Capítulo 1. Introducción a las tecnologías Java

A. J. Pérez

Orígenes de Java
La plataforma Java
- La máquina virtual Java (JRE)
- Las bibliotecas base de la plataforma Java (API's)
- El lenguaje de programación Java (JDK)

Distribuciones Java
- Java SE
- Java EE
- Java ME
- MV Android

Aspectos y características de Java
Mitos y realidades
Tipos de programas
- Standalone programs
- Applets
- Servlets
- Midlet
- App

Compilar aplicaciones y applets
El programa "HolaMundo"
- 1) Creación en un fichero fuente
- 2) Compilación del archivo fuente
  - 2b) Bytecode
- 3) Ejecución del programa

El Applet "HolaMundo"
- 1) Creación del archivo fuente
- 2) Compilación del archivo fuente
- 3) Creación de un archivo HTML que Incluya el Applet
- 4) Visualización del archivo HTML

Sintaxis del lenguaje Java
Ejercicios
Fuentes y bibliografía

Introducción a las tecnologías Java

Orígenes de Java

Java surgió en 1991 cuando un grupo de ingenieros de Sun Microsystems encabezado por James Gosling diseñaron un nuevo lenguaje de programación destinado a electrodomésticos y pequeños dispositivos electrónicos. La reducida potencia de cálculo y memoria de estos dispositivos llevó a crear un lenguaje sencillo capaz de generar código de tamaño muy compacto. Además, querían abandonar la metodología habitual de trabajo que se basaba en crear un nuevo compilador -o incluso lenguaje de programación- para cada nueva CPU o chip utilizado. **La gran idea consistió en que, para una aplicación, se podía disponer de un *código neutro* capaz de ser ejecutado sobre una *máquina genérica*, denominada *Java Virtual Machine (JVM)***. La JVM o máquina virtual pasó a ser una marca registrada de la plataforma Java.

A pesar de los esfuerzos realizados por sus creadores, ninguna empresa de electrodomésticos se interesó por el nuevo lenguaje.

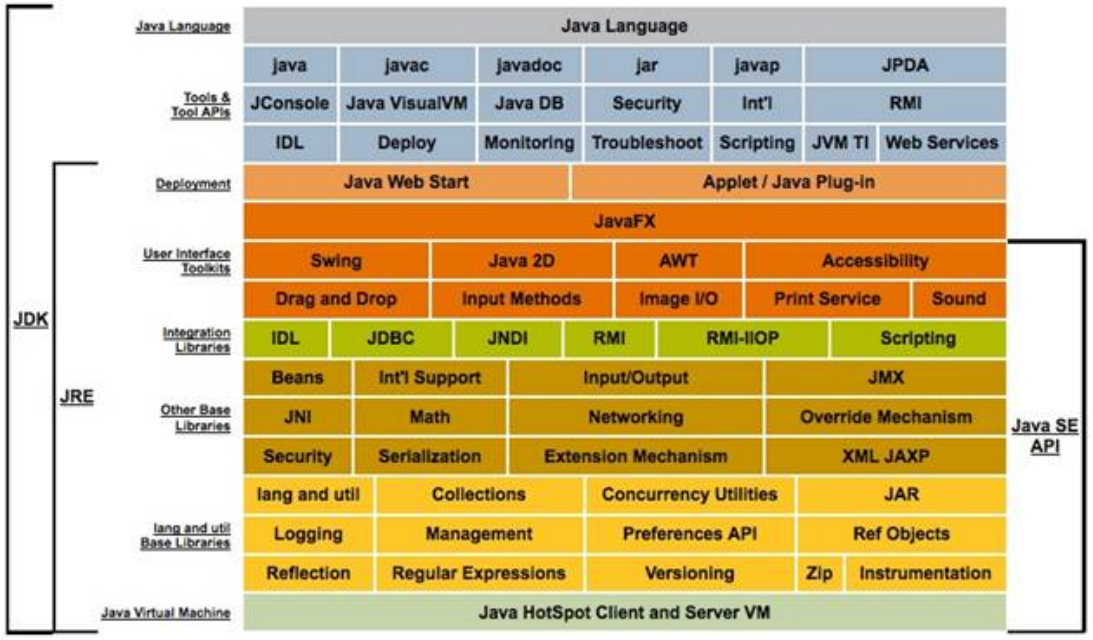
Lejos de su uso previsto original, **Java resurgió como una opción interesante en el contexto de Internet, donde se requería distribuir aplicaciones -*applets* inicialmente- destinadas a ejecutarse en una plataforma indeterminada y que el desarrollador no podía controlar sus características**. A finales de 1995, Java hizo acto de presencia tras la incorporación de un **intérprete Java en el programa navegador Netscape**; produjo una revolución en Internet. Después, en 1997, apareció Java 1.1 mejorando en gran medida la primera versión del lenguaje.

La portabilidad auténtica, es la primera característica singular de Java, diferente a la de cualquier otro lenguaje. Pero además, **Java aplica completamente el paradigma orientado a objetos**. Al programar en **Java no se parte de cero; cualquier aplicación que se desarrolle se basa en un gran número de clases preexistentes**. Algunas de las clases son creadas por el propio programador, otras son adquiridas de terceros, pero la mayor parte son proporcionadas por el propio lenguaje en forma de API (*Application Programming Interface*).

Java incorpora, de manera estándar, sencilla y clara, muchos aspectos que en cualquier otro lenguaje son extensiones propiedad de empresas de software o fabricantes de ordenadores (*threads, ejecución remota, componentes, seguridad, acceso a bases de datos*, etc.).

Como Java fue creado pensando en productos de consumo; resultó ser un **lenguaje robusto y apto para aplicaciones importantes, de misión crítica en entornos distribuidos, siendo más una plataforma que un lenguaje, debido a la extensión de su aplicación y portabilidad**.

La plataforma Java



fuente: http://docs.oracle.com/javase/8/docs/

La máquina virtual Java (JRE)

(JVM: Java Virtual Machine)(Java HotSpot Client and Server VM)

El *JRE* es el entorno de sistema imprescindible para la ejecución de aplicaciones Java. Incluye la máquina virtual Java y otros componentes que hacen posible ejecutar el Bytecode sobre un sistema operativo.

Las bibliotecas base de la plataforma Java (API's)

(Java Application Programming Interface = Java API)

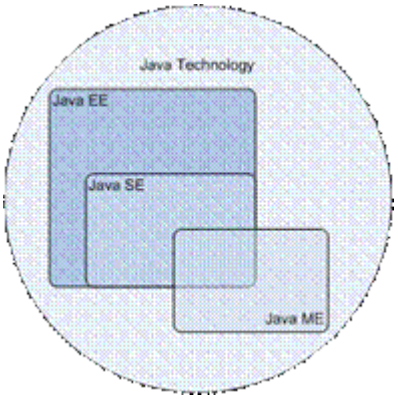
La *API Java* está formada por varias capas y bibliotecas de componentes estándar orientadas a necesidades específicas.

Desarrollo con el lenguaje Java (JDK)

El *JDK* es el conjunto de herramientas, utilidades y elementos para editar, compilar y mantener aplicaciones Java. Incluye el *JRE.*

Distribuciones Java

Actualmente hay tres distribuciones oficiales de la plataforma Java genuina; además de la MV *Dalvik* de *Android* que es compatible con Java -ya en desuso-. Cada una de ellas se corresponde con una plataforma que incluye las correspondientes API's con paquetes y elementos específicos.



Java SE

Java Standard Edition. Anteriormente conocida como *J2SE* (el dos se refiere a Java 2). **Permite escribir código Java relacionado con la creación de aplicaciones de escritorio (*standalone programs*) y applets en lenguaje Java.** Esta plataforma incluye:

- Compilador Java.
- Java Virtual Machine.
- Soporte para *interfaz gráfica de usuario (GUI)*.
- Soporte para *interfaz de red TCP/IP*.
- Soporte para trabajo con *XML*.
- Soporte para el manejo de ficheros en la máquina física.
- Interfaz para *ejecución nativa de código* en la plataforma física (JNI).
- Soporte para *bases de datos (JDBC)*.
- Soporte para *invocación de método remoto (RMI-IIOP)*.
- Soporte para gráficos 2D/3D.
- Soporte para la seguridad.
- Soporte para herramientas genéricas.

Java EE

Java Enterprise Edition. Todavía conocida como *J2EE*. **Prevista para la creación de aplicaciones Java distribuidas de carácter empresarial -sobre todo en lado del servidor-**. Incluye todos los componentes indicados para *SE* añadiendo otros componentes relacionados con las tecnologías siguientes:

- *HTTP* de servidor y contenedor de *Servlets* para el desarrollo de aplicaciones web distribuidas.
- Contenedor *EJB* para el *desarrollo de componentes reutilizables (RPC)*.

- *Transmisión de mensajes (JMS)*.
- Elementos de seguridad ampliada con respecto a *SE*.
- Los *servicios Web*.

Java ME

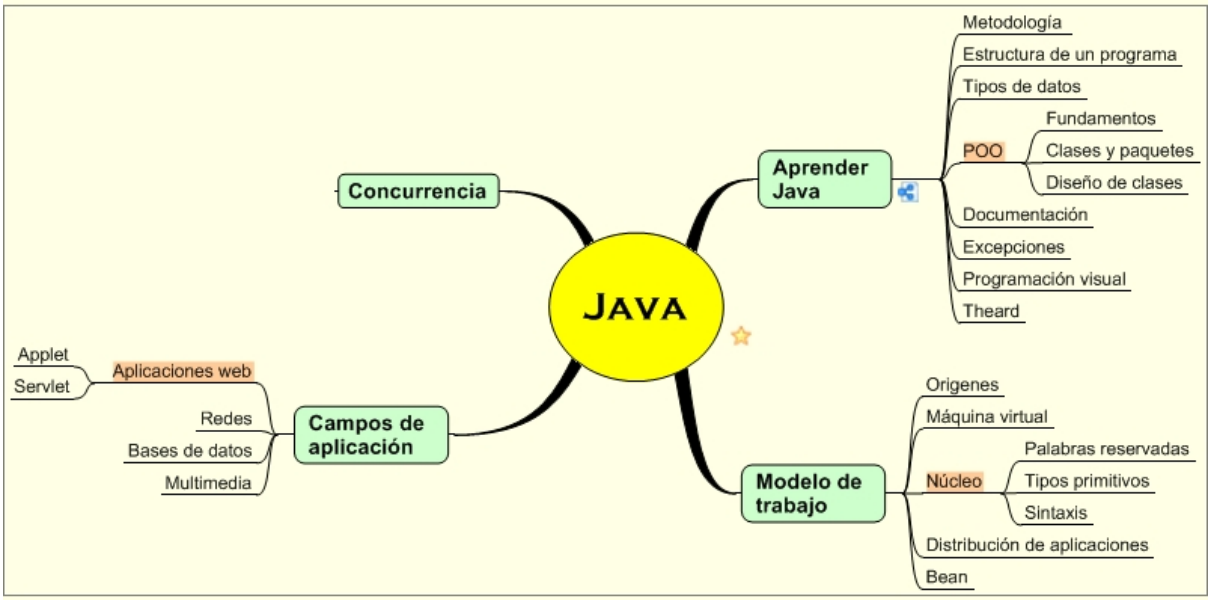
Java Mobile Edition. También conocida como *J2ME*. **Prevista para la creación de aplicaciones Java para pequeños equipos y dispositivos móviles**. Para lograr un rendimiento satisfactorio en dispositivos de características hardware limitadas, ***Java ME* simplifica y reduce las posibilidades de *Java SE* en varios aspectos; limitando el conjunto de operaciones a las consideradas básicas. Esto afecta tanto el compilador y el *JVM*.**

MV Android

Dalvik* ha sido hasta hace no muchos años la máquina virtual no Java original diseñada específicamente por Google para Android. Esta máquina virtual, compatible Java, estaba optimizada para dispositivos móviles que funcionan con batería y que tienen memoria y procesador limitados.** Las aplicaciones escritas en Java no son ejecutadas como tales, sino que **el bytecode Java es recompilado en un ejecutable *Dalvik y corre en la *VM Dalvik*.

Recientemente Google ha renovado completamente la *MV Android*. La nueva máquina virtual se denomina ART que proporciona mayor rendimiento y menor consumo; pero, al parecer, no está basada en bytecode Java, sino en lenguaje C++. A pesar de estas características de la máquina virtual ART, el código fuente de las aplicaciones sigue siendo en Java.

Aspectos y características de Java



Java está descrito como un lenguaje simple, orientado a objetos, distribuido, interpretado -el bytecode- en la JVM, robusto, seguro, neutro, portable, de altas prestaciones y concurrente. Cada una de estas palabras es una característica del lenguaje Java.

Muchas de las características mencionadas pueden usarse para describir otros lenguajes; todas juntas son una combinación única en Java y su ambiente.

- **Simple:** Se dice que Java es simple porque aunque **tiene todas las características de los lenguajes avanzados, descarta las características menos usadas y más confusas de éstos**. C y C++ son lenguajes muy conocidos, por eso la sintaxis Java se diseñó para ser parecida a C++ y así facilitar su aprendizaje. Java no usa o no permite:
  - Liberar memoria.
  - Aritmética de punteros.
  - Registros.
  - Definición de tipos (typedef).
  - Macros.
  - Herencia múltiple.

- **Orientado a objetos:** Desde el punto de vista de usuario, **al ejecutar un programa; no se percibe para nada la *orientación a objetos*, no importa si está escrito con programación convencional o con enfoque orientado a objetos**. **Sólo desde la perspectiva de un programador se aprecia que un *programa orientado a objetos* implica más requisitos de diseño e implementación**. Estos requisitos y los ajustes asociados, son probablemente la parte más difícil en el aprendizaje de Java.

- **Distribuido:** Java viene con una ***API de red* que facilita al programador el trabajo con protocolos de bajo nivel como *TCP/IP (Transport Control Protocol / Internet Protocol)* o alto nivel como *HTTP (Hyper Text Transfert Protocol)* y *FTP (File Transfer Protocol*).**

- **Interpretado:** La ejecución de un programa en Java pasa a través de dos fases.
  1. En tiempo de compilación: **El compilador de java traduce el código fuente en *código máquina Java* de bajo nivel; es el llamado *bytecode*.**
  2. En tiempo de ejecución: **El *bytecode* del programa es interpretado por la *JVM* que lo procesa y traduce a instrucciones de la máquina física. La máquina física a su vez interpreta el código máquina nativo.**

- **Robusto:** Java no sólo **verifica el código en busca de problemas en *tiempo de compilación;* también lo hace en *tiempo de ejecución***. La comprobación de tipos en Java ayuda a detectar errores, lo antes posible, en el ciclo de desarrollo. Entre otras cosas proporciona la comprobación de punteros, límites de arrays, y manejo de excepciones, etc.

- **Seguro:** Debido a que **Java fue diseñado para su ejecución en un ambiente distribuido en red**, tiene restricciones de seguridad. Además **durante la ejecución, la *JVM* usa un mecanismo para verificar que el *bytecode* cargado a través de la red no viola ninguna restricción del lenguaje Java**.

- **Concurrente:** En la actualidad muchas aplicaciones permiten al usuario realizar múltiples tareas en forma simultánea. **La capacidad para ejecutar varios procesos al mismo tiempo se conoce como *multihilo*. Java proporciona mecanismos y soporte para implementar aplicaciones multihilo.**

- **Dinámico:** Java es un lenguaje dinámico porque puede adaptarse a un ambiente cambiante y en desarrollo. **Java no intenta enlazar todos los módulos que conforman una aplicación hasta el *tiempo de ejecución.***

- **Neutro:** La red Internet es la composición de muchos ordenadores interconectados con procesadores y sistemas operativos muy diferentes. **Java fue creado con el objetivo de ser capaz de ejecutar sus programas en cualquier tipo de ordenador sin tener que recompilar el código fuente.** Por esta razón, **cuando se compila un programa fuente de Java, el resultado no es un código máquina nativo, sino un *bytecode* para una máquina de arquitectura genérica o neutra de bajo nivel que representa los datos en un formato independiente de la máquina física subyacente.** El intérprete de Java puede -más tarde- leer este *bytecode* y traducirlo a llamadas de sistema dependientes de máquina.

- **Portable:** Ser de arquitectura neutra es sólo una parte de ser portable. Java además **implementa otros estándares de portabilidad para facilitar el desarrollo, por ejemplo los enteros son siempre de 32 bits, la construcción de interfaces de usuarios se realiza utilizando las clases del paquete *AWT/Swing (Abstract Window Toolkit)* de forma que las ventanas puedan ser implantadas razonablemente en cualquier entorno.**

Mitos y realidades

- Mito: **"Escribe una vez, ejecuta en cualquier sitio"** (Write once, run anywhere)
  Realidad: Se puede conseguir, pero **no siempre es cierto... Puede ocurrir que aplicaciones Java ejecuten código nativo local** y además las interfaces gráficas de usuario pueden comportarse de forma algo diferente según las plataformas subyacentes.

- Mito: **"Java es un lenguaje de programación para la web"**.
  Realidad: Java **realmente es un lenguaje de programación de propósito general con bastante implantación de aplicaciones empresariales distribuidas y aplicaciones de escritorio**.

- Mito: **"Java es un lenguaje interpretado y por tanto lento en su ejecución"**.
  Realidad: Actualmente **Java incorpora en la JVM tecnología de compiladores *JIT (Just In Time) (compilación en tiempo de ejecución)*** para ejecutar el bytecode.

- Mito: **"La seguridad, la robustez y la independencia de la máquina apenas tiene coste"**.
  Realidad: **A pesar del uso de *compiladores JIT* en la máquina virtual, las aplicaciones resultantes son en la actualidad del orden de un 20% más lentas que en C++**.

- Mito: **"Java desplazará a tal o cual lenguaje o sistema (Microsoft, C++...)"**
  Realidad: **Siempre existen ventajas e inconvenientes**. Además existen iniciativas equivalentes de otros fabricantes como la plataforma .NET de Microsoft. Por otro lado -**determinadas aplicaciones- puede que sea mejor escribirlas buscando la máxima eficiencia y altas prestaciones;** siendo más adecuados lenguajes como C++, ANSI C o incluso ensambladores.

Tipos de programas

Standalone programs

Son programas independientes o aplicaciones creadas con características similares a las producidas con el resto de *lenguajes convencionales* de *3ª generación*. Según tengan un interfaz de usuario de texto o gráfico se las conoce como aplicaciones de consola o aplicaciones gráficas de escritorio. Se utiliza el JDK estándar de Java.

Applets

Son programas Java pensados para ser ejecutados en una página web utilizando el navegador como plataforma. Pueden ser interpretados por cualquier navegador con capacidades Java. Estos programas se insertan en las páginas usando una etiqueta especial (como también se insertan vídeos, animaciones flash u otros objetos).

Los *applets* son programas independientes, pero al estar incluidos dentro de una página web las reglas de éstas le afectan. Normalmente un *applet* sólo puede actuar sobre el navegador.

Mediante *applets* se pueden integrar en las páginas web aplicaciones multimedia avanzadas (incluso con imágenes 3D o sonido y vídeo de alta calidad)

Servlets

Son programas que se ejecutan en un servidor de aplicaciones web distribuidas y que como resultado interactúan o crean páginas web dinámicas en algún sistema cliente, a través de una red. Se utilizan los kits *SE JDK* y *EE SDK*

Midlet

Son aplicaciones creadas con Java puro para su ejecución en sistemas de propósito simple o dispositivos móviles. Los juegos Java creados para teléfonos móviles son *midlets*. Se utiliza específicamente el kit *ME SDK* y no se pueden crear con *SE JDK*

App

Son aplicaciones creadas con Java como medio para su ejecución en sistemas móviles, requieren la utilización de un conjunto de *API's* específicas de la *VM de Google (Dalvik)(ART)*. Se utiliza un kit de desarrollo específico, *Android SDK*, mantenido y distribuido sólo por Google y no es compatible, aunque adaptable, reescribiendo mucho código, al *ME SDK.*

Compilar aplicaciones y applets

Los programas más comunes de Java son:

- Las aplicaciones (*standalone programs).* **Se pueden ejecutar como cualquier otro lenguaje de programación, sin necesidad de ningún elemento soporte, directamente desde un sistema operativo.**

- Las *applets*. **Son pequeñas aplicaciones en Java, que se transfieren a través de la red y que necesitan, para su ejecución, un *browser* (navegador o visualizador) de Internet compatible con Java.**

Aquí se introduce, con un sencillo ejemplo, cómo se deben compilar las *aplicaciones* y *applets* Java

creadas con cualquier procesador de texto.

Para este ejemplo así como para el resto del manual se utiliza el *JDK (Java Developers Kit) SE*, que contiene todo lo necesario para desarrollar todo tipo de programas en Java.

El programa "HolaMundo"

Se muestra cómo crear y ejecutar un programa simple en Java.
Java requiere que todo programa esté constituido, al menos, por una clase principal donde reside el método `main()`. El método `main()` contiene las instrucciones del programa.

1) Creación en un fichero fuente

Hay que crear un fichero llamado `HolaMundo.java` con el código de Java mostrado aquí, asegurándose de que las mayúsculas del nombre del archivo coincidan con las de la clase.

```
/**
* HolaMundo.java
* Implementa una aplicación que muestra
* " Hola mundo..." en la salida estándar.
*/
public class HolaMundo {
	public static void main(String[] args) {
		System.out.println("Hola mundo...");
	}
}
```

El ejemplo de programa en código fuente presentado es lo que puede llamarse un ***programa mínimo* en Java: Una sola clase -la principal- que contiene sólo el método `main()`**. En este programa se puede encontrar:

`/**`
`* HolaMundo.java`
`* Implementa una aplicación que muestra`
`* " Hola mundo..." en la salida estándar.`
`*/`
	• Es un **bloque de comentarios** que el compilador ignora pero que ayudan a comprender lo que hace el programa.

`public class HolaMundo {`	• Es el **encabezado de la clase**.
	• `HolaMundo` Es el nombre de la clase principal del programa que debe coincidir exactamente con el nombre del fichero donde está contenida.
	• `public class` son dos palabras reservadas del lenguaje para indicar:
	  ○ que se está declarando una clase.
	  ○ Que el acceso a la clase es público. Es lo típico para la clase principal de un

	programa. • `{` es la llave de apertura, para indicar al compilador dónde empieza el bloque correspondiente a la clase que se está definido.

`public static void main(String[] args)`
`{`
	• Es el **encabezado del método `main()`**.
	• `main` es el nombre del método desde el que se empezará a ejecutar el programa cuando se lanze. Aunque no es una palabra reservada no puede utilizarse otro nombre.
	• `public static void` son tres palabras reservadas obligatorias para indicar:
	  ○ Que `main()` no devuelve resultado.
	  ○ Que `main()` siempre estará disponible en el programa.
	  ○ Que acceso a `main()` es público.
	• `(String[] args)` es lista de argumentos que opcionalmente puede recibir el método `main()` cuando se ejecute desde la línea de comandos. Hay que indicarla pero no se utiliza en programas sencillos como el ejemplo.
	• `{` es la llave de apertura, para indicar al compilador dónde empieza el bloque de instrucciones correspondientes al método `main()`.

`System.out.println("Hola mundo...");`	• Es la **sentencia** que imprime un texto en la pantalla. • `System` es el sistema físico del ordenador. • `.out` es la pantalla. El punto indica que es relativo a `System`. • `.println` es un método que muestra una línea de texto en la salida estándar -pantalla- • `("Hola mundo...")` es el argumento que contiene el texto a imprimir. Debe ir entrecomillado si es literal. Puede ser una variable que contenga el mensaje. • `;` es el fin de una sentencia en Java.
`}`	• Llave de cierre del bloque correspondiente al método `main()`. Debe tener siempre su pareja de apertura.
`}`	• Llave de cierre del bloque correspondiente a la clase `HolaMundo`. Debe tener siempre su pareja de apertura.

2) Compilación del archivo fuente

Para compilar el programa fuente de Java se puede utilizar el compilador de Java, *javac*,

dando el nombre del archivo fuente. Según la plataforma en la que se trabaje:

```
UNIX: javac HolaMundo.java
Windows (cmd): javac HolaMundo.java
MacOS: Arrastra el icono del fichero HolaMundo.java al icono del compilador
javac.
```

El compilador *javac* crea un archivo llamado `HolaMundo.class` que contiene el *bytecode* compilado del programa independientemente del procesador. Si la compilación fracasó, hay que asegurarse de haber escrito y nombrado correctamente el programa, tal y como se muestra.

2b) Bytecode

El compilador de Java compila su código fuente en un *bytecode*. **El término de *bytecode* viene del hecho que cada parte de su programa en Java se reduce a una secuencia de *bytes* que representan instrucciones en una máquina virtual.** El intérprete Java carga el *bytecode* del programa procesando cada instrucción.

3) Ejecución del programa

Para ejecutar el programa es necesario disponer de un intérprete de Java que cargue esta nueva clase y la ejecute. **Se hace pasando al intérprete el nombre de la clase `HelloWolrdApp`**. Según la plataforma en la que se trabaje:

```
UNIX: java HolaMundo
Windows (cmd): java HolaMundo
MacOS: Doble-click en el icono del fichero HolaMundo.class
```

Como resultado, se debería ver en la pantalla el mensaje:

```
Hola mundo...
```

El Applet "HolaMundo"

Siguiendo estos pasos se podrá crear un applet en Java.

1) Creación del archivo fuente

Hay que crear un archivo llamado `HelloWorld.java` con el siguiente código de Java, asegurándose que las mayúsculas del nombre del archivo coincidan con el de la clase.

```
import java.applet.Applet;
import java.awt.Graphics;
```

```
public class HolaMundo extends Applet {
       public void paint(Graphics g) {
          g.drawString("Hola mundo...", 50, 25);
       }
}
```

2) Compilación del archivo fuente

Windows y Unix: En una consola de comandos hay que escribir:

```
javac HolaMundo.java
```

MacOS: Hay que arrastra el icono de el fichero `HolaMundo.java` al icono del compilador *javac*.

El compilador *javac* crea un archivo llamado `HolaMundo.class` que contiene el *bytecode* compilado del programa. Si la compilación falla, hay que comprobar que se ha escrito y nombrado correctamente el programa, tal y como se indica.

3) Creación de un archivo HTML que Incluya el Applet

Hay que crear un archivo en el mismo directorio con el nombre `Hello.html`, que deberá contener la llamada a `HolaMundo.class` creada anteriormente.

El texto contenido del archivo HTML es el siguiente:

```
<HTML>
       <HEAD>
              <TITLE> A Simple Program </TITLE>
       </HEAD>
       <BODY>
              Hola mundo.
               <APPLET CODE = "HolaMundo.class" WIDTH=150 HEIGHT=25>
              </APPLET>
       </BODY>
</HTML>
```

4) Visualización del archivo HTML

Hay que cargar el Archivo HTML en un navegador que admita *applets* de Java, como por ejemplo el *Appletviewer* que se proporciona con el *JDK (Java Developers Kit)*.

Windows y Unix: En una consola de comandos hay que escribir:

```
appletviewer file:/home/java/Hola.html
```

MacOS: Hay que ejecutar el *AppletViewer* desde el *menú* ***File***, eligiendo abrir URL introduciendo la URL del fichero HTML que ha creado, por ejemplo:

```
file:/home/java/Hola.html
```

Para cargar el archivo HTML, se necesitará especificar al navegador la URL del archivo HTML creado. Por ejemplo, como URL se podría introducir:

```
file:/home/HTML/Hola.html
```

Una vez completados correctamente los pasos, debería verse en la ventana del *navegador web*:

```
Hola Mundo…
```

Sintaxis del lenguaje Java

Ver el enlace: http://www.arrakis.es/~abelp/ApuntesJava/Sintaxis.htm

Ejercicios

Fuentes y bibliografía

- NAKOV, S. *Fundamentos de programación con Java*. Ed. Faber, Veliko Tarnovo, 2009. [en linea][traducción automática del búlgaro] http://www.introprogramming.info/intro-java-book/
- CARRERES, J. *Manual de Java*. [en línea] http://www.oocities.org/collegepark/quad/8901/indice.html
- BEL PUCHOL, A. *Apuntes lenguaje Java.* [en línea] http://www.arrakis.es/~abelp/ApuntesJava/indice.htm http://www.geocities.com/belpuchol/ApuntesJava/indice.htm