBUTE,
      findHolidaysRequestParameters);
        }

    }

// USER PROFILE ACTIOS

    /**
     * User tries to login.
     * @param request
     * @param loginName
     * @param clearPassword
     * @throws application.util.exceptions.InstanceNotFoundException
     * @throws application.util.exceptions.IncorrectPasswordException
     * @throws application.util.exceptions.InternalErrorException
     */
    public final static void login(HttpServletRequest request, String
      loginName, String clearPassword) throws
      InstanceNotFoundException,
            IncorrectPasswordException, InternalErrorException {

        /* Gets user facade. */
        UserProfileServiceFacade userFacadeDelegate =
                getUserFacadeDelegate(request);

        /* Checks "loginName" and "clearPassword". */
        UserFacadeVO loginResultVO =
                userFacadeDelegate.login(loginName, clearPassword,
            false);

        /* Inserts necessary objects in the session. */
        startSesssionForAuthenticatedUser(request,
            loginResultVO.getLoginName(),
                  loginResultVO.getIsAdministrator());
```

```
}

/**
 * User logs out.
 * @param request
 * @throws application.util.exceptions.InternalErrorException
 */
public final static void logout(HttpServletRequest request) throws
        InternalErrorException{

    try {

        HttpSession session =request.getSession(true);
        session.invalidate();

    } catch (Exception e) {
        throw new InternalErrorException(e);
    }
}

/**
 * Register new user in user profile service.
 * @param request
 * @param loginName
 * @param clearPassword
 * @param userProfileDetailsVO
 * @param phone
 * @param userProfileAccessVO
 * @throws
 * application.util.exceptions.AdministratorRequiredException
 * @throws application.util.exceptions.DuplicateInstanceException
 * @throws application.util.exceptions.InternalErrorException
 */
public final static void registerUser(HttpServletRequest request,
        String loginName, String clearPassword,
        UserProfileDetailsVO userProfileDetailsVO, String phone,
        UserProfileAccessVO userProfileAccessVO) throws
        AdministratorRequiredException,
  DuplicateInstanceException, InternalErrorException{

    /* Gets user facade. */
    UserProfileServiceFacade userFacadeDelegate =
            getUserFacadeDelegate(request);

    /* Register user. This action load user in facade. */
    userFacadeDelegate.registerUser(loginName, clearPassword,
            userProfileDetailsVO, phone, userProfileAccessVO);

    /* Inserts necessary objects in the session. */
    updateSesssionForAuthenticatedUser(request, loginName);

}

/**
 * This action updates only user profile details.
 * @param request
 * @param userProfileDetailsVO
 * @throws application.util.exceptions.InternalErrorException
 */
public final static void
  updateUserProfileDetails(HttpServletRequest
        request, UserProfileDetailsVO userProfileDetailsVO)
        throws InternalErrorException {

    /* Gets facade. */
    UserProfileServiceFacade userFacadeDelegate =
            getUserFacadeDelegate(request);

    /* Updates user profile details. */
userFacadeDelegate.updateUserProfileDetails(userProfileDetailsVO);

}

/**
 * This action updates only user profile access details.
 * @param request
 * @param userProfileAccessVO
 * @throws
 * application.util.exceptions.AdministratorRequiredException
 * @throws application.util.exceptions.InternalErrorException
 */
public final static void
  updateUserProfileAccess(HttpServletRequest request,
        UserProfileAccessVO userProfileAccessVO) throws
        AdministratorRequiredException, InternalErrorException {

    /* Gets facade. */
    UserProfileServiceFacade userFacadeDelegate =
            getUserFacadeDelegate(request);

    /* Updates user profile details. */
  userFacadeDelegate.updateUserProfileAccess(userProfileAccessVO);

}

/**
 * This action changes user password.
 * @param request
 * @param oldClearPassword
 * @param newClearPassword
 * @throws application.util.exceptions.IncorrectPasswordException
 * @throws application.util.exceptions.InternalErrorException
 */
public final static void changePassword(HttpServletRequest
  request, String oldClearPassword, String newClearPassword)
  throws
        IncorrectPasswordException, InternalErrorException  {

    /* Gets facade. */
    UserProfileServiceFacade userFacadeDelegate =
            getUserFacadeDelegate(request);
    /* Changes password. */
    userFacadeDelegate.changePassword(oldClearPassword,
            newClearPassword, false);

}

/**
 * This administrator action allows give a new password to the
 * user load in session.
 * @param request
 * @param password
 * @param newClearPassword
 * @throws
 * application.util.exceptions.AdministratorRequiredException
 * @throws application.util.exceptions.IncorrectPasswordException
 * @throws application.util.exceptions.InternalErrorException
 */
public final static void givePassword(HttpServletRequest request,
        String password, String newClearPassword) throws
        AdministratorRequiredException,
  IncorrectPasswordException,
        InternalErrorException {

    /* Gets facade. */
    UserProfileServiceFacade userFacadeDelegate =
            getUserFacadeDelegate(request);

    /* Gives new user password. */
    userFacadeDelegate.givePassword(password, newClearPassword,
  false);
}

/**
 * This actions returns the user load in session profile.
 * @param request
 * @throws application.util.exceptions.InternalErrorException
 * @return
 */
public final static UserProfileVO
  findUserProfile(HttpServletRequest
        request) throws InternalErrorException {

    /* Gets facade. */
    UserProfileServiceFacade userFacadeDelegate =
            getUserFacadeDelegate(request);

    /* Finds user profile. */
    return userFacadeDelegate.findUserProfile();

}

/**
 * Changes user contact details.
 * @param request
 * @param password
 * @param newPhone
```

```
 * @throws application.util.exceptions.IncorrectPasswordException
 * @throws application.util.exceptions.InternalErrorException
 */
public final static void changePhone(HttpServletRequest request,
        String password, String newPhone) throws
        IncorrectPasswordException, InternalErrorException  {

    /* Gets user facade. */
    UserProfileServiceFacade userFacadeDelegate =
            getUserFacadeDelegate(request);

    /* Changes phone. */
    userFacadeDelegate.changePhone(password, newPhone, false);

}

/**
 * Changes user contact details overwriting existing one.
 * @param request
 * @param password
 * @param newPhone
 * @throws
 * application.util.exceptions.AdministratorRequiredException
 * @throws application.util.exceptions.IncorrectPasswordException
 * @throws application.util.exceptions.InternalErrorException
 */
public final static void givePhone(HttpServletRequest request,
        String password, String newPhone) throws
        AdministratorRequiredException,
  IncorrectPasswordException, InternalErrorException {

    /* Gets user facade. */
    UserProfileServiceFacade userFacadeDelegate =
            getUserFacadeDelegate(request);

    /* Inserts new contact detail. */
    userFacadeDelegate.givePhone(password, newPhone, false);

}

/**
 * This action allows an administrator user load other user in
 * session.
 * @param request
 * @param password
 * @param loginName
 * @throws
 * application.util.exceptions.AdministratorRequiredException
 * @throws application.util.exceptions.IncorrectPasswordException
 * @throws application.util.exceptions.InstanceNotFoundException
 * @throws application.util.exceptions.InternalErrorException
 */
public final static void loginAsUser(HttpServletRequest request,
        String password, String loginName) throws
        AdministratorRequiredException,
  IncorrectPasswordException,
        InstanceNotFoundException, InternalErrorException {

    /* Gets user facade. */
    UserProfileServiceFacade userFacadeDelegate =
            getUserFacadeDelegate(request);

    userFacadeDelegate.loginAsUser(password, loginName, false);

    updateSesssionForAuthenticatedUser(request, loginName);

}

/**
 * Finds profile with a loginName given.
 * @param request
 * @param loginName
 * @throws
 * application.util.exceptions.AdministratorRequiredException
 * @throws application.util.exceptions.InstanceNotFoundException
 * @throws application.util.exceptions.InternalErrorException
 * @return
 */
public final static UserProfileVO findUser(HttpServletRequest
  request,
        String loginName) throws AdministratorRequiredException,
        InstanceNotFoundException, InternalErrorException {

    /* Gets facade. */
    UserProfileServiceFacade userFacadeDelegate =
                getUserFacadeDelegate(request);
    /* Finds user profile. */
    return userFacadeDelegate.findUserProfile(loginName);

}

/**
 * Carries out an advanced search.
 * @param request
 * @param propertyInName
 * @param pattern
 * @param propertyByName
 * @param startIndex
 * @param count
 * @throws application.util.exceptions.InternalErrorException
 * @return
 */
public final static Collection findUserAdvanced(HttpServletRequest
        request, String propertyInName, String pattern, String
  propertyByName, Integer startIndex, Integer count) throws
        InternalErrorException {

    /* Gets user facade. */
    UserProfileServiceFacade userFacadeDelegate =
            getUserFacadeDelegate(request);

    /* Finds. */
    return userFacadeDelegate.findInBy(propertyInName, pattern,
            propertyByName, startIndex, count);

}

//HOLIDAYS SERVICE.

/**
 * Register a new period of holildays for an user.
 * @param request
 * @param period
 * @param days
 * @throws
 * application.util.exceptions.AdministratorRequiredException
 * @throws application.util.exceptions.DuplicateInstanceException
 * @throws application.util.exceptions.InternalErrorException
 */
public final static void
  registerHolidaysService(HttpServletRequest request,
        Calendar period, Integer days) throws
  AdministratorRequiredException,
        DuplicateInstanceException, InternalErrorException {

    /* Gets holidays service facade. */
    HolidaysRequestServiceFacade holidaysServiceFacadeDelegate =
            getHolidaysServiceFacadeDelegate(request);

    /* Registers holidays service. */
    holidaysServiceFacadeDelegate.registerHolidaysService(period,
        days);
}

/**
 * Finds all user holidays services.
 * @param request
 * @param startIndex
 * @param count
 * @throws application.util.exceptions.InternalErrorException
 * @return
 */
public final static Collection findUserHolidaysService(
        HttpServletRequest request, Integer startIndex, Integer
  count) throws InternalErrorException {

    /* Gets facade. */
    HolidaysRequestServiceFacade holidaysServiceFacadeDelegate =
            getHolidaysServiceFacadeDelegate(request);

    /* Finds. */
    return holidaysServiceFacadeDelegate.findUserHolidaysServices(
            startIndex, count);

}

/**
 * Finds all users holidays services in a period given.
 * @param request
 * @param period
 * @param startIndex
 * @param count
 * @throws
application.util.exceptions.AdministratorRequiredException
 * @throws application.util.exceptions.InternalErrorException
 * @return
 */
public final static Collection findPeriodHolidaysService(
        HttpServletRequest request, Calendar period, Integer
  startIndex, Integer count) throws
  AdministratorRequiredException, InternalErrorException {

    /* Gets facade. */
    HolidaysRequestServiceFacade holidaysServiceFacadeDelegate =
            getHolidaysServiceFacadeDelegate(request);

    /* Finds. */
    return
  holidaysServiceFacadeDelegate.findPeriodHolidaysServices(period,
            startIndex, count);

}

/**
 * Finds all users holidays services.
 * @param request
 * @param startIndex
 * @param count
 * @throws
 * application.util.exceptions.AdministratorRequiredException
 * @throws application.util.exceptions.InternalErrorException
 * @return
 */
public final static Collection findHolidaysServices(
        HttpServletRequest request, Integer startIndex, Integer
  count) throws AdministratorRequiredException,
  InternalErrorException {

    /* Gets facade. */
    HolidaysRequestServiceFacade holidaysServiceFacadeDelegate =
            getHolidaysServiceFacadeDelegate(request);

    /* Finds. */
    return
   holidaysServiceFacadeDelegate.findHolidaysServices(startIndex,
            count);

}

/**
 * Changes days of holidays in a user holidays service registered.
 * @param request
 * @param period
 * @param days
```

```
 * @throws
 * application.util.exceptions.AdministratorRequiredException
 * @throws application.util.exceptions.InstanceNotFoundException
 * @throws application.util.exceptions.InternalErrorException
 */
public final static void
  updateUserHolidaysService(HttpServletRequest request,
        Calendar period, Integer days) throws
        AdministratorRequiredException, InstanceNotFoundException,
        InternalErrorException {

    /* Gets facade. */
    HolidaysRequestServiceFacade holidaysServiceDelegate =
            getHolidaysServiceFacadeDelegate(request);

    /* Finds. */
    holidaysServiceDelegate.updateHolidaysService(period, days);

}

/**
 * Finds user holidays services in a period given.
 * @param request
 * @param period
 * @throws application.util.exceptions.InstanceNotFoundException
 * @throws application.util.exceptions.InternalErrorException
 * @return
 */
public final static HolidaysServiceVO findHolidaysService(
        HttpServletRequest request, Calendar period) throws
        InstanceNotFoundException, InternalErrorException {

    /* Gets facade. */
    HolidaysRequestServiceFacade holidaysServiceFacadeDelegate =
                getHolidaysServiceFacadeDelegate(request);

    /* Finds. */
    return
       holidaysServiceFacadeDelegate.findHolidaysService(period);

}

/**
 * Registered a holidays request for the user load in session.
 * @param request
 * @param startDate
 * @param finalDate
 * @param details
 * @throws application.util.exceptions.InstanceNotFoundException
 * @throws application.util.exceptions.DuplicateInstanceException
 * @throws application.util.exceptions.InternalErrorException
 * @return
 */
public final static Integer
  registerHolidaysRequest(HttpServletRequest request,
        Calendar startDate, Calendar finalDate, String details)
  throws InstanceNotFoundException,
  DuplicateInstanceException, InternalErrorException {

    /* Gets facade. */
    HolidaysRequestServiceFacade holidaysServiceFacadeDelegate =
            getHolidaysServiceFacadeDelegate(request);

    /* Registers request and returns auto generated id. */
    return
  holidaysServiceFacadeDelegate.registerHolidaysRequest(startDate,
            finalDate, details);

}

/**
 * Finds an user holidays request with an identifier given
 * @return
 * @param request
 * @param autoId
 * @throws application.util.exceptions.InstanceNotFoundException
 * @throws application.util.exceptions.InternalErrorException
 */
public final static HolidaysRequestVO
  findUserHolidaysRequest(HttpServletRequest request,
        Integer autoId) throws InstanceNotFoundException,
  InternalErrorException {

    /* Gets facade. */
    HolidaysRequestServiceFacade holidaysServiceFacadeDelegate =
            getHolidaysServiceFacadeDelegate(request);

    /* Finds. */
    return
  holidaysServiceFacadeDelegate.findUserIdHolidaysRequest(autoId);

}

/**
 * Finds a holidays request with an identifier given
 * @param request
 * @param autoId
 * @throws
 * application.util.exceptions.AdministratorRequiredException
 * @throws application.util.exceptions.InstanceNotFoundException
 * @throws application.util.exceptions.InternalErrorException
 * @return
 */
public final static HolidaysRequestVO
  findHolidaysRequest(HttpServletRequest request,
        Integer autoId) throws AdministratorRequiredException,
        InstanceNotFoundException, InternalErrorException {

    /* Gets facade. */
    HolidaysRequestServiceFacade holidaysServiceFacadeDelegate =
            getHolidaysServiceFacadeDelegate(request);
    /* Finds. */
    return
      holidaysServiceFacadeDelegate.findIdHolidaysRequest(autoId);

}

/**
 * Carries out an advanced search in holidays request.
 * @param request
 * @param isUserLoaded
 * @param autoId
 * @param period
 * @param sinceDate
 * @param state
 * @param propertyName
 * @param isDesc
 * @param startIndex
 * @param count
 * @throws
 * application.util.exceptions.AdministratorRequiredException
 * @throws application.util.exceptions.InternalErrorException
 * @return
 */
public final static Collection
  findAdvancedHolidaysRequest(HttpServletRequest
        request, Boolean isUserLoaded, Integer autoId, Calendar
  period, Calendar sinceDate, Short state, String
  propertyName, Boolean isDesc, Integer startIndex, Integer
  count) throws
        AdministratorRequiredException, InternalErrorException {

    /* Gets facade. */
    HolidaysRequestServiceFacade holidaysServiceFacadeDelegate =
            getHolidaysServiceFacadeDelegate(request);

    /* Finds. */
    return
       holidaysServiceFacadeDelegate.findAdvanced(isUserLoaded,
  autoId, period, sinceDate, state, propertyName, isDesc,
  startIndex, count);

}

/**
 * Carries out an advanced search in holidays request.
 * @param request
 * @param autoId
 * @param period
 * @param sinceDate
 * @param state
 * @param propertyName
 * @param isDesc
 * @param startIndex
 * @param count
 * @throws application.util.exceptions.InternalErrorException
 * @return
 */
public final static Collection findAdvancedUserHolidaysRequest(
        HttpServletRequest request, Integer autoId, Calendar
  period, Calendar sinceDate, Short state, String
  propertyName, Boolean isDesc, Integer startIndex, Integer
  count) throws InternalErrorException {

    /* Gets facade. */
    HolidaysRequestServiceFacade holidaysServiceFacadeDelegate =
            getHolidaysServiceFacadeDelegate(request);

    /* Finds. */
    return holidaysServiceFacadeDelegate.findAdvancedUser(autoId,
        period, sinceDate, state, propertyName, isDesc,
  startIndex, count);

}

/**
 * Updates holidays request state if actual state is not
 * tramitted.
 * @param request
 * @param autoId
 * @param state
 * @throws
 * application.util.exceptions.AdministratorRequiredException
 * @throws application.util.exceptions.InstanceNotFoundException
 * @throws application.util.exceptions.InternalErrorException
 */
public final static void updateHolidaysRequest(HttpServletRequest
  request, Integer autoId, Short state) throws
  AdministratorRequiredException,
        InstanceNotFoundException, InternalErrorException {

    /* Gets facade. */
    HolidaysRequestServiceFacade holidaysServiceFacadeDelegate =
            getHolidaysServiceFacadeDelegate(request);

    try {

    /* Updates state if actual state is not tramitted. */
    holidaysServiceFacadeDelegate.updateHolidaysRequest(autoId,
  state);
    } catch (OperationNotAllowedException e) {
        throw new InternalErrorException(e);
    }

}

/**
 * Cancels an user holidays request.
 * @param request
 * @param autoId
 * @throws application.util.exceptions.InstanceNotFoundException
 * @throws application.util.exceptions.InternalErrorException
 */
public final static void cancelHolidaysRequest(HttpServletRequest
  request, Integer autoId) throws InstanceNotFoundException,
        InternalErrorException {

    /* Gets facade. */
    HolidaysRequestServiceFacade holidaysServiceFacadeDelegate =
            getHolidaysServiceFacadeDelegate(request);

    try {

    /* Updates state if actual state is not tramitted. */
    holidaysServiceFacadeDelegate.cancelHolidaysRequest(autoId);

    } catch (OperationNotAllowedException e) {
        throw new InternalErrorException(e);
    }

}

/**
 * Finds all user holidays profiles.
 * @param request
 * @param startIndex
 * @param count
 * @throws application.util.exceptions.InternalErrorException
 * @return
 */
public final static Collection findHolidaysProfile
  (HttpServletRequest request, Integer startIndex, Integer
  count) throws InternalErrorException {

    /* Gets facade. */
    HolidaysRequestServiceFacade holidaysServiceFacadeDelegate =
            getHolidaysServiceFacadeDelegate(request);

    /* Finds.*/
    return
  holidaysServiceFacadeDelegate.findHolidaysProfile(startIndex,
  count);

}

/**
 * Finds an user holidays profile in period given.
 * @param request
 * @param loginName
 * @param period
 * @throws
 * application.util.exceptions.AdministratorRequiredException
 * @throws application.util.exceptions.InstanceNotFoundException
 * @throws application.util.exceptions.InternalErrorException
 * @return
 */
public final static HolidaysProfileVO
  findUserHolidaysProfilePeriod (
        HttpServletRequest request, String loginName, Calendar
  period) throws AdministratorRequiredException,
  InstanceNotFoundException, InternalErrorException {

    /* Gets facade. */
    HolidaysRequestServiceFacade holidaysServiceFacadeDelegate =
            getHolidaysServiceFacadeDelegate(request);

    /* Finds. */
    return
      holidaysServiceFacadeDelegate.findUserHolidaysProfilePeriod(
            loginName, period);

}


    /**
     * Finds all holidays profile in a period.
     * @param request
     * @param period
     * @throws
     * application.util.exceptions.InstanceNotFoundException
     * @throws application.util.exceptions.InternalErrorException
     * @return
     */
    public final static HolidaysProfileVO
        findHolidaysProfilePeriod (HttpServletRequest
  request, Calendar period)
              throws InstanceNotFoundException,
        InternalErrorException {

    /* Gets facade. */
    HolidaysRequestServiceFacade holidaysServiceFacadeDelegate =
            getHolidaysServiceFacadeDelegate(request);

    /* Finds. */
    return
  holidaysServiceFacadeDelegate.findHolidaysProfilePeriod(period);

}

/**
 * Returns all periods in which exists at least one user holidays
 * profile.
 * @param request
 * @param startIndex
 * @param count
 * @throws
 * application.util.exceptions.AdministratorRequiredException
 * @throws application.util.exceptions.InternalErrorException
 * @return
 */
public final static Collection findHolidaysProfiles(
        HttpServletRequest request, Integer startIndex, Integer
  count) throws AdministratorRequiredException,
  InternalErrorException {
        /* Gets facade. */
        HolidaysRequestServiceFacade holidaysServiceFacadeDelegate =
                  getHolidaysServiceFacadeDelegate(request);

        /* Finds. */
        return
       holidaysServiceFacadeDelegate.findHolidaysProfiles(startIndex,
                 count);

    }

    /**
     * Finds all holidays profiles in a period.
     * @param request
     * @param period
     * @param startIndex
     * @param count
     * @throws
     * application.util.exceptions.AdministratorRequiredException
     * @throws application.util.exceptions.InternalErrorException
     * @return
     */
    public final static Collection FindPeriodHolidaysProfile
      (HttpServletRequest request,
            Calendar period, Integer startIndex, Integer count) throws
            AdministratorRequiredException, InternalErrorException {

        /* Gets facade. */
        HolidaysRequestServiceFacade holidaysServiceFacadeDelegate =
                getHolidaysServiceFacadeDelegate(request);

        /* Finds. */
        return
       holidaysServiceFacadeDelegate.findPeriodHolidaysProfile(period,
                startIndex, count);

    }

//MANAGES SESSION OBJECTS

    /**
     * Inserts objects in session when an user is authenticated.
     * @param request
     * @param loginName
     * @param isAdministrator
     * @throws application.util.exceptions.InternalErrorException
     */
    private final static void startSesssionForAuthenticatedUser(
            HttpServletRequest request, String loginName, boolean
      isAdministrator) throws InternalErrorException {

        try {
            HttpSession session = request.getSession(true);
            session.setAttribute(LOGIN_NAME_SESSION_ATTRIBUTE,
            loginName);
            if(isAdministrator){

          session.setAttribute(ADMINISTRATOR_NAME_SESSION_ATTRIBUTE,
                          loginName);
            }
            updateHolidaysServiceFacadeDelegateSession(request);

        } catch (Exception e) {
            throw new InternalErrorException(e);
        }
    }

    /**
     * Updates objects in session.
     * @param request
     * @param loginName
     * @throws application.util.exceptions.InternalErrorException
     */
    private final static void updateSesssionForAuthenticatedUser(
            HttpServletRequest request, String loginName) throws
            InternalErrorException {

        try {

            HttpSession session = request.getSession(true);
            session.setAttribute(LOGIN_NAME_SESSION_ATTRIBUTE,
            loginName);
            updateHolidaysServiceFacadeDelegateSession(request);

        } catch (Exception e) {
            throw new InternalErrorException(e);
        }
    }

    /**
     * Initialitiates or updates holidays service facade.
     * @param request
     * @throws application.util.exceptions.InternalErrorException
     */
    private final static void
      updateHolidaysServiceFacadeDelegateSession(
            HttpServletRequest request) throws InternalErrorException{

        try{
            HolidaysRequestServiceFacade holidaysServiceFacadeDelegate
                  = getHolidaysServiceFacadeDelegate(request);

            holidaysServiceFacadeDelegate.initHolidaysRequestService(
                    getUserFacadeDelegate(request));
        } catch (Exception e) {
            throw new InternalErrorException(e);
        }

    }

//GETS FACADES
```

```
    /**
     * Gets user facade from session or creates new one.
     * @param request
     * @throws application.util.exceptions.InternalErrorException
     * @return
     */
    private final static UserProfileServiceFacade
      getUserFacadeDelegate(HttpServletRequest request) throws
      InternalErrorException {

        HttpSession session = request.getSession(true);
        UserProfileServiceFacade userFacadeDelegate =
                (UserProfileServiceFacade) session.getAttribute(
                USER_FACADE_SESSION_ATTRIBUTE);

        if (userFacadeDelegate == null) {

            userFacadeDelegate = (UserProfileServiceFacade)
            UserProfileServiceFacadeFactory.getFacade();

            session.setAttribute(USER_FACADE_SESSION_ATTRIBUTE,
                    userFacadeDelegate);
        }

        return userFacadeDelegate;
    }

    /**
     * Gets holidays service facade from session or creates new one.
     * @param request
     * @throws application.util.exceptions.InternalErrorException
     * @return
     */
    private final static HolidaysRequestServiceFacade
      getHolidaysServiceFacadeDelegate(HttpServletRequest
      request) throws InternalErrorException {

        HttpSession session = request.getSession(true);
        HolidaysRequestServiceFacade holidaysServiceFacadeDelegate =
                (HolidaysRequestServiceFacade) session.getAttribute(
                HOLIDAYS_SERVICE_FACADE_SESSION_ATTRIBUTE);

        if (holidaysServiceFacadeDelegate == null) {
            holidaysServiceFacadeDelegate =
                    HolidaysRequestServiceFacadeFactory.getFacade();

            session.setAttribute(
                    HOLIDAYS_SERVICE_FACADE_SESSION_ATTRIBUTE,
                    holidaysServiceFacadeDelegate);
        }

        return holidaysServiceFacadeDelegate;
    }

}
```

10.5.4.2 Intercepting Filter

10.5.4.2.a Establecimiento de sessión por el usuario

10.5.4.2.a.1 application.http.controller.filtering

AuthenticationPreProcessingFilter

```
package application.http.controller.filtering;

import java.io.IOException;

import javax.servlet.http.HttpServletRequest;

import javax.servlet.ServletException;
import javax.servlet.ServletRequest;
import javax.servlet.ServletResponse;

import application.http.controller.session.SessionManager;

import application.util.exceptions.InternalErrorException;

import application.util.controller.filter.DefaultFilter;

/**
 * A filter to check if the action to be executed requires that the
 * user had been authenticated. If the user has not been authenticated
 * and the action requires it, doFilterProcess returns the
 * ActionForward returned by
 * mapping.findForward("AuthenticationRequiredPage").
 */
public class AuthenticationPreProcessingFilter extends DefaultFilter {

    protected Boolean doFilterProcess(ServletRequest request,
            ServletResponse response) throws IOException,
      ServletException, InternalErrorException {

        return SessionManager.isUserAuthenticated((HttpServletRequest)
                  request);
    }

}
```

AdministratorPreProcessingFilter

```java
package application.http.controller.filtering;

import java.io.IOException;

import javax.servlet.http.HttpServletRequest;

import javax.servlet.ServletException;
import javax.servlet.ServletRequest;
import javax.servlet.ServletResponse;

import application.http.controller.session.SessionManager;

import application.util.exceptions.InternalErrorException;

import application.util.controller.filter.DefaultFilter;

/**
 * A filter to check if the action to be executed requires that the
 * user had been authenticated as administrator. If the user has not
 * been authenticated
 * as administrator and the action requires it, doFilterProcess
 * returns the ActionForward returned by
 * mapping.findForward("AdministratorRequiredPage").
 */
public class AdministratorPreProcessingFilter extends DefaultFilter {

    protected Boolean doFilterProcess(ServletRequest request,
            ServletResponse response) throws IOException,
      ServletException, InternalErrorException {

        return SessionManager.isUserAdministrator((HttpServletRequest)
            request);
    }

}
```

10.5.4.2.b Acciones del servicio de perfil de usuario

10.5.4.2.b.1 application.http.controller.actions.userprofile

LogoutAction

```java
package application.http.controller.actions.userprofile;

import java.io.IOException;

import javax.servlet.ServletException;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

import org.apache.struts.action.ActionForm;
import org.apache.struts.action.ActionForward;
import org.apache.struts.action.ActionMapping;

import application.http.controller.session.SessionManager;

import application.util.exceptions.InternalErrorException;

import application.util.controller.struts.DefaultAction;

/**
 * Closes an user session.
 */
public class LogoutAction extends DefaultAction {

      /**
       *
       * @param mapping
       * @param form
       * @param request
       * @param response
       * @throws java.io.IOException
       * @throws javax.servlet.ServletException
       * @throws application.util.exceptions.InternalErrorException
       * @return
       */
      public ActionForward doExecute(ActionMapping mapping, ActionForm
                  form, HttpServletRequest request, HttpServletResponse
                  response) throws IOException, ServletException,
            InternalErrorException {

            /* Do logout. */
            SessionManager.logout(request);

            /* Return ActionForward. */
            return mapping.findForward("LogoutAction");

        }

}
```

LoginAction

```
package application.http.controller.actions.userprofile;

import java.io.IOException;

import javax.servlet.ServletException;
```

```java
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

import org.apache.struts.action.ActionMessages;
import org.apache.struts.action.ActionMessage;
import org.apache.struts.action.ActionForm;
import org.apache.struts.action.ActionForward;
import org.apache.struts.action.ActionMapping;

import org.apache.struts.validator.DynaValidatorForm;

import application.http.controller.session.SessionManager;

import application.util.exceptions.InstanceNotFoundException;
import application.util.exceptions.IncorrectPasswordException;
import application.util.exceptions.InternalErrorException;

import application.util.controller.struts.DefaultAction;

/**
 * Log in Action.
 */
public class LoginAction extends DefaultAction {

    /**
     * @param mapping
     * @param form
     * @param request
     * @param response
     * @throws java.io.IOException
     * @throws javax.servlet.ServletException
     * @throws application.util.exceptions.InternalErrorException
     * @return
     */
    public ActionForward doExecute(ActionMapping mapping, ActionForm
      form, HttpServletRequest request, HttpServletResponse
      response) throws IOException, ServletException,
      InternalErrorException {

        /* Gets data. */
        DynaValidatorForm loginForm = (DynaValidatorForm) form;

        String loginName = (String) loginForm.get("loginName");
        String password = (String) loginForm.get("password");

        /* Prepares errors to catch. */
        ActionMessages errors = new ActionMessages();

        try {

            SessionManager.login(request, loginName, password);

        } catch (InstanceNotFoundException e) {
            errors.add("loginName", new
                  ActionMessage("errorsMessage.notFound.loginName"));
        } catch (IncorrectPasswordException e) {
            errors.add("password", new
                  ActionMessage("errorsMessage.incorrect.password"));
        }

        /* Returns ActionForward. */
        if (errors.isEmpty()) {
            return mapping.findForward("LoginAction");
        } else {
            saveErrors(request, errors);
            return new ActionForward(mapping.getInput());
        }
    }

}
```

LoginAsUserAction

```
package application.http.controller.actions.userprofile;

import java.io.IOException;

import javax.servlet.ServletException;

import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

import org.apache.struts.action.ActionMessages;
import org.apache.struts.action.ActionMessage;
import org.apache.struts.action.ActionForm;
import org.apache.struts.action.ActionForward;
import org.apache.struts.action.ActionMapping;

import org.apache.struts.validator.DynaValidatorForm;

import application.http.controller.session.SessionManager;

import application.util.exceptions.AdministratorRequiredException;
import application.util.exceptions.InstanceNotFoundException;
import application.util.exceptions.IncorrectPasswordException;
import application.util.exceptions.InternalErrorException;

import application.util.controller.struts.AdministratorDefaultAction;

public class LoginAsUserAction extends AdministratorDefaultAction {

    public ActionForward doExecute(ActionMapping mapping, ActionForm
      form, HttpServletRequest request, HttpServletResponse
      response) throws IOException, ServletException,
      InternalErrorException, AdministratorRequiredException {

        /* Gets data. */
        DynaValidatorForm loginAsUserForm = (DynaValidatorForm) form;
        String administratorPassword =
                (String) loginAsUserForm.get("password");
        String loginName = (String) loginAsUserForm.get("loginName");

        /* Prepares errors to catch. */
        ActionMessages errors = new ActionMessages();

        try {

            SessionManager.loginAsUser(request, administratorPassword,
                    loginName);
        } catch (InstanceNotFoundException e) {
            errors.add("loginName", new
            ActionMessage("errorsMessage.notFound.loginName"));
        } catch (IncorrectPasswordException e) {
            errors.add("password", new
            ActionMessage("errorMessage.incorrect.password"));
        }

        /* Returns ActionForward. */
        if (errors.isEmpty()) {

            return mapping.findForward("LoginAsUserAction");

        } else {
            saveErrors(request, errors);
            return new ActionForward(mapping.getInput());
        }
    }

}
```

RegisterUserAction

```
package application.http.controller.actions.userprofile;

import java.io.IOException;

import javax.servlet.ServletException;

import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

import org.apache.struts.action.ActionMessages;
import org.apache.struts.action.ActionMessage;
import org.apache.struts.action.ActionForm;
import org.apache.struts.action.ActionForward;
import org.apache.struts.action.ActionMapping;

import org.apache.struts.validator.DynaValidatorActionForm;

import application.http.controller.session.SessionManager;

import application.model.userprofileservice.vo.UserProfileVO;
import application.model.userprofileservice.vo.UserProfileDetailsVO;
import application.model.userprofileservice.vo.UserProfileAccessVO;

import application.util.exceptions.AdministratorRequiredException;
import application.util.exceptions.DuplicateInstanceException;
import application.util.exceptions.InternalErrorException;

import application.util.controller.struts.AdministratorDefaultAction;

public class RegisterUserAction extends AdministratorDefaultAction {

    public ActionForward doExecute(ActionMapping mapping, ActionForm
      form, HttpServletRequest request, HttpServletResponse
      response) throws IOException, ServletException,
      AdministratorRequiredException, InternalErrorException {

        /* Gets data. */
        DynaValidatorActionForm userProfileForm =
      (DynaValidatorActionForm) form;

        String loginName = (String) userProfileForm.get("loginName");
        String password = (String) userProfileForm.get("password");

        UserProfileDetailsVO userProfileDetailsVO = new
      UserProfileDetailsVO(
                  (String) userProfileForm.get("firstName"),
                  (String) userProfileForm.get("surname"),
                  (String) userProfileForm.get("document"),
                  (String) userProfileForm.get("email"),
                  (String) userProfileForm.get("address"),
                  (String) userProfileForm.get("details"));

        String phone = (String) userProfileForm.get("phone");

        Boolean isAdministrator=
                (Boolean) userProfileForm.get("isAdministrator");

        if(isAdministrator != Boolean.TRUE){
            isAdministrator = Boolean.FALSE;
        }

        UserProfileAccessVO userProfileAccessVO =
                new UserProfileAccessVO(isAdministrator);

        /* Prepares errors to catch. */
        ActionMessages errors = new ActionMessages();

        try {

            SessionManager.registerUser(request, loginName, password,
                    userProfileDetailsVO, phone, userProfileAccessVO);

        } catch (DuplicateInstanceException e) {

            errors.add("loginName", new
           ActionMessage("errorsMessage.duplicated.loginName"));
        }

        /* Return ActionForward. */
        if (errors.isEmpty()) {
            return mapping.findForward("RegisterUserAction");
        } else {
            saveErrors(request, errors);
            return new ActionForward(mapping.getInput());
        }
    }
}
```

EditUserProfileAction

```
package application.http.controller.actions.userprofile;

import java.io.IOException;

import javax.servlet.ServletException;

import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

import org.apache.struts.Globals;
import org.apache.struts.action.ActionForm;
import org.apache.struts.action.ActionForward;
import org.apache.struts.action.ActionMapping;

import org.apache.struts.validator.DynaValidatorActionForm;

import application.model.userprofileservice.vo.UserProfileVO;
import application.model.userprofileservice.vo.UserProfileDetailsVO;
import application.model.userprofileservice.vo.UserProfileAccessVO;

import application.http.controller.session.SessionManager;

import application.util.controller.struts.DefaultAction;

import application.util.exceptions.InternalErrorException;

public class EditUserProfileAction extends DefaultAction {

    public ActionForward doExecute(ActionMapping mapping, ActionForm
      form, HttpServletRequest request, HttpServletResponse
      response) throws IOException, ServletException,
      InternalErrorException {

        /* Gets data from form. */
        DynaValidatorActionForm userProfileForm =
                (DynaValidatorActionForm) form;

        /* If the request is to allow the user to correct errors in
* the form, "userProfileForm" must not be modified. */
```

```
        if (request.getAttribute(Globals.ERROR_KEY) == null) {

            /* If the user is viewign or updating his/her profile, set
             * the rest of attributes. */
            UserProfileVO userProfileVO =
                    SessionManager.findUserProfile(request);

            UserProfileDetailsVO userProfileDetailsVO =
                    userProfileVO.getUserProfileDetailsVO();

            UserProfileAccessVO userProfileAccessVO =
                    userProfileVO.getUserProfileAccessVO();

            userProfileForm.set("loginName",
            userProfileVO.getLoginName());

            userProfileForm.set("firstName",
                    userProfileDetailsVO.getFirstName());
            userProfileForm.set("surname",
            userProfileDetailsVO.getSurname());
            userProfileForm.set("document",
            userProfileDetailsVO.getDocument());
            userProfileForm.set("email",
            userProfileDetailsVO.getEmail());
            userProfileForm.set("address",
            userProfileDetailsVO.getAddress());
            userProfileForm.set("details",
            userProfileDetailsVO.getDetails());

            userProfileForm.set("phone", userProfileVO.getPhone());

            userProfileForm.set("isAdministrator",
                    userProfileAccessVO.getIsAdministrator());

        }

        return new
      ActionForward(mapping.findForward("EditUserProfileAction"));

    }

}
```

EditPhoneAction

```
package application.http.controller.actions.userprofile;

import java.io.IOException;

import javax.servlet.ServletException;

import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import org.apache.struts.Globals;
import org.apache.struts.action.ActionForm;
import org.apache.struts.action.ActionForward;
import org.apache.struts.action.ActionMapping;

import org.apache.struts.validator.DynaValidatorForm;

import application.model.userprofileservice.vo.UserProfileVO;

import application.http.controller.session.SessionManager;

import application.util.controller.struts.DefaultAction;

import application.util.exceptions.InternalErrorException;

public class EditPhoneAction extends DefaultAction {

    public ActionForward doExecute(ActionMapping mapping, ActionForm
      form, HttpServletRequest request, HttpServletResponse
      response) throws IOException, ServletException,
      InternalErrorException {

        /* Gets data from form. */
        DynaValidatorForm changePhoneForm =
                (DynaValidatorForm) form;

        /* If the request is to allow the user to correct errors in
         * the form, "userProfileForm" must not be modified. */
        if (request.getAttribute(Globals.ERROR_KEY) == null) {

            /* If the user is updating his/her profile, set the rest
             * of attributes. */
            UserProfileVO userProfileVO =
                    SessionManager.findUserProfile(request);

            changePhoneForm.set("phone", userProfileVO.getPhone());

        }

        return new
      ActionForward(mapping.findForward("EditPhoneAction"));

    }

}
```

ChangePasswordAction

```
package application.http.controller.actions.userprofile;

import java.io.IOException;
import javax.servlet.ServletException;

import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

import org.apache.struts.action.ActionMessages;
import org.apache.struts.action.ActionMessage;
import org.apache.struts.action.ActionForm;
import org.apache.struts.action.ActionForward;
import org.apache.struts.action.ActionMapping;

import org.apache.struts.validator.DynaValidatorForm;

import application.http.controller.session.SessionManager;

import application.util.exceptions.IncorrectPasswordException;
import application.util.exceptions.InternalErrorException;

import application.util.controller.struts.DefaultAction;

public class ChangePasswordAction extends DefaultAction {
    public ActionForward doExecute(ActionMapping mapping, ActionForm
      form, HttpServletRequest request, HttpServletResponse
      response) throws IOException, ServletException,
      InternalErrorException {

        /* Gets data. */
        DynaValidatorForm changePasswordForm = (DynaValidatorForm)
      form;

        String oldPassword = (String)
      changePasswordForm.get("oldPassword");
        String newPassword = (String)
      changePasswordForm.get("newPassword");

        /* Prepares errors to catch. */
        ActionMessages errors = new ActionMessages();

        try {

            SessionManager.changePassword(request, oldPassword,
                    newPassword);

        } catch (IncorrectPasswordException e) {
            errors.add("oldPassword", new
            ActionMessage("errorsMessage.incorrect.password"));
        }

        /* Returns ActionForward. */
        if (errors.isEmpty()) {
            return mapping.findForward("ChangePasswordAction");
        } else {
            saveErrors(request, errors);
            return new ActionForward(mapping.getInput());
        }
    }
}
```

GivePasswordAction

```
package application.http.controller.actions.userprofile;

import java.io.IOException;

import javax.servlet.ServletException;

import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

import org.apache.struts.action.ActionMessages;
import org.apache.struts.action.ActionMessage;
import org.apache.struts.action.ActionForm;
import org.apache.struts.action.ActionForward;
import org.apache.struts.action.ActionMapping;

import org.apache.struts.validator.DynaValidatorForm;

import application.http.controller.session.SessionManager;

import application.util.exceptions.AdministratorRequiredException;
import application.util.exceptions.IncorrectPasswordException;
import application.util.exceptions.InternalErrorException;

import application.util.controller.struts.AdministratorDefaultAction;

public class GivePasswordAction extends AdministratorDefaultAction {

    public ActionForward doExecute(ActionMapping mapping, ActionForm
      form, HttpServletRequest request, HttpServletResponse
      response) throws IOException, ServletException,
      AdministratorRequiredException, InternalErrorException {

        /* Gets data. */
        DynaValidatorForm givePasswordForm = (DynaValidatorForm) form;

        String administratorPassword = (String)
      givePasswordForm.get("administratorPassword");
        String newPassword = (String)
      givePasswordForm.get("newPassword");

        /* Prepares errors to catch. */
        ActionMessages errors = new ActionMessages();

        try {

            SessionManager.givePassword(request,
            administratorPassword, newPassword);
        } catch (IncorrectPasswordException e) {
            errors.add("administratorPassword", new
            ActionMessage("errorsMessage.incorrect.password"));
        }

        /* Returns ActionForward. */
        if (errors.isEmpty()) {
            return mapping.findForward("GivePasswordAction");
        } else {
            saveErrors(request, errors);
            return new ActionForward(mapping.getInput());
        }
    }

}
```

UpdateUserProfileAction

```
package application.http.controller.actions.userprofile;

import java.io.IOException;

import javax.servlet.ServletException;

import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

import org.apache.struts.action.ActionForm;
import org.apache.struts.action.ActionForward;
import org.apache.struts.action.ActionMapping;

import org.apache.struts.validator.DynaValidatorActionForm;

import application.http.controller.session.SessionManager;

import application.model.userprofileservice.vo.UserProfileDetailsVO;

import application.util.exceptions.InternalErrorException;

import application.util.controller.struts.DefaultAction;

public class UpdateUserProfileAction extends DefaultAction {

    public ActionForward doExecute(ActionMapping mapping, ActionForm
      form, HttpServletRequest request, HttpServletResponse
      response) throws IOException, ServletException,
      InternalErrorException {

        /* Gets data. */
        DynaValidatorActionForm userProfileForm =
      (DynaValidatorActionForm) form;
```

```
        UserProfileDetailsVO userProfileDetailsVO = new
      UserProfileDetailsVO(
                (String) userProfileForm.get("firstName"),
                (String) userProfileForm.get("surname"),
                (String) userProfileForm.get("document"),
                (String) userProfileForm.get("email"),
                (String) userProfileForm.get("address"),
                (String) userProfileForm.get("details"));

        SessionManager.updateUserProfileDetails(request,
      userProfileDetailsVO);

        /* Return ActionForward. */
        return mapping.findForward("UpdateUserProfileAction");

    }

}
```

AdministratorUpdateUserProfileAction

```
package application.http.controller.actions.userprofile;

import java.io.IOException;

import javax.servlet.ServletException;

import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

import org.apache.struts.action.ActionForm;
import org.apache.struts.action.ActionForward;
import org.apache.struts.action.ActionMapping;

import org.apache.struts.validator.DynaValidatorActionForm;

import application.http.controller.session.SessionManager;

import application.model.userprofileservice.vo.UserProfileDetailsVO;
import application.model.userprofileservice.vo.UserProfileAccessVO;

import application.util.exceptions.AdministratorRequiredException;
import application.util.exceptions.InternalErrorException;

import application.util.controller.struts.AdministratorDefaultAction;

public class AdministratorUpdateUserProfileAction extends
AdministratorDefaultAction {

    public ActionForward doExecute(ActionMapping mapping, ActionForm
      form, HttpServletRequest request, HttpServletResponse
      response) throws IOException, ServletException,
      AdministratorRequiredException, InternalErrorException {

        /* Gets data. */
        DynaValidatorActionForm userProfileForm =
                (DynaValidatorActionForm) form;

        UserProfileDetailsVO userProfileDetailsVO = new
      UserProfileDetailsVO(
                  (String) userProfileForm.get("firstName"),
                  (String) userProfileForm.get("surname"),
                  (String) userProfileForm.get("document"),
                  (String) userProfileForm.get("email"),
                  (String) userProfileForm.get("address"),
                  (String) userProfileForm.get("details"));

        Boolean isAdministrator=
                (Boolean) userProfileForm.get("isAdministrator");

        if(isAdministrator != Boolean.TRUE){
            isAdministrator = Boolean.FALSE;
        }

        UserProfileAccessVO userProfileAccessVO =
                new UserProfileAccessVO(isAdministrator);

        SessionManager.updateUserProfileDetails(request,
      userProfileDetailsVO);
        SessionManager.updateUserProfileAccess(request,
      userProfileAccessVO);

        /* Return ActionForward. */
        return
      mapping.findForward("AdministratorUpdateUserProfileAction");

    }
}
```

ChangePhoneAction

```
package application.http.controller.actions.userprofile;

import java.io.IOException;

import javax.servlet.ServletException;

import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import org.apache.struts.action.ActionMessages;
import org.apache.struts.action.ActionMessage;
import org.apache.struts.action.ActionForm;
import org.apache.struts.action.ActionForward;
import org.apache.struts.action.ActionMapping;

import org.apache.struts.validator.DynaValidatorForm;

import application.http.controller.session.SessionManager;

import application.util.exceptions.IncorrectPasswordException;
import application.util.exceptions.InternalErrorException;

import application.util.controller.struts.DefaultAction;

public class ChangePhoneAction extends DefaultAction {

    public ActionForward doExecute(ActionMapping mapping, ActionForm
      form, HttpServletRequest request, HttpServletResponse
      response) throws IOException, ServletException,
      InternalErrorException {

        /* Gets data. */
        DynaValidatorForm changePasswordForm =
            (DynaValidatorForm) form;

        String password = (String) changePasswordForm.get("password");
        String phone = (String) changePasswordForm.get("phone");

        /* Prepares errors to catch. */
        ActionMessages errors = new ActionMessages();

        try {

            SessionManager.changePhone(request, password, phone);

        } catch (IncorrectPasswordException e) {
            errors.add("password", new
            ActionMessage("errorsMessage.incorrect.password"));
        }

        /* Returns ActionForward. */
        if (errors.isEmpty()) {
            return mapping.findForward("ChangePhoneAction");
        } else {
            saveErrors(request, errors);
            return new ActionForward(mapping.getInput());
        }
    }

}
```

GivePhoneAction

```
package application.http.controller.actions.userprofile;

import java.io.IOException;

import javax.servlet.ServletException;

import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

import org.apache.struts.action.ActionMessages;
import org.apache.struts.action.ActionMessage;
import org.apache.struts.action.ActionForm;
import org.apache.struts.action.ActionForward;
import org.apache.struts.action.ActionMapping;

import org.apache.struts.validator.DynaValidatorForm;

import application.http.controller.session.SessionManager;

import application.util.exceptions.AdministratorRequiredException;
import application.util.exceptions.IncorrectPasswordException;
import application.util.exceptions.InternalErrorException;

import application.util.controller.struts.AdministratorDefaultAction;

public class GivePhoneAction extends AdministratorDefaultAction {

    public ActionForward doExecute(ActionMapping mapping, ActionForm
      form, HttpServletRequest request, HttpServletResponse
      response) throws IOException, ServletException,
      AdministratorRequiredException, InternalErrorException {

        /* Gets data. */
        DynaValidatorForm givePasswordForm = (DynaValidatorForm) form;

        String password =
                (String) givePasswordForm.get("password");
        String phone = (String) givePasswordForm.get("phone");

        /* Prepares errors to catch. */
        ActionMessages errors = new ActionMessages();

        try {

            SessionManager.givePhone(request, password, phone);

        } catch (IncorrectPasswordException e) {
            errors.add("password", new
            ActionMessage("errorsMessage.incorrect.password"));
        }

        /* Returns ActionForward. */
        if (errors.isEmpty()) {
            return mapping.findForward("GivePhoneAction");
        } else {
            saveErrors(request, errors);
            return new ActionForward(mapping.getInput());
        }
    }

}
```

FindUserAction

```
package application.http.controller.actions.userprofile;

import java.io.IOException;

import java.util.ArrayList;
import java.util.Collection;
import java.util.Map;
import java.util.HashMap;

import javax.servlet.ServletException;

import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

import org.apache.struts.action.ActionForm;
import org.apache.struts.action.ActionForward;
import org.apache.struts.action.ActionMapping;

import org.apache.struts.validator.DynaValidatorForm;

import application.http.controller.session.SessionManager;

import application.util.controller.struts.DefaultAction;

import application.util.exceptions.InternalErrorException;

/**
 * Find user action. This action allows to carry out a search with
 * different criterias in user profiles.
 */
public class FindUserAction extends DefaultAction {

    private static String propertyInName;
    private static String pattern;
    private static String propertyByName;

    private static Integer startIndex;
    private static Integer count;

    /**
     *
     * @param mapping
     * @param form
     * @param request
```

```
  * @param response
  * @throws java.io.IOException
  * @throws javax.servlet.ServletException
  * @throws application.util.exceptions.InternalErrorException
  * @return
  */
 public ActionForward doExecute(ActionMapping mapping, ActionForm
   form, HttpServletRequest request, HttpServletResponse
   response) throws IOException, ServletException,
   InternalErrorException {

     /* Gets data. */
     DynaValidatorForm findUserForm = (DynaValidatorForm) form;

     this.propertyInName = (String)
   findUserForm.get("propertyInName");
     this.pattern = (String) findUserForm.get("pattern");
     this.propertyByName = (String)
   findUserForm.get("propertyByName");

     this.startIndex = (Integer) findUserForm.get("startIndex");
     this.count = (Integer) findUserForm.get("count");

     /* Prepares finding results. */
     Collection userProfileVOs = new ArrayList();

     userProfileVOs = SessionManager.findUserAdvanced(request,
                    propertyInName, pattern+"%", propertyByName,
              startIndex, count);

     /* Prepares request. */

     request.setAttribute("action",
             request.getRequestURI().replaceFirst(
             request.getContextPath()+"/","/"));

     if (userProfileVOs.size() > 0) {
             request.setAttribute("users", userProfileVOs);
     }

     /* Generate parameters for previous and next links. */
     if(count != 0) {
     Map<String, Object> previousLinkParameters =
   getPreviousLinkParameters(startIndex, count);
     if (previousLinkParameters != null) {
             doLinkParameters(previousLinkParameters);
             request.setAttribute("previous",
         previousLinkParameters);
     }
     }

     Map<String, Object> nextLinkParameters =
   getNextLinkParameters(
             startIndex, count, userProfileVOs.size());
     if (nextLinkParameters != null) {
             doLinkParameters(nextLinkParameters);
```

```
            request.setAttribute("next", nextLinkParameters);
        }

        /*Generates parameters for showing all results.*/
        Map<String, Object> printLinkParameters = new HashMap<String,
      Object>();
        printLinkParameters.put("startIndex", new Integer(1));
        printLinkParameters.put("count",  new Integer(0));
        doLinkParameters(printLinkParameters);
        request.setAttribute("all", printLinkParameters);

        /* Finishes action and returns ActionForward. */
        return mapping.findForward("FindUserAction");
    }

    /**
     * Utility for creating common parameters for "all", "next" and
     * "previous" links.
     * @param linkParameters
     */
    private final static void doLinkParameters(Map<String, Object>
      linkParameters) {

        linkParameters.put("propertyInName", propertyInName);
        linkParameters.put("pattern", pattern);
        linkParameters.put("propertyByName", propertyByName);


    }

    /**
     * Utility for creating "previous" request parameter.
     * @param startIndex
     * @param count
     * @return
     */
    private final static Map<String, Object>
      getPreviousLinkParameters(int startIndex, int count) {

        Map<String, Object> linkAttributes = null;

        if ( (startIndex-count) > 0 ) {
            linkAttributes = getCommonPreviousNextLinkParameters(
                    startIndex, count);
            linkAttributes.put("startIndex", new Integer(startIndex-
            count));
        }
        return linkAttributes;
    }

    /**
     * Utility for creating "next" request parameter.
     * @param startIndex
     * @param count
     * @param currentChunkSize
     * @return
     */
    private final static Map<String, Object> getNextLinkParameters(
      int startIndex, int count, int currentChunkSize) {

        Map<String, Object> linkAttributes = null;

        if (currentChunkSize == count) {
            linkAttributes = getCommonPreviousNextLinkParameters(
                    startIndex, count);
            linkAttributes.put("startIndex", new
                  Integer(startIndex+count));
        }
        return linkAttributes;
    }

    /**
     * Utility for creating "previous" and "next" request parameters.
     * @param startIndex
     * @param count
     * @return
     */
    private final static Map<String, Object>
      getCommonPreviousNextLinkParameters(int startIndex,
            int count) {

        Map<String, Object> linkAttributes = new HashMap<String,
      Object>();
        linkAttributes.put("count", new Integer(count));

        return linkAttributes;
    }
}
```

ViewUserProfileDetailsAction

```
package application.http.controller.actions.userprofile;

import java.io.IOException;

import javax.servlet.ServletException;

import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

import org.apache.struts.Globals;
import org.apache.struts.action.ActionForm;
import org.apache.struts.action.ActionForward;
import org.apache.struts.action.ActionMapping;

import org.apache.struts.validator.DynaValidatorActionForm;

import application.model.userprofileservice.vo.UserProfileVO;
import application.model.userprofileservice.vo.UserProfileDetailsVO;
import application.model.userprofileservice.vo.UserProfileAccessVO;

import application.http.controller.session.SessionManager;

import application.util.controller.struts.AdministratorDefaultAction;

import application.util.exceptions.AdministratorRequiredException;
import application.util.exceptions.InstanceNotFoundException;
import application.util.exceptions.InternalErrorException;

/**
 * View user details action allows an administrator to view profiles
 * of users not loaded in session.
 */
public class ViewUserDetailsAction extends AdministratorDefaultAction
{

    /**
     *
     * @param mapping
     * @param form
     * @param request
     * @param response
     * @throws java.io.IOException
     * @throws javax.servlet.ServletException
     * @throws
     * application.util.exceptions.AdministratorRequiredException
     * @throws application.util.exceptions.InternalErrorException
     * @return
     */
    public ActionForward doExecute(ActionMapping mapping, ActionForm
      form, HttpServletRequest request, HttpServletResponse
      response) throws IOException, ServletException,
      AdministratorRequiredException,
            InternalErrorException {

        /* Gets data from form. */
        DynaValidatorActionForm userProfileForm =
                (DynaValidatorActionForm) form;

        String loginName = (String) userProfileForm.get("loginName");

        /* If the request is to allow the user to correct errors in
         * the form, "userProfileForm" must not be modified. */

        if (request.getAttribute(Globals.ERROR_KEY) == null) {

            /* If the user is updating his/her profile, set the rest
             * of attributes. */

            try{
            UserProfileVO userProfileVO =
                    SessionManager.findUser(request, loginName);


            UserProfileDetailsVO userProfileDetailsVO =
                    userProfileVO.getUserProfileDetailsVO();

            UserProfileAccessVO userProfileAccessVO =
                    userProfileVO.getUserProfileAccessVO();


            userProfileForm.set("firstName",
                        userProfileDetailsVO.getFirstName());
            userProfileForm.set("surname",
                  userProfileDetailsVO.getSurname());
            userProfileForm.set("document",
                  userProfileDetailsVO.getDocument());
            userProfileForm.set("email",
                  userProfileDetailsVO.getEmail());
            userProfileForm.set("address",
                  userProfileDetailsVO.getAddress());
            userProfileForm.set("details",
                  userProfileDetailsVO.getDetails());

            userProfileForm.set("phone", userProfileVO.getPhone());

            userProfileForm.set("isAdministrator",
                    userProfileAccessVO.getIsAdministrator());


            } catch (InstanceNotFoundException e) {
                throw new InternalErrorException(e);
            }


        }

        return new
          ActionForward(mapping.findForward("ViewUserDetailsAction"));

    }

}
```

10.5.4.2.c Acciones del servicio de solicitud de vacaciones

10.5.4.2.c.1 application.http.controller.actions.holidaysservice

AdministratorFindHolidaysRequestAction

```
package application.http.controller.actions.holidaysservice;

import java.io.IOException;
import java.util.Calendar;
import java.util.Collection;
import java.util.HashMap;
import java.util.Map;

import java.util.ArrayList;
import java.util.Iterator;

import javax.servlet.ServletException;

import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

import org.apache.struts.action.ActionForm;
import org.apache.struts.action.ActionForward;
import org.apache.struts.action.ActionMapping;

import org.apache.struts.validator.DynaValidatorForm;

import application.model.holidaysrequestservice.vo.holidaysrequest.
HolidaysRequestVO;
import
application.model.holidaysrequestservice.facade.vo.HolidaysProfileVO;
import application.model.holidaysrequestservice.facade.vo.HolidaysVO;

import application.util.controller.struts.AdministratorDefaultAction;

import application.util.exceptions.AdministratorRequiredException;
import application.util.exceptions.InstanceNotFoundException;
import application.util.exceptions.InternalErrorException;

import application.http.controller.session.SessionManager;

/**
 * Admministrator find holidays request action carries out an advanced
 * search in all user holidays requests.
 */
public class AdministratorFindHolidaysRequestAction extends
      AdministratorDefaultAction {

    private static Boolean isUserLoaded;
    private static String autoId;
    private static String period;
    private static String sinceDate;
    private static Short state;
    private static String propertyByName;
    private static Boolean isDesc;
    private static Integer startIndex;
    private static Integer count;

    /**
     *
     * @param mapping
     * @param form
     * @param request
```

```
  * @param response
  * @throws java.io.IOException
  * @throws javax.servlet.ServletException
  * @throws
  * application.util.exceptions.AdministratorRequiredException
  * @throws application.util.exceptions.InternalErrorException
  * @return
  */
public ActionForward doExecute(ActionMapping mapping, ActionForm
  form, HttpServletRequest request, HttpServletResponse
  response) throws IOException, ServletException,
  AdministratorRequiredException, InternalErrorException {

    /* Gets data. */
    DynaValidatorForm findHolidaysRequestForm =
            (DynaValidatorForm) form;

    this.isUserLoaded = (Boolean)
  findHolidaysRequestForm.get("isUserLoaded");
    this.autoId = (String) findHolidaysRequestForm.get("autoId");
    this.period = (String) findHolidaysRequestForm.get("period");
    this.sinceDate = (String)
  findHolidaysRequestForm.get("sinceDate");
    this.state = (Short) findHolidaysRequestForm.get("state");
    this.propertyByName = (String)
  findHolidaysRequestForm.get("propertyByName");
this.isDesc = (Boolean) findHolidaysRequestForm.get("isDesc");

    this.startIndex = (Integer)
  findHolidaysRequestForm.get("startIndex");
    this.count = (Integer) findHolidaysRequestForm.get("count");


    if (this.startIndex == null) {
        this.startIndex = 1;
    }

    if (this.count == null) {
        this.count = 0;
    }

    if(this.isDesc == null) {
        this.isDesc = false;
    }

    if(this.isUserLoaded == null) {
        this.isUserLoaded = false;
    }

    Integer autoIdParsed = null;
    if(autoId.length() > 0) {
        autoIdParsed = Integer.valueOf(autoId);
    }

    Calendar periodOfRequest = Calendar.getInstance();
    if(period.length()>0) {
        periodOfRequest.clear();
        periodOfRequest.set(Calendar.YEAR,
        Integer.valueOf(period));
    } else {
        periodOfRequest = null;
    }

    Calendar sinceDateOfRequest = Calendar.getInstance();;

    if(sinceDate.length()>0) {
        sinceDateOfRequest.clear();

        String[] sinceDateFormatted = sinceDate.split("-");

    sinceDateOfRequest.set(Integer.valueOf(sinceDateFormatted[2]),
                Integer.valueOf(sinceDateFormatted[1])-1,
                Integer.valueOf(sinceDateFormatted[0]));

    } else {
        sinceDateOfRequest = null;
    }


    /* Prepares finding result. */
    Collection holidaysRequestVOs;
    Collection<HolidaysVO> holidaysVOs = new
  ArrayList<HolidaysVO>();

    holidaysRequestVOs =
            SessionManager.findAdvancedHolidaysRequest(request,
        isUserLoaded, autoIdParsed, periodOfRequest,
  sinceDateOfRequest, state, propertyByName, isDesc,
  startIndex, count);

    if(holidaysRequestVOs.size()>0) {

        HolidaysProfileVO holidaysProfileVO;


            for (Iterator iterator1 =
                    holidaysRequestVOs.iterator();
              iterator1.hasNext(); ) {

                 HolidaysRequestVO holidaysRequestVO =
                        (HolidaysRequestVO) iterator1.next();

            try {
            holidaysProfileVO =
        SessionManager.findUserHolidaysProfilePeriod(request,
              holidaysRequestVO.getLoginName(),
        holidaysRequestVO.getPeriod());
            }  catch (InstanceNotFoundException e) {
                throw new InternalErrorException(e);
            }
            holidaysVOs.add(new HolidaysVO(holidaysRequestVO,
        holidaysProfileVO));
            }
    }


            request.setAttribute("action",
            request.getRequestURI().replaceFirst(
            request.getContextPath()+"/","/"));

    if (holidaysVOs.size() > 0) {
            request.setAttribute("holidayss", holidaysVOs);
    }

    /*Generates parameters for repeating finding. */

    Map<String, Object> repeatFindParameters = new HashMap<String,
        Object>();
    doLinkParameters(repeatFindParameters);
    SessionManager.storeLastFindHolidaysRequest(request,
  repeatFindParameters);

    /* Generates parameters for previous and next links. */
    if(count != 0) {

    Map<String, Object> previousLinkParameters =
  getPreviousLinkParameters(
            startIndex, count);
    if (previousLinkParameters != null) {
        doLinkParameters(previousLinkParameters);
            request.setAttribute("previous",
              previousLinkParameters);
    }

    }

    Map<String, Object> nextLinkParameters =
  getNextLinkParameters(
            startIndex, count, holidaysRequestVOs.size());
    if (nextLinkParameters != null) {
            doLinkParameters(nextLinkParameters);
            request.setAttribute("next", nextLinkParameters);
    }

    /*Generates parameters for showing all results.*/
    Map<String, Object> printLinkParameters = new HashMap<String,
  Object>();
    doLinkParameters(printLinkParameters);
    printLinkParameters.put("startIndex", new Integer(1));
    printLinkParameters.put("count",  new Integer(0));
    request.setAttribute("all", printLinkParameters);

    /* Finishes action and returns ActionForward. */
    return
   mapping.findForward("AdministratorFindHolidaysRequestAction");
```

```
}

private final static void doLinkParameters(Map<String, Object>
  linkParameters) {

    linkParameters.put("isUserLoaded", isUserLoaded);
    if(autoId.length()>0) {
    linkParameters.put("autoId", autoId);
    }
    if(period.length()>0){
    linkParameters.put("period", period);
    }
    if(sinceDate.length()>0){
    linkParameters.put("sinceDate", sinceDate);
    }
    if(!(state == null)) {
    linkParameters.put("state", state);
    }
    linkParameters.put("propertyByName", propertyByName);
    linkParameters.put("isDesc", isDesc);

}

private final static Map<String, Object>
  getPreviousLinkParameters(int startIndex, int count) {

    Map<String, Object> linkAttributes = null;

    if ( (startIndex-count) > 0 ) {
        linkAttributes = getCommonPreviousNextLinkParameters(
                startIndex, count);
        linkAttributes.put("startIndex", new Integer(startIndex-
        count));
    }
    return linkAttributes;
}

private final static Map<String, Object> getNextLinkParameters(
  int startIndex, int count, int currentChunkSize) {

    Map<String, Object> linkAttributes = null;

    if (currentChunkSize == count) {
        linkAttributes = getCommonPreviousNextLinkParameters(
                startIndex, count);
        linkAttributes.put("startIndex", new
        Integer(startIndex+count));
    }
    return linkAttributes;
}

private final static Map<String, Object>
  getCommonPreviousNextLinkParameters(int startIndex,
        int count) {
```

```
        Map<String, Object> linkAttributes = new HashMap<String,
      Object>();
        linkAttributes.put("count", new Integer(count));

        return linkAttributes;
    }

}
```

__CancelHolidaysRequestAction__

```
package application.http.controller.actions.holidaysservice;

import java.io.IOException;

import java.util.Calendar;

import javax.servlet.ServletException;

import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

import org.apache.struts.action.ActionForm;
import org.apache.struts.action.ActionForward;
import org.apache.struts.action.ActionMapping;

import org.apache.struts.validator.DynaValidatorActionForm;

import application.http.controller.session.SessionManager;

import application.model.holidaysrequestservice.vo.holidaysservice.
HolidaysServiceVO;

import application.util.exceptions.InstanceNotFoundException;
import application.util.exceptions.InternalErrorException;

import application.util.controller.struts.DefaultAction;

/**
 * Cancel holidays request action allows an user to cancel his
 * requests.
 */
public class CancelHolidaysRequestAction extends DefaultAction {

    /**
     *
     * @param mapping
     * @param form
     * @param request
     * @param response
     * @throws java.io.IOException
     * @throws javax.servlet.ServletException
     * @throws application.util.exceptions.InternalErrorException
     * @return
     */
    public ActionForward doExecute(ActionMapping mapping, ActionForm
      form, HttpServletRequest request, HttpServletResponse
      response) throws IOException, ServletException,
      InternalErrorException {

        /* Gets data. */
        DynaValidatorActionForm holidaysRequestForm =
      (DynaValidatorActionForm) form;

        String autoId = (String) holidaysRequestForm.get("autoId");
        try {

            SessionManager.cancelHolidaysRequest(request,
            Integer.valueOf(autoId));

        } catch (InstanceNotFoundException e) {
            throw new InternalErrorException(e);
        }

        /* Return ActionForward. */
        return mapping.findForward("CancelHolidaysRequestAction");

    }

}
```

EditHolidaysRequestAction

```
package application.http.controller.actions.holidaysservice;

import java.io.IOException;

import java.util.Calendar;

import javax.servlet.ServletException;

import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

import org.apache.struts.Globals;
import org.apache.struts.action.ActionForm;
import org.apache.struts.action.ActionForward;
import org.apache.struts.action.ActionMapping;

import org.apache.struts.validator.DynaValidatorActionForm;

import application.model.holidaysrequestservice.vo.holidaysrequest.
HolidaysRequestVO;

import application.http.controller.session.SessionManager;
import application.util.controller.struts.AdministratorDefaultAction;

import application.util.exceptions.AdministratorRequiredException;
import application.util.exceptions.InstanceNotFoundException;
import application.util.exceptions.InternalErrorException;

/**
 * Edit holidays request action prepares holidays request data.
 * Administrator required.
 */
public class EditHolidaysRequestAction extends
      AdministratorDefaultAction {

    /**
     *
     * @param mapping
     * @param form
     * @param request
     * @param response
     * @throws java.io.IOException
     * @throws javax.servlet.ServletException
     * @throws
     * application.util.exceptions.AdministratorRequiredException
     * @throws application.util.exceptions.InternalErrorException
     * @return
     */
    public ActionForward doExecute(ActionMapping mapping, ActionForm
      form, HttpServletRequest request, HttpServletResponse
      response) throws IOException, ServletException,
      AdministratorRequiredException, InternalErrorException {

        /* Gets data from form. */

        DynaValidatorActionForm holidaysRequestForm =
                (DynaValidatorActionForm) form;

        String autoId = (String) holidaysRequestForm.get("autoId");

        Integer id = null;

        if(autoId.length()== 0) {
            id = (Integer) request.getAttribute("autoId");
        } else {
            id = Integer.valueOf(autoId);
        }

        /* If the request is to allow the user to correct errors in
* the form, "holidaysReeqUestForm" must not be modified. */

        if (request.getAttribute(Globals.ERROR_KEY) == null) {

            try{

            HolidaysRequestVO holidaysRequestVO =
                    SessionManager.findHolidaysRequest(request, id);
            request.setAttribute("holidaysRequest",
            holidaysRequestVO);

            } catch (InstanceNotFoundException e) {
                throw new InternalErrorException(e);
            }

        }

        return new
      ActionForward(mapping.findForward("EditHolidaysRequestAction"));

    }

}
```

EditHolidaysServiceAction

```
package application.http.controller.actions.holidaysservice;

import java.io.IOException;

import java.util.Calendar;

import javax.servlet.ServletException;

import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

import org.apache.struts.Globals;
import org.apache.struts.action.ActionForm;
import org.apache.struts.action.ActionForward;
import org.apache.struts.action.ActionMapping;

import org.apache.struts.validator.DynaValidatorActionForm;

import application.model.holidaysrequestservice.vo.holidaysservice.
HolidaysServiceVO;

import application.http.controller.session.SessionManager;

import application.util.controller.struts.DefaultAction;

import application.util.exceptions.InstanceNotFoundException;
import application.util.exceptions.InternalErrorException;

/**
 * Edit holidays service action prepares user holidays service data
 * for using it.
 */
public class EditHolidaysServiceAction extends DefaultAction {

    /**
```

```
     *
     * @param mapping
     * @param form
     * @param request
     * @param response
     * @throws java.io.IOException
     * @throws javax.servlet.ServletException
     * @throws application.util.exceptions.InternalErrorException
     * @return
     */
    public ActionForward doExecute(ActionMapping mapping, ActionForm
      form, HttpServletRequest request, HttpServletResponse
      response) throws IOException, ServletException,
      InternalErrorException {

        /* Gets data from form. */

        DynaValidatorActionForm holidaysServiceForm =
                (DynaValidatorActionForm) form;

        String period = (String) holidaysServiceForm.get("period");

        Calendar date = Calendar.getInstance();
        date.clear();
        date.set(Calendar.YEAR, Integer.parseInt(period));

        /* If the request is to allow the user to correct errors in
         * the form, "holidaysServiceForm" must not be modified. */

        if (request.getAttribute(Globals.ERROR_KEY) == null) {

            try {
            HolidaysServiceVO holidaysServiceVO =
                    SessionManager.findHolidaysService(request, date);


            holidaysServiceForm.set("loginName",
            holidaysServiceVO.getLoginName());

            holidaysServiceForm.set("period",
    String.valueOf(holidaysServiceVO.getPeriod().get(Calendar.YEAR)));
            holidaysServiceForm.set("days",
            String.valueOf(holidaysServiceVO.getDays()));
            } catch(InstanceNotFoundException e) {
                throw new InternalErrorException(e);
            }

        }

        return new
      ActionForward(mapping.findForward("EditHolidaysServiceAction"));

    }

}
```

EditUserHolidaysRequestAction

```
package application.http.controller.actions.holidaysservice;

import java.io.IOException;

import java.util.Calendar;

import javax.servlet.ServletException;

import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

import org.apache.struts.Globals;
import org.apache.struts.action.ActionForm;
import org.apache.struts.action.ActionForward;
import org.apache.struts.action.ActionMapping;

import org.apache.struts.validator.DynaValidatorActionForm;

import application.model.holidaysrequestservice.vo.holidaysrequest.
HolidaysRequestVO;

import application.http.controller.session.SessionManager;

import application.util.controller.struts.DefaultAction;

import application.util.exceptions.InstanceNotFoundException;
import application.util.exceptions.InternalErrorException;

/**
 * Edit user holidays request action prepares holidays request data.
 */
public class EditUserHolidaysRequestAction extends DefaultAction {

    /**
     *
     * @param mapping
     * @param form
     * @param request
     * @param response
     * @throws java.io.IOException
     * @throws javax.servlet.ServletException
     * @throws application.util.exceptions.InternalErrorException
     * @return
     */
    public ActionForward doExecute(ActionMapping mapping, ActionForm
      form, HttpServletRequest request, HttpServletResponse
      response) throws IOException, ServletException,
      InternalErrorException {

        /* Gets data from form. */

        DynaValidatorActionForm holidaysRequestForm =
                (DynaValidatorActionForm) form;
        String autoId = (String) holidaysRequestForm.get("autoId");

        Integer id = null;

        if(autoId.length()== 0) {
            id = (Integer) request.getAttribute("autoId");
        } else {
            id = Integer.valueOf(autoId);
        }


        if (request.getAttribute(Globals.ERROR_KEY) == null) {


            try{

            HolidaysRequestVO holidaysRequestVO =
                    SessionManager.findUserHolidaysRequest(request,
                  id);

            request.setAttribute("holidaysRequest",
            holidaysRequestVO);

            } catch (InstanceNotFoundException e) {
                throw new InternalErrorException(e);
            }

        }

        return new
      ActionForward(mapping.findForward("EditHolidaysRequestAction"));

    }

}
```

<u>FindHolidaysProfileAction</u>

```
package application.http.controller.actions.holidaysservice;

import java.io.IOException;

import java.util.Calendar;
import java.util.Collection;
import java.util.HashMap;
import java.util.Map;

import javax.servlet.ServletException;

import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import org.apache.struts.action.ActionForm;
import org.apache.struts.action.ActionForward;
import org.apache.struts.action.ActionMapping;

import org.apache.struts.validator.DynaValidatorActionForm;

import application.util.controller.struts.DefaultAction;

import application.util.exceptions.InternalErrorException;

import application.http.controller.session.SessionManager;

/**
 * Finds holidays profile action finds all user holidays profiles.
 */
public class FindHolidaysProfileAction extends DefaultAction {

    private static Integer startIndex;
    private static Integer count;

    /**
     *
     * @param mapping
     * @param form
     * @param request
     * @param response
     * @throws java.io.IOException
     * @throws javax.servlet.ServletException
     * @throws application.util.exceptions.InternalErrorException
     * @return
     */
    public ActionForward doExecute(ActionMapping mapping, ActionForm
      form, HttpServletRequest request, HttpServletResponse
      response) throws IOException, ServletException,
      InternalErrorException {

        /* Gets data. */
        DynaValidatorActionForm findHolidaysProfileForm =
                (DynaValidatorActionForm) form;

        this.startIndex = (Integer)
      findHolidaysProfileForm.get("startIndex");
        this.count = (Integer) findHolidaysProfileForm.get("count");

        if (this.startIndex == null) {
            this.startIndex = 1;
        }

        if (this.count == null) {
            this.count = 0;
        }

        /* Prepares finding result. */
        Collection holidaysProfileVOs;

        holidaysProfileVOs =
                SessionManager.findHolidaysProfile(request,
            startIndex, count);

                request.setAttribute("action",
                request.getRequestURI().replaceFirst(
                request.getContextPath()+"/","/"));

        if (holidaysProfileVOs.size() > 0) {
                request.setAttribute("holidaysProfiles",
                  holidaysProfileVOs);
        }

        /* Generates parameters for previous and next links. */
        if(count != 0) {

        Map<String, Object> previousLinkParameters =
      getPreviousLinkParameters(
                startIndex, count);
        if (previousLinkParameters != null) {
            request.setAttribute("previous", previousLinkParameters);
        }

        }

        Map<String, Object> nextLinkParameters =
      getNextLinkParameters(
                startIndex, count, holidaysProfileVOs.size());
        if (nextLinkParameters != null) {
                request.setAttribute("next", nextLinkParameters);
        }

        /*Generates parameters for showing all results.*/
        Map<String, Object> printLinkParameters = new HashMap<String,
      Object>();
        printLinkParameters.put("startIndex", new Integer(1));
        printLinkParameters.put("count",  new Integer(0));
        request.setAttribute("all", printLinkParameters);

        /* Finishes action and returns ActionForward. */
        return mapping.findForward("FindHolidaysProfileAction");
    }

    private final static Map<String, Object>
      getPreviousLinkParameters(int startIndex,
            int count) {

        Map<String, Object> linkAttributes = null;

        if ( (startIndex-count) > 0 ) {
            linkAttributes = getCommonPreviousNextLinkParameters(
                    startIndex, count);
            linkAttributes.put("startIndex", new Integer(startIndex-
            count));
        }
        return linkAttributes;
    }
```

```
    private final static Map<String, Object> getNextLinkParameters(
      int startIndex, int count,
            int currentChunkSize) {

        Map<String, Object> linkAttributes = null;

        if (currentChunkSize == count) {
            linkAttributes = getCommonPreviousNextLinkParameters(
                    startIndex, count);
            linkAttributes.put("startIndex", new
            Integer(startIndex+count));
        }
        return linkAttributes;
    }

    private final static Map<String, Object>
      getCommonPreviousNextLinkParameters(int startIndex,
            int count) {

        Map<String, Object> linkAttributes = new HashMap<String,
      Object>();
        linkAttributes.put("count", new Integer(count));

        return linkAttributes;
    }

}
```

FindHolidaysProfilesAction

```
package application.http.controller.actions.holidaysservice;

import java.io.IOException;

import java.util.Collection;
import java.util.HashMap;
import java.util.Map;

import javax.servlet.ServletException;

import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

import org.apache.struts.action.ActionForm;
import org.apache.struts.action.ActionForward;
import org.apache.struts.action.ActionMapping;

import org.apache.struts.validator.DynaValidatorActionForm;

import application.util.controller.struts.AdministratorDefaultAction;

import application.util.exceptions.AdministratorRequiredException;
import application.util.exceptions.InternalErrorException;

import application.http.controller.session.SessionManager;

/**
 * Find holidays profiles action returns all periods with at least one
 * user holidays profile. Administrator required.
 */
public class FindHolidaysProfilesAction extends
AdministratorDefaultAction {

    private static Integer startIndex;
    private static Integer count;

    /**
     *
     * @return
     * @param mapping
     * @param form
     * @param request
     * @param response
     * @throws
application.util.exceptions.AdministratorRequiredException
     * @throws java.io.IOException
     * @throws javax.servlet.ServletException
     * @throws application.util.exceptions.InternalErrorException
     */
    public ActionForward doExecute(ActionMapping mapping, ActionForm
      form, HttpServletRequest request, HttpServletResponse
      response) throws IOException, ServletException,
      AdministratorRequiredException,
            InternalErrorException {


        /* Prepares finding result. */
        Collection periods =
      SessionManager.findHolidaysProfiles(request, 1,
                0);

        if (periods.size() > 0) {
                request.setAttribute("periods", periods);
        }

        /* Finishes action and returns ActionForward. */
        return mapping.findForward("FindHolidaysProfilesAction");
    }

}
```

FindHolidaysRequestAction

```
package application.http.controller.actions.holidaysservice;
```

```
import java.io.IOException;

import java.util.Calendar;
import java.util.Collection;
import java.util.HashMap;
import java.util.Map;

import java.util.ArrayList;
import java.util.Iterator;

import javax.servlet.ServletException;

import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

import org.apache.struts.action.ActionForm;
import org.apache.struts.action.ActionForward;
import org.apache.struts.action.ActionMapping;

import org.apache.struts.validator.DynaValidatorForm;

import application.util.controller.struts.DefaultAction;

import application.model.holidaysrequestservice.vo.holidaysrequest.
HolidaysRequestVO;
import
application.model.holidaysrequestservice.facade.vo.HolidaysProfileVO;
import application.model.holidaysrequestservice.facade.vo.HolidaysVO;

import application.util.exceptions.InstanceNotFoundException;
import application.util.exceptions.InternalErrorException;

import application.http.controller.session.SessionManager;

/**
 * Find holidays request action carries out an advanced search in an
 * user holidays requests.
 */
public class FindHolidaysRequestAction extends DefaultAction {

    private static String autoId;
    private static String period;
    private static String sinceDate;
    private static Short state;
    private static String propertyByName;
    private static Boolean isDesc;
    private static Integer startIndex;
    private static Integer count;

    /**
     *
     * @param mapping
     * @param form
     * @param request
     * @param response
     * @throws java.io.IOException
```

```
  * @throws javax.servlet.ServletException
  * @throws application.util.exceptions.InternalErrorException
  * @return
  */
public ActionForward doExecute(ActionMapping mapping, ActionForm
  form, HttpServletRequest request, HttpServletResponse
  response) throws
        IOException, ServletException, InternalErrorException {

    /* Gets data. */
    DynaValidatorForm findHolidaysRequestForm =
            (DynaValidatorForm) form;

    this.autoId = (String) findHolidaysRequestForm.get("autoId");
    this.period = (String) findHolidaysRequestForm.get("period");
    this.sinceDate = (String)
  findHolidaysRequestForm.get("sinceDate");
    this.state = (Short) findHolidaysRequestForm.get("state");
    this.propertyByName = (String)
  findHolidaysRequestForm.get("propertyByName");
    this.isDesc = (Boolean) findHolidaysRequestForm.get("isDesc");
    this.startIndex = (Integer)
  findHolidaysRequestForm.get("startIndex");
    this.count = (Integer) findHolidaysRequestForm.get("count");

    if (this.startIndex == null) {
        this.startIndex = 1;
    }

    if (this.count == null) {
        this.count = 0;
    }

    if(this.isDesc == null) {
        this.isDesc = false;
    }

    Integer autoIdParsed = null;
    if(autoId.length() > 0) {
        autoIdParsed = Integer.valueOf(autoId);
    }

    Calendar periodOfRequest = Calendar.getInstance();

    if(period.length()>0) {
        periodOfRequest.clear();
        periodOfRequest.set(Calendar.YEAR,
        Integer.valueOf(period));
    } else {
        periodOfRequest = null;
    }

    Calendar sinceDateOfRequest = Calendar.getInstance();;

    if(sinceDate.length()>0) {
        sinceDateOfRequest.clear();
        String[] sinceDateFormatted = sinceDate.split("-");

    sinceDateOfRequest.set(Integer.valueOf(sinceDateFormatted[2]),
                Integer.valueOf(sinceDateFormatted[1])-1,
                Integer.valueOf(sinceDateFormatted[0]));

    } else {
        sinceDateOfRequest = null;
    }


    /* Prepares finding result. */
    Collection holidaysRequestVOs;
    Collection<HolidaysVO> holidaysVOs = new
  ArrayList<HolidaysVO>();

    holidaysRequestVOs =
           SessionManager.findAdvancedUserHolidaysRequest(request,
              autoIdParsed, periodOfRequest, sinceDateOfRequest,
        state, propertyByName, isDesc, startIndex, count);

    if(holidaysRequestVOs.size()>0) {

        HolidaysProfileVO holidaysProfileVO;


            for (Iterator iterator1 =
                    holidaysRequestVOs.iterator();
              iterator1.hasNext(); ) {

                HolidaysRequestVO holidaysRequestVO =
                       (HolidaysRequestVO) iterator1.next();

            try {
            holidaysProfileVO =
        SessionManager.findHolidaysProfilePeriod(request,
        holidaysRequestVO.getPeriod());
            }  catch (InstanceNotFoundException e) {
                throw new InternalErrorException(e);
            }

            holidaysVOs.add(new HolidaysVO(holidaysRequestVO,
        holidaysProfileVO));
            }
    }


            request.setAttribute("action",
            request.getRequestURI().replaceFirst(
            request.getContextPath()+"/","/"));

    if (holidaysVOs.size() > 0) {
            request.setAttribute("holidayss", holidaysVOs);
    }
    /*Generates parameters for repeating finding. */

    Map<String, Object> repeatFindParameters = new HashMap<String,
        Object>();
    doLinkParameters(repeatFindParameters);
    SessionManager.storeLastFindHolidaysRequest(request,
  repeatFindParameters);

    /* Generates parameters for previous and next links. */
    if(count != 0) {

    Map<String, Object> previousLinkParameters =
  getPreviousLinkParameters(
            startIndex, count);
    if (previousLinkParameters != null) {
        doLinkParameters(previousLinkParameters);
            request.setAttribute("previous",
        previousLinkParameters);
    }

    }

    Map<String, Object> nextLinkParameters =
  getNextLinkParameters(
            startIndex, count, holidaysRequestVOs.size());
    if (nextLinkParameters != null) {
            doLinkParameters(nextLinkParameters);
            request.setAttribute("next", nextLinkParameters);
    }

    /*Generates parameters for showing all results.*/
    Map<String, Object> printLinkParameters = new HashMap<String,
  Object>();
    doLinkParameters(printLinkParameters);
    printLinkParameters.put("startIndex", new Integer(1));
    printLinkParameters.put("count",  new Integer(0));
    request.setAttribute("all", printLinkParameters);

    /* Finishes action and returns ActionForward. */
    return mapping.findForward("FindHolidaysRequestAction");

}

private final static void doLinkParameters(Map<String, Object>
  linkParameters) {

    if(autoId.length()>0) {
    linkParameters.put("autoId", autoId);
    }
    if(period.length()>0){
    linkParameters.put("period", period);
    }
    if(sinceDate.length()>0){
    linkParameters.put("sinceDate", sinceDate);
    }
    if(!(state == null)) {
        linkParameters.put("state", state);
        }
        linkParameters.put("propertyByName", propertyByName);
        linkParameters.put("isDesc", isDesc);

    }

    private final static Map<String, Object>
      getPreviousLinkParameters(int startIndex,
            int count) {

        Map<String, Object> linkAttributes = null;

        if ( (startIndex-count) > 0 ) {
            linkAttributes = getCommonPreviousNextLinkParameters(
                    startIndex, count);
            linkAttributes.put("startIndex", new Integer(startIndex-
            count));
        }
        return linkAttributes;
    }

    private final static Map<String, Object> getNextLinkParameters(
      int startIndex, int count,
            int currentChunkSize) {

        Map<String, Object> linkAttributes = null;

        if (currentChunkSize == count) {
            linkAttributes = getCommonPreviousNextLinkParameters(
                    startIndex, count);
            linkAttributes.put("startIndex", new
            Integer(startIndex+count));
        }
        return linkAttributes;
    }

    private final static Map<String, Object>
      getCommonPreviousNextLinkParameters(int startIndex,
            int count) {

        Map<String, Object> linkAttributes = new HashMap<String,
      Object>();
        linkAttributes.put("count", new Integer(count));

        return linkAttributes;
    }

}
```

FindHolidaysServicesAction

```
package application.http.controller.actions.holidaysservice;
```

```java
import java.io.IOException;

import java.util.Collection;
import java.util.HashMap;
import java.util.Map;

import javax.servlet.ServletException;

import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

import org.apache.struts.action.ActionForm;
import org.apache.struts.action.ActionForward;
import org.apache.struts.action.ActionMapping;

import org.apache.struts.validator.DynaValidatorActionForm;

import application.util.controller.struts.AdministratorDefaultAction;

import application.util.exceptions.AdministratorRequiredException;
import application.util.exceptions.InternalErrorException;

import application.http.controller.session.SessionManager;

/**
 * Find holidays services action provides a list of periods with at
 * least an user holidays service.
 */
public class FindHolidaysServicesAction extends
      AdministratorDefaultAction {

    private static Integer startIndex;
    private static Integer count;

    /**
     *
     * @param mapping
     * @param form
     * @param request
     * @param response
     * @throws java.io.IOException
     * @throws javax.servlet.ServletException
     * @throws
     * application.util.exceptions.AdministratorRequiredException
     * @throws application.util.exceptions.InternalErrorException
     * @return
     */
    public ActionForward doExecute(ActionMapping mapping, ActionForm
      form, HttpServletRequest request, HttpServletResponse
      response) throws IOException, ServletException,
      AdministratorRequiredException, InternalErrorException {

        /* Gets data. */
        DynaValidatorActionForm findHolidaysServiceForm =
                (DynaValidatorActionForm) form;
        this.startIndex = (Integer)
      findHolidaysServiceForm.get("startIndex");
        this.count = (Integer) findHolidaysServiceForm.get("count");

        if (this.startIndex == null) {
            this.startIndex = 1;
        }

        if (this.count == null) {
            this.count = 0;
        }

        /* Prepares finding result. */
        Collection periods;

        periods = SessionManager.findHolidaysServices(request,
                  startIndex, count);

                request.setAttribute("action",
                request.getRequestURI().replaceFirst(
                request.getContextPath()+"/","/"));

        if (periods.size() > 0) {
                request.setAttribute("periods", periods);
        }

        /* Generate parameters for previous and next links. */
        if(count != 0) {

        Map previousLinkParameters = getPreviousLinkParameters(
                startIndex, count);
        if (previousLinkParameters != null) {
                request.setAttribute("previous",
            previousLinkParameters);
        }

        }

        Map nextLinkParameters = getNextLinkParameters(
                startIndex, count, periods.size());
        if (nextLinkParameters != null) {
                request.setAttribute("next", nextLinkParameters);
        }

        /*Generates parameters for showing all results.*/
        Map<String, Object> printLinkParameters = new HashMap<String,
      Object>();
        printLinkParameters.put("startIndex", new Integer(1));
        printLinkParameters.put("count",  new Integer(0));
        request.setAttribute("all", printLinkParameters);

        /* Finishes action and returns ActionForward. */
        return mapping.findForward("FindHolidaysServicesAction");

    }
```

```
/**
 *
 * @param startIndex
 * @param count
 * @return
 */
private final static Map getPreviousLinkParameters(int startIndex,
        int count) {

    Map<String, Object> linkAttributes = null;

    if ( (startIndex-count) > 0 ) {
        linkAttributes = getCommonPreviousNextLinkParameters(
                startIndex, count);
        linkAttributes.put("startIndex", new Integer(startIndex-
        count));
    }
    return linkAttributes;
}

/**
 *
 * @param startIndex
 * @param count
 * @param currentChunkSize
 * @return
 */
private final static Map<String, Object> getNextLinkParameters(
  int startIndex, int count,
        int currentChunkSize) {

    Map<String, Object> linkAttributes = null;

    if (currentChunkSize == count) {
        linkAttributes = getCommonPreviousNextLinkParameters(
                startIndex, count);
        linkAttributes.put("startIndex", new
        Integer(startIndex+count));
    }
    return linkAttributes;
}

/**
 *
 * @param startIndex
 * @param count
 * @return
 */
private final static Map<String, Object>
  getCommonPreviousNextLinkParameters(int startIndex,
        int count) {

    Map<String, Object> linkAttributes = new HashMap<String,
  Object>();
    linkAttributes.put("count", new Integer(count));
```

```
        return linkAttributes;
    }

}
```

FindPeriodHolidaysProfileAction

```
package application.http.controller.actions.holidaysservice;

import java.io.IOException;

import java.util.Calendar;
import java.util.Collection;
import java.util.HashMap;
import java.util.Map;

import javax.servlet.ServletException;

import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

import org.apache.struts.action.ActionForm;
import org.apache.struts.action.ActionForward;
import org.apache.struts.action.ActionMapping;

import org.apache.struts.validator.DynaValidatorActionForm;

import application.util.controller.struts.AdministratorDefaultAction;

import application.util.exceptions.AdministratorRequiredException;
import application.util.exceptions.InternalErrorException;

import application.http.controller.session.SessionManager;

/**
 * Find period holidays profile action finds all user holidays profile
 * in a period given.
 */
public class FindPeriodHolidaysProfileAction extends
      AdministratorDefaultAction {

    private static String period;
    private static Integer startIndex;
    private static Integer count;

    /**
     *
     * @param mapping
     * @param form
     * @param request
     * @param response
     * @throws java.io.IOException
     * @throws javax.servlet.ServletException
     * @throws application.util.exceptions.InternalErrorException
     * @return
     */
    public ActionForward doExecute(ActionMapping mapping, ActionForm
      form, HttpServletRequest request, HttpServletResponse
      response) throws
            IOException, ServletException,
      AdministratorRequiredException, InternalErrorException {

        /* Gets data. */
        DynaValidatorActionForm findHolidaysProfileForm =
                (DynaValidatorActionForm) form;

        this.period = (String) findHolidaysProfileForm.get("period");
        this.startIndex = (Integer)
      findHolidaysProfileForm.get("startIndex");
        this.count = (Integer) findHolidaysProfileForm.get("count");


        if (this.startIndex == null) {
            this.startIndex = 1;
        }

        if (this.count == null) {
            this.count = 0;
        }

        /* Prepares finding result. */

        Calendar periodParsed = Calendar.getInstance();
        periodParsed.clear();
        periodParsed.set(Calendar.YEAR, Integer.parseInt(period));

        Collection holidaysProfileVOs;

        holidaysProfileVOs =
                SessionManager.FindPeriodHolidaysProfile(request,
            periodParsed, startIndex, count);

                request.setAttribute("action",
                request.getRequestURI().replaceFirst(
                request.getContextPath()+"/","/"));

        if (holidaysProfileVOs.size() > 0) {
                request.setAttribute("holidaysProfiles",
            holidaysProfileVOs);
        }

        /* Generate parameters for previous and next links. */
        if(count != 0) {

        Map<String, Object> previousLinkParameters =
      getPreviousLinkParameters(
                startIndex, count);
        if (previousLinkParameters != null) {
            doLinkParameters(previousLinkParameters);
                request.setAttribute("previous",
            previousLinkParameters);
        }

        }

        Map<String, Object> nextLinkParameters =
      getNextLinkParameters(
                startIndex, count, holidaysProfileVOs.size());
        if (nextLinkParameters != null) {
                doLinkParameters(nextLinkParameters);
                request.setAttribute("next", nextLinkParameters);
        }

        /*Generates parameters for showing all results.*/
        Map<String, Object> printLinkParameters = new HashMap<String,
      Object>();
        doLinkParameters(printLinkParameters);
        printLinkParameters.put("startIndex", new Integer(1));
        printLinkParameters.put("count",  new Integer(0));
        request.setAttribute("all", printLinkParameters);

        /* Finishes action and returns ActionForward. */
        return mapping.findForward("FindPeriodHolidaysProfileAction");
    }

    private final static void doLinkParameters(Map<String, Object>
      linkParameters) {
        linkParameters.put("period", period);
    }

    private final static Map<String, Object>
      getPreviousLinkParameters(int startIndex,
            int count) {

        Map<String, Object> linkAttributes = null;

        if ( (startIndex-count) > 0 ) {
            linkAttributes = getCommonPreviousNextLinkParameters(
                    startIndex, count);
            linkAttributes.put("startIndex", new Integer(startIndex-
            count));
        }
        return linkAttributes;
    }

    private final static Map<String, Object> getNextLinkParameters(
      int startIndex, int count,
            int currentChunkSize) {

        Map<String, Object> linkAttributes = null;

        if (currentChunkSize == count) {
            linkAttributes = getCommonPreviousNextLinkParameters(
                    startIndex, count);
            linkAttributes.put("startIndex", new
            Integer(startIndex+count));
        }
        return linkAttributes;
    }

    private final static Map<String, Object>
      getCommonPreviousNextLinkParameters(int startIndex,
            int count) {

        Map<String, Object> linkAttributes = new HashMap<String,
      Object>();
        linkAttributes.put("count", new Integer(count));

        return linkAttributes;
    }

}
```

FindPeriodHolidaysServiceAction

```
package application.http.controller.actions.holidaysservice;

import java.io.IOException;

import java.util.Calendar;
import java.util.Collection;
import java.util.HashMap;
import java.util.Map;

import javax.servlet.ServletException;

import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

import org.apache.struts.action.ActionForm;
import org.apache.struts.action.ActionForward;
import org.apache.struts.action.ActionMapping;

import org.apache.struts.validator.DynaValidatorActionForm;

import application.util.controller.struts.AdministratorDefaultAction;

import application.util.exceptions.AdministratorRequiredException;
import application.util.exceptions.InternalErrorException;

import application.http.controller.session.SessionManager;

/**
 * Find period holilidays service action finds all user holidays
 * services in a period given. Administrator required.
 */
```

```
public class FindPeriodHolidaysServiceAction extends
      AdministratorDefaultAction {

    private static String period;
    private static Integer startIndex;
    private static Integer count;

    /**
     *
     * @param mapping
     * @param form
     * @param request
     * @param response
     * @throws java.io.IOException
     * @throws javax.servlet.ServletException
     * @throws
     * application.util.exceptions.AdministratorRequiredException
     * @throws application.util.exceptions.InternalErrorException
     * @return
     */
    public ActionForward doExecute(ActionMapping mapping, ActionForm
      form, HttpServletRequest request, HttpServletResponse
      response) throws IOException, ServletException,
      AdministratorRequiredException,
            InternalErrorException {

        /* Gets data. */
        DynaValidatorActionForm findHolidaysServiceForm =
                (DynaValidatorActionForm) form;

        this.period = (String) findHolidaysServiceForm.get("period");
        this.startIndex = (Integer)
      findHolidaysServiceForm.get("startIndex");
        this.count = (Integer) findHolidaysServiceForm.get("count");

        Calendar date = Calendar.getInstance();
        date.clear();
        date.set(Calendar.YEAR, Integer.parseInt(period));

        if (this.startIndex == null) {
            this.startIndex = 1;
        }

        if (this.count == null) {
            this.count = 0;
        }

        /* Prepares finding result. */
        Collection holidaysServiceVOs;

        holidaysServiceVOs =
                SessionManager.findPeriodHolidaysService(request,
            date, startIndex,
                  count);

                request.setAttribute("action",
                request.getRequestURI().replaceFirst(
                request.getContextPath()+"/","/"));

        if (holidaysServiceVOs.size() > 0) {
                request.setAttribute("holidaysServices",
            holidaysServiceVOs);
        }

        /* Generate parameters for previous and next links. */
        if(count != 0) {

        Map<String, Object> previousLinkParameters =
      getPreviousLinkParameters(
                startIndex, count);
        if (previousLinkParameters != null) {
            doLinkParameters(previousLinkParameters);
                request.setAttribute("previous",
            previousLinkParameters);
        }

        }

        Map<String, Object> nextLinkParameters =
      getNextLinkParameters(
                startIndex, count, holidaysServiceVOs.size());
        if (nextLinkParameters != null) {
                doLinkParameters(nextLinkParameters);
                request.setAttribute("next", nextLinkParameters);
        }

        /*Generates parameters for showing all results.*/
        Map<String, Object> printLinkParameters = new HashMap<String,
      Object>();
        doLinkParameters(printLinkParameters);
        printLinkParameters.put("startIndex", new Integer(1));
        printLinkParameters.put("count",  new Integer(0));
        request.setAttribute("all", printLinkParameters);

        /* Finishes action and returns ActionForward. */
        return mapping.findForward("FindPeriodHolidaysServiceAction");

    }

    /**
     *
     * @param linkParameters
     */
    private final static void doLinkParameters(Map<String, Object>
      linkParameters) {

        linkParameters.put("period", period);

    }

    /**
     *
```

```
  * @param startIndex
  * @param count
  * @return
  */
private final static Map<String, Object>
  getPreviousLinkParameters(int startIndex,
        int count) {

    Map<String, Object> linkAttributes = null;

    if ( (startIndex-count) > 0 ) {
        linkAttributes = getCommonPreviousNextLinkParameters(
                startIndex, count);
        linkAttributes.put("startIndex", new Integer(startIndex-
        count));
    }
    return linkAttributes;
}

/**
 *
 * @param startIndex
 * @param count
 * @param currentChunkSize
 * @return
 */
private final static Map<String, Object> getNextLinkParameters(
  int startIndex, int count,
        int currentChunkSize) {

    Map<String, Object> linkAttributes = null;

    if (currentChunkSize == count) {
        linkAttributes = getCommonPreviousNextLinkParameters(
                startIndex, count);
        linkAttributes.put("startIndex", new
        Integer(startIndex+count));
    }
    return linkAttributes;
}

/**
 *
 * @param startIndex
 * @param count
 * @return
 */
private final static Map<String, Object>
  getCommonPreviousNextLinkParameters(int startIndex,
        int count) {

    Map<String, Object> linkAttributes = new HashMap<String,
  Object>();
    linkAttributes.put("count", new Integer(count));

    return linkAttributes;
    }

}
```

FindUserHolidaysServiceAction

```
package application.http.controller.actions.holidaysservice;

import java.io.IOException;

import java.util.Collection;
import java.util.HashMap;
import java.util.Map;

import javax.servlet.ServletException;

import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

import org.apache.struts.action.ActionForm;
import org.apache.struts.action.ActionForward;
import org.apache.struts.action.ActionMapping;

import org.apache.struts.validator.DynaValidatorActionForm;

import application.util.controller.struts.DefaultAction;

import application.util.exceptions.InternalErrorException;

import application.http.controller.session.SessionManager;

/**
 * Find user holidays service action finds all holidays service of an
 * user.
 */
public class FindUserHolidaysServiceAction extends DefaultAction {

    private static Integer startIndex;
    private static Integer count;

    /**
     *
     * @param mapping
     * @param form
     * @param request
     * @param response
     * @throws java.io.IOException
     * @throws javax.servlet.ServletException
     * @throws application.util.exceptions.InternalErrorException
     * @return
     */
```

```
public ActionForward doExecute(ActionMapping mapping, ActionForm
  form, HttpServletRequest request, HttpServletResponse
  response) throws
        IOException, ServletException, InternalErrorException {

    /* Gets data. */
    DynaValidatorActionForm findHolidaysServiceForm =
            (DynaValidatorActionForm) form;

    this.startIndex = (Integer)
  findHolidaysServiceForm.get("startIndex");
    this.count = (Integer) findHolidaysServiceForm.get("count");

    if (this.startIndex == null) {
        this.startIndex = 1;
    }

    if(this.count == null) {
        this.count = 0;
    }

    /* Prepares finding result. */
    Collection holidaysServiceVOs;

    holidaysServiceVOs =
            SessionManager.findUserHolidaysService(request,
        startIndex, count);

            request.setAttribute("action",
            request.getRequestURI().replaceFirst(
            request.getContextPath()+"/","/"));

    if (holidaysServiceVOs.size() > 0) {
            request.setAttribute("holidaysServices",
        holidaysServiceVOs);
    }

    /* Generate parameters for previous and next links. */
    if(count != 0) {

    Map previousLinkParameters = getPreviousLinkParameters(
            startIndex, count);
    if (previousLinkParameters != null) {
            request.setAttribute("previous",
        previousLinkParameters);
    }

    }

    Map nextLinkParameters = getNextLinkParameters(
            startIndex, count, holidaysServiceVOs.size());
    if (nextLinkParameters != null) {
            request.setAttribute("next", nextLinkParameters);
    }

    /*Generates parameters for showing all results.*/
```

```
        Map<String, Object> printLinkParameters = new HashMap<String,
      Object>();
        printLinkParameters.put("startIndex", new Integer(1));
        printLinkParameters.put("count",  new Integer(0));
        request.setAttribute("all", printLinkParameters);

        /* Finishes action and returns ActionForward. */
        return mapping.findForward("FindUserHolidaysServiceAction");
    }

    /**
     *
     * @param startIndex
     * @param count
     * @return
     */
    private final static Map getPreviousLinkParameters(int startIndex,
            int count) {

        Map<String, Object> linkAttributes = null;

        if ( (startIndex-count) > 0 ) {
            linkAttributes = getCommonPreviousNextLinkParameters(
                    startIndex, count);
            linkAttributes.put("startIndex", new Integer(startIndex-
      count));
        }
        return linkAttributes;
    }

    /**
     * @param startIndex
     * @param count
     * @param currentChunkSize
     * @return
     */
    private final static Map<String, Object> getNextLinkParameters(
      int startIndex, int count,
            int currentChunkSize) {

        Map<String, Object> linkAttributes = null;

        if (currentChunkSize == count) {
            linkAttributes = getCommonPreviousNextLinkParameters(
                    startIndex, count);
            linkAttributes.put("startIndex", new
           Integer(startIndex+count));
        }
        return linkAttributes;
    }

    /**
     * @param startIndex
     * @param count
     * @return
     */
    private final static Map<String, Object>
      getCommonPreviousNextLinkParameters(int startIndex,
            int count) {

        Map<String, Object> linkAttributes = new HashMap<String,
      Object>();
        linkAttributes.put("count", new Integer(count));

        return linkAttributes;
    }

}
```

MapUpdatesHolidaysRequestAction

```
package application.http.controller.actions.holidaysservice;

import java.io.IOException;

import java.util.Map;
import java.util.Set;
import java.util.Iterator;

import javax.servlet.ServletException;

import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

import org.apache.struts.action.ActionForm;
import org.apache.struts.action.ActionForward;
import org.apache.struts.action.ActionMapping;

import org.apache.struts.validator.LazyValidatorForm;

import application.http.controller.session.SessionManager;

import application.model.holidaysrequestservice.vo.holidaysservice.
HolidaysServiceVO;

import application.util.exceptions.AdministratorRequiredException;
import application.util.exceptions.InstanceNotFoundException;
import application.util.exceptions.InternalErrorException;

import application.util.controller.struts.AdministratorDefaultAction;

/**
 * Map updates holidays request action allow to update a collection of
 * users holidays request. Administrator required.
 */
public class MapUpdatesHolidaysRequestAction extends
      AdministratorDefaultAction {

    /**
```

```
     *
     * @param mapping
     * @param form
     * @param request
     * @param response
     * @throws java.io.IOException
     * @throws javax.servlet.ServletException
     * @throws application.util.exceptions.InternalErrorException
     * @return
     */
    public ActionForward doExecute(ActionMapping mapping, ActionForm
      form, HttpServletRequest request, HttpServletResponse
      response) throws IOException, ServletException,
      AdministratorRequiredException,
            InternalErrorException {

        /* Gets data. */
        LazyValidatorForm mapUpdatesForm = (LazyValidatorForm) form;

        Map<String, String> doUpdate = (Map<String, String>)
      mapUpdatesForm.get("doUpdate");

        for(Iterator iterator =
                        doUpdate.entrySet().iterator();
                  iterator.hasNext();) {

            Map.Entry entry = (Map.Entry) iterator.next();
            String autoId = (String) entry.getKey();
            String state = (String) entry.getValue();


            if(state != null){

            try {
            SessionManager.updateHolidaysRequest(request,
                  Integer.decode(autoId), Short.decode(state));
            } catch (InstanceNotFoundException e) {
                throw new InternalErrorException(e);
            }
            }
        }


        /* Returns ActionForward. */

        return mapping.findForward("MapUpdatesHolidaysRequestAction");

    }

}
```

RegisterHolidaysRequestAction

```java
package application.http.controller.actions.holidaysservice;

import java.io.IOException;

import java.util.Calendar;

import javax.servlet.ServletException;

import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

import org.apache.struts.action.ActionMessages;
import org.apache.struts.action.ActionMessage;
import org.apache.struts.action.ActionForm;
import org.apache.struts.action.ActionForward;
import org.apache.struts.action.ActionMapping;

import org.apache.struts.validator.DynaValidatorActionForm;

import application.http.controller.session.SessionManager;

import application.model.holidaysrequestservice.vo.holidaysrequest.
HolidaysRequestVO;

import application.util.exceptions.InstanceNotFoundException;
import application.util.exceptions.DuplicateInstanceException;
import application.util.exceptions.InternalErrorException;

import application.util.controller.struts.DefaultAction;

/**
 * Register holidays request action. This action has two steps; the
 * first one is to preview a new holidays request, and the second one
 * is to confirm the holidays request.
 */
public class RegisterHolidaysRequestAction extends DefaultAction {

    /**
     *
     * @param mapping
     * @param form
     * @param request
     * @param response
     * @throws java.io.IOException
     * @throws javax.servlet.ServletException
     * @throws application.util.exceptions.InternalErrorException
     * @return
     */
    public ActionForward doExecute(ActionMapping mapping, ActionForm
      form, HttpServletRequest request, HttpServletResponse
      response) throws
            IOException, ServletException, InternalErrorException {

        if(isCancelled(request)) { return new
      ActionForward(mapping.getInput());
        }else {
```

```
    /* Gets data. */
    DynaValidatorActionForm holidaysRequestForm =
  (DynaValidatorActionForm) form;

    String startDate = (String)
  holidaysRequestForm.get("startDate");
    String finalDate = (String)
  holidaysRequestForm.get("finalDate");
    String details = (String) holidaysRequestForm.get("details");
    Boolean confirm = (Boolean)
  holidaysRequestForm.get("confirm");

    String[] startDateFormatted = startDate.split("-");
    String[] finalDateFormatted = finalDate.split("-");

    Calendar date1 = Calendar.getInstance();
    date1.clear();
    date1.set(Integer.parseInt(startDateFormatted[2]),
            Integer.parseInt(startDateFormatted[1])-1,
            Integer.parseInt(startDateFormatted[0]));

    Calendar date2 = Calendar.getInstance();
    date2.clear();
    date2.set(Integer.parseInt(finalDateFormatted[2]),
            Integer.parseInt(finalDateFormatted[1])-1,
            Integer.parseInt(finalDateFormatted[0]));

    /* Prepares errors to catch. */

    ActionMessages errors = new ActionMessages();

    if(date1.get(Calendar.YEAR) != date2.get(Calendar.YEAR)) {
        errors.add("startDate", new
 ActionMessage("errorsMessage.notMachingPeriod.holidaysRequest"));
    } else {

    if (date2.before(date1)) {
        errors.add("startDate", new
       ActionMessage("errorsMessage.incorrect.holidaysrequest"));
    }

    }

    if(confirm) {
         if (errors.isEmpty()) {

            HolidaysRequestVO holidaysRequestVO = new
        HolidaysRequestVO(0, "", date1,
        Calendar.getInstance(), date1, date2,
                    Short.valueOf("0"), details);

            request.setAttribute("holidaysRequest",
        holidaysRequestVO);
         }
    } else {
            try {

                Integer autoId =
            SessionManager.registerHolidaysRequest(request,
            date1, date2, details);

                request.setAttribute("autoId", autoId);

            } catch (InstanceNotFoundException e) {
                errors.add("startDate", new
                  ActionMessage("errorsMessage.serviceNotAvaliable"));
            } catch (DuplicateInstanceException e) {
                errors.add("startDate", new
                  ActionMessage("errorsMessage.requestAlreadyExist"));
            }

        }


        /* Returns ActionForward. */
        if (errors.isEmpty()) {
            return
                  mapping.findForward("RegisterHolidaysRequestAction");
        } else {
            saveErrors(request, errors);
            return new ActionForward(mapping.getInput());
        }


        }
    }

}
```

RegisterHolidaysServiceAction

```
package application.http.controller.actions.holidaysservice;

import java.io.IOException;

import java.util.Calendar;

import javax.servlet.ServletException;

import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

import org.apache.struts.action.ActionMessages;
import org.apache.struts.action.ActionMessage;
import org.apache.struts.action.ActionForm;
import org.apache.struts.action.ActionForward;
import org.apache.struts.action.ActionMapping;
import org.apache.struts.validator.DynaValidatorActionForm;

import application.http.controller.session.SessionManager;

import application.model.userprofileservice.vo.UserProfileVO;
import application.model.userprofileservice.vo.UserProfileDetailsVO;
import application.model.userprofileservice.vo.UserProfileAccessVO;

import application.util.exceptions.AdministratorRequiredException;
import application.util.exceptions.DuplicateInstanceException;
import application.util.exceptions.InternalErrorException;

import application.util.controller.struts.AdministratorDefaultAction;

/**
 * Register holidays service action. Administrator required.
 */
public class RegisterHolidaysServiceAction extends
      AdministratorDefaultAction {

    /**
     *
     * @param mapping
     * @param form
     * @param request
     * @param response
     * @throws java.io.IOException
     * @throws javax.servlet.ServletException
     * @throws application.util.exceptions.InternalErrorException
     * @return
     */
    public ActionForward doExecute(ActionMapping mapping, ActionForm
      form, HttpServletRequest request, HttpServletResponse
      response) throws IOException, ServletException,
      AdministratorRequiredException, InternalErrorException {

        /* Gets data. */

        DynaValidatorActionForm holidaysServiceForm =
      (DynaValidatorActionForm) form;

        String period = (String) holidaysServiceForm.get("period");
        String days = (String) holidaysServiceForm.get("days");

        Calendar date = Calendar.getInstance();
        date.clear();
        date.set(Calendar.YEAR, Integer.parseInt(period));

        /* Prepares errors to catch. */

        ActionMessages errors = new ActionMessages();

        try {

            SessionManager.registerHolidaysService(request, date,
                    Integer.valueOf(days));
        } catch (DuplicateInstanceException e) {

            errors.add("period", new
                  ActionMessage("errorsMessage.duplicated.period"));
        }

        /* Return ActionForward. */
        if (errors.isEmpty()) {
            return
                  mapping.findForward("RegisterHolidaysServiceAction");
        } else {
            saveErrors(request, errors);
            return new ActionForward(mapping.getInput());
        }

    }

}
```

UpdateHolidaysServiceAction

```
package application.http.controller.actions.holidaysservice;

import java.io.IOException;

import java.util.Calendar;

import javax.servlet.ServletException;

import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

import org.apache.struts.action.ActionForm;
import org.apache.struts.action.ActionForward;
import org.apache.struts.action.ActionMapping;

import org.apache.struts.validator.DynaValidatorActionForm;

import application.http.controller.session.SessionManager;

import
application.model.holidaysrequestservice.vo.holidaysservice.HolidaysSe
rviceVO;

import application.util.exceptions.AdministratorRequiredException;
import application.util.exceptions.InstanceNotFoundException;
import application.util.exceptions.InternalErrorException;

import application.util.controller.struts.AdministratorDefaultAction;
```

```
/**
 * Update holidays service action. This action allows to change the
 * days of holidays in a user holidays service. Administrator
 * required.
 */
public class UpdateHolidaysServiceAction extends
      AdministratorDefaultAction {

    /**
     *
     * @param mapping
     * @param form
     * @param request
     * @param response
     * @throws java.io.IOException
     * @throws javax.servlet.ServletException
     * @throws application.util.exceptions.InternalErrorException
     * @return
     */
    public ActionForward doExecute(ActionMapping mapping, ActionForm
      form, HttpServletRequest request, HttpServletResponse
      response) throws IOException, ServletException,
      AdministratorRequiredException, InternalErrorException {

        /* Gets data. */

        DynaValidatorActionForm holidaysServiceForm =
      (DynaValidatorActionForm) form;

        String loginName = (String)
      holidaysServiceForm.get("loginName");
        String period = (String) holidaysServiceForm.get("period");
        String days = (String) holidaysServiceForm.get("days");

        Calendar date = Calendar.getInstance();
        date.clear();
        date.set(Calendar.YEAR, Integer.parseInt(period));

        try {
        SessionManager.updateUserHolidaysService(request, date,
      Integer.valueOf(days));
        } catch (InstanceNotFoundException e) {
            throw new InternalErrorException(e);
        }

        /* Returns ActionForward. */

        return mapping.findForward("UpdateHolidaysServiceAction");

    }
}
```