INTRODUCCIÓN A JAVA

1. HISTORIA

Una de las características para poder clasificar los lenguajes de programación es su nivel de abstracción. Este nivel puede ser expresado en base a la complejidad del problema que se está intentando resolver. Por ejemplo, el lenguaje Ensamblador, primer lenguaje de programación, tiene un pequeño nivel de abstracción relacionado totalmente con la máquina en la que se está ejecutando, por lo que el nivel de abstracción que se aplicaba al ámbito de la solución era muy bajo.

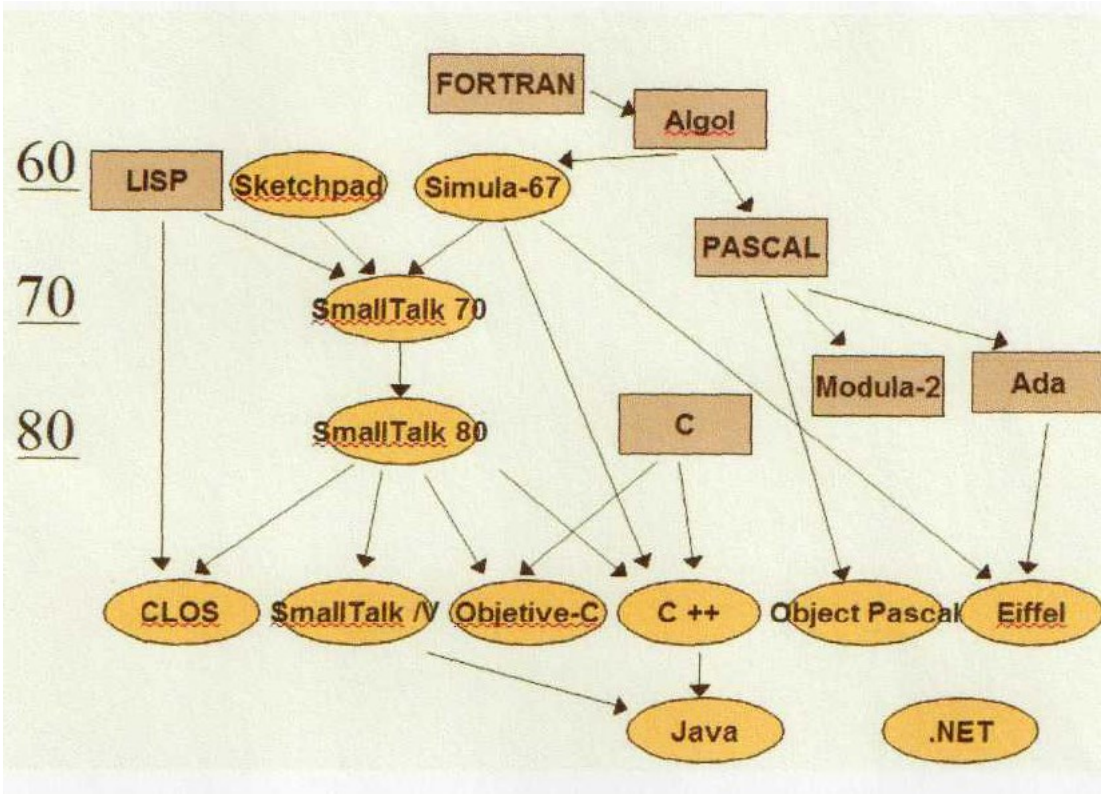
Muchos de los lenguajes que siguieron al Ensamblador, llamados lenguajes imperativos, como es el caso de Fortran, Basic y C, fueron abstracciones de este primer lenguaje de programación. El nivel de abstracción de estos lenguajes es mucho más elevado que el del lenguaje original, pero siguen estando muy relacionados con la estructura del ordenador en el que se ejecutan, en lugar de la estructura del problema a resolver y del mundo real. Debido a esta relación, los programas desarrollados son difíciles de escribir y bastante costosos de mantener.

Así, acercando el modelo de abstracción al problema a resolver y no a la máquina física, aparecieron en los años 60 los primeros lenguajes Orientados a Objetos, también denominados O.O., tales como LISP (todos los problemas se reducen a listas), APL (todos los problemas se reducen a algoritmos) y PROLOG (todos los problemas se reducen a cadenas de decisión).

El primer lenguaje considerado totalmente Orientado a Objetos y sobre el que se basa Java, es Smalltalk. Este lenguaje recoge las cinco principales características que tienen que tener estos lenguajes de programación.

- **Todo puede ser representado como un objeto**, siendo capaz de almacenar cierta información y realizar operaciones sobre ella.
- **Un programa es un conjunto de objetos colaborando entre sí**, indicando que es lo que hay que hacer mediante el envío de mensajes.
- **Cada objeto esta construido en base *a* otros objetos**, permitiendo alcanzar grados mayores de complejidad.
- **Cada objeto pertenece *a* un tipo**, denominado clase.
- **Todos los objetos del mismo tipo pueden recibir los mismos mensajes.**

A continuación, se muestra la evolución de los lenguajes Orientados a Objetos hasta la aparición del lenguaje Java, objeto de nuestro estudio.



2. CARACTERÍSTICAS DE JAVA

Sun Microsystems, la empresa propietaria de Java, no imaginó en 1991, año en que se desarrolló la primera versión del lenguaje, que unos diez años después, éste se iba a convertir en el lenguaje de programación más utilizado por la comunidad mundial de desarrolladores y, mucho menos, que la mayoría de los grandes fabricantes de software del momento IBM, Oracle, Borland, BEA, etc., desarrollarían sus productos para, de alguna u otra manera, dar soporte al lenguaje Java.

Java es, sin duda alguna, el lenguaje de programación que más impacto ha tenido en los últimos años, especialmente en el mundo de desarrollo para la Web. Probablemente, Internet no sería lo que es hoy sin la existencia de Java.

He aquí los principales puntos en los que se apoya la tecnología Java:

- **Lenguaje totalmente orientado a objetos.** Todos los conceptos en los que se apoya esta técnica, encapsulación, herencia, polimorfismo, etc., están presentes en Java.

- **Disponibilidad de un amplio conjunto de librerías.** Como ya se mencionó anteriormente, Java es algo más que un lenguaje. La programación de aplicaciones con Java se basa no sólo en el empleo del juego de instrucciones que componen el lenguaje, sino, fundamentalmente, en la posibilidad de utilizar el amplísimo conjunto de clases que Sun pone a disposición del programador y con las cuales es posible realizar, prácticamente, cualquier tipo de aplicación.

  En este amplio abanico, encontramos clases para la creación de interfaces gráficas, gestión de red, multitarea, acceso a datos y un largo etcétera.

- **Aplicaciones multiplataforma.** Ésta es, posiblemente, la característica más importante de Java y la que ha propiciado su amplia aceptación en la comunidad de desarrolladores y fabricantes software. Que las aplicaciones Java sean multiplataforma significa que, una vez se ha compilado el programa, éste puede ser ejecutado en diferentes sistemas operativos sin necesidad de realizar cambios en el código fuente y sin que haya que volver a compilar el programa, es lo que en el mundo Java se expresa con la frase "compila una vez y ejecuta en cualquier plataforma".

  Esta independencia de la plataforma se consigue gracias al concepto de **máquina virtual,** el cual trataremos con detalle en el siguiente punto.

- **Ejecución segura de aplicaciones.** La seguridad de las aplicaciones Java se manifiesta en varios aspectos. Por un lado, el lenguaje carece de instrucciones que puedan provocar accesos descontrolados a la memoria, éste es el caso de los punteros, una característica muy potente y peligrosa del lenguaje C/C++ que en Java no está presente. Por otro lado, la máquina virtual, que es el entorno en el que se ejecutan las aplicaciones Java, impone ciertas restricciones a las aplicaciones para garantizar una ejecución segura.

- **Amplio soporte de fabricantes software.** Esta característica se deriva en parte de las anteriores, sobre todo, del hecho de que los programas Java no estén vinculados a un determinado sistema operativo.

  Hoy en día, encontramos una amplia variedad de productos software

de diferentes fabricantes que dan soporte a Java, como puede ser el caso de los entornos de desarrollo o los servidores de aplicaciones.

3. LA MAQUINA VIRTUAL JAVA (JVM)

La **Máquina Virtual Java** o **JVM** es un entorno de ejecución para aplicaciones Java, cuya principal finalidad es la de adaptar los programas Java compilados a las características del sistema operativo donde se van a ejecutar.

En la figura 1 tenemos un esquema en el que se ilustra todo el proceso de compilación y ejecución de aplicaciones.

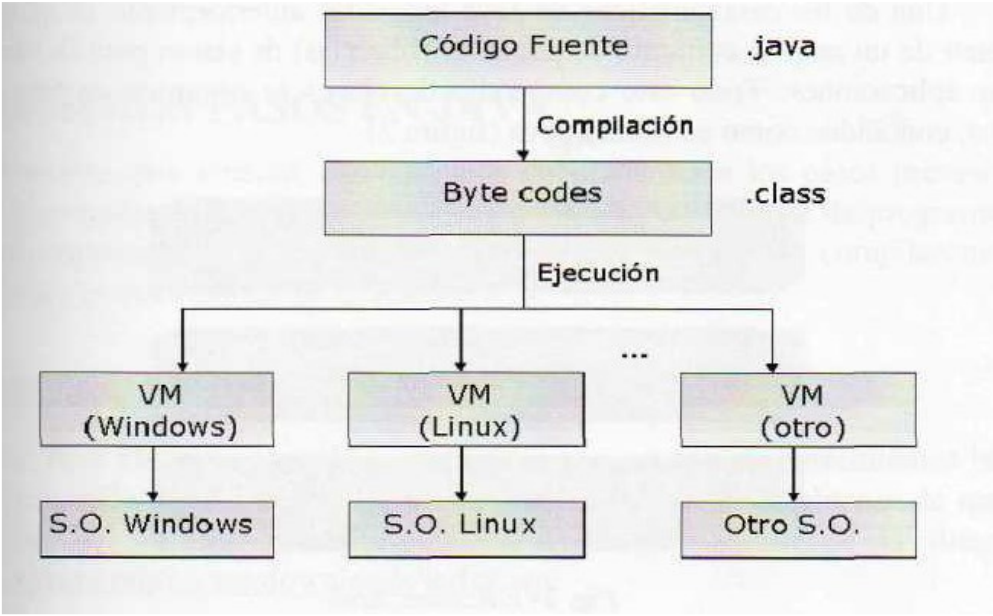


Fig. 1 - Proceso de compilación y ejecución de aplicaciones Java.

Todo programa Java está organizado en clases, éstas se codifican en archivos de texto con extensión .java. Cada archivo de código fuente .java puede c0ntener una o varias clases, aunque lo normal es que haya un archivo por clase.

Cuando se compila un .java se genera uno o varios archivos .class de código binario (uno por cada clase), denominados **bytecodes,** que son independientes de la arquitectura.

Esta independencia supone que los bytecodes no pueden ser ejecutados directamente por ningún sistema operativo, es durante la fase de ejecución cuando ios archivos .class se someten a un proceso de interpretación, consistente en traducir los bytecodes a código ejecutable por el sistema operativo. Esta operación es realizada por un software conocido como **Máquina Virtual Java.**

Cada sistema operativo proporciona su propia implementación de la **JVM,** todas ellas ofrecen el mismo "aspecto" de cara a los bytecodes, sin embargo, cada una realiza la interpretación de acuerdo a las características del sistema operativo para el que ha sido diseñada.

Hoy en día encontramos implementación de *máquina virtual para* la mayoría de los sistemas operativos existentes, en la mayoría de ellos la **JVM** es un componente más del propio sistema operativo.

4. EDICIONES JAVA

Una de las características de Java indicadas anteriormente es el hecho de disponer de un amplio conjunto de paquetes (librerías) de clases para la realización de las aplicaciones. Todo este compendio de clases se organiza en tres grandes grupos, conocidos como ediciones Java (figura 2).

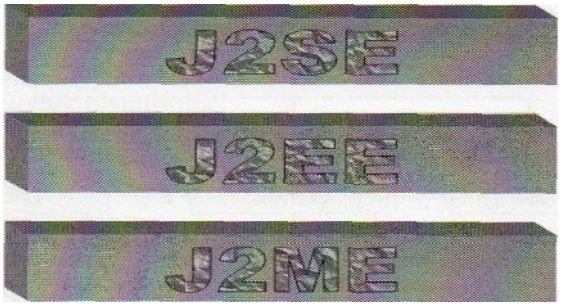


Fig. 2- Ediciones Java

Las tres ediciones en las que actualmente se organiza la tecnología Java

Java Standar Edition (Java SE). Forman parte de este grupo los paquetes de clases de uso general (tratamiento de cadenas, colecciones, acceso a datos, etc.), es decir, aquéllos que se utilizan en cualquier tipo de aplicación. Java SE incluye también los paquetes de clases para la creación de entornos gráficos y aplicaciones para navegadores Internet (applets). Esta edición será en la que nos centraremos durante esta parte del curso.

Java Enterprise Edition (Java EE). Proporciona los paquetes y tecnologías necesarias para la creación de aplicaciones empresariales multicapa, entre ellas, las aplicaciones que se van a ejecutar en entorno Web.

Java Micro Edition (Java ME). También los dispositivos electrónicos, tales como agendas electrónicas, PDAs o teléfonos móviles, pueden beneficiarse de la tecnología Java. Esta edición, incluye una serie de paquetes y especificaciones que posibilitan la creación de aplicaciones Java ejecutables en dispositivos electrónicos de capacidades limitadas.

5. PRIMEROS PASOS EN JAVA

• Creación del primer programa en Java

Aunque aún carecemos del conocimiento del lenguaje, vamos a presentar un primer programa Java, consistente en la impresión de un mensaje de saludo en la pantalla. Este programa nos va a servir para conocer el procedimiento general que se debe seguir para crear, compilar y ejecutar programas con Java estándar.

a) CODIFICACIÓN

Utilizando cualquier editor de texto, por ejemplo el bloc de notas, procederemos a escribir el código mostrado en la figura 9. Hay que tener en cuenta Java hace distinción entre mayúsculas y minúsculas, por lo que hay que codificarlo tal cual se muestra.

```
public class Saludo
{
    public static void main (String [ ] args)
    {
        System.out.println ("Bienvenido a Java");
    }
}
```

Fig. 9 - Programa para mostrar un texto de saludo

Después, procedemos a guardar este programa en un archivo de texto llamado Saludo.java (el nombre del archivo debe ser el mismo que se le ha dado a la clase). Si se está utilizando el bloc de notas, antes de guardar el archivo se debe elegir "Todos los archivos" en la opción "Tipo", dentro del cuadro de diálogo "Guardar", especificando en la opción "Nombre" el nombre del archivo y su extensión (figura 10).

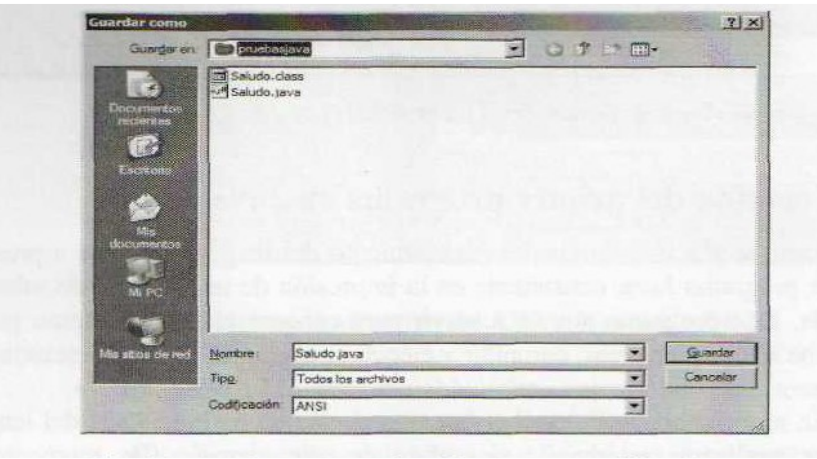


Fig. 10 - Guardar archivos de código Java con el bloc de notas

b) COMPILACIÓN

La compilación de un archivo de código fuente .java se realiza a través del comando ***javac.exe*** del **JDK**. Si se ha establecido correctamente la variable de entorno PATH, *javac* podrá ser invocado desde el directorio en el que se encuentre el archivo .java (figura 11). Tras ejecutar este comando, se generarán tantos archivos *.class* como clases existan en el código fuente, en este ejemplo se creará solamente ***Saludo. class.***

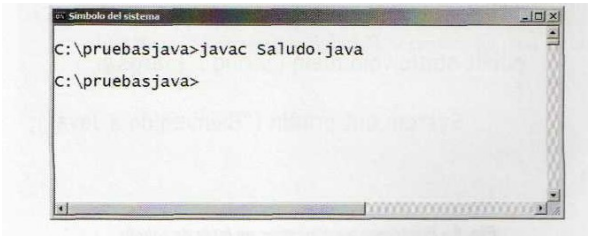


Fig. 11 - Compilación de un archivo de código fuente Java

En caso de que existan errores sintácticos en el código fuente, el compilador nos habría informado de ello y, por supuesto, el .class no se generaría. Por ejemplo, si en el código anterior cambiamos *System* por *system,* al intentar la compilación obtendríamos un mensaje de error como el indicado en la figura 12.

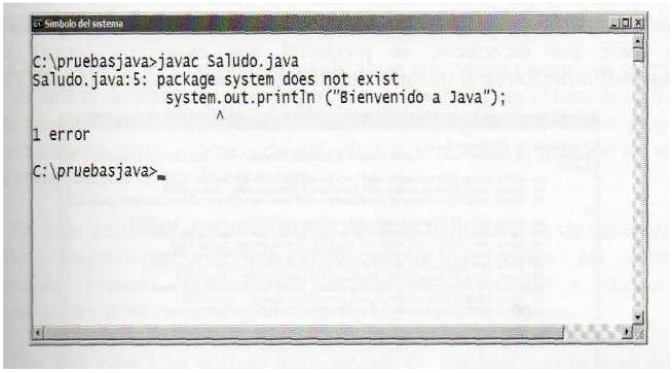


Fig. 12 - Error en la compilación de una clase

c) EJECUCIÓN

Para ejecutar el programa, utilizaremos el comando ***java.exe,*** seguido del de la clase que contiene el método ***main(),*** en nuestro caso será Saludo, que es la única clase existente. Es necesario que la variable de entorno CLASSPATH esté correctamente configurada e incluya el carácter **"."** (Directorio actual) en la direcciones, lo que permitirá invocar al comando java desde el directorio en £ encuentra el ,class (figura 13).

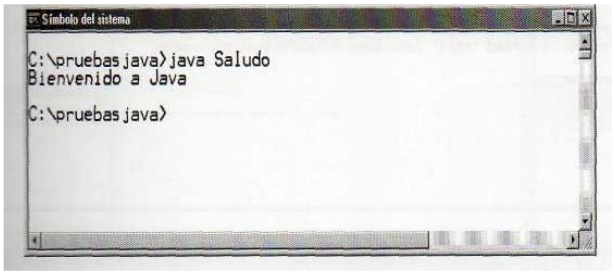


Fig. 13 - Ejecución de la clase principal

La llamada a **java.exe** insta a la máquina virtual a buscar en la clase indicada un método llamado *main()* y proceder a su ejecución.

En caso de que java.exe no encuentre la clase, bien porque la dirección del directorio actual (.) no figure en la variable CLASSPATH o bien porque el nombre de la clase sea incorrecto, se producirá una excepción (error) de tipo **NoClassDefFoundError** al intentar ejecutar el comando java.exe (figura 14).

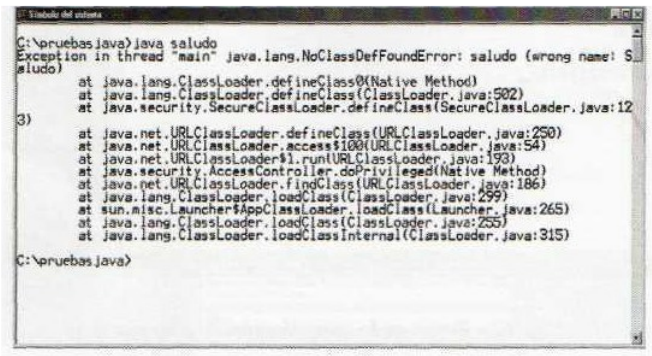


Fig. 14 - Error de ejecución de la clase

Si el problema no es la dirección de la clase, sino que el formato del método ***main()*** no es correcto, el programa compilará correctamente pero se producirá una excepción de tipo **NoSuchMethodError** (figura 15) al ejecutar el comando.

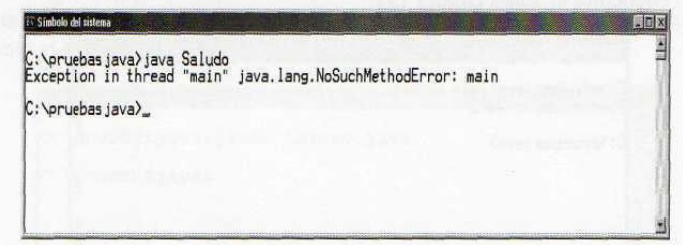


Fig. 15 - Si el formato del método main() no es correcto la JVM no lo encuentra