Introducción a la tecnología Java

Ricardo Pérez López

IES Doñana, curso 2020/2021

1. Introducción

2. Compilación vs. interpretación

3. Características de Java

4. Tipado estático vs. dinámico

5. El primer programa Java

1. Introducción

1.1 Historia

1.2 Versiones

1.3 Características principales

1.1. Historia

Historia

- Java es un lenguaje creado en 1995 por James Gosling en la empresa Sun Microsystems.
- La versión 1.0 se publicó en enero de 1996.
- Desde 2002, la evolución del lenguaje ha sido regulada por el JCP (Java Community Process).

Logo de Java

- El 13 de noviembre de 2006, Sun publicó gran parte de la tecnología Java como software libre, bajo los términos de la Licencia Pública General GNU (GPL).
- En 2010, Oracle Corporation compra Sun Microsystems, por lo que Java pasa a formar parte de Oracle.

1.2. Versiones

Versiones

Versión	Publicación
JDK Beta	1995
JDK1.0	23-ene-1996
JDK 1.1	19-feb-1997
J2SE 1.2	8-dic-1998
J2SE 1.3	8-may-2000
J2SE 1.4	6-feb-2002
J2SE 5.0	30-sep-2004
Java SE 6	11-dic-2006
Java SE 7	28-jul-2011

Versión	Publicación
Java SE 8	18-mar-2014
Java SE 9	21-sep-2017
Java SE 10	20-mar-2018
Java SE 11	25-sep-2018
Java SE 12	19-mar-2019
Java SE 13	17-sep-2019
Java SE 14	17-mar-2020
Java SE 15	15-sep-2020

1.3. Características principales

Características principales

- Los cinco objetivos principales que se plantearon al diseñar el lenguaje Java, y que a día de hoy siguen siendo sus características principales, son:
  - Debe ser sencillo, orientado a objetos y basado en una sintaxis conocida.
  - Debe ser robusto y seguro.
  - Debe ser portable e independiente de la arquitectura, permitiendo la ejecución de un mismo programa en varios sistemas operativos.
  - Debe ejecutarse con gran rendimiento.
  - Debe ser interpretado, multihilo y de enlace dinámico.

2. Compilación vs. interpretación

2.1 Máquinas reales vs. virtuales

2.2 Código objeto (*bytecode*)

2.3 La plataforma Java

2.4 El entorno de ejecución de Java (JRE)

2.5 Las herramientas de desarrollo de Java (JDK)

2.1. Máquinas reales vs. virtuales

Máquinas reales vs. virtuales

- Una **máquina abstracta** es una máquina diseñada independientemente de una determinada tecnología de fabricación.
- Su finalidad no es la de ser construida, sino servir como **modelo de computación teórica**.
- Una **máquina virtual** es una máquina emulada mediante hardware o software.
- Las máquinas virtuales pueden ser emulaciones de máquinas reales o abstractas.

2.2. Código objeto (*bytecode*)

Código objeto (*bytecode*)

- El compilador de Java traduce el código fuente (archivos con extensión `.java`) en código objeto (código binario almacenado en archivos con extensión `.class`) para una máquina virtual llamada ***Java Virtual Machine* (JVM)**.
- Al código objeto generado por el compilador de Java se le denomina ***bytecode***.
- Por tanto, el *bytecode* es el lenguaje máquina al que compila el compilador de Java y es, además, el único lenguaje que entiende la JVM.

2.3. La plataforma Java

2.3.1 La máquina virtual de Java (JVM)

2.3.2 La API de Java

La plataforma Java

- La **plataforma Java** es el nombre de una plataforma de desarrollo y ejecución de programas que se compone de un amplio abanico de tecnologías:
  - El **lenguaje de programación** Java.
  - La **biblioteca estándar** de Java.
  - La **máquina virtual** de Java (***Java Virtual Machine* (JVM)**).
  - La **implementación** de la JVM y de la biblioteca estándar (***Java Runtime Environment* (JRE)**).
  - Las **herramientas de desarrollo** (***Java Development Kit* (JDK)**).
- Para poder desarrollar y ejecutar programas Java, necesitamos una implementación de la plataforma Java que funcione en nuestro sistema operativo y nuestra arquitectura hardware.

- Existen cuatro *ediciones distintas* de la plataforma Java (cuatro «plataformas Java»), centradas en diferentes entornos de aplicaciones y segmentando mucha de sus API.
- Las plataformas son:
  - **Java Card**: para tarjetas inteligentes.
  - **Java Platform, Micro Edition (Java ME)**: para entornos con recursos limitados.
  - **Java Platform, Standard Edition (Java SE)**: para entornos de estaciones de trabajo.
  - **Java Platform, Enterprise Edition (Java EE)**: para grandes empresas o entornos de Internet.

La máquina virtual de Java (JVM)

- La **máquina virtual de Java** (del inglés, *Java Virtual Machine* o **JVM**) es una máquina virtual capaz de interpretar y ejecutar instrucciones expresadas en un código binario especial llamado *bytecode*, el cual es generado por el compilador del lenguaje Java, entre otros.
- El *bytecode* de Java no es un lenguaje de alto nivel, sino un verdadero **código máquina de bajo nivel**, viable incluso como lenguaje de entrada para un microprocesador físico.
- La gran ventaja de usar la JVM es la **portabilidad**, de manera que se han creado diferentes implementaciones de la misma máquina virtual para diferentes arquitecturas, y, así, un código objeto (archivo `.class`) puede ser ejecutado en cualquier sistema operativo y arquitectura hardware que disponga de una implementación de la máquina virtual.

- La regla máxima del diseño de Java es:

  «Escríbelo una vez, ejecútalo en cualquier parte.»

  «*Write once, run anywhere*.»

- La JVM puede estar implementada en software, hardware, una herramienta de desarrollo o un navegador web.
- Lee y ejecuta código *bytecode* independiente de la plataforma en la que está implementada la JVM.
- La JVM proporciona definiciones para un conjunto de instrucciones, un conjunto de registros, un formato para archivos de clases, la pila, un montículo con recolector de basura y un área de memoria.
- La definición detallada de la JVM está especificada mediante un estándar.
- Por tanto, toda implementación de la JVM debe cumplir con la especificación.

La API de Java

- Una **API (*Application Programming Interface*)** define un conjunto de funcionalidades recogidas en funciones y/o métodos que ofrece una determinada biblioteca para ser utilizado como una capa de abstracción por otro software o por el programador de un lenguaje de programación.
- Los sistemas operativos ofrecen servicios para simplificar la tarea de programación.
- Esos servicios se ofrecen en forma de un conjunto de bibliotecas dinámicas que las aplicaciones pueden llamar cuando lo necesiten.
- Como la plataforma Java está pensada para ser independiente del sistema operativo subyacente, las aplicaciones no pueden apoyarse en servicios ofrecidos por cada sistema en concreto.
- Por tanto, lo que hace la plataforma Java es ofrecer una **biblioteca estándar** que contiene mucha de las funciones disponibles en los sistemas operativos actuales.

- Esa biblioteca es accesible desde Java a través de la API de Java.
- Por tanto, **la API de Java especifica el contenido de esa biblioteca**, que ofrece sus servicios en forma de **clases** y otros elementos relacionados (como *interfaces*).
- La documentación del API de la versión 14 de la plataforma *Java Standard Edition* (Java SE) se encuentra bajo la siguiente dirección:

  https://docs.oracle.com/en/java/javase/14/docs/api/index.html

- Allí podemos comprobar que el API de Java SE está dividido en *módulos*, que a su vez se dividen en *paquetes*, que a su vez se dividen en *clases* e *interfaces*.
- Es fundamental tener siempre a mano la documentación de la API para poder programar con agilidad en este lenguaje.

- La **biblioteca de Java** tienen **tres funciones principales** dentro de la plataforma Java:
  - Ofrecen al programador un conjunto bien definido de funciones para realizar **tareas comunes**, como manejar listas de elementos u operar de forma sofisticada sobre cadenas de caracteres.
  - Proporcionan una **interfaz abstracta** para tareas que son altamente **dependientes del hardware** de la plataforma destino y de su sistema operativo.
  - No todas las plataformas soportan todas las funciones que una aplicación Java espera. En estos casos, las bibliotecas bien pueden **emular esas funciones** usando lo que esté disponible, o bien ofrecer un mecanismo para comprobar si una funcionalidad concreta está presente.

2.4. El entorno de ejecución de Java (JRE)

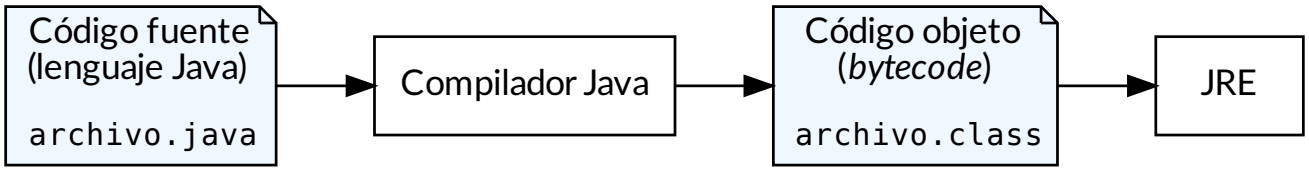
2.4.1 El intérprete `java`

El entorno de ejecución de Java (JRE)

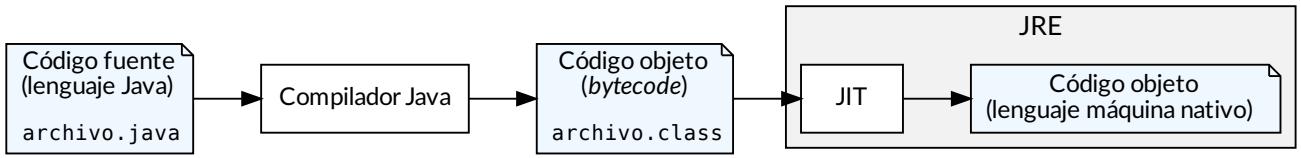
- El **entorno de ejecución de Java** (del inglés ***Java Runtime Environment*** o **JRE**) es el software necesario para ejecutar programas Java en un determinado sistema operativo y arquitectura hardware.
- Para cada dispositivo (ya sea un teléfono móvil, un PC con Linux o un microondas) debe haber un JRE.
- El JRE actúa como intermediario entre Java y el sistema operativo.
- Básicamente, consiste en una **implementación de la JVM y de la biblioteca estándar** (el API).
- Ambas (JVM y API) deben ser compatibles entre sí, de ahí que sean distribuidas conjuntamente en forma de JRE.

$$\text{JRE} = \text{JVM} + \text{API}$$

- El JRE actúa como un emulador de la JVM y un intérprete de *bytecode*.
- El JRE lee el código objeto (los archivos `.class`) y va ejecutando (interpretando) paso a paso las instrucciones compiladas en *bytecode* que se va encontrando.



- Desde hace ya tiempo, el JRE lleva a cabo un proceso previo de **compilación *Just In Time* (JIT)**, que convierte el *bytecode* a código nativo de la arquitectura donde se está ejecutando el JRE. Esto permite una ejecución mucho más rápida a costa de perder algo de tiempo al arrancar el programa.



El intérprete `java`

- El programa `java` es el intérprete que implementa la JVM en el JRE.
- Es el programa que usamos para ejecutar los programas compilados a *bytecode* almacenados en archivos `.class`.
- Como su extensión indica, un archivo `.class` contiene la definición de una **clase Java** compilada en *bytecode*.
- En Java, las instrucciones que forman un programa están prácticamente todas contenidas en clases. Por tanto, la ejecución de un programa Java empezará siempre desde una clase concreta.
- Para ejecutar un programa Java, debemos pasarle al intérprete `java` el nombre de la clase desde la cual queremos iniciar la ejecución del programa.

- Por ejemplo, si tenemos una clase `Principal` compilada en el archivo `Principal.class`, podemos indicar que queremos empezar desde ahí la ejecución de nuestro programa:

```
$ java Principal
```

- Es importante no confundir el nombre de la clase con el nombre del archivo que contiene el código compilado de la clase. En este caso, el nombre de la clase es `Principal`, no `Principal.class`:

```
$ java Principal.class
Error: no se ha encontrado o cargado la clase principal Principal.class
Causado por: java.lang.ClassNotFoundException: Principal.class
```

2.5. Las herramientas de desarrollo de Java (JDK)

2.5.1 El compilador `javac`

2.5.2 El intérprete interactivo `jshell`

3. Características de Java

4. Tipado estático vs. dinámico

5. El primer programa Java

5.1 El método `main()`

5.2 La clase `System`

5.3 El objeto `out`

5.4 El método `println()`

5.5 Compilación y ejecución en consola y en el IDE

5.1. El método `main()`

5.2. La clase `System`

5.3. El objeto `out`

5.4. El método `println()`

5.5. Compilación y ejecución en consola y en el IDE