Programación Avanzada
UD1

Autor:
P. Pablo Garrido Abenza pgarrido@umh.es

Universidad Miguel Hernández

01 de Octubre de 2012

Copyright ©2012- P. Pablo Garrido Abenza

ÍNDICE GENERAL

Copyright ©2012- P. Pablo Garrido Abenza

PRESENTACIÓN

Los presentes apuntes pretenden ser una ayuda en el aprendizaje del lenguaje de programación Java, cubriendo gran parte de los conocimientos que son necesarios para el desarrollo de programas de consola (modo texto), aplicaciones gráficas, y *applets* (aplicaciones descargables desde Internet).

La tecnología Java [1] es tanto un lenguaje de programación como un conjunto plataformas especializadas: J2SE (Java SE), J2EE (Java EE), J2ME (Java ME), y JavaCard. En estos apuntes se cubre la plataforma J2SE (Java SE), es decir, la Java Standard Edition.

Copyright ©2012- P. Pablo Garrido Abenza

[1] La marca Java y los distintos logotipos de Java son marcas comerciales o registradas de Sun Microsystems, Inc.

CAPÍTULO 1

INTRODUCCIÓN

Copyright © 2012 P. Pablo Garrido Abenza

1.1. El lenguaje de programación Java

En el documento (*White paper*) The Java Language Environment [**?**] se detallan los objetivos perseguidos al comienzo del desarrollo del proyecto del lenguaje Java, y se muestran sus características.

Se necesitaba alguna tecnología que permitiese el desarrollo de programas capaces de ejecutarse en entornos distribuidos y heterogéneos, es decir, programas ejecutables en muchas plataformas hardware y software, principalmente dispositivos electrónicos embebidos. La idea inicial fue el desarrollo de un sistema operativo portable que requiriese pocos recursos, en tiempo-real y distribuido, pero finalmente se obtuvo un lenguaje de programación.

Con el crecimiento masivo de Internet (red heterogénea de ordenadores distribuidos y conectados entre sí) se vio una nueva forma de distribuir los programas. Sin embargo, la forma tradicional de distribuir programas por medio de los archivos binarios resultado de un proceso de compilación no era válida en este caso, debido principalmente a que Internet se trata, como hemos dicho, de un entorno heterogéneo.

El lenguaje Java es un lenguaje de alto nivel que tiene una serie de **características** destacables:

- **Familiar**: el lenguaje Java se desarrolló desde cero como un nuevo lenguaje, pero su sintáxis es muy similar al lenguaje C o C++, por lo que se facilita la migración de aquellos desarrolladores ya familiarizados con dichos lenguajes. Aunque se adoptó la sintáxis, se eliminaron todas aquellas características conflictivas de aquellos lenguajes, como el uso de punteros para el acceso directo a una dirección de la memoria, o la gestión de la memoria dinámica de forma manual.

- **Sencillo**: aunque Java puede tener una curva de aprendizaje dura, el conocer otros lenguajes de programación similares facilita su aprendizaje. Además, lo aprendido sirve para el desarrollo en distintos dispositivos para los que tradicionalmente se utilizaban herramientas y lenguajes muy diferentes entre sí, como era el caso del software de control de los electrodomésticos, electrónica de consumo (televisión, vídeo, ...), o actualmente, teléfonos inteligentes o *smart-phones* (Android, etc.).

- **Multiplataforma**: Java fué diseñado específicamente para ser "*Write Once, Run Anywhere*", es decir, escribir (y compilar) una sola vez en una plataforma, y ejecutar en cualquier otra, sin necesidad de recompilar, ni siquiera tener que modificar el código fuente. En Java, los tipos de datos primitivos siempre ocupan lo mismo, independientemente de la plataforma en la que se ejecuten; con otros lenguajes como C o C++, el tamaño de un `int` (por ejemplo) varía dependiendo de si el programa se compila y ejecuta en Windows o en Linux, y esto puede obligarnos a tener que modificar el código en caso de querer migrar un programa de una plataforma a otra. La portabilidad de los programas Java se consigue porque Java es un lenguaje interpretado, aunque también compilado para mejorar el rendimiento de la interpretación; por ello se dice que Java es mitad compilado, mitad interpretado. Cuando un programa Java se compila, se genera un código intermedio independiente de la plataforma denominado *bytecodes*. Estos *bytecodes* serán los que se distribuyan para su ejecución, siendo interpretados por la Máquina Virtual Java (JVM) específica de la plataforma del usuario.

- **Alto rendimiento**: los programas Java no son tan rápidos como con otros lenguajes que se compilan para una plataforma (hardware + sistema operativo) concreta, ya que son interpretados durante su ejecución. Sin embargo, no son tan lentos como los lenguajes que son exclusivamente interpretados, es decir, aquellos que van interpretando el código fuente línea por línea, comprobando en cada caso la sintáxis. No obstante, si para alguna tarea concreta se requiriese una mayor velocidad, Java puede integrarse con fragmentos de código nativo escrito en C o ensamblador, por ejemplo. Finalmente, comentar que existen compiladores nativos para Java: *Just-In-Time* (JIT) y *Ahead-On-Time* (AOT), que consiguen un rendimiento similar a los lenguajes tradicionales, aunque todo esto sale del ámbito de estos apuntes.

- **Robusto**: durante la compilación de un programa Java se comprueba la sintáxis y ciertas situaciones que en con otros lenguajes compilados no se comprobaban, pudiendo dar lugar a resultados inesperados (variables sin valor inicial, asignaciones entre variables de distinto tipo, etc.). Posteriormente, durante la ejecución se vuelven a realizar otras comprobaciones, por lo que un programa escrito en Java es bastante robusto. Por otra parte, hay ciertas características que en otros lenguajes era el programador el encargado de realizarlas o comprobarlas de forma manual y con Java no, como la gestión de la memoria dinámica que realiza el recolector de basura o *garbage collector*. También hay cosas que su uso puede ser arriesgado que se han eliminado en Java, como el uso de punteros para acceder de forma directa a determinadas direcciones de memoria.

- **Orientado a objetos**: Java sigue este paradigma de programación, que es el utilizado en la actualidad. Esta metodología de programación es muy flexible, pues facilita todo el ciclo de vida del software, desde el análisis y diseño, hasta el mantenimiento, pasando por la implementación propiamente dicha.

- **Distribuido**: Java dispone de una librería de clases que permiten la comunicación entre programas ejecutados en ordenadores remotos conectados en red (distribuido), en un entorno heterogéneo, y de forma segura, esto es, evitando la intrusión por otros programas. El uso de la Programación Orientada a Objetos (POO) es muy apropiado para el envío de mensajes en entornos distribuidos.

- **Concurrente**: Java permite el desarrollo de programas concurrentes o multi-hilo, para conseguir un mejor rendimiento y aprovechamiento del procesador cuando es necesario realizar varias tareas de forma simultánea.

Copyright © 2012 P. Pablo Garrido Abenza

De todo lo anterior podemos sacar como conclusión que Java proporciona una arquitectura muy flexible y robusta para el desarrollo de programas para prácticamente para cualquier situación y plataforma.

1.2. Comparación con otros lenguajes: C / C++

Como se ha comentado, los desarrolladores de Java se inspiraron en el lenguaje C / C++ para su desarrollo, principalmente para facilitar su aprendizaje por parte de quien ya estaba familiarizado con la programación en estos lenguajes. Se adoptaron muchas carcaterísticas, pero se eliminaron aquellas que pudieran ser fuentes de problemas.

Lo siguiente presenta una comparativa resumida entre estos lenguajes, lo cual será de utilidad al lector que ya tenga algo de experiencia en programación utilizando el lenguaje C:

- Tienen una **sintaxis** muy similar: bloques de código entre llaves, el carácter de punto y coma `;` tras cada instrucción, función `main()` como punto de entrada, forma de escribir comentarios, ...
- Los **tipos** de datos primitivos (`int`, `long`, `float`, `double`, `char`) son los mismos que en C, aunque Java incorpora algunos otros (`boolean` o `byte`). Sin embargo, los modificadores de tipo (p.e. `unsigned`) que ofrece C no están disponibles en Java. Una diferencia importante en relación a los tipos es que en Java, los tipos ocupan lo mismo independientemente de la plataforma en la que se ejecute el programa, cosa que en C no ocurre y es algo a tener en cuenta a la hora de migrar un programa hacia otras plataformas. El lenguaje C permite realizar conversiones automáticas entre los distintos tipos, incluso aunque pueda haber pérdida de información por tener un rango distinto; en ocasiones, tal circunstancia se notifica al programador mediante un aviso (*warning*), pero el programa puede compilar y no mantener la integridad de los datos. Java también reliza conversiones de tipo automáticas, pero de forma mucho más restringida: en caso de detectar una posible pérdida de información se genera un error de compilación, debiendose corregir mediante el cambio de tipo de alguna variable, o la realización de conversión de forma explícita (*casting*).
- En Java no existen los **punteros**, es decir, variables que almacenan una dirección de memoria concreta para acceder a ella. Esto es peligroso, ya que puede accederse a una zona de memoria protegida por el sistema, produciendo resultados impredecibles. No obstante, existe el concepto de **referencia**, similar a un puntero pero del que desconocemos su valor. Sin embargo, al no disponer de punteros se impide el paso de valores de tipo primitivo por referencia a un método, es decir, siempre se pasan por valor.
- En Java la **gestión de la memoria dinámica** se realiza de forma automática, liberando al programador de esa tarea. Los objetos se liberan automáticamente cuando ya no se necesitan (cuando la ejecución sale de su ámbito de visibilidad), realizando esta tarea el recolector de basura o *garbage collector*.
- Java no permite el uso de **estructuras** (`struct`) o **uniones** (`union`), aunque esto no es una limitación ya que se dispone del concepto de clase (`class`). Un `struct` es un registro o variable múltiple, es decir, es un conjunto de variables que se gestionan como un todo; por otra parte una clase permite eso (variables de clase) y definir también los métodos (funciones) para acceder a tales datos.
- Java se diseñó para utilizar la **Programación Orientada a Objetos** (POO) y no es posible programar sin ella. Por el contrario, C++ fue una extensión del lenguaje C para dar soporte a esta nueva metodología de programación, y es posible escribir programas mixtos, es decir, módulos que utilicen la POO y

Copyright © 2012 P. Pablo Garrido Abenza

otros que no (programación estructurada). En relación también con la POO, C++ permite utilizar herencia múltiple entre clases, es decir, permite escribir una clase que hereda de dos o más clases; puesto que esto puede generar inconsistencias se ha eliminado en Java, es decir, Java permite unicamente la herencia simple.

- Java no soporta la **sobrecarga de operadores**, esto es, poder redefinir el comportamiento de los típicos operadores aritméticos (p.e. la suma +) para realizar ciertas operaciones sobre algún objeto de una clase. Sin embargo, la palabra que se utiliza en C++ para ello (`operator`) es una palabra reservada en Java, por lo que quizás se implemente esta característica en futuras versiones del lenguaje.
- Java no soporta `typedef` para definir sinónimos de otros tipos o estructuras.
- Java no permite el uso de comandos de preprocesador: `#define`, `#include`, `#ifdef`, etc.

1.3. Software necesario

Para el desarrollo de programas Java, y poder ejecutar los programas de los ejemplos del libro se requiere el siguiente software:

- Java Platform Standard Edition (J2SE o Java SE)
- Entorno de Desarrollo Integrado (IDE): NetBeans, Eclipse, ...

Java Platform Standard Edition (J2SE o Java SE)

Es necesario disponer del kit de desarrollo de Sun Microsystems (ahora Oracle) para poder desarrollar y ejecutar programas utilizando el lenguaje de programación Java. Para ejecutar programas Java tan sólo es necesario el Java Runtime Environment (JRE), pero para su desarrollo es necesario el Java Development Kit (JDK), el cual incluye también el JRE. En algunas plataformas ya viene instalado de serie en el sistema operativo, como es el caso de Linux o Mac OS X.

Podemos descargar la última versión desde el siguiente enlace, disponible para Microsoft Windows, Linux, y Mac OS X: `http://www.oracle.com/technetwork/java/javase/downloads/index.html`.

En caso de utilizar el IDE NetBeans, desde esa misma página podemos descargar en un único instalador tanto el JDK como el NetBeans (JDK + NetBeans bundle), disponible también para las tres plataformas anteriores.

IDE NetBeans

El Entorno de Desarrollo Integrado (IDE) NetBeans es un entorno gratuito para el desarrollo de programas Java estándar (stand-alone) y applets. Además, NetBeans también facilita la tarea del desarrollo de aplicaciones empresariales (Java EE), permitiendo la compilación, empaquetado, distribución y ejecución dentro del mismo entorno, aunque en este documento tan sólo nos ocuparemos de las primeras.

Se puede descargar NetBeans desde el enlace: `http://www.netbeans.org/`.

Copyright © 2012 P. Pablo Garrido Abenza

El programa está disponible para las principales plataformas: Microsoft Windows, Linux, y Mac OS X. Antes de instalar el programa es necesario tener instalado el JDK de la Java Platform Standard Edition (J2SE o Java SE). Se recomienda descargar NetBeans desde la página de Oracle, pues existe un instalador que ya integra tanto el JDK como el IDE NetBeans; ver el apartado anterior.

IDE Eclipse

Eclipse también es un entorno de desarrollo gratuito que da soporte a Java SE y Java EE, así como a otros lenguajes por medio de plug-ins (C/C++, Python, PHP, LaTeX, ...). Al igual que NetBeans, está disponible para Microsoft Windows, Linux, y Mac OS X

Se puede descargar NetBeans desde el enlace: `http://www.eclipse.org/`.

Documentación en línea

La mayoría de los entornos de desarrollo integrados (IDE), como NetBeans o Eclipse que acabamos de mencionar, nos ayudan bastante en la escritura de código, como mostrarnos una lista de sugerencias según el contexto en el que estemos conforme escribimos, como por ejemplo una lista con los métodos (funciones) que se pueden ejecutar con un determinado objeto. No obstante, en ocasiones podremos necesitar consultar la documentación en línea de las distintas clases de Java (API), para averiguar las constantes, campos, constructores disponibles, y métodos, así como consultar una explicación detallada de su funcionamiento, argumentos recibidos, o valor retornado.

La documentación la podemos consultar directamente en Internet en el siguiente enlace (tal como se ve en la figura 1.1), o descargarla en formato HTMLHelp (archivo .chm) o WinHelp (archivo .hlp de Windows), desde aquí. NOTA: el formato WinHelp tan solo aparece para alguna versión bastante antigua.

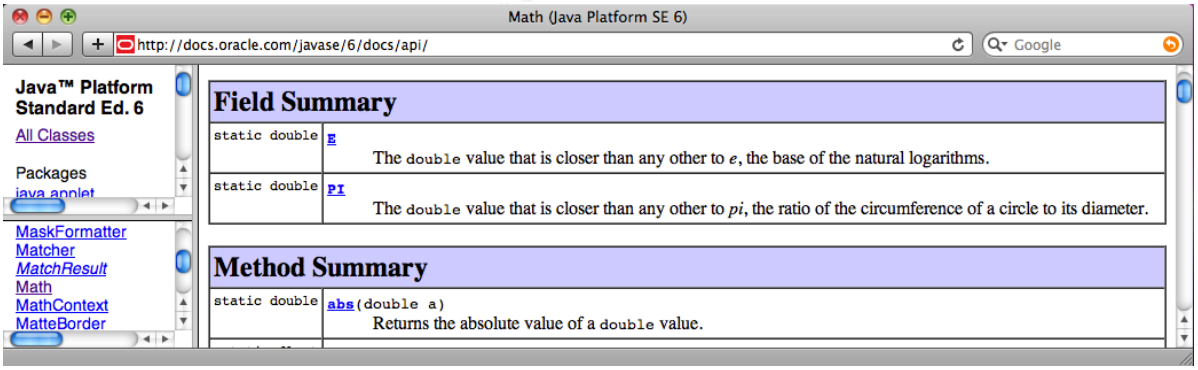


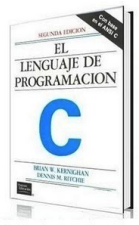
Figura 1.1: Documentación Java API

©2012 P. Pablo Garrido Abenza

1.4. El primer ejemplo: `HolaMundo`

En esta sección vamos a mostrar un primer programa escrito en Java, comparándolo con su correspondiente programa escrito en C. La sintáxis necesaria se explicará en el siguiente capítulo, aunque no es imprescindible comprenderlo con todo detalle. También se explica como compilar y ejecutar el programa.

Los desarrolladores del lenguaje C (Kernighan & Ritchie), en su libro "El Lenguaje de Programación C" [**?**], mostraron por vez primera el ejemplo "`holamundo`", que se ha convertido en el ejemplo clásico para mostrar como primer ejemplo a la hora de aprender un nuevo lenguaje de programación. Este programa simplemente se encarga simplemente de mostrar un mensaje de texto por la pantalla (`Hola mundo!`), terminando a continuación:




```c
/* ====================
 * Archivo: holamundo.c
 * ==================== */

#include <stdio.h>

void main () {
   // Mostramos un mensaje
   printf ("Hola mundo!\n");
}
```

Para ejecutar un programa C es necesario compilarlo previamente para la plataforma concreta para la que vaya a ejecutarse. El lenguaje C es bastante portable a nivel de código fuente, pero no lo es a nivel de código binario, es decir, un programa ya compilado en una plataforma no puede ejecutarse en otra plataforma distinta. Lo que hay que hacer es copiar el código fuente en la plataforma destino y recompilarlo, sin apenas realizar modificaciones en el código. Por ejemplo, utilizando el compilador GCC podemos compilar y ejecutar el programa anterior tanto desde la ventana del MS-DOS en Windows (a la izquierda) como desde la consola de Linux (a la derecha) del siguiente modo:

```
C:\ gcc holamundo.c -o holamundo
C:\ holamundo
Hola mundo!
C:\ _
```

```
$ gcc holamundo.c -o holamundo
$ holamundo
Hola mundo!
$ _
```

Vemos que en ambos casos los pasos son idénticos, siendo necesario la recompilación al cambiar de sistema. El programa `holamundo.c` se compila, y si no tiene errores de compilación se genera un fichero ejecutable `holamundo.exe` en Windows, o `holamundo` en Linux. Después se ejecuta el programa directamente desde el símbolo del sistema, puesto que se trata de un ejecutable nativo de la plataforma para la que se ha compilado. NOTA: el símbolo `C:\` es el símbolo del sistema de MS-DOS (*prompt*), al igual que el signo dolar `$` es el símbolo del sistema Linux, por lo que no hay que escribirlo (ver figura 1.2); del mismo modo, el subrayado al final `_` representa el cursor.

En su versión Java, el programa quedaría del siguiente modo. Si queremos probarlo, podemos utilizar cualquier editor de texto plano; en Windows el Bloc de notas (no el WordPad, ya que es un procesador de textos), en sistemas tipo Unix: nano, vi, pico, o emacs:

Copyright © 2012 P. Pablo Garrido Abenza

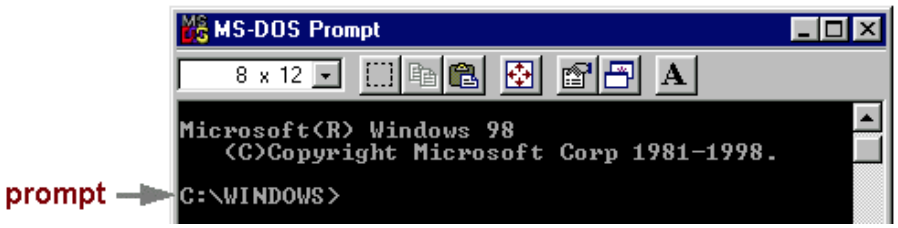


Figura 1.2: Ventana del MS-DOS (Windows)

```
/* ======================
 * Archivo: HolaMundo.java
 * ====================== */
class HolaMundo {
   public static void main (String args[]) {
      // Mostramos un mensaje
      System.out.println ("Hola mundo!");
   }
}
```

Con este pequeño programa vemos el gran parecido que hay entre el lenguaje Java y el lenguaje C: el punto de entrada al programa es la función `main()`, comentarios, cadenas de texto entre comillas dobles, bloques de código entre llaves, el símbolo punto y coma ; al final de cada instrucción, etc.

La diferencia más apreciable es que Java utiliza siempre la Programación Orientada a Objetos (POO), por lo que todo código está dentro de una clase, llamada en este caso `HolaMundo`:

```
class HolaMundo {
   ...
}
```

Para compilar el programa anterior utilizaremos el compilador Java desde línea de órdenes incluido en el kit de desarrollo de Java (JSDK): `javac`. Al igual que antes, si el código es correcto se produce un archivo ejecutable, en este caso llamado `HolaMundo.class` (tanto en Windows como en Linux), conteniendo un código intermedio o *bytecodes*, que no es nativo de la plataforma donde se ha compilado sino que es multiplataforma. Por ello, para ejecutar el programa compilado no se puede invocar directamente desde el símbolo del sistema como antes, sino que se pasa como argumento a la Máquina Virtual Java o JVM (orden `java`), que es quien interpretará esos *bytecodes*:

```
C:\ javac HolaMundo.java
C:\ java HolaMundo
Hola mundo!
C:\ _
```

```
$ javac HolaMundo.java
$ java HolaMundo
Hola mundo!
$ _
```

Al igual que antes, vemos que que los pasos son idénticos en ambas plataformas, pero, a diferencia que lo que ocurría con el lenguaje C, sólo es necesario compilar en una de las plataformas (cualquiera de ellas), y el archivo compilado puede portarse a cualquier otra plataforma y será ejecutable sin necesidad de recompilar. Para que esta portabilidad a nivel binario sea posible, y observando la forma de ejecutar los programas, pode-

Copyright © 2012 P. Pablo Garrido Abenza

mos sacar la conclusión de que para poder ejecutar un programa Java es necesario tener instalada una JVM. Actualmente existen máquinas virtuales Java para casi cualquier plataforma, no sólo desarrolladas por Sun Microsystems (ahora Oracle), sino por otras empresas. Algunas de las plataformas soportadas son: Windows, Linux, Mac OS X, Solaris, AIX, HP_UX, OS/400, Windows CE, Android, Symbian, ...

Notar que aunque el fichero ejecutable es `HolaMundo.class`, a la hora de ejecutarlo no se especifica la extensión `.class`. Además, es necesario respetar las mayúsculas/minúsculas.

1.5. Problemas comunes

Durante la compilación o ejecución de programas nos pueden surgir errores. Aunque la mayoría serán errores de sintáxis durante la compilación, no vamos a entrar en ellos, pues son muchos los posibles errores y muchas más causas; el alumno dispone del "Manual de prácticas", que incluye dos apartados llamados "Ayudante de compilación" y "Ayudante de ejecución", donde se da solución a los problemas más comunes.

Copyright ©2012- P. Pablo Garrido Abenza

GLOSARIO DE ACRÓNIMOS

Siglas	Significado
ANSI	American National Standards Institute
AOT	Ahead-Of-Time (técnica de compilación)
API	Application Programming Interface
ASCII	American Standard Code for Information Interchange
AWT	Abstract Window Toolkit
CISC	Complex Instruction Set Computing
DBMS	DataBase Management System
DLL	Dynamic Linked Library
ETSI	European Telecommunications Standards Institute
GNU	GNU Not Unix
GPL	GNU General Public License
GUI	Graphical User Interface
IDE	Integrated Development Environment
IEEE	Institute of Electrical and Electronic Engineer
ISO	International Standards Organization
JAR	Java ARchive
J2SE	Java 2 Platform, Standard Edition (~ Java SE)
J2EE	Java 2 Platform, Enterprise Edition (~ Java EE)
J2ME	Java 2 Platform, Micro Edition (~ Java ME)
JCP	Java Community Process
JDBC	Java DataBase Conectivity
JDK	Java Development Kit (~ Java 2 SDK y J2SDK)
JFC	Java Foundation Classes
JIT	Just-In-Time (técnica de compilación)
JNDI	Java Naming and Directory Interface
JNI	Java Native Interface
JRE	Java Runtime Environment
JVM	Java Virtual Machine
KNI	K Native Interface (~ JNI)
ODBC	Open Database Connectivity
OSI	International Organization of Standarization
PC	Personal Computer

Copyright © 2012 P. Pablo Garrido Abenza

PDA	Personal Digital/Data Assistant
RDBMS	Relational DataBase Management System
RISC	Reduced Instruction Set Computing
ROM	Read Only Memory
SDK	Standard Development Kit
SQL	Structured Query Language
SRAM	Static Random Access Memory
URL	Uniform Resource Locator
UTF-16	Unicode Transformation Format (16-bit)
WORA	Write Once, Run Anywhere

Copyright ©2012- P. Pablo Garrido Abenza