Capítulo 11

CÓDIGO DE LA APLICACIÓN

Clase IdMgr.java

```
 1 /* IdMgr.java
 2
 3  * Clase que gestiona los identificadores
 4  * Antonio Salado Manzorro
 5  *
 6  *
 7 */
 8
 9 package PID.ServiceClasses;
10
11 public interface IdMgr extends
12              IdentificationComponent{
13
14     /**
15      * Metodo register_new_ids() Genera nuevos identificadores
16      * y los asocia a los perfiles que se le pasan como parámetro
17      * @param profiles_to_register Secuencia de perfiles a registrar
18      * @return secuencia de identificadores que se han generado.
19      * @exception ProfilesExist,DuplicateProfiles,MultipleTraits,
20      * InvalidAdministrativeIds
21      */
22     public PersonIdSeq register_new_ids(ProfileSeq profiles_to_register)
23             throws ProfilesExist,DuplicateProfiles,MultipleTraits,
24             InvalidAdministrativeIds;
25
26
27     /**
28      * Metodo que genera nuevos identificadores y los asocia a los que
29      * se pasan como parámetro. A diferencia de register_new_ids, busca los
30      * perfiles en la Base de Datos, generando los nuevos identificadores que
31      * sean necesarios.
32      * @param profiles_to_register Secuencia de perfiles a registrar
33      * @return Secuencia con los nuevos PersonId
34      * @exception DuplicateProfiles,MultipleTraits,InvalidAdministrativeIds
35      */
```

```
36      public PersonIdSeq find_or_register_ids(ProfileSeq profiles_to_register)
37              throws DuplicateProfiles, MultipleTraits, InvalidAdministrativeIds;
38
39      /**
40       * Registra y asocia los identificadores indicados
41       * a los perfiles que se pasan.
42       * No está implementado de acuerdo a la política de generación de
43       * identificadores utilizada.
44       * @param profiles_to_register Secuencia de TaggedProfiles
45       * @exception NotImplemented
46       */
47      public void register_these_ids(TaggedProfileSeq profiles_to_register)
48              throws NotImplemented;
49
50      /**
51       * Método que genera identificadores en estado temporal.
52       * No comprueba si hay un posible perfil registrado que coincida con los
53       * que se pasan
54       * @param profiles_to_register Secuencia de perfiles a registrar
55       * @return Secuencia de PersonIds creada.
56       * @exception MultipleTraits
57       */
58      public PersonIdSeq create_temporary_ids(ProfileSeq profiles_to_register)
59              throws MultipleTraits;
60
61
62      /**
63       * Cambia el estado de los identificadores a permanente
64       * @param ids_to_modify Secuencia de identificadores
65       * @return Secuencia de PersonIds cuyo estado ha sido cambiado.
66       * @exception InvalidIds,DuplicateIds,RequiredTraits,MultipleTraits
67       */
68      public PersonIdSeq make_ids_permanent(PersonIdSeq ids_to_modify)
69              throws InvalidIds,DuplicateIds,RequiredTraits,MultipleTraits;
70
71      /**
72       * Fusiona los identificadores indicados en la MergeStruct
73       * @param ids_to_merge Secuencia de MergeStruct
74       * @return Secuencia de IdInfo de los identificadores fusionados.
75       * @exception InvalidIds,DuplicateIds
76       */
77      public IdInfoSeq merge_ids(MergeStructSeq ids_to_merge)
78              throws InvalidIds,DuplicateIds;
79
80      /**
81       * Deshace la fusión entre identificadores.
82       * @param ids_to_unmerge Secuencia de identificadores
83       * @return Secuencia de IdInfo
84       * @exception InvalidIds,DuplicateIds
85       */
86      public IdInfoSeq unmerge_ids(PersonIdSeq ids_to_unmerge)
87              throws InvalidIds,DuplicateIds;
88
89      /**
90       * Cambia el estado de los identificadores a Desactivado
91       * @param ids_to_deprecate Secuencia de identificadores a desactivar
92       * @return Secuencia de IdInfo
93       * @exception InvalidIds,DuplicateIds
94       */
95      public IdInfoSeq deprecate_ids(PersonIdSeq ids_to_deprecate)
96              throws InvalidIds,DuplicateIds;
97
98
99  }
```

Clase IdMgrImpl.java

```
 1 package PID.ServiceClasses;
 2 import PID.DataBase.*;
 3 import java.util.*;
 4 import java.sql.*;
 5 import PID.IdGest.*;
 6 import PID.BasicTypes.*;
 7
 8 public class IdMgrImpl implements
 9               IdMgr {
10     /**
11      * Manejador de la base de datos.
12      */
13     private DB_Manager DBM=null;
14     /**
15      * Para acceder a los métodos que interactuan
16      * con la base de datos.
17      */
18     private DB_Person BP=null;
19     // Para realizar comprobaciones sobre los Perfiles a registrar
20     private Check CH=null;
21     // Para acceder al registro de idnetificadores en la BBDD
22     private IdGestImpl IdGest=null;
23
24     /**
25      * Crea una nueva instancia de
26      * <B>IdMgrImpl</B>.
27      * @param DBMValue Valor que toma DBM.
28      */
29     public IdMgrImpl(DB_Manager DBMValue){
30         DBM=DBMValue;
31         BP=new DB_Person(DBMValue);
32         CH=new Check(DBMValue);
33         IdGest=new IdGestImpl(DBMValue);
34     }
35     /**
36      * Método register_new_ids
37      * Registra un conjunto de perfiles asignando a cada uno un
38      * identificador
39      * @param profiles_to_register Secuencia de Perfiles a registrar
40      * @return Devuelve la secuencia de PersonIds generada
41      * @throws ProfilesExist,DuplicateProfiles,MultipleTraits
42     **/
43     public PersonIdSeq register_new_ids(ProfileSeq profiles_to_register)
44         throws ProfilesExist,DuplicateProfiles,MultipleTraits,
45             InvalidAdministrativeIds{
46         boolean error=false;
47         PersonIdSeq ids=null;//Secuencia de PersonIds creados que se devuelve
48         IndexSeq perfiles=null;
49         //Comprobar Traits duplicados en cada perfil que se le pasa
50         CH.MultipleTraits(profiles_to_register);
51         //Encuentra perfiles repetidos
52         error=CH.MultipleProfiles(profiles_to_register);
53         //Comprueba si los nombres de los Traits son válidos
54         CH.traitNames(profiles_to_register);
55        //Mira si ya está en la Base de Datos y que no contiene Identificadores
56        // Externos que ya esten registrados
57         perfiles=CH.ProfileExists(profiles_to_register, (float)1.0);
58         if(perfiles.GetIndexSeq().length!=0){
59             throw new ProfilesExist(perfiles);
60         }
61         CH.administrativeIds(profiles_to_register);
62         //Generar ID
63         ids=generatePersonId(profiles_to_register.seq.length);
64         //Registrar Identificadores
65         for(int i=0;i<ids.GetPersonIdSeq().length;i++){
66             try{
```

```
67                  IdGest.registroId(ids.GetPersonIdSeq()[i].GetPersonId(),
68                      getLocalDomainOID(), "", getIVL()) ;
69              }catch(PID.IdGest.InvalidId e){
70                  System.out.println(e);
71              }catch(InvalidAAN e){
72                  System.out.println(e);
73              }catch(InvalidIVL e){
74                  System.out.println(e);
75              }
76        }
77       return ids;
78     }
79      /**
80      * Metodo que genera nuevos identificadores y los asocia a los que
81      * se pasan como parámetro. A diferencia de register_new_ids, busca los
82      * perfiles en la Base de Datos, generando los nuevos identificadores que
83      * sean necesarios.
84      * @param profiles_to_register Secuencia de perfiles a registrar
85      * @return Secuencia con los nuevos PersonId
86      * @exception DuplicateProfiles,MultipleTraits,InvalidAdministrativeIds
87      */
88     public PersonIdSeq find_or_register_ids(ProfileSeq profiles_to_register)
89         throws DuplicateProfiles,MultipleTraits,InvalidAdministrativeIds{
90         PersonIdSeq ids=null; //Secuencia de PersonIds creados que se devuelve
91         ResultSet Answer=null;
92         String SQL="SELECT Valor FROM InfoPID WHERE Nombre="Confidence"";
93         float peso=0;
94         IndexSeq perfiles=null;
95         ProfileSeq new_profiles=new ProfileSeq();
96         Profile aux[]=null;
97         int count =0;
98         boolean error=false;
99         //Comprobar Traits duplicados en cada perfil que se le pasa
100        CH.MultipleTraits(profiles_to_register);
101        //Encuentra perfiles repetidos
102        error=CH.MultipleProfiles(profiles_to_register);
103        //Comprueba si los nombres de los Traits son válidos
104        CH.traitNames(profiles_to_register);
105
106        //Obtenemos el grado de confianza de la búsqueda del perfil
107        Answer=DBM.Ask(SQL);
108        try{
109            if(Answer.next()){
110                peso=Answer.getFloat(1);
111            }
112       }catch (SQLException e){
113            System.out.print(e);
114       }
115       //Mira si ya está en la Base de Datos y que no contiene Identificadores
116       // Externos que ya esten registrados
117       perfiles=CH.ProfileExists(profiles_to_register, peso);
118       CH.administrativeIds(profiles_to_register);
119
120       //si hace falta generar nuevos Identificadores
121       if(perfiles.GetIndexSeq().length!=profiles_to_register.seq.length){
122        //Generar ID para los perfiles que no están
123        //Copiamos a una secuencia auxiliar los perfiles que hay que registrar
124            aux=new Profile[profiles_to_register.seq.length-
125                    perfiles.GetIndexSeq().length];
126            for(int i=0;i<profiles_to_register.seq.length;i++)
127                for(int j=0;j<perfiles.GetIndexSeq().length;j++)
128                    if(perfiles.GetIndexSeq()[j].getIndex()!=i){
129                        aux[count]=profiles_to_register.seq[i];
130                        count++;
131                    }
132          //new_profile es una secuencia que contiene los Profiles que no se
133          //corresponden a ninguno en la BBDD por lo que es necesario generar
134          // nuevos Ids y asignárselos
```

```
135             new_profiles.seq=aux;
136             ids=generatePersonId(new_profiles.seq.length);
137             //Registrar Identificadores
138             for(int i=0;i<ids.GetPersonIdSeq().length;i++){
139                 try{
140                 IdGest.registroId(ids.GetPersonIdSeq()[i].GetPersonId(),
141                         getLocalDomainOID(), "",  getIVL()) ;
142                 }catch(PID.IdGest.InvalidId e){
143                     System.out.println(e);
144                 }catch(InvalidAAN e){
145                     System.out.println(e);
146                 }catch(InvalidIVL e){
147                     System.out.println(e);
148                 }
149             }
150         }
151         return ids;
152     }
153 
154     /**
155      * Registra y asocia los identificadores indicados
156      * a los perfiles que se pasan.
157      * No está implementado de acuerdo a la política de generación de
158      * identificadores utilizada.
159      * @param profiles_to_register Secuencia de TaggedProfiles
160      * @exception NotImplemented
161      */
162     public void register_these_ids(TaggedProfileSeq profiles_to_register)
163             throws NotImplemented{
164 
165         throw new NotImplemented();
166     }
167      /**
168      * Método que genera identificadores en estado temporal.
169      * No comprueba si hay un posible perfil registrado que coincida con los
170      * que se pasan
171      * @param profiles_to_register Secuencia de perfiles a registrar
172      * @return Secuencia de PersonIds creada.
173      * @exception MultipleTraits
174      */
175     public PersonIdSeq create_temporary_ids(ProfileSeq profiles_to_register)
176             throws MultipleTraits{
177         //Secuencia de PersonIds creados que se devuelve
178         PersonIdSeq ids=null;
179         //Comprobar Traits duplicados en cada perfil que se le pasa
180         CH.MultipleTraits(profiles_to_register);
181         //Comprueba si los nombres de los Traits son válidos
182         CH.traitNames(profiles_to_register);
183         //Generar ID
184         ids=generatePersonId(profiles_to_register.seq.length);
185         //Registrar Identificadores
186         //Obtenemos los valores adecuados para registrarlos
187         for(int i=0;i<ids.GetPersonIdSeq().length;i++){
188             try{
189                 IdGest.registroId(ids.GetPersonIdSeq()[i].GetPersonId(),
190                     getLocalDomainOID(), "",  getIVL()) ;
191             }catch(PID.IdGest.InvalidId e){
192                 System.out.println(e);
193             }catch(InvalidAAN e){
194                 System.out.println(e);
195             }catch(InvalidIVL e){
196                 System.out.println(e);
197             }
198         }
199         return ids;
200     }
201 
202 
```

```
203
204     /**
205      * Cambia el estado de los identificadores a permanente
206      * @param ids_to_modify Secuencia de identificadores
207      * @return Secuencia de PersonIds cuyo estado ha sido cambiado.
208      * @exception InvalidIds,DuplicateIds,RequiredTraits,MultipleTraits
209      */
210     public PersonIdSeq make_ids_permanent(PersonIdSeq ids_to_modify)
211             throws InvalidIds,DuplicateIds,RequiredTraits,MultipleTraits{
212         IdInfoSeq info=null;
213         PersonId aux[]=new PersonId[ids_to_modify.GetPersonIdSeq().length];
214         PersonId error[]=null;
215         PersonIdSeq error_ids=new PersonIdSeq();
216         String SQL=null;
217         int count=0;
218         //Comprobamos si los PersonId que se pasan están registrados
219         CH.IdExist(ids_to_modify);
220         //Comprobamos que no hay Ids repetidos en la secuencia.
221         CH.MultipleIds(ids_to_modify);
222         //Comprobamos que están presentes los Traits obligatorios.
223         CH.Mandatory(ids_to_modify);
224         //Vemos que todos los PersonId están en estado Temporary
225         info=CH.getIdInfo(ids_to_modify);
226         for(int i=0;i<info.GetInfoIdSeq().length;i++){
227             if(info.GetInfoIdSeq()[i].GetState().toString()!="TEMPORARY"){
228                 aux[i]=info.GetInfoIdSeq()[i].GetId();
229                 count++;
230             }
231             else
232                 aux[i]=null;
233         }
234         if(count!=0){
235             error=new PersonId[count];
236             count=0;
237             for(int i=0;i<aux.length;i++){
238                if(aux[i]!=null){
239                     error[count]=aux[i];
240                     count++;
241                 }
242             }
243             error_ids.SetPersonIdSeq(error);
244             info=CH.getIdInfo(error_ids);
245             throw new InvalidIds(info);
246         }
247         //Si no hay error cambiamos los estados
248         for(int i=0;i<ids_to_modify.GetPersonIdSeq().length;i++){
249             SQL="UPDATE PersonIdentifiers SET state="PERMANENT" WHERE " +
250                   "extension=""+ids_to_modify.GetPersonIdSeq()[i].GetPersonId()+
251                   "" ";
252             DBM.change(SQL);
253         }
254         //Devolvemos la secuencia de PersonIds
255         return ids_to_modify;
256     }
257     /**
258      * Fusiona los identificadores indicados en la MergeStruct
259      * @param ids_to_merge Secuencia de MergeStruct
260      * @return Secuencia de IdInfo de los identificadores fusionados.
261      * @exception InvalidIds,DuplicateIds
262      */
263     public IdInfoSeq merge_ids(MergeStructSeq ids_to_merge)
264             throws InvalidIds,DuplicateIds{
265         boolean active=false;
266         String SQL=null;
267         PersonIdSeq idseq=new PersonIdSeq();
```

```
268         PersonId ids[]=null;
269         //Comprobamos si los identificadores existen
270         CH.IdExist(ids_to_merge);
271         //Comprobamos que los dos
272         CH.different_ids(ids_to_merge);
273         CH.isActiveState(ids_to_merge);
274         //UPDATE tabla identificadores, estado=DEACT,preferred_id
275         for(int i=0;i<ids_to_merge.seq.length;i++){
276             SQL="UPDATE PersonIdentifiers SET state="DEACTIVATED"," +
277             "preferredId=""+ids_to_merge.seq[i].Getpreferred_Id().GetPersonId()+
278              "" WHERE extension=""+ids_to_merge.seq[i].GetId().GetPersonId()+
279                    """;
280             int a=DBM.change(SQL);
281         }
282         //Creo un IDInfoSeq
283         ids=new PersonId[ids_to_merge.seq.length];
284         for(int i=0;i<ids_to_merge.seq.length;i++){
285             ids[i]=ids_to_merge.seq[i].GetId();
286         }
287         idseq.SetPersonIdSeq(ids);
288         IdInfoSeq infoseq=CH.getIdInfo(idseq);
289         //Devolver el IDInfoSeq
290         return infoseq;
291     }
292     /**
293      * Deshace la fusión entre identificadores.
294      * @param ids_to_unmerge Secuencia de identificadores
295      * @return Secuencia de IdInfo
296      * @exception InvalidIds,DuplicateIds
297      */
298     public IdInfoSeq unmerge_ids(PersonIdSeq ids_to_unmerge)
299             throws InvalidIds,DuplicateIds{
300         IdStateSeq estado=null;
301         PersonIdSeq errorids=null;
302         IdInfoSeq idinfo=null;
303         String SQL=null;
304         //Compruebo que todos los identificadores existen
305         CH.IdExist(ids_to_unmerge);
306         //Compruebo que en la secuencia no hay ninguno repetido
307         CH.MultipleIds(ids_to_unmerge);
308         //Compruebo que estan en estado DEACTIVATED
309         errorids=CH.not_in_State(ids_to_unmerge, "DEACTIVATED");
310         //si se ha devuelto alguno es que no se encuentra desactivado Excepción
311         if(errorids!=null){
312             idinfo=CH.getIdInfo(errorids);
313             throw new InvalidIds(idinfo);
314         }
315         // si todo correcto se actualiza la BBDD.
316         else
317             //UPDATE tabla identificadores, estado=DEACT,preferred_id
318             for(int i=0;i<ids_to_unmerge.GetPersonIdSeq().length;i++){
319                SQL="UPDATE PersonIdentifiers SET state="TEMPORARY"," +
320                        "preferredId="" WHERE extension=""+
321                        ids_to_unmerge.GetPersonIdSeq()[i].GetPersonId()+"" ";
322                int j=DBM.change(SQL);
323             }
324             idinfo=CH.getIdInfo(ids_to_unmerge);
325         return idinfo;
326     }
327
328
```

```
329      /**
330       * Cambia el estado de los identificadores a Desactivado
331       * @param ids_to_deprecate Secuencia de identificadores a desactivar
332       * @return Secuencia de IdInfo
333       * @exception InvalidIds,DuplicateIds
334       */
335      public IdInfoSeq deprecate_ids(PersonIdSeq ids_to_deprecate)
336              throws InvalidIds,DuplicateIds{
337          String SQL=null;
338          IdInfoSeq idinfo=null;
339          //Compruebo que los Ids existen
340          CH.IdExist(ids_to_deprecate);
341          //Compruebo que están en estado activo
342          CH.isActiveState(ids_to_deprecate);
343          //Si no hay error cambio el estado de los identificadores
344          for(int i=0;i<ids_to_deprecate.GetPersonIdSeq().length;i++){
345              SQL="UPDATE PersonIdentifiers SET state="DEACTIVATED" WHERE " +
346                 "extension=""+ids_to_deprecate.GetPersonIdSeq()[i].GetPersonId()+
347                      """;
348              DBM.change(SQL);
349          }
350          idinfo=CH.getIdInfo(ids_to_deprecate);
351          return idinfo;
352      }
353
354
355      /**
356       * Genera una secuencia de PersonIds con tantos elementos como el
357       * número indicado como parámetro
358       * @param n Número de PersonIds a crear.
359       * @return Secuencia con los PersonIds generados
360       */
361      private PersonIdSeq generatePersonId(int n){
362          PersonIdSeq ids=new PersonIdSeq();
363          PersonId aux[]=new PersonId[n];
364          ResultSet Answer=null;
365          int id=0;
366          String aux1=null;
367          String SQL="SELECT valor FROM InfoPID WHERE" +
368                  " Nombre="LastPersonID"";
369          Answer=DBM.Ask(SQL);
370          try{
371              if(Answer.next()){
372                  id=Answer.getInt(1);
373              }
374         }catch (SQLException e){
375              System.out.print(e);
376         }
377         for(int i=0;i<n;i++){
378              id++;
379              aux1=""+id;
380              aux[i]=new PersonId(aux1);
381         }
382         SQL="UPDATE InfoPID SET valor=""+aux1+"" WHERE Nombre="LastPersonID"";
383         DBM.change(SQL);
384         ids.SetPersonIdSeq(aux);
385         return ids;
386      }
387
388      /**
389       * Función que devuelve el OID del dominio local
390       * @return OID del Dominio al que pertenece
391       */
392      private OID getLocalDomainOID(){
393          OID root=null;
394          ResultSet Answer=null;
395          String SQL="SELECT valor FROM InfoPID WHERE Nombre="LocalDomainOID"";
396          Answer=DBM.Ask(SQL);
```

```
397         try{
398             if(Answer.next()){
399                 root=new OID(Answer.getString(1));
400             }
401        }catch (SQLException e){
402             System.out.print(e);
403        }
404         return root;
405     }
406 
407     /**
408      * Genera un IVL a partir de la fecha actual y el intervalo de validez
409      * indicado en la tabla InfoPid.
410      * @return Devuelve el IVL creado
411      */
412     private IVL_TS getIVL(){
413         IVL_TS validTime=null;
414         Timestamp now=null;
415         Timestamp later=null;
416         String time="";
417         int months=0;
418         Calendar a=Calendar.getInstance();
419         //Obtenemos el instante actual
420         time=time+a.get(Calendar.YEAR)+"-"+(a.get(Calendar.MONTH)+1)+"-"+
421                a.get(Calendar.DAY_OF_MONTH)+" "+a.get(Calendar.HOUR_OF_DAY)+":"+
422                 a.get(Calendar.MINUTE)+":"+a.get(Calendar.SECOND);
423 
424         now=Timestamp.valueOf(time);
425         ResultSet Answer=null;
426         String SQL="SELECT valor FROM InfoPID WHERE Nombre="DefaultIVL"";
427         Answer=DBM.Ask(SQL);
428         try{
429             if(Answer.next()){
430                 months=Answer.getInt(1);
431             }
432        }catch (SQLException e){
433             System.out.print(e);
434        }
435         a.add(Calendar.MONTH,months);
436         time="";
437         time=time+a.get(Calendar.YEAR)+"-"+(a.get(Calendar.MONTH)+1)+"-"+
438               a.get(Calendar.DAY_OF_MONTH)+" "+a.get(Calendar.HOUR_OF_DAY)+":"+
439               a.get(Calendar.MINUTE)+":"+a.get(Calendar.SECOND);
440         later=Timestamp.valueOf(time);
441         validTime=new IVL_TS(now,later,false,false,"");
442 
443         return validTime;
444 
445     }
446 
447 
448 
449 }
450 
```

Clase Check.java

```
 1 /*
 2  * Clase Check.java
 3  * Contiene los métodos destinados a
 4  * hacer las comprobaciones en las operaciones de la
 5  * interfaz IdMgr.
 6  */
 7
 8 package PID.ServiceClasses;
 9 import PID.DataBase.*;
10 import java.util.*;
11 import java.sql.*;
12 import PID.IdGest.*;
13 import PID.BasicTypes.*;
14
15 public class Check{
16     private DB_Manager DBM=null;
17     private Comprueba CH=null;
18     /**
19      * Para acceder a los métodos que interactuan
20      * con la base de datos.
21      */
22     private DB_Person BP=null;
23     public Check(DB_Manager DB){
24         DBM=DB;
25         BP=new DB_Person(DB);
26         CH=new Comprueba(DB);
27     }
28
29
30     /**
31      * Comprueba que en la secuencia de perfiles que se pasa no hay
32      * ninguno repetido
33      * @param profiles Secuencia de perfiles a comprobar
34      * @exception DuplicateProfiles
35      * @return False si no hay perfiles repetidos en la secuencia.
36      */
37   public boolean MultipleProfiles(ProfileSeq profiles) throws DuplicateProfiles{
38         boolean error=false;
39         String duplicate="";
40         int count=0;
41         Vector a=new Vector();
42         for(int i=0; i<profiles.seq.length; i++){
43             for(int j=i+1; j<profiles.seq.length; j++){
44              //si los perfiles tienen el mismo número de Traits los comparamos
45               if(profiles.seq[i].seq.length==profiles.seq[j].seq.length){
46                 count=0;
47                 for(int k=0;k<profiles.seq[i].seq.length;k++){
48                   for(int l=0;l<profiles.seq[j].seq.length;l++) {
49                   if(profiles.seq[i].seq[k].value==profiles.seq[j].seq[l].value)
50                      if(profiles.seq[i].seq[k].name==profiles.seq[j].seq[l].name
51                        &&(profiles.seq[i].seq[k].value.toString().compareTo
52                        (profiles.seq[j].seq[l].value.toString())==0)){
53                             //Contamos el número de Traits que coinciden
54                             count++;
55                       }
56                   }
57                 }
58               }
59                 //Si coinciden todos los Traits añadimos el indice del Profile
60                 //solo si no estaba ya.
61                 if (count==profiles.seq[i].seq.length){
62                     if(duplicate.indexOf(" "+i)==-1) {
63                         duplicate=duplicate+" "+i;
64                         a.add(new Index(i));
65                     }
```

```
66                      if(duplicate.indexOf(" "+j)==-1){
67                          duplicate=duplicate+" "+j;
68                          a.add(new Index(j));
69                      }
70                  }
71              }
72          }
73          // Si hay dos Perfiles iguales lanzamos la excepcion correspondiente
74          if(a.size()!=0){
75              error=true;
76              throw new DuplicateProfiles(getErrorIndexSeq(a));
77          }
78          return error;
79  }
80
81      /**
82       * Comprueba que en la secuencia de perfiles que se pasa no hay ninguno
83       * que contenga Trais repetidos
84       * @param profiles Secuencia de perfiles a comprobar
85       * @exception MultipleTraits
86       * @return False si no hay traits repetidos.
87       */
88      public boolean MultipleTraits(ProfileSeq profiles) throws MultipleTraits{
89          int aux=0;
90          boolean error=false;
91          TraitNameSeq errorTraits=new TraitNameSeq();
92          //Sec que contiene los nombres de los Traits repetidos
93          MultipleFailure[] mfsdef=null;
94          //Secuencia que se devuelve;
95          Vector a=new Vector();
96          Vector b=new Vector();
97          String c[]=null;
98          II id1=null;
99          II id2=null;
100         for(int i=0;i<profiles.seq.length;i++){
101             for(int j=0;j<profiles.seq[i].seq.length;j++){
102                 for(int k=j+1;k<profiles.seq[i].seq.length;k++){
103                    if(profiles.seq[i].seq[j].name==profiles.seq[i].seq[k].name){
104                    //Si el trait que se repitees un II  o un entityNamePart,
105                    //vemos si el valor es el mismo.
106                    // si no se repite el valor no lanzamos ninguna excepcion.
107                         if(profiles.seq[i].seq[j].name=="II"){
108                             id1=(II)profiles.seq[i].seq[j].value;
109                             id2=(II)profiles.seq[i].seq[k].value;
110                             if( (id1.getExtension()==id2.getExtension()) &&
111                              (id1.getRoot().getOID()==id2.getRoot().getOID()) ){
112                                 a.add(profiles.seq[i].seq[k].name);
113                             }
114                         }
115                         else if(profiles.seq[i].seq[j].name=="entityNamePart"){
116                             if(profiles.seq[i].seq[j].value==
117                                profiles.seq[i].seq[k].value){
118                                  a.add(profiles.seq[i].seq[k].name);
119                             }
120                         }
121                         else{
122                             a.add(profiles.seq[i].seq[k].name);
123                         }
124                     }
125                 }
126           }
127           if(a.size()!=0){
128                 c=new String[a.size()];
129                 if(a.size()!=0){
130                     aux=a.size();
131                     for(int l=0;l<aux;l++ ){
132                         c[l]=a.remove(0).toString();
133                     }
```

```
134                     errorTraits=new TraitNameSeq(c);
135                     b.add((new MultipleFailure(new Index(i),
136                         new ExceptionReason("DUPLICATE_TRAITS"), errorTraits)));
137                 }
138             }
139         }
140         if(b.size()!=0){
141             mfsdef=new MultipleFailure[b.size()];
142             aux=0;
143             while(!b.isEmpty()){
144                 mfsdef[aux]=(MultipleFailure)b.remove(0);
145                 aux++;
146             }
147             error=true;
148             throw new MultipleTraits(new MultipleFailureSeq(mfsdef));
149         }
150         return error;
151     }
152 
153 
154     /**
155      * Comprueba que en la secuencia de identificadores que se pasa
156      * no hay ninguno repetido
157      * @param ids Secuencia de identificadores a comprobar
158      * @exception DuplicateIds
159      * @return False si no hay identificadores repetidos en la secuencia.
160      */
161     public boolean MultipleIds (PersonIdSeq ids) throws DuplicateIds{
162         String aux="";
163         Vector a=new Vector();
164         boolean error=false;
165         for (int i=0;i<ids.GetPersonIdSeq().length;i++){
166           for (int j=i+1;j<ids.GetPersonIdSeq().length;j++){
167              if(Integer.parseInt(ids.GetPersonIdSeq()[i].GetPersonId())==
168               Integer.parseInt(ids.GetPersonIdSeq()[j].GetPersonId())){
169                 if(aux.indexOf(""+ids.GetPersonIdSeq()[j].GetPersonId()+"")==-1)
170                 {
171                    aux=aux+ids.GetPersonIdSeq()[j].GetPersonId();
172                    a.add(new PersonId(ids.GetPersonIdSeq()[j].GetPersonId()));
173                 }
174               }
175            }
176         }
177         if(a.size()!=0){
178             error=true;
179             throw new DuplicateIds(getErrorPersonIdSeq(a));
180         }
181         return error;
182     }
183     /**
184      * Comprueba que cada MergeStruct de la secuencia no contiene 2
185      * identificadores iguales y que no hay MergeStructs repetidas
186      * en la secuencia
187      * @param ids Secuencia de MergeStruct
188      * @exception InvalidId
189      * @return False si no hay identificadores repetidos en la mergeStruct.
190      */
191     public boolean different_ids (MergeStructSeq ids) throws DuplicateIds{
192         String aux="";
193         Vector a=new Vector();
194         boolean error=false;
195         //Comprobar que los 2 Ids de cada estructura no sean iguales
196         for(int i=0;i<ids.seq.length;i++){
197             if(Integer.parseInt(ids.seq[i].GetId().GetPersonId())==
198               Integer.parseInt(ids.seq[i].Getpreferred_Id().GetPersonId()))
199             {
200                 a.add(ids.seq[i].GetId());
201                 System.out.println("1°");
```

```
202             }
203         }
204         if(a.size()!=0){
205             error=true;
206             throw new DuplicateIds(getErrorPersonIdSeq(a));
207         }
208         //Comprobar que no se pasan 2 veces el mismo Id para ser fusionado
209         for (int i=0;i<ids.seq.length;i++){
210             for (int j=i+1;j<ids.seq.length;j++){
211                 if(Integer.parseInt(ids.seq[i].GetId().GetPersonId())==
212                         Integer.parseInt(ids.seq[j].GetId().GetPersonId()))
213                 {
214                     if(aux.indexOf(""+ids.seq[i].GetId()+"")==-1)
215                     {
216                         aux=aux+" "+ids.seq[i].GetId();
217                         a.add(ids.seq[i].GetId());
218                         System.out.println("2°");
219                     }
220                 }
221             }
222         }
223         if(a.size()!=0){
224             error=true;
225             throw new DuplicateIds(getErrorPersonIdSeq(a));
226         }
227         return error;
228     }
229
230
231
232     /**
233      * Comprueba que los perfiles que se pasan no existen ya en la Base de Datos
234      * y que si contienen un II que ya ha sido registrado
235      * @param profiles Secuencia de perfiles a comprobar
236      * @exception ProfilesExist,IIExist
237      */
238     public IndexSeq ProfileExists(ProfileSeq profiles,float peso) {
239         Vector q=new Vector();
240         Vector a=new Vector();
241         TraitSelector[] profile_selector;
242         for(int k=0;k<profiles.seq.length;k++){
243             profile_selector=new TraitSelector[profiles.seq[k].seq.length];
244             //Creamos un TraitSelector[] para pasarlo a CandidatePersonId
245           for(int i=0;i<profiles.seq[k].seq.length;i++)
246             {
247             profile_selector[i]=new TraitSelector(profiles.seq[k].seq[i], peso);
248             }
249           try{
250             //Vemos si el Perfil corresponde a algun identificador
251                 q=BP.CandidatePersonId(profile_selector,peso);
252                 if(q!=null){
253                     if(q.size()!=0)
254                     {
255                         a.add(new Index(k));
256                     }
257                 }
258             }
259             // Captura de excepcion, que nunca se produce porque controlamos
260             //los parámetros.
261             catch(InvalidWeight e){
262                }
263             catch(UnknownTraits e){
264                }
265           }
266         return getErrorIndexSeq(a);
267      }
268
```

```
269     /**
270      * Comprueba que los identificadores que forman parte de cada MergeStruct
271      * que se pasa existe en la BBDD. Si no lanza la excepcion InvalidId
272      * indicando el identificador y el indice del MergeStruct correspondiente
273      * @param ids Secuencia de MergeStruct a comprobar
274      * @exception InvalidIds
275      * @return False si el identificador de la secuencia no existe en la BBDD
276      */
277     public boolean IdExist(MergeStructSeq ids) throws InvalidIds{
278         PersonId id=null;
279         String aux="";
280         String aux1="";
281         ResultSet Answer=null;
282         String SQL=null;
283         Vector a=new Vector();
284         boolean error=false;
285         for (int i=0;i<ids.seq.length;i++)
286         {
287             id=ids.seq[i].GetId();
288             // Comprueba en la BBDD si ya hay un Identificador igual a este.
289             SQL="SELECT extension FROM PersonIdentifiers WHERE extension=""+
290                     id.GetPersonId()+""";
291             Answer=DBM.Ask(SQL);
292             try{
293                 if(!Answer.next()){
294                     if(aux.indexOf(""+ids.seq[i].GetId()+"")==-1){
295                         aux=aux+" "+ids.seq[i].GetId();
296                         a.add(ids.seq[i].GetId());
297                     }
298                 }
299             }catch (SQLException e){
300                 System.out.print(e);
301             }
302 
303             // Comprueba que existe un identificador (preferredId)  en la BBDD
304             id=ids.seq[i].Getpreferred_Id();
305             SQL="SELECT extension FROM PersonIdentifiers WHERE extension=""+
306                     id.GetPersonId()+""";
307             Answer=DBM.Ask(SQL);
308             try{
309                 if(!Answer.next()){
310                     if(aux.indexOf(""+ids.seq[i].Getpreferred_Id()+"")==-1){
311                         aux=aux+" "+ids.seq[i].Getpreferred_Id();
312                         a.add(ids.seq[i].Getpreferred_Id());
313                     }
314                 }
315                 }catch (SQLException e){
316                     System.out.print(e);
317                 }
318           }
319             //Si alguno no está en la BBDD
320         if(a.size()!=0)
321         {
322             error=true;
323             throw new InvalidIds(getIdInfo(getErrorPersonIdSeq(a)));
324         }
325         return error;
326     }
327 
328 
329 
330 
```

```
331     /**
332      * Comprueba que los identificadores que forman parte de cada PersonId
333      * que se pasa existe en la BBDD. Si no lanza la excepcion InvalidId
334      * indicando el identificador
335      * @param ids Secuencia de PersonId a comprobar
336      * @exception InvalidIds
337      * @return False si los identificadores de la secuencia no están registrados
338      * en la BBDD.
339      */
340     public boolean IdExist(PersonIdSeq ids) throws InvalidIds{
341         String id;
342         ResultSet Answer=null;
343         String SQL=null;
344         String aux="";
345         String aux1="";
346         Vector a=new Vector();
347         boolean error=false;
348         for (int i=0;i<ids.GetPersonIdSeq().length;i++)
349         {
350          id=ids.GetPersonIdSeq()[i].GetPersonId();
351          // Comprueba en la BBDD si ya hay un Identificador igual a este.
352          SQL="SELECT extension FROM PersonIdentifiers WHERE extension=""+id+""";
353          Answer=DBM.Ask(SQL);
354          try
355          {
356               if(!Answer.next())
357               {
358                     if(aux.indexOf(""+i+"")==-1)
359                     {
360                         aux=aux+" "+i;
361                         a.add(new PersonId(id));
362                     }
363               }
364           }catch (SQLException e){
365                 System.out.print(e);
366             }
367         }
368         if(a.size()!=0){
369             error=true;
370             throw new InvalidIds(getIdInfo(getErrorPersonIdSeq(a)));
371         }
372         return error;
373     }
374
375     /**
376      * Devuelve el estado en que se encuentran los PersonId,o los preferredIds
377      * que forman parte de cada  MergeStruct que se pasa.
378      * @param ids Identificador del que se desea conocer el estado
379      * @return Secuencia de IdState con los estados de los Ids
380      */
381     public IdStateSeq ask_state(MergeStructSeq ids,String desiredIds){
382         IdStateSeq state=null;
383         PersonId[] idx=new PersonId[ids.seq.length];
384         if(desiredIds=="PersonId"){
385             for(int i=0;i<ids.seq.length;i++)
386                 idx[i]=ids.seq[i].GetId();
387         }
388         if(desiredIds=="preferredId"){
389             for(int i=0;i<ids.seq.length;i++)
390                 idx[i]=ids.seq[i].Getpreferred_Id();
391         }
392         PersonIdSeq idseq=new PersonIdSeq(idx);
393         state=ask_state(idseq);
394         return state;
395     }
```

```
396     /**
397      * Devuelve el estado en que se encuentran los PersonId que se pasan.
398      *@param ids Secuencia de Identificadores del que se desea conocer el estado
399      * @return Secuencia de IdState
400      */
401     public IdStateSeq ask_state(PersonIdSeq ids){
402         IdStateSeq state=null;
403         ResultSet Answer=null;
404         String SQL=null;
405         String id=null;
406         IdState[] stateaux=new IdState[ids.GetPersonIdSeq().length];
407         for(int i=0;i<ids.GetPersonIdSeq().length;i++)
408         {
409             SQL="SELECT state FROM PersonIdentifiers WHERE extension=""+
410                     ids.GetPersonIdSeq()[i].GetPersonId()+""";
411             Answer=DBM.Ask(SQL);
412             try{
413                 if(Answer.next()){
414                 id=Answer.getString(1);
415 
416                      if(id.compareTo("DEACTIVATED")==0)
417                         id="DEACTIVATED";
418                      if(id.compareTo("INVALID")==0)
419                         id="INVALID";
420                      if(id.compareTo("PERMANENT")==0)
421                         id="PERMANENT";
422                      if(id.compareTo("TEMPORARY")==0)
423                         id="TEMPORARY";
424                      if(id.compareTo("UNKNOWN")==0)
425                         id="UNKNOWN";
426                  stateaux[i]=new IdState(id);
427                 }
428                 else id="INVALID";
429                 stateaux[i]=new IdState(id);
430             }catch (SQLException e){
431                 System.out.print(e);
432             }
433         }
434         state=new IdStateSeq(stateaux);
435         return state;
436     }
437 
438     /**
439      * Comprueba si los Identificadores que se pasan están
440      * en un estado Activo (Temporary o Permanent)
441      * @param ids Secuencia de Identificadores a comprobar
442      * @exception InvalidIds
443      * @return True si el estado es Activo y False si no lo es.
444      */
445     public boolean isActiveState(PersonIdSeq ids) throws InvalidIds{
446         IdStateSeq states=null;
447         String aux="";
448         Vector a=new Vector();
449         states=ask_state(ids);
450         boolean error=false;
451         for(int i=0;i<states.GetIdStateSeq().length;i++){
452           if((states.GetIdStateSeq()[i].GetIdState()!="PERMANENT") ){
453              if(states.GetIdStateSeq()[i].GetIdState()!="TEMPORARY"){
454                 if(aux.indexOf(""+ids.GetPersonIdSeq()[i].GetPersonId()+"")==-1)
455                 {
456                     aux=aux+" "+ids.GetPersonIdSeq()[i].GetPersonId();
457                     a.add(new PersonId(ids.GetPersonIdSeq()[i].GetPersonId()));
458                 }
459              }
460           }
461         }
```

```
462         if(a.size()!=0){
463             error=true;
464             throw new InvalidIds(getIdInfo(getErrorPersonIdSeq(a)));
465         }
466         return error;
467     }
468 
469     /**
470      * Comprueba si los Identificadores que forman el MergeStructSeq
471      * que se pasan están en un estado Activo (Temporary o Permanent)
472      * @param ids Secuencia de Identificadores a comprobar
473      * @throws InvalidIds
474      * @return True si el estado es Activo y False si no lo es.
475      */
476     public boolean isActiveState(MergeStructSeq ids) throws InvalidIds{
477         IdStateSeq states=null;
478         String aux="";
479         Vector a =new Vector();
480         states=ask_state(ids, "PersonId");
481         boolean error=false;
482         for(int i=0;i<states.GetIdStateSeq().length;i++){
483             if((states.GetIdStateSeq()[i].GetIdState()!="PERMANENT") ){
484                 if(states.GetIdStateSeq()[i].GetIdState()!="TEMPORARY"){
485                    if(aux.indexOf(""+ids.seq[i].GetId()+"")==-1){
486                         aux=aux+" "+ids.seq[i].GetId();
487                         a.add(ids.seq[i].GetId());
488                    }
489                 }
490             }
491         }
492         states=ask_state(ids, "preferredId");
493         for(int i=0;i<states.GetIdStateSeq().length;i++){
494             if((states.GetIdStateSeq()[i].GetIdState()!="PERMANENT") ){
495                 if(states.GetIdStateSeq()[i].GetIdState()!="TEMPORARY"){
496                    if(aux.indexOf(""+ids.seq[i].Getpreferred_Id()+"")==-1){
497                         aux=aux+" "+ids.seq[i].Getpreferred_Id();
498                         a.add(ids.seq[i].Getpreferred_Id());
499                    }
500                 }
501             }
502         }
503         if(a.size()!=0){
504             error=true;
505             throw new InvalidIds(getIdInfo(getErrorPersonIdSeq(a)));
506         }
507         return error;
508     }
509 
510     /**
511      * Comprueba si los Identificadores de la secuencia tienen
512      * el campo preferred_id distinto a "" sino lanza InvalidId
513      * @param ids Secuencia de Identificadores a comprobar
514      * @exception InvalidId
515      * @return False si el campo preferredId es distinto a ""
516      */
517     public boolean PreferredId(PersonIdSeq ids) throws InvalidIds{
518         ResultSet Answer=null;
519         String SQL=null;
520         String id=null;
521         Vector a=new Vector();
522         boolean error=false;
523         //Para cada PersonId de la secuencia consultamos la BBDD
524         for(int i=0;i<ids.GetPersonIdSeq().length;i++){
525             SQL="SELECT preferredId FROM PersonIdentifiers WHERE extension=""+
526                     ids.GetPersonIdSeq()[i].GetPersonId()+""";
527             Answer=DBM.Ask(SQL);
```

```
528             try{
529                 while(Answer.next()){
530                     id=Answer.getString(1);
531                     if(id.compareTo("")==0){
532                     //Vemos si el preferredId es distinto a ""
533                         a.add(ids.GetPersonIdSeq()[i]);
534                     }
535                 }
536             }catch (SQLException e){
537                 System.out.print(e);
538             }
539         }
540         if(a.size()!=0){
541             error=true;
542             throw new InvalidIds(getIdInfo(getErrorPersonIdSeq(a)));
543         }
544         return error;
545     }
546 
547     /**
548      * Comprueba si en los perfiles que se pasan hay identificadores
549      * administrativos que ya estan registrados. Y que el II tiene rellenos los
550      * campos extension y root
551      * @param profiles
552      * @return False si no hay error.
553      */
554     public boolean administrativeIds(ProfileSeq profiles) throws
555             InvalidAdministrativeIds{
556         II id=null;
557         String extension=null;
558         OID root=null;
559         String AAN=null;
560         boolean existe=false;
561         Vector a=new Vector();
562         boolean error=false;
563         for(int i=0;i<profiles.seq.length;i++){
564             for(int j=0;j<profiles.seq[i].seq.length;j++){
565                 if(profiles.seq[i].seq[j].name=="II"){
566                     id=(II)profiles.seq[i].seq[j].value;
567                     root=id.getRoot();
568                     extension=id.getExtension();
569                     AAN=id.getAAN();
570                     // Compruebo si en el perfil están, o bien la extension,
571                     // o bien el root por separado
572                     if((extension!=null&&root==null&&AAN==null)||
573                             (extension==null&&root!=null)
574                             ||extension==null&&AAN!=null)
575                     {
576                         a.add(new Index(i));
577                     }
578                     if(extension!=null&&root!=null){
579                         try{
580                             existe=CH.IdExist(extension,root);
581                             //si no existe la Entidad asignadora también
582                             //añadimos el perfil a la lista de erróneos
583                         }catch(InvalidAAN e){
584                             a.add(new Index(i));
585                         }
586                         if(existe==true){
587                             //Si ya esta el Id se añade el indice del perfil
588                             // a la lista.
589                             a.add(new Index(i));
590                         }
591                     }
592                     else if(extension!=null&&AAN!=null&&root==null){
```

```
593                         try{
594                             existe=CH.IdExist(extension,AAN);
595                            //si no existe la Entidad asignadora tambian añadimos
596                            //el perfil a la lista de erróneos
597                         }catch(InvalidAAN e){
598                             a.add(new Index(i));
599                         }catch(InvalidAAName e){
600                             a.add(new Index(i));
601                         }catch(DuplicateAAN e){
602                             a.add(new Index(i));
603                         }
604                         if(existe==true){
605                           //Si ya esta el Id se añade el indice del perfil a la
606                           //lista.
607                             a.add(new Index(i));
608                         }
609
610
611                     }
612                     existe=false;
613                     extension=null;
614                     root=null;
615                 }
616             }
617         }
618         if(a.size()!=0){
619             error=true;
620             throw new InvalidAdministrativeIds(getErrorIndexSeq(a));
621         }
622         return error;
623     }
624
625
626
627     /**
628      * Comprueba que para cada Identificador que se pasa, la informacion
629      * indicada como obligatoria esta presente en la BBDD
630      * @param ids Secuencia de Identificadores a comprobar
631      * @throws MultipleTraits
632      *
633      * @return True si esta registrada.
634      */
635     public boolean Mandatory(PersonIdSeq ids) throws MultipleTraits{
636           // Secuencia que contiene los Traits que estan almacenados para cada
Id
637         TraitNameSeq[] traitNames=new TraitNameSeq[ids.GetPersonIdSeq().length];
638         //Sec que contiene los nombres de los Traits que faltan
639         TraitNameSeq errorTraits=new TraitNameSeq();
640         //Secuencia que se devuelve
641         MultipleFailure[] mfsdef=null;
642         //Entero auxiliar para almacenar el PersonId
643         int auxp=0;
644         boolean existe=false;
645         Vector a=new Vector();
646         Vector b=new Vector();
647         //Tabla que guardara los nombres de los Traits obligatorios que no están
648         //registrados
649         String auxtraits[]=null;
650         boolean ok=true;
651         // Mira en la BBDD los Traits que hay registrados para cada
652         // PersonId de la secuencia parámetro.
653         for(int i=0;i<ids.GetPersonIdSeq().length;i++){
654              traitNames[i]=BP.TraitsKnown(Integer.parseInt
655                      (ids.GetPersonIdSeq()[i].GetPersonId()));
656         }
657         String traits[]=getMandatoryTraits();
658       //Ya tenemos una tabla con los Traits Obligatorios.
```

```
659       //Lo comparamos con los que están registrados para un PersonId en concreto
660            for(int i=0;i<traitNames.length;i++){ //Recorremos los PersonId
661                for(int j=0;j<traits.length;j++){ //recorremos los traits
662                    for(int k=0;k<traitNames[i].seq.length;k++){
663                        if(traitNames[i].seq[k]!=null){
664                            if(traits[j].compareTo(traitNames[i].seq[k])==0){
665                                   existe=true;
666                            }
667                        }
668                    }
669                    if(existe==false){//si el trait obligatorio no esta presente
670                        b.add(traits[j]);
671                        }
672                    else existe=false;
673                }
674           auxtraits=new String[b.size()];
675           //Copiamos los Trait que faltan a una tabla auxiliar
676           if(b.size()!=0){
677                auxp=b.size();
678                for(int l=0;l<auxp;l++ )
679                    auxtraits[l]=b.remove(0).toString();
680                errorTraits=new TraitNameSeq(auxtraits);
681                //creamos el elemento correspondiente de la MultipleFailureSeq
682                a.add((new MultipleFailure(new Index(i),
683                        new ExceptionReason("REQUIRED_TRAITS"), errorTraits)));
684            }
685        }
686        if(a.size()!=0){
687            mfsdef=new MultipleFailure[a.size()];
688            auxp=a.size();
689            for(int l=0;l<auxp;l++ ){
690                mfsdef[l]=(MultipleFailure)a.remove(0);
691                }
692            ok=false;
693            throw new MultipleTraits(new MultipleFailureSeq(mfsdef));
694            }
695        return ok;
696    }
697
698    /**
699     * Genera un PersonIdSeq con los PreferredIds de los PersonId que se pasan
700     * como parámetro.
701     * @param ids Secuencia de PersonIds
702     * @return PersonIdSeq con los preferredIds.
703     */
704    public PersonIdSeq ask_preferredId(PersonIdSeq ids){
705        PersonIdSeq preferredIds=new PersonIdSeq();
706        PersonId aux[]=new PersonId[ids.GetPersonIdSeq().length];
707        String preferredId="";
708        String SQL=null;
709        ResultSet Answer=null;
710        for(int i=0;i<ids.GetPersonIdSeq().length;i++){
711           SQL="SELECT preferredId FROM PersonIdentifiers WHERE extension=""+
712                   ids.GetPersonIdSeq()[i].GetPersonId()+""";
713           Answer=DBM.Ask(SQL);
714           try{
715                if(Answer.next())
716                    preferredId=Answer.getString(1);
717                else
718                    preferredId="";
719                aux[i]=new PersonId(preferredId);
720            }catch(SQLException e){
721                System.out.print(e);
722            }
723        }
724        preferredIds.SetPersonIdSeq(aux);
725        return preferredIds;
726    }
```

```
727
728     /**
729      * Genera la IdInfo de la secuencia de identificadores que se le pasa.
730      * @param ids Secuencia de PersonIds
731      * @return Devuelve un IdInfoSeq
732      */
733     public IdInfoSeq getIdInfo(PersonIdSeq ids){
734         IdInfoSeq idinfo=null;
735         IdInfo seq[]=new IdInfo[ids.GetPersonIdSeq().length];
736         IdStateSeq estados=ask_state(ids);
737         PersonIdSeq preferredId=ask_preferredId(ids);
738
739         for(int i=0;i<ids.GetPersonIdSeq().length;i++){
740             seq[i]=new IdInfo();
741             seq[i].SetId(new PersonId(ids.GetPersonIdSeq()[i].GetPersonId()));
742              seq[i].SetState(new dState(estados.GetIdStateSeq()[i].GetIdState()));
743             seq[i].Setpreferred_Id(
744                    new PersonId(preferredId.GetPersonIdSeq()[i].GetPersonId()));
745         }
746         idinfo=new IdInfoSeq(seq);
747         return idinfo;
748     }
749
750     /**
751      * Comprueba si en la secuencia de Identificadores que se pasa hay alguno
752      * que no se encuentra en el estado indicado. Deolviéndose estot últimos
753      * @param ids_to_check Secuencia de Identificadores a comprobar
754      * @param state Estado a comprobar
755      * @return Devuelve una secuencia con los PersonId que no se encuentran en
756      * dicho estado.
757      * Null si no hay ninguno.
758      */
759     public PersonIdSeq not_in_State(PersonIdSeq ids_to_check,String state){
760         IdStateSeq estado=null;
761         PersonIdSeq errorIds=null;
762         Vector a = new Vector();
763         estado=ask_state(ids_to_check);
764         for(int i=0;i<estado.GetIdStateSeq().length;i++){
765           if(estado.GetIdStateSeq()[i].GetIdState()!=state)
766           {
767        a.add(new PersonId(ids_to_check.GetPersonIdSeq()[i].GetPersonId()));
768           }
769         }
770         if(a.size()!=0){
771              errorIds=new PersonIdSeq();
772              errorIds=getErrorPersonIdSeq(a);
773         }
774         return errorIds;
775     }
776
777     /**
778      * A partir de una tabla de Index, genera un IndexSeq con los índices
779      * correspondientes a cada error
780      * error detectado.
781      * @param error Tabla de Index
782      * @param count Número de elementos de la tabla diferentes a null.
783      * @return Devuelve un IndexSeq que contiene los indices
784      * correspondientes al error detectado.
785      */
786     private IndexSeq getErrorIndexSeq(Vector indicesParam){
787         IndexSeq indices=new IndexSeq();
788         Index auxindex[]=null;
789         int aux=0;
790         auxindex=new Index[indicesParam.size()];
791         while(!indicesParam.isEmpty()){
792             auxindex[aux]=(Index)indicesParam.remove(0);
793             aux++;
794         }
```

```
795         indices=new IndexSeq();
796         indices.SetIndexSeq(auxindex);
797         return indices;
798     }
799     /**
800      * A partir de una tabla de PersonIds, genera un PersonIdSeq con los
801      * PersonId correspondientes a cada error encontrado.
802      * @param error Tabla de PersonIds
803      * @param count Número de elementos de la tabla diferentes a null.
804      * @return
805      */
806     private PersonIdSeq getErrorPersonIdSeq(Vector indicesParam){
807         PersonId duplicateids[]=null;
808         PersonIdSeq idseq=null;
809         duplicateids=new PersonId[indicesParam.size()];
810         int count=0;
811         while(!indicesParam.isEmpty()){
812             duplicateids[count]=(PersonId)indicesParam.remove(0);
813             count++;
814         }
815        idseq=new PersonIdSeq();
816        idseq.SetPersonIdSeq(duplicateids);
817        return idseq;
818     }
819
820     /**
821      * Devuelve nombres de los Traits que son obligatorios
822      *  leyendolos de la tabla InfoPID.
823      * @return Nombre de los Traits obligatorios
824      */
825     private String[] getMandatoryTraits(){
826         String traits[]=null;
827         ResultSet Answer=null;
828         int aux=0;
829         Vector a=new Vector();
830         // Obtenemos los Traits que son Obligatorios
831     String SQL="SELECT Valor FROM InfoPID WHERE Nombre="MandatoryTraits"";
832         Answer=DBM.Ask(SQL);
833         try{
834             while(Answer.next()){
835                 a.add(Answer.getString(1));
836             }
837             }catch(SQLException e){
838                 System.out.print(e);
839             }
840
841         traits=new String[a.size()];
842         aux=0;
843         while(!a.isEmpty()){
844             traits[aux]=a.remove(0).toString();
845             aux++;
846         }
847         return traits;
848     }
849
850     /**
851      * Comprueba que los traits que se pasan tienen un nombre válido
852      * @param profiles
853      */
854     public boolean traitNames(ProfileSeq profiles) throws MultipleTraits{
855         boolean valid=true;
856         Vector a=new Vector();
857         Vector b=new Vector();
858         boolean error=false;
859         int aux=0;
860         String c[]=null;
861         MultipleFailure[] mfsdef=null;
862         // Comprobamos que los nombres de los traits son válidos
```

```
863         for(int i=0;i<profiles.seq.length;i++)
864         {
865             for(int j=0;j<profiles.seq[i].seq.length;j++)
866             {
867                 valid=BP.Traits(profiles.seq[i].seq[j].name);
868                 if(valid==false)
869                 {// Los nombres no válidos se añaden al vector a
870                     error=true;
871                     a.add(profiles.seq[i].seq[j].name);
872                 }
873             }
874             // Si ha habido algún error preparamos el MultipleFailure
875             if(error==true)
876             {
877                 c=new String[a.size()];
878                 if(a.size()!=0)
879                 {
880                     aux=a.size();
881                     for(int l=0;l<aux;l++ )
882                     {
883                         c[l]=a.remove(0).toString();
884                     }
885                     b.add(new MultipleFailure(new Index(i),
886                             new ExceptionReason("UNKNOWN_TRAITS"),
887                             new TraitNameSeq(c)));
888                 }
889             }
890         }
891         // Lanzamos la excepcion correspondiente si es necesario
892         if(b.size()!=0)
893         {
894             mfsdef=new MultipleFailure[b.size()];
895             aux=0;
896             while(!b.isEmpty())
897             {
898                 mfsdef[aux]=(MultipleFailure)b.remove(0);
899                 aux++;
900             }
901             throw new MultipleTraits(new MultipleFailureSeq(mfsdef));
902         }
903         return error;
904     }
905
906 }
```

Clase ExceptionReason.java

```
 1
 2
 3 /* Clase IdState, representa los estados que pueden tomar los
 4  * identificadores
 5  */
 6 package PID.ServiceClasses;
 7
 8 public final class ExceptionReason{
 9     private String Unknown="UNKNOWN_TRAITS";
10     private String Duplicate="DUPLICATE_TRAITS";
11     private String Format="WRONG_TRAIT_FORMAT";
12     private String Required="REQUIRED_TRAITS";
13     private String ReadOnly="READONLY_TRAITS";
14     private String Remove="CANNOT_REMOVE";
15     private String ModifyOrDelete="MODIFY_OR_DELETE";
16     private int value;
17
18
```

```
19      public ExceptionReason(){
20          value=0;
21      }
22
23      public ExceptionReason(String State){
24          if(State=="UNKNOWN_TRAITS")
25              value=0;
26          else if(State=="DUPLICATE_TRAITS")
27              value=1;
28          else if(State=="WRONG_TRAIT_FORMAT")
29              value=2;
30          else if(State=="REQUIRED_TRAITS")
31              value=3;
32          else if(State=="READONLY_TRAITS")
33              value=4;
34          else if(State=="CANNOT_REMOVE")
35              value=5;
36          else if(State=="MODIFY_OR_DELETE")
37              value=6;
38          else
39              System.out.println("Valor invalido: "+State);
40      }
41
42      /**
43       * Funcion para obtener el Estado
44       * @return Devuelve un String con el Estado
45       */
46      public String GetExceptionReason(){
47          String State="";
48          switch(value){
49              case 0: State=Unknown;
50                  break;
51              case 1: State=Duplicate;
52                  break;
53              case 2: State=Format;
54                  break;
55              case 3: State=Required;
56                  break;
57              case 4: State=ReadOnly;
58                  break;
59              case 5: State=Remove;
60                  break;
61              case 6: State=ModifyOrDelete;
62                  break;
63              }
64           return State;
65      }
66
67      /**
68       * Función que establece el motivo de la excepción
69       * @param State Estado que se quiere establecer
70       */
71      public void SetExceptionReason(String State){
72
73          if(State=="UNKNOWN_TRAITS")
74              value=0;
75          else if(State=="DUPLICATE_TRAITS")
76              value=1;
77          else if(State=="WRONG_TRAIT_FORMAT")
78              value=2;
79          else if(State=="REQUIRED_TRAITS")
80              value=3;
81          else if(State=="READONLY_TRAITS")
82              value=4;
83          else if(State=="CANNOT_REMOVE")
84              value=5;
85          else if(State=="MODIFY_OR_DELETE")
86              value=6;
```

```
87          else
88              System.out.println("Valor invalido: "+State);
89
90    }
91      public String toString(){
92          String aux=GetExceptionReason();
93          return aux;
94      }
95  }
96
97
```

Clase IdInfo.java

```
 1 /**
 2  * Clase IdInfo, implementa el tipo de dato IdInfo del
 3  * estándar PIDS de Corbamed.
 4  */
 5
 6 package PID.ServiceClasses;
 7
 8
 9     public final class IdInfo {
10
11              private PersonId Id=null;
12              private IdState state=null;
13              private PersonId preferred_Id=null;
14
15     public  void SetId(PersonId a){
16         Id=a;
17     }
18     public  void SetState(IdState a){
19         state=a;
20     }
21     public  void Setpreferred_Id(PersonId a){
22         preferred_Id=a;
23     }
24
25     public PersonId GetId(){
26         return Id;
27     }
28     public IdState GetState(){
29         return state;
30     }
31     public  PersonId Getpreferred_Id(){
32         return preferred_Id;
33     }
34
35     public String toString(){
36       return Id.GetPersonId()+" "+state.GetIdState()+"
37               +preferred_Id.GetPersonId()+" \n";}
38  }
39
```

Clase IdInfoSeq.java

```
 1 /**
 2  * Clase que representa una secuencia de IdInfo
 3  */
 4
 5 package PID.ServiceClasses;
 6
```

```
 7
 8 public class IdInfoSeq {
 9     private IdInfo seq[];
10     public IdInfoSeq(IdInfo[] ids) {
11         seq=ids;
12     }
13     /**
14      * Establece el valor de la secuencia de IdInfo
15      * @param ids Secuencia de IdInfo
16      */
17     public void SetIdInfoSeq(IdInfo[] ids) {
18         seq=ids;
19     }
20
21     /**
22      * Acceso a la secuencia de IdInfo
23      * @return Secuencia de IdInfo
24      */
25     public IdInfo[] GetInfoIdSeq() {
26         return this.seq;
27     }
28
29     public String toString(){
30         String aux="";
31         for (int i=0;i<seq.length;i++){
32             aux=aux+seq[i].toString();
33         }
34         return aux;
35     }
36
37 }
38
```

Clase IdState.java

```
 1 /* Clase IdState, representa los estados que pueden tomar los
 2  * identificadores
 3  */
 4 package PID.ServiceClasses;
 5
 6 public final class IdState{
 7     private String Unknown="UNKNOWN";
 8     private String Invalid="INVALID";
 9     private String Temporary="TEMPORARY";
10     private String Permanent="PERMANENT";
11     private String Deactivated="DEACTIVATED";
12     private int value;
13
14     public IdState(){
15         value=0;
16     }
17
18     public IdState(String State){
19         if(State=="UNKNOWN")
20             value=0;
21         else if(State=="INVALID")
22             value=1;
23         else if(State=="TEMPORARY")
24             value=2;
25         else if(State=="PERMANENT")
26             value=3;
27         else if(State=="DEACTIVATED")
28             value=4;
29         else
30             System.out.println("Valor invalido: "+State);
31     }
32
```

```
33      /**
34       * Funcion para obtener el Estado
35       * @return Devuelve un String con el Estado
36       */
37      public String GetIdState(){
38          String State="";
39          switch(value){
40              case 0: State=Unknown;
41                  break;
42              case 1: State=Invalid;
43                  break;
44              case 2: State=Temporary;
45                  break;
46              case 3: State=Permanent;
47                  break;
48              case 4: State=Deactivated;
49                  break;
50              }
51           return State;
52      }
53
54      /**
55       * Función que establece el estado
56       * @param State Estado que se quiere establecer
57       */
58      public void SetIdState(String State){
59
60          if(State=="UNKNOWN")
61              value=0;
62          else if(State=="INVALID")
63              value=1;
64          else if(State=="TEMPORARY")
65              value=2;
66          else if(State=="PERMANENT")
67              value=3;
68          else if(State=="DEACTIVATED")
69              value=4;
70          else
71              System.out.println("Valor invalido: "+State);
72     }
73      public String toString(){
74          String estado=null;
75          if(value==0)
76              estado="UNKNOWN";
77          else if(value==1)
78              estado="INVALID";
79          else if(value==2)
80              estado="TEMPORARY";
81          else if(value==3)
82              estado="PERMANENT";
83          else if(value==4)
84              estado="DEACTIVATED";
85
86          return estado;
87      }
88  }
```

Clase IdStateSeq.java

```
1  /**
2   * Clase que representa una secuencia de IdState.
3   */
4
5
6  package PID.ServiceClasses;
7
```

```
 8
 9 public final class IdStateSeq {
10     /**
11      * Secuencia de Identificadores
12      */
13     private IdState seq[]=null;
14
15     /**
16      * Crea una nueva instancia de <B>IdStateSeq</B>.
17      */
18     public IdStateSeq(IdState[] ids) {
19         seq=ids;
20         }
21     /**
22      * Establece el valor de la secuencia
23      */
24     public void SetIdStateSeq(IdState[] ids) {
25         seq=ids;
26     }
27
28     /**
29      * Para acceder a la secuencia
30      */
31     public IdState[] GetIdStateSeq() {
32         return this.seq;
33     }
34 }
35
```

Clase IIExist.java

```
 1 /*
 2  * IIExist.java
 3  * Clase que representa la excepcion IIExist, se lanza cuando se intenta
 4  * registrar un perfil que contiene un Idetificador administrativo que ya está
 5  * registrado.
 6  */
 7
 8 package PID.ServiceClasses;
 9
10 public class IIExist extends Exception {
11     private String Exist=null;
12
13     public IIExist(String param) {
14         Exist=param;
15     }
16     public String toString(){
17         return "Identificador Externo duplicado: "+Exist;
18     }
19 }
20
```

Clase Index.java

```
 1 /*
 2  * Clase Index.java
 3  * Representa el tipo de datos Index del Estándar PIDS
 4  * de CorbaMed
 5  */
 6
 7
```

```
 8 package PID.ServiceClasses;
 9
10 public final class Index {
11     private long index;
12
13
14     public Index(int indice){
15         index=indice;
16     }
17
18     public Index(String indice){
19         index=Long.parseLong(indice);
20     }
21     /**
22      * Método que devuelve el valor del índice
23      * @return Valoer del índice
24      */
25     public long getIndex(){
26         return index;
27     }
28     /**
29      * Metodo que establece el valor del índice
30      * @param indice Valor nuevo del índice
31      */
32     public void setIndex(String indice){
33         index=Long.parseLong(indice);
34     }
35     /**
36      *
37      * @return Cadena que representa al índice
38      */
39     public String toString(){
40         String aux="";
41         aux=aux+index+" ";
42         return aux;
43     }
44
45 }
46
```

Clase IndexSeq.java

```
 1 /**
 2  * Clase que representa una secuencia de Index.
 3  */
 4
 5 package PID.ServiceClasses;
 6
 7
 8 public final class IndexSeq{
 9
10     private Index seq[]=null;
11
12     public IndexSeq(){
13
14     }
15     public IndexSeq(Index[] ids) {
16         seq=ids;
17         }
18      /**
19      * Establece el valor de la secuencia
20      */
21     public void SetIndexSeq(Index[] ids) {
22         seq=ids;
23     }
```

```
24
25      /**
26       * Para acceder a la secuencia
27       */
28      public Index[] GetIndexSeq() {
29          return this.seq;
30      }
31      public String toString(){
32          String aux="";
33          for(int i=0;i<seq.length;i++){
34              aux=aux+seq[i]+" ";
35          }
36          return aux;
37      }
38
39  }
```

Clase InvalidAdministrativeIds.java

```
 1  /*
 2   * Clase Invalidadministrativelds.java
 3   * Implementa la excepción que se lanza cuando se intenta
 4   * registrar un perfil que contiene un identificador
 5   * administrativo que ya se encuentra registrado.
 6   */
 7
 8
 9
10  package PID.ServiceClasses;
11
12
13  public class InvalidAdministrativeIds extends Exception{
14
15      private IndexSeq indices=null;
16
17      public InvalidAdministrativeIds(IndexSeq indicesParam) {
18          indices=indicesParam;
19      }
20       public String toString(){
21          return "InvalidAdministrativeIds: "+indices.toString();
22      }
23
24
25  }
```

Clase MergeStruct.java

```
 1  /**
 2   * Clase MergeStruct usada para las operacione de Merge
 3   * Estructura que contiene el identificador que se va a desactivar
 4   * y el identificador preferido.
 5   *
 6   */
 7
 8
 9  package PID.ServiceClasses;
10
11
12      public final class MergeStruct {
13
14              private PersonId Id=null;
```

```
15              private PersonId preferred_Id=null;
16
17
18     public MergeStruct(){
19     }
20
21      public MergeStruct(PersonId id_param,PersonId pref_param){
22          Id=id_param;
23          preferred_Id=pref_param;
24      }
25
26      /**
27       * Establece el valor del identificado a fusionar
28       * @param Id_param
29       */
30      public  void SetId (PersonId Id_param){
31          Id=Id_param;
32      }
33      /**
34       * Establece el valor del preferredId
35       * @param preferredId_param
36       */
37      public  void Setpreferred_Id (PersonId preferredId_param ){
38          preferred_Id=preferredId_param ;
39      }
40
41      /**
42       * Devuelve el valor del identificador a fusionar
43       * @return Valor del identificador
44       */
45      public PersonId GetId (){
46          return Id;
47      }
48      /**
49       * Devuelve el valor del preferredId
50       * @return Valor del preferredId
51       */
52      public  PersonId Getpreferred_Id (){
53          return preferred_Id;
54      }
55  }
56
```

Clase MergeStructSeq.java

```
 1 /**
 2  * Clase MergeStructSeq
 3  * Secuencia de MergeStruct.
 4  */
 5
 6 package PID.ServiceClasses;
 7
 8
 9 public final class MergeStructSeq {
10
11       public MergeStruct seq[]=null;
12
13 public MergeStructSeq(){
14 }
15
16       public MergeStructSeq(MergeStruct[] ids){
17                 seq=ids;
18            }
19
20
21  }
```

Clase MultipleFailure.java

```
 1 /*
 2  * Clase MultipleFailure.java
 3  *
 4  * Implementa la excepción MultipleFailure
 5  */
 6
 7 package PID.ServiceClasses;
 8
 9 public final class MultipleFailure{
10
11     private Index index;
12     private ExceptionReason reason;
13     private TraitNameSeq traits;
14
15  public MultipleFailure(Index indexParam,ExceptionReason reasonParam,
        TraitNameSeq traitsParam){
16         index=indexParam;
17         reason=reasonParam;
18         traits=traitsParam;
19     }
20
21     public TraitNameSeq getTraitNames(){
22         return traits;
23     }
24     public String toString(){
25         String aux="";
26         aux=aux+"Indice:"+index.getIndex()+"Causa:"+reason.toString()+"Traits:
       +traits.toString();
27         return aux;
28     }
29
30 }
31
```

Clase MultipleFailureSeq.java

```
 1 /*
 2  * Clase MultipleFailureSeq.java
 3  * Implementa una secuencia de MultipleFailure
 4  */
 5
 6
 7 package PID.ServiceClasses;
 8
 9 public final class MultipleFailureSeq{
10
11     MultipleFailure seq[]=null;
12
13     public MultipleFailureSeq(MultipleFailure[] param){
14         seq=param;
15     }
16
17 }
18
```

Clase MultipleTraits.java

```
 1 /**
 2  * Clase Multipletraits.java
 3  *
 4  * Implamenta la excepción MultipleTraits, fallo múltiple
 5  */
 6
 7
 8 package PID.ServiceClasses;
 9
10
11 public class MultipleTraits extends Exception{
12
13     private MultipleFailureSeq mfs=null;
14
15
16
17     public MultipleTraits(MultipleFailureSeq mfsParam) {
18         mfs=mfsParam;
19
20     }
21
22     /**
23      * @return Devuelve una cadena con el valor del objeto.
24      */
25     public String toString(){
26         String aux="";
27         for(int i=0;i<mfs.seq.length;i++)
28                 aux=aux+mfs.seq[i].toString()+"\n";
29         return "MultipleTraits:\n "+aux;
30     }
31 }
32
33
```

Clase NotImplemented.java

```
 1 /*
 2  * Clase NotImplemented.java
 3  * Excepción NotImplementes. se lanza cuando se intenta ejecutar alguna
 4  * funcionalidad del PIDS que no se ha implementado
 5  */
 6
 7
 8
 9 package PID.ServiceClasses;
10
11 public class NotImplemented extends Exception{
12
13
14
15     public NotImplemented() {
16             }
17
18     /**
19      * @return Devuelve una cadena con el valor del objeto.
20      */
21     public String toString(){
22         return "Not implemented";
23     }
24
25 }
```

Clase PersonId.java

```
 1 /**
 2  * Clase que representa el tipo PersonId del Estándar PIDS de Corbamed
 3  *
 4  */
 5
 6 package PID.ServiceClasses;
 7
 8
 9     public final class PersonId {
10
11             private String Id="";
12             //private PersonId ax;
13
14     public PersonId(){
15
16     }
17     public PersonId(String id_param){
18         Id=id_param;
19     }
20
21     /**
22      * Método que establece el valor de PersonId
23      * @param id_param Nuevo valor del identificador
24      */
25     public  void SetPersonId (String id_param){
26         this.Id=id_param;
27     }
28
29     /**
30      * Método que devuelve el valor del PersonId
31      * @return Valor del identificador
32      */
33     public String GetPersonId (){
34          return Id;
35     }
36
37     public String toString(){
38         return Id;
39     }
40 }
```

Clase PersonIdSeq.java

```
 1 /**
 2  *clase que implementa una secuencia de PersonIds
 3  *
 4  */
 5
 6 package PID.ServiceClasses;
 7
 8
 9 public final class PersonIdSeq {
10     /**
11      * Secuencia de Identificadores
12      */
13     private PersonId seq[]=null;
14
15     /**
16      * Crea una nueva instancia de <B>PersonIdSeq</B>.
17      */
18     public PersonIdSeq(PersonId[] ids) {
19         seq=ids;
20         }
```

```
21
22      public PersonIdSeq(){
23
24      }
25      /**
26       * Establece el valor de la secuencia de identificadores
27       * @param ids Secuencia de PersonIds
28       */
29      public void SetPersonIdSeq(PersonId[] ids) {
30          seq=ids;
31      }
32
33      /**
34       * Para acceder a la secuencia de identificadores
35       * @return Secuencia de Identificadores
36       */
37      public PersonId[] GetPersonIdSeq() {
38          return this.seq;
39      }
40
41      public String toString(){
42          String aux="";
43          for(int i=0;i<seq.length;i++){
44             aux=aux+seq[i].GetPersonId()+" ";
45          }
46          return aux;
47      }
48
49
50  }
51
```

Clase RequiredTraits.java

```
 1  /*
 2   * Clase RequiredTraits.java
 3   * Implementa la excepción RequiredTraits
 4   */
 5
 6
 7  package PID.ServiceClasses;
 8
 9  public class RequiredTraits extends Exception{
10
11      String traits;
12
13      public RequiredTraits(String traitsValue) {
14          traits=traitsValue;
15      }
16
17      /**
18       * @return Devuelve una cadena con el valor del objeto.
19       */
20      public String toString(){
21          return "RequiredTraits: "+traits;
22      }
23
24  }
25
```

Paquete PID.IdGest.

Clase IdGest.java

```
 1 /* Clase que contiene las interfaces de los métodos necesarios para
 2 * gestionar los identificadores.
 3 */
 4 package PID.IdGest;
 5
 6 import PID.BasicTypes.*;
 7 import PID.ServiceClasses.*;
 8
 9 public interface IdGest {
10
11     /**
12      * Método para registrar los identificadores en la BBDD
13      * @param extension Extensión del identificador a registrar
14      * @param root OID de la entidad asignadora
15      * @param preferredId Identificador al que se encuentra asociado
16      * @param IVL Periodo de validez
17      * @return InstanceIdentifier con el valor del identificador
18      * @exception InvalidId,InvalidAAN,InvalidIVL
19      */
20     public II registroId (String extension,OID root, String preferredId,
21             IVL_TS IVL)  throws InvalidId,InvalidAAN,InvalidIVL;
22  /**
23  * Método para registrar los identificadores en la BBDD
24  * @param extension Extensión del identificador a registrar
25  * @param AAN Nombre de la entidad asignadora
26  * @param preferredId Identificador al que se encuentra asociado
27  * @param IVL Periodo de validez
28  * @return InstanceIdentifier con el valor del identificador generado
29  * @exception InvalidId,InvalidAAN,InvalidIVL,InvalidAAName,DuplicateAAN
30      */
31     public II registroId (String extension,String AAN,
32           String preferredId, IVL_TS IVL)
33        throws InvalidId,InvalidAAN,InvalidIVL,InvalidAAName,DuplicateAAN;
34  /**
35  * Método para obtener los identificadores según la entidad a la que
36  * pertenezcan.
37  * @param root Entidad de la que se quieren obtener los identificadores
38  * @return Secuencia de InstanceIdentifiers con los identificadores
39  * @exception InvalidAAN
40  */
41     public IISeq getIdsbyAAN (OID root) throws InvalidAAN;
42  /**
43  * Método para obtener los identificadores según la entidad a la que
44  * pertenezcan.
45  * @param AAN Entidad de la que se quieren obtener los identificadores
46  * @return Secuendia de InstanceIdentifiers con los identificadores
47  * @exception InvalidAAName,DuplicateAAN,InvalidAAN
48  */
49     public IISeq getIdsbyAAN (String AAN)
50             throws InvalidAAName,DuplicateAAN,InvalidAAN;
```

```
 51   /**
 52   * Método para obtener identificadores cuyo periodo de validez esté
 53   * dentro del indicado como parametro.
 54   * @param IVL Periodo de validez
 55      * @return Secuencia de InstanceIdentifiers con los identificadores
 56      * @exception InvalidIVL
 57      */
 58     public IISeq getIdsbyIVL (IVL_TS IVL) throws InvalidIVL;
 59     /**
 60      * Método para obtener los identificadores que se encuentran
 61      * fusionados con el que se pasa.
 62      * @param extension Entensión del identificador
 63      * @param root OID de la entidad asignadora.
 64      * @return Secuencia de identificadores.
 65      */
 66     public IISeq getMergedIds (String extension, OID root);
 67     /**
 68      * Metodo para obtener los identificadores que se encuentran
 69      * fusionados con el que se pasa.
 70      * @param extension Entensión del identificador
 71      * @param AAN Nombre de la entidad asignadora.
 72      * @return Secuencia de identificadores.
 73      */
 74     public IISeq getMergedIds (String extension, String AAN);
 75   /**
 76   * Método que permite obtener los identificadores fusionados con el que
se
 77   * pasa y que además pertenezcan a una entidad asignadora en cuestión.
 78   * @param extension Entensión del identificador
 79   * @param root OID de la entidad asignadora.
 80   * @param AANroot OID de la entidad asignadora por la que se pregunta.
 81   * @return Secuencia de identificadores
 82   * @exception InvalidAAN
 83   */
 84     public IISeq getMergedIdsbyAAN(String extension,OID root,OID AANroot)
 85             throws InvalidAAN;
 86     /**
 87      * Método que permite obtener identificadores fusionados con el que
 88      * se pasa y que además pertenezcan a una entidad asignadora en
 89      * @param extension Entensión del identificador
 90      * @param root OID de la entidad asignadora.
 91      * @param AAN Nombre de la entidad asignadora por la que se pregunta.
 92      * @return Secuencia de identificadores
 93      * @exception InvalidAAN,DuplicateAAN,InvalidAAName
 94      */
 95     public IISeq getMergedIdsbyAAN String extension, OID root,String AAN)
 96             throws InvalidAAN,DuplicateAAN,InvalidAAName;
 97     /**
 98      * Método que permite obtener identificadores fusionados con el que
 99      * se pasa y que además pertenezcan a una entidad asignadora en
100      * @param extension Entensión del identificador
101      * @param AAN Nombre de la entidad asignadora.
102      * @param AANroot OID de la entidad asignadora.
103      * @return Secuencia de identificadores
104      * @exception InvalidAAN,DuplicateAAN,InvalidAAName
105      */
```

```
106   public IISeq getMergedIdsbyAAN(String extension,String AAN,OID AANroot)
107             throws InvalidAAN,DuplicateAAN,InvalidAAName;
108     /**
109      * Método que permite obtener identificadores fusionados con el que
110      * se pasa y que además pertenezcan a una entidad asignadora en
111      * @param extension Entensión del identificador
112      * @param AAN Nombre de la entidad asignadora.
113      * @param AANDes Nombre de la entidad asignadora.
114      * @return Secuencia de identificadores
115      * @exception InvalidAAN,DuplicateAAN,InvalidAAName
116      */
117 public IISeq getMergedIdsbyAAN(String extension,String AAN,String AANDes)
118             throws InvalidAAN,DuplicateAAN,InvalidAAName;
119
120     /**
121      * Método que devuelve el identificador que se pasa en el formato
122      * correspondiente al estándar OpenEHR
123      * @param extension Extension del identificador
124      * @param root OID de la entidad asignadora del identificador
125      * @return DV_Identifier.
126      * @exception InvalidAAN,InvalidId
127      */
128     public DV_IDENTIFIER getOEhrId(String extension, OID root)
129             throws InvalidAAN,InvalidId;
130     /**
131      * Método que devuelve el identificador que se pasa en el formato
132      * correspondiente al estándar CorbaMed
133      * @param extension Extension del identificador
134      * @param root OID de la entidad asignadora del identificador
135      * @return QualifiedPersonId.
136      * @exception InvalidAAN,InvalidId
137      */
138     public QualifiedPersonId getPIDSId(String extension,OID root)
139             throws InvalidAAN,InvalidId;
140
141     /**
142      * Método para fusionar identificadores administrativos.
143      * @param extension Extension del Identificador
144      * @param root Entidad asignadora
145      * @param preferredId Extension del Identificador con el que se va
146      * a fusionar
147      * @return Instance identifier con el valor del identificador
148      * @exception InvalidAAN,InvalidId
149      */
150     public II mergeIds(String extension, OID root,String preferredId)
151             throws InvalidAAN,InvalidId;
152     /**
153      * Método para fusionar identificadores administrativos.
154      * @param extension Extension del Identificador
155      * @param AAN Nombre de la Entidad asignadora
156      * @param preferredId Extension del Identificador con el que se va
157      * a fusionar
158      * @return Instance identifier con el valor del identificador
159      * @exception InvalidAAN,DuplicateAAN,InvalidAAName,InvalidId
160      */
161     public II mergeIds(String extension,String AAN,String preferredId)
162             throws InvalidAAN,DuplicateAAN,InvalidAAName,InvalidId;
```

```
163
164     /**
165      * Método para cambiar el periodo de validez del identificador
166      * administrativo
167      * @param IVL Nuevo periodo de validez
168      * @param extension Extension del identificador a cambiar
169      * @param root Entidad asignadora
170      * @return Instance Identifier con el valor actualizado.
171      * @exception InvalidId,InvalidAAN
172      */
173     public II setIVL (IVL_TS IVL,String extension, OID root )
174             throws InvalidId,InvalidAAN;
175     /**
176      * Método para cambiar el periodo de validez del identificador
177      * administrativo
178      * @param IVL Nuevo periodo de validez
179      * @param extension Extension del identificador a cambiar
180      * @param AAN Nombre entidad asignadora
181      * @return Instance Identifier con el valor actualizado.
182      * @exception InvalidAAN,DuplicateAAN,InvalidAAName,InvalidId
183      */
184     public II setIVL (IVL_TS IVL,String extension, String AAN)
185             throws InvalidAAN,DuplicateAAN,InvalidAAName,InvalidId;
186
187     /**
188      * Método para registrar una nueva entidad asignadora
189      * en la BBDD
190      * @param root OID de la entidad asignadora
191      * @param AAN Nombre de la entidad.
192      * @return true si todo va bien.
193      * @exception InvalidAAN
194      */
195     public boolean registroAAN(OID root,String AAN) throws InvalidAAN;
196 }
197
```

Clase IdGestImpl.java

```
 1 package PID.IdGest;
 2
 3 import PID.BasicTypes.*;
 4 import PID.IdGest.InvalidId;
 5 import java.sql.*;
 6 import java.util.*;
 7 import PID.DataBase.*;
 8 import PID.ServiceClasses.*;
 9
10 public class IdGestImpl implements IdGest {
11
12       DB_Manager DBM=null;
13       private Comprueba ch=null;
14
15
16       public IdGestImpl(DB_Manager DB){
17        DBM=DB;
18        ch=new Comprueba(DBM);
19
20       }
21     /**
22      * Método para registrar los identificadores en la BBDD
23      * @param extension Extensión del identificador a registrar
```

```
24      * @param root OID de la entidad asignadora
25      * @param preferredId Identificador al que se encuentra asociado
26      * @param IVL Periodo de validez
27      * @return InstanceIdentifier con el valor del identificador
28      * @exception InvalidId,InvalidAAN,InvalidIVL
29      */
30     public II registroId (String extension,OID root, String preferredId,
31             IVL_TS IVL)  throws InvalidId,InvalidAAN,InvalidIVL{
32         boolean error=false;
33         ResultSet Answer=null;
34         String SQL=null;
35         //Comprobamos si el identificador está registrado.
36         if(root.getOID().compareTo(getLocalDomainOID().getOID())==0)
37         {
38             ch.IdExist(extension);
39         }
40         else
41             error= ch.IdExist(extension,root);
42
43         if(error)
44             throw new InvalidId(extension, root);
45         if(!error)
46              // Comprobamos que la AAN está registrada true si lo esta
47             error=ch.AAN(root);
48         if(error&&(IVL!=null))
49             //si la AAN esta registrada compruebo el IVL true si OK
50             error=ch.IVL(IVL);
51         if(error){ //si esta ok compruebo el preferredId
52             if(root.getOID().compareTo(getLocalDomainOID().getOID())!=0)
53                 error=ch.IdExist(preferredId);
54             if(!error) //si error=false no existe
55                 throw new InvalidId(preferredId, getLocalDomainOID());
56         }
57         if(error){ //si error=true todo bien.->registro el Identificador
58             if(root.getOID().compareTo(getLocalDomainOID().getOID())==0)
59             {
60           SQL="INSERT INTO Person(PersonId) VALUES (""+extension+"")";
61           DBM.change(SQL);
62           SQL="INSERT INTO PersonIdentifiers(extension,state) VALUES (""
63                       +extension+"","TEMPORARY")";
64             }
65             else
66           SQL="INSERT INTO AdministrativeIdentifiers(extension,root," +
67                    "preferredId) VALUES (""+extension+"",""+root+"",""+
68                       preferredId+"")";
69
70             DBM.change(SQL);
71             setIVL(IVL,extension,root);
72         }
73 //Obtenemos el II correspondiente al identificador que se ha registrado
74         II Id=getII(extension, root);
75
76         return Id;
77     }
78
```

```
 79  /**
 80  * Método para registrar los identificadores en la BBDD
 81  * @param extension Extensión del identificador a registrar
 82  * @param AAN Nombre de la entidad asignadora
 83  * @param preferredId Identificador al que se encuentra asociado
 84  * @param IVL Periodo de validez
 85  * @return InstanceIdentifier con el valor del identificador generado
 86  * @exception InvalidId,InvalidAAN,InvalidIVL,InvalidAAName,DuplicateAAN
 87  */
 88    public II registroId (String extension,String AAN, String preferredId,
 89        IVL_TS IVL)  throws InvalidId,InvalidAAN,InvalidIVL,InvalidAAName,
 90        DuplicateAAN{
 91         OID root=null;
 92         II Ids=null;
 93         root=ch.AAN(AAN);  // Comprobamos que la AAN está registrada
 94         if(root!=null)
 95             Ids=registroId(extension,root,preferredId,IVL);
 96         return Ids;
 97     }
 98   /**
 99   * Método para obtener los identificadores según la entidad a la que
100   * pertenezcan.
101   * @param root Entidad de la que se quieren obtener los identificadores
102   * @return Secuencia de InstanceIdentifiers con los identificadores
103   * @exception InvalidAAN
104   */
105   public IISeq getIdsbyAAN (OID root) throws InvalidAAN{
106     boolean error;
107     error=ch.AAN(root);     // Comprobamos que la AAN está registrada
108     String SQL="SELECT extension,root FROM AdministrativeIdentifiers " +
109                 "WHERE (root=""+root+"")";
110         ResultSet Answer=null;
111         Answer=DBM.Ask(SQL);
112         Vector IIVector=new Vector();
113         IISeq IIS=null;
114         try{ // Añadimos al Vector los identificadores devueltos
115             while(Answer.next()){
116                 IIVector.add(getII(Answer.getString(1),
117                         new OID(Answer.getString(2))));
118                 }
119            }catch(SQLException e){
120                 System.out.print(e);
121            }
122         int size=IIVector.size();
123         II[] IIs=new II[size];
124         for(int i=0;i<IIs.length;i++){
125             IIs[i]=(II)IIVector.remove(0);
126         }
127         // Generamos la secuencia de II a devolver.
128         IIS= new IISeq(IIs);
129         return IIS;
130     }
131
132
133   /**
134   * Método para obtener los identificadores según la entidad a la que
135   * pertenezcan.
136   * @param AAN Entidad de la que se quieren obtener los identificadores
137   * @return Secuencia de InstanceIdentifiers con los identificadores
138   * @exception InvalidAAName,DuplicateAAN,InvalidAAN
139   */
```

```
140     public IISeq getIdsbyAAN (String AAN) throws DuplicateAAN,
141             InvalidAAName,InvalidAAN{
142         OID root=null;
143         IISeq IIS=null;
144         // Comprobamos que la AAN existe y no hay ambigüedad
145         root=ch.AAN(AAN);
146         // Llamamos al método correspondiente.
147         if(root!=null)
148             IIS=getIdsbyAAN(root);
149         return IIS;
150     }
151
152     /**
153      * Método para obtener identificadores cuyo periodo de validez esté
154      * dentro del indicado como parametro.
155      * @param IVL Periodo de validez
156      * @return Secuencia de InstanceIdentifiers con los identificadores
157      * @exception InvalidIVL
158      */
159     public IISeq getIdsbyIVL (IVL_TS IVL) throws InvalidIVL{
160         Vector IIVector=new Vector();
161         IISeq IIS=null;
162         ResultSet Answer=null;
163         // Comprobamos que el formato del periodo de validez es correcto.
164         ch.IVL(IVL);
165         //Primero miramos en la tabla de Ids administrativos
166         //Selecciono los Ids entre las 2 fechas dadas.
167         //(Quitando la ultima parte de la hora)
168     String SQL="SELECT extension,root FROM AdministrativeIdentifiers " +
169                 "WHERE (low>#"+IVL.getLow().toString().substring(
170                 0,IVL.getLow().toString().length()-2)+"# AND high<#"+
171                 IVL.getHigh().toString().substring(
172                 0,IVL.getLow().toString().length()-2)+"#)";
173         Answer=DBM.Ask(SQL);
174         try {
175             while(Answer.next()){
176                 IIVector.add(getII(Answer.getString(1),
177                     new OID(Answer.getString(2))));
178                 }
179            }catch(SQLException e){
180                 System.out.print(e);
181            }
182         //Luego comprobamos la tabla de PersonIds
183         SQL="SELECT extension,root FROM PersonIdentifiers WHERE (low>#"+
184              IVL.getLow().toString().substring(0,IVL.getLow().toString().
185                 length()-2)+"# AND high<#"+IVL.getHigh().toString().
186                 substring(0,IVL.getLow().toString().length()-2)+"#)";
187         Answer=DBM.Ask(SQL);
188         try {
189            while(Answer.next()){
190                 IIVector.add(getII(Answer.getString(1),
191                     new OID(Answer.getString(2))));
192                 }
193            }catch(SQLException e){
194                 System.out.print(e);
195            }
196         int size=IIVector.size();
197         II[] IIs=new II[size];
198         for(int i=0;i<IIs.length;i++){
199              IIs[i]=(II)IIVector.remove(0);
200         }
```

```
201         // Generamos la secuencia a devolver.
202         IIS= new IISeq(IIs);
203 
204         return IIS;
205     }
206     /**
207      * Método para obtener los identificadores que se encuentran
208      * fusionados con el que se pasa.
209      * @param extension Entensión del identificador
210      * @param root OID de la entidad asignadora.
211      * @return Secuencia de identificadores.
212      */
213     public IISeq getMergedIds (String extension, OID root){
214         Vector IIVector=new Vector();
215         Vector IIVectoraux=new Vector();
216         IISeq IIS=null;
217         String preferredId=null;
218         ResultSet Answer;
219         II aux=new II();
220         String SQL=null;
221         // Miramos si el identificador es un PersonID,si lo es
222         // vemos si esta fusionado con algún otro.
223         if(root.getOID().compareTo(getLocalDomainOID().getOID())==0){
224             SQL="SELECT preferredId FROM PersonIdentifiers  WHERE " +
225                     "(extension=""+extension+"")";}
226         else
227         SQL="SELECT preferredId FROM AdministrativeIdentifiers WHERE " +
228              "(extension=""+extension+"" AND root=""+root.getOID()+"")";
229 
230         Answer=DBM.Ask(SQL);
231         try{
232             if(Answer.next()){
233                 //si esta fusionado con otro vemos que otros PersonId
234                 //que están fusionados con ese.
235                 preferredId=Answer.getString(1);
236                 if(preferredId.length()!=0){
237                   IIVector.add(getII(preferredId,getLocalDomainOID()));
238                    //Buscamos el resto de identificadores
239                   SQL="SELECT extension FROM PersonIdentifiers  " +
240                       "WHERE preferredId=""+preferredId+""";
241                   Answer=DBM.Ask(SQL);
242                   while(Answer.next()){ //añadimos el resto de PersonIds
243                         IIVector.add(getII(Answer.getString(1),
244                                 getLocalDomainOID()));
245                     }
246                     //Ya tenemos todos los PersonId con los que está
247                     //fusionado el que se pasa.
248                 }
249             else{
250                 // Añado el identificador
251                 IIVector.add(getII(extension,getLocalDomainOID()));
252                 // Miro si es el preferredId de algún otro
253                 SQL="SELECT extension FROM PersonIdentifiers  " +
254                         "WHERE preferredId=""+extension+""";
255                 // busco si hay alguno asociado a él.
256                  Answer=DBM.Ask(SQL);
257                  while(Answer.next()){ //añadimos el resto de PersonIds
258                         IIVector.add(getII(Answer.getString(1),
259                                 getLocalDomainOID()));
260                     }
261             }
```

```
262             //Finalmente tengo una lista de PersonId relacionados
263             //añadimos los identificadores administrativos
264             //asociados a cada PersonId correspondiente
265             int k=IIVector.size();
266             for(int i=0;i<k;i++){
267                 aux=(II)IIVector.get(i);
268                 SQL="SELECT extension,root FROM " +
269                     "AdministrativeIdentifiers  WHERE " +
270                     "preferredId=""+aux.getExtension()+""";
271                 //Añadimos el resto de Ids que tienen
272                 //el mismo preferredID
273                 Answer=DBM.Ask(SQL);
274                 while(Answer.next()){
275                     IIVector.add(getII(Answer.getString(1),
276                          new OID(Answer.getString(2))));
277                }
278             }
279         }
280         }catch(SQLException e){
281             System.out.println(e);
282         }
283         int size=IIVector.size();
284         // si hay identificadores genero la secuencia a devolver.
285         if(size>0){
286             II[] IIs=new II[size];
287             for(int i=0;i<IIs.length;i++){
288                  IIs[i]=(II)IIVector.remove(0);
289              }
290             IIS= new IISeq(IIs);
291         }
292         return IIS;
293     }
294
295     /**
296      * Metodo para obtener los identificadores que se encuentran
297      * fusionados con el que se pasa.
298      * @param extension Entensión del identificador
299      * @param AAN Nombre de la entidad asignadora.
300      * @return Secuencia de identificadores.
301      */
302     public IISeq getMergedIds (String extension, String AAN){
303         OID root=null;
304         IISeq IIS=null;
305         try{
306             // Comprobamos que la AAN existe y no hay ambigüedad
307             root=ch.AAN(AAN);
308             }catch(DuplicateAAN error){
309                 System.out.print(error);
310             }
311             catch(InvalidAAName error){
312                 System.out.print(error);
313             }
314         if(root!=null)
315             IIS=getMergedIds(extension,root);
316         return IIS;
317     }
318     /**
319      * Método que permite obtener los identificadores fusionados con el
320      * que se pasa y que pertenezcan a una entidad asignadora dada
321      * @param extension Entensión del identificador
322      * @param root OID de la entidad asignadora.
```

```
323      * @param AANroot OID de la entidad asignadora por la que se
324      * @return Secuencia de identificadores
325      * @exception InvalidAAN
326      */
327     public IISeq getMergedIdsbyAAN (String extension, OID root,
328             OID AANroot) throws InvalidAAN{
329         String SQL=null;
330         ResultSet Answer;
331         Vector IIVector=new Vector();
332         IISeq IIS=null;
333         boolean error=false;
334         ch.AAN(root);
335         ch.AAN(AANroot);
336         String preferredId=null;
337         II aux=new II();
338         if(root.getOID().compareTo(getLocalDomainOID().getOID())==0){
339           SQL="SELECT preferredId FROM PersonIdentifiers  WHERE " +
340                     "(extension=""+extension+"")";
341         }
342         else
343         SQL="SELECT preferredId FROM AdministrativeIdentifiers WHERE " +
344              "(extension=""+extension+"" AND root=""+root.getOID()+"")";
345           Answer=DBM.Ask(SQL);
346           try{
347             if(Answer.next()){
348                 //si esta fusionado con otro vemos el resto
349                 //que están fusionados con ese preferredId
350                 preferredId=Answer.getString(1);
351                 if(preferredId.length()!=0){
352                     // Primero Añadimos el preferredId a la lista
353                   IIVector.add(getII(preferredId,getLocalDomainOID()));
354                     //Buscamos el resto de identificadores
355                   SQL="SELECT extension FROM PersonIdentifiers  " +
356                         "WHERE preferredId=""+preferredId+""";
357                   Answer=DBM.Ask(SQL);
358                   while(Answer.next()){ //añadimos el resto de PersonIds
359                         IIVector.add(getII(Answer.getString(1),
360                                 getLocalDomainOID()));
361                     }
362                //Ya tenemos todos los PersonId con los que está fusionado
363                   //el que se pasa.
364                 }
365                 else{
366                     // Añado el identificador
367                     IIVector.add(getII(extension,getLocalDomainOID()));
368
369                     // Miro si es el preferredId de algún otro
370                     SQL="SELECT extension FROM PersonIdentifiers  " +
371                             "WHERE preferredId=""+extension+""";
372                     // busco si hay alguno asociado a él.
373                     Answer=DBM.Ask(SQL);
374                  while(Answer.next()){ //añadimos el resto de PersonIds
375                         IIVector.add(getII(Answer.getString(1),
376                                 getLocalDomainOID()));
377                     }
378                 }
379                 //Finalmente tengo una lista de PersonId relacionados
380                 //añadimos los identificadores administrativos
381                 //asociados a cada PersonId correspondiente
382                 int k=IIVector.size();
383                 for(int i=0;i<k;i++){
```

```
384                     aux=(II)IIVector.remove(0);
385                     SQL="SELECT extension,root FROM " +
386                     "AdministrativeIdentifiers  WHERE " +
387                     "preferredId=""+aux.getExtension()+
388                     "" AND root=""+AANroot.getOID()+""";
389                     //Añadimos el resto de Ids que tienen
390                     //el mismo preferredID
391                     Answer=DBM.Ask(SQL);
392                     while(Answer.next()){
393                         IIVector.add(getII(Answer.getString(1),
394                              new OID(Answer.getString(2))));
395                     }
396                 }
397             }
398         }catch(SQLException e){
399             System.out.println(e);
400         }
401         int size=IIVector.size();
402         if(size>0){
403             II[] IIs=new II[size];
404             for(int i=0;i<IIs.length;i++){
405                  IIs[i]=(II)IIVector.remove(0);
406             }
407             IIS= new IISeq(IIs);
408         }
409 
410         return IIS;
411     }
412     /**
413      * Método que permite obtener los identificadores fusionados con el
414      * que se pasa y que pertenezcan a una entidad asignadora dada
415      * @param extension Entensión del identificador
416      * @param root OID de la entidad asignadora.
417      * @param AAN Nombre de la entidad asignadora por la que se pregunta.
418      * @return Secuencia de identificadores
419      * @exception InvalidAAN,DuplicateAAN,InvalidAAName
420      */
421     public IISeq getMergedIdsbyAAN(String extension, OID root,String AAN)
422     throws InvalidAAN,DuplicateAAN,InvalidAAName{
423         OID rootAAN=null;
424         IISeq IIS=null;
425         // Comprobamos que la AAN existe y no hay ambigüedad
426         rootAAN=ch.AAN(AAN);
427         if(rootAAN!=null)
428             IIS=getMergedIdsbyAAN(extension,root,rootAAN);
429          return IIS;
430     }
431     /**
432      * Método que permite obtener los identificadores fusionados con el
433      * que se pasa y que pertenezcan a una entidad asignadora en dada
434      * @param extension Entensión del identificador
435      * @param AAN Nombre de la entidad asignadora.
436      * @param AANroot OID de la entidad asignadora
437      * @return Secuencia de identificadores
438      * @exception InvalidAAN,DuplicateAAN,InvalidAAName
439      */
440   public IISeq getMergedIdsbyAAN(String extension,String AAN,OID AANroot)
441     throws InvalidAAN,DuplicateAAN,InvalidAAName{
442         OID root=null;
443         IISeq IIS=null;
444         root=ch.AAN(AAN);//Comprobamos que la AAN
```

```
445         if(root!=null)
446             IIS=getMergedIdsbyAAN(extension,root,AANroot);
447          return IIS;
448     }
449     /**
450      * Método que permite obtener los identificadores fusionados con el
451      * que se pasa y que además pertenezcan a una entidad dada.
452      * @param extension Entensión del identificador
453      * @param AAN Nombre de la entidad asignadora.
454      * @param AANDes Nombre de la entidad asignadora por la que se
455      * @return Secuencia de identificadores
456      * @exception InvalidAAN,DuplicateAAN,InvalidAAName
457      */
458 public IISeq getMergedIdsbyAAN(String extension,String AAN,String AANDes)
459    throws InvalidAAN,DuplicateAAN,InvalidAAName{
460         OID root=null;
461         OID rootAAN=null;
462         IISeq IIS=null;
463         root=ch.AAN(AAN); // Comprobamos que la AAN existe y no hay
464         rootAAN=ch.AAN(AANDes);
465         if(root!=null)
466             IIS=getMergedIdsbyAAN(extension,root,rootAAN);
467          return IIS;
468     }
469
470
471     /**
472      * Método que devuelve el identificador que se pasa en el formato
473      * correspondiente al estándar OpenEHR
474      * @param extension Extension del identificador
475      * @param root OID de la entidad asignadora del identificador
476      * @return DV_Identifier.
477      * @exception InvalidAAN,InvalidId
478      */
479     public DV_IDENTIFIER getOEhrId(String extension, OID root)
480     throws InvalidAAN,InvalidId{
481
482         DV_IDENTIFIER ident = new DV_IDENTIFIER(root.getOID(),
483             root.getOID(),extension,null);
484         return ident;
485     }
486     /**
487      * Método que devuelve el identificador que se pasa en el formato
488      * correspondiente al estándar CorbaMed
489      * @param extension Extension del identificador
490      * @param root OID de la entidad asignadora del identificador
491      * @return QualifiedPersonId.
492      * @exception InvalidAAN,InvalidId
493      */
494     public QualifiedPersonId getPIDSId(String extension,OID root )
495     throws InvalidAAN,InvalidId{
496         //Comprobamos que existe el identificador
497         boolean ok=false;
498         ok=ch.IdExist(extension,root);
499         if(ok==false)
500             throw new InvalidId(extension, root);
501
502         QualifiedPersonId id=new QualifiedPersonId(new DomainName(
503          new RegistrationAuthority("ISO"),
504          new NamingEntity(getDomainName(root))),new PersonId(extension));
505         return id;
```

```
506     }
507      /**
508      * Método para fusionar identificadores administrativos.
509      * @param extension Extension del Identificador
510      * @param root Entidad asignadora
511      * @param preferredId Extension del Identificador con el que se va
512      * a fusionar
513      * @return Instance identifier con el valor del identificador
514      * @exception InvalidAAN,InvalidId
515      */
516     public II mergeIds(String extension, OID root,String preferredId)
517     throws InvalidAAN,InvalidId{
518         boolean ok=false;
519         ResultSet Answer=null;
520         OID LocalDomainOID=new OID();
521         String SQL=null;
522         //Comprobamos que el identificador a fusionar existe
523         ok=ch.IdExist(extension,root);
524         if(ok==false)
525             throw new InvalidId(extension, root);
526 //Comprobamos que el PersonId con el que se va a fusionar también existe
527         ok=ch.IdExist(preferredId);
528         if(ok==false)
529             throw new InvalidId(preferredId, getLocalDomainOID());
530    SQL="UPDATE AdministrativeIdentifiers SET preferredId=""+preferredId+
531                 "" WHERE extension=""+extension+"" AND root=""+root+""";
532         int i=DBM.change(SQL);
533         II Id=null;
534         Id=getII(extension,root);
535         return Id;
536     }
537     /**
538      * Método para fusionar identificadores administrativos.
539      * @param extension Extension del Identificador
540      * @param AAN Nombre de la Entidad asignadora
541      * @param preferredId Extension del Identificador con el que se va
542      * a fusionar
543      * @return Instance identifier con el valor del identificador
544      * @exception InvalidAAN,DuplicateAAN,InvalidAAName,InvalidId
545      */
546     public II mergeIds(String extension,String AAN,String preferredId)
547     throws InvalidAAN,DuplicateAAN,InvalidAAName,InvalidId{
548         II Id=null;
549         // Obtenemos el OID correspondiente a la entidad asignadora del
550         // identificador administrativo.
551         OID root=ch.AAN(AAN);
552         if(root!=null)
553             Id=mergeIds(extension, root, preferredId);
554         return Id;
555     }
556      /**
557      * Método para cambiar el periodo de validez del identificador
558      * administrativo
559      * @param IVL Nuevo periodo de validez
560      * @param extension Extension del identificador a cambiar
561      * @param root Entidad asignadora
562      * @return Instance Identifier con el valor actualizado.
563      * @exception InvalidId,InvalidAAN
564      */
```

```
565     public II setIVL (IVL_TS IVL,String extension, OID root )
566     throws InvalidId,InvalidAAN{
567        Short lowClosed=0,highClosed=0;
568        II Id=null;
569        boolean ok=false;
570        String SQL=null;
571        if(root.getOID().compareTo(getLocalDomainOID().getOID())==0){
572        SQL="UPDATE PersonIdentifiers SET low=#"+IVL.getLow().toString().
573             substring(0,IVL.getLow().toString().length()-2)
574             +"#, high=#"+IVL.getHigh().toString().substring(
575             0,IVL.getLow().toString().length()-2)+"#" +
576             " ,lowClosed=""+lowClosed+"" ,highClosed=""+highClosed+
577         "" ,width=""+IVL.getwidth()+"" WHERE extension=""+extension+""";
578            ok=ch.IdExist(extension);
579        }
580        else{
581             SQL="UPDATE AdministrativeIdentifiers SET low=#"+
582                  IVL.getLow().toString().substring(
583                  0,IVL.getLow().toString().length()-2)+"#, high=#"+
584                  IVL.getHigh().toString().substring(
585                  0,IVL.getLow().toString().length()-2)+"#" +
586                  ,lowClosed=""+lowClosed+"" ,highClosed=""+highClosed+
587                   "" ,width=""+IVL.getwidth()+"" WHERE (extension=""+
588                     extension+"" AND " +"root=""+root+"")";
589            ok=ch.IdExist(extension,root);
590        }
591        if(ok==false)
592            throw new InvalidId(extension, root);
593        if (IVL.getlowClosed()==true)
594            lowClosed=-1;
595        if (IVL.gethighClosed()==true)
596             highClosed=-1;
597
598        DBM.change(SQL);
599        Id=getII(extension, root);
600
601         return Id;
602     }
603     /**
604      * Método para cambiar el periodo de validez del identificador
605      * administrativo
606      * @param IVL Nuevo periodo de validez
607      * @param extension Extension del identificador a cambiar
608      * @param AAN Nombre entidad asignadora
609      * @return Instance Identifier con el valor actualizado.
610      * @exception InvalidAAN,DuplicateAAN,InvalidAAName,InvalidId
611      */
612     public II setIVL (IVL_TS IVL,String extension, String AAN)
613             throws InvalidAAN,DuplicateAAN,InvalidAAName,InvalidId {
614        II Id=null;
615        OID root=ch.AAN(AAN);
616        if(root!=null)
617             Id=setIVL (IVL, extension, root );
618        return Id;
619     }
620
```

```
621     /**
622      * Dadas la extension y AAN de un Id devuelve el II correspondiente.
623      * @param extension Extension del identificador.
624      * @param root OID de la entidad asignadora correspondiente
625      * @return Instance Identifier correspondiente al identificador.
626      */
627     public II getII(String extension,OID root){
628         //
629
630         String SQL="";
631         if(root.getOID().compareTo(getLocalDomainOID().getOID())==0){
632             SQL="SELECT * FROM PersonIdentifiers WHERE (extension=""+
633                     extension+"")";
634         }
635       else if root.getOID().compareTo(getLocalDomainOID().getOID())!=0){
636        SQL="SELECT * FROM AdministrativeIdentifiers WHERE (root=""+root+
637                     "" AND extension=""+extension+"")";
638         }
639         ResultSet Answer=null;
640         ResultSet Answer1=null;
641         String SQL1="SELECT AssigningAuthorityName FROM
642                AssigningAuthorityName"+ " WHERE root=""+root+""";
643         Answer1=DBM.Ask(SQL1);
644         Answer=DBM.Ask(SQL);
645         String AAN=null;
646         Vector IIVector=new Vector();
647         IISeq IIS=null;
648         II Id=null;
649         try{  //Obtenemos el nombre de la entidad asignadora
650             if(Answer1.next())
651                 AAN=Answer1.getString(1);
652             //Obtenemos los datos del identificador
653         while(Answer.next()){
654           if(root.getOID().compareTo(getLocalDomainOID().getOID())==0){
655               Id= new II(Answer.getString(1),getLocalDomainOID()
656           ,AAN,new IVL_TS(Answer.getTimestamp(2),Answer.getTimestamp(3),
657                   Answer.getBoolean(4), Answer.getBoolean(5),
658                   Answer.getString(6)));
659                }
660                else{
661               Id= new II(Answer.getString(1),new OID(Answer.getString(2))
662            ,AAN,new IVL_TS(Answer.getTimestamp(3),Answer.getTimestamp(4),
663                   Answer.getBoolean(5), Answer.getBoolean(6),
664                   Answer.getString(7)));
665                }
666
667             }
668         }catch(SQLException e){
669             System.out.print(e);
670             }
671         return Id;
672     }
```

```
673      /**
674       * Método para registrar una nueva entidad asignadora
675       * en la BBDD
676       * @param root OID de la entidad asignadora
677       * @param AAN Nombre de la entidad.
678       * @return true si todo va bien.
679       * @exception InvalidAAN
680       */
681     public boolean registroAAN(OID root,String AAN) throws InvalidAAN{
682         boolean error=false;
683         String SQL="";
684         ResultSet Answer=null;
685     SQL="SELECT root FROM AssigningAuthorityName WHERE root=""+root+""";
686         try{  //Obtenemos el nombre de la entidad asignadora,
687             Answer=DBM.Ask(SQL);
688             if(Answer.next())
689                 throw new InvalidAAN(root);
690         }catch(SQLException e){
691                 System.out.print(e);
692         }
693         // si se indica el nombre de la entidad asignadora se registra.
694         // sino se le asigna el nombre unknown.
695         if (AAN.trim().length()==0){
696          SQL="INSERT INTO AssigningAuthorityName
697          (root,AssigningAuthorityName)"+" VALUES (""+root+"","Unknown")";
698          DBM.change(SQL);
699         }
700         else{
701          SQL="INSERT INTO AssigningAuthorityName
702         (root,AssigningAuthorityName)"+" VALUES (""+root+"",""+AAN+"")";
703          DBM.change(SQL);
704         }
705         return error;
706
707     }
708
709     /**
710      * Metodo que devuelve el OID del dominio local
711      * @return OID correspondiente al dominio local.
712      */
713     private OID getLocalDomainOID(){
714        OID root=null;
715        ResultSet Answer=null;
716     String SQL="SELECT valor FROM InfoPID WHERE Nombre="LocalDomainOID"";
717         Answer=DBM.Ask(SQL);
718         try{
719             if(Answer.next()){
720                 root=new OID(Answer.getString(1));
721             }
722        }catch (SQLException e){
723             System.out.print(e);
724        }
725        return root;
726     }
727
728     /**
729      * Método que devuelve el NameForm del OID indicado.
730      * @param rootname OID de la Entidad asignadora.
731      * @return Nombre de la Entidad.
732      */
733     public String getDomainName(OID rootname){
```

```
734         String name="";
735         String OID=rootname.getOID();
736         boolean end=false;
737         String aux1="";
738         String aux2="";
739         ResultSet Answer=null;
740         String SQL=null;
741         //Primero obtengo el nombre del dominio
742   SQL="SELECT assigningAuthorityName FROM AssigningAuthorityName WHERE "
743               +  "root=""+OID+""";
744         Answer=DBM.Ask(SQL);
745         try{
746             if(Answer.next()){
747                 aux2=Answer.getString(1);
748                 name=aux2;
749             }
750             // Si la entidad no tiene registrado su nombre le asignamos
751             // Unknown.
752             else{
753                 aux2="Unknown";
754                 aux2=aux2+" "+name;
755                 name=aux2;
756             }
757         }catch (SQLException e){
758             System.out.print(e);
759         }
760         // Vamos retrocediendo en la jerarquia de dominios. Y obteniendo
761         // el nombre correspondiente.
762         while(end!=true){
763             if(OID.indexOf(".")!=-1){
764                aux1=OID.substring(0,OID.lastIndexOf("."));
765                SQL="SELECT assigningAuthorityName FROM
766                AssigningAuthorityName "+"WHERE root=""+aux1+""";
767                Answer=DBM.Ask(SQL);
768                OID=aux1;
769                try{
770                   if(Answer.next()){
771                       aux2=Answer.getString(1);
772                       aux2=aux2+" "+name;
773                       name=aux2;
774                   }
775                   else{
776                       aux2="Unknown";
777                       aux2=aux2+" "+name;
778                       name=aux2;
779                   }
780                 }catch (SQLException e){
781                     System.out.print(e);
782                 }
783
784             }
785             else
786                 end=true;
787         }
788         return name;
789     }
790
791 }
792
```

Clase Comprueba.java

```
 1 /* Clase Comprueba.java
 2  *
 3  * Contiene los métodos para hacer las comprobaciones
 4  * necesarias para el correcto funcionamiento de
 5  * la gestión de IDs
 6 */
 7 package PID.IdGest;
 8
 9 import PID.DataBase.*;
10 import PID.BasicTypes.*;
11 import PID.CAG_GPIC.*;
12 import java.sql.*;
13 import java.util.*;
14
15 public class Comprueba{
16
17     DB_Manager DBM=null;
18
19     /**
20      * Constructor de la clase
21      * @param DB Manejador de la Base de Datos
22      */
23     public Comprueba( DB_Manager DB){
24          DBM=DB;
25         }
26
27     /**
28      * Método que comprueba si el Id que se pasa está ya en la BBDD a
29      * partir de su extensión y el OID de la entidad asignadora
30      * @param extension Extension del identificador
31      * @param root OID de la Entidad Asignadora
32      * @return true si el Id esta registrado
33      * @exception InvalidAAN
34      */
35     public boolean IdExist(String extension,OID root) throws InvalidAAN{
36         Boolean existe=false;
37         Boolean AAN=false;
38         ResultSet Answer=null;
39         String SQL=null;
40         //Comprobamos si el identificador administrativo está registrado
41     SQL="SELECT extension FROM AdministrativeIdentifiers WHERE
42                 (extension=""+extension+"" AND root=""+root+"")";
43         AAN=AAN(root);    // Comprobamos que la AAN está registrada
44         if(AAN)
45         {                     // Comprobamos que el ID está registrado
46             Answer=DBM.Ask(SQL);
47             String aux="";
48             try{
49                     if(Answer.next())
50                         existe=true;
51                     }catch(SQLException error){
52                         System.out.print(error);
53                 }
54         }
55        return existe;
56     }
57
```

```
58      /**
59       * Comprueba si un Identificador está en la BBDD a partir de su
60       * extensión y el nombre de la Entidad Asignadora.
61       * @param extension Extension del identificador a comprobar
62       * @param AAN Nombre de la Entidad Asignadora
63       * @return true si existe.
64       * @exception InvalidAAName
65       */
66      public boolean IdExist(String extension, String AAN) throws
67            InvalidAAName,DuplicateAAN,InvalidAAN{
68          Boolean existe=false;
69          OID root=null;
70       root=AAN(AAN); // Comprobamos que la AAN existe y no hay ambigüedad
71          if(root!=null)
72              existe=IdExist(extension,root);
73          return existe;
74      }
75
76
77      /**
78       * Comprueba si un PersonId está en la BBDD a partir de su extensión.
79       * @param extension Extension del identificador
80       * @return True si existe.
81       */
82      public boolean IdExist(String extension) {
83          boolean existe=false;
84          Boolean AAN=false;
85          ResultSet Answer=null;
86          String SQL=null;
87          //Comprobamos si el identificador administrativo está registrado
88          SQL="SELECT extension FROM PersonIdentifiers WHERE extension=""
89                  +extension+""" ;
90          // Comprobamos que el ID está registrado
91              Answer=DBM.Ask(SQL);
92              String aux="";
93              try{
94                      if(Answer.next())
95                          existe=true;
96                  }catch(SQLException error){
97                      System.out.print(error);
98                  }
99         return existe;
100     }
101
102     /**
103      * Comprueba si una Entidad Asignadora está ya presente en la BBDD a
104      * partir de su OID.
105      * @param root OID de la Entidad a comprobar.
106      * @return true si la Entidad Asignadora está registrada
107      * @exception InvalidAAN
108      */
109     public boolean AAN (OID root) throws InvalidAAN{
110         boolean existe=false;
111         ResultSet Answer=null;
112         String SQL;
113    SQL="SELECT root FROM AssigningAuthorityName WHERE (root=""+root+"")";
114         Answer=DBM.Ask(SQL);
115         try{
116                 if(Answer.next())
117                     existe=true;
118                 }catch(SQLException error){
```

```
119                     System.out.print(error);
120                 }
121          if(!existe){
122             throw new InvalidAAN(root);
123          }
124         return existe;
125    }
126
127      /**
128       * Comprueba si una Entidad Asignadora está ya presente en la BBDD a
129       * partir de su Nombre. Si existen varias Entidades con el mismo
130       * nombre lanza la excepción correspondiente.
131       * @param AAN
132       * @return root: Devuelve el OID de la AAN
133       * @exception DuplicateAAN,InvalidAAName
134       */
135  public OID AAN (String AAN) throws DuplicateAAN,InvalidAAName{
136
137  OID root=null;
138  ResultSet Answer=null;
139  ResultSet Answer1=null;
140  String aux="Nombre de la Entidad Asignadora "+AAN+" está duplicado, \n";
141  int i=0;
142  String SQL;
143  SQL="SELECT root,AssigningAuthorityName FROM AssigningAuthorityName " +
144                  "WHERE (AssigningAuthorityName=""+AAN+"")";
145  Answer=DBM.Ask(SQL);
146  String aux1="";
147  String aux2="";
148         try{
149            while(Answer.next()){
150                 aux2=Answer.getObject(1).toString();
151                 aux1=aux2.substring(0,aux2.lastIndexOf("."));
152                 aux+="La Entidad con OID: "+aux2+" ";
153                 //si solo hay una AAN
154                 if(i==0){
155                     root=new OID(aux2);
156                 SQL="SELECT AssigningAuthorityName,root FROM " +
157                   "AssigningAuthorityName WHERE (root=""+aux1+"")";
158                     Answer1=DBM.Ask(SQL);
159                     if(Answer1.next())
160                   aux+="Pertenece a : "+Answer1.getObject(1).toString()+
161                             " \n";
162                     }
163                 //si no es único
164                 if(i>0){
165                     root=null;
166                     SQL="SELECT AssigningAuthorityName,root FROM " +
167                     "AssigningAuthorityName WHERE (root=""+aux1+"")";
168                     Answer1=DBM.Ask(SQL);
169                     Answer1.next();
170                   aux+="Pertenece a : "+Answer1.getObject(1).toString()+
171                             " \n";
172                     }
173                 i++;
174            }
175         }catch(SQLException error){
176                     System.out.print(error);
177                     }
```

```
178         // Si no existe la Entidad con ese nombre
179         if(i==0){
180             root=null;
181             throw new InvalidAAName(AAN);
182         }
183         // Si ha habido mas de una AAN -> Excepcion InvalidAAN
184         if(i>1)
185           throw new DuplicateAAN(aux);
186         return root;
187     }
188 
189     /**
190      * Comprueba si el campo low de un IVL tiene fecha anterior al high
191      * @param IVL: IVL a comprobar.
192      * @return error:  false si no es cierto.
193      * @exception InvalidIVL
194      */
195     public boolean IVL( IVL_TS IVL) throws InvalidIVL{
196         boolean isOK=true;
197         IVL_TS Intvl=IVL;
198         if(Intvl.getLow().after(Intvl.getHigh())){
199             isOK=false;
200             throw new InvalidIVL(IVL);
201         }
202         return isOK;
203     }
204 }
205 
206 
```

Clase IISeq.java

```
 1 package PID.IdGest;
 2 import PID.BasicTypes.*;
 3 
 4 /**
 5  * Clase que representa una secuencia de II
 6  * @author  Antonio Salado Manzorro
 7  */
 8 public class IISeq {
 9     private II seq[];
10     /** Creates a new instance of IISeq */
11     public IISeq(II[] seqValue) {
12         seq=seqValue;
13     }
14     /**
15      * Establece el valor de la secuencia de II
16      * @param ids Secuencia de II
17      */
18     public void SetIISeq(II[] ids) {
19         seq=ids;
20     }
21 
22     /**
23      * Acceso a la secuencia de IdInfo
24      * @return Secuencia de IdInfo
25      */
26     public II[] GetIISeq() {
27         return this.seq;
28     }
```

```
29
30      public String toString(){
31          String aux="";
32          for(int i=0;i<seq.length;i++)
33              aux=aux+seq[i].toString();
34          return aux;
35      }
36
37  }
38
39
```

Clase AuthorityId.java

```
 1  /*
 2   * AuthorityId.java
 3   * clase que representa el tipo AuthorityId del estándar PIDS de Corba
 4   */
 5
 6  package PID.IdGest;
 7
 8
 9  public class AuthorityId {
10
11      RegistrationAuthority auth=null;
12      NamingEntity   naming=null;
13
14      /** Constructores de la clase */
15      public AuthorityId() {
16      }
17      /**
18       *
19       * @param auth_param Autoridad asignadora
20       * @param naming_param Nombre de la entidad asignadora
21       *  de acuerdo al formato especificado.
22       */
23      public AuthorityId(RegistrationAuthority auth_param,
24              NamingEntity naming_param){
25          auth=auth_param;
26          naming=naming_param;
27      }
28
29      public String toString(){
30   String aux=" Registration authority: "+auth+"\n  Naming entity: "+naming;
31          return aux;
32      }
33
34  }
35
```

Clase Data_Value.java.java

```
 1  /*
 2   * Data_Value.java
 3   *
 4   * Clase que implementa la clase abstracta DataValue del OpenEHR.
 5   *
 6   */
 7
 8  package PID.IdGest;
 9
```

```
10 /**
11  *
12  * @author Antonio
13  */
14 abstract class Data_Value {
15
16     /**
17      * Crea una nueva instancia de <B>DataValue</B>.
18      */
19     Data_Value() {
20     }
21
22 }
23
```

Clase DV_IDENTIFIER.java

```
 1 /*
 2  * DV_IDENTIFIER.java
 3  *
 4  * Clase que representa el tipo DV_IDENTIFIER del Estándar OpenEHR
 5  *
 6  *
 7  */
 8
 9 package PID.IdGest;
10
11 public class DV_IDENTIFIER extends Data_Value {
12
13     // Entidad que genera el identificador
14     private String issuer="";
15     // Entidad que asigna el identificador
16     private String assigner="";
17     // Valor del identificador
18     private String id="";
19     // Tipo (opcional)
20     private String type="";
21
22
23     /** Creates a new instance of DV_IDENTIFIER */
24     public DV_IDENTIFIER() {
25     }
26
27     /**
28      *
29      * @param issuer_param Entidad que genera el Id
30      * @param assigner_param Entidad que asigna el Id
31      * @param id_param Identificador
32      * @param type_param Tipo
33      */
34     public DV_IDENTIFIER(String issuer_param, String assigner_param,
35     String id_param, String type_param){
36      issuer=issuer_param;
37      assigner=assigner_param;
38      id=id_param;
39      type=type_param;
40     }
41
42
```

```
43      /**
44       * Metodo para cambiar los valores de los campos del DV_Identifier
45       * @param issuer_param Entidad que genera el Id
46       * @param assigner_param Entidad que asigna el Id
47       * @param id_param Identificador
48       * @param type_param Tipo
49       */
50      public void setDV_IDENTIFIER(String issuer_param, String
        assigner_param, String id_param, String type_param){
51          issuer=issuer_param;
52          assigner=assigner_param;
53          id=id_param;
54          type=type_param;
55      }
56
57      public String toString(){
58       String aux="Issuer: "+issuer+" assigner: "+assigner+" id:"+id+"
59          type:"+type;
60      return aux;
61      }
62
63
64
65
66  }
67
```

Clase DomainName.java

```
 1  /*
 2   * DomainName.java
 3   * Clase que implementa el tipo de dato DomainName del Estándar PIDS
 4   * de CorbaMed
 5   */
 6
 7  package PID.IdGest;
 8
 9  /**
10   *
11   * @author Antonio
12   */
13  public class DomainName {
14
15      AuthorityId domain=null;
16
17      public DomainName() {
18      }
19      public DomainName(RegistrationAuthority auth_param,
20              NamingEntity naming_param){
21          domain=new AuthorityId(auth_param, naming_param);
22      }
23
24      public String toString(){
25          String aux=domain.toString();
26          return aux;
27      }
28  }
29
```

Clase NamingEntity.java

```
 1 /*
 2  * Clase NamingEntity.java
 3  *
 4  * Implementa el tipo de datos NamingEntity del PIDS de CorbaMed
 5  */
 6
 7 package PID.IdGest;
 8
 9
10 public class NamingEntity {
11
12     String entity=null;
13
14     public NamingEntity() {
15     }
16     /**
17      * Constructor de la clase
18      * @param naming_param Nombre de la entidad
19      */
20     public NamingEntity(String naming_param){
21         entity=naming_param;
22     }
23     public String getNamingEntity(){
24         return entity;
25     }
26
27     public String toString(){
28         return entity;
29     }
30
31 }
32
```

Clase QualifiedPersonId.java

```
 1 /*
 2  * QualifiedPersonId.java
 3  * Implementa el tipo de datos QualifiedPersonId del PIDS de CorbaMed
 4  *
 5  */
 6
 7 package PID.IdGest;
 8 import PID.ServiceClasses.*;
 9
10
11 public class QualifiedPersonId {
12
13     DomainName domain=null;
14     PersonId id=null;
15
16
17
18     public QualifiedPersonId() {
19     }
20
```

```
21      /**
22       * Constructor de la clase
23       * @param domain_param Parámetro DomainName
24       * @param id_param PersonId correspondiente
25       */
26      public QualifiedPersonId(DomainName domain_param, PersonId id_param){
27          domain=domain_param;
28          id=id_param;
29      }
30
31      /**
32       * Establece los valores del Nombre del dominio y del identificador.
33       * @param domain_param Nombre del dominio
34       * @param id_param PersonId
35       */
36     public void setQualifiedPersonId(DomainName domain_param,PersonId
37         id_param){
38          domain=domain_param;
39          id=id_param;
40      }
41
42      /**
43       *
44       * @return Descripcion del identificador
45       */
46      public String toString(){
47      String aux="Dominio: "+ domain.toString()+"\n Identificador:
48          "+id.toString();
49      return aux;}
50
51
52
53 }
54
```

Clase registrationAuthority.java

```
 1 /*
 2  * RegistrationAuthority.java
 3  * Clase que implementa el tipo RegistrationAuthority del estándar PIDS
 4  * de CorbaMed
 5  */
 6
 7 package PID.IdGest;
 8
 9 /**
10  *
11  * @author Antonio
12  */
13 public class RegistrationAuthority {
14
15
16      private String other="OTHER";
17      private String iso="ISO";
18      private String dns="DNS";
19      private String idl="IDL";
20      private String dce="DCE";
21      private int value;
22
23
```

```
24      public RegistrationAuthority(){
25          value=0;
26      }
27
28      public RegistrationAuthority(String auth){
29          if(auth=="OTHER")
30              value=1;
31          else if(auth=="ISO")
32              value=2;
33          else if(auth=="DNS")
34              value=3;
35          else if(auth=="IDL")
36              value=4;
37          else if(auth=="DCE")
38              value=5;
39          else
40              System.out.println("Valor invalido: "+auth);
41      }
42
43      /**
44       * Funcion para obtener el nombre de la entidad que genera el
45       * identificador (tipo de identificador)
46       * @return Devuelve un String con el nombre
47       */
48      public String GetAuth(){
49          String auth="";
50          switch(value){
51              case 1: auth=other;
52                  break;
53              case 2: auth=iso;
54                  break;
55              case 3: auth=dns;
56                  break;
57              case 4: auth=idl;
58                  break;
59              case 5: auth=dce;
60                  break;
61              }
62           return auth;
63      }
64
65
66
67      public String toString(){
68          String auth=null;
69          if(value==1)
70              auth="OTHER";
71          else if(value==2)
72              auth="ISO";
73          else if(value==3)
74              auth="DNS";
75          else if(value==4)
76              auth="IDL";
77          else if(value==5)
78              auth="DCE";
79
80          return auth;
81      }
82  }
83
```

Clase DuplicateAAN.java

```
 1 /**
 2  * Clase que representa la Excepcion DuplicateAAN
 3  * Se lanza cuando existen más de una entidad asignadora con el mismo
 4  * nombre
 5 */
 6
 7 package PID.IdGest;
 8 import PID.BasicTypes.*;
 9
10 public class DuplicateAAN extends Exception{
11
12     String error;
13
14     /**
15      * Crea una nueva instancia de <B>DuplicateAAN</B>.
16      * @param duplicateValue Valor que toma duplicate.
17      */
18     public DuplicateAAN(String e) {
19         error=e;
20     }
21
22     /*
23      * @return Devuelve una cadena con el valor del objeto.
24      */
25     public String toString(){
26         return error;
27     }
28 }
29
```

Clase InvalidAAN.java

```
 1 /**
 2  * Clase InvalidAAN.java
 3  *
 4  * Implementa la excepcion InvalidAAN
 5  * Se lanza cuando la entidad asignadora no está registrada
 6  */
 7
 8
 9 package PID.IdGest;
10 import PID.BasicTypes.*;
11
12 public class InvalidAAN extends Exception{
13     OID error=null;
14
15     /**
16      * Crea una nueva instancia de <B>InvalidAAN</B>.
17      * @param e OID de la entidad asignadora.
18      */
19     public InvalidAAN(OID e) {
20         error=e;
21     }
22
23
24     /**
```

```
25      * @return Devuelve una cadena con el valor del objeto.
26      */
27     public String toString(){
28         return "Invalid AAN: "+ error.toString();
29     }
30 }
31
```

Clase InvalidAAName.java

```
 1 /**
 2  * Clase InvalidAAName.java
 3  *
 4  * Implementa la excepcion InvalidAAName
 5  * Se lanza cuando el nombre indicado para una entidad asignadora no es
 6  * válido.
 7 */
 8 package PID.IdGest;
 9 import PID.BasicTypes.*;
10
11 public class InvalidAAName extends Exception{
12     String error=null;
13
14     /**
15      * Crea una nueva instancia de <B>InvalidAAName</B>.
16      * @param duplicateValue Valor que toma duplicate.
17      */
18     public InvalidAAName(String e) {
19         error=e;
20     }
21
22
23     /**
24
25      * @return Devuelve una cadena con el valor del objeto.
26      */
27     public String toString(){
28         return "Invalid AAN: "+error;
29     }
30 }
31
```

Clase InvalidId.java

```
 1 /**
 2  * Clase InvalidId.java
 3  *
 4  * Implementa la excepcion InvalidId
 5  * Se lanza cuando el identificador que se ha pasado no es válido.
 6  *
 7  */
 8
 9 package PID.IdGest;
10 import PID.BasicTypes.*;
11
12 public class InvalidId extends Exception{
13
14     private String extension=null;
15     private OID root=null;
```

```
16
17      /**
18       * Crea una nueva instancia de <B>InvalidId</B>.
19       * @param extensionValue Valor de la extensionn del identificador.
20       * @param rootValue Valor que del OID de la entidad asignadora del
21       * identificador
22       */
23      public InvalidId(String extensionValue,OID rootValue) {
24          root=rootValue;
25          extension=extensionValue;
26      }
27
28      /**
29      * @return Devuelve una cadena con el valor del objeto.
30       */
31      public String toString(){
32          return "El Id no es válido: extension: "+extension+" root:"+root;
33      }
34  }
35
```

Clase InvalidIVL.java

```
 1  /**
 2   * Clase InvalidIVL.java
 3   *
 4   * Implementa la excepcion InvalidIVL
 5   * Se lanza si en el IVL especificado la fecha final
 6   * es menor que la inicial.
 7   */
 8
 9  package PID.IdGest;
10  import PID.BasicTypes.*;
11
12  public class InvalidIVL extends Exception{
13
14      private IVL_TS IVL=null;
15
16      /**
17       * Crea una nueva instancia de <B>InvalidIVL</B>.
18       * @param IVLValue Valor del periodo de validez.
19       */
20      public InvalidIVL(IVL_TS IVLValue) {
21          IVL=IVLValue;
22      }
23
24      /**
25       * @return Devuelve una cadena con el valor del objeto.
26       */
27      public String toString(){
28          return "InvalidIVL: "+IVL.toString();
29      }
30  }
31
```