
TEMA 1: INTRODUCCIÓN

[1.1- La evolución del modelo de objetos](#)

[1.2- Evolución de los lenguajes de programación](#)

[1.3- Orígenes de Java](#)

[1.4- Definición](#)

[1.5- Características](#)

[1.6- Java es interpretado](#)

[1.7- Tipos de programas Java](#)

[1.8- API](#)

1.1.- LA EVOLUCIÓN DEL MODELO DE OBJETOS

- **Datos y subprogramas:** su principio fundamental era dividir el programa en subprogramas más pequeños y fáciles de resolver, hasta llegar a niveles de complejidad elementales. La idea principal es *¿qué debe hacer el programa?* Las limitaciones de este método de diseño son:
 - No favorece la reutilización de código
 - Si dos subprogramas comparten una misma función, reutilizando así código que define la función, y más adelante queremos modificar la función porque hay un cambio en uno de los subprogramas que la utilizan, la modificación afectará también al otro programa, razón por la que ahora tendremos que realizar dos funciones.

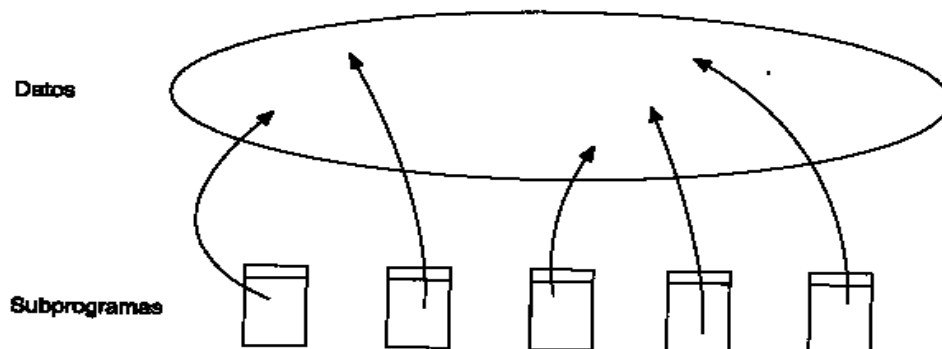


Figura 1.1: Datos y subprogramas

- Tipos abstractos de datos

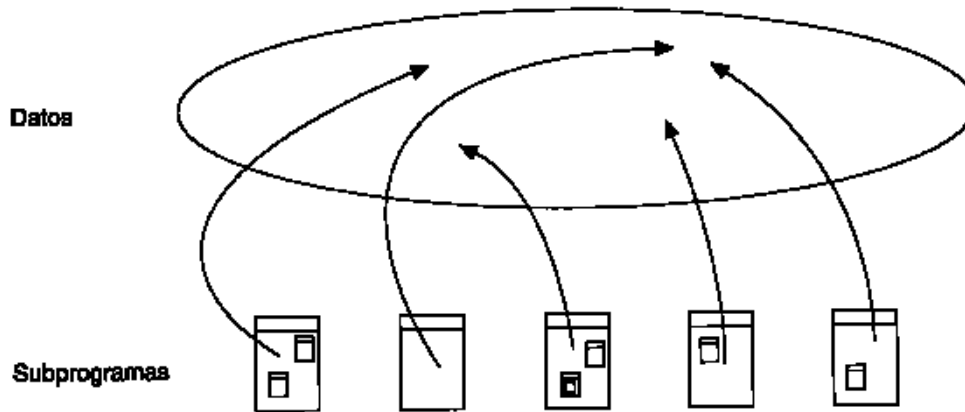


Figura 1.2: Tipos abstractos de datos

- Pequeñas aplicaciones orientadas a objetos:

La programación orientada a objetos se interesa primero por los datos y luego asocia los procedimientos. La idea principal es *¿de qué trata el programa?* El programa se desarrolla alrededor de los datos manipulados. Esto es eficaz porque en la vida real los elementos más estables son los datos.

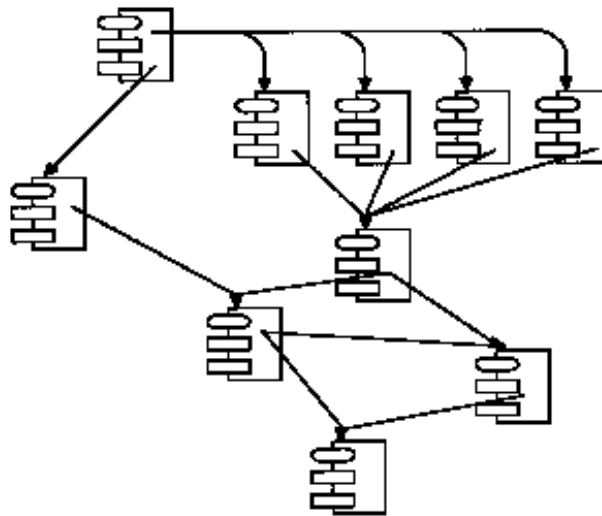


Figura 1.3: Aplicaciones orientadas a objetos

1.2.- EVOLUCIÓN DE LOS LENGUAJES DE PROGRAMACIÓN

Son varios los lenguajes que han contribuido a la evolución de los lenguajes orientados a objetos (LOO) de hoy: LISP en la década de los 50, Simula en los 60 y más tarde Pascal, C, Modula y Ada. Aunque estos lenguajes no incluyen los mecanismos para la POO, sus características sirvieron de base para la construcción de estos mecanismos.

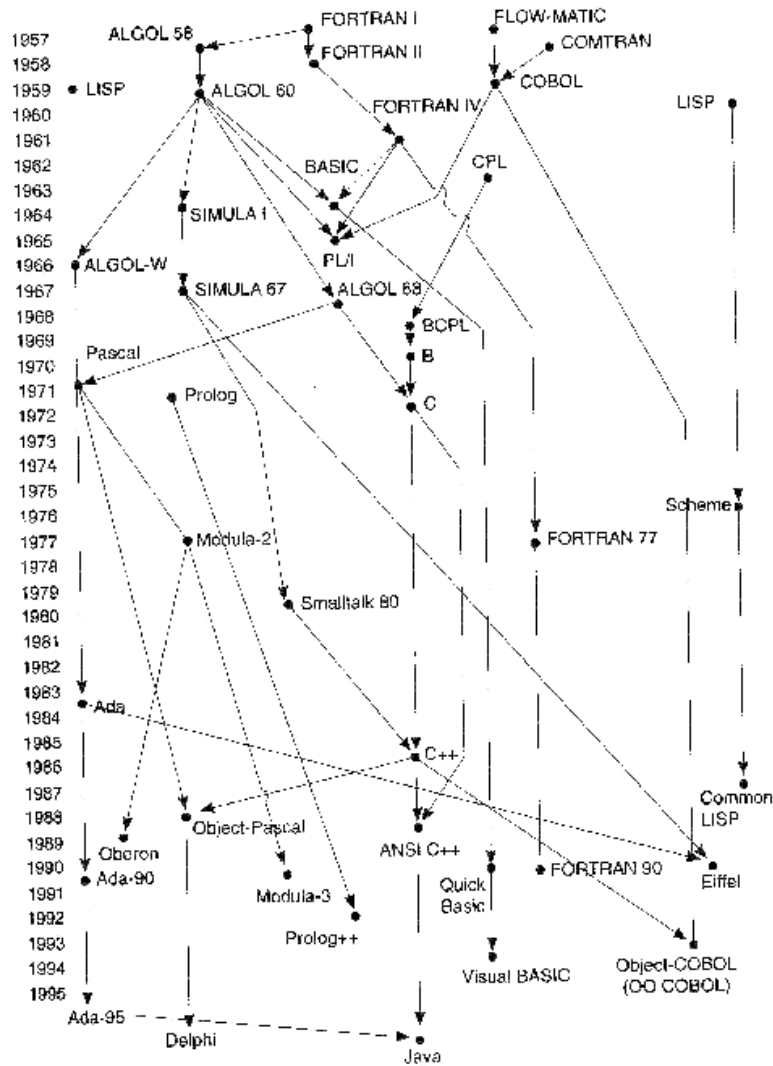


Figura 1.4: Evolución de los lenguajes de programación

En la década de los 70 aparece Smalltalk como LOO puro. Es en la década de los 80 cuando los avances son mayores, debido a la disponibilidad de extensiones orientadas a objetos en dos de los lenguajes más populares, C y Pascal. Surgieron los LOO híbridos C++ y Pascal orientado a objetos, que guardan la compatibilidad con sus antecesores.

1.3.- ORÍGENES DE JAVA

En 1991 un grupo de ingenieros de Sun Microsystems en California, liderados por James Gosling, se propuso diseñar un nuevo lenguaje para aparatos electrónicos, tostadoras, microondas, etc. El proyecto se llamó "Green". El lenguaje propuesto debía ser:

- pequeño
- eficiente
- portable

El lenguaje generaría un código intermedio, que se interpretaría en todos los aparatos. El código intermedio generaba programas pequeños, y los intérpretes eran sencillos. Ese código intermedio serían los bytecodes de la Máquina Virtual de Java (JVM).

El lenguaje de alto nivel en el que se basaron no fue el Pascal sino C y C++. Sin embargo lo abandonaron debido a su complejidad. Decidieron crear un nuevo lenguaje orientado a objetos (OO) basado sólo en partes de C++. Lo llamaron Oak, llamado así por el roble existente fuera de las oficinas de Gosling, como el nombre ya estaba siendo utilizado comercialmente lo denominaron Java.

En 1992 el proyecto Green sacó su primer producto, un control remoto inteligente llamado "*7", pero fue un fracaso comercial. Intentaron llevarse un contrato de un nuevo servicio de televisión por cable y no se lo concedieron.

Durante 1993 y 1994 intentaron vender su revolucionaria tecnología, visto el poco éxito tuvieron que disolver el proyecto.

Mientras todo esto ocurría estaba naciendo la Web. Comenzaban a salir los primeros navegadores (Mosaic, Netscape, etc.). Se dieron cuenta de que lo que le hacía falta a la Web era justo su lenguaje, ya que era:

- portable
- seguro
- pequeño

Así que construyeron el navegador HotJava para mostrar las posibilidades de su lenguaje. HotJava no sólo estaba hecho en Java, sino que además, ejecutaba programas Java (applets). Permitía incrustar aplicaciones en las, hasta entonces, aburridas páginas Web.

Fue presentado en el SunWorld de 1995 y ahí empezó su expansión. Pero, para muchos, lo que lanzó definitivamente a Java fue el hecho de que Netscape decidiera integrar el intérprete en su navegador. A continuación lo licenciaron IBM, Borland, Microsoft, Symantec, ...

1.4.- DEFINICIÓN

Java NO:

- es una extensión de HTML
- tiene un Entorno de Desarrollo Integrado (IDE) avanzado
- es rápido
- es un lenguaje sólo de páginas Web: se suele decir que es un lenguaje para Web porque se usa para escribir programas llamados applets que se ejecutan en un navegador Web. Pero no está limitado a escribir applets.

Java es un lenguaje de programación moderno:

- Orientado a objetos (OO)

- Tiene extensiones que permiten diseñar Interfaces de Usuario Gráficas (GUI): AWT, SWING
- Tiene muy poco de novedoso
- De C y C++ toma la sintaxis y el ámbito de las variables
- De Smalltalk toma la gestión de memoria dinámica y la ejecución multi-hilos (multithread)
- A esto Java añade seguridad, un simple paradigma de programación, y más...

1.5.- CARACTERÍSTICAS

Java está revolucionando el mundo de la programación. Ningún lenguaje se había desarrollado tanto en tan poco tiempo y ningún lenguaje había conseguido tantos adeptos (y enemigos) tan rápidamente.

En uno de los primeros artículos sobre Java, Sun lo describía como:

"A simple, object-oriented, distributed, interpreted, robust, secure, architecture neutral, portable, high-performance, multithreaded, and dynamic language"

Simple

Java es un lenguaje simple. Para que el aprendizaje no supusiese grandes esfuerzos se diseñó una pequeña cantidad de construcciones en el lenguaje. También se intentó que fuese similar a otros lenguajes de programación, como C o C++.

Suprimieron una serie de características de C y C++ que llevaban a una mala programación y que raramente se utilizaban.

- Java no permite la sentencia *goto*, para lo que ofrece otras posibilidades como *break* etiquetadas y excepciones
- Java no tiene ficheros de cabecera y elimina el preprocesador
- Como es orientado a objetos se han eliminado las uniones y estructuras
- No se utilizan punteros, Java gestiona automáticamente las direcciones (&) e indirecciones (*) de los objetos
- Implementa automáticamente la recolección de basura (garbage collection), por lo que no hay que preocuparse de la gestión de memoria: referencias no válidas a memoria, reserva y liberación de memoria, etc.

Para el programador de C++, Java es un lenguaje mucho más sencillo. Sin embargo, no opinan lo mismo los programadores de Visual Basic.

Otra ventaja derivada de ser simple es ser pequeño. Como estaba pensado para aparatos electrónicos debía generar programas compactos. Un intérprete más las librerías básicas sólo ocupa unos 200 Kb.

Orientado a Objetos

La POO es el paradigma más utilizado actualmente. Java es OO y, por tanto, se basa en la construcción de aplicaciones a base de objetos.

Como programador, esto significa que se centra en los datos de la aplicación y en los métodos que los manipulan, más que hablar estrictamente en términos de procedimientos.

Al contrario que C++, Java fue diseñado, desde el principio, orientado a objetos, es orientado a objetos *puro*.

La programación orientada a objetos (POO) proporciona una forma más simple de modelar un problema, y produce un diseño más claro y manejable. La programación se hace más agradable y aumenta la productividad por la potencia de las herramientas de POO y de las librerías proporcionadas.

La mayor desventaja sería el coste del aprendizaje. Pensar en objetos resulta un cambio muy grande con respecto a la programación procedural, sobre todo si se intenta crear objetos reusables.

Para algunos no es un lenguaje OO tan puro como Eiffel o Smalltalk y para otros no es tan de bajo nivel como C++. Es el buscado término medio para algunos programadores.

Robusto

Java fue diseñado para escribir software fiable y robusto. No se crean programas completamente seguros, sigue siendo posible escribir programas con fallos de seguridad en Java, pero elimina ciertos errores que hacen bastante más fácil crear programas fiables. Las principales causas de error de lenguajes como C o C++, y que han sido eliminados en Java son:

- Aritmética de punteros
- Accesos fuera de rango (strings y arrays)
- Corrupción de la memoria (al eliminar los punteros y al hacer garbage collection)

Independiente de la arquitectura

Java no se compila a código máquina, genera "bytecodes" para la Máquina Virtual de Java (JVM). Por tanto, se podrá ejecutar en cualquier plataforma que tenga el intérprete.

Actualmente, el intérprete se encuentra en las principales plataformas (Windows, Unix, Mac, Linux, etc.).

Portable

Como el bytecode generado por el compilador de Java es independiente de la arquitectura, una aplicación Java se puede ejecutar en cualquier plataforma que implemente la JVM. Esto es especialmente importante para aplicaciones distribuidas en Internet o en redes heterogéneas.

Más allá de las redes, esta característica permite que una misma aplicación se ejecute en PC, Mac, estaciones UNIX, etc. y para diferentes sistemas dentro de estos: Windows95, NT, etc.

Java asegura no tener aspectos "dependientes de la implementación". Por ejemplo, explícitamente especifica el tamaño de cada tipo primitivo de datos, así como su aritmética. Esto no sucede en C.

Distribuido

Java es el lenguaje más capacitado para Internet. Cualquiera que haya usado otros lenguajes apreciará la sencillez y la potencia de Java.

Además, dado que es portable, está especialmente capacitado para la distribución de software a través de Internet.

Actualmente todos los navegadores (Netscape, Internet Explorer) pueden ejecutar programas Java que incorporan audio, vídeo y animación directamente en las páginas Web. Dando un mayor dinamismo a las, tradicionalmente estáticas, páginas Web.

Seguro

Java fue diseñado, desde el principio, pensando en la seguridad. Esto es especialmente importante dada la naturaleza distribuida de Java. Sin ciertas garantías nadie querría bajar código de Internet y ejecutarlo en su ordenador, que es lo que hacen todo los días millones de personas con los applets de Java.

El no poder manejar punteros a memoria, o sobrepasar el límite de los arrays o strings son una de las principales defensas contra el código mal intencionado.

La segunda línea de defensa es la verificación del bytecode que realizan todos los intérpretes. Esta verificación asegura, por ejemplo, que el código esté bien formado, que no acceda de forma incorrecta a la pila o que no contenga bytecode ilegal.

Otra capa de seguridad es la denominada "Sandbox model": el código sospechoso se coloca en una caja de arena, donde se puede ejecutar de forma segura, sin poder causar daño al mundo real o al entorno Java. Cuando hay código ejecutándose en la "sandbox" existen ciertas restricciones en cuanto a lo que se puede o no hacer. La más obvia es que no se puede acceder a ficheros locales del sistema.

Existe otra solución al problema de la seguridad: añadir una "firma digital" al código Java, en la que se especifica el origen del código.

Java no es un lenguaje totalmente seguro, ninguno lo es, lo que sí ha hecho es tratar de contemplar los métodos que tradicionalmente se han utilizado para corromper el software.

Eficiente

Según los creadores, la velocidad de interpretación de Java era "adecuada" para la mayoría de las aplicaciones. En los casos en los que se necesitara mayor rendimiento podía usarse un compilador nativo.

En el momento de estas afirmaciones, nada más lejos de la realidad: Java era muy lento. Actualmente, debido a las tecnologías JIT y a numerosas optimizaciones de las JVM, Java ya tiene un rendimiento aceptable.

Multihilo (Multithread)

Java no sólo ofrece la posibilidad de crear programas multithread que se ejecuten en distintas plataformas, además, es sencillo de utilizar.

La ejecución multithread es la idea de ejecutar más de una tarea al mismo tiempo. Generalmente es la forma de reservar tiempo de un único procesador, si el SO soporta el multiprocesamiento se puede asignar un procesador a cada hilo de ejecución.

La ventaja de soportar la ejecución multithread desde el lenguaje de programación es que el programador no tiene que preocuparse de si hay varios procesos ejecutándose o sólo uno.

1.6.- JAVA ES INTERPRETADO

Esto es una verdad a medias. El código Java se precompila, generando bytecodes (ficheros ".class"). Posteriormente, es la máquina virtual de Java (JVM) quien, para poder ejecutar este bytecode, lo interpreta. Por lo que se pueden ejecutar programas Java en cualquier plataforma que tenga el intérprete de Java y el entorno de ejecución.

El bytecode integra lenguaje y E/S (sistema operativo). Esto permite la portabilidad e independencia de la arquitectura. El doble paso de la interpretación es el siguiente:

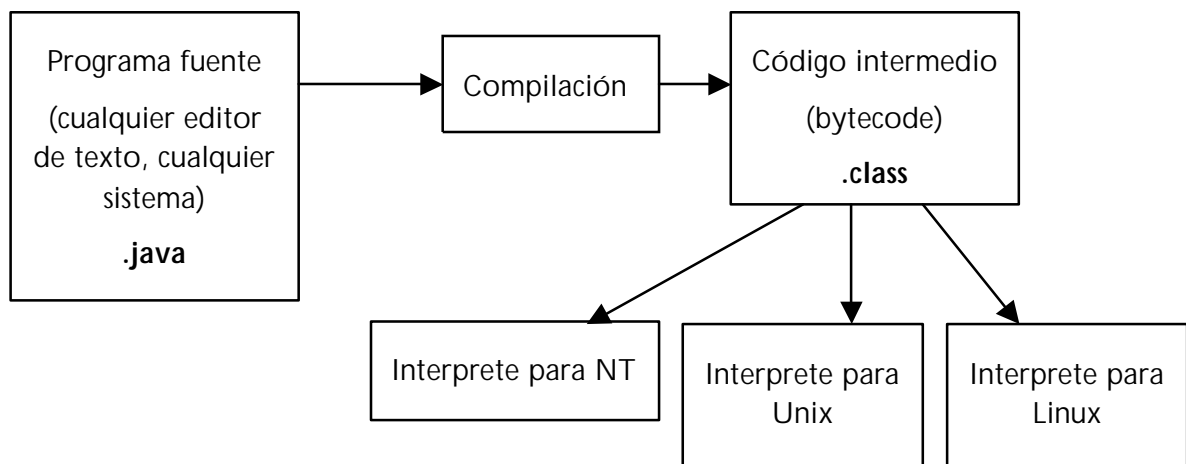
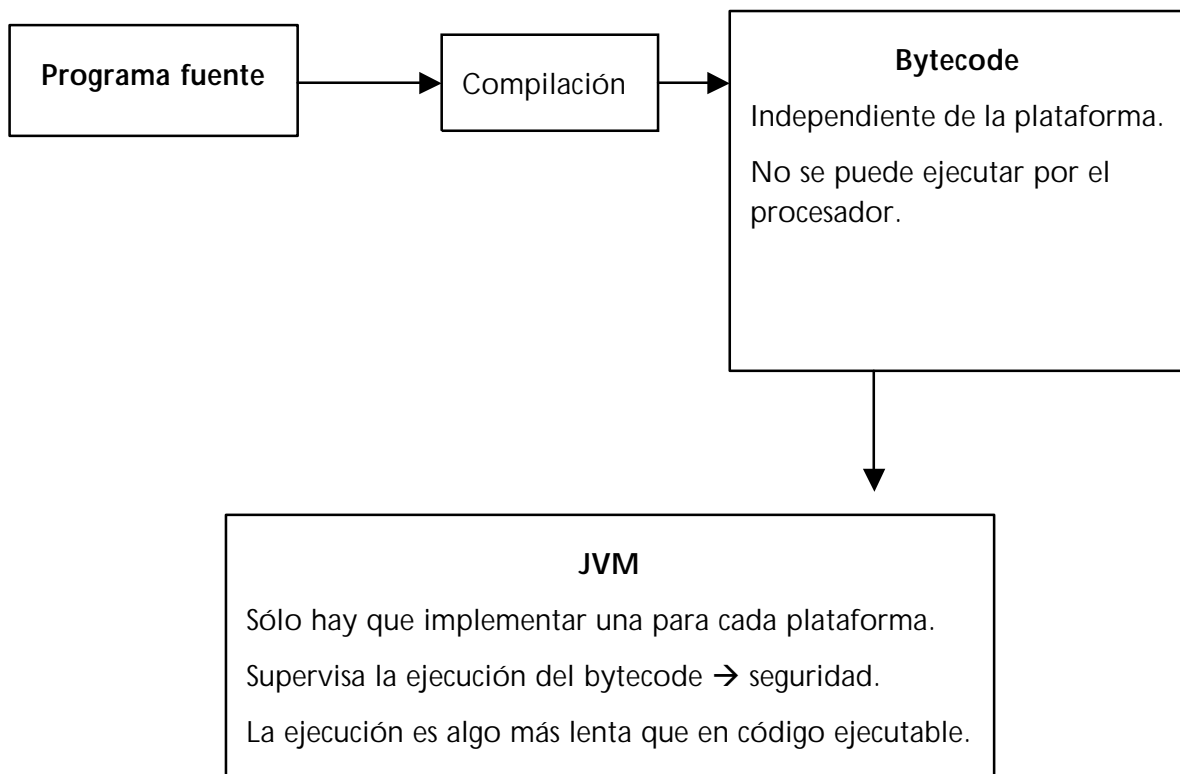


Figura 1.5: Proceso de Compilación e interpretación de un programa Java

1.7.- MÁQUINA VIRTUAL JAVA (JVM)



1.8.- TIPOS DE PROGRAMAS JAVA

Los programas Java tienen dos formas: aplicaciones autónomas (standalone) ejecutables desde la línea de comandos y applets, que se ejecutan en el navegador y están incrustados en otros lenguajes, por ejemplo HTML. Ambas se diferencian ligeramente en su estructura.

Los applets necesitan unas condiciones de seguridad para evitar daños en el sistema en el que está funcionando el navegador, por tanto tiene algunas limitaciones:

- No pueden leer/escribir en el sistema de archivos del ordenador, pues en caso contrario podría producir daños en archivos y propagar virus.
- No pueden establecer conexiones entre el ordenador de un usuario y otro ordenador, excepto que sea el servidor donde están almacenados los applets (esta actividad comienza ya a desarrollarse con las tecnologías Napster, a primeros del año 2000. Otras tecnologías similares son Gnutella, Scour, etc. y se las conoce de modo genérico como tecnologías P2P (Peer-to-peer)).
- No pueden ejecutar programas del ordenador donde reside el navegador, dado que podrían originar daños al sistema.

“Si su programa no necesita ejecutarse en un navegador Web, elija crear aplicaciones.”

Escribir, compilar y ejecutar una aplicación

```
public class HelloWorld
{
    public static void main (String[] args)
    {
        System.out.println("Hello World!!");
    }
}
```

Como en programas de C y C++, main() es lo primero que se ejecuta.

Paso 1

Escribir el programa en un editor de texto y almacenarlo en un archivo. Utilizar el nombre de la clase principal y el sufijo `.java` para el nombre del archivo, este archivo se denomina `archivo fuente`.

En el ejemplo el nombre del archivo sería `HelloWorld.java`

¡¡Asegurarse de que el editor no incluye la extensión `.txt` al archivo!!

Un archivo puede contener la definición de **más de una clase**, el nombre lo toma de la clase principal (la que contiene el `main`).

En Java todo **el código debe residir en el interior de una clase**, al contrario que en C++, en el que la definición de métodos podía ir fuera de la clase.

Paso 2:

Compilar el archivo fuente. Cuando la compilación tiene éxito (no hay ningún error) se crea la versión compilada del archivo fuente, llamada código objeto. El archivo objeto generado por el compilador tiene el mismo nombre que el archivo fuente pero con la extensión `.class`. El código objeto generado por el compilador de Java se llama *bytecode*.

```
javac HelloWorld.java //Javac es el nombre del compilador que vamos a
                        utilizar
```

Un **error muy común** es escribir de forma incorrecta las palabras clave. Hay que tener en cuenta que Java es sensible a las letras mayúsculas y minúsculas (case sensitive), por lo que no es lo mismo escribir `class` que escribir `Class`. Las palabras clave en Java siempre se escriben en minúscula.

Paso 3:

Ejecución del código objeto. Un intérprete recorre el código objeto y ejecuta las instrucciones que existen dentro de él una a una. Si un programa está libre de errores, aparecerá una ventana en al pantalla.

```
java HelloWorld //Java es el nombre del intérprete
```

El resultado de la ejecución sería:

```
Hello World
```

A continuación del nombre del intérprete (`java`), se debe escribir el nombre de la clase que contiene el programa principal (`main`) y no el nombre del fichero con su extensión.

El siguiente ejemplo sería un **error típico** en las primeras veces que se ejecuta un código Java:

```
java HelloWorld.class /* Nunca debe ponerse la extensión al ejecutar el
código */
```

Escribir, compilar y visualizar un applet

El applet tiene un aspecto algo diferente a la aplicación:

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

public class HelloWorldApplet extends Applet
{
    public void paint (Graphics g)
    {
        g.drawString("Hello World!!", 50, 25);
    }
}
```

No existe una sentencia `main()`. Utiliza los recursos gráficos del navegador en el que serán visualizadas. Las aplicaciones, si son de naturaleza gráfica, necesitan una programación explícita para crear la ventana en la que serán visualizadas.

También se utiliza el compilador `javac` para generar el bytecode, pero en vez de invocar el intérprete desde la línea de comandos se incluyen etiquetas HTML que hacen referencia al Applet.

```
<TITLE>Hello World Applet</TITLE>
<APPLET CODE="HelloWorldApplet.class" WIDTH=250 HEIGHT=80></APPLET>
```

1.9.- API

Java no es difícil por el lenguaje en sí, sino por su inmensa API. Requiere mucho tiempo familiarizarse con todos los paquetes y clases predefinidas. Hay unas 150 clases en el API núcleo (core): JDK (Java Development Toolkit). Hay muchas más clases definidas por Sun, y muchísimas más que se pueden encontrar en Internet.

Por tanto, la reusabilidad es muy alta. Pero la complejidad también.

1.10.-EVOLUCIÓN DE LAS VERSIONES DE JAVA

1991 – Proyecto Oak

1995 – Salida al mercado de Java, versión 1.0. Una pequeña versión centrada en la Web disponible uniformemente para todos los navegadores Web populares.

1997 – Java 1.1. Muchos más cambios que los que refleja la numeración. Mejoras en la interfaz de usuario, manipulación de eventos, reescrita totalmente, y una tecnología de componentes denominada *JavaBeans*.

1998 – Java 2 con SDK 1.2. Una versión ampliada significativamente, con características de interfaces gráficas de usuario, conectividad de bases de datos y muchas otras mejoras.

2000 – Java 2 con SDK 1.3. Multimedia mejorada, más accesibilidad y compilación más rápida.

2001 - Java 2 con SDK 1.4. Posibilidad de trabajar con XML.

Especificaciones en torno a Java

J2EE (Java 2 Enterprise Edition)

- Es un auténtico estándar, reconocido por la industria.
- Aplicaciones de propósito empresarial o para servidores.

J2ME (Java 2 MicroEdition)

- Estándar para dispositivos inalámbricos que requieren una integración en dispositivos con poco espacio físico y memoria de trabajo. Está haciendo que Internet llegue a dispositivos electrónicos de todo tipo.

Herramientas comerciales de desarrollo

- Symantec Visual Café
- Borland JBuilder
- IBM Visual Age for Java
- Sun Forte for Java

Especificación del lenguaje Java

- *Java Language Specification*, de James Gosling, Bill Joy y Guy Steele (Addison Wesley, 1996).
- www.sun.com
- www.javasoft.com

1.11.- EJERCICIOS

1. ¿Por qué Java es más portable que otros lenguajes de programación de alto nivel que conozcas? ¿Por qué es idóneo para programar en Internet?
2. ¿Cuáles de las siguientes afirmaciones son verdaderas y cuáles son falsas?
 - Cada clase debe tener un método `main`.
 - Las llaves (`{ }`) marcan el comienzo y final de un bloque.
 - No es posible anidar bloques.
 - Es posible utilizar caracteres de subrayado (`_`) en los identificadores.
 - Todas las sentencias terminan con el símbolo punto y coma (`;`).
3. ¿Qué relación existe entre los archivos `.java` y `.class` y entre los programas `javac` y `java`?