UT 1: IDENTIFICACIÓN DE UN PROGRAMA INFORMÁTICO

DESCRIPCIÓN BREVE

El objetivo del curso será aprender, dado un problema, a diseñar una solución aplicando el paradigma de la programación orientada a objetos y expresando dicha solución en el lenguaje de programación Java. En este tema de introducción aprenderemos algunos conceptos importantes en relación con la programación de ordenadores, la evolución histórica que han tenido los lenguajes de programación y las razones por las que hoy en día la orientación a objetos se aplica en el desarrollo de software.

César Díaz

1º DAW | Programación

Tabla de contenido

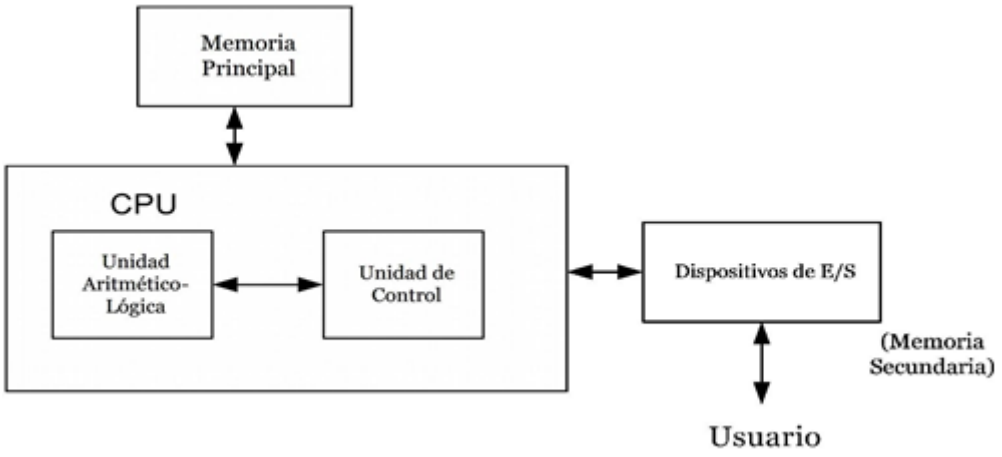
1.1 ESTRUCTURA DE UN ORDENADOR

La arquitectura básica de un ordenador digital se basa en la arquitectura de una máquina descrita por Von Newman en 1944. La arquitectura Von Newmann divide a un ordenador en cuatro partes:

- **Unidad de control:** es el componente básico de un ordenador ya que controla la ejecución de las operaciones y dirige el funcionamiento de todos los demás componentes de tal forma que el trabajo conjunto de todos conduzca a la consecución de las tareas específicas programadas en el sistema.
- **Unidad Aritmético-Lógica:** es la parte encargada de realizar las operaciones aritméticas (suma, resta, ….) y las operaciones lógicas (comparación, ….). También realiza otras funciones más complejas (raíces, funciones trigonométricas, ..). Al conjunto Unidad de Control y Unidad Aritmético-Lógica se le conoce como **CPU** (**Central Process Unit – Unidad Central de proceso)**. Así tenemos los procesadores Intel, Motorola, AMD.
- **Memoria principal**: es la memoria de almacenamiento interno. Opera a gran velocidad. Aquí se ubican los programas: las instrucciones junto con los datos sobre los que actúan. La memoria principal está formada por una serie de *celdas* o posiciones identificadas por una *dirección* de memoria.
- Por otro lado está la *memoria secundaria* o *memoria externa* que permite resolver los problemas de volatilidad y capacidad de la memoria principal (discos duros, CD, …).
- **Dispositivos de Entrada / Salida** – son los que facilitan la interacción del usuario (el mundo exterior) con la máquina (teclado, monitor, impresora, etc.).

Lenguaje máquina: es el lenguaje propio del ordenador basado en el código binario.

Los ordenadores digitales utilizan internamente el código binario. La mínima información manipulable es un dígito binario, 0 y 1, el bit. Tanto los datos como las instrucciones que ejecuta la CPU han de expresarse en este código para poder ser almacenados en memoria. Al conjunto de instrucciones codificadas en binario se le conoce como lenguaje máquina y es el lenguaje más básico y el único que entiende el ordenador.



1.2 ALGORITMOS Y PROGRAMAS

Un algoritmo es una secuencia ordenada y finita de pasos a seguir para resolver un determinado problema. En todo algoritmo se distingue:

- el procesador – es el que entiende los pasos del algoritmo y los lleva a cabo (por ej. un cocinero / un ordenador)
- el entorno – los materiales necesarios para la ejecución del algoritmo (por ej. huevos, patatas, cebolla / datos en un programa)
- las acciones – los actos del procesador sobre el entorno (cascar, batir, freír / sumar, restar, comparar, asignar)

Para que un ordenador pueda ejecutar un algoritmo ha de expresarse en forma de programa a través de un lenguaje de programación.

Los diagramas de flujo sirven para representar algoritmos de forma gráfica.

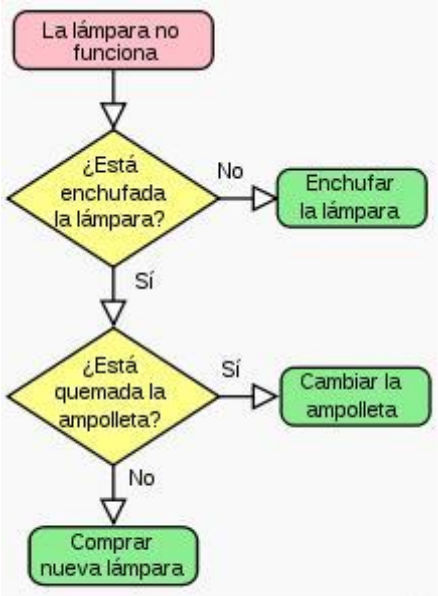


Diagrama de flujo sencillo con los pasos a seguir si una lámpara no funciona.

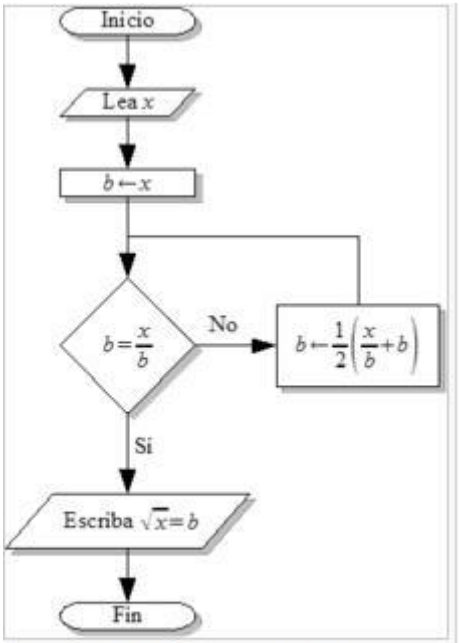


Diagrama de flujo que expresa un algoritmo para calcular la raíz cuadrada de un número x

Al igual que el lenguaje natural consta de sintaxis, semántica y vocabulario, un lenguaje de programación se compone de un conjunto de símbolos (léxico), un conjunto de reglas de sintaxis (que indican cómo construir correctamente las instrucciones) y una semántica (reglas que determinan el significado de las construcciones del lenguaje), que el ordenador es capaz de entender y procesar.

Cuando un algoritmo se expresa en un lenguaje de programación tenemos un programa comprensible por un ordenador. Al hecho de expresar el algoritmo en un lenguaje de programación dado se denomina codificar un programa.

Un lenguaje de programación es un lenguaje artificial utilizado para controlar el comportamiento de una máquina.

1.3 EVOLUCIÓN DE LOS LENGUAJES DE PROGRAMACIÓN

Un programa es un conjunto de instrucciones que actúan sobre unos datos y que están expresados en un lenguaje de programación.

Los lenguajes de programación se pueden clasificar utilizando el criterio de proximidad del lenguaje con la máquina o con nuestro lenguaje natural:

- **Lenguaje máquina** - es el lenguaje básico y el único que entiende el ordenador. Está formado por un conjunto de instrucciones codificadas en binario (dos únicos símbolos, o y 1, llamados bits – binary digit -) dependientes totalmente del hardware del ordenador. Inicialmente se programaba así pero resultaba tedioso y además era muy difícil verificar y poner a punto el programa.
- **Lenguaje de bajo nivel o lenguaje ensamblador** – es una versión simbólica del lenguaje máquina. Cada instrucción lleva asociado un símbolo (una palabra nemotécnica – ADD,

SUB, …) para que resulte más fácil la programación. Requiere una fase de traducción al lenguaje máquina (con un traductor – assembler). El lenguaje ensamblador, como el lenguaje máquina, es dependiente del hardware, sólo funciona en un tipo de ordenador y no en otro.

- **Lenguaje de alto nivel** – es independiente de la máquina y, por tanto, portable. Es más sencillo de aprender ya que es más cercano a nuestro lenguaje natural. Las modificaciones y puestas a punto son más fáciles. Los programas escritos en un lenguaje de alto nivel necesitan una etapa de traducción al lenguaje máquina que realiza un programa compilador. El programa resultante de la traducción ocupa más memoria y su velocidad de ejecución aumenta en relación a los programas escritos directamente en código máquina (Pascal, C, C++, Java, …).

Otra posible clasificación de los lenguajes de programación puede hacerse en relación al paradigma de programación que utilizan:

- **Lenguajes imperativos o procedimentales** – están orientados a instrucciones (Pascal, C)
- **Lenguajes orientados a objetos** – se centran más en los datos y su estructura (SmallTalk, Java, C#, Eiffel).
- **Lenguajes declarativos (funcionales y lógicos)** – se construyen mediante descripciones de funciones matemáticas (Haskell, Lisp) o expresiones lógicas (Prolog)

1.4 TRADUCTORES: COMPILADORES E INTÉRPRETES

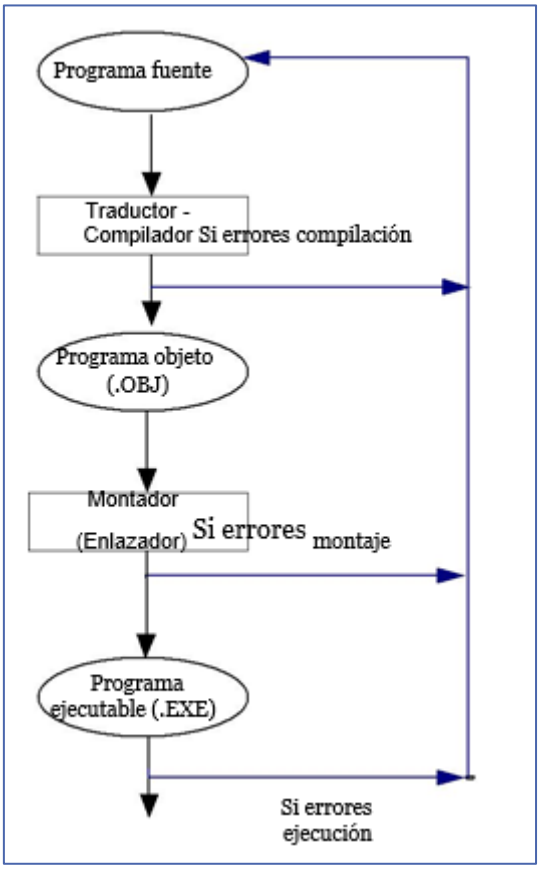
Un ***traductor*** es un programa que recibe un *programa fuente* escrito en un lenguaje de alto nivel y obtiene como salida un programa traducido a código máquina.

Los traductores se dividen en:

- **Intérpretes** – toman un programa fuente y lo traducen de forma simultánea a su ejecución.
- **Compiladores** – traducen un programa fuente a lenguaje máquina. Al proceso de traducción se le denomina en este caso *compilación* y al programa resultado de la compilación se le denomina *programa objeto (*con extensión .**OBJ**). Los compiladores detectan además los errores de sintaxis en el programa fuente.

En la mayoría de los lenguajes compilados, el programa objeto no es directamente ejecutable. Es preciso efectuar el *montaje* o *linkedición* con ayuda del programa *montador* o *enlazador (linker)* para obtener un *programa ejecutable* (**.EXE**).

En este proceso de montaje se enlazan algunas rutinas del sistema o subprogramas compilados separadamente.



1.5 PARADIGMAS DE PROGRAMACIÓN

Un ***paradigma de programación*** es un conjunto de teorías, estándares, que indican la forma de organizar los programas sobre la base de algún modelo conceptual y un lenguaje apropiado que lo soporte.

Un paradigma representa un enfoque particular de la construcción del software

Además de los paradigmas funcional y lógico, los dos paradigmas fundamentales de programación son:

- ***paradigma procedural***
- ***paradigma orientado a objetos***

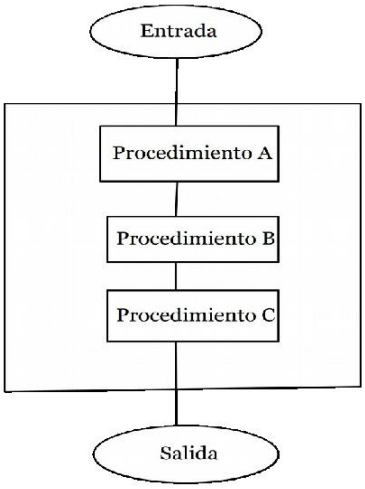
Ambos utilizan la descomposición ("*divide y vencerás*") para afrontar la complejidad de los problemas.

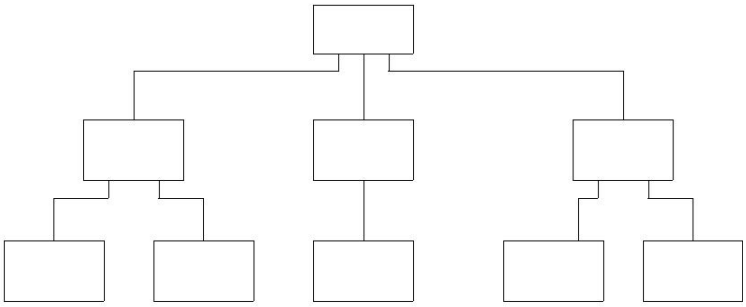
1.5.1 PARADIGMA PROCEDURAL (PROGRAMACIÓN ESTRUCTURADA)

Ampliamente utilizado en la década de los 70. Los lenguajes Pascal y C son los máximos exponentes de este tipo de programación. Se basa en la descomposición funcional (o algorítmica).

El bloque principal de desarrollo es el ***procedimiento*** o ***función***. Los algoritmos que implementan estos procedimientos utilizan para su construcción las tres estructuras básicas de la programación estructurada: *secuencia, selección e iteración*.

Aquí, una aplicación está formada por una jerarquía de módulos que se organizan en torno a un programa principal. Los datos tienen un papel secundario. La programación estructurada se resume en la expresión: "*Algoritmos + Estructuras de datos = Programas*".





Este paradigma tiene sus ventajas cuando se trata de resolver tareas sencillas.

1.5.2 PARADIGMA ORIENTADO A OBJETOS

Tuvo su gran impacto en la década de los 90. Hoy día es ampliamente utilizado en el desarrollo de software, tanto en análisis como en diseño y programación. Vino a resolver los problemas de complejidad y tamaño en el desarrollo de software.

Este paradigma utiliza para resolver la complejidad de un problema la descomposición orientada a objetos. Un programa es un conjunto de objetos que cooperan entre sí para resolver una determinada tarea. El bloque principal de construcción del programa es el ***objeto***, entidad extraída del espacio (o dominio) del problema, con una identidad, unos atributos y un comportamiento. Los datos son el punto central de atención de los programas orientados a objetos. En la POO: *"Objetos + Flujo de mensajes = Programas".*

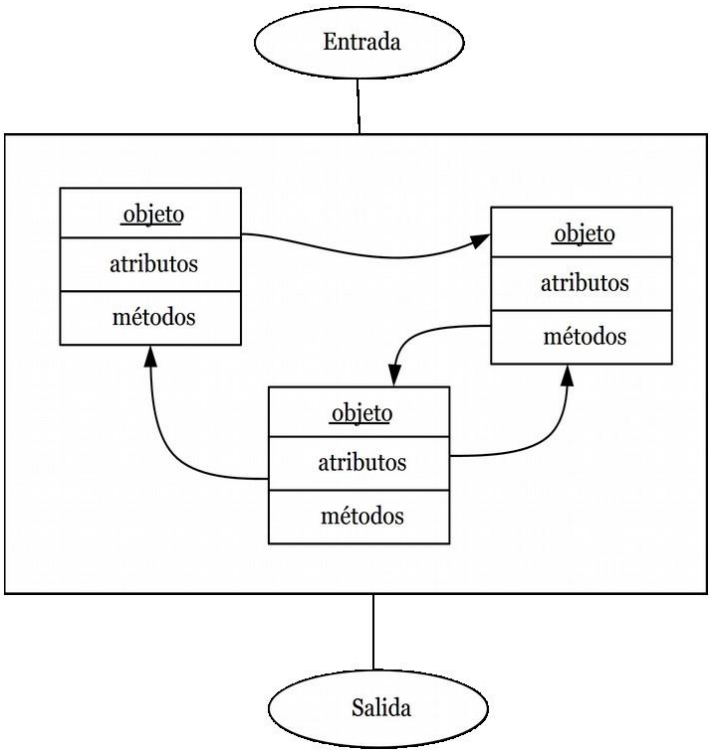
Este paradigma ofrece ventajas sobre el paradigma procedural:

- modela mejor el mundo real, es más intuitivo, describe el problema en términos similares a los que utiliza la mente humana
- maneja mejor la complejidad del software
- permite la reutilización
- obtiene programas más robustos y estables
- facilita la extensibilidad, la escalabilidad de las aplicaciones

La POO se basa en los siguientes principios:

- ***Abstracción*** – mecanismo que maneja la complejidad. Permite captar lo esencial ignorando los detalles.
- ***Encapsulación*** – permite mantener oculta la implementación de la abstracción.
- ***Jerarquía*** **-** la jerarquización ordena las abstracciones, las organiza, simplificando así su desarrollo. Permite la implementación de la herencia y el polimorfismo.

- ***Modularización*** – división del programa en módulos para facilitar su diseño, mantenimiento y reutilización.



1.6 CALIDAD DE LOS PROGRAMAS

Todo buen programa debe ser:

- ***Correcto*** – sin errores y cumpliendo los requisitos que satisfacen el problema
- ***Robusto*** - capaz de funcionar incluso en situaciones anormales
- ***Legible*** - claro y fácil de leer
- ***Modificable*** - fácil de modificar y mantener
- ***Eficiente*** - utiliza adecuadamente los recursos de la máquina (memoria y tiempo de ejecución)
- ***Reutilizable*** – fácil de reutilizar, todo o parte, en otros programas
- ***Modular*** – dividido en partes cada una de ellas resolviendo una determinada tarea

1.7 LENGUAJES ORIENTADOS A OBJETOS

Los lenguajes de programación orientada a objetos son aquellos que soportan las características de la POO. Dentro de ellos se distinguen:

- ***Lenguajes puros orientados a objetos*** – proceden del lenguaje Simula (Simula 67 fue el primer lenguaje OO diseñado en el año 67). Trabajan exclusivamente con objetos y clases. Entre ellos están SmallTalk, Eiffel, Java, C#.

- ***Lenguajes híbridos*** – están basados en lenguajes procedimentales. Soportan por tanto, la programación procedural (estructurada) y la POO. Estos lenguajes se construyen a partir de otros ya existentes, como C o Pascal. Entre ellos destaca: Ada, Modula y Object Pascal proceden del lenguaje Pascal, C++ procede del lenguaje C.

1.8 DESARROLLO DE SOFTWARE ORIENTADO A OBJETOS

El proceso de desarrollo de software es un conjunto de actividades que se necesitan llevar a cabo para transformar los requerimientos de un problema en el producto software deseado. Describe un enfoque para la construcción, desarrollo y mantenimiento del software. **RUP** (*Rational Process Unified* – Proceso Unificado de Rational) es una de las metodologías de desarrollo de software tradicional en la construcción de software que utiliza el desarrollo iterativo e incremental y el lenguaje UML.

El desarrollo iterativo e incremental que utilizan las metodologías de desarrollo de software orientado a objetos consiste en:

a) dividir un proyecto en mini proyectos más fáciles de manejar

b) cada mini proyecto se denomina iteración

c) en cada iteración se cubre el ciclo entero de desarrollo de una aplicación informática (análisis OO, diseño OO, programación OO, pruebas e integración y mantenimiento)

d) cada iteración genera una versión parcialmente completa del sistema

e) las sucesivas iteraciones se construyen unas sobre otras hasta que el sistema se ha completado

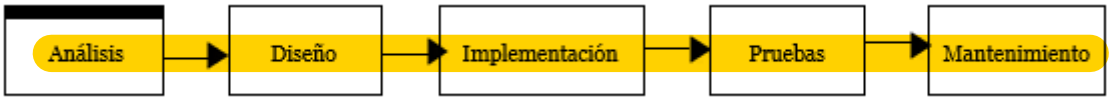
f) la diferencia entre una y otra iteración se denomina incremento

Las *metodologías ágiles* son muy populares actualmente (**Extreme Programming, Scrum**). Basadas también en el desarrollo iterativo e incremental se adhieren al llamado "*manifiesto ágil*" adaptándose mucho mejor a los continuos cambios en los requerimientos software enfocándose en la gente y los resultados.

Veamos brevemente las diferentes etapas en el desarrollo OO de una aplicación informática (*ciclo de desarrollo del software):*

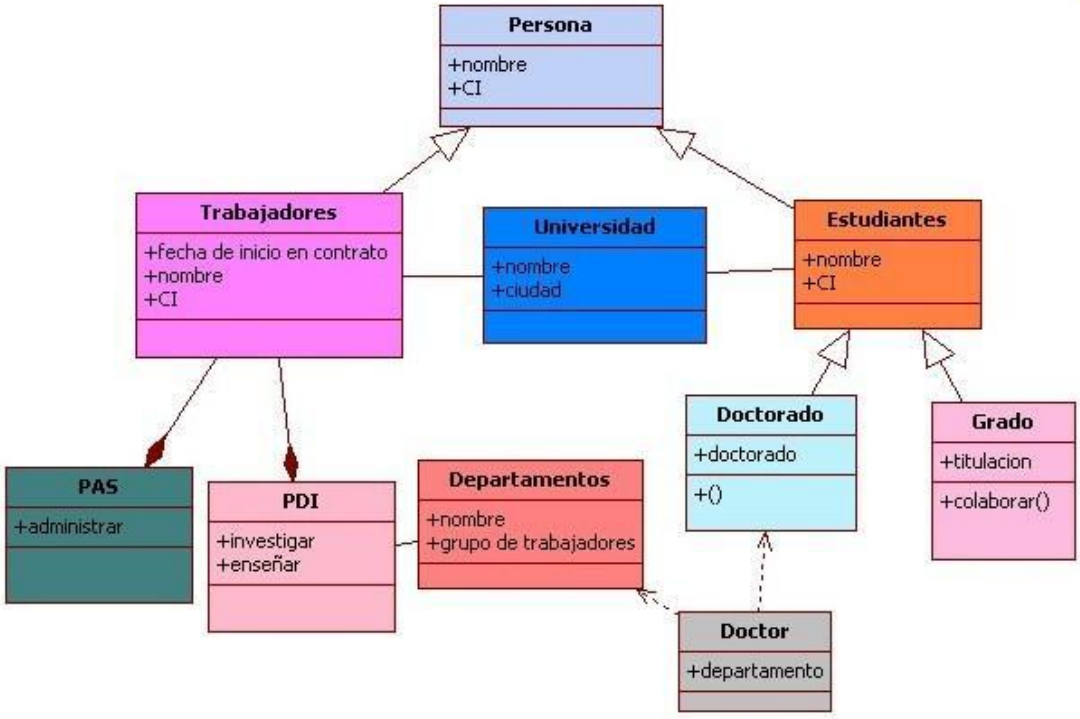
- **Análisis OO** – se investiga el problema y los requisitos en vez de solucionarlo. Se habla de análisis de requisitos o análisis de objetos (estudio de los objetos del dominio). Se presta especial atención a encontrar y describir los objetos (las clases) en el dominio del problema y se construye un modelo (Libro, Biblioteca, Socio, … en el sistema de información de una biblioteca).
- **Diseño OO** – hay que encontrar una solución que satisfaga los requisitos, sin realizar todavía la implementación. Se diseñan los objetos (las clases), prestando especial atención a los objetos software y estudiando cómo van a colaborar para satisfacer los requisitos. En el sistema de información de la biblioteca un objeto de la clase Libro podría tener un atributo título y un método numeroVecesPrestado().
- **Implementación** – (Programación OO - codificación) – Se traduce el diseño a un lenguaje de programación concreto.

- **Pruebas** - se verifica que el producto construido hace lo deseado. Se preparan tests con datos de prueba para comprobar el correcto funcionamiento.
- **Mantenimiento** - actividades que incluyen modificaciones del producto, tanto del código como de la documentación, debido a errores o a la necesidad de mejora o/y adaptación



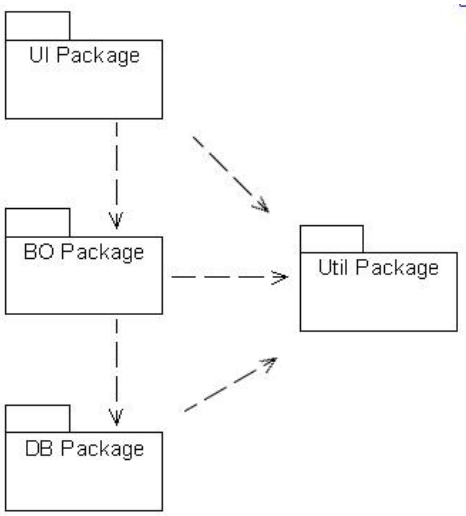
1.8.1 EL LENGUAJE UML

UML (Unified Modeling Language – Lenguaje Unificado de Modelado) es un lenguaje utilizado para describir modelos. Un modelo es una descripción abstracta de un sistema, una representación simplificada que permite comprenderlo y simularlo.



UML no es una metodología de desarrollo de software OO sino una notación para especificar, construir, visualizar y documentar los elementos de un sistema de software.

UML define una serie de diagramas o representaciones gráficas de elementos interconectados. Entre otros diagramas, UML permite representar diagramas de clases, que muestran la estructura estática de un modelo, es decir, las clases que lo componen y sus relaciones.



1.9 EL LENGUAJE DE PROGRAMACIÓN JAVA

Java es un lenguaje de programación de alto nivel totalmente orientado a objetos desarrollado por el equipo de James Gosling de Sun Microsystems en 1995. Este lenguaje fue diseñado para dispositivos electrónicos como calculadoras, microondas y TV interactiva antes de que diese comienzo la era world wide web.

Inicialmente se llamó Oak (*roble* en inglés) pero tuvo que cambiar de denominación ya que dicho nombre estaba ya registrado por otra empresa.

El proyecto Green fue el primero en que se aplicó Java y consistía en un sistema de control completo de aparatos electrónicos y el entorno de un hogar. En ese proyecto aparecía ya *Duke*, la actual mascota de Java.

Lo cierto es que los primeros proyectos no tuvieron éxito y Java entró en un letargo. No fue hasta la expansión de Internet que se produjo el resurgimiento de Java.

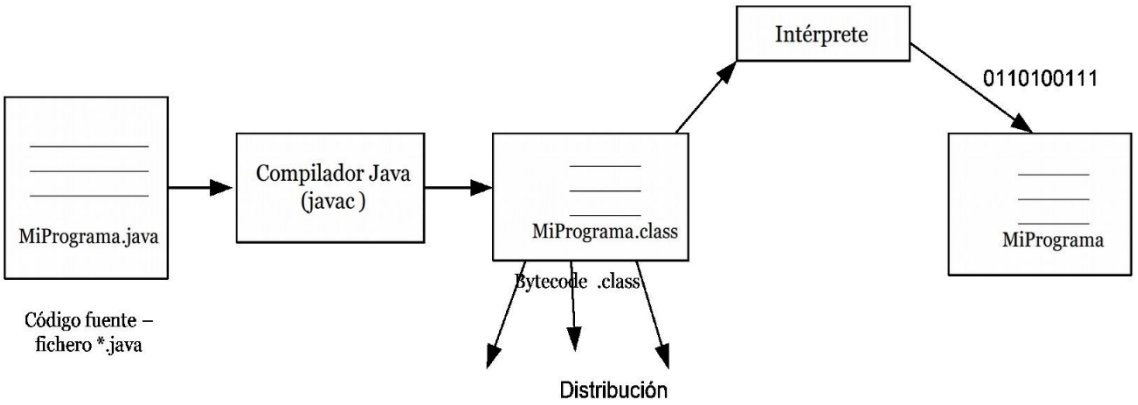
Java ganó rápidamente muchos adeptos principalmente por su neutralidad respecto a la plataforma de ejecución y porque los programas Java podían ejecutarse dentro de un navegador (los *applets*).

Sin embargo hay muchas otras características que hacen de Java un lenguaje atractivo cuya evolución ha sido muy rápida.

1.9.1 CARACTERÍSTICAS DE JAVA

- ***Sencillo*** - Java es un lenguaje sencillo (en comparación con su predecesor C++) y elegante. Java elimina los punteros de C++, no permite la herencia múltiple. El *garbage collector* (recolector de basura) permite la gestión automática de la memoria dinámica. Proporciona una sintaxis sencilla, elegante, hay pocas construcciones de programa.
- ***Orientado a objetos*** – Java fue diseñado desde el principio para ser un lenguaje OO, lo que facilita la construcción de software siguiendo el paradigma de la POO.
- ***Distribuido*** – la adaptabilidad de Java para trabajar en entornos de red es inherente a su arquitectura ya que fue diseñado para ello. Las clases que conforman una aplicación pueden estar ubicadas en distintas máquinas de la red.

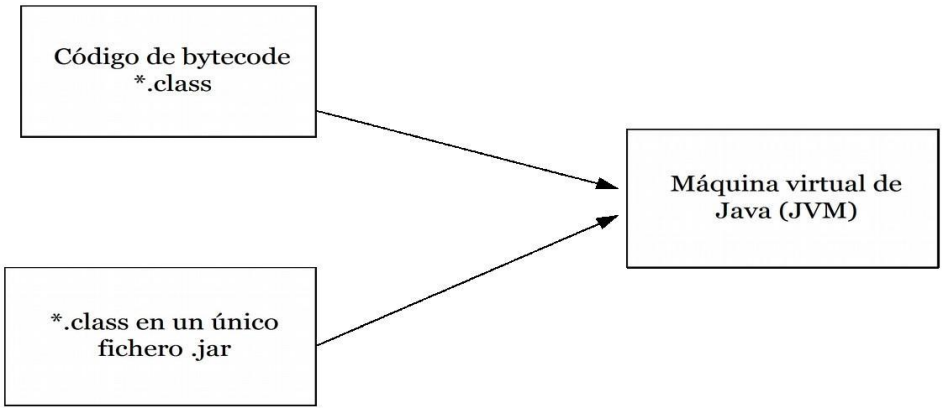
- ***Robusto*** – confiable, no permite construcciones peligrosas (uso de punteros), pone especial énfasis en la verificación temprana de errores (en tiempo de compilación), gestiona excepciones en tiempo de ejecución.
- ***Seguro*** – proporciona mecanismos de seguridad que protegen el sistema.
- ***Interpretado*** – Java no funciona como la mayoría de los lenguajes de programación compilados donde el compilador traduce el código fuente al lenguaje máquina concreto del procesador. Java es compilado e interpretado a la vez. Se necesita un intérprete para ejecutar los programas Java.



Los programas fuente escritos en java *(*.java*) se compilan (con el compilador *javac*) para obtener ficheros con extensión *.class*. Estos ficheros ya pueden ser distribuidos, no hay proceso de enlace (no se necesita un montador).

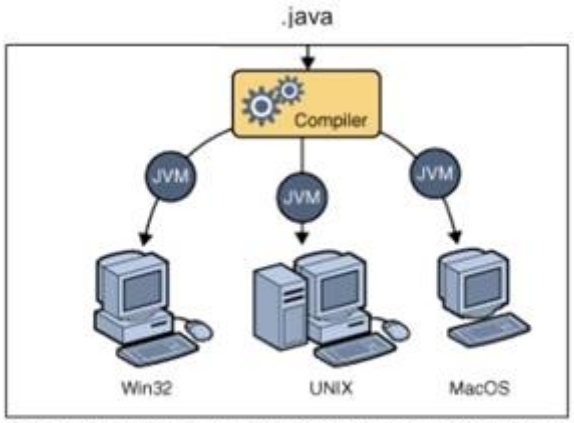
Los ficheros **.class* no contienen código máquina comprensible por ningún procesador sino que contiene *bytecodes*, una especie de código de bajo nivel (un lenguaje intermedio) que será interpretado por la máquina virtual de Java. Esta máquina abstracta es la que ha de estar instalada en una máquina real para poder ejecutar la aplicación java. El código de *bytecodes* es el mismo para todos los sistemas.

El hecho de que Java sea interpretado hace que su rendimiento sea menor en relación a otros lenguajes aunque va mejorando en cada nueva versión de la JVM (no hay ejecutable).



- *De arquitectura neutral y portable* – ya que Java es interpretado permite que sea independiente de la plataforma en la que se ejecuta.

- El lema de Java es "*escribe una vez, ejecuta en cualquier lugar*" (*write once, run anywhere*). El programa se escribe y compila una vez y se ejecuta en cualquier plataforma que tenga instalada la JVM.

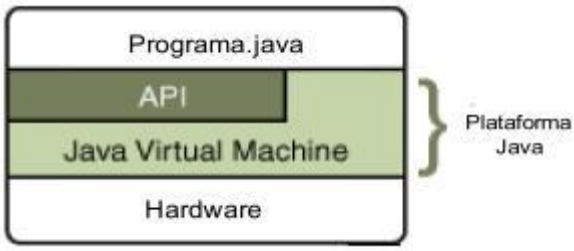


Gracias a la JVM la misma aplicación puede ejecutarse en múltiples plataformas

- *Multihilo* – permite la ejecución de varias tareas simultáneamente. Esta característica es especialmente útil en la programación gráfica (GUI – animaciones) y en la programación en red (un servidor puede atender a múltiples clientes al mismo tiempo).

1.10 LA PLATAFORMA JAVA

Java no es sólo un lenguaje de programación sino toda una plataforma de desarrollo que, además del lenguaje que da nombre a la plataforma, consta de:

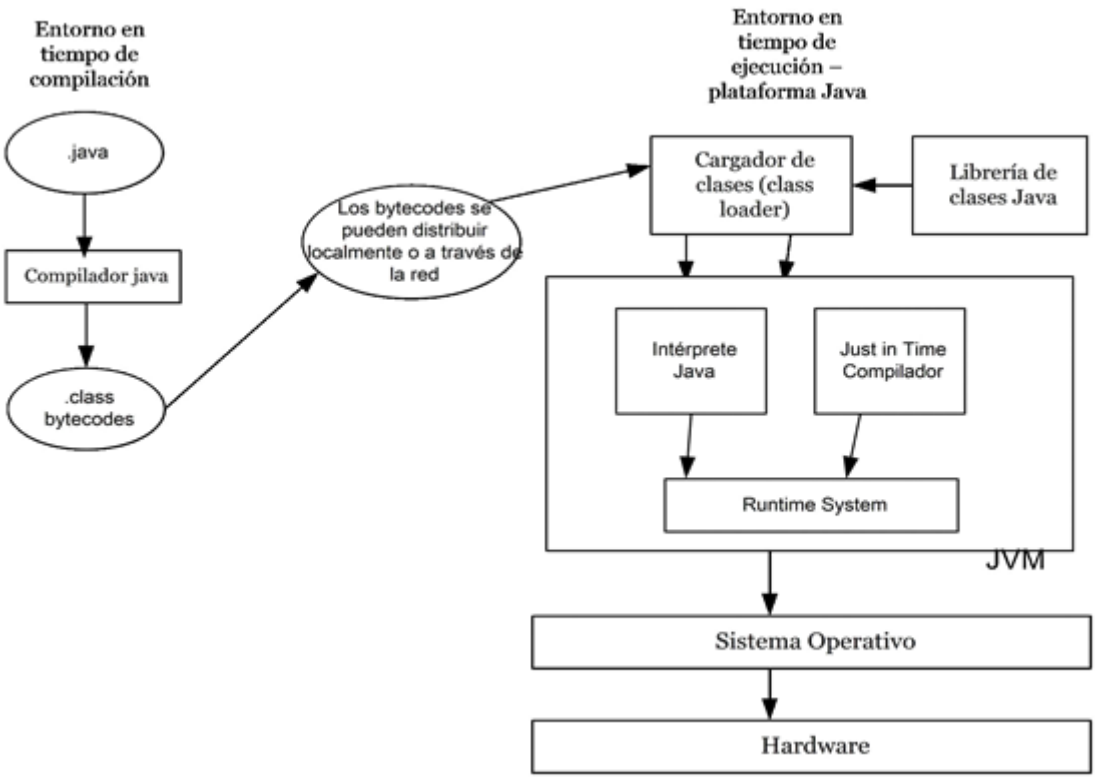


1.10.1 LA JVM – JAVA VIRTUAL MACHINE – LA MÁQUINA VIRTUAL DE JAVA

Es la máquina abstracta (en realidad es software) que lee los ficheros *.class* que contienen *bytecodes* y los traduce a código máquina.

La JVM:

- se requiere para ejecutar cualquier programa Java
- es dependiente de la máquina (JVM para Windows, JVM para Linux, etc.)
- existe también dentro de los navegadores (para poder ejecutar los *applets*)



Justo antes de ejecutar un programa, Java utiliza un cargador de clases (*class loader*) para ubicar los bytecodes de todas las clases a utilizar por el programa en la memoria del ordenador.

1.10.2 LA API DE JAVA – JAVA APPLICATION PROGRAMMING INTERFACE

Es una colección de componentes software, clases, que el programador puede incluir en sus programas. La API ofrece capacidades de todo tipo: gráficas, matemáticas, componentes de red, etc. Todos estos componentes se agrupan en librerías de clases relacionadas (paquetes).

La API junto con la máquina virtual (JVM) conforman el JRE (Java Runtime Environment). Para ejecutar programas Java simplemente (no para desarrollarlos) es suficiente tener instalado el JRE en el sistema.

1.11 HERRAMIENTAS DE DESARROLLO (JAVA SOFTWARE DEVELOPMENT KIT)

Además del JRE para ejecutar programas si queremos desarrollarlos Java proporciona un conjunto de herramientas de desarrollo tales como:

- **javac** – compilador a bytecodes
- **java** –llamada a la máquina virtual (lanzador de aplicaciones Java)
- **jar** – herramienta para crear archivos *.jar*
- ***javadoc*** – *generador de documentación*
- **jdb** – depurador de consola

Estas herramientas junto con el JRE constituyen el *Java SE Development Kit (JDK).*

El JDK puede descargarse desde la página de Oracle: http://www.oracle.com/technetwork/java/javase/downloads/index.html

1.12 VERSIONES DE JAVA

Java ha evolucionado muy rápidamente desde su aparición en 1995.

La versión inicial fue Java 1.0. Java 1.1 introdujo cambios significativos. A partir de aquí la numeración de las nuevas versiones es algo confusa. La siguiente versión fue Java 2 v.1.2.

Java 2 v. 1.5 , conocida también como Java 5.0 , supuso un cambio cualitativo en cuanto a introducción de mejoras en el lenguaje (colecciones genéricas, bucle *for-each*, enumerados, autoboxing/unboxing, ...).

Java SE 6 fue lanzada en Diciembre 2006, es la primera versión de Java para trabajar con Windows Vista. Mejoró el rendimiento de ejecución de las aplicaciones, ciertos elementos de la edición empresarial (J2EE) se trasladaron a la versión de escritorio (J2SE), elJ2SE incorporó un servidor web de desarrollo, el paquete Swing fue optimizado, se incluyó soporte para lenguajes de script (por ej, JavaScript), etc.

En Julio de 2011 se liberó la versión Java SE 7. Es la primera versión lanzada desde que Oracle adquiriese Sun. Rendimiento, estabilidad y seguridad se mejoraron, se incluyó la API de New File System, que ofrece acceso nativo a varias operaciones sobre archivos y carpetas, mejoras en el soporte para internacionalización, soporte dinámico de lenguajes tipados, … Se incorporó también con el JDK la tecnología JavaFX, un conjunto de librerías para crear y desplegar aplicaciones con un aspecto vanguardista y contenidos avanzados, audio y vídeo.

En Marzo de 2014 fue liberada la versión 8 de Java. Era muy esperada ya que incluye notables cambios en el lenguaje y mejoras en la plataforma, entre ellos la introducción de las expresiones *lambda* (o *closures*) que da a Java la característica de programación funcional de otros lenguajes como Phyton, nueva API para fechas, mejoras de seguridad, etc.

En la actualidad, la última versión estable es la 16.

Han sido desarrolladas diferentes ediciones de Java para diferentes tipos de aplicaciones:

- **Java Standard Edition (Java SE) -** para escribir, desarrollar y ejecutar aplicaciones de escritorio, aplicaciones web y applets.
- **Java EE (Entreprise Edition) -** Java Edición Empresarial, versión ampliada de la Java SE que incluye las API necesarias para construir aplicaciones para arquitectura distribuidas multicapa, aplicaciones distribuidas, servicios web.
- **Java 2 Platform MicroEdition (Java ME) –** edición especial para dispositivos móviles.

1.13 TIPOS DE APLICACIONES JAVA

- **Aplicaciones normales –** aplicaciones autónomas (*stand-alone*) que se ejecutan en un ordenador que tiene la JVM. Pueden producir salida de texto en consola o utilizar un interface gráfico (GUI). Requieren de un método main() (punto entrada a la aplicación). Desarrolladas usando Java SE.

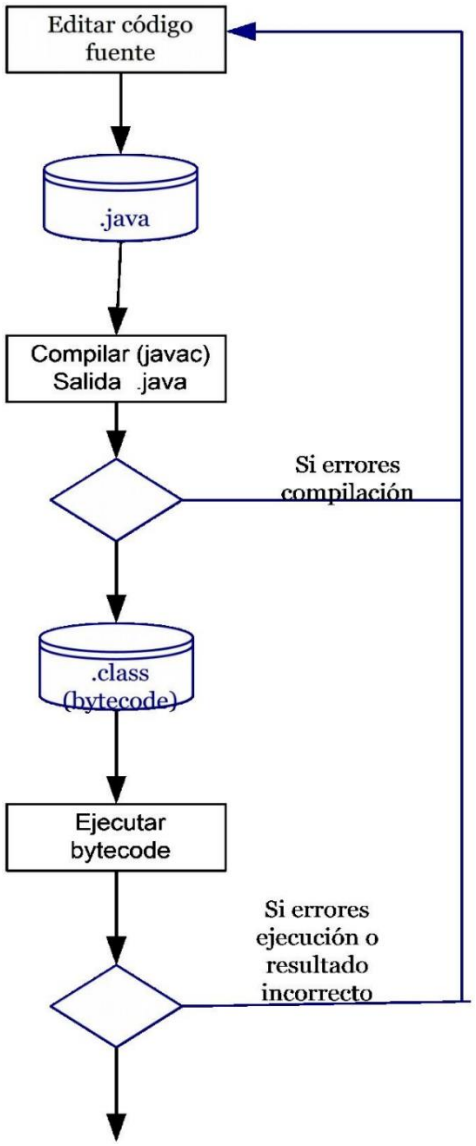
- **Aplicaciones web –** aplicaciones que se ejecutan en un servidor. Las aplicaciones web utilizan tecnologías llamadas *servlets* y JSP – *Java Server Pages*. Las aplicaciones web interactúan normalmente con bases de datos. Desarrolladas usando Java SE o Java EE.
- **Applets –** un applet es un programa diseñado para ejecutarse dentro de un navegador compatible con Java. La salida es gráfica. Cada vez están más en desuso.

1.14 EDITAR, COMPILAR Y EJECUTAR UN PROGRAMA JAVA

El único software que se necesita para desarrollar y ejecutar un programa Java es el JDK y un editor de textos.

El fichero fuente se escribe con el editor y se salva como un fichero con extensión *.java*.

El compilador java, *javac*, se invoca desde la línea de comandos para producir un fichero con extensión *.class*. El fichero *.class* se invoca también desde línea de comandos llamando a *java*, la máquina virtual.



1.15 ENTORNOS DE DESARROLLO

Existen numerosos entornos de desarrollo (IDE – Integrated Development Environment) que facilitan la tarea de editar, compilar y ejecutar un programa Java. Entre otros: Eclipse, NetBeans, IntelliJ IDEA, Dr. Java, JCreator.

Nosotros utilizaremos a lo largo del curso fundamentalmente BlueJ aunque manejaremos también alguno de los dos entornos profesionales más utilizados al trabajar en Java, Eclipse y/o NetBeans:

- **BlueJ** - es un excelente entorno de desarrollo diseñado por las universidades de Kent (Reino Unido), Deakin (Australia) y Southern (Dinamarca) específicamente para la enseñanza y aprendizaje de la POO en Java. Incluye herramientas educacionales tales como visualización e interacción de objetos que permiten aprender desde el inicio los conceptos de la POO de una manera sencilla. Es gratuito. Se puede descargar desde http://www.bluej.org
- **Eclipse** - es un entorno de desarrollo integrado de código abierto multiplataforma que emplea módulos (plugin) para proporcionar toda su funcionalidad. A través de los plugins se pueden montar herramientas de desarrollo para cualquier lenguaje. https://www.eclipse.org/downloads/
- **NetBeans** – es un entorno de desarrollo integrado profesional gratuito de código abierto. Incluye una gran variedad de herramientas que ayudan en el desarrollo de cualquier tipo de aplicación Java: aplicaciones web, applets, aplicaciones con GUI, JSP, etc. https://netbeans.org/
- **IntelliJ IDEA** – completísimo entorno de desarrollo para desarrollar aplicaciones Java creado por JetBrains. Existe una versión open source y otra comercial. https://www.jetbrains.com/idea/

1.16 COMENTARIOS

Todas las sentencias en Java terminan en ";".

Se pueden incluir comentarios en el código fuente de la clase para aclarar su lectura, para hacer el código más legible:

- un comentario de una sola línea se escribe precedido de //

```
// El precio de un ticket en esta máquina
private int precio;
```

- si un comentario ocupa varias líneas se precede de /* y termina en */
- Java soporta además los comentarios javadoc que empiezan con /** y terminan en */ (son los que aparecerán en formato HTML cuando se genera la documentación de un proyecto)

Los comentarios son ignorados por el compilador. No son sentencias ejecutables.

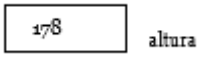
1.17 LOS TIPOS DE DATOS EN JAVA

Ya hemos visto como los atributos de una clase, los parámetros y el valor de retorno de un método (si lo hay) tienen un determinado tipo de datos. El tipo de datos determina los valores que pueden tomar. En general, veremos que cualquier variable (no solo los atributos y parámetros) posee un determinado tipo de datos.

Java clasifica los tipos de datos en:

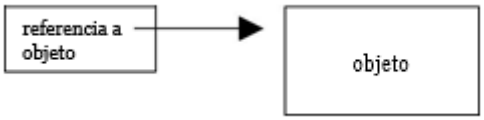
- tipos **primitivos** – son los tipos *simples*. El atributo o parámetro (la variable, en general) guarda directamente el valor de ese tipo.

```
private int altura;
```




```
private boolean esVisible;
```

- tipos **referencia** – almacenan una referencia (*puntero*) a un objeto.




```
private Estudiante estudiante1;
private Circulo sol;
private String nombreEstudiante;
```

De momento veremos únicamente los tipos primitivos.

1.17.1 LOS TIPOS PRIMITIVOS EN JAVA

Las siguientes tablas muestran todos los tipos primitivos del lenguaje Java. Java tiene seis tipos numéricos: cuatro para enteros y dos para números reales. Proporciona además un tipo para representar un carácter simple y otro tipo para representar valores lógicos.

Tipo de variable	Bytes que ocupa	Rango de valores
boolean	2	true, false
byte	1	-128 a 127
short	2	-32.768 a 32.767
int	4	-2.147.483.648 a 2.147.483.649
long	8	$-9 \cdot 10^{18}$ a $9 \cdot 10^{18}$
double	8	$-1{,}79 \cdot 10^{308}$ a $1{,}79 \cdot 10^{308}$
float	4	$-3{,}4 \cdot 10^{38}$ a $3{,}4 \cdot 10^{38}$
char	2	Caracteres (en Unicode)

- Un número sin punto decimal se interpreta como un int (entero). Si queremos que sea un entero largo (long) hay que añadir una L. Ej. 46 - 46L.

- Un número con punto decimal se interpreta como un double. Si se especifica una F al final será float. Ej. 46.89 46.89F.
- Un número real también puede representarse en notación exponencial. Ej. 24.5e-3
- Java considera el código Unicode (codificación UTF-16) para representar los caracteres. Es un código de 16 bits (cada carácter es representado por 4 dígitos hexadecimales, '\u0041' es la letra 'A') que permite representar más de 65000 caracteres de cualquier idioma del mundo. Unicode incluye el código ASCII utilizado por la mayoría de los ordenadores para el intercambio de información.
- Un carácter se escribe entre comillas simples o como un valor Unicode de 4 dígitos hexadecimales precedido de \u. Ej. 'A' , 'a', '?' '\u00F6'.
- Algunos caracteres se pueden especificar mediante secuencias de escape, comenzando por la barra invertida \, y tienen un significado especial.

Secuencia de Escape	Descripción
\n	Salto de línea
\t	Tabulador
\\	Diagonal Inversa \
\"	Comillas Dobles
\'	Comilla Simple
\r	Retorno de Carro (Solo en modo Administrador)
\b	Borrado a la Izquierda (Solo en modo Administrador)

Los valores booleanos son *true* y *false*.

1.17.2 LOS OPERADORES Y LAS EXPRESIONES EN JAVA

Sobre cada uno de los tipos anteriores se pueden utilizar un conjunto de operadores para formar expresiones.

Una ***expresión*** se construye agrupando operadores y operandos. Las expresiones se *evalúan* y producen un resultado de un determinado tipo.

1.18 LAS EXPRESIONES ARITMÉTICAS

Se construyen con operadores aritméticos. Los operandos son valores de un tipo primitivo numérico, entero o real. Cuando se evalúan producen como resultado un valor numérico.

Operadores aritméticos	+ - * / % (módulo o resto de una división)

El resultado de aplicar los operadores / y % depende de si los operandos son enteros o reales. Si son enteros la división es entera, si son reales (o al menos uno de los operandos lo es) el resultado es real.

Ejemplos	5 + 3 ⇒ 8 5 / 3 ⇒ 1 5.0 / 3 ⇒ 1.66666	5 / 2 ⇒ 2 7 % 3 ⇒ 1

Cuando aparece más de un operador en una expresión se aplican las reglas de precedencia de operadores (la tabla se muestra en la siguiente pregunta). Para cambiar la prioridad de los operadores se utilizan los paréntesis. Si varios operadores tienen la misma prioridad la evaluación se realiza de izquierda a derecha (excepto en el caso de los operadores de asignación que se evalúan de derecha a izquierda).

Ejemplos	51 * 3 - 53 ⇒ 100 154 - 2 * 27 ⇒ 100 (200 - 5) / 2 ⇒ 97 2 * (47 + 3) ⇒ 100

Ejer.1.1. Evalúa las siguientes expresiones aritméticas. Indica el tipo de resultado, entero o real.

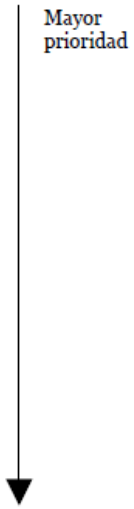
25 + 20 - 15	
20 * 10 + 15 * 10	
20 * 10 / 2 - 20 + 3 * 3	
15 / 10 * 2 + 3 / 4 * 8	
46 % 9 + 4 * 4 - 2	
45 + 43 % 5 * (23 + 3 % 2)	
1.5 * 3	

Ejer.1.2. Transforma las siguientes expresiones matemáticas en expresiones Java:

2πradio (π = 3.1416)	
2πradio²	
$a^2+\frac{b^2}{c}$	
$\frac{4}{3(r+34)}-9(a+bc)+3+\frac{d(2+a)}{a+bd}$	
$-b+\frac{\sqrt{(b^2-4ac)}}{2a}$	

1.19 PRECEDENCIA DE LOS OPERADORES

Operadores	
( )	Paréntesis
++ --	Incremento / Decremento
+ - !	Suma / Resta (Unario)
* / %	Producto / División / Resto
+ -	Suma / Resta
< <= > >=	Comparación
== !=	Igual / Distinto
&&	boolean (y /and)
||	boolean (or / o)
= += -= *= /= %=	Operadores de asignación



1.20 LAS EXPRESIONES BOOLEANAS

Producen, al evaluarlas, un resultado lógico (booleano), true (cierto) o false (falso). Se construyen utilizando operadores relacionales y/o lógicos.

Los operadores relacionales usualmente se combinan con operandos y operadores aritméticos (también con operandos de tipo char).

Operadores relacionales	== < <= > >= !=

Los operadores lógicos (o booleanos) combinan expresiones booleanas para producir un resultado booleano.

Operadores booleanos	&& || !

Se evalúan según las siguientes tablas de verdad:

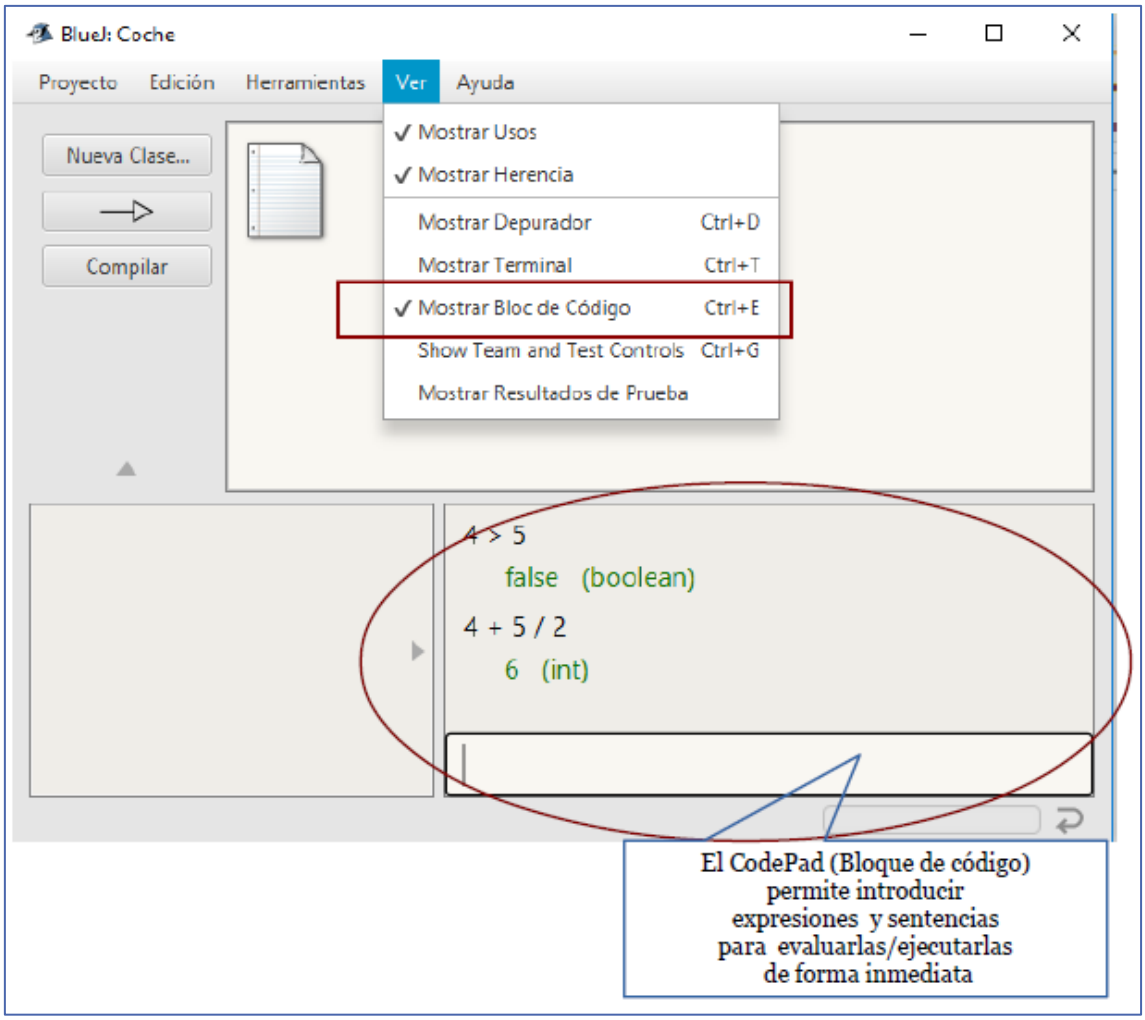
a	b	a && b	a || b	!a
false	*false*	*false*	*false*	*true*
false	*true*	*false*	*true*	*true*
true	*false*	*false*	*true*	*false*
true	*true*	*true*	*true*	*false*

a,b: expresiones booleanas

Java utiliza la evaluación en cortocircuito (short circuiting) en las expresiones que contienen operadores lógicos. Tan pronto como se conoce el resultado final de la expresión no se evalúan el resto de las condiciones. Con el operador && la evaluación se para con el primer false, en el caso del operador || con el primer true.

sexo == 'M' && edad >= 65	si *sexo* no es igual a 'M' la condición es *false* y el resultado final de la expresión será también *false*, no se evalúa la segunda condición
notaTeoria >= 5 || notaPractica > 7	si *notaTeoria* es mayor o igual a 5 la condición es *true* y el resultado final de la expresión será también *true*, no se evalúa la segunda condición

1.20.1 BLUEJ Y EL CODEPAD (BLOC DE CÓDIGO)



Ejer.1.3. Evalúa las siguientes expresiones lógicas, suponiendo x con valor 7:

(true) && (3 > 4)	
(true) && (x > 4)	
x != 3	
(x > 0) || (x < 0)	
25 > 20 && 13 > 5	
10 + 4 < 15 - 3 || 2 * 5 + 1 > 14 - 2 * 2	
4 * 2 <= 8 || 2 * 2 < 5 && 4 > 3 + 1	

Ejer.1.4. Define los siguientes atributos indicando su tipo:

a) estaVacia, que indica si una urna está o no vacía

b) edad, que indica la edad de una persona

c) facturaLuz, que denota el importe de la factura de la luz

d) estadoCivil que indica el estado civil de una persona ('S' 'C' 'V' 'D')

e) nombreAsignatura que indica el nombre de la asignatura que se está cursando

f) areaFigura que indica el área de una determinada figura

g) estaAprobado que indica si un alumno ha aprobado o no cierta asignatura

Ejer.1.5. Sean los siguientes atributos:

```
private int numero;
private int edad;
private int dia;
private int nota;
private char estadoCivil;
```

Construye las siguientes expresiones lógicas de tal manera que al evaluarlas sean ciertas si *numero*:

a) es distinto de 0

b) es igual a 0

c) está comprendido entre 1 y 100

d) está entre 1 y 100 o es negativo

e) es par

f) es múltiplo de 4

g) es múltiplo de 4 pero no de 100

sea cierta:

h) si una persona es adulta

sea cierta si *dia*:

i) suponiendo que corresponde al mes de enero, es correcto

j) suponiendo que corresponde al mes de abril, es incorrecto

sea cierta si una persona:

k) ha aprobado un examen

l) es soltera o casada

1.21 VARIABLES

Las variables Java son un espacio de memoria en el que guardamos un determinado valor (o dato). Para definir una variable seguiremos la estructura:

```
[privacidad] tipo_variable identificador;
```

Java es un lenguaje de tipado estático. Por lo cual todas las variables tendrán un tipo de dato (ya sea un tipo de dato primitivo o una clase) y un nombre de identificador.

El tipo de dato se asignará a la hora de definir la variable. Además, en el caso de que las variables sean propiedades de objetos tendrán una privacidad (por tanto, de momento nos olvidamos de ello).

Ejemplos de variables serían:

```
int numero = 2;
String cadena = "Hola";
long decimal = 2.4;
boolean flag = true;
```

Cuando vayamos a dar un nombre a una variable deberemos de tener en cuenta una serie de normas. Es decir, no podemos poner el nombre que queramos a una variable.

Los identificadores son secuencias de texto unicode, sensibles a mayúsculas cuyo primer carácter solo puede ser una letra, número, símbolo dólar $ o subrayado _ . Si bien es verdad que el símbolo dólar no es utilizado por convención.

Es recomendable que los nombres de los identificadores sean legibles y no acrónimos que no podamos leer. De tal manera que a la hora de verlos se auto-documenten. Además estos identificadores nunca podrán coincidir con las palabras reservadas.

Algunas reglas no escritas, pero que se han asumido por convención son:

Los identificadores siempre se escriben en minúsculas. (pe. nombre). Y si son dos o más palabras, el inicio de cada siguiente palabra se escriba en mayúsculas (pe. nombrePersona). Es decir, siguen el formato camelCase.

1.22 CONSTANTES

El valor de una variable puede cambiar a lo largo de la ejecución de un programa. Una constante, sin embargo, representa a un valor que nunca cambia.

Para declarar una constante en Java:

```
final tipo nombre_constante = valor;
final double PI = 3.1416;
private final double IVA = 0.16;
```

La declaración es similar a la de un atributo salvo que:

- se incluye la palabra reservada final antes del tipo de la constante. Esto indica a Java que el valor que se declara no puede cambiar.
- hay que inicializar la constante en el momento de la declaración
- por convención se escriben en mayúsculas