Capítulo 6

JAVA como herramienta de programación para la aplicación cliente

[22]

6.1. Introducción

La aplicación cliente de nuestro proyecto sigue todos los pasos del estándar DICOM en la implementación de las funciones. Se ha querido realizar un programa multiplataforma y que esté de acuerdo con las especificaciones requeridas.

Este programa ha sido elaborado en Java en el entorno JBuilder con soporte de librerías SDK 1.4.2 y JDT de DICOM. A continuación, vamos a exponer algunas de las características de Java para comprender los motivos de la elección de este lenguaje de programación.

6.2. Aspectos generales de la programación orientada a objetos

Clases y objetos

Una *clase* es un modelo abstracto de un tipo de objeto, define sus métodos y atributos. Un *objeto* es una instancia de una clase, es decir, la implementación con valores de un modelo abstracto.

Las clases no son entidades independientes, sino que se agrupan jerárquicamente heredando características y atributos. Cada instancia o implementación real de una clase constituirá un nuevo objeto, por lo que se pueden crear infinitos objetos distintos a partir de una sola clase.

Encapsulamiento

Podemos definir la *encapsulación* como el proceso de empaquetar juntos los métodos y los datos de un objeto. Con esto se consigue un mayor nivel de seguridad en el acceso a los datos de un objeto y permite abstraer los detalles internos de funcionamiento del objeto. El objeto se encarga de ocultar sus datos al resto de objetos y proporciona métodos para modificarlos y obtener sus valores, por lo que el acceso depende directamente de cada objeto.

Intercambio de mensajes

Los objetos se comunican entre sí mediante mensajes de invocación a métodos, que son funciones implementadas dentro de las clases.

Herencia

Es el concepto que define la adopción de todas las características de una clase por parte de otra clase que es definida como descendiente o heredera de la primera. La

principal consecuencia de la herencia es la posibilidad de reutilizar clases, ya que se pueden crear nuevas a partir de las ya creadas. La herencia puede ser de dos tipos:

- Simple, si sólo es posible heredar características de una sola clase.
- Múltiple, si se pueden heredar características de varias clases.

Polimorfismo

El concepto de polimorfismo define la posibilidad de crear una instancia de una clase y utilizarla como si fuese una instancia de una clase superior jerárquicamente.

6.3. Programación en JAVA

Java surgió en 1991 cuando un grupo de ingenieros de *Sun Microsystems* trataron de diseñar un nuevo lenguaje de programación destinado a electrodomésticos. Debido a la existencia de distintos tipos de CPUs y a los continuos cambios, era importante conseguir una herramienta independiente del tipo de CPU utilizada. Se ejecuta sobre una "máquina hipotética o virtual" denominada *Java Virtual Machine* (JVM). Es la JVM quien interpreta el código neutro convirtiéndolo a código particular de la CPU utilizada. Cualquier aplicación que se desarrolle se apoya en un gran número de clases preexistentes (el API o *Application Programming Interface* de Java).

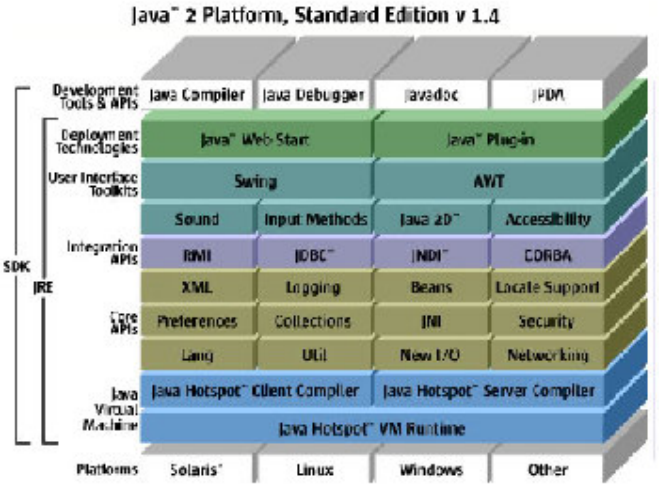

Figura 6.1. Evolución de Java

La ejecución de programas en Java tiene muchas posibilidades: ejecución como aplicación independiente (*Stand-alone Application*), ejecución como *applet*, ejecución como *servlet*, etc. Un applet es una aplicación especial que se ejecuta dentro de un navegador o *browser* (por ejemplo *Netscape Navigator*, *Apache* o *Internet Explorer*) al cargar una página HTML desde un servidor Web. El applet se descarga desde el servidor y no requiere instalación en el ordenador donde se encuentra el browser. Un servlet es una aplicación sin interfaz gráfica que se ejecuta en un servidor de Internet.

Java permite fácilmente el desarrollo tanto de arquitecturas cliente-servidor como de aplicaciones distribuidas, consistentes en crear aplicaciones capaces de conectarse a otros ordenadores y ejecutar tareas en varios ordenadores simultáneamente.

6.3.1. Características básicas de Java

Las características principales que nos ofrece Java respecto a cualquier otro lenguaje de programación son:

Simple

Java ofrece toda la funcionalidad de un lenguaje potente, pero sin las características menos usadas y más confusas de éstos. El lenguaje C/C++ muestra una falta de seguridad, pero estos lenguajes están más difundidos, por ello Java se diseñó para ser parecido a C++ y así proporcionar un rápido y fácil aprendizaje.

Java elimina muchas de las características de otros lenguajes como C++ para mantener reducidas las especificaciones del lenguaje y añadir características muy útiles como el *garbage collector* (reciclador de memoria dinámica). No es necesario preocuparse por liberar memoria, el reciclador se encarga de ello y como es un *thread* de baja prioridad (un programa que se ejecuta concurrentemente), cuando entra en acción, permite liberar bloques de memoria muy grandes, lo que reduce la fragmentación de la memoria.

Además, Java reduce en un 50% los errores más comunes de programación con lenguajes como C y C++, al eliminar muchas de las características de éstos como son la aritmética de punteros, las referencias, los registros (*struct*), las macros…

Orientado a objetos

Java implementa como ya hemos dicho la tecnología básica de C++ con algunas mejoras y elimina algunas cosas para mantener el objetivo de simplicidad del lenguaje. Java trabaja con sus datos como objetos y con interfaces a esos objetos. Soporta las tres características fundamentales de la orientación a objetos: encapsulación, herencia y polimorfismo. Las plantillas de objetos son llamadas, como en C++, clases y sus copias, instancias. Estas instancias, como en C++, necesitan ser construidas y destruidas en espacios de memoria.

Java incorpora funcionalidades inexistentes en C++, como la resolución dinámica de métodos. Esta característica deriva del lenguaje *Objective C*, propietario del sistema operativo *Next*. En C++ se suele trabajar con librerías dinámicas (DLLs) que obligan a recompilar la aplicación cuando se retocan funciones contenidas en su interior. Este inconveniente es resuelto por Java mediante una interfaz específica llamada RTTI (*Run Time Type Identification*) que define la interacción entre objetos excluyendo variables de instancias o implementación de métodos. Las clases de Java tienen una representación en el run-time que permite a los programadores interpretar el tipo de clase y enlazar dinámicamente la clase con el resultado de la búsqueda.

Distribuido

Java se ha construido con extensas capacidades de interconexión TCP/IP. Existen librerías de rutinas para acceder e interactuar con protocolos como *http* y *ftp*. Esto permite a los programadores acceder a la información a través de la red con tanta facilidad como a los ficheros locales.

Lo cierto es que Java en sí no es distribuido, sino que proporciona las librerías y herramientas necesarias para que los programas puedan ser distribuidos, es decir, que se ejecuten en varias máquinas, interactuando.

Robusto

Java realiza verificaciones en busca de problemas tanto en tiempo de compilación como en tiempo de ejecución. La comprobación de tipos en Java ayuda a detectar errores, lo antes posible, en el ciclo de desarrollo. Java obliga a la declaración explícita de métodos, reduciendo así las posibilidades de error. Maneja la memoria para eliminar las preocupaciones por parte del programador de la liberación o corrupción de la misma.

También implementa los *arrays* auténticos, en vez de listas enlazadas de punteros, con comprobación de límites, para evitar la posibilidad de sobrescribir o corromper memoria resultado de punteros que señalan a zonas equivocadas. Estas características reducen drásticamente el tiempo de desarrollo de aplicaciones en Java.

Además, para asegurar el funcionamiento de la aplicación, realiza una verificación de los *byte-codes*, que son el resultado de la compilación de un programa Java. Es un código de máquina virtual que es interpretado por el intérprete Java. No es el código máquina directamente entendible por el hardware, pero ya ha pasado todas las fases del compilador: análisis de instrucciones, orden de operadores, etc., y ya tiene generada la pila de ejecución de órdenes.

Por tanto, Java proporciona:

- Comprobación de punteros.
- Comprobación de límites de arrays.
- Excepciones.
- Verificación de los byte-codes.

Arquitectura neutral

Para establecer Java como parte integral de la red, el compilador Java compila su código a un fichero objeto de formato independiente de la arquitectura de la máquina en que se ejecutará. Por este motivo se dice que Java es un lenguaje multiplataforma, ya que cualquier máquina que tenga el sistema de ejecución (run-time) podrá ejecutar dicho código objeto, sin importar en modo alguno la máquina en la que ha sido generado. Actualmente existen sistemas run-time para Windows, Solares 2.x, Linux, SunOs 4.1.x, Irix, Aix, Mac, Apple…

Seguro

La seguridad en Java tiene dos facetas. Si otros programadores conocen nuestras estructuras de datos, podrían extraer información confidencial de nuestro sistema. En Java, características como los punteros o el casting implícito que hacen los compiladores de C y C++ se eliminan para prevenir el acceso ilegal a la memoria.

Otro tipo de ataque es el Caballo de Troya, en el que se presenta un programa como una utilidad, resultando tener una funcionalidad destructiva. El código Java pasa muchos tests antes de ejecutarse en una máquina. Como hemos dicho antes, se pasa a través de un verificador de byte-codes que comprueba el formato con los fragmentos de código y aplica un probador de teoremas para detectar fragmentos de código ilegal, código que falsea punteros, viola derechos de acceso sobre objetos o intenta cambiar el tipo o clase de un objeto. Si los byte-codes pasan la verificación sin generar ningún mensaje de error, entonces sabemos que:

- El código no produce desbordamiento de operandos en la pila.
- El tipo de los parámetros de todos los códigos de operación son conocidos y correctos.
- No ha ocurrido ninguna conversión ilegal de datos, tal como convertir enteros en punteros.
- El acceso a los campos de un objeto se sabe que es legal: public, private, protected.

- No hay ningún intento de violar las reglas de acceso y seguridad establecidas.

El Cargador de Clases también ayuda a Java a mantener su seguridad, separando el espacio de nombres del sistema de ficheros local del de los recursos procedentes de la red. Esto limita cualquier aplicación del tipo Caballo de Troya, ya que las clases se buscan primero entre las locales y luego entre las procedentes del exterior.

Las clases importadas de la red se almacenan en un espacio de nombres privado, asociado con el origen. Cuando una clase del espacio de nombres privado accede a otra clase, primero se busca en las clases predefinidas (del sistema local) y luego en el espacio de nombres de la clase que hace la referencia. Esto imposibilita que una clase suplante a una predefinida.

En resumen, las aplicaciones de Java resultan extremadamente seguras, ya que no acceden a zonas delicadas de memoria o de sistema, con lo que evitan la interacción de ciertos virus. Java no posee una semántica específica para modificar la pila de programa, la memoria libre o utilizar objetos y métodos de un programa sin los privilegios del *kernel* (núcleo) del sistema operativo. Además, para evitar modificaciones por parte de los *crackers* de la red, implementa un método ultraseguro de autentificación por clave pública. El Cargador de Clases puede verificar una firma digital antes de realizar una instancia de un objeto. Por tanto, ningún objeto se crea y almacena en memoria sin que se validen los privilegios de acceso. Es decir, la seguridad se integra en el momento de compilación, con el nivel de detalle y de privilegio que sea necesario.

Portable

Más allá de la portabilidad básica por ser multiplataforma, Java implementa otros estándares de portabilidad para facilitar el desarrollo. Los enteros son siempre enteros y además, enteros de 32 bits en complemento a 2. También debemos señalar que Java construye sus interfaces de usuario a través de un sistema abstracto de ventanas de forma que las ventanas puedan ser implantadas en entornos Unix, Pc o Mac.

Interpretado

El intérprete Java (sistema run-time) puede ejecutar directamente el código objeto. Enlazar (linkar) un programa normalmente consume menos recursos que compilarlo, por lo que los desarrolladores con Java pasarán más tiempo desarrollando y menos esperando por el ordenador. No obstante, el compilador actual del JDK es un poco lento, pero esta desventaja se va resolviendo poco a poco a medida que aparecen nuevas versiones del mismo.

Multithread

Al ser *multithread* (multihilo), Java permite muchas actividades simultáneas en un programa. Los *threads* (a veces llamados procesos ligeros o *hilos de ejecución*), son básicamente pequeños procesos o piezas independientes de un gran proceso. Al estar los threads construidos en el lenguaje, son más fáciles de usar y más robustos que sus homólogos en C o C++.

El beneficio de ser multithread se refleja en un mejor rendimiento interactivo y mejor comportamiento en tiempo real. Aunque el comportamiento en tiempo real está limitado a las capacidades del sistema operativo sobre el que se apoye (Unix, Windows…), aún supera a los entornos de flujo único de programa (*single-threaded*) tanto en facilidad de desarrollo como en rendimiento.

Dinámico

Java se beneficia todo lo posible de la tecnología orientada a objetos. Java no intenta conectar todos los módulos que comprenden una aplicación hasta el tiempo de ejecución. Las librerías nuevas o actualizadas no paralizarán las aplicaciones anteriores (siempre que mantengan el API anterior).

Java también simplifica el uso de protocolos nuevos o actualizados. Si un sistema ejecuta una aplicación Java sobre la red y encuentra una pieza de la aplicación

que no sabe manejar, tal como se ha explicado en párrafos anteriores, Java es capaz de traer automáticamente cualquiera de esas piezas que el sistema necesita para funcionar.

Para evitar que haya que estar trayendo de la red los módulos de byte-codes o los objetos o nuevas clases cada vez que se necesiten, Java implementa las opciones de persistencia, de modo que no se eliminen cada vez que se limpie la caché de la máquina.

En definitiva, Java es un lenguaje de programación orientado a objetos, interpretado, independiente de la arquitectura y portable, fuertemente tipificado, con gestión automática de la memoria (gracias al *garbage collector* o recogedor de basura), con gestión de excepciones y concurrencia (multihilo) y principalmente con las características de encapsulación, herencia de clases y polimorfismo.

El lenguaje Java es un lenguaje sencillo extendido mediante una serie de bibliotecas o *packages* (paquetes), entre ellos los más comunes son:

- Package *lang*: clase con funcionalidades básicas. E/S, excepciones, hilos.
- Package *util*: Utilidades (números aleatorios, vectores).
- Package *net*: Conectividad y trabajo con redes.
- Package *applet*: Desarrollo de aplicaciones ejecutables en navegadores.
- Package *awt* y *swing*: Desarrollo de interfaces gráficas de usuario.

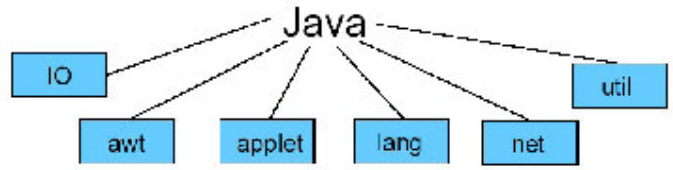

Figura 6.2. Bibliotecas de Java

6.3.2. Instalación de la JVM

Hay dos instalaciones posibles de la JVM, o bien la instalación del JRE[1], la cual sirve para que se puedan ejecutar las aplicaciones desarrolladas en Java pero no posee

[1] Java Runtime Environment

compilador para los ficheros fuente, o bien la instalación del SDK (*Software Development Kit*), que sí tiene compilador para desarrollar aplicaciones.

Para el desarrollo de aplicaciones en Java, la instalación debe comprender el SDK, que es un software libre y puede ser descargado de http://java.sun.com. Al instalar el SDK, se crea una carpeta en el directorio raíz (j2sdk1.4.2_04)[2] , y dentro de esta hay otra llamada "bin" que es la que contiene el compilador.

El siguiente paso es dar el *PATH* al compilador y crear un *CLASSPATH* que contenga las librerías que vaya a utilizar el compilador. Los pasos a seguir difieren dependiendo del sistema operativo que se utilice.

Los pasos a seguir en un sistema Windows de red (NT, 2000, XP):

1. Pinchar en Inicio → Configuración → Panel de Control → Sistema, ir a la pestaña Opciones Avanzadas y hacer click en Variables de Entorno.

2. En variables del sistema ir a la variable *Path* y escribir seguido a lo que haya el camino a seguir hasta la carpeta *bin* del SDK:

   C:\J2SDK1.4.1_03\BIN;

3. Para crear el *Classpath*, en variables del sistema seleccionar "Nueva. . . " y escribir en nombre de variable:

   CLASSPATH

   y en valor de variable el camino hasta la carpeta *lib* donde está dt.jar que son las clases del JDK:

   C:\j2sdk1.4.1_03\lib\dt.jar;.;

[2] Para la versión 1.4.2

El valor de ;.; es para poder compilar cualquier fichero fuente desde cualquier directorio y no deba ser desde el directorio donde está el compilador.

Los pasos a seguir en un sistema Windows´9X o MSDOS:

1. Abrir el fichero Autoexec.bat que se encuentra en el directorio raíz del sistema, pinchando en él con el botón derecho del ratón y dejando pulsada la tecla Shift, de esta forma sale el menú contextual con la opción "Abrir con. . . ", se pincha esta opción y se elige un editor de texto (por ejemplo el WordPad).

2. Una vez abierto en la línea de *SET PATH* hay que incluir la carpeta del compilador:

   SET PATH = . . . ;C:\J2SDK1.4.1-_03\Bin; %PATH%

3. Crear el *CLASSPATH* y darle la localización de las librerías del JDK, para ello escribir debajo del PATH:

   SET CLASSPATH = C:\j2sdk1.4.1_03\lib\dt.jar;.; %CLASSPATH%

6.3.3. El compilador de JAVA

Es una herramienta del JDK o SDK[3], ya que el JRE por sí solo no compila. Realiza un análisis de la sintaxis del código escrito en los ficheros fuente de Java (con extensión *.java). Si no encuentra errores en el código genera los ficheros compilados (con extensión *.class).En el JDK de Sun dicho compilador se llama *javac.exe*. *Java.exe* es el intérprete para sistemas PC/Windows. *Appletviewer.exe* es un visualizador de applets.

Una vez compilado no debería ser necesaria ninguna modificación por el hecho de cambiar de procesador o de ejecutarlo en otra máquina. La clave para que esto fuese posible consistió en desarrollar un código "neutro" que estuviera preparado para ser ejecutado sobre la JVM.

[3] JDK para las antiguas versiones y SDK para las nuevas versiones

Para realizar una aplicación con el compilador del SDK, se debe escribir el código en Java en un editor de texto cualquiera como puede ser el "Bloc de Notas". Una vez escrito debe ser guardado el fichero con extensión *.java*, por ejemplo nombre.java. La compilación debe realizarse con un "shell" de comandos, como el del MSDOS. Para realizar la compilación escribir:

X:\. . .\javac nombre.java

Con la compilación se genera un archivo *.class*, en el caso de ejemplo sería nombre.class. Después ya se puede ejecutar o interpretar, para esto escribir en el Shell:

X:\. . .\java nombre

Todo ello se realiza en el directorio donde se encuentre el fichero fuente .java. Ver figura 6.3.

Java Sun ha realizado un API para la ayuda y consulta de las clases que se pueden utilizar con el SDK. Esta ayuda es de importancia vital para el buen uso y comprensión de java. Ver http://java.sun.com .

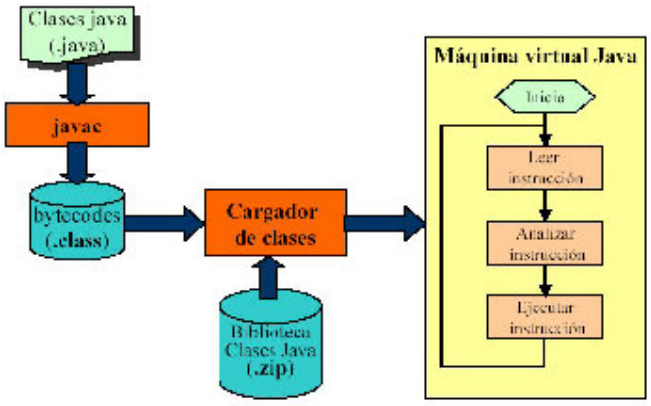

Figura 6.3. Compilación y Ejecución

6.3.4. Javadocs

En el diseño del lenguaje se ha tenido en cuenta la documentación de los programas y el mantenimiento de dicha documentación. La documentación y el código se incluyen dentro del mismo fichero.

El tipo de comentario específico para documentar debe ser:

*/** Comentario de documentación */*

La generación de la documentación se realiza en formato HTML. Se pueden crear etiquetas que luego en la documentación en HTML, saldrán como un pequeño apartado, esto se realiza poniendo @ delante de la línea que queramos que sea una etiqueta.

Los comentarios deben aparecer inmediatamente antes de los elementos a comentar. La utilidad de documentación javadoc es un programa que se suministra dentro de la distribución de J2SE[4].

```
/**
 * La clase ObjetoDicom crea un fichero con formato compatible
 *    al estándar a partir de la imagen de la lesión y los datos
 *    del paciente
 * @author Mª José Aguilar Porro
 * @version 1.0
 */
```

Modo de uso:

Javadoc [opciones] [paquetes] [archivosFuente] [@ficheros]

Hay más de 40 opciones (consultar API) que modifican el funcionamiento de Javadoc.

[4] Java 2 Standard Edition

6.3.5. Creación de ejecutables *.jar

Esta utilidad es usada para generar archivos .jar que contienen todas las clases de la aplicación realizada. Esto se realiza para su posterior distribución. Ejecutando el .jar generado, la Máquina Virtual de Java podrá ejecutar el programa desarrollado.

El formato básico del comando para crear un fichero JAR es:

jar cf fichero-jar fichero(s)-de entrada

Este comando tiene varias opciones y argumentos:

- c - indica que se quiere crear un fichero JAR.
- f - indica que se quiere que la salida vaya a un fichero en vez de a la salida estándar *(stdout*).
- v - produce un salida verbosa en *stderr* (salida estándar para errores, en la versión 1.1) o en stdout (en versión 1.2) mientras se construye el fichero. La salida verbosa da el nombre de cada fichero añadido al fichero JAR.
- 0 - indica que no se quiere que el fichero JAR sea comprimido.
- M - indica que no se debería producir el fichero de manifiesto por defecto.
- m - utilizada para incluir información de manifiesto desde un fichero de manifiesto existente. El formato utilizado por esta opción es: *jar cmf existing-manifest output-file input-file(s)*
- x - indica que se quieren extraer los ficheros de un archivo JAR.
- u - actualiza el fichero JAR existente.
- fichero - file es el nombre que se quiere para el fichero JAR resultante. Se puede utilizar cualquier nombre de fichero. Por convención, a los ficheros JAR se les da la extensión .jar, aunque no es obligatorio.
- El argumento fichero(s)-de entrada es una lista delimitada por espacios de uno o más ficheros que deben ser situados dentro de nuestro fichero JAR. Este argumento puede tener el símbolo del comodín: *. Si alguno de los fichero(s)-de entrada es un directorio, el contenido de dicho directorio se añadirá al fichero JAR recursivamente.

Este comando generará un fichero JAR comprimido y lo situará en el directorio actual. El comando también genera un fichero de manifiesto, por defecto. META-INF / MANIFEST.MF, para el archivo JAR.