TEMA 4: EL LENGUAJE JAVA.

1. EVOLUCIÓN HISTÓRICA

Java surgió en 1991 cuando un grupo de ingenieros de *Sun Microsystems* trataron de diseñar un nuevo lenguaje de programación destinado a electrodomésticos (como vídeos, televisores, equipos de sonido, etc.). La reducida potencia de cálculo y memoria de los electrodomésticos llevó a desarrollar un lenguaje sencillo capaz de generar código de tamaño muy reducido.

Debido a la existencia de distintos tipos de CPUs y a los continuos cambios, era importante conseguir una herramienta independiente del tipo de CPU utilizada. Desarrollan un código "neutro" que no depende del tipo de electrodoméstico, el cual se ejecutaba sobre una "*máquina hipotética o virtual*" denominada ***Java Virtual Machine*** (***JVM***). Es la ***JVM*** quien interpreta el código neutro convirtiéndolo a código particular de la CPU utilizada. Esto permitía lo que luego se ha convertido en el principal lema del lenguaje: "*Write Once, Run Everywhere*". A pesar de los esfuerzos realizados por sus creadores, ninguna empresa de electrodomésticos se interesó por el nuevo lenguaje.

Como lenguaje de programación para computadores, ***Java*** se introdujo a finales de 1995. La clave fue la incorporación de un intérprete ***Java*** en la versión 2.0 del programa Netscape Navigator, produciendo una verdadera revolución en Internet. Comparativamente, Internet era como un gran conjunto de electrodomésticos, cada uno con un procesador diferente. Y es cierto, básicamente Internet es una gran red mundial que conecta múltiples ordenadores con diferentes sistemas operativos y diferentes arquitecturas de microprocesadores, pero todos tienen en común un navegador que utilizan para comunicarse entre sí. ***Java 1.1*** apareció a principios de 1997, mejorando sustancialmente la primera versión del lenguaje. ***Java 1.2***, más tarde fue rebautizado ***como Java 2***, nació a finales de 1998. Posteriormente han surgió las versiones ***1.3, 1.4, 1.5, 1.6*** y en Julio de 2011 la esperada versión 7 de ***Java 2 SDK, Standard Edition (J2SE)***. (Consultar anexo al tema).

Al programar en ***Java*** no se parte de cero. Cualquier aplicación que se desarrolle "cuelga" (o se apoya, según como se quiera ver) en un gran número de ***clases*** preexistentes. Algunas de ellas las ha podido hacer el propio usuario, otras pueden ser comerciales, pero siempre hay un número muy importante de clases que forman parte del propio lenguaje (el ***API*** o ***Application Programming Interface*** de ***Java***). ***Java*** incorpora 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.). Por eso muchos expertos opinan que ***Java*** es un lenguaje ideal para aprender la informática moderna, porque incorpora todos estos conceptos de un modo estándar, mucho más sencillo y claro que con las citadas extensiones de otros lenguajes. Esto es consecuencia de haber sido diseñado más recientemente y por un único equipo.

El principal objetivo del lenguaje ***Java*** es llegar a ser el "nexo universal" que conecte a los usuarios con la información, esté ésta situada en el ordenador local, en un servidor de ***Web***, en una base de datos o en cualquier otro lugar.

Java es un lenguaje muy completo (se está convirtiendo en un macro-lenguaje: ***Java 1.0*** tenía 12 packages; ***Java 1.1*** tenía 23, ***Java 1.2*** tenía 59, y continúa aumentando el número de paquetes en cada nueva versión). En cierta forma casi todo depende de casi todo. Por ello, hay que aprenderlo de modo iterativo: primero una visión muy general, que se va refinando en sucesivas iteraciones.

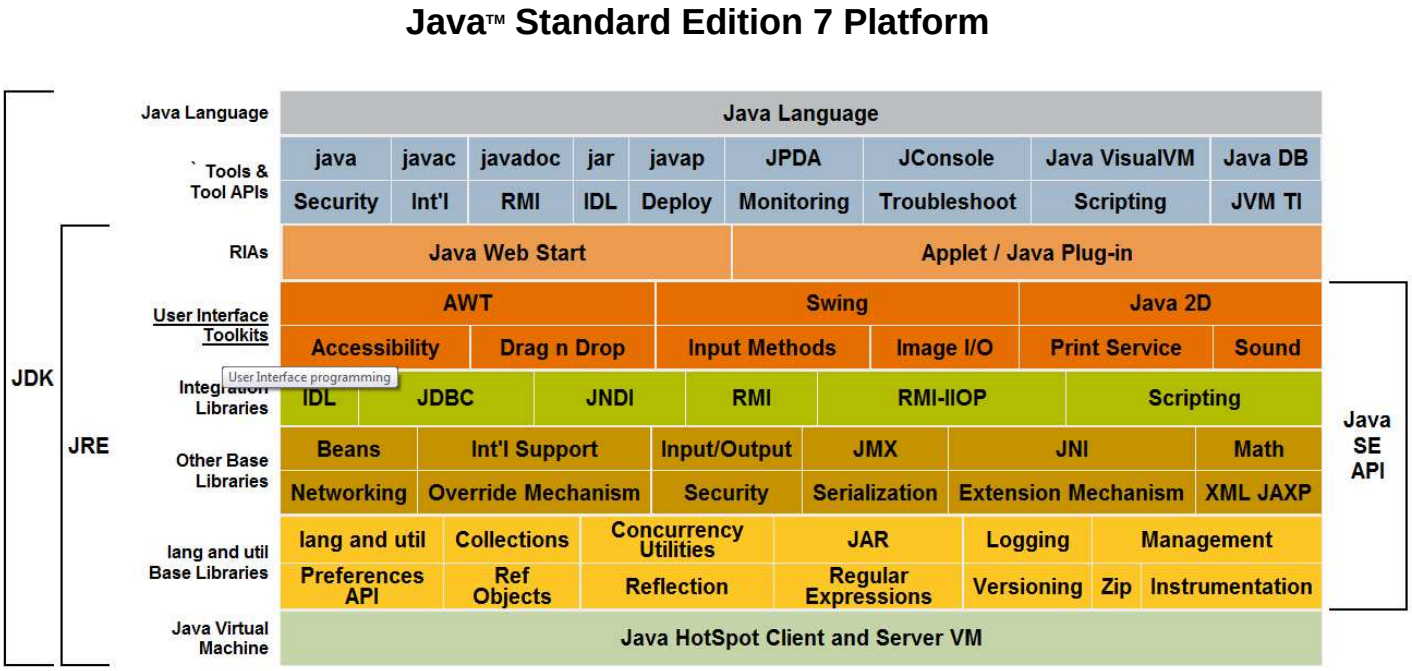
La compañía ***Sun*** describe el lenguaje ***Java*** como "*simple, orientado a objetos, distribuido, interpretado, robusto, seguro, de arquitectura neutra, portable, de altas prestaciones, multitarea y dinámico*". Además de una serie de halagos por parte de *Sun* hacia su propia criatura, el hecho es que todo ello describe bastante bien el lenguaje ***Java***.

En abril de 2009 llega la noticia del anuncio oficial de venta de Sun Microsystems al gigante empresarial **ORACLE** por 7.400 millones de dólares. Las especulaciones sobre el futuro de los productos y servicios ofrecidos por Sun Microsystems son muchas e inciertas. Con esta adquisición Oracle cuenta con las soluciones más completas en el mundo IT, ofreciendo: el DBMS empresarial "Oracle", DBMS mediano "MySQL", Lenguaje de programación JAVA, Plataforma de desarrollo "NetBeans" y "JDeveloper", SOA, Oracle WebLogic , Oracle BPM Suite, SPARC, Sistema Operativo Server "SOLARIS", Software de virtualización "VirtualBox", OpenOffice…

2. ¿QUÉ ES JAVA 2?

Java 2 (antes llamado ***Java 1.2*** o ***JDK 1.2***) es la tercera versión importante del lenguaje de programación ***Java***. No hay cambios conceptuales importantes respecto a ***Java 1.1*** (en ***Java 1.1*** sí los hubo respecto a ***Java 1.0***), sino extensiones y ampliaciones, lo cual hace que a muchos efectos sea casi lo mismo trabajar con ***Java 1.1*** o con ***Java 1.2***.

Nosotros trabajaremos con el kit de desarrollo de java (**JDK**) - **J2SE Development Kit** - en la versión última que se encuentre disponible, que es un regalo gratuito de *Sun* (ahora Oracle) a la comunidad que programa en Java.



Los programas desarrollados en ***Java*** presentan diversas ventajas frente a los desarrollados en otros lenguajes como C/C++. La ejecución de programas en ***Java*** tiene muchas posibilidades: ejecución como aplicación independiente (***Stand-alone Application***), ejecución como ***applet***, ejecución como ***servlet***, etc.

Un ***applet*** es una aplicación especial que se ejecuta dentro de un navegador o browser al cargar una página HTML desde un servidor ***Web***. El ***applet*** se descarga desde el servidor y no requiere instalación en el ordenador donde se encuentra el browser. Un applet es independientemente del sistema operativo y de la arquitectura del ordenador, para ejecutarlo sólo se necesita un navegador que soporte la ***JVM*** como, por ejemplo, *Netscape Navigator* o *Internet Explorer*

Un ***servlet*** es una aplicación sin interface gráfica que se ejecuta en un servidor de Internet. Los clientes pueden invocarlo utilizando el protocolo HTTP. Comparativamente, lo mismo que un *applet* es cargado y ejecutado por un explorador, un *servlet* es cargado y ejecutado por un servidor Web. Si

recordamos un poco, una de las primeras formas que se encontraron para dar dinamismo a la páginas HTML fue la CGI (*Common Gateway Interface*). Una alternativa a la CGI fue ISAPI. Posteriormente, estas técnicas fueron sustituidas por la incorporación de secuencias de órdenes (scripts) en JavaScript o VBScript, por páginas ASP de Microsoft, páginas JSP (Java Server Page – Java activada en el Servidor), PHP, etc. Las JSP no son interpretadas por el motor de JSPs, sino que son compiladas de manera automática la primera vez que se solicita una página. El resultado de la compilación es un ***servlet***.

Además de incorporar la ejecución como ***Applet***, ***Java*** permite fácilmente el desarrollo tanto de arquitecturas cliente-servidor como de aplicaciones distribuidas, consistentes en crear aplicaciones capaces de conectarse a otros ordenadores y ejecutar tareas en varios ordenadores simultáneamente, repartiendo por lo tanto el trabajo. Aunque también otros lenguajes de programación permiten crear aplicaciones de este tipo, ***Java*** incorpora en su propio ***API*** estas funcionalidades.

3. CARACTERÍSTICAS DE JAVA

Veamos algunas de las principales características de JAVA:

- ***Simple.*** Está basado en C++, pero hay que destacar que no existen punteros ni aritmética de punteros, las cadenas de caracteres son objetos y la administración de memoria es automática, lo que elimina la problemática que presenta C++ con las lagunas de memoria al olvidar liberar bloques de la misma que fueron asignados dinámicamente.

- ***Orientado a Objetos***. Incluye:
  - Encapsulación, herencia y polimorfismo.
  - Interfaces para suplir la carencia de herencia múltiple.
  - Resolución dinámica de métodos.

- ***Distribuido.***
  - Extensas capacidades de comunicaciones.
  - Permite actuar con http y ftp.
  - El lenguaje no es distribuido, pero incorpora facilidades para construir programas distribuidos.
  - Se están incorporando características para hacerlo distribuido.

- ***Robusto.*** Realiza continuos chequeos en tiempo de compilación y en tiempo de ejecución. Chequeos de punteros nulos, chequeo de límite en vectores, excepciones, verificación de código binario, recolección automática de basuras, etc.)

- ***Seguro.***
  - No hay punteros
  - El *cast* (promoción) hacia lo general es implícito.
  - Los *bytecodes*, (ficheros compilados con extensión .class) pasan varios test antes de ser ejecutados.

- ***Arquitectura neutral*** (mismo código en distintas arquitecturas).
  - Define la longitud de sus tipos independientemente de la plataforma.
  - Construye sus interfaces en base a un sistema abstracto de ventanas.

- ***Interpretado.***
  - Para conseguir la independencia del S.O. genera bytecodes.
  - El intérprete toma cada bytecode y lo interpreta.
  - El mismo intérprete corre en distintas arquitecturas.

- ***Multihebras.***
  - Permite crear tareas o hebras.
  - Sincronización de métodos.

  - Comunicación de tareas.

- ***Dinámico*.**
  - Conecta los módulos que intervienen en una aplicación en el momento de su ejecución
  - No hay necesidad de enlazar previamente.

4. EL ENTORNO DE DESARROLLO DE JAVA

Para poder realizar programas en lenguaje Java, es necesario disponer de un mínimo de herramientas que nos permitan editar, compilar e interpretar el código que diseñamos. Para escribir físicamente los programas, podemos utilizar cualquier editor de textos como por ejemplo, el bloc de notas, el WordPad, etc., o algún editor un poco más especializado, como por ejemplo el *EditPlus*. Para compilar y ejecutar los programas existen dos opciones:

- Utilizar un entorno integrado de desarrollo (***IDE*** - *Integrated Development Environment).* Un IDE es una aplicación que reúne varios programas necesarios para el desarrollador, en uno solo, en el caso, de Java: editor, compilador, depurador, etc.. Ejemplos de IDE's son, *JBuilder* de *Borland, Visual J++* de *Microsoft, Visual Café de Symantec, Kawa de Tek-Tools, Visual Age Windows de IBM, pcGRASP de Auburn University, Forte de Sun,* NetBeans de Oracle, Eclipse, etc., destacar que estos dos últimos son gratuitos.
- Emplear el software básico de desarrollo (SDK) de Oracle.

La primera opción resulta especialmente interesante para afrontar la creación de aplicaciones de manera eficiente, puesto que estos entornos facilitan enormemente el diseño, escritura y depuración de los programas. La segunda opción es mucho más adecuada para aprender a programar en Java, porque no existe la generación automática de código que incorporan los entornos integrados de desarrollo.

La compañía Oracle (anteriormente ***Sun)***, creadora de ***Java***, distribuye gratuitamente el *Java(tm) Development Kit* (***JDK***), también es conocido como ***SDK*** (*Java Software Development*). Se trata de un conjunto de programas y librerías que permiten desarrollar, compilar y ejecutar programas en ***Java***.

Incorpora además la posibilidad de ejecutar parcialmente el programa, deteniendo la ejecución en el punto deseado y estudiando en cada momento el valor de cada una de las variables (es el denominado ***Debugger***). Cualquier programador con un mínimo de experiencia sabe que una parte muy importante (muchas veces la mayor parte) del tiempo destinado a la elaboración de un programa se destina a la detección y corrección de errores. Existe también una versión reducida del ***JDK***, denominada ***JRE*** (*Java Runtime Environment*) destinada únicamente a ejecutar código ***Java*** (no permite compilar).

Las herramientas más importantes que suministra el JDK son:

Herramientas	**Uso**
javac	Compilador java.
java	Interprete Java, utilizado para ejecutar programas compilados.
appletviewer	Utilizado para visualizar el *applet* tal como puede ser visto por el navegador.
jdb	Depurador.
Javadoc	Generador de documentación.

Con los ***IDE***s, en un mismo programa es posible escribir el código ***Java***, compilarlo y ejecutarlo sin tener que cambiar de aplicación. Algunos incluyen una herramienta para realizar *debug (depurador)* gráficamente. Estos entornos integrados permiten desarrollar las aplicaciones de forma mucho más rápida, incorporando en muchos casos librerías con ***componentes*** ya desarrollados, los cuales se incorporan al proyecto o programa. Como inconvenientes se pueden señalar algunos fallos de compatibilidad entre plataformas y ficheros resultantes de mayor tamaño que los basados en clases estándar.

4.1. ¿Qué utilizar para empezar a programar en Java?

Para empezar a desarrollar programas en Java necesitaré:

- Instalar y configurar el entorno de desarrollo (SDK) y utilizar un editor (editplus, wordpad, bloc de notas, …) para escribir mis programas.

o bien

- Instalar un IDE que ya incorpora el entorno de desarrollo, el editor, … y normalmente configura todo lo necesario.

Como ya se ha mencionado anteriormente, para desarrollar programas Java bastaría con el Bloc de notas, pero es un proceso engorroso que se ve facilitado si se usa un editor de texto que permita analizar y realzar la sintaxis con colores, de esta manera el trabajo es más cómodo y productivo en términos de tiempo y esfuerzo.

De los muchos editores existentes, EditPlus es un editor de texto sencillo y potente con capacidad de realzar la sintaxis de numerosos lenguajes de programación. EditPlus soporta análisis y realzado de sintaxis de una forma potente y configurable. Por defecto, el programa trae incorporada esa función para determinados tipos de archivos: HTML, CSS, PHP, C/C++, Java, JavaScript, Perl y VBScript.

El uso de EditPlus como editor de código fuente Java puede mejorarse con facilidad extendiendo sus herramientas de usuario (User tools) para compilar y ejecutar los programas Java desde el propio editor de texto.

En cuanto a los IDE's es difícil decidirse por uno o por otro, ya que se elija el que se elija, siempre habrá defensores y detractores del mismo. Para finalizar este apartado destacar que hay dos grandes IDE's **gratuitos** que se están teniendo mucha aceptación actualmente: NetBeans y Eclipse. Nosotros durante este curso vamos a utilizar **Eclipse**.

4.2. Instalación y configuración

Ver documento "Instalación y configuración de Java". La propia instalación rellenará la variable de entorno ***CLASSPATH*** con los valores adecuados (donde el entorno debe buscar los ficheros compilados *.class*), y la variable de entorno **PATH** para que el sistema operativo encuentre los ficheros ejecutables del JDK. Se profundizará sobre estas variables cuando se vayan realizando las prácticas.

A continuación, veremos brevemente lo que contiene cada una de las carpetas que se han creado dentro del directorio de java:

- ✓ La carpeta ***bin*** contiene las herramientas de desarrollo. Esto es los programas para compilar, ejecutar, depurar y documentar, y otras herramientas como *appletviewer*, *jar* para manipular ficheros .jar (un fichero .jar es una colección de clases Java y otros ficheros empaquetados en uno solo), *javah* que es un fichero de cabecera para escribir métodos nativos, *javap* para descompilar ficheros compilados y *extcheck* para detectar confictos *jar*.
- ✓ La carpeta ***jre*** es el entorno de ejecución de java utilizado por el SDK. Es similar al interprete de Java, pero destinado a usuarios finales quo no requieran todas las opciones de desarrollo proporcionadas con la utilidad java. Incluye la maquina virtual, la biblioteca de clases, y otros ficheros quo soportan la ejecución de programas escritos en Java.
- ✓ La carpeta ***lib*** contiene bibliotecas de clases adicionales y ficheros de soporte requeridos por las herramientas de desarrollo.
- ✓ La carpeta ***demo*** contiene ejemplos.
- ✓ La carpeta ***include*** contiene los ficheros de cabecera que dan soporte para añadir a un programa Java código nativo (escrito en un lenguaje distinto de Java, por ejemplo Ada o C++).
- ✓ También incluye en un fichero ***src.zip*** el código fuente de todas las clases que forman el núcleo de java.

4.3. El compilador de Java.

Se trata de una de las herramientas de desarrollo incluidas en el ***JDK***. Realiza un análisis de sintaxis del código escrito en los ficheros fuente de ***Java*** (con extensión ****.java***). Si no encuentra errores en el código genera los ficheros compilados (con extensión ****.class***). En otro caso muestra la línea o líneas erróneas. En el ***JDK*** de ***Sun*** dicho compilador se llama ***javac.exe***. Tiene numerosas opciones, algunas de las cuales varían de una versión a otra. Se aconseja consultar la documentación de la versión del ***JDK*** utilizada para obtener una información detallada de las distintas posibilidades.

Veamos a continuación un ejemplo para probar el compilador:

```
1      // Aplicacion Hola mundo
2      class PrimerEjemplo
3      {
4        public static void main(String [] args)
5         {
6            System.out.println("Hola mundo");
7         }
8      }
```

Introducir el código anterior sin los números de línea con un editor de texto, en un fichero al que llamaremos ejemplo1.java, y que almacenaremos en una carpeta que previamente habremos creado para colocar nuestros programas en java. Una vez editado lo compilaremos de la siguiente forma:

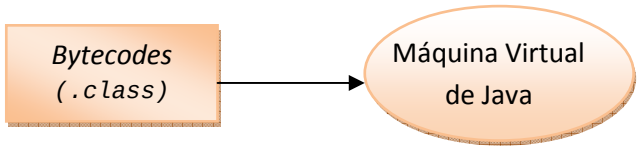
```
javac ejemplo1.java
```

Si existen errores, se obtendrá un listado de ellos. Si no los hay, no aparece nada, y habremos obtenido el fichero `PrimerEjemplo.class`.

Hagamos ahora una breve descripción del ejemplo, aunque ya se estudiará todo en detalle un poco más adelante. La línea 1 es un comentario, la línea 2 define la clase PrimerEjemplo, la línea 3 es una llave de comienzo que indica el comienzo de la clase, la cual termina en la línea 8. La línea 4 define un método de la clase PrimerEjemplo, este método es especial y le indica a Java el comienzo de nuestra aplicación, su nombre es *main* y siempre lleva un parámetro *String []*, el método es público y estático. El contenido del método *main*, se encuentra delimitado entre la llave de inicio (línea 5) y la llave de fin (línea 7), y únicamente contiene la instrucción situada en la línea 6, que imprime el texto que aparece entre comillas. *println* es un método del objeto *out*, de la clase *System.*

4.4. El intérprete de Java: La Java Virtual Machine (JVM)

Tal y como ya se ha comentado, la existencia de distintos tipos de procesadores y ordenadores llevó a los ingenieros de ***Sun*** a la conclusión de que era muy importante conseguir un software que no dependiera del tipo de procesador utilizado. Se plantea la necesidad de conseguir un código capaz de ejecutarse en cualquier tipo de máquina. Una vez compilado no debería ser necesaria ninguna modificación por el hecho de cambiar de procesador o de ejecutarlo en otra máquina. La clave consistió en desarrollar un código "neutro" el cual estuviera preparado para ser ejecutado sobre una "*máquina hipotética o virtual*", denominada ***Java Virtual Machine*** (***JVM***). Es esta ***JVM*** quien ***interpreta*** este código neutro convirtiéndolo a código particular de la CPU o chip utilizada. Se evita tener que realizar un programa diferente para cada CPU o plataforma.



La ***JVM*** es el intérprete de ***Java*** y se llama *java.exe*. Ejecuta los "***bytecodes***" (ficheros compilados con extensión ****.class***) creados por el compilador de ***Java***. Tiene numerosas opciones entre las que destaca la posibilidad de utilizar el denominado ***JIT*** (*Just-In-Time Compiler*), que puede mejorar entre 10 y 20 veces la velocidad de ejecución de un programa. Un compilador JIT interacciona con la máquina virtual para convertir los *bytecodes* en código máquina nativo.

Ya podemos ejecutar nuestro ejemplo, para ello:

```
java PrimerEjemplo
```

Si todo ha ido bien nos habrá salido por pantalla "Hola mundo".

Hoy en día casi todas las compañías de sistemas operativos y de navegadores han implementado máquinas virtuales según las especificaciones publicadas por Sun Microsystems, para que sean compatibles con el lenguaje Java. Para los *applets* la maquina virtual está incluida en el navegador y para las aplicaciones Java convencionales, puede venir en el sistema operativo, con el paquete Java, o bien se puede obtener a través de Internet.

5. NOMENCLATURA HABITUAL EN LA PROGRAMACIÓN EN JAVA.

Los nombres de ***Java*** son sensibles a las letras mayúsculas y minúsculas. Así, las variables ***masa***, ***Masa*** y ***MASA*** son consideradas variables completamente diferentes. Las reglas del lenguaje respecto a los nombres de variables son muy amplias y permiten mucha libertad al programador, pero es habitual seguir ciertas normas que facilitan la lectura y el mantenimiento de los programas de ordenador. Se recomienda seguir las siguientes instrucciones:

1. En ***Java*** es habitual utilizar nombres con minúsculas, con las excepciones que se indican en los puntos siguientes.
2. Cuando un nombre consta de varias palabras es habitual poner una a continuación de otra, poniendo con mayúscula la primera letra de la palabra que sigue a otra (Ejemplos:*elMayor()*, *ventanaCerrable*, *rectanguloGrafico*, *addWindowListener()*).
3. Los nombres de ***clases*** e ***interfaces*** comienzan siempre por mayúscula (Ejemplos: *Geometria*, *Rectangulo*, *Dibujable*, *Graphics*, *Vector*, *Enumeration*).
4. Los nombres de ***objetos***, los nombres de ***métodos*** y ***variables miembro***, y los nombres de las ***variables locales*** de los métodos, comienzan siempre por minúscula (Ejemplos: *main()*, *dibujar()*, *numRectangulos*, *x*, *y*, *r*).
5. Los nombres de las ***variables finales***, es decir de las constantes, se definen siempre con mayúsculas (Ejemplo: *PI*)

6. ESTRUCTURA GENERAL DE UN PROGRAMA JAVA.

La estructura habitual de un programa realizado en cualquier lenguaje ***orientado a objetos*** u ***OOP*** (*Object Oriented Programming*), y en particular en el lenguaje ***Java*** es la siguiente: aparece una clase que contiene el programa principal (aquel que contiene la función ***main()***) y algunas clases de usuario (las específicas de la aplicación que se está desarrollando) que son utilizadas por el programa principal. Los ficheros fuente tienen la extensión ****.java***, mientras que los ficheros compilados tienen la extensión ****.class***.

Un fichero fuente (***.java***) puede contener más de una clase, pero sólo una puede ser ***public***. El nombre del fichero fuente debe coincidir con el de la clase ***public*** (con la extensión ****.java***). Si por ejemplo en un fichero aparece la declaración (***public class MiClase {...}***) entonces el nombre del fichero deberá ser ***MiClase.java***. Es importante que coincidan mayúsculas y minúsculas ya que ***MiClase.java*** y ***miclase.java*** serían clases diferentes para ***Java***. Si la clase no es ***public***, no es necesario que su nombre coincida con el del fichero. Una clase puede ser ***public*** o ***package*** (default), pero no ***private*** o ***protected***. Estos conceptos se explican posteriormente.

De ordinario una aplicación está constituida por varios ficheros ****.class***. Cada clase realiza unas funciones particulares, permitiendo construir las aplicaciones con gran modularidad e independencia entre clases. La aplicación se ejecuta por medio del nombre de la clase que contiene la función ***main()*** (sin la extensión ****.class***). Las clases de ***Java*** se agrupan en ***packages***, que son librerías de clases. Si las clases no se definen como pertenecientes a un ***package***, se utiliza un ***package*** por defecto (*default*) que es el directorio activo. Los ***packages*** se estudian con más detenimiento en siguientes apartados.

Para familiarizarnos un poco con todo esto, vamos a probar el siguiente ejemplo que consta de dos clases la clase “EjCadena” y la clase “Cadena”:

```
// fichero EjCadena.java
public class EjCadena {
   public static void main(String args[]) {
       Cadena cad=new Cadena(args[0]);
       cad.invierteCadena();
       cad.visualizaCadena();
       cad.encriptaCadena();
       cad.visualizaCadena();
       cad.desencriptaCadena();
       cad.visualizaCadena();
   }
}

// fichero Cadena.java
public class Cadena {
   public String cadena="";
   public Cadena(String cadena) {
       this.cadena=cadena;
   }
   public int ordinal(char c) {
       return ((int) c);
   }
   public char ascii(int i) {
       return ((char) i);
   }
   public void invierteCadena() {
       String cadena2="";
       for(int i=cadena.length()-1;i>=0;i--)
          cadena2=cadena2+cadena.charAt(i);
       cadena=cadena2;
   }
   public void encriptaCadena() {
       String cadena2="";
       char c;
       for(int i=0;i<cadena.length();i++) {
          c=cadena.charAt(i);
          cadena2=cadena2+ascii(ordinal(c)+3);
       }
       cadena=cadena2;
   }
   public void desencriptaCadena() {
       String cadena2="";
       char c;
       for(int i=0;i<cadena.length();i++) {
          c=cadena.charAt(i);
          cadena2=cadena2+ascii(ordinal(c)-3);
       }
```

```
            cadena=cadena2;
        }
        public void visualizaCadena() {
            System.out.println(cadena);
        }
    }
```

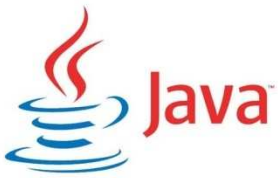
Una vez que esté funcionando, es decir tengamos los ficheros Cadena.class y EjCadena.class para probar su funcionamiento con la cadena “Hola” habrá que hacer:

Desde MS-DOS:

```
java EjCadena “Hola”
```

Desde Eclipse:

Menu Run – Run Configurations – Pestaña Arguments



ANEXO: VERSIONES DE JAVA

- Java 1
    - El 23 Mayo de 1995, en la conferencia SunWorld '95, John Gage, de Sun Microsystems, y Marc Andreessen, cofundador y vicepresidente de Netscape, anunciaban la versión ***alpha*** de Java, que en ese momento solo corría en Solaris, y el hecho de que Java iba a ser incorporado en Netscape Navigator, el navegador mas utilizado de Internet.
    - Con la segunda alpha de Java en Julio, se añade el soporte para Windows NT y en la tercera, en Agosto, para Windows 95. En Enero de 1996, Sun crea JavaSoft para desarrollar la nueva tecnología y ese mismo mes aparece la versión 1.0 del JDK.

    - Java 1.0 (Enero 1996) - 8 paquetes, 212 clases - Primera versión pública. La presión hizo que se hiciera pública demasiado pronto, lo cual significa que el diseño del lenguaje no es demasiado bueno y hay montones de errores. Respecto a seguridad, es restrictivo por defecto, no dejando hacer demasiado al código no fiable.
    - Java 1.1 (Marzo 1997) - 23 paquetes, 504 clases - mejoras de rendimiento en la JVM, nuevo modelo de eventos en AWT, clases anidadas, serialización de objetos, API de JavaBeans, archivos jar, internacionalización, API Reflection (Reflexión), JDBC (Java Data base Connectivity), RMI (Remote Method Invocation). Se añade la firma del código y la autentificación. Es la primera versión lo suficientemente estable y robusta.
- Java 2
    - Java 1.2 (Diciembre 1998) - 59 paquetes, 1520 clases - JFC (Swing), Drag and Drop, Java2D, Corba, API Collections. Se producen notables mejoras a todos los niveles. Para enfatizar esto Sun lo renombra como "Java 2". El JDK (Java Development Kit) se renombra como SDK (Software Development Kit). Se divide en J2SE, J2EE y J2ME.
    - Java 1.3 (Abril 2000) - 77 paquetes, 1595 clases - Orientada sobre todo a la resolución de errores y a la mejora del rendimiento; se producen algunos cambios menores como la inclusión de JNDI (Java Naming and Directory Interface) y la API Java Sound. También incluye un nuevo compilador de alto rendimiento JIT (Just In Time).
    - Java 1.4 (2002) - 103 paquetes, 2175 clases - También conocido como Merlin. Mejora notablemente el rendimiento y añade entre otros soporte de expresiones regulares, una nueva API de entrada/salida de bajo nivel (NIO, New I/O), clases para el trabajo con Collections, procesado de XML; y mejoras de seguridad como el soporte para la criptografía mediante las Java Cryptography Extension (JCE), la inclusión de la Java Secure Socket Extension (JSSE) y el Java Authentication and Authorization Service (JAAS).
    - Java 1.5 (Octubre 2004) - 131 paquetes, 2656 clases - También conocido como Tiger, renombrado por motivos de marketing como Java 5.0. Incluye como principales novedades:
        - tipos genéricos (generics)
        - autoboxing/unboxing conversiones implícitas entre tipos primitivos y los wrappers correspondientes.
        - Enumerados
        - Bucles simplificados
        - printf
        - Funciones con número de parámetros variable.
        - Metadatos en clases y métodos.

- Java 1.6 (Diciembre 2006) también conocida como Mustang, incluye como mejoras o novedades las siguientes:
  - Mejoras de seguridad (incluyendo soporte para el nuevo Smart Card I/O de Java, los servicios criptográficos/PKI de Microsoft en Windows, y autenticación LDAP entre otros).
  - Diagnosis mejorada a la hora de detectar problemas en la gestión de memoria
  - Mejoras en la velocidad de los componentes Swing/AWT (incluyendo soporte para APIs nativas en el SO)
  - Integración de una API para los servicios Web (nuevo cliente y la denominada Core Java Architecture para las APIs XML-Web Services 2.0, nuevo soporte para la arquitectura Java para XML Binding 2.0, etc.)
  - Soporte de scripting - lo que permite el soporte para combinar JavaScript (o tu propio plug-in con un motor de scripting) dentro del código fuente de Java.

  Aunque un gran número de esas nuevas características se usan en una gran parte del desarrollo de Java por parte de la comunidad, probablemente no existen demasiadas funcionalidades novedosas que aseguren que los desarrolladores la utilicen inmediatamente. No ocurrió así en el caso de la versión 5.0, que fue utilizada con anticipación masivamente por parte de desarrolladores y programadores. Java SE 7.0, con nombre clave Dolphin, ya se encuentra en fase de desarrollo.

- Java 1.6 Update 4 (14-01-2008). **Sun** libera una nueva versión de **JAVA SE 6**, siendo esta la **Update 4**, bajo los nombres técnicos propuestos por la empresa. Esta actualización, solventa fallos en unos 370 errores reportados, incluyendo algunos relacionados con la seguridad propia del software.

  Es recomendable actualizar, dado que existen problemas graves de seguridad, que podrían llevar a una caída del sistema, y dado que de momento se desconoce si los mismos podrían ser usados de forma remota, es conveniente mantenerse protegidos.

- Java 1.6 Update 12 (Diciembre 2008). **Sun** ha lanzado una nueva versión **Java SE 6 Update 12** y su código incluye dos nuevas características, un plug-in para navegadores de 64 bits y soporte para **Windows 2008**. Según Sun, el plug-in para **navegadores de 64 bits** se ha desarrollado después de que las personas que han estado utilizando sistemas operativos de 64 bits se quejaban de su limitación de navegadores de 32 bits.

- Java 1.6 Update 18 (Enero 2010). Algunas de las características que aporta esta nueva versión son: actualizaciones a JavaDB, visualVM 1.2, hotspot VM 16.0, mejora en la velocidad de creación de archivos jar, habilidad para leer archivos zip de mayor tamaño (hasta 4 gb), se incluye también una amplia lista de bugfixes, etc. Para más información: http://java.sun.com/javase/6/webnotes/6u18.html.

- **Java 7** (29 de Julio 2011). **La versión final de Java 7 ha sido publicada** de forma oficial por Oracle, tras más de 4 años desde la última actualización. Comenta Mark Reinhold (Arquitecto Jefe de Oracle) en su blog, que se han corregido 9.494 bugs, se han implementado 1.966 mejoras y 9.018 cambios. Toda una revisión en profundidad.

  Pero Java 7 es algo más que una colección de errores corregidos. La nueva versión ha realizado cambios en el lenguaje para facilitar el trabajo de los programadores, ahora dispone de una nueva API del sistema de archivos y un marco de trabajo nuevo enfocado al trabajo con procesadores de varios núcleos.

  Java 7 permite utilizar otros lenguajes, (tales como Ruby), gestión automática de recursos, mejoras en la liberación de la memoria de recursos, compatibilidad con el estándar Unicode

6.0, un nuevo interfaz visual y otro conjunto de funcionalidades que agradecerán los programadores.

Java despierta tantas pasiones a favor como en contra, pero aquí está desde principios de los 90, aunque le han dado por muerto en muchas ocasiones. Publicaciones especializadas en programación, como Genbeta Dev desgranarán hasta el último detalle esta importante actualización que va a permitir la creación de aplicaciones mejores más en consonancia con las necesidades actuales. Cuatro años en este sector, es mucho tiempo. **Java 7 está disponible para Windows, Linux y Solaris**, la versión para Mac OS X aún no está lista.