

# UT1 Introducción a la programación:

1. Concepto de programa informático. Instrucciones y datos.
2. Ejecución de programas en ordenadores:
   1. Datos, algoritmos y programas.
   2. Hardware vs. Software.
   3. Estructura funcional de un ordenador: procesador, memoria.
   4. Tipos de software:
      1. BIOS.
      2. Sistema Operativo.
      3. Aplicaciones.
   5. Código fuente, código objeto y código ejecutable; máquinas virtuales.
3. Lenguajes de programación:
   1. Tipos de lenguajes de programación.
   2. Características de los lenguajes más difundidos.
4. Introducción a la ingeniería del software:
   1. Proceso software y ciclo de vida del software.
   2. Fases del desarrollo de una aplicación:
      1. Análisis.
      2. Diseño.
      3. Codificación.
      4. Pruebas.
      5. Documentación.
      6. Explotación.
      7. Mantenimiento.
   3. Modelos de proceso de desarrollo software:
      1. Cascada.
      2. Evolutivo.
      3. Iterativo.
   4. Metodologías de desarrollo software.
      1. Características.
      2. Técnicas.
      3. Objetivos.
      4. Tipos de metodologías.
   5. Herramientas CASE (Computer Aided Software Engineering).
5. Proceso de obtención de código ejecutable a partir del código fuente, herramientas implicadas:
   1. Editores.
   2. Compiladores.
   3. Enlazadores.
6. Errores en el desarrollo de programas.
7. Importancia de la reutilización de código.

## Concepto de programa informático. Instrucciones y datos.

Según la **RAE**, el **software** es un **conjunto de programas, instrucciones y reglas informáticas** que permiten ejecutar distintas tareas en una computadora.

Se considera que el software es el **equipamiento lógico e intangible** de un ordenador. En otras palabras, el concepto de software abarca a todas las **aplicaciones informáticas**, como los procesadores de textos, las hojas de cálculo y los editores de imágenes.

Un **programa** **informático** es un conjunto de [instrucciones](http://www.alegsa.com.ar/Dic/comando.php) escritas en algún [lenguaje de programación](http://www.alegsa.com.ar/Dic/lenguaje de programacion.php). El programa debe ser [compilado](http://www.alegsa.com.ar/Dic/compilar.php) o [interpretado](http://www.alegsa.com.ar/Dic/interpretar.php) para poder ser [ejecutado](http://www.alegsa.com.ar/Dic/ejecutar.php) y así cumplir su objetivo.

Un **programa** está formado por una serie de **INSTRUCCIONES** y de estructuras de **DATOS**. Un programa al ejecutarse en un ordenador, en general, acepta una serie de datos de **ENTRADA** y produce unos resultados de **SALIDA**, ejecutando para ello las instrucciones y manejando las estructuras de datos que componen el programa.

**Definición de dato:** representación formal de hechos, conceptos o instrucciones adecuada para su comunicación, interpretación y procesamiento por seres humanos o medios automáticos.

Los tipos de datos se caracterizan por:

* **Dominio de posibles valores**: qué valores puede tomar.
* **Cómo se representan**: cuál es la representación interna y cómo se representan en el lenguaje de programación.
* **Operadores asociados**: qué operaciones o cálculos se pueden realizar con esos datos.

Algunos tipos de datos en Java:

* Entero (byte, short, int, long)
* Booleano (boolean)
* Carácter (char)
* Real (float, double)

**ENTRADA  PROGRAMA  SALIDA**

## 1. Ejecución de programas en ordenadores

### **Datos, algoritmos y programas**.

**PROGRAMA = DATOS + INSTRUCCIONES**

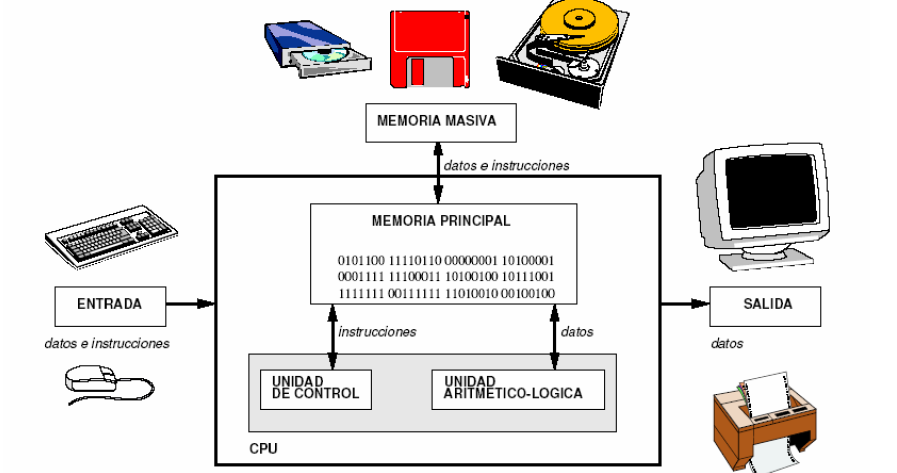
Por otra parte, el término **algoritmo** consiste en una lista ordenada de operaciones que tienen el propósito de buscar la solución a un problema en matemáticas, informática y disciplinas afines (también aplicable en la vida cotidiana).

Un programa es el producto de la traducción de un algoritmo mediante un lenguaje de programación.

**Hardware vs. Software.**

El Hardware son todos los componentes y dispositivos físicos y tangibles que forman una computadora como la CPU o la placa base, mientras que el Software es el equipamiento lógico e intangible como los programas y datos que almacena el ordenador.

### **Estructura funcional de un ordenador: procesador, memoria**.



Los programas se almacenan en una [memoria no volátil](http://es.wikipedia.org/wiki/Memoria_no_volátil" \t "Memoria no volátil) (por ejemplo un disco), para que luego el usuario del ordenador, directa o indirectamente, solicite su ejecución. En el momento de dicha solicitud, el programa es cargado en la [memoria de acceso aleatorio](http://es.wikipedia.org/wiki/Memoria_de_acceso_aleatorio" \t "Memoria de acceso aleatorio) o [RAM](http://es.wikipedia.org/wiki/RAM" \t "RAM) del equipo, bajo el control del [software](http://es.wikipedia.org/wiki/Software" \t "Software) llamado [sistema operativo](http://es.wikipedia.org/wiki/Sistema_operativo" \t "Sistema operativo), el cual puede acceder directamente al [procesador](http://es.wikipedia.org/wiki/Microprocesador" \t "Microprocesador). El procesador ejecuta el programa, instrucción por instrucción hasta que termina. La memoria se estructura en posiciones o palabras de memoria a las que se accede por su dirección.

A un programa en ejecución se le suele llamar también [proceso](http://es.wikipedia.org/wiki/Proceso_(informática)" \t "Proceso (informática)). Un programa puede terminar su ejecución en forma normal o por causa de un error, dicho error puede ser de software o de hardware.

**Ejecución de un programa - IMPORTANTE**

Una vez cargado el programa en la memoria se le cede el control del ordenador:

1. Se lee una instrucción del programa.
2. La unidad de control decodifica la instrucción.
3. La unidad de control envía las señales necesarias para ejecutar la instrucción:

* Se leen los datos de entrada.
* Se efectúa una operación con ellos en la ALU (UAL, Unidad Aritmético Lógica).
* Se almacena el resultado.

1. Se determina cuál es la siguiente instrucción que se debe ejecutar.
2. Se vuelve al paso 1.

### **Tipos de software**.

#### **BIOS**/UEFI

La BIOS (siglas en inglés de basic input/output system; en español "sistema básico de entrada y salida") es un [software](http://es.wikipedia.org/wiki/Software" \t "Software) que localiza y reconoce todos los dispositivos necesarios para cargar el [sistema operativo](http://es.wikipedia.org/wiki/Sistema_operativo" \t "Sistema operativo) en la [memoria RAM](http://es.wikipedia.org/wiki/Memoria_de_acceso_aleatorio" \t "Memoria de acceso aleatorio); es un software muy básico instalado en la [placa base](http://es.wikipedia.org/wiki/Placa_base" \t "Placa base) que permite que ésta cumpla su cometido. Proporciona la comunicación de bajo nivel, el funcionamiento y configuración del [hardware](http://es.wikipedia.org/wiki/Hardware" \t "Hardware) del sistema que, como mínimo, maneja el [teclado](http://es.wikipedia.org/wiki/Teclado_(informática)" \t "Teclado (informática)) y proporciona una salida básica (emitiendo pitidos normalizados por el altavoz de la computadora si se producen fallos) durante el arranque. La BIOS **usualmente está escrita en [lenguaje ensamblador](http://es.wikipedia.org/wiki/Lenguaje_ensamblador" \t "Lenguaje ensamblador)**.

La BIOS es un programa tipo *firmware*. El firmware no es más que un bloque de instrucciones para propósitos muy concretos, estas instrucciones están grabadas en una [memoria](http://www.monografias.com/trabajos13/memor/memor.shtml) de solo [lectura](http://www.monografias.com/trabajos14/textos-escrit/textos-escrit.shtml) o ROM (Read Only Memory). Establecen la [lógica](http://www.monografias.com/trabajos15/logica-metodologia/logica-metodologia.shtml) de más bajo nivel para [poder](http://www.monografias.com/trabajos35/el-poder/el-poder.shtml) **controlar los [circuitos](http://www.monografias.com/trabajos10/infoba/infoba.shtml" \l "circuito) electrónicos de un dispositivo** de cualquier tipo.

UEFI (*Unified Extensible Firmware Interface*) es la versión avanzada y más actual de las BIOS antiguas.

#### Sistema Operativo

Un Sistema Operativo es el software encargado de ejercer el control y coordinar el uso del hardware entre diferentes programas de aplicación y los diferentes usuarios. Es un administrador de los recursos de hardware del sistema.

En una definición informal, es un programa que realiza una distribución ordenada y controlada de los procesadores, memorias y dispositivos de E/S entre los diversos programas que compiten por ellos.

A pesar de que todos nosotros usamos sistemas operativos casi a diario, es difícil definir qué es un sistema operativo. En parte, esto se debe a que los sistemas operativos realizan dos funciones diferentes:

* Proveer una máquina virtual, es decir, un ambiente en el cual el usuario pueda ejecutar programas de manera conveniente, protegiéndolo de los detalles y complejidades del hardware.
* Administrar eficientemente los recursos del computador.

*Ejemplos*: Windows, Linux, Mac OS…

#### Aplicaciones

En [Informática](http://es.wikipedia.org/wiki/Informática" \t "Informática), una aplicación es un tipo de [programa informático](http://es.wikipedia.org/wiki/Programa_informático" \t "Programa informático) diseñado como herramienta para permitir a un usuario realizar uno o diversos tipos de trabajo.

Suele ser una solución informática para la [automatización](http://es.wikipedia.org/wiki/Automatización" \t "Automatización) de ciertas tareas complicadas como pueden ser la [contabilidad](http://es.wikipedia.org/wiki/Contabilidad" \t "Contabilidad), la redacción de documentos, o la gestión de un almacén. Algunos ejemplos de programas de aplicación son los [procesadores de textos](http://es.wikipedia.org/wiki/Procesadores_de_textos" \t "Procesadores de textos), [hojas de cálculo](http://es.wikipedia.org/wiki/Hojas_de_cálculo" \t "Hojas de cálculo), y [bases de datos](http://es.wikipedia.org/wiki/Base_de_datos" \t "Base de datos).

Las aplicaciones desarrolladas “a medida” suelen ofrecer una gran potencia ya que están exclusivamente diseñadas para resolver un problema específico. Otros, llamados “paquetes integrados de software”, ofrecen menos potencia pero a cambio incluyen varias aplicaciones, como un programa procesador de textos, de hoja de cálculo y de base de datos.

Otros ejemplos de programas de aplicación pueden ser: programas de comunicación de datos, [multimedia](http://es.wikipedia.org/wiki/Multimedia" \t "Multimedia), presentaciones, [diseño gráfico](http://es.wikipedia.org/wiki/Diseño_gráfico" \t "Diseño gráfico), cálculo, [finanzas](http://es.wikipedia.org/wiki/Finanzas" \t "Finanzas), [correo electrónico](http://es.wikipedia.org/wiki/Correo_electrónico" \t "Correo electrónico), compresión de archivos, presupuestos de obras, gestión de empresas, etc.

### Código fuente, código objeto y código ejecutable. Máquinas virtuales. - IMPORTANTE

***Código fuente:*** El código fuente de un [programa informático](http://es.wikipedia.org/wiki/Programa_informático" \t "Programa informático) (o [software](http://es.wikipedia.org/wiki/Software" \t "Software)) es un conjunto de líneas de texto que son las instrucciones que debe seguir el ordenador para ejecutar dicho programa.

***Código objeto:*** Se llama código objeto al código que resulta de la [compilación](http://es.wikipedia.org/wiki/Compilación" \t "Compilación) del [código fuente](http://es.wikipedia.org/wiki/Código_fuente" \t "Código fuente). Para ello, se usa un programa llamado compilador.

El código objeto consiste en [lenguaje máquina](http://es.wikipedia.org/wiki/Lenguaje_máquina" \t "Lenguaje máquina) o [bytecode](http://es.wikipedia.org/wiki/Bytecode" \t "Bytecode) y se distribuye en uno o varios archivos que corresponden a cada código fuente compilado.

***Código ejecutable:*** Para obtener un [programa ejecutable](http://es.wikipedia.org/wiki/Ejecutable" \t "Ejecutable) se han de enlazar todos los archivos de código objeto con un programa llamado [enlazador](http://es.wikipedia.org/wiki/Enlazador" \t "Enlazador) (linker).

***Máquinas virtuales***: Una máquina virtual Java (en inglés Java Virtual Machine, JVM) es una [máquina virtual de proceso](http://es.wikipedia.org/w/index.php?title=Máquina_virtual_de_proceso&action=edit&redlink=1" \t "Máquina virtual de proceso (aún no redactado)) nativo, es decir, ejecutable en una plataforma específica, capaz de interpretar y ejecutar instrucciones expresadas en un [código binario](http://es.wikipedia.org/wiki/Código_binario" \t "Código binario) especial (el [Java bytecode](http://es.wikipedia.org/wiki/Java_bytecode" \t "Java bytecode)), el cual es generado por el [compilador](http://es.wikipedia.org/wiki/Compilador" \t "Compilador) del lenguaje [Java](http://es.wikipedia.org/wiki/Lenguaje_de_programación_Java" \t "Lenguaje de programación Java).

El código binario de Java no es un [lenguaje de alto nivel](http://es.wikipedia.org/wiki/Lenguaje_de_alto_nivel" \t "Lenguaje de alto nivel), sino un [intermedio](http://es.wikipedia.org/wiki/Código_máquina" \t "Código máquina) de bajo nivel (para la JVM). Como todas las piezas del rompecabezas Java, fue desarrollado originalmente por [Sun Microsystems](http://es.wikipedia.org/wiki/Sun_Microsystems" \t "Sun Microsystems). Actualmente el propietario es la empresa Oracle.

La máquina virtual Java es en última instancia la que convierte el código bytecode a código nativo del dispositivo final.

La gran **ventaja** de la máquina virtual java es aportar portabilidad al lenguaje de manera que desde Sun Microsystems se han creado diferentes máquinas virtuales java para diferentes arquitecturas y así un programa (extensión .class) escrito en un [Windows](http://es.wikipedia.org/wiki/Windows" \t "Windows) puede ser interpretado en un entorno [Linux](http://es.wikipedia.org/wiki/Linux" \t "Linux). Tan solo es necesario disponer de dicha máquina virtual para dichos entornos.

*En* ***resumen****:*

Una de las características más relevantes del lenguaje de programación Java es el hecho de que utiliza una [máquina virtual](http://es.wikipedia.org/wiki/Máquina_virtual_Java" \t "_blank) para la ejecución de los programas, haciendo esta de intermediaria entre la máquina real (HW + SO) y los programas desarrollados por los programadores.

Esto supuso una gran ventaja a la hora de poder distribuir un mismo código ya compilado en diferentes tipos de máquinas físicas con diferentes sistemas operativos. Basta con compilar el código fuente de alto nivel a [código intermedio](http://es.wikipedia.org/wiki/Compilador" \l "_blank) (que es código máquina para la JVM), también llamado [bytecode](http://es.wikipedia.org/wiki/Bytecode" \t "_blank), para que pudiera ser empleado en máquinas virtuales que corren sobre máquinas físicas.

<https://www.adictosaltrabajo.com/tutoriales/byte-code>

## Lenguajes de programación.

### Tipos de lenguajes de programación.

Los lenguajes de programación se pueden clasificar según varios criterios. Se pueden encontrar hasta 11 criterios diferentes: Nivel de abstracción, propósito, evolución histórica, manera de ejecutarse, manera de abordar la tarea a realizar, paradigma de programación (nos va a interesar), lugar de ejecución, concurrencia, interactividad, realización visual y determinismo.

Hay que tener en cuenta también que en la práctica, la mayoría de lenguajes no pueden ser puramente clasificados en una categoría, pues surgen incorporando ideas de otros lenguajes y de otras filosofías de programación, pero no importa al establecer las clasificaciones, pues el auténtico objetivo de las mismas es mostrar los rangos, las posibilidades y tipos de lenguajes que hay.

**1. Nivel de abstracción.**

Según el nivel de abstracción, según el grado de cercanía a la máquina:

* **Lenguajes de [bajo nivel](http://es.wikipedia.org/wiki/Lenguajes_de_bajo_nivel):** La programación se realiza teniendo muy en cuenta las características del procesador. Ejemplo: Lenguajes tipo ensamblador.
* **Lenguajes de nivel medio:** Permiten un mayor grado de abstracción pero al mismo tiempo mantienen algunas cualidades de los lenguajes de bajo nivel. Ejemplo: C puede realizar operaciones lógicas y de desplazamiento con bits.
* **Lenguajes de [alto nivel](http://es.wikipedia.org/wiki/Lenguajes_de_alto_nivel):** Más parecidos al lenguaje humano. Manejan conceptos, tipos de datos, etc., de una manera cercana al pensamiento humano ignorando (abstrayéndose) del funcionamiento de la máquina. Ejemplos: Java, Ruby, PHP, Pascal, LISP, ADA.

Hay quien sólo considera lenguajes de bajo nivel y de alto nivel, (en ese caso, C es considerado de alto nivel).

**2. Propósito.**

Según el propósito, es decir, el tipo de problemas a tratar con ellos:

* **Lenguajes de propósito [general](http://es.wikipedia.org/wiki/Lenguajes_de_propósito_general):** Aptos para todo tipo de tareas. Ejemplo: C.
* **Lenguajes de propósito específico:** Hechos para un objetivo muy concreto. Ejemplo: [Csound](http://en.wikipedia.org/wiki/Csound) (para crear ficheros de audio).
* **Lenguajes de [programación de sistemas](http://en.wikipedia.org/wiki/Systems_programming):** Diseñados para realizar sistemas operativos o drivers. Ejemplo: [C](http://es.wikipedia.org/wiki/Lenguaje_de_programación_C).
* **Lenguajes de [script](http://en.wikipedia.org/wiki/Scripting_language):** Para realizar tareas varias de control y auxiliares. Antiguamente eran los llamados lenguajes de procesamiento por lotes (batch) o [JCL](http://en.wikipedia.org/wiki/Job_Control_Language) (“Job Control Languages”). Se subdividen en varias clases (de shell, de GUI, de programación web, etc.). Ejemplos: [bash](http://es.wikipedia.org/wiki/Bash) (shell), [JavaScript](http://es.wikipedia.org/wiki/JavaScript) (programación web).

**3. Evolución histórica.**

Con el paso del tiempo, se va incrementando el nivel de abstracción, pero en la práctica, los de una generación no terminan de sustituir a los de la anterior:

* **Lenguajes de** [primera generación (1GL)](http://en.wikipedia.org/wiki/First-generation_programming_language): Código máquina.
* **Lenguajes de** [segunda generación (2GL)](http://en.wikipedia.org/wiki/Second-generation_programming_language): Lenguajes ensamblador.
* **Lenguajes de** [tercera generación (3GL)](http://en.wikipedia.org/wiki/Third-generation_programming_language): La mayoría de los lenguajes modernos, diseñados para facilitar la programación a los humanos. Ejemplos: C, Java.
* **Lenguajes de** [cuarta generación (4GL)](http://en.wikipedia.org/wiki/Fourth-generation_programming_language)**:** se ha dado este nombre a ciertas herramientas que permiten construir aplicaciones sencillas combinando piezas prefabricadas. Hoy se piensa que estas herramientas no son, propiamente hablando, lenguajes. Algunos proponen reservar el nombre de cuarta generación para la [programación orientada a objetos](http://es.wikipedia.org/wiki/Programación_orientada_a_objetos" \t "Programación orientada a objetos).
* **Lenguajes de** [quinta generación (5GL)](http://en.wikipedia.org/wiki/Fifth-generation_programming_language)**:** La intención es que el programador establezca qué problema ha de ser resuelto y las condiciones a reunir, y la máquina lo resuelve. Se usan en inteligencia artificial. Ejemplo: [Prolog](http://es.wikipedia.org/wiki/Prolog).

**4. Manera de ejecutarse.** **(“INTERESANTE RECALCARLO”)**

Según la manera de ejecutarse:

* Lenguajes [compilados](http://es.wikipedia.org/wiki/Lenguajes_compilados): Un programa traductor traduce el código del programa (código fuente) en código máquina (código objeto). Otro programa, el enlazador, unirá los ficheros de código objeto del programa principal con los de las librerías para producir el programa ejecutable. Ejemplo: **C**, Pascal, Fortran.
* Lenguajes [interpretados](http://es.wikipedia.org/wiki/Lenguajes_interpretado): Un programa (intérprete), ejecuta las instrucciones del programa de manera directa. Ejemplo: [Lisp](http://en.wikipedia.org/wiki/Lisp_(programming_language)), **Javascript**.
* También los hay **mixtos**, como Java, que primero pasan por una fase de compilación en la que el código fuente se transforma en “**bytecode**”, y este “bytecode” puede ser ejecutado luego (interpretado) en ordenadores con distintas arquitecturas (procesadores) que tengan todos instalados la [“máquina virtual”](http://en.wikipedia.org/wiki/Java_virtual_machine) **[Java](http://en.wikipedia.org/wiki/Java_virtual_machine)**.

**5. Manera de abordar la tarea a realizar.**

Según la manera de abordar la tarea a realizar, pueden ser:

* Lenguajes [imperativos](http://en.wikipedia.org/wiki/Imperative_programming_language): Indican cómo hay que hacer la tarea, es decir, expresan los pasos a realizar. Ejemplo: C.
* Lenguajes [declarativos](http://en.wikipedia.org/wiki/Declarative_programming_language): Indican qué hay que hacer. Ejemplos: Lisp, Prolog. Otros ejemplos de lenguajes declarativos, pero que no son lenguajes de programación propiamente dicha, son [HTML](http://es.wikipedia.org/wiki/HTML) (para describir páginas web) o [SQL](http://es.wikipedia.org/wiki/SQL) (para consultar bases de datos).

**6. Paradigma de programación. (“IMPORTANTE RECALCARLO”)**

El [paradigma de programación](http://en.wikipedia.org/wiki/Programming_paradigm) es el estilo de programación empleado. Los principales son:

* Lenguajes de[programación procedural](http://en.wikipedia.org/wiki/Procedural_programming): Divide el problema en partes más pequeñas, que serán realizadas por subprogramas (subrutinas, funciones, procedimientos), que se llaman unas a otras para ser ejecutadas. Ejemplos: C, Pascal.
* Lenguajes de [programación orientada a objetos](http://es.wikipedia.org/wiki/Programación_orientada_a_objetos): Crean un sistema de clases y objetos siguiendo el ejemplo del mundo real, en el que unos objetos realizan acciones y se comunican con otros objetos. Ejemplos: C++, Java.
* Lenguajes de [programación funcional](http://es.wikipedia.org/wiki/Programación_funcional): La tarea se realiza evaluando funciones (como en Matemáticas) de manera recursiva. Ejemplo: Lisp.
* Lenguajes de [programación lógica](http://es.wikipedia.org/wiki/Programación_lógica): La tarea a realizar se expresa empleando lógica formal matemática. Ejemplo: Prolog.

**7. Lugar de ejecución.**

En [sistemas distribuidos](http://es.wikipedia.org/wiki/Computación_distribuida), según dónde se ejecute:

* Lenguajes de [servidor](http://es.wikipedia.org/wiki/Servidor): Se ejecutan en el servidor. Ejemplo: [PHP](http://www.php.net/) es el más utilizado en servidores web.
* Lenguajes de [cliente](http://es.wikipedia.org/wiki/Cliente_(informática)): Se ejecutan en el cliente. Ejemplo: JavaScript en navegadores web.

**8. Concurrencia.**

Según admitan o no concurrencia de procesos, esto es, la ejecución simultánea de varios procesos lanzados por el programa:

* Lenguajes [concurrentes](http://es.wikipedia.org/wiki/Programación_concurrente). Ejemplo: [Ada](http://es.wikipedia.org/wiki/Lenguaje_de_programación_Ada), Java.
* Lenguajes **no concurrentes**. Ejemplo: C.

**9. Interactividad.**

Según la interactividad del programa con el usuario u otros programas:

* Lenguajes **orientados a sucesos (eventos)**: El flujo del programa es controlado por la interacción con el usuario o por mensajes de otros programas/sistema operativo, como editores de texto, interfaces gráficos de usuario (GUI) o kernels. Ejemplo: [VisualBasic](http://es.wikipedia.org/wiki/Microsoft_Visual_Basic), lenguajes de programación declarativos.
* Lenguajes **no orientados a sucesos (eventos)**: El flujo del programa no depende de sucesos exteriores, sino que se conoce de antemano, siendo los procesos batch el ejemplo más claro (actualizaciones de bases de datos, colas de impresión de documentos, etc.). Ejemplos: Lenguajes de programación imperativos.

**10. Realización visual.**

Según la realización visual o no del programa:

* Lenguajes de [programación visual](http://en.wikipedia.org/wiki/Visual_programming): El programa se realiza moviendo bloques de construcción de programas (objetos visuales) en un interfaz adecuado para ello. No confundir con entornos de programación visual, como Microsoft Visual Studio y sus lenguajes de programación textuales (como Visual C#). Ejemplo: [Mindscript](http://mindscript.familjemarknaden.se/doc/mindscript.introduction.html" \l "whatisms), Alice, Scratch.
* Lenguajes de **programación textual**: El código del programa se realiza escribiéndolo. Ejemplos: C, Java, Lisp.

**11. Determinismo.**

Según se pueda predecir o no el siguiente estado del programa a partir del estado actual:

* Lenguajes **deterministas**. Ejemplos: Todos los anteriores.
* Lenguajes [probabilísticos o](http://en.wikipedia.org/wiki/Esoteric_programming_language" \l "Nondeterministic_language) no [deterministas](http://en.wikipedia.org/wiki/Esoteric_programming_language" \l "Nondeterministic_language)**:** Sirven para explorar grandes espacios de búsqueda, (como gramáticas), y en la investigación teórica de hipercomputación. Ejemplo: [mutt](http://www.schneertz.com/rmutt/) (generador de texto aleatorio).
  1. **Características de los lenguajes más difundidos a lo largo del tiempo.**

**BASIC**. Durante mucho [tiempo](http://www.monografias.com/trabajos901/evolucion-historica-concepciones-tiempo/evolucion-historica-concepciones-tiempo.shtml) se ha considerado un buen lenguaje para comenzar a aprender, por su sencillez, aunque se podía tender a crear programas poco legibles. A pesar de esta "sencillez" hay versiones muy potentes, incluso para programar en entornos [gráficos](http://www.monografias.com/trabajos11/estadi/estadi.shtml" \l "METODOS) como [Windows](http://www.monografias.com/trabajos15/ms-windows/ms-windows.shtml).

**COBOL**. Fue muy utilizado para [negocios](http://www.monografias.com/trabajos15/plan-negocio/plan-negocio.shtml) (para crear [software](http://www.monografias.com/Computacion/Software/) de [gestión](http://www.monografias.com/trabajos15/sistemas-control/sistemas-control.shtml) que tuviese que manipular grandes cantidades de [datos](http://www.monografias.com/trabajos11/basda/basda.shtml)). Sigue usándose en el campo de la Banca.

**FORTRAN**. Concebido para [ingeniería](http://www.monografias.com/trabajos14/historiaingenieria/historiaingenieria.shtml), [operaciones](http://www.monografias.com/trabajos6/diop/diop.shtml) [matemáticas](http://www.monografias.com/Matematicas/index.shtml), etc. También va quedando desplazado.

**ENSAMBLADOR.**  Muy cercano al código máquina (es un lenguaje de "bajo nivel"), pero sustituye las secuencias de ceros y unos (bits) por palabras más fáciles de recordar, como MOV, ADD, CALL o JMP.

**PASCAL**. El lenguaje estructurado por excelencia, y que en algunas versiones tiene una potencia comparable a la del [lenguaje C](http://www.monografias.com/trabajos4/lenguajec/lenguajec.shtml).

**C**. Uno de los mejor considerados actualmente (junto con C++ y [Java](http://www.monografias.com/trabajos16/java/java.shtml), que mencionaremos a continuación), porque no es demasiado difícil de aprender y permite un grado de [control](http://www.monografias.com/trabajos14/control/control.shtml) del ordenador muy alto, combinando características de lenguajes de alto y bajo nivel. Además, es muy transportable: existe un **estándar**, el **ANSI C**, lo que asegura que se pueden convertir programas en C de un ordenador a otro o de un [sistema operativo](http://www.monografias.com/Computacion/Sistemas_Operativos/) a otro con bastante menos esfuerzo que en otros lenguajes.

**C++**. Un lenguaje desarrollado a partir de C, que permite [Programación Orientada a Objetos](http://www.monografias.com/trabajos/objetos/objetos.shtml).

**Java**. Desarrollado a su vez a partir de C++, elimina algunos de sus inconvenientes y ha alcanzado una gran difusión gracias a su uso en [Internet](http://www.monografias.com/Computacion/Internet/). Al igual que C++ es Orientado a Objetos.

**Python**.

**Kotlin**.

## Introducción a la ingeniería del software. - HA DICHO QUE NOS LO SALTAMOS

### Proceso software y ciclo de vida del software

El software de ordenadores es el producto que diseñan y construyen los ingenieros del software. El software es un elemento lógico, no físico. El software tiene unas características considerablemente distintas a las del hardware:

1. El software se desarrolla no se fabrica.
2. La mayoría del software se realiza a medida.

La **Ingeniería del Software** es una disciplina o área de la Informática que ofrece métodos y técnicas para desarrollar y mantener software de calidad.

La Ingeniería del Software aborda todas las fases de ciclo de vida de cualquier tipo de sistema de información.

**Sistema de Información**. '[Sistema](http://es.wikipedia.org/wiki/Sistema" \t "Sistema) de información' (SI) es un conjunto de elementos orientados al tratamiento y administración de [datos](http://es.wikipedia.org/wiki/Dato" \t "Dato) e [información](http://es.wikipedia.org/wiki/Información" \t "Información), organizados y listos para su posterior uso, generados para cubrir una necesidad (objetivo). Dichos elementos formarán parte de alguna de estas categorías:

* [Personas](http://es.wikipedia.org/wiki/Persona" \t "Persona).
* [Datos](http://es.wikipedia.org/wiki/Dato" \t "Dato).
* [Actividades](http://es.wikipedia.org/wiki/Actividad" \t "Actividad) o técnicas de trabajo.
* Recursos materiales en general, típicamente [recursos informáticos y de comunicación](http://es.wikipedia.org/wiki/Tecnologías_de_la_información_y_la_comunicación" \t "Tecnologías de la información y la comunicación), aunque no tienen por qué ser de este tipo obligatoriamente.

Existen múltiples definiciones de Ingeniería del Software, casi tantas como autores que han escrito sobre el tema. Vamos a mencionar la de **IEEE** (“Institute of Electrical and Electronics Engineers”):

“Ingeniería del Software es la aplicación de un enfoque sistemático, disciplinado y cuantificable hacia el desarrollo, operación y mantenimiento del software. Es decir, la aplicación de la Ingeniería al software”.

El **ciclo de vida** de un sistema abarca toda la vida del sistema, desde su concepción hasta que deja de utilizarse. Se habla también de ciclo de vida del desarrollo software, abarcando las etapas que comienzan en el análisis y finalizan con la entrega del sistema al usuario.

### Fases del desarrollo de una aplicación

#### 3.2.1. Análisis

Para comprender la naturaleza de los programas a construir el analista, (= ingeniero del software) debe comprender el dominio de información del software, así como las funciones requeridas, comportamiento, rendimiento e interconexión.

En esta fase, el analista ha de recoger en colaboración con el cliente, toda la información relativa a los requisitos del *sistema a desarrollar*: su función, rendimiento e interfaces. Hay que analizar costes, recursos y definir tiempos para las siguientes tareas a realizar.

#### 3.2.2. Diseño

El diseño del software es realmente un proceso de muchos pasos que se centra en cuatro atributos distintos de programa:

* Estructura de datos
* Arquitectura de software
* Representaciones de interfaz
* Algoritmo (detalle procedimental)

“El proceso de diseño traduce requisitos en una representación del software donde se pueda evaluar su calidad antes de que comience la codificación”.

#### 3.2.3. Codificación

El diseño se debe traducir en una forma legible por la máquina. Esta tarea se lleva a cabo en el paso de Generación de código (Codificación). Si el diseño se ha realizado de una forma detallada, la generación de código se realiza mecánicamente.

#### 3.2.4. Pruebas

Cuando se ha finalizado la generación del código comienzan las pruebas del programa. El proceso de pruebas se centra en los procesos lógicos internos del software. Se realizan las pruebas para la detección de errores y para asegurar que la entrada definida produce resultados reales de acuerdo con los resultados requeridos.

#### 3.2.5. Documentación

Cada una de las fases por las que pasa un proyecto de Software debe ir acompañada de la correspondiente documentación.

#### 3.2.6 Explotación

La etapa de explotación transcurre una vez que el sistema está a disposición del usuario.

#### 3.2.7. Mantenimiento

El software una vez entregado sufrirá cambios. Los motivos de estos cambios pueden ser:

* Errores que pudieran encontrarse.
* El software debe adaptarse a cambios en su entorno externo (por ejemplo un cambio de sistema operativo).
* El software requiere mejoras funcionales o de rendimiento.

El proceso de soporte y mantenimiento implica que el programa que se va a modificar tendrá que pasar por cada una de las fases precedentes.

### Modelos de proceso de desarrollo software

La estrategia de desarrollo (modelo o proceso) que se elige para realizar una aplicación concreta recibe el nombre de:

**Ciclo de vida = modelo de desarrollo**

Cada modelo de desarrollo representa un intento de ordenar la actividad de desarrollo del software. Todos los modelos esperan contribuir al control y la coordinación de un proyecto de software real.

Dependiendo de la naturaleza de un proyecto de desarrollo, puede haber razones para adoptar un método en lugar de otro o incluso una combinación de ellos.

#### 3.3.1 Cascada

En este modelo se distinguen una serie de fases, que pueden diferir en algunos autores y libros en nombre y número pero todas tienen en común dos aspectos:

* Implantación ascendente.
* Progresión lineal y secuencial de una fase a otra.



Razones por las que a veces no se debe utilizar el modelo lineal:

* A veces es difícil que el usuario declare explícitamente todos los requisitos al principio del proyecto. El modelo lineal secuencial requiere que esto ocurra y resulta complicado compaginarlo con la incertidumbre existente al comienzo de muchos proyectos.
* Una versión visible del trabajo realizado en el proyecto no estará disponible hasta que todo el trabajo esté realizado, haciendo que si ocurre un error ya sea demasiado tarde.

A pesar de estos problemas el modelo lineal secuencial proporciona una plantilla para el desarrollo del proyecto. Siempre es mejor que realizar un desarrollo al azar.

El modelo lineal es el modelo más antiguo y el que más se ha utilizado en la ingeniería del software.

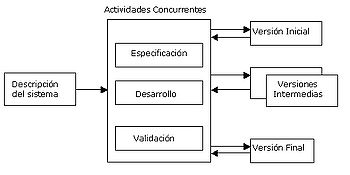
Cuándo se debe utilizar:

“El modelo lineal se puede utilizar y es aconsejable utilizarlo cuando los requisitos se han entendido y definido correctamente”.

#### 3.3.2. Evolutivo

En el desarrollo en cascada no se tiene en cuenta la naturaleza evolutiva del software. Se debe de tener una especificación totalmente detallada de **TODOS** los requerimientos que debe satisfacer el software que desarrollemos para poder iniciar las diferentes etapas de desarrollo.

El desarrollo evolutivo se basa en la idea de desarrollar una implementación inicial e ir refinándola a través de diferentes versiones hasta desarrollar un sistema software que satisfaga todos los requerimientos del cliente.



Un enfoque evolutivo para el desarrollo de software suele ser más efectivo que el desarrollo en cascada ya que desde un principio se le entrega al cliente una versión que satisface los requerimientos principales.

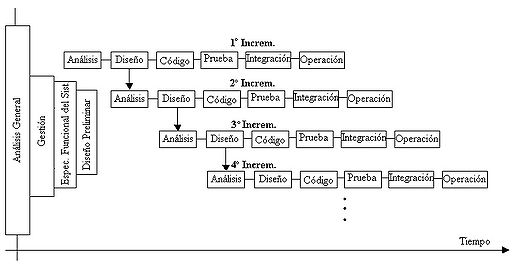
**El modelo iterativo incremental y el modelo en espiral son dos modelos de tipo evolutivo.**

#### 3.3.3. Iterativo

**Como ya hemos dicho el modelo en cascada requiere tener una especificación de requerimientos totalmente detallada para poder iniciar la etapa del diseño. Si a lo largo del proceso de desarrollo se cambian los requisitos hay que rehacer parte del trabajo de diseño e implementación para poder hacer frente a los nuevos cambios.**

**El desarrollo evolutivo permite que los requerimientos no estén totalmente especificados para comenzar con el desarrollo del software, esto hace que el software desarrollado pueda estar mal estructurado y sea difícil de mantener pero permite adaptarse a los cambios en los requisitos.**

**Vamos a ver un modelo de proceso basado en el desarrollo evolutivo: el modelo iterativo incremental (no lo entiendo).**



**Siguiendo este modelo, los clientes especifican a grandes rasgos los servicios que tiene que proporcionar el sistema software que se quiere desarrollar. Entonces se definen varios incrementos para desarrollar cada uno un subconjunto de la funcionalidad del sistema software dando prioridad a los requerimientos más importantes.**

**Cada incremento puede desarrollarse siguiendo un modelo en cascada típico, por lo que al inicio de cada incremento se debe tener una especificación detallada de la funcionalidad que se pretende desarrollar en ese incremento.**

**Cuando el incremento se completa se entrega y los clientes pueden utilizarlo a la espera del siguiente incremento. Esto significa que los clientes obtienen una parte de la funcionalidad del sistema software que necesitan de manera temprana, y se quedan a la espera de nuevos incrementos que mejoren la funcionalidad del sistema software.**

**Este proceso de desarrollo incremental tiene varias ventajas en contraste al desarrollo en cascada:**

**- Los clientes no tienen que esperar hasta la etapa final para sacar provecho del sistema software ya que el primer incremento satisface los requerimientos principales.**

**- Los clientes pueden utilizar cada incremento para analizar nuevos requerimientos para incrementos posteriores.**

**- Existe un bajo riesgo de un fallo total del proyecto, ya que los errores encontrados en un incremento pueden arreglarse en el incremento posterior.**

**Este enfoque de desarrollo es aconsejable si el software que se pretende desarrollar posee una serie de funcionalidades bien definidas que se pueden desarrollar con total independencia.**

### Metodologías de desarrollo software

***Metodología de desarrollo software*: Conjunto de procedimientos, técnicas, herramientas y un soporte documental que ayuda a los desarrolladores a realizar nuevo software.**

#### Características

* **Existencia de reglas predefinidas**
* **Cobertura total del ciclo de desarrollo**
* **Verificaciones intermedias**
* **Planificación y control**
* **Comunicación efectiva**
* **Utilización sobre un abanico amplio de proyectos**
* **Fácil formación**
* **Uso de herramientas CASE**
* **Actividades que mejoren el proceso de desarrollo**
* **Soporte al mantenimiento**
* **Soporte de la reutilización de software**

#### Técnicas

***Técnica*: Definición de la forma de ejecutar una tarea.**

**En las distintas fases de una metodología se pueden usar técnicas como:**

* **Diagramas de casos de uso**
* **Diagramas de clase**
* **Diagramas de Flujo de datos**
* **Diagramas de estructura**

#### Objetivos

**Una metodología puede seguir uno o varios modelos de ciclo de vida, es decir, el ciclo de vida indica qué es lo que hay que obtener a lo largo del desarrollo del proyecto pero no cómo hacerlo.**

**El objetivo de las metodologías es indicar cómo hay que obtener los distintos productos parciales y finales.**

#### Tipos de metodologías

* **Metodologías estructuradas:**
  + **Orientadas a procesos.**
  + **Orientadas a datos.**
  + **Mixtas.**
* **Metodologías orientadas a Objetos.**

### Herramientas CASE (Computer Aided Software Engineering).

***C****omputer* ***A****ided* ***S****oftware* ***E****ngineering*

Se puede definir a las Herramientas CASE como un conjunto de programas y ayudas que dan asistencia a los analistas, ingenieros de software y desarrolladores, durante todos los pasos del Ciclo de Vida de desarrollo de un Software: Análisis, Diseño, Implementación e Instalación.

Una herramienta CASE suele incluir:

* Un **[diccionario](http://www.monografias.com/trabajos12/diccienc/diccienc.shtml) de datos** para almacenar información sobre los datos de la aplicación.
* **Herramientas de diseño** para dar apoyo al análisis de datos.
* **Herramientas** que permitan desarrollar el **[modelo](http://www.monografias.com/trabajos/adolmodin/adolmodin.shtml) de datos** corporativo, así como los esquemas conceptual y lógico.
* **Herramientas** para desarrollar los **prototipos** de las aplicaciones.

El uso de las herramientas CASE puede mejorar la [productividad](http://www.monografias.com/trabajos6/prod/prod.shtml) en el desarrollo de una aplicación de bases de datos.http://www.monografias.com/images04/trans.gif

**3.6. Ejemplos de Herramientas Case más utilizadas.**

**ERwin:**

ERwin es una herramienta para el diseño de bases de datos, que proporciona [productividad](http://www.monografias.com/trabajos6/prod/prod.shtml) en el diseño, generación, y mantenimiento de aplicaciones. Desde un modelo lógico de los requerimientos de información, hasta el modelo físico perfeccionado para las características específicas de la base de datos diseñada, además ERwin permite visualizar la estructura, los elementos importantes, y optimizar el diseño de la base de datos. Genera automáticamente las tablas y miles de líneas de procedimientos almacenados y disparadores para los principales tipos de base de datos.

**EasyCASE**

EasyCASE Profesional - Es un [producto](http://www.monografias.com/trabajos12/elproduc/elproduc.shtml) para la generación de esquemas de base de datos e ingeniería inversa - trabaja para proveer una solución comprensible para el diseño, consistencia y documentación del sistema en conjunto.

Esta herramienta permite automatizar las fases de análisis y diseño dentro del desarrollo de una aplicación, para poder crear las aplicaciones eficazmente – desde el procesamiento de transacciones a la aplicación de bases de datos de [cliente](http://www.monografias.com/trabajos11/sercli/sercli.shtml)/[servidor](http://www.monografias.com/trabajos12/rete/rete.shtml), así como sistemas de tiempo real.

**Oracle Designer:**

Oracle Designer es un conjunto de herramientas para guardar las definiciones que necesita el usuario y automatizar la construcción rápida de aplicaciones cliente/servidor gráficas. Integrado con Oracle Developer, Oracle Designer provee una solución para desarrollar sistemas empresariales de segunda generación.http://www.monografias.com/images04/trans.gif

Todos los datos ingresados por cualquier herramienta de Oracle Designer, en cualquier fase de desarrollo, se guardan en un repositorio central, habilitando [el trabajo](http://www.monografias.com/trabajos/fintrabajo/fintrabajo.shtml) fácil del equipo y la [dirección](http://www.monografias.com/trabajos15/direccion/direccion.shtml) del proyecto.

En el lado del Servidor, Oracle Designer soporta la definición, generación y captura de diseño de los siguientes tipos de bases de datos, por conexión de Oracle:

## Proceso de obtención de código ejecutable a partir del código fuente, herramientas implicadas

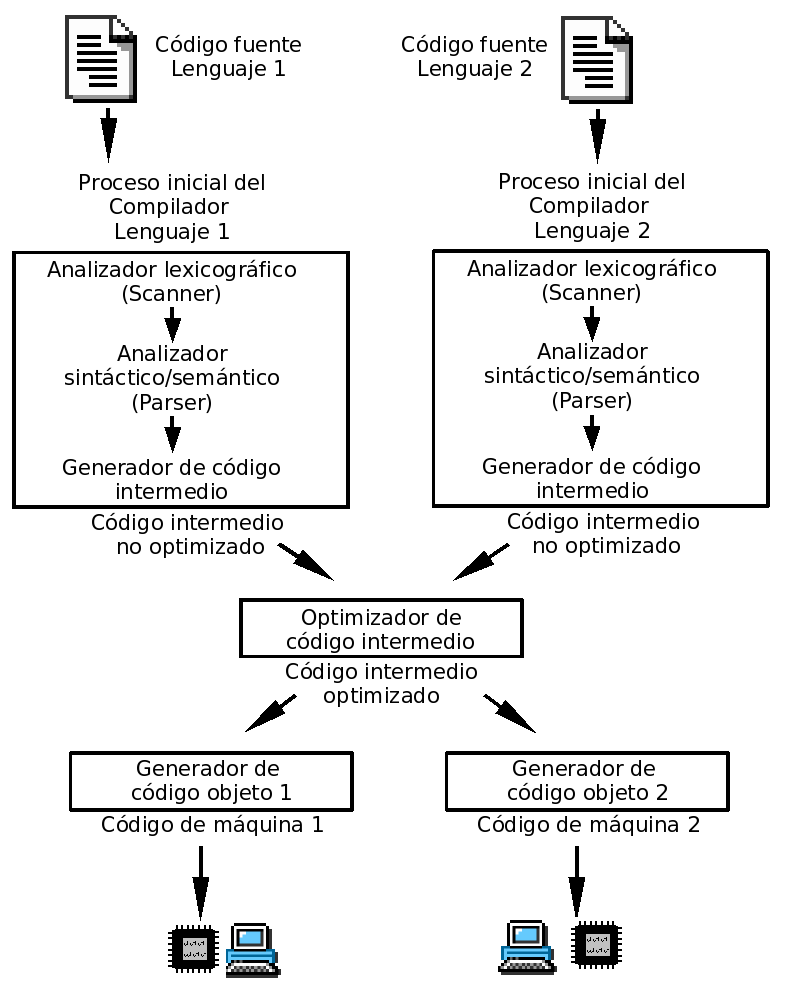
### Editores

Un editor es un programa informático que nos permite crear, modificar (editar) y guardar archivos de texto en la memoria secundaria del ordenador.

Los editores de programas especializados nos ofrecen distintas facilidades para escribir el código: marcar las palabras reservadas en distinto color, terminar las palabras reservadas automáticamente, ayuda online, etc.

### Compiladores

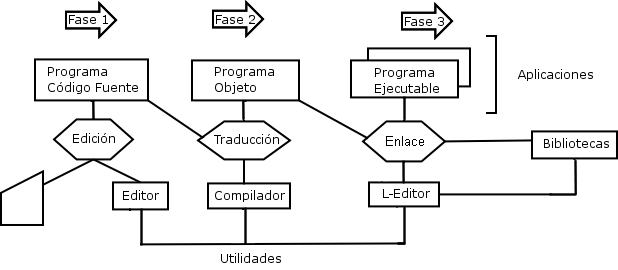
Un **compilador** es un [programa informático](http://es.wikipedia.org/wiki/Programa_informático" \t "Programa informático) que [traduce un programa](http://es.wikipedia.org/wiki/Proceso_de_traducción_de_programas" \t "Proceso de traducción de programas) escrito en un [lenguaje de programación](http://es.wikipedia.org/wiki/Lenguaje_de_programación" \t "Lenguaje de programación) a otro lenguaje de programación, generando un programa equivalente que la máquina será capaz de interpretar. Usualmente el segundo lenguaje es [lenguaje de máquina](http://es.wikipedia.org/wiki/Lenguaje_de_máquina" \t "Lenguaje de máquina), pero también puede ser un lenguaje intermedio (bytecode) o simplemente texto. Este proceso de traducción se conoce como **compilación**.



### Enlazadores

En [programación](http://www.alegsa.com.ar/Dic/programacion.php), un **enlazador** (en [inglés](http://es.wikipedia.org/wiki/Idioma_inglés" \t "Idioma inglés), *linker*) es un módulo o programa que une los ficheros de [código objeto](http://www.alegsa.com.ar/Dic/codigo objeto.php) (generados en la [compilación](http://www.alegsa.com.ar/Dic/compilar.php)), la información de todos los recursos necesarios (librerías/bibliotecas), elimina los recursos que no se necesitan y enlaza el código objeto con sus librerías. Finalmente produce el fichero ejecutable o una librería.

Existen programas que se enlazan dinámicamente, esto significa que este proceso se hace en el momento que se carga el programa.



## Errores en el desarrollo de programas

En el proceso de desarrollo de un programa, así como durante todo el ciclo de vida de una aplicación, se pueden producir distintos errores. Estos errores pueden ser:

**Errores sintácticos.** Se detectarán en la fase de compilación del código.

**Errores de ejecución.** Se detectarán una vez compilado el programa, cuando intentemos ejecutarlo. Un ejemplo de esto es la división por cero. Suponga que tiene la instrucción siguiente:

velocidad = kilometros / horas

Si la variable horas tiene un valor de 0, se produce un error en tiempo de ejecución en la operación de división. El programa se debe ejecutar para que se pueda detectar este error y si horas contiene un valor válido, no se producirá el error.

**Errores lógicos.** Los errores lógicos son errores que impiden que un programa haga lo que estaba previsto. El código puede compilarse y ejecutarse sin errores, pero el resultado de una operación puede generar un resultado no esperado.

Por ejemplo, puede tener una variable llamada *nombre* y establecida inicialmente en una cadena vacía. Después en el programa, puede concatenar nombre con otra variable denominada *apellido* para mostrar un nombre completo. Si olvida asignar un valor a nombre, sólo se mostrará el apellido, no el nombre completo como pretendía.

Los errores lógicos son los más difíciles de detectar y corregir, las herramientas de depuración facilitan el trabajo de encontrar los errores lógicos.

<http://wiki.elhacker.net/bugs-y-exploits/introduccion/errores-de-programacion-comunes>

## Importancia de la reutilización de código

En [programación](http://www.alegsa.com.ar/Dic/programacion.php), reutilización de código es el uso de software existente para desarrollar un nuevo software. La reutilización de código ha sido empleada desde los primeros días de la [programación](http://www.alegsa.com.ar/Dic/programacion.php). Los [programadores](http://www.alegsa.com.ar/Dic/programador.php) siempre han reusado partes de un código, hojas de cálculo, funciones o procedimientos.

La idea es que parte o todo el código de un programa informático escrito una vez, sea o pueda ser usado en otros programas. La reutilización de código es una técnica común que intenta ahorrar tiempo y energía, reduciendo el trabajo redundante.

Las [bibliotecas](http://www.alegsa.com.ar/Dic/biblioteca.php) o librerías de software son un buen ejemplo. Al utilizarlas se está reutilizando código.

El software más fácilmente reutilizable tiene ciertas características: [modularidad](http://www.alegsa.com.ar/Dic/modularidad.php), [bajo acoplamiento](http://www.alegsa.com.ar/Dic/acoplamiento.php), [alta cohesión](http://www.alegsa.com.ar/Dic/cohesion.php), [ocultación de información](http://www.alegsa.com.ar/Dic/ocultacion de informacion.php), etc.

# UT1 Ejercicios

## 1. Secuenciales

**Ejercicio 1**Escribir un programa que pregunte al usuario su nombre, y luego le salude.

**Ejercicio 2**Calcular el perímetro y área de un rectángulo dada su base y su altura.

**Ejercicio 3**Dados los catetos de un triángulo rectángulo, calcular su hipotenusa.

**Ejercicio 4**Dados dos números, mostrar la suma, resta, división y multiplicación de ambos.

**Ejercicio 5**Escribir un programa que convierta un valor dado en grados Fahrenheit a grados Celsius. Recordad que la fórmula para la conversión es:

C = (F-32) \* 5 / 9

**Ejercicio 6**Calcular la media de tres números pedidos por teclado.

**Ejercicio 7**

Realiza un programa que reciba una cantidad de minutos y muestre por pantalla a cuántas horas y minutos corresponde. Por ejemplo: 1000 minutos son 16 horas y 40 minutos.

**Ejercicio 8**Un vendedor recibe un sueldo base más un 10% extra por comisión de sus ventas (se incrementa su sueldo en un 10% de la cantidad vendida), el vendedor desea saber cuánto dinero obtendrá por concepto de comisiones por las tres ventas que realiza en el mes y el total que recibirá en el mes tomando en cuenta su sueldo base y comisiones.

**Ejercicio 9**Una tienda ofrece un descuento del 15% sobre el total de la compra y un cliente desea saber cuánto deberá pagar finalmente por su compra.

**Ejercicio 10**Un alumno desea saber cuál será su calificación final en la materia de Algoritmos. Dicha calificación se compone de los siguientes porcentajes:

• 55% del promedio de sus tres calificaciones parciales.  
• 30% de la calificación del examen final.  
• 15% de la calificación de un trabajo final.

**Ejercicio 11**Pide al usuario dos números y muestra la “distancia” entre ellos (el valor absoluto de su diferencia, de modo que el resultado sea siempre positivo).

**Ejercicio 12**Pide al usuario dos pares de números x1, y2 y x2, y2, que representen dos puntos en el plano. Calcula y muestra la distancia entre ellos.

Diagrama

Descripción generada automáticamente

**Ejercicio 13**Realizar un algoritmo que lea un número y que muestre su raíz cuadrada y su raíz cúbica. PSeInt no tiene ninguna función predefinida que permita calcular la raíz cúbica, ¿Cómo se puede calcular?

**Ejercicio 14**Dado un número de dos cifras, diseñe un algoritmo que permita obtener el número invertido. Ejemplo, si se introduce 23 que muestre 32.

**Ejercicio 15**Dadas dos variables numéricas A y B, que el usuario debe teclear, se pide realizar un algoritmo que intercambie los valores de ambas variables y muestre cuánto valen al final las dos variables.

**Ejercicio 16**Dos vehículos viajan a diferentes velocidades (v1 y v2) y están distanciados por una distancia d. El que está detrás viaja a una velocidad mayor. Se pide hacer un algoritmo para ingresar la distancia entre los dos vehículos (km) y sus respectivas velocidades (km/h) y con esto determinar y mostrar en que tiempo (minutos) alcanzará el vehículo más rápido al otro.

**Ejercicio 17**Un ciclista parte de una ciudad A a las HH horas, MM minutos y SS segundos. El tiempo de viaje hasta llegar a otra ciudad B es de T segundos. Escribir un algoritmo que determine la hora de llegada a la ciudad B, también en HH horas, MM minutos y SS segundos.

**Ejercicio 18**Pedir el nombre y los dos apellidos de una persona y mostrar las iniciales.

**Ejercicio 19**Escribir un algoritmo para calcular la nota final de un estudiante, considerando que: por cada respuesta correcta 5 puntos, por una incorrecta -1 y por respuestas en blanco 0. Imprime el resultado obtenido por el estudiante.

**Ejercicio 20**Diseñar un algoritmo que nos diga el dinero que tenemos (en euros y céntimos) después de pedirnos cuantas monedas tenemos (de 2€, 1€, 50 céntimos, 20 céntimos o 10 céntimos).

## 2. Condicionales (estructuras alternativas)

**Ejercicio 1**Algoritmo que pida dos números e indique si el primero es mayor que el segundo o no.

**Ejercicio 2**Algoritmo que pida un número y diga si es positivo, negativo o 0.

**Ejercicio 3**Escribe un programa que lea un número e indique si es par o impar.

**Ejercicio 4**Crea un programa que pida al usuario dos números y muestre su división si el segundo no es cero, o un mensaje de aviso en caso contrario.

**Ejercicio 5**Escribe un programa que pida un nombre de usuario y una contraseña y si se ha introducido “pepe” y “asdasd” se indica “Has entrado al sistema”, si no, se da un error.

**Ejercicio 6**Programa que lea una cadena por teclado y compruebe si es una letra mayúscula.

**Ejercicio 7**Realiza un algoritmo que calcule la potencia, para ello pide por teclado la base y el exponente. Pueden ocurrir tres cosas:

• El exponente sea positivo, sólo tienes que imprimir la potencia.  
• El exponente sea 0, el resultado es 1.  
• El exponente sea negativo, el resultado es 1/potencia con el exponente positivo.

**Ejercicio 8**Algoritmo que pida dos números ‘nota’ y ‘edad’ y un carácter ‘sexo’ y muestre el mensaje ‘ACEPTADA’ si la nota es mayor o igual a cinco, la edad es mayor o igual a dieciocho y el sexo es ‘F’. En caso de que se cumpla lo mismo, pero el sexo sea ‘M’, debe imprimir ‘POSIBLE’. Si no se cumplen dichas condiciones se debe mostrar ‘NO ACEPTADA’.

**Ejercicio 9**Algoritmo que pida tres números y los muestre ordenados (de mayor a menor);

**Ejercicio 10**Algoritmo que pida los puntos centrales x1,y1, x2,y2 y los radios r1,r2 de dos circunferencias y las clasifique en uno de estos estados:

• exteriores  
• tangentes exteriores  
• secantes  
• tangentes interiores  
• interiores  
• concéntricas

Ayuda: http://mimosa.pntic.mec.es/clobo/geoweb/circun3.htm

**Ejercicio 11**Programa que lea 3 datos de entrada A, B y C. Estos corresponden a las dimensiones de los lados de un triángulo. El programa debe determinar qué tipo de triangulo es, teniendo en cuenta lo siguiente:

• Si se cumple Pitágoras entonces es triángulo rectángulo.  
• Si sólo dos lados del triángulo son iguales entonces es isósceles.  
• Si los 3 lados son iguales entonces es equilátero.  
• Si no se cumple ninguna de las condiciones anteriores, es escaleno.

**Ejercicio 12**Escribir un programa que lea un año e indicar si es bisiesto. Nota: un año es bisiesto si es un número divisible por 4, pero no si es divisible por 100, excepto que también sea divisible por 400.

**Ejercicio 13**Escribe un programa que pida una fecha (día, mes y año) y diga si es correcta.

**Ejercicio 14**La asociación de vinicultores tiene como política fijar un precio inicial al kilo de uva, la cual se clasifica en tipos A y B, y además en tamaños 1 y 2. Cuando se realiza la venta del producto, ésta es de un solo tipo y tamaño, se requiere determinar cuánto recibirá un productor por la uva que entrega en un embarque, considerando lo siguiente: si es de tipo A, se le cargan 20 céntimos al precio inicial cuando es de tamaño 1; y 30 céntimos si es de tamaño 2. Si es de tipo B, se rebajan 30 céntimos cuando es de tamaño 1, y 50 céntimos cuando es de tamaño 2. Realice un algoritmo para determinar la ganancia obtenida.

**Ejercicio 15**El director de una escuela está organizando un viaje de estudios, y requiere determinar cuánto debe cobrar a cada alumno y cuánto debe pagar a la compañía de viajes por el servicio. La forma de cobrar es la siguiente: si son 100 alumnos o más, el costo por cada alumno es de 65 euros; de 50 a 99 alumnos, el costo es de 70 euros, de 30 a 49, de 95 euros, y si son menos de 30, el costo de la renta del autobús es de 4000 euros, sin importar el número de alumnos. Realice un algoritmo que permita determinar el pago a la compañía de autobuses y lo que debe pagar cada alumno por el viaje.

**Ejercicio 16**La política de cobro de una compañía telefónica es: cuando se realiza una llamada, el cobro es por el tiempo que ésta dura, de tal forma que los primeros cinco minutos cuestan 1 euro, los siguientes tres, 80 céntimos, los siguientes dos minutos, 70 céntimos, y a partir del décimo minuto, 50 céntimos. Además, se carga un impuesto de 3 % cuando es domingo, y si es otro día, en turno de mañana, 15 %, y en turno de tarde, 10 %. Realice un algoritmo para determinar cuánto debe pagar por cada concepto una persona que realiza una llamada.

**Ejercicio 17**Realiza un programa que pida por teclado el resultado (dato entero) obtenido al lanzar un dado de seis caras y muestre por pantalla el número en letras (dato cadena) de la cara opuesta al resultado obtenido.

• Nota 1: En las caras opuestas de un dado de seis caras están los números: 1-6, 2-5 y 3-4.• Nota 2: Si el número del dado introducido es menor que 1 o mayor que 6, se mostrará el   
 mensaje: “ERROR: número incorrecto.”.

Ejemplo:

Introduzca número del dado: 5  
En la cara opuesta está el "dos".

**Ejercicio 18**Realiza un programa que pida el día de la semana (del 1 al 7) y escriba el día correspondiente. Si introducimos otro número nos da un error.

**Ejercicio 19**Escribe un programa que pida un número entero entre uno y doce e imprima el número de días que tiene el mes correspondiente.

**Ejercicio 20**Una compañía de transporte internacional tiene servicio en algunos países de América del Norte, América Central, América del Sur, Europa y Asia. El costo por el servicio de transporte se basa en el peso del paquete y la zona a la que va dirigido. Lo anterior se muestra en la tabla:

|  |  |  |
| --- | --- | --- |
| **Zona** | **Ubicación** | **Costo/gramo** |
| 1 | América del Norte | 24.00 euros |
| 2 | América Central | 20.00 euros |
| 3 | América del Sur | 21.00 euros |
| 4 | Europa | 10.00 euros |
| 5 | Asia | 18.00 euros |

Parte de su política implica que los paquetes con un peso superior a 5 kg no son transportados, esto por cuestiones de logística y de seguridad. Realice un algoritmo para determinar el cobro por la entrega de un paquete o, en su caso, el rechazo de la entrega.

## 3. Estructuras repetitivas

**Ejercicio 1**Crea una aplicación que pida un número y calcule su factorial (El factorial de un número es el producto de todos los enteros entre 1 y el propio número y se representa por el número seguido de un signo de exclamación. Por ejemplo: 5! = 1x2x3x4x5=120),

**Ejercicio 2**Crea una aplicación que permita adivinar un número. La aplicación genera un número aleatorio del 1 al 100. A continuación, va pidiendo números y va respondiendo si el número a adivinar es mayor o menor que el introducido, además de informarle de los intentos que le quedan (tiene 10 intentos para acertarlo). El programa termina cuando se acierta el número (además te dice en cuantos intentos lo has acertado), si se llega al límite de intentos te muestra el número que había generado.

**Ejercicio 3**Algoritmo que pida números hasta que se introduzca un cero. Debe imprimir la suma y la media de todos los números introducidos hasta ese momento.

**Ejercicio 4**Realizar un algoritmo que pida números (se pedirá por teclado la cantidad de números a introducir). El programa debe informar de cuantos números introducidos son mayores que 0, menores que 0 e iguales a 0.

**Ejercicio 5**Algoritmo que pida caracteres e imprima ‘VOCAL’ si son vocales y ‘NO VOCAL’ en caso contrario, el programa termina cuando se introduce un espacio.

**Ejercicio 6**Escribir un programa que imprima todos los números pares entre dos números que se le pidan al usuario.

**Ejercicio 7**Realizar un algoritmo que muestre la tabla de multiplicar de un número introducido por teclado.

**Ejercicio 8**Escribe un programa que pida el límite inferior y superior de un intervalo. Si el límite inferior es mayor que el superior lo tiene que volver a pedir. A continuación, se van introduciendo números hasta que introduzcamos el 0. Cuando termine el programa dará las siguientes informaciones:

• La suma de los números que están dentro del intervalo (intervalo abierto).  
• Cuántos números están fuera del intervalo.  
• Informa si hemos introducido algún número igual a los límites del intervalo (solo si ha   
 habido alguno o no).

**Ejercicio 9**Escribe un programa que, dados dos números, uno real (base) y un entero positivo (exponente), saque por pantalla el resultado de la potencia. No se puede utilizar el operador de potencia.

**Ejercicio 10**Algoritmo que muestre la tabla de multiplicar de los números 1,2,3,4 y 5.

**Ejercicio 11**Escribe un programa que diga si un número introducido por teclado es o no primo. Un número primo es aquel que sólo es divisible entre él mismo y la unidad. Nota: Es suficiente probar hasta la raíz cuadrada del número para ver si es divisible por algún otro número.

**Ejercicio 12**Realizar un algoritmo para determinar cuánto ahorrará una persona en un año si al final de cada mes deposita cantidades variables de dinero. Además, se quiere saber cuánto lleva ahorrado cada mes.

**Ejercicio 13**Una empresa tiene el registro de las horas que trabaja diariamente un empleado durante la semana (seis días) y requiere determinar el total de éstas, así como el sueldo que recibirá por las horas trabajadas.

**Ejercicio 14**Una persona se encuentra en el kilómetro 70 de una carretera, otra se encuentra en el km 150, los coches tienen sentido opuesto y tienen la misma velocidad. Realizar un programa para determinar en qué kilómetro de esa carretera se encontrarán.

**Ejercicio 15**Una persona adquirió un producto para pagar en 20 meses. El primer mes pagó 10 €, el segundo 20 €, el tercero 40 € y así sucesivamente (cada mes, el doble que el mes anterior). Realizar un algoritmo para determinar cuánto debe pagar mensualmente y el total de lo que pagó después de los 20 meses.

**Ejercicio 16**Una empresa les paga a sus empleados en base a las horas trabajadas en la semana y a un precio por hora dado. Realiza un algoritmo para determinar el sueldo semanal de N trabajadores y, además, calcula cuánto pagó la empresa por los N empleados.

**Ejercicio 17**Una empresa les paga a sus empleados con base en las horas trabajadas en la semana. Para esto, se registran los días que trabajó y las horas de cada día. Realice un algoritmo para determinar el sueldo semanal de N trabajadores y además calcule cuánto pagó la empresa por los N empleados.

**Ejercicio 18**Hacer un programa que muestre un cronometro, indicando las horas, minutos y segundos.

**Ejercicio 19**Realizar un ejemplo de menú, donde podemos escoger las distintas opciones hasta que seleccionamos la opción de “Salir”.

**Ejercicio 20**Mostrar en pantalla los N primeros números primos. Se pide por teclado la cantidad de números primos que queremos mostrar.

# UT2 Identificación de los elementos de un programa informático en Java

1.- Estructura y bloques fundamentales.   
2.- Variables.   
3.- Tipos de datos.   
4.- Literales.   
5.- Constantes.   
6.- Operadores y expresiones.   
7.- Conversiones de tipo.   
8.- Comentarios.

## Estructura y bloques fundamentales de un programa Java

El código fuente de un programa en Java debe estar escrito en un **fichero de texto con extensión ".java"**. Para crear o editar el fichero debe utilizarse un editor de texto sin formato como el bloc de notas de Windows, el editor de texto de Linux, o alguno de los programas y entornos de desarrollo específicos para la creación de aplicaciones.

public class HolaMundo {

public static void main(String args[]) {

System.out.println("Hola Mundo!");

}

}

La primera línea de código indica el nombre de la clase Java que estamos desarrollando. Todos los programas Java están formados por una o más clases. Es importante tener en cuenta que el nombre de la clase debe corresponder exactamente con el nombre del fichero de texto que contiene el código fuente. En el caso de este ejemplo, el código debe almacenarse en un fichero de texto denominado "HolaMundo.java".

public class HolaMundo

Las llaves { } indican el inicio y fin de cada bloque. Siempre deben ir en pareja, es decir, que por cada llave de apertura debe existir siempre una llave de cierre.

La siguiente línea realiza la declaración del método llamado "main". Cada clase Java que forma una aplicación contendrá uno o varios métodos, que son bloques de código que permiten tenerlo más organizado. Si una determinada clase Java contiene un método con el nombre "main", la ejecución de dicha clase comenzará por el código contenido en ese método.

public static void main(String args[])

La última línea de código del ejemplo, contiene las sentencias que se van a ejecutar. En este caso tan solo hay una sentencia, pero podría haber todas las que se quisiera. Aquí en concreto se utiliza la llamada a *System.out.println()* que permite mostrar en pantalla una serie de caracteres. El texto mostrado es el indicado entre comillas: "Hola Mundo!".

System.out.println("Hola Mundo!");

## Variables

Las ***variables*** identifican datos mediante un nombre simbólico, haciendo referencia a un espacio de memoria principal en los que se sitúan los datos, para que puedan ser utilizados por el procesador, y así poder hacer cualquier tipo de operación con ellos.

Los **nombres utilizados para identificar a las variables** deben cumplir una serie de **condiciones**:

* No pueden empezar por un dígito numérico.
* No pueden utilizarse espacios, y los únicos caracteres especiales válidos son el guión bajo (\_) y el símbolo del dólar ($).
* Son sensibles a las mayúsculas y minúsculas, es decir, las variables "suma" y "Suma" se consideran variables distintas.

No pueden utilizarse como nombres de variables las palabras reservadas de Java, que son las siguientes:



Es costumbre empezar los nombres de las variables por una letra minúscula.

Cuando el nombre de una variable está formado por más de una palabra, se suele utilizar una letra mayúscula para distinguir el comienzo de las palabras. Por ejemplo: sumaTotal.

Es recomendable utilizar nombres que hagan referencia al contenido que va a almacenar para facilitar la comprensión del código. Es mucho más claro utilizar el nombre "suma" que "s".

**Ejemplos** de nombres de variables **válidos**: indice, ventas, compras, saldoGeneral, importetotal, contador\_lineas, $valor, num2.

**Ejemplos** de nombres de variables **no válidos**: 3valores, suma&total, super, edad media.

Antes de poder utilizar una variable, esta debe ser declarada en el programa. La declaración de variables se debe realizar siguiendo el siguiente formato de sentencia:

tipoDato nombreVariable;

Donde tipoDato es uno de los tipos de datos básicos o el nombre de una clase (byte, short, int, long, float, double, boolean, char, String, etc), y nombreVariable es el nombre que se desea asignar a la variable siguiendo las normas anteriores.

Es posible declarar más de una variable de un mismo tipo en la misma línea separando los nombres con comas:

tipoDato nombreVariable1, nombreVariable2, nombreVariable3;

***Ejemplos*** de declaraciones de variables:

int num1, num2, suma;

char letraNIF;

String saludoInicial;

boolean mayorEdad;

Es posible asignar un valor inicial a las variables en el momento de declararlas. Para inicializar variables se debe seguir el siguiente formato de sentencia:

tipoDato nombreVariable = valorInicial;

Donde valorInicial puede ser un valor literal, otra variable declarada anteriormente o una expresión combinando valores literales y variables con operadores. El valorInicial debe ser del mismo tipo de dato que la variable que se está declarando.

Ejemplos de declaraciones de variables con inicialización:

int num1 = 34;

int doble = num1 \* 2;

String saludo = "Hola";

char letraA = 'A', letraB = 'B';

En el código del programa es posible asignar valores a las variables que previamente han sido declaradas. Al hacerlo, el valor que guardaba la variable anteriormente se perderá. Se debe utilizar el siguiente formato:

nombreVariable = valor;

Donde valor puede ser de nuevo un valor literal, una variable declarada anteriormente (puede ser la misma variable) o una expresión combinando valores literales y variables con operadores. El valor debe ser del mismo tipo de dato que la variable a la que se está asignando el nuevo valor.

package ejemplos;

public class AsignacionVariables {

public static void main(String[] args) {

int a=5, b=0, c;

b = a \* 3; // Se cambia el valor de b a 15

c = a; // Se guarda en c el valor de a que es 5

a = a + 6; // Se suma 6 al valor que tenía a. Ahora a vale 11

b = a - c; // b guarda 11 - 5 , luego vale 6

System.out.println("La variable a contiene: " + a);

System.out.println("La variable b contiene: " + b);

System.out.println("La variable c contiene: " + c);

}

}

Se muestra lo siguiente:

La variable a contiene: 11

La variable b contiene: 6

La variable c contiene: 5

Las variables pueden ser utilizadas dentro del bloque de código en el que han sido declaradas, es decir, dentro de las llaves "{" y "}" que marcan el inicio y el fin de un bloque de código. Se denomina **ámbito** de la variable al bloque de código en el que se declara la variable.

package ejemplos;

public class AmbitoVariables {

static int variableGlobal;

public static void main(String[] args) {

int variableDelMain = 10;

/\*Aquí se pueden usar variableGlobal y

variableDelMain. No se puede usar

variableDeOtroMetodo \*/

System.out.println("variableGlobal " + variableGlobal);

System.out.println("variableDelMain " + variableDelMain);

otroMetodo();

}

static void otroMetodo() {

int variableDeOtroMetodo=90;

/\* Aquí se pueden usar variableGlobal y

variableDeOtroMetodo. No se puede usar

variableDelMain \*/

System.out.println("variableGlobal " + variableGlobal);

//System.out.println("variableDelMain " + variableDelMain);

System.out.println("variableDeOtroMetodo " + variableDeOtroMetodo);

}

}

## Tipos de datos básicos

El lenguaje de programación Java permite la utilización de los siguientes tipos de datos básicos:

**Números enteros**: Representan a los números enteros (sin parte decimal) con signo (pueden ser positivos o negativos). Se dispone de varios tipos de datos, ocupando cada uno de ellos un espacio distinto en memoria. Cuanta más capacidad de almacenamiento, más grande es el rango de valores permitidos, aunque ocupará más espacio de memoria principal. Se dispone de los siguientes tipos:

* **byte**: Ocupan 8 bits (1 byte), permitiendo almacenar valores entre -128 y 127.
* (2⁸ = 256; necesitas 1 bit para el signo y uno de los valores será 0; 2⁷ = 128;)
* **short**: Ocupan 16 bits (2 bytes), permitiendo almacenar valores entre -32.768 y 32.767.
* **int**: Ocupan 32 bits (4 bytes), permitiendo almacenar valores entre -2.147.483.648 y 2.147.483.647. Es el tipo de datos por defecto para los valores numéricos enteros. Este tipo de datos es lo suficientemente grande para almacenar los valores numéricos que vayan a usar tus programas. solo se suelen usar los tipos anteriores si se pueden producir problemas con el espacio de memoria.
* **long**: Ocupan 64 bits (8 bytes), permitiendo almacenar valores entre

-9.223.372.036.854.775.808 y 9.223.372.036.854.775.807.

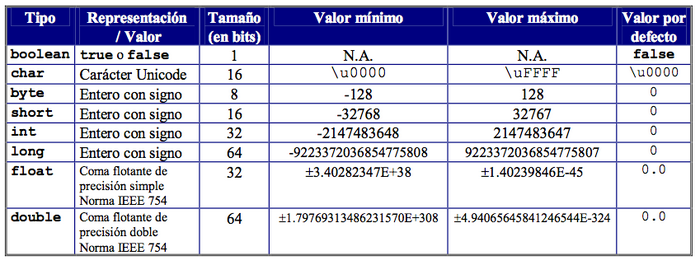
**Números reales**: Representan a los números reales con parte decimal y signo positivo o negativo. Hay dos tipos de datos numéricos reales que permiten obtener mayor o menor precisión. Utilizan un método para almacenar los datos que puede ocasionar que el valor original varíe levemente del valor almacenado realmente. Cuanta más precisión se utilice, habrá menor variación.

* **float:** Ocupan 32 bits (4 bytes). Se le denomina de simple precisión. Almacenan valores desde -3.40282347E+38 a + 3.40282347E+38

(E+x significa: 10 elevado a x)

* **double**: Ocupan 64 bits (8 bytes). Se le denomina de doble precisión. Es el tipo de datos por defecto para los valores numéricos reales. Almacenan valores desde:

- 1.79769313486231570E+308 a +1.79769313486231570E+308



**Valores lógicos:** Representan dos únicos posibles valores: verdadero y falso.

* **boolean:** Ocupan 1 bit, pudiendo almacenar los valores true (verdadero) y false (falso).

**Caracteres:** Representan las letras, dígitos numéricos y símbolos contenidos en la tabla de caracteres Unicode.

* **char:** Ocupan 16 bits (2 bytes). Permite representar un único carácter, encerrado entre comillas simples, por ejemplo la letra 'A'.

(https://unicode-table.com/en/)

* **String:** Realmente **no es un tipo de dato básico de Java**, pero por su interés se incluye aquí. Permite representar un conjunto de caracteres, encerrado entre comillas dobles, por ejemplo: "Saludos para todos".

(String empieza con mayúsculas porque es el nombre de una clase, una clase es el nombre de una plantilla)

## Valores literales – *Leer en casa*

Un valor literal es la representación de un valor fijo en el código fuente de un programa.

Los valores correspondientes a los tipos de datos numéricos enteros (byte, short, int y long) se pueden expresar usando el sistema numérico decimal, octal o hexadecimal. Los números en sistema decimal se expresan de la manera habitual, simplemente escribiendo su valor con los dígitos numéricos, por ejemplo, 284. No se pueden emplear separadores de millares, por lo que para indicar el valor 1.000.000 (un millón) se debe escribir como 1000000. Para representar valores negativos se añade el carácter "-" (guion) delante del número, como es habitual es la escritura normal, por ej -376.

Para representar un valor numérico entero en sistema octal, debe ir precedido del carácter 0 (cero), por ejemplo, el valor 284 se representa en octal como 0434. Asimismo, para representar un valor en el sistema hexadecimal, se debe emplear el prefijo 0x, por lo que el valor 284 se representa en hexadecimal como 0x11C.

Por defecto, los valores literales numéricos enteros se almacenan en memoria con el formato del tipo de dato "int". Si se desea almacenar como "long", con el fin de poder obtener resultados con valores muy altos en los cálculos, se debe emplear el sufijo L en mayúscula o minúscula (sería recomendable utilizar la L mayúscula por el parecido de la letra minúscula con el valor 1). Por ejemplo, el valor 284L correspondería al valor entero 284 utilizando 64 bits (tipo long) para almacenarlo en memoria.

Para representar valores literales de los tipos de datos numéricos reales (float y double) se puede emplear el sistema decimal o la notación científica. En el sistema decimal se expresan los números con parte decimal de la forma usual, utilizando el punto como separador de la parte entera y decimal. En este caso tampoco se puede emplear los separadores de millares. Así, por ejemplo, el valor 21.843,83 se debe expresar como 21843.83 en el código fuente.

Los números reales expresados en notación científica deben emplear la letra "E" o "e" para separar la parte correspondiente al exponente. El valor 7,433·10^6 se debe expresar como 7.433e6 en el código fuente. Si el exponente es negativo se escribe el guion detrás de la letra E, por ejemplo, 7,433·10^-6 se expresa como 7.433e-6, y si el valor es negativo se indica el guion al principio, por ejemplo, -7,433·10^6 se expresa como -7.433e6.

Por defecto, los valores literales numéricos reales se almacenan en memoria con el formato del tipo de dato "double". Si se desea almacenar como "float", con el fin de emplear menos espacio de memoria y necesitar menos precisión en los resultados de los cálculos, se debe emplear el **sufijo F en mayúscula o minúscula**. Por ejemplo, el valor 21843.83F correspondería al valor entero 21843.83 utilizando 32 bits (tipo float) para almacenarlo en memoria, en vez de 64 bits.

Los valores literales de los tipos char (carácter) y String (cadena de caracteres), pueden contener cualquier carácter Unicode. Los valores de tipo char se debe expresar encerrados entre comillas simples (en los teclados españoles habituales se encuentra junto a la tecla del cero), por ejemplo, la letra A se debe expresar como 'A'. Por otro lado, las cadenas de caracteres (tipo String) se expresan entre comillas dobles (en la tecla del 2), por ejemplo, el texto Saludos a todos, se debe escribir como "Saludos a todos".

**En caso de que se necesite escribir un carácter de la tabla de caracteres Unicode que no se encuentre en el teclado, se puede hacer indicando el código hexadecimal correspondiente a dicho carácter precedido del modificador \u (utilizando la barra invertida situada en la tecla junto al 1)**. En todo caso se debe encerrar entre comillas simples o dobles según se vaya a tratar como carácter o dentro de una cadena de caracteres. Por ejemplo, para escribir la letra griega beta (ß) se puede emplear '\u00DF', o para escribir la palabra España es posible utilizar "Espa\u00F1a".

El lenguaje de programación Java también soporta un pequeño conjunto de caracteres especiales que se pueden utilizar en los valores literales char y String:

\b (retroceso), \t (tabulador), \n (nueva línea), \f (salto de página), \r (retorno de carro), \" (comilla doble), \' (comilla simple), \\ (barra invertida).

Por ejemplo, la cadena de caracteres "La palabra "hola" es un saludo" se tiene que escribir en el código fuente como: "La palabra \"hola\" es un saludo", para que no confunda las comillas dobles de inicio y fin de la cadena de caracteres.

El carácter especial '\n' permite introducir un salto de línea dentro de una cadena de caracteres. Por ejemplo, "Primera línea\nSegunda línea" mostraría ese texto separado en dos líneas.

Los valores literales para el tipo de dato boolean sólo pueden ser true o false, que corresponden a los valores Verdadero y Falso. Hay que observar que se deben indicar sin comillas de ningún tipo, ya que no son cadenas de caracteres.

En un programa, es posible mostrar cualquiera de estos valores literales a través de la salida estándar (terminal, ventana de salida, etc) utilizando la siguiente sentencia:

System.out.println(valorLiteral);

Donde valorLiteral debe ser el valor que se desea mostrar manteniendo las normas comentadas.

Ejemplos:

//Mostrar un valor numérico entero

System.out.print("Número entero: ");

System.out.println(284);

//Mostrar un valor numérico real

System.out.print("Número real: ");

System.out.println(21843.83);

//Mostrar un carácter

System.out.print("Carácter: ");

System.out.println('A');

//Mostrar una cadena de caracteres

System.out.print("Cadena de caracteres: ");

System.out.println("Saludos a todos");

//Mostrar un valor lógico

System.out.print("Valor lógico: ");

System.out.println(true);

## Constantes

Son similares a las variables en cuanto que son datos a los que se hace referencia mediante un nombre y a los que se les asigna un valor. Pero a diferencia de las variables, a las constantes no se les puede modificar el valor asignado.

El formato de declaración de las constantes es prácticamente igual que el utilizado para las variables. La única diferencia es que se debe indicar el modificador final delante del tipo de dato que almacenará:

**final** tipoDato NOMBRECONSTANTE = valor;

Al igual que en la declaración de variables, el valor que se asigna en la declaración puede ser un valor literal, una variable o una expresión.

Ejemplo:

**final double** PI = 3.1415926536;

Por convenio, el nombre de las constantes se escribe en mayúsculas.

El beneficio de usar constantes es evitar la repetición de escribir un mismo valor en el programa y facilitar su modificación.

El empleo de las constantes a lo largo del programa es igual que el utilizado con las variables. Se indica su nombre dentro de cualquier expresión o como parámetro para métodos como println().

System.out.println("Perímetro = " + 2\*PI\*radio);

## Operadores

El lenguaje de programación Java incorpora una serie de operadores que permiten realizar cálculos y escribir expresiones que realicen una serie de operaciones sobre los datos.

### Operadores Aritméticos:

+ (suma)

- (resta)

\* (multiplicación)

/ (división entera o con decimales según operandos)

% (resto de la división)

Todos estos operadores aritméticos deben utilizarse con dos operandos, situados delante y detrás de los operadores, pudiéndose encadenar las operaciones. Se pueden incluir espacios para aclarar más el código. Los datos usados como operandos deben ser de alguno de los tipos de datos numéricos (byte, short, int, long, float o double).

***Ejemplos***:

4 + 3

8 - 5 + 2

6 \* 2 / 3

8.5 - 3 + 4.3

El resultado de la división tendrá decimales o no según el tipo de operandos que se utilice. Si los dos son enteros, el resultado no tendrá decimales, pero si al menos uno de los operandos es de tipo numérico real (float o double) el resultado será de ese tipo. Ejemplos:

8 / 2 resulta 4

7 / 2 resulta 3

7.0 / 2 resulta 3.5

7 / 2.0 resulta 3.5

7.4 / 2 resulta 3.7

8 / 2.5 resulta 3.2

8.5 / 2.5 resulta 3.4

Es posible modificar el tipo de dato de cualquier operando indicando delante el nuevo tipo de dato entre paréntesis. Así se hace una conversión de tipo (casting):

(double)7 / 2 resulta 3.5

7 / (float)2 resulta 3.5

El operador resto se debe utilizar con tipos de datos numéricos enteros *(a mi me ha funcionado también con double)*. El resultado será el resto de la división entre los dos operandos. Ejemplos:

7 % 2 resulta 1

8 % 3 resulta 2

El resultado de cualquier operación aritmética será del tipo de dato más grande que se utilice en los operandos. Por ejemplo, si se hace una operación entre dos números enteros (int) el resultado será del mismo tipo, pero si se hace entre un int y un long es resultado es de tipo long.

Ejemplos:

2147483647 \* 2 resulta un dato incorrecto, en concreto -2, porque se están multiplicando dos int y el resultado sobrepasa el límite de los enteros.

2147483647L \* 2 resulta 4294967294 porque el primer operando es de tipo long (se ha indicado L al final).

Los caracteres pueden ser utilizados para realizar cálculos aritméticos. En caso de que aparezca algún carácter en una expresión aritmética, se toma el valor numérico que le corresponde a cada carácter en la tabla de codificación Unicode.

Ejemplos:

'A' + 1 resulta 66, ya que el carácter 'A' tiene el código 65.

(char) ('A' + 1) resulta 'B' ya que se ha hecho una conversión de tipos del resultado.

### Operadores Relacionales

Los operadores relacionales permiten comparar dos valores numéricos.

> (mayor que)

>= (mayor o igual que)

< (menor que)

<= (menor o igual que)

== (igual que)

!= (distinto de)

Cada uno de estos operadores relacionales debe emplearse con dos valores numéricos a ambos lados, pudiendo ser dos valores literales o resultados de expresiones aritméticas.

Ejemplos:

4 > 3 resulta true.

7 <= 2 resulta false.

5 + 2 == 4 + 3 resulta true.

4 \* 3 != 12 resulta false.

### Operadores Lógicos

Los operadores lógicos permiten unir valores o expresiones lógicas, obteniendo como resultado si es verdadera o falsa la expresión combinada. Son los siguientes:

&& (Y lógico - conjunción)

|| (O lógico - disyunción)

! (NO lógico)

Los operadores && y || deben utilizarse con dos valores o expresiones lógicas a ambos lados, mientras que el operador de negación ! solo se aplica al valor o expresión lógica que tenga a su derecha. El resultado que se obtiene utilizando estos operadores se obtiene de la siguiente tabla de verdad:

El **operador Y** (&&) resulta true solo si ambos operandos son true:

true && true resulta true

true && false resulta false

false && true resulta false

false && false resulta false

El **operador O** (||) resulta true si al menos uno de los operandos son true:

true || true resulta true

true || false resulta true

false || true resulta true

false || false resulta false

El **operador NO** (!) resulta true si el operando es false:

!true resulta false.

!false resulta true.

Ejemplos:

4 > 3 && 5 <= 5 resulta true,

porque las dos expresiones son true.

4 > 3 && 5 != 5 resulta false,

porque al menos una expresión es false.

4 > 3 || 5 != 5 resulta true,

porque al menos una expresión es true.

4 > 3 && !(5 != 5) resulta true,

porque las dos expresiones son true.

### Operadores de Asignación

Permiten asignar valores a variables. El operador de asignación elemental es el igual (=) que asigna un valor o el resultado de una expresión a una variable, siguiendo el siguiente formato:

nombreVariable = valorAsignado;

El tipo de dato del valor asignado debe ser del mismo tipo que el de la variable a la que se pretende asignar. En caso de que no sean del mismo tipo, hay que utilizar algún tipo de conversión como el casting, utilizando entre paréntesis el nombre del tipo de dato al que se quiere convertir.

int numEntero;

numEntero = 364; //Correcto

numEntero = 264.38; //Incorrecto

numEntero = (int)264.83; //Correcto, se asigna el valor 264 (lo trunca)

También se puede utilizar el operador de asignación para guardar en una variable el resultado de una expresión que dé como resultado un valor del mismo tipo que la variable.

nombreVariable = Expresión;

int numEntero;

numEntero = 364 \* 2 + 266;

Además del operador igual (=), se pueden emplear otros operadores de asignación, que a la vez que asignan un valor realizan un cálculo:

**+=** (le suma a la variable un valor y guarda el resultado en la misma variable).

**-=** (le resta a la variable un valor y guarda el resultado en la misma variable).

**\*=** (multiplica la variable por un valor y guarda el resultado en la misma variable).

**/=** (divide la variable por un valor y guarda el resultado en la misma variable).

**%=** (obtiene el resto de dividir la variable por un valor y guarda el resultado en la misma variable).

calculo = 5;

calculo += 6; //Incrementa en 6 el valor de la variable: calculo = calculo +6

System.out.println(calculo); //Muestra 11

//Duplica el valor de cálculo: calculo = calculo\*2

calculo \*= 2;

System.out.println(calculo); //Muestra 22

Por tanto, una sentencia como a+=3 es lo mismo que a=a+3.

### Operadores Incrementales

Son los operadores que nos permiten incrementar las variables en una unidad

++ (incrementa en 1 el valor de una variable).

-- (decrementa en 1 el valor de una variable).

Ejemplo:

num1 = 3;

num2 = 7;

num1++; //Suma 1 a num1

System.out.println(num1); //Muestra 4

num2--; //resta 1 a num2

System.out.println(num2); //Muestra 6

Por tanto, una sentencia como x++ es los mismo que x=x+1, y x-- es lo mismo que x=x-1.

Hay dos formas de utilizar el incremento y el decremento, x++ ó ++x. La diferencia estriba en el modo en el que se comporta la asignación.

Ejemplo:

int x=5, y=5, z;

z=x++; /\* z vale 5, x vale 6 porque primero se asigna

el valor de x a z después se incrementa x \*/

z=++y; /\* z vale 6, y vale 6 porque primero incrementa

la variable y, después se asigna el valor a z \*/

### Operador Condicional. - (ha dicho que nos cuesta entenderlo)

*(No es lo mismo un operador condicional que una sentencia)*

Existe un operador condicional, que permite asignar a una variable un valor u otro dependiendo de una expresión condicional. El formato es el siguiente:

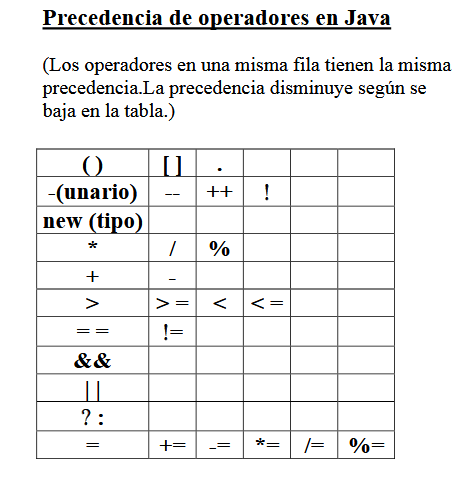
variable = condición ? valor1 : valor2;

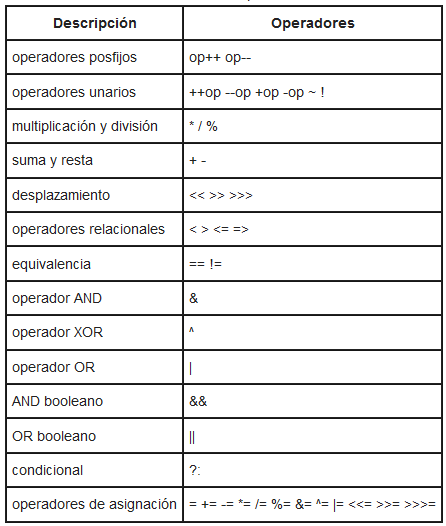
La condición que se indica debe ser una variable booleana o una expresión condicional que resulte un valor de tipo booleano (true o false).

Ejemplo:

mensaje = (num1 >= 0) ? "Positivo" : "Negativo";

Si se mostrara en pantalla posteriormente el valor de la variable mensaje, aparecerá "Positivo" si el valor de la variable num1 es mayor o igual que cero, y "Negativo" en caso contrario.





## Conversiones **d**e tipo

Java es un lenguaje fuertemente tipificado, lo que significa que es bastante estricto al momento de asignar valores a una variable (*esto ya no suele ser así, pero en versiones previas lo era*). El compilador sólo permite asignar un valor del tipo declarado en la variable; no obstante, en ciertas circunstancias es posible realizar conversiones que permiten almacenar en una variable un tipo diferente al declarado. En Java es posible realizar conversiones en todos los tipos básicos, con excepción de boolean, que es incompatible con el resto de los tipos.

Las conversiones de tipo pueden realizarse de dos maneras: implícitamente y explícitamente.

### Conversión implícita

Las conversiones implícitas se realizan de manera automática, es decir, el valor o expresión que se va a asignar a una variable es convertido automáticamente por el compilador, antes de almacenarlo en la variable.

Para que una conversión pueda realizarse de manera automática (implícitamente), el tipo de la variable destino debe ser de tamaño igual o superior al tipo de origen, si bien esta regla tiene dos excepciones:

a) Cuando la variable destino es entera y el origen es decimal (float o double), la conversión no podrá ser automática.

b) Cuando la variable destino es "char" y el origen es numérico; independientemente del tipo específico, la conversión no podrá ser automática.

Ejemplos de conversiones implícitas:

// Declaraciones.

int j = 5, i;

short s = 10;

char c = 'ñ';

float f;

// Conversiones implícitas.

i = c; // Conversión implícita de char a int.

f = j; // Conversión implícita de int a float.

i = s; // Conversión implícita de short a int.

Los siguientes ejemplos de conversión implícita provocarían un error:

// Declaraciones.

int i;

long l = 20;

float ft = 2.4f;

char c;

byte b = 4;

// Conversiones implícitas.

i = l; // Error, el tipo destino es menor al tipo origen.

c = b; // Cuando la variable destino es "char" y el origen es numérico; independientemente del tipo específico, la conversión no podrá ser automática.

i = ft; // Cuando la variable destino es entera y el origen es decimal (float o double), la conversión no podrá ser automática.

### Conversión explícita

Cuando no se cumplan las condiciones para una conversión implícita, ésta podrá realizarse de manera explícita utilizando la expresión:

variable\_destino = (tipo\_destino) dato\_origen;

Con esta expresión se obliga al compilador que convierta "dato\_origen" a "tipo\_destino" para que pueda ser almacenado en "variable\_destino". A esta operación se le conoce como "casting" o "conversión”.

Ejemplos de conversiones explícitas:

// Declaraciones.

char c;

byte b;

int i = 400;

double d = 34.6;

// Conversiones explícitas.

c = (char)d; // Se elimina la parte decimal (trunca), no se redondea.

b = (byte)i; // Se provoca una pérdida de datos, pero la conversión es posible.

public class Conversiones {

public static void main (String [] args) {

int a = 2;

double b = 3.0;

float c = (float) (20000\*a/b + 5);

System.out.println("Valor en formato float: " + c);

System.out.println("Valor en formato double: " +

(double) c);

System.out.println("Valor en formato byte: " +

(byte) c);

System.out.println("Valor en formato short: " +

(short) c);

System.out.println("Valor en formato int: " + (int) c);

System.out.println("Valor en formato long: " + (long) c);

}

}

## Comentarios

Cuando se escribe código en general es útil realizar comentarios explicativos. Los comentarios no tienen efecto como instrucciones para el ordenador, simplemente sirven para que cuando una persona lea el código pueda comprender mejor lo que lee. En Java existen dos formas de poner comentarios.

La primera es cuando la línea de comentario solo ocupa una línea de código. En este caso se pone dos barras inclinadas (//) antes del texto.

// Comentario de una línea

En el caso de comentarios de más de una línea se pone /\* para empezar y \*/ para finalizar.

El código nos quedará de la siguiente forma:

/\* Comentario

de varias

líneas \*/

# UT2 Literales en Java

## LITERAL JAVA

Un literal Java es un valor de tipo entero, real, lógico, carácter, cadena de caracteres o un valor nulo (null) que puede aparecer dentro de un programa.

Por ejemplo: 150, 12.4, “ANA”, null, ‘t’.

### LITERAL JAVA DE TIPO ENTERO

Puede expresarse en decimal (base 10), octal (base 8) y hexadecimal (base 16).

El signo + al principio es opcional y el signo – será obligatorio si el número es negativo.

El tipo de un literal entero es **int** a no ser que su valor absoluto sea mayor que el de un int o se especifique el sufijo l o L en cuyo caso será de tipo **long**.

#### Literal Java de tipo entero en Base decimal

Está formado por 1 o más dígitos del 0 al 9.

El primer dígito debe ser distinto de cero.

Por ejemplo:

1234 literal java entero de tipo int

1234L literal java entero de tipo long

123400000000 literal java entero de tipo long

#### Literal Java de tipo entero en Base octal

Está formado por 1 o más dígitos del 0 al 7.

El primer dígito debe ser cero.

Por ejemplo:

01234

025

#### Literal Java de tipo entero en Base hexadecimal

Está formado por 1 o más dígitos del 0 al 9 y letras de la A a la F (mayúsculas o minúsculas).

Debe comenzar por 0x ó 0X.

Por ejemplo:

0x1A2

0x430

0xf4

### LITERAL JAVA DE TIPO REAL

Son números en base 10, que deben llevar un parte entera, un punto decimal y una parte fraccionaria.  Si la parte entera es cero, puede omitirse.

El signo + al principio es opcional y el signo – será obligatorio si el número es negativo.

Por ejemplo:

12.2303

-3.44

+10.33

0.456

.96

También pueden representarse utilizando la notación científica. En este caso se utiliza una E (mayúscula o minúscula) seguida del exponente (positivo o negativo). El exponente está formado por dígitos del 0 al 9.

Por ejemplo, el número en notación científica 14\*10-3 se escribe: 14E-3

Más ejemplos de literal Java de tipo real:

2.15E2 -> 2.15 \* 102

.0007E4 -> 0.0007 \* 104

-50.445e-10 -> -50.445 \* 10-10

El tipo de estos literales es siempre **double**, a no ser que se añada el sufijo F ó f para indicar que es float.

Por ejemplo:

2.15 literal real de tipo double

2.15F literal real de tipo float

También se pueden utilizar los sufijos d ó D para indicar que el literal es de tipo double:

12.002d literal real de tipo double

### LITERAL JAVA DE TIPO CARÁCTER

Contiene un solo carácter encerrado entre comillas simples.

Es de tipo **char**.

Las secuencias de escape se consideran literales de tipo carácter.

Por ejemplo:

'a'

'4'

'\n'

'\u0061'

### LITERAL JAVA DE CADENAS DE CARACTERES

Está formado por 0 ó más caracteres encerrados entre comillas dobles.

Pueden incluir secuencias de escape.

Por ejemplo:

“Esto es una cadena de caracteres”

“Pulsa \”C\” para continuar”

“” -> cadena vacía

“T” -> cadena de un solo carácter, es diferente a ‘t’ que es un carácter

Las cadenas de caracteres en Java son objetos de tipo **String**.

*Nota: Sobre atajos de teclado en Eclipse:  
Ctrl+May+L nos visualiza en Eclipse todos los atajos de teclado*

# UT2 Ampliación Operadores

## Operadores lógicos

|  |  |  |  |
| --- | --- | --- | --- |
| **Operador** | **Descripción** | **Uso** | **Retorna cierto si** |
| && | and | op1 && op2 | op1 y op2 son true.  Solo evalúa op2 si op1 es true |
| || | or | op1 || op2 | op1 o op2 es true.  Solo evalúa op2 si op1 es false |
| ! | not | !op | op es false |

## Operadores lógicos a nivel de bits

|  |  |  |
| --- | --- | --- |
| **Operador** | **Descripción** | **Uso** |
| & | and | op1 & op2 |
| | | or | op1 | op2 |
| ^ | or exclusivo | op1 ^ op2 |
| ~ | complemento | ~op1 |

El operador **AND** **bit por bit (&)** realiza un and lógico entre cada par de bits paralelos que forman los operandos. Si ambos bits valen 1 el bit resultante vale 1, de lo contrario vale 0. Por ejemplo:

|  |
| --- |
| int bits\_orig = 6; --------// binario 00000000 00000000 00000000 00000**110** |
| int bits\_bandera = 5; -----// binario 00000000 00000000 00000000 00000**101** |
| int respuesta = (bits\_orig & bits\_bandera);--------- --// 000000 00000**100** |

El operador **OR bit por bit (|)** realiza un or lógico entre cada par de bits paralelos que forman los operandos. Si alguno de los bits vale 1 el bit resultante vale 1, de lo contrario vale 0. Por ejemplo:

|  |
| --- |
| int bits\_orig = 6; --------// binario 00000000 00000000 00000000 00000**110** |
| int bits\_bandera = 5; -----// binario 00000000 00000000 00000000 00000**101** |
| int respuesta = (bits\_orig | bits\_bandera);--------- --// 000000 00000**111** |

El operador **XOR bit por bit (^)** realiza un or exclusivo entre cada par de bits paralelos que forman los operandos. Si ambos bits tienen valores opuestos el bit resultante vale 1, de lo contrario vale 0. Por ejemplo:

|  |
| --- |
| int bits\_orig = 6; --------// binario 00000000 00000000 00000000 00000**110** |
| int bits\_bandera = 5; -----// binario 00000000 00000000 00000000 00000**101** |
| int respuesta = (bits\_orig ^ bits\_bandera);--------- --// 000000 00000**011** |

El operador **NOT bit por bit (~)** invierte los bits del operando. Por ejemplo:

|  |
| --- |
| byte valor = 6;-------------------// binario **00000110** |
| byte respuesta = (~ valor);---- --// binario **11111001** |

## Operadores de desplazamiento de bits

Existen muchas operaciones que requieren desplazar los valores de los bits que componen un número a la izquierda o a la derecha. Los operadores de desplazamiento de bits suelen utilizarse para llevar a cabo operaciones muy rápidas de multiplicación y de división de enteros. Un desplazamiento a la izquierda equivale a una multiplicación por 2 y un desplazamiento a la derecha a una división por 2.

Un desplazamiento no es una rotación. A medida que se desplazan los bits hacia un extremo se van rellenando con ceros por el otro extremo. Los bits que salen se pierden.

|  |  |  |
| --- | --- | --- |
| **Operador** | **Uso** | **Operación** |
| << | op1 << op2 | Desplaza a la izquierda los bits del primer operando op1 tantas veces como indica el segundo operando op2( por la derecha siempre entra un cero). |
| >> | op1 >> op2 | Desplaza a la derecha los bits del primer operando op1 tantas veces como indica el segundo operando op2( por la izquierda entra siempre el bit más significativo anterior). |
| >>> | op1 >>> op2 | Desplaza a la derecha los bits del primer operando op1 tantas veces como indica el segundo operando op2(sin signo- por la izquierda entra siempre un cero) |

A diferencia de C y C++, Java siempre conserva el bit de signo (el bit izquierdo) después de hacer un desplazamiento. Este tipo de desplazamiento se conoce como desplazamiento aritmético o desplazamiento de extensión de signo. Las siguientes instrucciones desplazan un valor, primero dos bits a la izquierda y luego dos bits a la derecha, con los operadores de desplazamiento de bits de Java:

|  |
| --- |
| int original = -3; ---------------- //- binario 10000000 00000000 00000000 00000011 |
| int izquierda = original << 2; ---- // izquierda vale -12 ---binario 10000000 00000000 00000000 00001100 |
| int derecha = izquierda >> 2;-----  // derecha vale -3 --- --binario 10000000 00000000 00000000 00000011 |
| int x = 7; ----------------------- // x vale 7--------------binario 00000000 00000000 00000000 00000111 |
| x = x << 1;----------------------- // x vale 14------------ binario 00000000 00000000 00000000 00001110 |

El **operador >>>** realiza un desplazamiento a la derecha similar al operador >>. La diferencia es que el operador >>> no conserva el bit de signo, sino que coloca el bit de signo en cero, a diferencia del operador >>, que lo pone en uno cuando es negativo, de forma que permanezca negativo. Normalmente se utiliza el operador >>> cuando el operador no representa una cantidad con signo, sino que es una máscara de bits de algún tipo en el que no importa el bit de signo. El siguiente ejemplo muestra el uso del operador >>> para desplazar el valor 256 dos bits a la derecha, dando como resultado 64:

|  |
| --- |
| int mascara = 256;------------------ // mascara 256-------------- binario 00000000 00000000 00000001 00000000 |
| int respuesta = mascara >>> 2;------ // respuesta 64------------- binario 00000000 00000000 00000000 01000000 |

Nota:

Java convierte a tipo int los tipo short y byte antes de realizar el desplazamiento de bits. Esto significa que el tipo byte tendrá 32 bits en lugar de 8. Después de ejecutar el desplazamiento trunca el dato para convertirlo al tipo original.

# UT2 Java Scanner para la lectura de datos

## Java Scanner para lectura de datos

La clase Scanner está disponible a partir de Java 5 y facilita la lectura de datos en los programas Java. Primero veremos varios ejemplos de lectura de datos en Java con Scanner y después explicaremos en detalle cómo funciona.

Para utilizar Scanner en un programa tendremos que hacer lo siguiente:

### 1. Escribir el import

La clase Scanner se encuentra en el paquete java.util por lo tanto se debe incluir al inicio del programa la instrucción:

import java.util.Scanner;  
 import java.util.\*;

### 2. Crear un objeto Scanner

Tenemos que crear un objeto de la clase Scanner asociado al dispositivo de entrada. Si el dispositivo de entrada es el teclado escribiremos:

Scanner sc = new Scanner(System.in);

*//(System.in representa la entrada de datos por defecto en mi programa)*

Se ha creado el objeto sc asociado al teclado representado por System.in, una vez hecho esto podemos leer datos por teclado.

#### Ejemplos de lectura:

Para leer podemos usar el método nextXxx() donde Xxx indica el tipo, por ejemplo nextInt() para leer un entero, nextDouble() para leer un double, etc.

*Ejemplo* de lectura por teclado de un número entero:

int n;

System.out.print("Introduzca un número entero: ");

n = sc.nextInt();

*Ejemplo* de lectura de un número de tipo double:

double x;

System.out.print("Introduzca número de tipo double: ");

x = sc.nextDouble();

*Ejemplo* de lectura de una cadena de caracteres (una línea hasta intro):

String s;

System.out.print("Introduzca texto: ");

s = sc.nextLine();

*Ejemplo* de programa Java con lectura de datos con Scanner:

El programa pide que se introduzca el nombre de la persona y lo muestra por pantalla. A continuación lee por teclado el radio de una circunferencia de tipo double y muestra su longitud. Además lee un entero y muestra su cuadrado.

import java.util.Scanner;

public class Main {

public static void main(String[] args) {

Scanner sc = new Scanner(System.in); //crear un objeto Scanner

String nombre;

double radio;

int n;

System.out.print("Introduzca su nombre: ");

nombre = sc.nextLine(); //leer un String/linea

System.out.println("Hola " + nombre + "!!!");

System.out.print("Introduzca el radio de la circunferencia: ");

radio = sc.nextDouble(); //leer un double

System.out.println("Longitud de la circunferencia: " + 2\*Math.PI\*radio);

System.out.print("Introduzca un número entero: ");

n = sc.nextInt(); //leer un entero

System.out.println("El cuadrado es: " + Math.pow(n,2));

}

}

#### Funcionamiento de la clase Java Scanner.

De forma resumida podemos decir que cuando se introducen caracteres por teclado, el objeto Scanner toma toda la cadena introducida y la divide en elementos llamados tokens (los guarda en un buffer interno).

El carácter predeterminado que sirve de separador de tokens (*en el .nextLine()* ) es el espacio en blanco. Por ejemplo, si introducimos:

Esto es un ejemplo, lectura de datos.

Scanner divide la cadena en los siguientes tokens:

Esto

es

un

ejemplo,

lectura

de

datos.

Si introducimos la cadena:

12 20.001 Lucas w

Los tokens que se crean son:

12

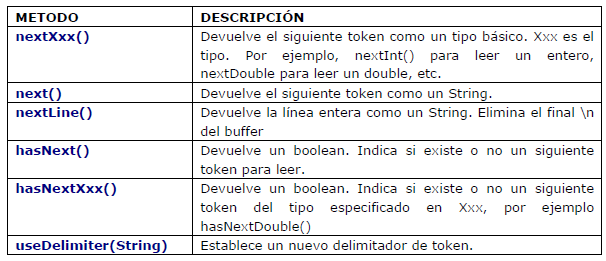
20.001

Lucas

W

A continuación, utilizando los métodos que proporciona la clase Scanner se puede acceder a esos tokens y trabajar con ellos en el programa.

Ya hemos visto el método nextXxx(). Además la clase Scanner proporciona otros métodos, algunos de los métodos más usados son:



#### Cómo limpiar el buffer de entrada en Java

Cuando en un programa se leen por teclado datos numéricos y datos de tipo carácter o String debemos tener en cuenta que al introducir los datos y pulsar intro estamos también introduciendo en el buffer de entrada el intro.

Es decir, cuando en un programa introducimos un dato y pulsamos el intro como final de entrada, el carácter intro también pasa al buffer de entrada.

Buffer de entrada si se introduce un 5: 5\n

En esta situación, la instrucción:

n = sc.nextInt();

Asigna a n el valor 5 pero el intro permanece en el buffer.

*Buffer de entrada* después de leer el entero: \n

Si ahora se pide que se introduzca por teclado una cadena de caracteres:

System.out.print("Introduzca su nombre: ");

nombre = sc.nextLine(); //leer un String

El método nextLine() extrae del buffer de entrada todos los caracteres hasta llegar a un intro y elimina el intro del buffer.

En este caso asigna una cadena vacía \n a la variable nombre y limpia el intro. Esto provoca que el programa no funcione correctamente, ya que no se detiene para que se introduzca el nombre.

***Solución***:

Se debe **limpiar el buffer** de entrada si se van a leer datos de tipo carácter a continuación de la lectura de datos numéricos.

La forma más sencilla de limpiar el buffer de entrada en Java es ejecutar la instrucción:

sc.nextLine();

Lo podemos comprobar si cambiamos el orden de lectura del ejemplo y leemos el nombre al final:

import java.util.Scanner;

public class Ejemplo {

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

String nombre;

double radio;

int n;

System.out.print("Introduzca el radio de la circunferencia: ");

radio = sc.nextDouble();

System.out.println("Longitud de la circunferencia: " + 2\*Math.PI\*radio);

System.out.print("Introduzca un número entero: ");

n = sc.nextInt();

System.out.println("El cuadrado es: " + Math.pow(n,2));

System.out.print("Introduzca su nombre: ");

nombre = sc.nextLine(); //leemos el String después del double

System.out.println("Hola " + nombre + "!!!");

}

}

Si lo ejecutamos obtendremos:

*Introduzca el radio de la circunferencia: 34*

*Longitud de la circunferencia: 213.62830044410595*

*Introduzca un número entero: 3*

*El cuadrado es: 9.0*

*Introduzca su nombre: Hola !!!*

Comprobamos que no se detiene para pedir el nombre.

Una solución es escribir la instrucción

sc.nextLine();

Después de la lectura del int y antes de leer el String:

n = sc.nextInt();

System.out.println("El cuadrado es: " + Math.pow(n,2));

sc.nextLine();

System.out.print("Introduzca su nombre: ");

nombre = sc.nextLine();

System.out.println("Hola " + nombre + "!!!");

Ahora la ejecución es correcta:

*Introduzca el radio de la circunferencia: 23*

*Longitud de la circunferencia: 144.51326206513048*

*Introduzca un número entero: 5*

*El cuadrado es: 25.0*

*Introduzca su nombre: Lucas*

*Hola Lucas!!!*

Hay una solución más elegante: leer siempre líneas con sc.nextLine(); y realizar la conversión al tipo correspondiente después:

import java.util.Scanner;

public class Scanner4 {

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

String nombre, radioS, nS;

double radio;

int n;

System.out.print("Introduzca el radio de la circunferencia: ");

radioS = sc.nextLine();

radio = Double.parseDouble(radioS);

//https://docs.oracle.com/en/java/javase/11/docs/api/java.base/java/lang/Double.html#parseDouble(java.lang.String)

System.out.println("Longitud de la circunferencia: " + 2\*Math.PI\*radio);

System.out.print("Introduzca un numero entero: ");

nS = sc.nextLine();

n = Integer.parseInt(nS);

System.out.println("El cuadrado es: " + Math.pow(n,2));

System.out.print("Introduzca su nombre: ");

nombre = sc.nextLine(); //leemos el String despues del double

System.out.println("Hola " + nombre + "!!!");

sc.close();

}

}

# UT2 Ejercicios de expresiones

**Ejemplos:**

|  |  |
| --- | --- |
| **Expresión algebraica** | **Expresión aritmética algorítmica** |
| [https://1.bp.blogspot.com/-Yprv285kbDM/VQDqr2B40EI/AAAAAAAAMIM/adExySKON-w/s1600/eq.png](http://1.bp.blogspot.com/-Yprv285kbDM/VQDqr2B40EI/AAAAAAAAMIM/adExySKON-w/s1600/eq.png) | x\*\*2 + y\*\*4 |
| [https://2.bp.blogspot.com/--Ib7mwPHA0c/VQDm5rltj8I/AAAAAAAAMIA/zSfRAD7yhZI/s1600/eq1.png](http://2.bp.blogspot.com/--Ib7mwPHA0c/VQDm5rltj8I/AAAAAAAAMIA/zSfRAD7yhZI/s1600/eq1.png) | x\*\*2 / (a\*\*3 + b\*\*3) |
| [https://4.bp.blogspot.com/-cf7MX76AhZI/VQDrLYDLD1I/AAAAAAAAMIU/jP69Uhnd89g/s1600/eq2.png](http://4.bp.blogspot.com/-cf7MX76AhZI/VQDrLYDLD1I/AAAAAAAAMIU/jP69Uhnd89g/s1600/eq2.png) | u + x\*\*2 / y |
| [https://1.bp.blogspot.com/-605-e1GP8xs/VQDsptXNVuI/AAAAAAAAMIo/4-JbjFtpFWs/s1600/eq3.png](http://1.bp.blogspot.com/-605-e1GP8xs/VQDsptXNVuI/AAAAAAAAMIo/4-JbjFtpFWs/s1600/eq3.png) | (a + b) / (a + c\*\*2 / (d + e)) |

**Deducir el valor de las expresiones siguientes: Siendo:   
A = 5; B = 25; C = 10**

1. A + B / C 7,5

2. (A + B) / C 3

3. A + B % C 10

**Si el valor de A es 4, el valor de B es 5 y el valor de C es 1, evaluar las siguientes expresiones:**

1. B \* A - B \* B / 4 \* C

20-25/4\*1 13,75

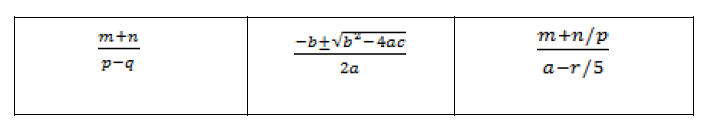
2. (A \* B) / 3 \*3

20/3\*3 20

3. ( ( ( B + C ) / 2 \* A + 10 ) \* 3 \* B ) - 6

((6/2\*4+10)\*3\*5)-6 22\*15-6=324

**Realizar las conversiones de expresiones matemáticas a expresiones algorítmicas indicando el orden de ejecución de cada una de ellas**



(m+n)/(p-q)

(-b+(Math.pow(Math.pow(b, 2) - 4\*a\*c, 1/2)))/(2\*a)

(-b-(Math.pow(Math.pow(b, 2) - 4\*a\*c, 1/2)))/(2\*a)

((m+n)/p)/((a-r)/5)

**Evaluar las expresiones lógicas aplicando la jerarquía de operadores.**

|  |  |
| --- | --- |
| 1. ((A \* B) < (B + C)) && (A= = C)  (12<6) && (3==2) FALSE | A=3, B=4 y C=2 |
| 2. ((A + B) > C) || ((B / D > B))  (7 > 3) || (5/5>5) T || (1>5) T | A=2, B=5, C=3 y D=5 |
| 3. (A/B) \* C + (A / B)  2\*3 + (2) 8 | A = 4, B = 2, C = 3 |
| 4. PI \* X\*X>Y || 2\* PI \* X <=Z  3.141592\*1\*1 > 4 ||  2\*3.141592\*1 <= 10  F | | T T  5. X>3 && Y==4 || X+Y<=Z  F && T || 5 <= 10 F || T T  6. X>3 && (Y==4 || X+Y<=Z)  F && (F || 5<=10) F && (F) F  7. !( Y/2==2\*X) && !( Y<PI-E\*Z)  !(4/2==2) && !(4<3.141592-27.18281)  !( T ) && !( F ) F && T F | X=1,  Y=4,  Z=10,  PI=3.141592  E=2.718281 |
| 8. A==B%C  5 == 4%3 5 == 1 F  9. 6/C < C % 6  6/3 < C % 6 2 < 3 T  10. C + B – 1 != A || B >= -B \* A && A \* A<=10  3 + 4 – 1 != 5 || 4 >= -4 \*5 && 5 \*5 <=10  6 != 5 || 4 >= -20 && 25 <= 10  T || T && F T || F T  11. B % A / C  4 % 5 / 3 4/3  12. !(X \*A > Y/B)  !(0.05 \* 5 > 2.3/4) !(0.25 > 0.575) !(F) T | A=5,  B=4,  C=3,  X=0.05,  Y=2.3 |

**Convertir en expresiones numéricas los siguientes enunciados.**

1. Elabore una expresión que sólo permita valores entre 1 y 10.

(x>=1 && x<=10)

2. Elabore una expresión que permita valores entre 1 y 3, y entre 5 y 7 exclusivamente.

(x>=1 && x<=3) || (x>=5 && x<=7)

3. Elabore una expresión que permita edades entre 18 y 25 años.

(edad>=18 && x<=25)

**Comprobar qué resultado se obtiene en estas 2 expresiones. ¿Son el mismo resultado? Justificar la respuesta**

1. 7==4+3 || 6<2 && 5>=8 T || F && F T || F T

2. (7==4+3 || 6<2) && 5>=8 (T || F) && F T && F F

# UT3 Estructuras de Control

**1. Estructuras de selección.**

**1.1 Estructura If.**

**1.2 Switch.**

**1.3 Operador Condicional.**

**2. Estructuras de repetición.**

**2.1 Bucle For.**

**2.2 Bucle While.**

**2.3 Bucle Do-While**

## 1.- Estructuras de selección.

Las estructuras de selección son aquellas que se utilizan cuando se quiere ejecutar una instrucción o bloque de instrucciones si se cumple una determinada condición, de ahí que también se conozcan con el nombre de condicionales.

### 1.1 Estructura If.

Permite ejecutar una instrucción (o secuencia de instrucciones) si se da una condición. La sentencia **if** es la sentencia básica de selección y existen tres variantes: simple, doble y selección múltiple, pero las 3 se basan en la misma idea.

a) **Selección Simple**. Solo tiene la parte positiva de la selección. Su sintaxis es:

|  |
| --- |
| **if (*condición*){**  ***sentencias;***  **}** |

Donde **condición** es una expresión booleana y **sentencias** representa una sentencia o bloque de sentencias, las cuales se ejecutarán si el valor de la condición es True (cierta).

Si es una **sentencia única**, se pueden quitar las llaves

**Ej:**

if (numero%2 != 0) {

System.out.println(“El numero es impar “);

cont++;

}

b) **Selección Doble.** Se añade una parte else a la sentencia if la cual se ejecutará si la condición es falsa. Su sintaxis es:

|  |
| --- |
| ***if (condición){***  ***sentencias1;***  ***}***  ***else{***  ***sentencias2;***  ***}*** |

**Ej:**

if (a>b)

System.out.println (“El número mayor es “ + a);

else

System.out.println(“El número mayor es “ + b+ “ o son iguales”);

c) **Selección Múltiple**. Es habitual cuando hay más de una condición. Su sintaxis es:

|  |
| --- |
| ***if (condición1){***  ***sentencias1;***  ***}***  ***else if (condición2){***  ***sentencias2;***  ***}***  ***else if (condición3){***  ***sentencias3;***  ***else {***  ***sentencias;***  ***}*** |

**Ej:**

if (mes == 12 || mes == 1 || mes == 2)

System.out.println ( “Invierno”);

else if (mes == 3 || mes == 4 || mes == 5)

System.out.println( “Primavera”);

else if (mes == 6 || mes == 7 || mes == 8)

System.out.println (“Verano”);

else

System.out.println (“Otoño”);

### 1.2 Switch.

Es una estructura de selección múltiple más cómoda de leer y utilizar que el else-if. Selecciona un bloque de sentencias dependiendo del valor de una expresión.

Su sintaxis es:

|  |
| --- |
| ***switch (expresion) {***  ***case valor1:sentencias;***  ***break;***  ***case valor2:sentencias;***  ***break;***  ***case valor3:sentencias;***  ***break;***  ***...***  ***default: sentencias;***  ***break;***  ***}*** |

Donde ***expresión*** tiene que tomar un valor entero o un carácter. A partir de la versión 7 de Java también se permite que la expresión sea un String (cadena), ***break*** indica que ha acabado la ejecución de ese caso y seguiría ejecutando la sentencia siguiente al switch y ***default*** es opcional y se ejecutará cuando la expresión tome un valor que no esté recogido en los distintos casos especificados.

**Ej:**

switch (mes) {

case 4:

case 6:

case 9:

case 11: dias = 30;

break;

case 2: dias = 28;

break;

default: dias = 31;

break;

}

Es posible juntar distintos casos, dejándolos en blanco y especificando las instrucciones en el último de los casos de grupo.

En el ejemplo, si la variable mes toma los valores 4, 6, 9 u 11 se hace en los 4 casos lo mismo, se le asigna a la variable día el valor 30.

### 1.3 Operador Condicional.

El operador condicional es un operador ternario, es decir, consta de tres operandos y su función es asignar un valor entre dos posibles a una variable si se cumple o no una condición.

Su sintaxis es la siguiente:

|  |
| --- |
| ***tipo variable = (condicion) ? v\_cond\_true: v\_cond\_false;*** |

Realmente es un if…else simple que podemos utilizar si solo queremos asignar un valor a una variable si se cumple o no una condición.

Ejemplo:

**public** **class** EjemploOperadorCondicional

{

**public** **static** **void** main([String](http://www.google.com/search?hl=en&q=allinurl%3Astring+java.sun.com&btnI=I'm Feeling Lucky)[] args)

   {

**int** edad = 18;

*//Utilizando el operador condicional*

[String](http://www.google.com/search?hl=en&q=allinurl%3Astring+java.sun.com&btnI=I'm Feeling Lucky) resultado = (edad >=18) ? "Mayor de edad." : "Menor de edad.";

[System](http://www.google.com/search?hl=en&q=allinurl%3Asystem+java.sun.com&btnI=I'm Feeling Lucky).out.println(resultado);

*//Utilizando if...else*

**if**(edad >= 18)

       {

[System](http://www.google.com/search?hl=en&q=allinurl%3Asystem+java.sun.com&btnI=I'm Feeling Lucky).out.println("Mayor de edad.");

       }

**else**

       {

[System](http://www.google.com/search?hl=en&q=allinurl%3Asystem+java.sun.com&btnI=I'm Feeling Lucky).out.println("Menor de edad.");

       }

   }

}

## 2.- Estructuras de repetición.

Estas estructuras se utilizan para repetir un bloque de sentencias un número de veces. Son también llamadas sentencias de iteración o bucles.

Los tipos de bucles que hay son: ***for***, cuando se sabe el número de veces que se va realizar y ***while o do-while***, cuando no se conoce de antemano el número de veces que se repetirá el bloque de sentencias.

### 2.1 Bucle For (PARA)

La sintaxis de la sentencia es:

|  |
| --- |
| ***for(inicialización;condición;incremento/decremento){***  ***sentencias;***  ***}*** |

Dónde:

* **Inicialización** se realiza solo una vez, **antes de la primera iteración**.
* **Condición** se comprueba **cada vez, antes de entrar** al bucle y si es cierta entra, si no lo es se termina la ejecución de la sentencia y pasa a ejecutarse la siguiente sentencia al for.
* **Incremento/decremento** se realiza siempre **después de terminar de ejecutar las sentencias del cuerpo de la iteración** y antes de volver a comprobar la condición de nuevo.

**Ej**:

int i;

for (i=0; i<5; i++)

System.out.println (i);

El funcionamiento de esta instrucción seria el siguiente:

**1º.** Se inicializa la i.

En el ejemplo i toma el valor 0.

**2º.** Se comprueba si cumple la condición, si se cumple entra en el bucle pero si no la cumple no entraría.

En el ejemplo sí la cumple pues 0<5.

**3º.** Se ejecuta el bloque de sentencias que pertenezcan al for.

En el ejemplo escribirá en pantalla un 0

**4º.** Vuelve al principio del bucle, incrementa la i.

En el ejemplo se incrementa en 1, con lo cual pasa a tomar valor 1.

**5º.** Se repite el proceso desde el paso 2, hasta que deje de cumplirse la condición.

En el ejemplo se repetirá hasta que i tome el valor de 5 pues no se cumple que 5<5 Los siguientes dos ejemplos son un bucle infinito y un bucle que nunca se llega a ejecutar.

**Bucle infinito**: for (i=1;i<5;i--) pues nunca se dejará de cumplir que i<5.

**Bucle que no ejecuta nunca sus sentencias**: for (i=5; i<3;i++) pues desde el principio no se cumple la condición pues 5 no es menor que 3.

### 2.2 Bucle While (MIENTRAS)

La sintaxis de la sentencia es:

|  |
| --- |
| ***while (condición){***  ***sentencias;***  ***}*** |

Donde **condición** es una expresión booleana que se evalúa al principio del bucle y antes de cada iteración de las sentencias.

Si la ***condición es verdadera***, se ejecuta el bloque de sentencias, y se vuelve al principio del bucle.

Si la ***condición es falsa***, no se ejecuta el bloque de sentencias, y se continúa con las siguientes sentencias del programa.

Si la condición es falsa desde un principio, entonces el bucle nunca se ejecuta.

Si la condición nunca llega a ser falsa, se tiene un bucle infinito.

**Ej:**

int i=0;

while (i<5){

System.out.println (i);

i++;

}

El resultado de ejecutar este bloque de instrucciones es el mismo que para el for, escribirá en pantalla del 0 al 4 en diferentes líneas.

Los dos siguientes ejemplos son también un bucle infinito y un bucle que no se ejecutará nunca.

**Bucle infinito:**

int i=0;

while (i<5){

System.out.println(i);

}

Sera infinito pues al no cambiar el valor de la i nunca, la condición siempre se está cumpliendo.

**Bucle que nunca se ejecutará:**

int i=6;

while (i<5){

System.out.println(i);

i++;

}

No se ejecuta nunca el bucle pues desde el principio no se cumple la condición que permita entrar en el while, pues 6 no es menor que 5.

### 2.3 Bucle Do-While (REPETIR HASTA QUE)

La sintaxis de la sentencia es:

|  |
| --- |
| *do{*  *sentencias;*  *}while (condición);* |

Es muy parecida a while. El bloque de sentencias se repite mientras se cumpla la condición pero en este caso, la condición se comprueba después de ejecutar el bloque de sentencias por lo que el bloque **se ejecuta siempre al menos una vez**.

*Este tipo de bucle será muy útil cuando se quiere obligar a que una determinada variable solo pueda tomar unos ciertos valores, o cuando se quiere comprobar una contraseña antes de seguir ejecutando el bucle, etc…*

**Ej**:

int i;

do {

System.out.println (“Introducir un numero distinto de 0”);

i = teclado.nextInt();

} while(i == 0);

En este ejemplo se pide introducir un número que no sea 0, con lo cual no se saldrá del bucle mientras se introduzca un 0. (Probarlo en clase).

# UT3 Java printf para dar formato a los datos de salida

Vamos a ver cómo utilizar printf para dar formato a los datos que se imprimen por pantalla en Java. Este problema se nos plantea por ejemplo cuando queremos mostrar un número de tipo float o double con un número determinado de decimales y no con los que por defecto muestra Java.

A partir de la versión Java 5 se incorporan los métodos format y printf que permiten aplicar un formato a la salida de datos por pantalla. Ambos realizan la misma función, tienen exactamente el mismo formato y emulan la impresión con formato printf() de C.

Veamos primero varios ejemplos de printf en Java y después explicaremos en detalle la sintaxis de printf. Si queremos mostrar el número 12.3698 de tipo double con dos decimales:

**System.out.printf("%.2f %n", 12.3698);**

El primer % indica que en esa posición se va a escribir un valor. El valor a escribir se encuentra a continuación de las comillas, .2 indica el número de decimales. La f indica que el número es de tipo **float** o **double**. En la tabla que aparece más adelante podéis ver todos los caracteres de conversión para todos los tipos de datos. %n indica un salto de línea, equivale a \n. Con printf podemos usar ambos para hacer un salto de línea.

**La salida por pantalla es: 12,37**

Comprobamos que printf realiza un redondeo para mostrar los decimales indicados.

Lo más común será que tengamos el valor en una variable, en ese caso si queremos escribir el valor de n con tres decimales:

double n = 1.25036;

System.out.printf("%.3f %n", n);

Salida:

1,250

Para mostrar el signo + en un número positivo:

double n = 1.25036;

System.out.printf("%+.3f %n", n);

Salida:

+1.250

Si el número a mostrar es un **entero** se utiliza el carácter d:

int x = 10;

System.out.printf("%d %n", x);

Salida:

10

Para mostrarlo con signo:

int x = 10;

System.out.printf("%+d %n", x);

Salida:

+10

Para mostrar varias variables pondremos **tantos % como valores vamos a mostrar**. Las variables se escriben a continuación de las comillas separadas por comas:

double n = 1.25036;

int x = 10;

System.out.printf("n = %.2f x = %d %n", n, x);

Salida:

n = 1,25 x = 10

Cuando hay varias variables podemos indicar de cuál de ellas es el valor a mostrar escribiendo 1$, 2$, 3$, ... indicando que el valor a mostrar es el de la primera variable que aparece a continuación de las comillas, de la segunda, etc.

La instrucción anterior la podemos escribir así:

System.out.printf("n = %1$.2f x = %2$d %n", n, x);

Este número es opcional, si no aparece se entenderá que el primer valor proviene de la primera variable, el segundo de la segunda, etc.

Si queremos mostrar el número 123.4567 y su cuadrado ambos con dos decimales debemos escribir:

double n = 123.4567;

System.out.printf("El cuadrado de %.2f es %.2f\n", n, n\*n);

Salida:

El cuadrado de 123,46 es 15241,56

**printf** permite mostrar valores con un ancho de campo determinado. Por ejemplo, si queremos mostrar el contenido de n en un ancho de campo de 10 caracteres escribimos:

double n = 1.25036;

System.out.printf("%+10.2f %n", n);

Salida:

*bbbbb*+1.25

Donde cada *b* indica un espacio en blanco.

El 10 indica el tamaño en caracteres que ocupará el número en pantalla. Se cuentan además de las cifras del número el punto decimal y el signo si lo lleva. En este caso el número ocupa un espacio de 5 caracteres (3 cifras, un punto y el signo) por lo tanto se añaden 5 espacios en blanco al principio para completar el tamaño de 10.

Si queremos que en lugar de espacios en blancos nos muestre el número completando el ancho con ceros escribimos:

System.out.printf("%+010.2f %n", n);

Salida:

+000001.25

Más ejemplos de printf:

Mostrar el número 1.22 en un ancho de campo de 10 caracteres y con dos decimales.

double precio = 1.22;

System.out.printf("%10.2f", precio);

Salida:

*bbbbbb*1.22

(el carácter b indica un espacio en blanco)

El número ocupa un espacio total de 10 caracteres incluyendo el punto y los dos decimales.

Mostrar la cadena "Total:" con un ancho de 10 caracteres y alineada a la izquierda:

System.out.printf("%-10s", "Total:");

Salida:

Total:*bbbb*

El caracter s indica que se va a mostrar una cadena de caracteres.

El signo - indica alineación a la izquierda.

Mostrar la cadena "Total:" con un ancho de 10 caracteres y alineada a la derecha:

System.out.printf("%10s", "Total:");

Salida:

*bbbb*Total:

Al final puedes ver un ejemplo completo con distintos usos de printf.

Veamos ahora detenidamente la sintaxis de printf:

La sintaxis general de printf es:

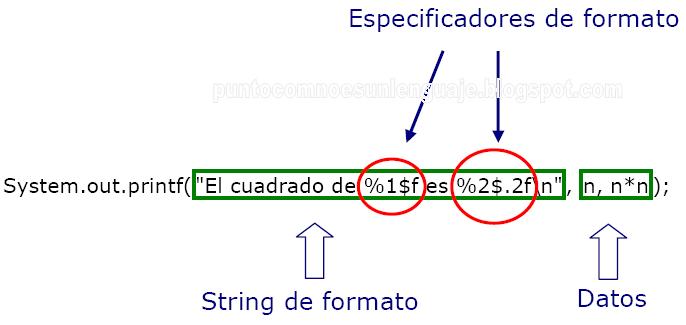
**printf (String de formato, Object … datos);**

El *String de formato* es una cadena de caracteres que contiene:

- **Texto fijo** que será mostrado tal cual.

- **Especificadores de formato** que determinan la forma en que se van a mostrar los datos.

Los *datos* representan la información que se va a mostrar y sobre la que se aplica el formato anterior. El número de datos que se pueden mostrar es variable.



Explicación de cada una de las partes que aparecen en la instrucción printf:

**Especificadores de formato:**

La sintaxis para los especificadores de formato de printf es:

%[posición\_dato$][indicador\_de\_formato][ancho][.precisión]carácter\_de\_conversión

Los elementos entre corchetes son opcionales.

**posición\_dato**: indica la posición del dato sobre el que se va aplicar el formato. El primero por la izquierda ocupa la posición 1.

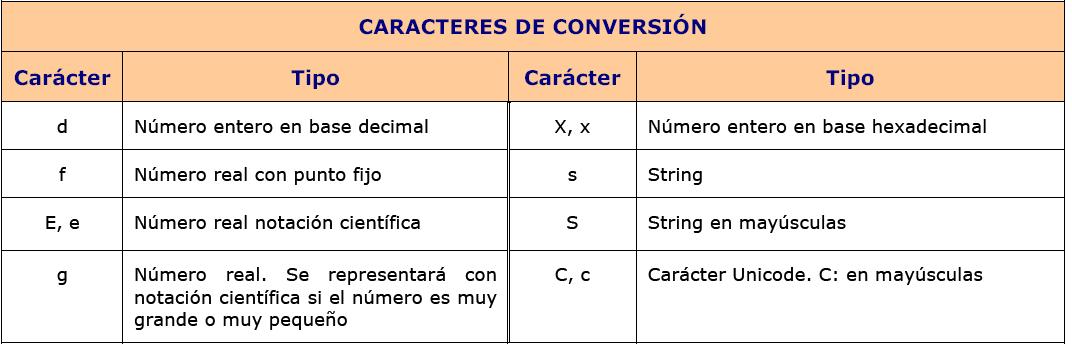
**indicador\_de\_formato**: es el conjunto de caracteres que determina el formato de salida. Los indicadores de formato de printf en Java son:



**ancho**: Indica el tamaño mínimo, medido en número de caracteres, que debe ocupar el dato en pantalla.

**.precisión**: Indica el número de decimales que serán representados. Solo aplicable a datos de tipo float o double.

**carácter\_de\_conversión**: Carácter que indica cómo tiene que ser formateado el dato. Los más utilizados se muestran en la tabla.



Ejemplo completo con distintos usos de printf en Java:

**public** **class** Printf2 {

**public** **static** **void** main(String[] args) {

**double** q = 1.0/3.0;

System.***out***.printf("1.0/3.0 = %5.3f %n", q);

System.***out***.printf("1.0/3.0 = %7.5f %n", q);

q = 1.0/2.0;

System.***out***.printf("1.0/2.0 = %09.3f %n", q);

q = 1000.0/3.0;

System.***out***.printf("1000/3.0 = %7.1e %n", q);

q = 3.0/4567.0;

System.***out***.printf("3.0/4567.0 = %7.3e %n", q);

q = -1.0/0.0;

System.***out***.printf("-1.0/0.0 = %7.2e %n", q);

q = 0.0/0.0;

System.***out***.printf("0.0/0.0 = %5.2e %n", q);

System.***out***.printf("pi = %5.3f; e = %10.4f %n", Math.***PI***, Math.***E***);

**double** r = 1.1;

System.***out***.printf("%1$.5f \*%1$.5f %n", Math.***PI***, r);

System.***out***.printf("%1$6.6f \*%1$6.6f %n", Math.***PI***, r);

System.***out***.printf("C = 2 \* %1$5.5f \* %2$4.1f, "+"A = %2$4.1f \* %2$4.1f \* %1$5.5f %n", Math.***PI***, r);

}

}

**Salida**:

1.0/3.0 = 0,333

1.0/3.0 = 0,33333

1.0/2.0 = 00000,500

1000/3.0 = 3,3e+02

3.0/4567.0 = 6,569e-04

-1.0/0.0 = -Infinity

0.0/0.0 = NaN

pi = 3,142; e = 2,7183

3,14159 \*3,14159

3,141593 \*3,141593

C = 2 \* 3,14159 \* 1,1, A = 1,1 \* 1,1 \* 3,14159

## 1. Formatter (clase) java.util.Formatter

Esta clase permite generar String a partir de datos usando una especificación de formato.

La funcionalidad se presta a diferentes clases para un uso cómodo:

String s =String.format(formato, valores ...);

System.out.printf(formato, valores ...);

System.out.format(formato, valores ...);

Es muy similar a printf.

Formatter fmtr = new Formatter();

String saludo =   
String.format( "Hola amigos, bienvenidos a %s !", "Madrid");

System.out.println(saludo);

saludo =   
String.format("Hola %2$s, bienvenidos %1$s !", "Madrid", "chicos");

System.out.println(saludo);

int numero = 425;

fmtr.format("%08d", numero);

System.out.println("El numero formateado " + fmtr);

Object result = fmtr.format("%1$4d - el año %2$4.2f", 1951, Math.PI);

System.out.println(result);

// Otra forma de hacerlo

System.out.format("%1$04d - el año %2$4.2f%n", 1951, Math.PI);

// Parecido al printf que ya vimos:

System.out.printf("%1$04d - el año %2$4.2f%n", 1951, Math.PI);

System.out.printf("PI es mas o menos %1$4.2f%n", Math.PI);

System.out.printf("Tu nombre es %1$s y el mio es %2$s%n", "Juan", "Pepi");

fmtr.close();

### 1.1. Como usar el Formateador decimal en Java

La clase **DecimalFormat**: Es una **clase de Java** que nos permite mostrar los números con un formato deseado, puede ser limitando los dígitos decimales, si usaremos punto o coma, etc, incluso podemos tomar los valores desde un **JTextField** y reconstruir el número, ejemplo:

//import requerido para el Formateador

import java.text.DecimalFormat;

DecimalFormat formateador = new DecimalFormat("####.####");

// Imprime esto con cuatro decimales, es decir: 7,1234

System.out.println (formateador.format (7.12342383));

Si utilizamos ceros en lugar de los #, los dígitos no existentes se rellenarán con ceros, ejemplo:

DecimalFormat formateador = new DecimalFormat("0000.0000");

// Imprime con 4 cifras enteras y 4 decimales: 0001,8200

System.out.println (formateador.format (1.82));

También podemos utilizar el signo de porcentaje (%) en la máscara y así el número se multiplicará automáticamente por 100 al momento de Imprimir.

DecimalFormat formateador = new DecimalFormat("###.##%");

// Imprime: 68,44%

System.out.println (formateador.format(0.6844));

## DecimalFormat (clase)

La clase **DecimalFormat** usa por defecto el formato para el lenguaje que tengamos instalado en el ordenador. Es decir, si nuestro sistema operativo está en español, se usará la coma para los decimales y el punto para los separadores de miles. Si estamos en inglés, se usará el punto decimal. Una opción para cambiar esto, es utilizar la clase **DecimalFormatSymbols**, que vendrá rellena con lo del idioma por defecto, y cambiar en ella el símbolo que nos interese. Por ejemplo, si estamos en español y queremos usar el punto decimal en vez de la coma, podemos:

import java.text.DecimalFormat;

import java.text.DecimalFormatSymbols;

DecimalFormatSymbols simbolos = new DecimalFormatSymbols();

simbolos.setDecimalSeparator('.');

DecimalFormat formateador = new DecimalFormat("####.####",simbolos);

// Imprime: 3.4324

System.out.println(formateador.format (3.43242383));

Reconstruyendo un Número:

Supongamos que leemos algún número desde consola o un JTextField y deseamos reconstruirlo con DecimalFormat, se hace de la siguiente forma:

JTextField textField = new JTextField();

DecimalFormat formateador = new DecimalFormat("####.####");

String texto = textField.getText();

try

{

// parse() lanza una ParseException en caso de fallo que hay que capturar.

Number numero = formateador.parse(texto);

double valor = numero.doubleValue();

// Optimizando las Lineas anteriores:

// double valor = formateador.parse(texto).doubleValue();

}

catch (ParseException e)

{

// Error. El usuario ha escrito algo que no se puede convertir a número.  
  
}

# UT3 Otra forma de leer del usuario

## 1. Alertas y diálogos en Java

La clase JOptionPane es una clase que hereda de JComponent. Esta clase nos permitirá crear alertas o cuadros de diálogo simples para poder solicitar o mostrar información al usuario.

Los métodos que veremos son:

1. JOptionPane.showMessageDialog(…);
2. JOptionPane.showConfirmDialog(….);
3. JOptionPane.showInputDialog(…);
4. JOptionPane.showOptionDialog(…);

Cada uno de estos métodos presenta una particularidad distinta pero todos ellos nos muestran una ventana pop up que nos permitirá captar información del usuario.

### showMessageDialog

**JOptionPane. showMessageDialog (Component componentePadre, Object mensaje, String titulo, int tipoDeMensaje)**

Nos sirve para mostrar información, por ejemplo alguna alerta que queremos hacerle al usuario. Veamos cuales son los principales argumentos del método:

***componentePadre***: es el Frame desde el cual lo llamamos. Si queremos lo podemos poner null de momento. Si hubiera un padre, el dialogo se mostraría sobre él.

***mensaje***: es lo que queremos que diga el cuadro de dialogo.

***titulo***: el título de la ventana.

***tipoDeMensaje***: son constantes que le dirán a java qué tipo de mensaje queremos mostrar. De acuerdo a esto serán los iconos que se mostrarán en el cuadro de dialogo.

Las opciones son:

* **ERROR\_MESSAGE**
* **INFORMATION\_MESSAGE**
* **WARNING\_MESSAGE**
* **QUESTION\_MESSAGE**
* **PLAIN\_MESSAGE**

Si quisiéramos mostrar un icono personalizado, podemos agregarlo al final como un argumento más.

|  |
| --- |
| Ejemplo:  **JOptionPane.showMessageDialog(null, "Acceso Denegado", "Error", JOptionPane.ERROR\_MESSAGE);** |

### showConfirmDialog

Este método sirve para pedirle al usuario una confirmación. Por ejemplo, una confirmación de salida del sistema

**Int respuesta = JOptionPane. showConfirmDialog(Component componentePadre, Object mensaje, String titulo, int tipoDeOpcion);**

**Lo anterior es la versión corta de los argumentos del método. La versión larga incluye el tipo de mensaje y el icono, por si queremos personalizarlo.**

Los argumentos son idénticos a los del método anterior. Excepto por el tipo de opción, que es otra constante y los valores pueden ser:

* **DEFAULT\_OPTION**
* **YES\_NO\_OPTION**
* **YES\_NO\_CANCEL\_OPTION**
* **OK\_CANCEL\_OPTION**

Como vemos, el método devuelve un entero que nos permitirá captar cual es la opción elegida por el usuario. Los valores serán **0** para **Sí**, **1** para **No**, **2** para **Cancelar** y **-1** para el **cierre de la ventana**. Así podremos preguntar cuál es el valor devuelto y realizar la acción que deseamos.

|  |
| --- |
| Ejemplo:  **int i =JOptionPane.showConfirmDialog(this, "¿Realmente Desea Salir de Hola Swing?",  "Confirmar Salida",JOptionPane.YES\_NO\_OPTION);**  **if(i= =0){**  **System.exit(0);**  **}** |

### showInputDialog

Este método nos muestra una ventana donde podremos insertar un String. Por ejemplo, cuando queremos que el usuario inserte su nombre. **La versión corta del método es**:

**String respuesta = JOptionPane.showInputDialog(Object mensaje)**

Este método devolverá un String para poder utilizarlo después. La versión larga de los argumentos del método es similar a los anteriores.

|  |
| --- |
| Ejemplo:  **String nombre =JOptionPane.showInputDialog("Inserte su nombre");** |

También podemos crear un cuadro de dialogo que contenga un combo con las opciones predeterminadas que le queremos dar al usuario.

|  |
| --- |
| Ejemplo:  **Object[] valoresPosibles = {"Pedro", "Juan", "Carlos" };**  **Object jefe = JOptionPane.showInputDialog(null, "Seleccione cual es su Jefe Inmediato", "Seleccionar Jefe", JOptionPane.QUESTION\_MESSAGE, null, valoresPosibles, valoresPosibles[0]);** |

El array de valores posibles nos muestra en un combo cuáles serán los jefes que podemos mostrar. El último argumento del método nos muestra cuál será la opción seleccionada por defecto

### showOptionDialog

Con este método podemos crear cuadros de diálogos con botones personalizados. Está bien para personalizar los cuadros de dialogo.

El método tiene la forma:

**int res = JOptionPane.showOptionDialog(Component componentePadre, Object mensaje, String titulo, int tipoDeOpcion, int tipoMensaje, Icon icono, Object[] botones, Object botonDefault)**

**Aquí lo único que varía con el resto, es el array de botones que vamos a tener, debemos destacar que no hace falta que creemos botones, solo tenemos que poner cuál será el texto que saldrá en él.**

|  |
| --- |
| Object[] botones = {"No, nada", "Un poquito", "Me estalla" };  int i=JOptionPane.showOptionDialog(null, "¿Te duele la cabeza?", "ventanita",  JOptionPane.DEFAULT\_OPTION, JOptionPane.WARNING\_MESSAGE,  null, botones, botones[0]);  System.out.println(i); |

Después podemos tomar la respuesta como lo hacíamos con el confirmDialog.

Se puede consultar el API y https://serprogramador.es/programando-mensajes-de-dialogo-en-java-parte-1/

## 2. Convertir String a número

**Instrucciones de Java: ParseInt, ParseDouble y ParseFloat:**

La función parseFloat() analiza una cadena y devuelve un número de punto flotante.

Float.parseFloat(String str)

La función parseInt() analiza una cadena y devuelve un entero.

Integer.parseInteger(String str)

La función parseDouble() analiza una cadena y devuelve un número decimal doble.

Double.parseDouble(String str)

# UT3 Sus Métodos en Java, funciones y procedimientos. Cómo hacerlos y usarlos

Los métodos en Java, las funciones y los procedimientos, especialmente en Java, son una herramienta indispensable para programar. Java nos permite crear o hacer nuestros propios métodos y usarlos sencillamente como también nos facilita hacer uso de los métodos de otras librerías (funciones matemáticas, aritméticas, de archivos, de fechas, etc. Cualquiera que sea el caso, las funciones permiten automatizar tareas que requiramos con frecuencia y que además se puedan generalizar por medio de parámetros o argumentos.

Aprender a crear métodos en Java y usarlos correctamente es de gran importancia, separar nuestro código en módulos y según las tareas que requerimos. En java una función debe contener la implementación de una utilidad de nuestra aplicación, esto nos pide que por cada utilidad básica (abrir, cerrar, cargar, mover, etc.) sería adecuado tener al menos una función asociada a ésta, pues sería muy complejo usar o crear un método que haga todo de una sola vez, por esto es muy buena idea separar cada tarea en una función o método (según corresponda).

En Java es mucho más común hablar de métodos que de funciones y procedimientos y esto se debe a que en realidad un método, una función y un procedimiento NO son lo mismo, veamos la diferencia:

## 1. ¿Funciones, métodos o procedimientos?

**Es muy común entre programadores que se hable indistintamente de estos tres términos, sin embargo poseen diferencias fundamentales.**

### Funciones:

Las funciones son un conjunto de líneas de código (instrucciones), encapsulados en un bloque, usualmente reciben parámetros, cuyos valores utilizan para efectuar operaciones y adicionalmente devuelven/retornan un valor. En otras palabras, una función puede recibir parámetros o argumentos (algunas no reciben nada), hace uso de dichos valores recibidos como sea necesario y devuelve un valor usando la instrucción **return**. **Si no devuelve algo, entonces no es una función (se suele llamar procedimiento en este caso)**.

### Métodos:

Los métodos y las funciones (y/o procedimientos) en Java pueden realizar las mismas tareas, es decir, son funcionalmente idénticos, pero su diferencia radica en la manera en que hacemos uso de uno u otro (el contexto). Un método también puede recibir valores, efectuar operaciones con estos y retornar valores, sin embargo un método está asociado a un objeto, **SIEMPRE**, **básicamente un método es una función que pertenece a un objeto o clase, mientras que una función existe por sí sola, sin necesidad de un objeto para ser usada**.

**Nota:** En Java se debe hablar de métodos y no de funciones, pues en Java estamos siempre obligados a crear un objeto para usar el método. Para que sea una función/procedimiento, esta debe ser static, para que no requiera de un objeto para ser llamada.

### Procedimientos:

Los procedimientos son básicamente un conjunto de instrucciones que se ejecutan sin retornar ningún valor, pueden recibir o no argumentos. En el contexto de Java un procedimiento es básicamente un método cuyo tipo de retorno es void, que no nos obliga a utilizar una sentencia return.

## 2. Crear un método en Java

La sintaxis para declarar una función es muy simple, veamos:

*Ejemplo:* ***public static void main (String[] args)***

**[acceso] [modificador] tipoDevuelto nombreFuncion([tipo nombreArgumento,[tipo nombreArgumento]...])**

***//Argumento es lo mismo que variable***

{

/\*

\* Bloque de instrucciones

\*/

return valor;

*//return: devuelve solo el valor de la variable, no la variable*

}

El primer componente corresponde al **[modificador de acceso](https://www.programarya.com/Cursos/Java/Modificadores-de-Acceso" \t "Modificadores de acceso en Java)**, que puede ser public, private o protected, este último es opcional, si no ponemos nada, se asume el modificador de acceso por defecto.

El segundo componente es el **[modificador](https://www.programarya.com/Cursos/Java/Sistema-de-Tipos/Final-y-Constantes" \t "Final y constantes en Java)** que puede ser final o static (o ambas), también es opcional. Recordemos que un método o función siempre retorna algo, por lo tanto es obligatorio declararle un **tipo** (el tercer componente de la sintaxis anterior), puede ser entero (int), booleano (boolean), o cualquiera que consideremos, incluso tipos complejos. Si no es función, pondremos void.

Luego debemos darle un **nombre** a dicha función/método/procedimiento, para poder identificarla y llamarla (invocarla) durante la ejecución. Después, en el interior del paréntesis, podemos poner los **argumentos** o parámetros.

Cuando hemos acabado con la definición de la "firma" o prototipo del método, definimos su funcionamiento entre llaves. Todo lo que esté dentro de las llaves, es parte del cuerpo del método y este se ejecuta hasta llegar a una instrucción *return* o cuando se alcanza la última sentencia a ejecutar.

## 3. Return y void

En algunas ocasiones, no es necesario que el método estático tenga que devolver un valor al finalizar su ejecución. En este caso, el tipo de dato que debe indicar en la cabecera de declaración del método es el tipo void y la sentencia return no viene seguida de ninguna expresión. Este return solitario se puede omitir.

Sintaxis:

return;

//Quiere decir vuelvo al sitio donde me han llamado para seguir con la siguiente instrucción, no continua a partir de aqui

**Ejemplo:**

Seguidamente se muestra un ejemplo de declaración y uso de un método que devuelve el cubo de un valor numérico real con una sentencia return:

/\*\*

\* Demostracion del metodo cubo

\*/

public class PruebaCubo {

public static void main (String [] args){

System.out.println("El cubo de 7.5 es: " + cubo(7.5));

// llamada dentro de la sentencia System.out.println

}

public static double cubo (double x) {

// devuelve un double

// declaracion

return x\*x\*x;

}

}

Si no hay sentencia return dentro de un método, su ejecución continúa hasta que se alcanza el final del método y entonces se devuelve la secuencia de ejecución al lugar dónde se invocó al método.

Un método cuyo tipo de retorno no es void necesita siempre devolver algo. Si el código de un método contiene varias sentencias if debe asegurarse de que cada una de las posibles opciones devuelve un valor. En caso contrario, se generaría un error de compilación.

Por ejemplo:

/\*\*

\* Demostracion de la funcion esPositivo

\*/

public class PruebaPositivo {

public static void main (String [] args) {

for (int i=5; i>=-5; i--)

System.out.println(i + " es positivo: " + esPositivo(i));

}

public static boolean esPositivo(int x) {

if (x<0) return false;

if (x>0) return true;

}

}

Ejemplo de intento de compilación del código anterior:

$>javac PruebaPositivo.java

pruebaPositivo.java:14: missing return statement

}^

***Lo que pasa es que falta el valor 0, que no es ni positivo ni negativo, por eso te da error.***

# UT3 Generación de números aleatorios

## 1. Números aleatorios en Java

Para generar números aleatorios en Java tenemos dos opciones. Por un lado, podemos usar *Math.random()*, por otro la clase *java.util.Random*. La primera es de uso más sencillo y rápido. La segunda nos da más opciones.

* [Math.random()](http://chuwiki.chuidiang.org/index.php?title=Generar_números_aleatorios_en_Java" \l "Math.random.28.29)
* [Clase java.util.Random](http://chuwiki.chuidiang.org/index.php?title=Generar_números_aleatorios_en_Java" \l "Clase_java.util.Random)

## 2. Math.random()

La llamada a *Math.random()* devuelve un número aleatorio entre 0.0 y 1.0, excluido este último valor, es decir, puede devolver 0.346442, 0.2344234, 0.98345,....

En muchas de nuestras aplicaciones no nos servirá este rango de valores. Por ejemplo, si queremos simular una tirada de dado, queremos números entre 1 y 6 sin decimales. Debemos echar unas cuentas para obtener lo deseado.

En primer lugar, miramos cuántos valores queremos. En nuestro caso del dado son 6 valores, del 1 al 6 ambos incluido. Debemos entonces multiplicar *Math.random()* por 6.

Math.random()\*6 // Esto da valores de 0.0 a 6.0, excluido el 6.0

[Math](http://www.google.com/search?hl=en&q=allinurl%3AMath+java.sun.com&btnI=I'm Feeling Lucky)**.random()\*limite+1;**

Donde límite será el número más alto del rango. Los números genéricos que generaremos irán entre el 1 y el límite.

Math.random()\*6 + 1

// Esto da valores entre 1.0 y 7.0 excluido el 7.0

Finalmente, para conseguir un entero, quitamos los decimales usando la clase *Math.floor()*

int valorDado = Math.floor(Math.random()\*6+1);

*// A mi me sale que Math.floor te lo pasa a un double, igual es Math.round*

O también haciendo un casting a un entero:

**int numeroAleatorio = (int) (**[Math](http://www.google.com/search?hl=en&q=allinurl%3AMath+java.sun.com&btnI=I'm Feeling Lucky)**.random()\*6+1);**

En general, para conseguir un número entero entre [N, M] con N < M y ambos incluidos, debemos usar esta fórmula:

int valorEntero = Math.floor(Math.random()\*(M-N+1)+N);

// Valor entre M y N, ambos incluidos.

## 3. Clase java.util.Random

La clase *java.util.Random* debemos instanciarla, a diferencia del método *Math.random()*. A cambio, tendremos bastantes más posibilidades.

Podemos usar un constructor sin parámetros o bien pasarle una semilla. Si instanciamos varias veces la clase con la misma semilla, tendremos siempre la misma secuencia de números aleatorios.

Random r1 = new Random();

Random r2 = new Random(4234);

Random r3 = new Random(4234); // r2 y r3 darán la misma secuencia.

Lo más fácil es usar el constructor sin parámetros, que normalmente dará secuencias distintas en cada instancia. De todas formas, una manera de obtener una semilla que sea distinta cada vez que ejecutemos nuestro programa puede ser obtener el tiempo actual en milisegundos con *System.currentTimeMillis()*, que dará números distintos salvo que hagamos la instancia justo en el mismo instante de tiempo.

Con esta clase, una vez instanciada, nuestro problema del dado sería bastante más sencillo, usando el método *nextInt(int n)*, que devuelve un valor entre 0 y n, excluido n

Random r = new Random();

int valorDado = r.nextInt(6)+1; // Entre 0 y 5, más 1. Es decir, 1-6

También tenemos funciones que nos dan un valor aleatorio siguiendo una curva de Gauss o que nos rellenan un array de bytes de forma aleatoria. Y por supuesto, el *nextDouble()* que devuelve un valor aleatorio entre 0.0 y 1.0, excluido este último.

La clase *Random* proporciona un generador de números aleatorios que es más flexible que la función estática [random](http://www.sc.ehu.es/sbweb/fisica/cursoJava/fundamentos/estatico/math/math.htm" \l "azar) de la clase *Math*.

Para crear una secuencia de números aleatorios tenemos que seguir los siguientes pasos:

1. Proporcionar a nuestro programa información acerca de la clase *Random.* Al principio del programa escribiremos la siguiente sentencia.

import java.util.Random;

1. Crear un objeto de la clase *Random*
2. Llamar a una de las funciones miembro que generan un número aleatorio
3. Usar el número aleatorio.

### 3.1 Constructores

La clase dispone de dos constructores, el primero crea un generador de números aleatorios cuya semilla es inicializada en base al instante de tiempo actual.

Random rnd = new Random();

El segundo, inicializa la semilla con un número del tipo **long**.

Random rnd = new Random(3816L);

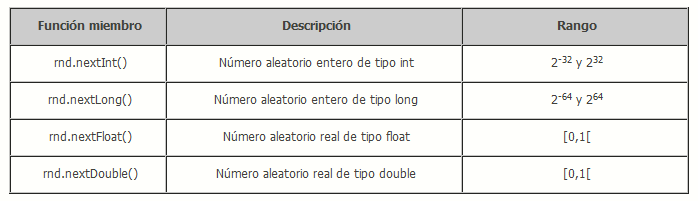
El sufijo L no es necesario, ya que aunque 3816 es un número **int** por defecto, es promocionado automáticamente a **long**.

Aunque no podemos predecir que números se generarán con una semilla particular, podemos sin embargo, duplicar una serie de números aleatorios usando la misma semilla. Es decir, cada vez que creamos un objeto de la clase *Random* con la misma semilla obtendremos la misma secuencia de números aleatorios. Esto no es útil en el caso de loterías, pero puede ser útil en el caso de juegos, exámenes basados en una secuencia de preguntas aleatorias, las mismas para cada uno de los estudiantes, simulaciones que se repitan de la misma forma una y otra vez, etc.

### 3.2 Funciones miembro

Podemos cambiar la semilla de los números aleatorios en cualquier momento, llamando a la función miembro *setSeed*.

rnd.setSeed(3816);



Podemos generar números aleatorios de cuatro formas diferentes:

rnd.nextInt();

Genera un número aleatorio entero de tipo **int**

rnd.nextLong();

Genera un número aleatorio entero de tipo **long**

rnd.nextFloat();

Genera un número aleatorio de tipo **float** entre 0.0 y 1.0, aunque siempre menor que 1.0

rnd.nextDouble();

Genera un número aleatorio de tipo **double** entre 0.0 y 1.0, aunque siempre menor que 1.0

Casi siempre usaremos esta última versión. Por ejemplo, para generar una secuencia de 10 números aleatorios entre 0.0 y 1.0 escribimos

for (int i = 0; i < 10; i++) {

System.out.println(rnd.nextDouble());;

}

Para crear una secuencia de 10 números aleatorios enteros comprendidos entre 0 y 9 ambos incluidos escribimos

int x;

for (int i = 0; i < 10; i++) {

x = (int)(rnd.nextDouble() \* 10.0);

System.out.println(x);

}

*(int)* transforma un número decimal **double** en entero **int** eliminando la parte decimal.

## 4. Comprobación de la uniformidad de los números aleatorios

Podemos comprobar la uniformidad de los números aleatorios generando una secuencia muy grande de números aleatorios enteros comprendidos entre 0 y 9 ambos inclusive. Contamos cuantos ceros aparecen en la secuencia, cuantos unos, ... cuantos nueves, y guardamos estos datos en los elementos de un array.

Primero creamos un array *ndigitos* de 10 de elementos que son enteros.

int[] ndigitos = new int[10];

Inicializamos los elementos del array a cero.

for (int i = 0; i < 10; i++) {

ndigitos[i] = 0;

}

Creamos una secuencia de 100000 números aleatorios enteros comprendidos entre 0 y 9 ambos inclusive (véase el apartado anterior)

for (long i=0; i < 100000L; i++) {

n = (int)(rnd.nextDouble() \* 10.0);

ndigitos[n]++;

}

Si *n* sale cero suma una unidad al contador de ceros, *ndigitos[0]*. Si *n* sale uno, suma una unidad al contador de unos, *ndigitos[1]*, y así sucesivamente.

Finalmente, se imprime el resultado, los números que guardan cada uno de los elementos del array *ndigitos*

for (int i = 0; i < 10; i++) {

System.out.println(i+": " + ndigitos[i]);

}

Observaremos en la consola que cada número 0, 1, 2...9 aparece aproximadamente 10000 veces.

## 5. Secuencias de números aleatorios

En la siguiente porción de código, se imprime dos secuencias de cinco números aleatorios uniformemente distribuídos entre [0, 1), separando los números de cada una de las secuencias por un carácter tabulador.

System.out.println("Primera secuencia");

for (int i = 0; i < 5; i++) {

System.out.print("\t"+rnd.nextDouble());

}

System.out.println("");

System.out.println("Segunda secuencia");

for (int i = 0; i < 5; i++) {

System.out.print("\t"+rnd.nextDouble());

}

System.out.println("");

Comprobaremos que los números que aparecen en las dos secuencias son distintos.

En la siguiente porción de código, se imprime dos secuencias iguales de números aleatorios uniformemente distribuidos entre [0, 1). Se establece la semilla de los números aleatorios con la función miembro *setSeed*.

**rnd.setSeed(3816);**

System.out.println("Primera secuencia");

for (int i = 0; i < 5; i++) {

System.out.print("\t"+rnd.nextDouble());

}

System.out.println("");

**rnd.setSeed(3816);**

System.out.println("Segunda secuencia");

for (int i = 0; i < 5; i++) {

System.out.print("\t"+rnd.nextDouble());

}

System.out.println("");

|  |
| --- |
| package azar;  import java.util.Random;  public class AzarApp {  public static void main (String[] args) {  int[] ndigitos = new int[10];  int n;    Random rnd = new Random();  // Inicializar el array  for (int i = 0; i < 10; i++) {  ndigitos[i] = 0;  }  // verificar que los números aleatorios están uniformente distribuídos  for (long i=0; i < 100000L; i++) {  // genera un número aleatorio entre 0 y 9  n = (int)(rnd.nextDouble() \* 10.0);  //Cuenta las veces que aparece un número  ndigitos[n]++;  }  // imprime los resultados  for (int i = 0; i < 10; i++) {  System.out.println(i+": " + ndigitos[i]);  }  //Dos secuencias de 5 número (distinta semilla)  System.out.println("Primera secuencia");  for (int i = 0; i < 5; i++) {  System.out.print("\t"+rnd.nextDouble());  }  System.out.println("");  System.out.println("Segunda secuencia");  for (int i = 0; i < 5; i++) {  System.out.print("\t"+rnd.nextDouble());  }  System.out.println("");  //Dos secuencias de 5 número (misma semilla)  rnd.setSeed(3816L);  System.out.println("Primera secuencia");  for (int i = 0; i < 5; i++) {  System.out.print("\t"+rnd.nextDouble());  }  System.out.println("");  rnd.setSeed(3816);  System.out.println("Segunda secuencia");  for (int i = 0; i < 5; i++) {  System.out.print("\t"+rnd.nextDouble());  }  System.out.println("");  }  } |

# UT3 Ejercicios

## Funciones 1

**1.-** Crea una subrutina en java a la que se le pase un número N que se le pedirá al usuario y muestre por pantalla la frase “Módulo ejecutándose” N veces.

**2.-**Muestra por pantalla los resultados de las 4 operaciones básicas entre 2 números que se le pedirán al usuario, realizando una subrutina para cada una de dichas operaciones. El resultado de la operación se imprime dentro de la subrutina. Y fuera

**3.-** Crea una función en java que reciba dos números reales y devuelva su suma.

**4.-** Crea una función en java que reciba un número entero y devuelva su factorial.

**5.-** Escribe una aplicación en java que le pida al usuario que escriba el nombre de una operación en java entre las siguientes: suma, resta, multiplicación y división. En caso de introducir una distinta se le comunicará el error y se pedirá de nuevo la operación.

Con una instrucción switch, diseña un menú que en función de cuál sea la operación elegida ejecutará una de las cuatro funciones que escribirás para llevar a cabo la operación (sumar(float, float), multiplicar(float, float), restar(float, float), dividir(float, float)). Cada función de las anteriores recibirá los dos operadores y devolverá al programa principal el resultado, desde donde se imprimirá. En el menú habrá una opción de salir para poder terminar el programa.

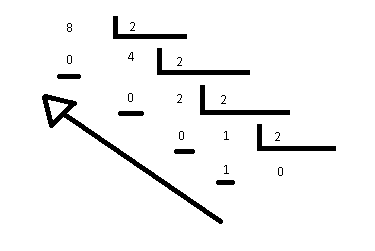
**6.-** Crea una aplicación que nos calcule el área de un círculo, cuadrado o un triángulo. Pediremos de qué figura queremos calcular su área (nos darán una de las tres) y según lo introducido pedirá los valores necesarios para calcular el área. Crea un método por cada figura para calcular cada área que devolverá un número real como valor del área. Muestra el resultado por pantalla. Añade un menú para mejorar el ejercicio. En el menú habrá una opción de salir para poder terminar el programa.

Datos para el cálculo del área de cada figura:

* **Circulo**: (radio^2)\*PI
* **Triangulo**: (base \* altura) / 2
* **Cuadrado**: lado \* lado

**7.-** Crea una aplicación que nos convierta un número en base decimal a binario. Esto lo realizará un método al que le pasaremos el número como parámetro, devolverá un String con el número convertido a binario. Para convertir un número decimal a binario, debemos dividir entre 2 el número y el resultado de esa división se divide entre 2 de nuevo hasta que no se pueda dividir más (la división da 0), el resto que obtengamos de cada división formara el número binario, de abajo a arriba.

Ejemplo: si introducimos un **8** nos deberá devolver **1000**



**8.-** Crea una aplicación que nos cuente el número de cifras de un número **entero positivo** (hay que controlarlo) pedido por teclado. Crea un método que realice esta acción, pasando el número por parámetro, devolverá el número de cifras.

**9.-** Crea un aplicación que nos convierta una cantidad de euros introducida por teclado a otra moneda, estas pueden ser dolares, yenes o libras. El método tendrá como parámetros, la cantidad de euros y la moneda a pasar que será una cadena. Haz dos versiones del programa:

El método no devolverá ningún valor, mostrará un mensaje indicando el cambio (void).

El método devolverá un valor que indica el cambio y este se mostrará en el programa principal.

El cambio de divisas es:

* 0.86 libras es un 1 €
* 1.28611 $ es un 1 €
* 129.852 yenes es un 1 €

## Funciones 2

Crea una biblioteca de funciones matemáticas que contenga las siguientes funciones. Recuerda que puedes usar unas dentro de otras si es necesario. Los números serán todos long. Todas las funciones devuelven al código llamante lo que se pide.

**1.-** **voltea**: Le da la vuelta a un número.

**2.-** **esCapicua**: Devuelve verdadero si el número que se pasa como parámetro es capicúa y falso en caso contrario.

**3.- digitos**: Cuenta el número de dígitos de un número.

**4.-** **digitoN**: Devuelve el dígito que está en la posición n de un número. Se empieza contando por el 0 y de izquierda a derecha.

**5.- posicionDeDigito**: Da la posición de la primera ocurrencia de un dígito dentro de un número. Si no se encuentra, devuelve -1.

**6.-** **quitaPorDetras**: Le quita a un número n dígitos por detrás (por la derecha).

**7.-** **quitaPorDelante**: Le quita a un número n dígitos por delante (por la izquierda).

**8.-** **pegaPorDetras**: Añade un número a otro número por detrás.

**9.-** **pegaPorDelante**: Añade un número a otro número por delante.

**10.-** **trozoDeNumero**: Toma como parámetros las posiciones inicial y final dentro de un número y devuelve el trozo correspondiente.

**11.-** **juntaNumeros**: Pega dos números para formar uno. Se usa para apoyar la solución de 8 y 9.

## Ejercicios Cadenas (Strings)

**1**.- Realiza un programa que lea 11 cadenas de caracteres, para mostrar al final la que contenga el mayor número de vocales, ya sean mayúsculas o minúsculas. Se consideran además de las 5 vocales, todas las vocales acentuadas y la ü.

**2.-** Realiza un programa que lea un número entero y creando una función llamada esCapicua devuelva si es capicúa o no lo es (true y false respectivamente). Para ello, conviértelo en una cadena de caracteres y utiliza métodos de cadenas/caracteres.

**3.-** Crea una función en Java que reciba una cadena de caracteres y devuelva esta cadena invertida.

**4.-** Crea una función en Java que reciba una palabra y la devuelva invertida con efecto espejo, esto es, se concatena a la palabra original su inversa, compartiendo la última letra, que hará de espejo, por lo que la palabra obtenida se lee igual hacia adelante que hacia atrás. Por ejemplo, al introducir “teclado” devolverá “tecladodalcet”  y al introducir “goma” devolverá “gomamog”.

**5.-** Crea una función en Java que reciba dos cadenas, la primera cadena será en la que se buscará y la segunda será la cadena buscada. La función devolverá cuántas veces aparece la segunda cadena en la primera.

**6.-** Crea una función en Java que reciba dos cadenas de caracteres y que devuelva la primera cadena, pero transformando en mayúsculas la parte que coincide con la segunda cadena introducida. Por ejemplo, si se introducen las cadenas “Este es mi amigo Juan” y “amigo”, devolverá “Este es mi AMIGO Juan”.

## Básicos Hoja 1

**1.** Hacer un programa que sume dos números leídos por teclado y escriba el resultado con el mayor detalle posible.

**2.** Haz un programa que convierta un número de galones en litros y escriba el resultado.

En un galón hay 3.7854 litros.

**3.** Haz un programa que solicite una medida en pies y realice la conversión a pulgadas, yardas, cm y metros. Ten en cuenta que un pie tiene 12 pulgadas, una pulgada equivale a 2.54 cm y 1 yarda son 0.9144 metros.

**4.** Un año es bisiesto si es múltiplo de 4, exceptuando los múltiplos de 100, que solo son bisiestos cuando son múltiplos además de 400.

Por ejemplo, el año 1900 no fue bisiesto, pero el año 2000 sí.

Haz un programa que dado un año A nos diga si es o no bisiesto.

**5.** Hacer un programa que dado un día D, un mes M y un año A, calcule cual es el día siguiente. Se debe tener en cuenta que en los años bisiestos Febrero tiene 29 días y en los no bisiestos 28.

**6.** Hacer un programa para resolver una ecuación de segundo grado, sin tener en cuenta las soluciones complejas.

**7.** Modificar el ejercicio 1 para sumar 10 números leídos por teclado.

**8.** Modificar el ejercicio 7 para que permita sumar N números. El valor de N se debe leer previamente por teclado.

**9.** Modifica el programa 2 para que imprima una tabla de conversión de 1 hasta 100 galones, cada 10 galones imprimirá una línea de salida en blanco.

**10.** Hacer un programa que permita escribir los primeros 100 números pares.

**11.** La sucesión de Fibonacci se define de la siguiente forma:

a1=1

a2=1

…

an=an-1+an-2 para n>2,

Es decir, los dos primeros son 1 y el resto cada uno es la suma de los dos anteriores, los primeros son: 1, 1, 2, 3, 5, 8, 13, 21,...

Haz un programa que calcule e imprima N términos de la sucesión, N se pedirá por teclado.

**12.** Haz un programa que calcule el total de una factura, partiendo de una lista de parejas importe, IVA pertenecientes a la misma factura. La lista finaliza cuando el importe sea 0. El IVA puede ser el 4%, el 10% o el 21%, en cualquier otro caso se rechazan importe e IVA y se deben introducir de nuevo.

Finalmente, hay que realizar un descuento en función de la suma de los importes. Dicho descuento es del 0% si es menor que 1000, del 5% si es menor que 10000 (y mayor o igual que 1000) y del 10% si es mayor o igual que 10000. El descuento se debe aplicar al importe final.

El programa debe imprimir el **importe total** sin descuento y el **importe final** al aplicar el descuento que corresponda al importe total.

**13.** Hacer un programa que lea N números (se piden al usuario), calcule la suma de los pares y el producto de los impares y escriba ambos resultados.

**14.** Hacer un diagrama de flujo para calcular el máximo común divisor de dos números enteros positivos N y M siguiendo el algoritmo de Euclides, que es el siguiente:

1. Se divide N por M, sea R el resto.

2. Si R=0, el máximo común divisor es M y se acaba.

3. Si no, se asigna a N el valor de M y a M el valor de R y volvemos al paso 1.

**15.** Hacer un programa para calcular el factorial de N (N!=1·2·3·...·N).

**16.** Hacer un programa para sumar los N primeros términos de una progresión geométrica de primer término A y razón R (dados por teclado). Se debe realizar la suma sin emplear la fórmula que existe para ello. Muestra también los términos de la serie.

***Ejemplo de ejecución:***

Introducir número de términos

6

Introducir el primer término

5

Introducir la razón

3

***Salida***:

5 15 45 135 405 1215

La suma de los términos de la serie es 1820

**17.** Se dice que un número N es perfecto si la suma de sus divisores, excluido el propio número es N. Por ejemplo, 28 es perfecto, pues sus divisores son: 1, 2, 4, 7 y 14 y su suma es 1+2+4+7+14=28.

Haz un programa que dado un número N nos diga si es o no perfecto. Cambia el programa para que siga pidiendo números mientras el número introducido sea distinto de cero, que será la señal para parar el programa.

**18.** Hacer un programa que sea capaz de calcular el impuesto sobre la renta y lo escriba por pantalla hasta que se introduzca un salario igual a cero. El impuesto de la renta es el 15% del salario anual de cada persona, al que previamente se debe realizar una deducción en función del número de hijos, que es del 0% si tiene 0, del 5% si tiene 1 o 2 y del 15% si tiene más de 2.

Tanto el salario como el número de hijos se pedirán por teclado.

**19.** Hacer un programa que lea por teclado un numero N e imprima un triángulo rectángulo, de N filas. EJ: N=5, se pintará lo siguiente:

\*

\* \*

\* \* \*

\* \* \* \*

\* \* \* \* \*

**20.** Hacer un programa en el que se pida por teclado un número mayor que 2 (el programa controlará que cumpla esto), y lo imprima de todas las formas posibles como producto de dos factores (no se tiene en cuenta la multiplicación por 1).

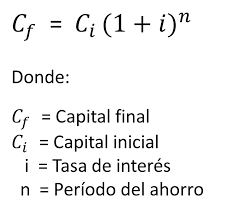
**Ej:** Con el número 36, tendría que visualizarse: 18\*2, 12\*3, 9\*4, 6\*6, 3\*12, 4\*9, 2\*18

## Básicos Hoja 2

**1.** Escribir un programa que imprima cada uno de los términos de la serie 2, 5, 7, 10, 12, 15, 17,..., 1800. Además calcule e imprima la suma de los términos.

**2.** Leer un capital C y averiguar e imprimir en cuantos meses se duplica, si lo colocamos a un interés compuesto del 5% mensual.

La fórmula a aplicar es:



**3.** En 1980 la ciudad A tenía 3.5 millones de habitantes y una tasa de crecimiento del 7% anual; y la ciudad B tenía 5 millones de habitantes y una tasa de crecimiento del 5% anual.

Si el crecimiento poblacional se mantiene constante en las dos ciudades, hacer un programa que calcule e imprima en qué año la población de la ciudad A es mayor que la de la ciudad B.

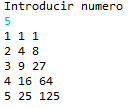
**4.** Haz un programa que lea un capital y calcule e imprima en cuantos meses se triplica si se coloca a un interés del 6% mensual.

**5.** Haz un programa que lea un número entero N y calcule el resultado de la siguiente serie: 1+1/2+1/3+1/4+1/5+...+1/N.

**6.** Haz un programa que lea un número entero N y calcule el resultado de la siguiente serie: 1-1/2+1/3-1/4+1/5-...+1/N.

**7.** Haz un programa que calcule la suma de los números pares comprendidos entre 10 y 50.

**8.** Haz un programa para imprimir una tabla de tres columnas y N filas con los cuadrados y los cubos de los N primeros números. Pide N al usuario.



**9.** Haz un programa que pida 2 números por teclado y calcule su producto mediante sumas sucesivas. Imprimir su resultado.

**10.** Hacer un programa que pida 2 números por teclado y calcule su división mediante restas sucesivas. Imprimir su resultado. Divide siempre el más grande entre el más pequeño.

***Ejemplo***: 1324 entre 312.

1324 - 312 = 1012 contamos una vez y seguimos porque 1012 >= 312

1012 - 312 = 700 contamos 2 veces y continuamos porque 700 >= 312

700 - 312 = 388 contamos 3 veces y continuamos porque 388 >= 312

388 - 312 = 76 contamos 4 veces y paramos porque 76 < 312

Luego la división es 4 y el resto 76.

**11.** Haz un programa que lea un número entero y genere y escriba el número resultante de invertir sus cifras.

**12.** Haz un programa que indique si un número entero N es primo o no.   
//Ejercicio 11 Estructuras repetitivas Pseint

**13.** Haz un programa que imprima el triángulo de Floyd hasta un valor dado. El triángulo contiene los números naturales correlativos, uno en la primera línea, dos en la segunda, etc.; es decir, en la fila n-ésima aparecen n valores.

Ejemplo:

1

1 2

1 2 3

1 2 3 4

1 2 3 4 5

**14.** Escribe un programa para calcular A elevado a B, siendo A un número real cualquiera y B un valor entero positivo o nulo (sin emplear métodos ya hechos, claro). Si el exponente no es positivo o nulo se sigue pidiendo hasta que lo sea.

//Ejercicio 9 Estructuras repetitivas Pseint

## Básicos Hoja 3

**1.** Haz un programa en Java para jugar contra el ordenador a adivinar un número, generado aleatoriamente, entre 1 y 200. El usuario debe introducir un número por teclado y el programa le dirá mediante los símbolos '<' o '>', si el número introducido es menor o mayor que el generado por el ordenador.

Finalmente, se mostrará un mensaje informando de cuantos intentos se han necesitado para adivinar el número y si no se adivina se mostrará un mensaje diciendo que ha perdido.

El número máximo de intentos se pedirá por teclado.

**Explicación:** Para generar el número aleatorio poner:

int numAleatorio = (int) (Math.random()\*200+1);

**2.** Haz un programa en Java que muestre si dos números son o no amigos. Los números se pedirán por teclado.

**Explicación:** Se dice que dos números son amigos si la suma de los divisores del primero (excluido el propio número) es el segundo número y viceversa.

Ej: Los números 220 y 284 son amigos

220 = 1+2+4+5+10+11+20+22+44+55+110 = 284

284 = 1+2+4+71+142 = 220

**3.** Haz un programa que muestre un contador con 3 dígitos. Mostrará los números del 0-0-0 al 9-9-9, con la particularidad que cada vez que aparezca un 3 lo sustituya por una E.

**4.**  Para obtener el número del tarot de una persona, hay que sumar los números de su fecha de nacimiento y reducirlo a un solo dígito.

Ej: 1 de Julio de 1990 →1+7+1990 = 1998 → 1+9+9+8 = 27 → 2+7=9 El número del tarot es el 9.

Haz un programa que lea la fecha de nacimiento por teclado y escriba el número del tarot. La fecha estará formada por 3 números enteros, el día, el mes y el año (4 dígitos).

## Básicos Hoja 4 *– Nivel Examen*

**1.** Haz un programa en Java que pida números hasta que se teclee uno negativo, y mostrar cuántos números se han introducido.

**2.** Crear un programa para calcular el salario semanal de los empleados a los que se les paga 15 euros por hora si estas no superan las 35 horas. Cada hora por encima de 35 se considerará extra y se paga a 22 €. El programa pide las horas del trabajador y devuelve el salario que se le debe pagar.

Además, el programa debe preguntar si deseamos calcular otro salario, si es así el programa se vuelve a repetir.

**3.** Haz un programa que pida un número n, y diga cuántos y cuáles son los números primos que hay entre 1 y n.

**4.** Haz un programa que vaya pidiendo números, hasta introducir uno negativo, y diga cuál es el mayor número introducido y cuantas veces se repite.

**5.** Haz un programa que pida un número, entre 0 y 10, y escriba un triángulo invertido con dichos números.

Ej: Si n=10, quedaría

0,1,2,3,4,5,6,7,8,9

1,2,3,4,5,6,7,8,9

2,3,4,5,6,7,8,9

3,4,5,6,7,8,9

4,5,6,7,8,9

5,6,7,8,9

6,7,8,9

7,8,9

8,9

9

**6.** Un alumno desea saber que nota necesita en el tercer examen para aprobar una evaluación. El promedio de la evaluación se calcula con la siguiente formula.

NC=(E1+E2+E3)/3

NF =NC\*0.7+NL\*0.3

Donde NC es el promedio de los exámenes, NL el promedio de laboratorio y NF la nota final.

Escribe un programa que pregunte al usuario las notas de los dos primeros exámenes y la nota promedio de laboratorio, y muestre la nota que necesita el alumno en el tercer examen para aprobar la evaluación con nota final 6.0.

*Ejemplo:*

Ingrese nota examen 1: 4.5

Ingrese nota examen 2: 5.5

Ingrese nota laboratorio: 6.5

Necesita nota 7.4 en el examen 3

**7.** La secuencia de Collatz de un número entero se construye de la siguiente forma:

* si el número es par, se divide por dos;
* si es impar, se multiplica tres y se le suma uno;
* la sucesión termina al llegar a uno.

La conjetura de Collatz afirma que, al partir desde cualquier número, la secuencia siempre llegará a 1. A pesar de ser una afirmación a simple vista muy simple, no se ha podido demostrar si es cierta o no.

Usando computadores, se ha verificado que la sucesión efectivamente llega a 1 partiendo desde cualquier número natural menor que 258.

a) Hacer un programa que muestre la secuencia de Collatz de un número entero, que se pedirá por teclado.

Ej:

n: **18**

18 9 28 14 7 22 11 34 17 52 26 13 40 20 10 5 16 8 4 2 1

n: **19**

19 58 29 88 44 22 11 34 17 52 26 13 40 20 10 5 16 8 4 2 1

n: **20**

20 10 5 16 8 4 2 1

b) Hacer un programa que grafique los largos de las secuencias de Collatz (número de elementos que hay que generar hasta que se llega al 1) de los números enteros positivos menores que el ingresado por el usuario:

n: **20**

1 \*

2 \*\*

3 \*\*\*\*\*\*\*\*

4 \*\*\*

5 \*\*\*\*\*\*

6 \*\*\*\*\*\*\*\*\*

7 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

8 \*\*\*\*

9 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

10 \*\*\*\*\*\*\*

11 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

12 \*\*\*\*\*\*\*\*\*\*

13 \*\*\*\*\*\*\*\*\*\*

14 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

15 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

16 \*\*\*\*\*

17 \*\*\*\*\*\*\*\*\*\*\*\*\*

18 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

19 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

20 \*\*\*\*\*\*\*\*

# UT4 Programación Orientada a Objetos

## **1. Programación Orientada a Objeto (POO)**

La programación Orientada a Objetos es una metodología que basa la estructura de los programas en torno a los objetos.

Los lenguajes de POO ofrecen medios y herramientas para describir los objetos manipulados por un programa. Más que describir cada objeto individualmente, estos lenguajes proveen una construcción (Clase) que describe a un conjunto de objetos que poseen las mismas propiedades.

## **2. Clase**

Una clase es un tipo definido por el usuario (plantilla) que describe los atributos y los métodos de los objetos que se crearán a partir de ella.

El estado de un objeto viene determinado por sus **atributos,** y su comportamiento está definido por sus **métodos.**

Dentro de las clases también se encuentran los **constructores** que permiten inicializar un objeto.

Los atributos y los métodos se denominan en general **miembros de la clase**.

La definición de una clase consta de dos partes: el nombre de la clase precedido por la palabra reservada **class** y el cuerpo de la clase entre llaves. La sintaxis queda:

|  |
| --- |
| class nombre-clase { |
| cuerpo de la clase |
| } |

Dentro del cuerpo de la clase se puede encontrar los atributos y métodos.

**Ej:**

class Circunferencia {

private double x, y, radio;

public Circunferencia(){ }

public Circunferencia(double cx, double cy, double r){

x=cx;

y=cy;

radio=r;

}

public void ponRadio(double r){

radio=r;

}

public double longitud(){

//No hay que pasarle el radio, coge el radio de la circunf

return 2\*Math.PI\*radio;

}

}

En el ejemplo, se define la **clase Circunferencia**, que puede ser usada dentro de un programa de la misma manera que cualquier otro tipo. Un objeto de esta clase tiene tres atributos (coordenadas del centro y valor del radio), dos constructores y dos métodos.

**Los constructores se distinguen fácilmente porque tienen el mismo nombre que la clase.**

Los atributos se declaran de la misma manera que cualquier variable. En una clase, cada atributo debe tener un nombre único.

Siguiendo las recomendaciones de la Programación Orientada a Objetos, cada clase se debe implementar en un fichero .java, de esta manera es más sencillo modificar la clase.

El fichero que contiene la clase debe llevar el nombre de la clase pública con la extensión .java. ***A veces tendremos varias clases en un .java, pero sólo una puede ser pública***

## 3. Métodos

Los métodos forman lo que se denomina interfaz de los objetos, definen las operaciones que se pueden realizar con los atributos. Desde el punto de vista de la Programación Orientada a Objetos, el conjunto de métodos se corresponde con el conjunto de mensajes a los que los objetos de una clase pueden responder.

**Los métodos implementan la funcionalidad asociada al objeto**. Los métodos son el equivalente a las funciones en Programación Estructurada. Se diferencian de ellos en que es posible **acceder** a las **variables de la clase de forma implícita** (atributos).

Cuando se desea realizar una acción sobre un objeto, se dice que se le manda un mensaje invocando a un método que realizará la acción.

Los métodos permiten al programador modularizar sus programas.

Todas las variables declaradas en las definiciones de métodos son variables locales, solo se conocen en el método que las define. Casi todos los métodos tienen una lista de parámetros que permiten comunicar información.

La sintaxis para definir un método es:

|  |
| --- |
| **<modificador-acceso> <modificador > tipoR nombre-método(<parámetros>){** |
| **<cuerpodelmétodo>** |
| **}** |

<modificador-acceso> indica cómo es el acceso a dicho método: public, private, protected y sin modificador.

<modificador > indica si es estática o no.

<tipoR> es el tipo de dato que retorna el método. Es obligatorio indicar un tipo. Para los métodos que no devuelven nada se utiliza la palabra reservada *void*.

Nombre-método es cómo el programador quiere llamar a su método y <parámetros> son los datos que se van a enviar al método para trabajar con ellos, no son obligatorios.

Para devolver el valor se utiliza el operador **return**. Una vez que se ejecute return, el control se devuelve al código que lo llamó y la ejecución del método termina. Los métodos que no lleven return, tendrán tipo void.

**Ej:**

int suma(int x, int y) {

return x+y;

}

En este ejemplo el método recibe 2 parámetros de tipo int, realiza su suma y devuelve esta suma, que es un valor de tipo int.

void imprimir(){

System.out.println(“Este método no devuelve nada y tampoco

recibe parámetros”);

}

En este ejemplo el método no devuelve ningún valor, ni recibe parámetros, aun así es necesario poner los paréntesis vacíos.

### 3.1 Métodos Estáticos o de Clase – hay otro word

Van precedidos del modificador **static**.

Para invocar a un **método** estático no se necesita crear un objeto de la clase en la que se define. Si **se invoca desde la clase** en la que se encuentra definido, basta con escribir su nombre.

Si **se le invoca desde una clase distinta**, debe anteponerse a su nombre, el de la clase en que se encuentra seguido del operador punto (.) La sintaxis es:

< NombreClase> .metodoEstatico( );

Suelen emplearse para realizar operaciones comunes a todos los objetos de la clase. No afectan a los estados de los objetos.

Por ejemplo, si se necesita un método para contabilizar el número de objetos creados de una clase, se define estático ya que su función, aumentar el valor de una variable entera, se realiza independientemente del objeto empleado para invocarle. Esa variable entera, sería única para todos los objetos.

Otra razón por la que tendríamos que usar métodos estáticos es si se utilizan fuera del contexto de cualquier instancia

Los métodos estáticos sólo pueden llamar a otros métodos static directamente, y no se pueden referir a **this** o **super** de ninguna manera.

No es conveniente usar muchos métodos estáticos, pues si bien se aumenta la rapidez de ejecución, se pierde flexibilidad, no se hace un uso efectivo de la memoria y no se trabaja según los principios de la POO.

Las **variables** también se pueden declarar como static, y es equivalente a declararlas como variables globales, que son accesibles desde cualquier fragmento de código.

Se puede declarar un bloque static que se ejecuta una sola vez si se necesitan realizar cálculos para inicializar las variables static.

Ej:

class Estatica {

static int a=3,b;

static{

System.out.println(“Bloque static inicializado“);

b=a\*4;

} **// Si no es static no compilará**

static void metodo(int x){

System.out.println(“x= “+x);

System.out.println(“a= “+a);

System.out.println(“b= “+b);

}

public static void main(String[] args){

metodo(42);

}

}

En el ejemplo la clase que tiene dos variables static, un bloque de inicialización static y un método static. La salida del programa es:

Bloque static inicializado:

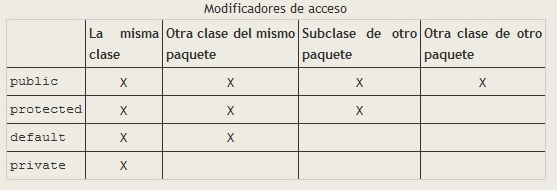
x = 42

a = 3

b = 12

### 3.2 Modificadores de acceso

*Se profundizará sobre los modificadores de acceso cuando veamos Herencia y Paquetes en Java*. De momento, pondremos que un atributo/método es privado (**private**) si solo lo utilizamos dentro de su clase, si lo queremos usar fuera lo pondremos público (**public**).



## 4. Objetos

Es una entidad (tangible o intangible) que posee características y acciones que realiza por sí solo o interactuando con otros objetos.

Un **objeto** es una entidad caracterizada por sus atributos propios y cuyo comportamiento está determinado por las acciones o funciones que pueden modificarlo (métodos), así como también las acciones que requiere de otros objetos.

Un objeto constituye una unidad que **oculta** tanto datos como la descripción de su manipulación. Puede ser definido como una **encapsulación** y una **abstracción**: una encapsulación de atributos y servicios, y una abstracción del mundo real.

**Para el contexto del Enfoque Orientado a Objetos (EOO) un objeto es una entidad que encapsula datos (atributos) y acciones o funciones que los manejan (métodos). También para el EOO un objeto se define como una instancia o particularización de una clase.**

Los objetos de interés durante el desarrollo de software no solo son tomados de la vida real (objetos visibles o tangibles), también pueden ser abstractos. En general, son entidades que juegan un rol bien definido en el dominio del problema. Un libro, una persona, un coche, un polígono son algunos **ejemplos** de objeto.

Cada objeto puede ser considerado como un proveedor de servicios utilizados por otros objetos que son sus clientes. Cada objeto puede ser a la vez proveedor y cliente. De ahí que un programa pueda ser visto como un conjunto de relaciones entre proveedores clientes. Los servicios ofrecidos por los objetos son de dos tipos:

1.- Los datos, que llamamos **atributos**.

2.- Las acciones o funciones, que llamamos **métodos**.

Un objeto consta de una estructura interna (los atributos) y de una interfaz que permite acceder y manipular dicha estructura (los métodos). Para **construir un objeto de una clase** cualquiera hay que llamar a un método de iniciación, **el constructor**. Para ello, se utiliza el operador **new**. La sintaxis es la siguiente:

|  |
| --- |
| **Nombre-clase nombre-objeto = new Nombre-clase (<valores>);** |

Donde **Nombre-clase** es el nombre de la clase de la cual se quiere crear el objeto, **nombre-objeto** es el nombre que el programador quiere dar a ese objeto y <valores> son los valores con los que se inicializa el objeto. Dichos valores son opcionales y aparecen en el constructor.

**Ej:** Circunferencia circ = new Circunferencia();

Se crea el objeto circ, de la clase Circunferencia, con los valores predeterminados.

Cuando se crea un objeto, Java hace lo siguiente:

* **Asignar memoria** al objeto por medio del operador ***new***.
* **Llamar al constructor** de la clase para inicializar los atributos de ese objeto con los valores iniciales o con los valores predeterminados por el sistema: los atributos numéricos a cero, los alfanuméricos a nulos y las referencias a objetos a null.

Si no hay suficiente memoria para ubicar el objeto, el operador new lanza una excepción OutOfMemoryError.

Para acceder desde un método de una clase a un miembro de un objeto de otra clase diferente se utiliza la sintaxis: **objeto.miembro**. Cuando el miembro accedido es un método se entiende que el objeto ha recibido un mensaje, el especificado por el nombre del método y responde ejecutando ese método. En este caso, después del nombre del método siempre se pone paréntesis, aunque no haya parámetros.

***Características Generales:***

* Un objeto posee **estados**. El estado de un objeto está determinado por los valores que poseen sus atributos en un momento dado.
* Un objeto tiene un **conjunto de métodos**. El comportamiento general de los objetos dentro de un sistema se describe o representa mediante sus operaciones o métodos. Los métodos se utilizarán para obtener o cambiar el estado de los objetos, así como para proporcionar un medio de **comunicación** entre objetos.
* Un objeto tiene un **conjunto de atributos**. Los atributos de un objeto contienen valores que determinan el estado del objeto durante su tiempo de vida. Se implementan con variables, constantes y estructuras de datos (similares a los campos de un registro).

### 4.1 Asignación

Una vez el objeto está creado, se tiene una **referencia a ese objeto en la variable** donde se asigna. Si se realiza la asignación de un objeto a otro los dos harán referencia al mismo objeto.

**Ej**:

Circunferencia c1 = new Circunferencia(0, 0, 15);

Circunferencia c2 = c1;

c1.ponRadio(25);

System.out.println(c2.radio); // Es el mismo objeto

En el ejemplo, la variable c1 apunta a un objeto de la clase Circunferencia. Debido a la asignación, la variable c2 apunta al mismo objeto. Después se modifica el valor del radio del objeto apuntado por c1.

Y por último, se visualiza c2 que será el valor 25.

### 4.2 Igualdad

Creamos dos objetos iguales (con los mismos valores de sus variables miembro), al preguntar si las variables que apuntan a esos objetos son iguales nos devolverá false, pues aunque tengan el mismo valor no son el mismo objeto.

**Ej:**

Circunferencia c1 = new Circunferencia(0,0,15);

Circunferencia c2 = new Circunferencia(0,0,15);

if (c1==c2) **/\* Esto es falso\*/**

### 4.3 Constructor

Un Constructor es un método especial en Java empleado para inicializar valores en instancias de objetos. A través de este tipo de métodos es posible generar distintos tipos de instancias para la clase en cuestión.

Los métodos constructores tienen las siguientes **características**:

* Se llaman igual que la clase.
* No devuelven nada, ni siquiera void.
* Puede haber varios constructores, que deberán distinguirse por el tipo de valores que reciban.
* De entre los que existan, solo uno se ejecutará al crear el objeto.
* El código de un constructor, generalmente, suele ser inicializaciones de variables y objetos, para conseguir que el objeto sea creado con dichos valores iniciales.
* Si no se define ningún constructor el compilador crea uno **por defecto** sin parámetros que, al ejecutarse, inicializa el valor de cada atributo de la nueva instancia a 0, false o null, dependiendo de si el atributo es numérico, alfanumérico o una referencia a otro objeto respectivamente, pero dicho constructor desaparece en el mismo momento en que se defina otro constructor, por lo que **si se quiere tener el de por defecto habrá que definirlo**.

La **declaración de constructores** sigue la siguiente sintaxis:

|  |
| --- |
| <modificador deVisibilidad> NombredelaClase ( <argumentos> ) { |
| < declaraciones> |
| } |

Donde <modificadorvisibilidad> es el tipo de modificador de acceso del constructor, <Nombredelaclase> es el nombre del constructor y debe coincidir con el de la clase y <argumentos> son las variables que recibe el constructor y contienen los valores con los que se inicializaran los atributos.

**Ej:**

class Circunferencia{

private double x, y, radio;

public Circunferencia(){ }

public Circunferencia(double cx, double cy, double r){

x=cx; y=cy; radio=r;

}

}

En este ejemplo existen 2 constructores, el primero que es el de por defecto y el segundo que inicializa los atributos x, y y radio con los valores cx, cy y r, respectivamente.

### 4.4 Referencia this

Java incluye un valor de referencia especial llamado **this**, que se utiliza dentro de cualquier método para referirse al objeto actual. **El valor this se refiere al objeto sobre el que ha sido llamado el método actual.**

Se puede utilizar this siempre que se requiera una referencia a un objeto del tipo de una clase actual. Si hay dos objetos que utilicen el mismo código, seleccionados a través de otras instancias, cada uno tiene su propio valor único de this.

Normalmente, dentro del cuerpo de un método de un objeto se puede referir directamente a las variables miembros del objeto. Sin embargo, algunas veces no se querrá tener ambigüedad sobre el nombre de la variable miembro y uno de los argumentos del método que tengan el mismo nombre y usaremos la referencia this.

**Ej:**

class Circunferencia {

private double x, y, radio;

public Circunferencia(double x, double y, double radio){

this.x=x;

this.y=y;

this.radio=radio;

}

}

En el ejemplo, el constructor de la clase inicializa las variables con los argumentos pasados al constructor. Se debe utilizar this en este constructor para evitar la ambigüedad entre los argumentos y los atributos miembro.

También se puede utilizar this para llamar a uno de los métodos del objeto actual. Esto solo es necesario si existe alguna ambigüedad con el nombre del método y se utiliza para intentar hacer el código más claro.

# UT4 Objetos y clases: Introducción

## 1. El concepto de clase

Si pensamos en las **clases de objetos** que vemos, tenemos coches, bares, semáforos, personas…

Si simplificamos esta definición de clases de objetos por simplemente clases:

¿Qué **clases** vemos? Coches, bares, semáforos, personas…

Y ya estaremos usando terminología de la **programación orientada a objetos**:

Tenemos la **clase** Coche, la clase Bar, la clase Semáforo, la clase Persona…

No nos importa si esa persona es Marta, está en el bar “Casa Manolo” (Bar Casa Manolo sería un objeto de clase Bar), si tiene un Seat Panda o si el semáforo está en rojo. Estamos pensando en términos **clasificatorios**.

Vamos a pensar ahora en cada una de estas clases:

¿Cómo son los coches? ¿Qué **características** pueden diferenciar uno de otro? Por ejemplo, podríamos indicar la marca, el modelo, el color, la cilindrada, etc.

A las características las vamos a denominar **atributos**, término muy utilizado en la **programación orientada a objetos**.

Atributo: variable dentro del objeto

|  |
| --- |
| ¿Qué atributos tiene la clase Bar? Todo bar tiene un nombre, una ubicación, un estado (si está abierto o cerrado), una lista de precios,… |
| Como atributos de la clase Semáforo podríamos indicar su ubicación y estado (rojo, verde o ámbar). |
| La clase Persona podría definir como atributos el nombre, sexo, edad, estado civil, altura, etc. |

A continuación vamos a pensar en el **comportamiento** de estas clases. Nos preguntaremos qué cosas hacen, qué tipo de acciones pueden realizar...

|  |
| --- |
| Un coche puede arrancar, detenerse, girar a la izquierda, acelerar, frenar, encender sus luces.  Método: funciones que cambian el estado, a veces cambian atributos |
| Un semáforo puede cambiar de estado. |
| Un bar puede abrir, cerrar, servirte una cerveza, cobrarla, modificar la lista de precios.  Una persona puede hablar, dormir, conducir un coche, tomarse una cerveza en un bar. |

En terminología de la **programación orientada a objetos**, a estas funciones que determinan el comportamiento de una clase se las conoce como **métodos**.

|  |
| --- |
| Sabemos lo que es una **clase** y que está compuesta de **atributos** y **métodos**. Nuestra labor ahora consistirá en **modelar** en Java estas clases. |

Modelar no es otra cosa sino crear una **abstracción** que represente de algún modo una determinada realidad.

Para crear en Java la clase Coche procederíamos del siguiente modo:

|  |
| --- |
| class Coche |
| { |
| } |

Entre las llaves introduciremos los atributos y métodos de que consta la clase. Usamos para ello la palabra reservada **class**.

El nombre de la clase Coche lo escribimos con la primera letra en mayúsculas, pues es de común acuerdo entre los programadores en Java que los nombres de clases empiecen así.

Introduzcamos algunos atributos:

|  |
| --- |
| class Coche |
| { |
| String marca; |
| String modelo; |
| String color; |
| int numeroDePuertas; |
| int cuentaKilometros; |
| int velocidad; |
| boolean arrancado; |
| } |

**Declarar atributos** es algo similar a declarar una variable normal. El nombre del atributo se precede por su tipo. Así, en el ejemplo, tenemos tres atributos de tipo String que contendrán cadenas de caracteres; otros tres de tipo int para almacenar valores enteros; finalmente, el atributo arrancado, que utilizaremos para indicar si el coche está en marcha o no, es de tipo boolean, admitiendo como posibles valores true o false.

Los tipos int y boolean forman parte de los tipos básicos de Java, conocidos como tipos **primitivos**. Los presentaremos formalmente a su debido momento; por ahora es suficiente con que conozcamos su existencia y cómo los utilizamos.

El tipo String, que escribimos con la primera letra en mayúsculas (lo que debería darte una pista), no es más que otra clase, como lo son las clases Coche y Persona. Es una clase muy importante en Java que encapsula un buen conjunto de métodos para trabajar con cadenas de caracteres.

El concepto importante que debes entender es que los atributos no necesariamente son siempre de tipos básicos, sino que también pueden ser de cualquier clase, incluso de una propia que nosotros mismos hayamos creado.

Por ejemplo, podríamos definir un nuevo atributo de la clase Coche, llamado conductor, en el que figure la persona (de tipo Persona) que lo conduce:

|  |
| --- |
| Persona conductor; |
|  |

Fíjate también en la convención utilizada para nombres de variables compuestos de varias palabras. Se escriben todas juntas, pero iniciando cada palabra en mayúsculas, a excepción de la primera. Esto forma parte también del estilo de escritura Java.

Ten en cuenta que Java distingue mayúsculas de minúsculas. No es lo mismo numeroDePuertas que numerodepuertas.

Definamos ahora los métodos de la clase Coche:

|  |
| --- |
| class Coche |
| { |
| String marca; |
| String modelo; |
| String color; |
| int numeroDePuertas; |
| int cuentaKilometros; |
| int velocidad; |
| boolean arrancado; |
|  |
| void arrancar() |
| { |
| } |
|  |
| void parar() |
| { |
| } |
|  |
| void acelerar() |
| { |
| } |
|  |
| void frenar() |
| { |
| } |
|  |
| void pitar() |
| { |
| } |
|  |
| int consultarCuentaKilometros() |
| { |
| } |
| } |

El nombre de método viene seguido por un par de paréntesis, pues en ocasiones los métodos podrán recibir argumentos que luego se utilizarán en el cuerpo del método. Aunque el método no requiera argumentos los paréntesis son absolutamente necesarios.

Por otro lado, el nombre del método va precedido por el tipo del valor que devuelve. Hay que indicarlo incluso si el método no devuelve explícitamente ningún valor. Ese caso se indica con el tipo **void**.

Entre el par de llaves { } introduciremos el cuerpo del método, las instrucciones que indican su operatividad.

Escribamos algo de código básico en cada uno de ellos:

|  |
| --- |
| class Coche |
| { |
| String marca; |
| String modelo; |
| String color; |
| int numeroDePuertas; |
| int cuentaKilometros; |
| int velocidad; |
| boolean arrancado; |
|  |
| void arrancar() |
| { |
| arrancado = true; |
| } |
|  |
| void parar() |
| { |
| arrancado = false; |
| } |
|  |
| void acelerar() |
| { |
| velocidad = velocidad + 1; |
| } |
|  |
| void frenar() |
| { |
| velocidad = velocidad - 1; |
| } |
|  |
| void pitar() |
| { |
| System.out.println("Piiiiiiiiiiiiiiiiii"); |
| } |
|  |
| int consultarCuentaKilometros() |
| { |
| return cuentaKilometros; |
| } |
| } |
|  |

Los métodos arrancar() y parar() establecen el atributo booleano arrancado a true y false, respectivamente. Los métodos acelerar() y frenar() incrementan y decrementan en una unidad, respectivamente, el atributo velocidad. El método pitar() imprime en consola una cadena de caracteres. El método consultarCuentaKilometros() devuelve **a quien lo invoca** (fíjate en el return) lo que contiene el atributo cuentaKilometros. Observa que es el único que devuelve explícitamente un valor (de tipo entero); los restantes, aunque algunos modifican los atributos de la misma clase, no devuelven ningún valor a quien los llama.

Podríamos modelar la clase Persona del siguiente modo:

|  |
| --- |
| class Persona |
| { |
| char sexo; |
| String nombre; |
| int edad; |
| Coche coche;   // El coche que conduce esa persona |
|  |
| void saludar() |
| { |
| System.out.println("Hola, me llamo " + nombre); |
| } |
|  |
| void dormir() |
| { |
| System.out.println("Zzzzzzzzzzz"); |
| } |
|  |
| int obtenerEdad() |
| { |
| return edad; |
| } |
| } |

Fíjate que uno de los atributos es precisamente de la clase Coche que acabamos de definir:

|  |
| --- |
| Coche coche |
|  |

## 2. El concepto de objeto

Una vez hemos entendido el concepto de **Clase** en el paradigma de la Programación Orientada a Objetos, es momento de matizar qué es un **Objeto** en su sentido más práctico.

*Una clase es una plantilla a partir de la cual vamos a crear objetos.*

Una **Clase**, como hemos visto, no es más que una especificación que define las características y el comportamiento de un determinado tipo de objetos. Piensa en ella como si se tratara de una plantilla, molde o esquema a partir del cual podremos construir **objetos concretos**.

Consideremos la clase Coche que definimos en el artículo anterior:

Con esta plantilla, vamos a crear coches **concretos** de la marca, modelo y color que nos apetezca. Cada uno que creemos será un **objeto**, o **instancia**, de la clase Coche.

Fíjate, en esta terminología, en la equivalencia de los términos objeto e instancia para referirnos, ahora sí, a **entidades concretas (objetos)** de una determinada clase.

Fabriquemos, entonces, nuestro primer coche…

Cada objeto, como todas las variables en Java, ha de ser **declarado** antes de ser utilizado:

|  |
| --- |
| Coche coche1 |
|  |

Con esta instrucción declaramos la variable coche1 de tipo Coche. En cuanto creemos, con el comando que escribiremos a continuación, el objeto concreto, coche1 contendrá una **referencia** a ese objeto, es decir, almacenará la dirección de memoria en la que realmente se halla el objeto propiamente dicho. **Esto es muy importante**: coche1 no contendrá el objeto en sí, sino una dirección de memoria que apunta a él.

Materialicemos nuestro primer coche del siguiente modo:

|  |
| --- |
| coche1 = new Coche() |
|  |

La palabra reservada **new** se emplea para **crear nuevos objetos**, instancias de una determinada clase que indicamos a continuación, seguida de un par de paréntesis.

Veremos esto a su debido momento, pero por ahora tenemos bastante con retener que se está invocando a un método especial que tienen todas las clases, que sirve para **construir** el objeto en cuestión facilitándole sus valores iniciales. A este método se le conoce como **constructor** de la clase.

Podríamos haber realizado la declaración y la creación en una sola instrucción:

|  |
| --- |
| Coche coche1 = new Coche() |

Ahora, vamos a decir de qué marca y modelo es:

|  |
| --- |
| coche1.marca = "Seat" |
| coche1.modelo = "Panda" |

*(entramos dentro del objeto coche1 y cambiamos los atributos)*

Consulta el cuadro de arriba con la definición de la clase y observa que tanto marca como modelo son dos atributos de tipo String, por lo que referencian cadenas de caracteres que escribimos entre comillas.

Observa con cuidado la **notación punto**. Separamos el nombre de la variable que referencia al objeto del atributo empleando un **punto como separador**.

Pintemos de azul nuestro vehículo:

|  |
| --- |
| coche1.color = "Azul" |

Además tiene tres puertas, el cuenta kilómetros indica 250.000 y su velocímetro refleja 0 Km/h. Valores correspondientes a los atributos numeroDePuertas, cuentaKilometros y velocidad, respectivamente, todos de tipo entero.

|  |
| --- |
| coche1.numeroDePuertas = 3 |
|  |
| coche1.cuentaKilometros = 250000 |
|  |
| coche1.velocidad = 0 |
|  |

Su motor está detenido, hecho que representamos a través de la variable booleana arrancado:

|  |
| --- |
| coche1.arrancado = false |

Ahora vamos a experimentar con los métodos. Arranquemos el Panda:

|  |
| --- |
| coche1.arrancar() |

De nuevo, empleamos también la notación punto para separar la variable del método.

Si observas el código verás que este método se limita a hacer que la variable booleana arrancado valga ahora true. Podrías decir que hubiéramos logrado el mismo resultado actuando sobre el atributo directamente, en lugar de invocar al método:

|  |
| --- |
| coche1.arrancado = true |

En efecto, es así; pero, como comprenderás más adelante, no suele ser buena idea dejar que los programas campen a sus anchas y modifiquen arbitrariamente los atributos de un objeto, siendo preferible que sean los métodos los que se ocupen de esa labor. Imagina que el programa intenta hacer que el cuenta kilómetros marque una cantidad negativa, o que el número de puertas sea igual a cincuenta. Un método correctamente diseñado podría gestionar que los valores estuvieran dentro del rango adecuado y asegurarse de que se cumplen las condiciones que permitirían la modificación del atributo.

La Programación Orientada a Objetos implementa un mecanismo, denominado **encapsulación**, que nos permite **ocultar** determinadas facetas de nuestro objeto y dejar solo accesibles aquellas partes que nos interesen. Pero vayamos por orden, todo a su momento…

Vamos a acelerar nuestro coche:

|  |
| --- |
| coche1.acelerar() |
|  |

Lo que provocará, si consultas el código, que el velocímetro incremente en una unidad su valor.

|  |
| --- |
| coche1.pitar() |

Lo que ocasionará un pitido.

Puedes crear todos los objetos de la clase Coche que desees:

|  |
| --- |
| Coche coche2 = new Coche() |

Cada uno con su propia colección de atributos:

|  |
| --- |
| coche2.marca = "Ford" |
| coche2.modelo = "Fiesta" |
| coche2.color = "Negro" |

Incluso podrías crear otros Seat Pandas, por supuesto. Aunque, en un instante dado, compartieran los mismos valores en sus atributos, se trataría de objetos distintos, **ubicados en direcciones de memoria diferentes** y cada uno podría seguir su propia trayectoria vital.

Extraído de:

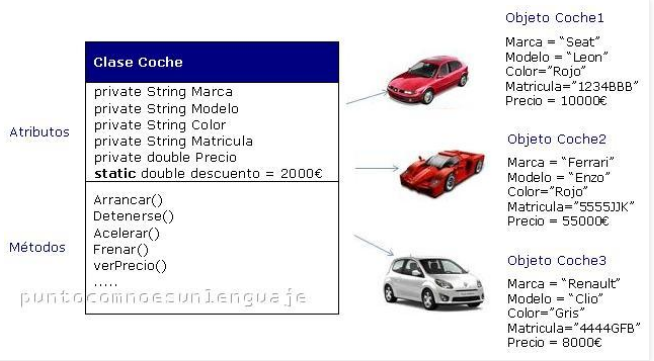
<http://elclubdelautodidacta.es/wp/indice-java/>

# UT4 Java static. Atributos y métodos estáticos o de clase

Los atributos y métodos estáticos también se llaman **atributos de clase** y **métodos de clase**. Se declaran como **static**.

Supongamos una clase Coche sencilla que se utiliza en un programa de compra-venta de coches usados y de la que se crean 3 objetos de tipo Coche.

La clase contiene 6 atributos: marca, modelo, color, matrícula, precio y descuento. Supongamos que el descuento es una cantidad que se aplica a todos los coches sobre el precio de venta. Como este dato es el mismo para todos los coches y es un valor que se puede modificar en cualquier momento no debe formar parte de cada coche sino que es un dato que deben compartir todos. Esto se consigue declarándolo como static.



## **1. Atributos static:**

Son **propios** únicamente **de la clase** **y no de los objetos que pueden crearse de la misma**, por lo tanto, sus valores son compartidos por todos los objetos de la clase. Van **precedidos del modificador static**.

Un atributo static:

* No es específico de cada objeto. Solo hay una copia del mismo y su valor es compartido por todos los objetos de la clase.
* Podemos considerarlo como una variable global a la que tienen acceso todos los objetos de la clase.
* Existe y puede utilizarse aunque no existan objetos de la clase.

Para invocar a una variable estática **no se necesita crear un objeto de la clase en la que se define**:

* Si se invoca **desde la clase** en la que se encuentra definido, basta con escribir su nombre.
* Si se le invoca **desde una clase distinta**, debe anteponerse a su nombre, el de la clase en la que se encuentra definido seguido del operador punto (.) **<NombreClase>.variableEstatica**.

**Ejemplo1:**

public class SerHumano{

String nombre;

String colorOjos;

int edad;

/\*

\* Declaración e inicialización de una variable de instancia estática

\* Tiene sentido declararla estática pues todos los objetos

\* de la clase, teniendo en cuenta que esta modela a un ser humano,

\* habitan en el mismo planeta

\*/

static String planeta="Tierra";

void mostrarCaracteristicas(){

System.out.println(nombre+" tiene "+edad+" años");

System.out.println("Sus ojos son "+colorOjos);

System.out.println("Su planeta es "+planeta);

}

void esMayorEdad(){

if(edad>=18){

System.out.println(nombre+" es mayor de edad");

System.out.println("Tiene "+edad+" años");

}

else{

System.out.println(nombre+" es menor de edad");

System.out.println("Tiene "+edad+" años ");

}

}

public static void main(String args[]){

SerHumano sh1=new SerHumano ();

sh1.nombre="Jesus";

sh1.colorOjos="azules";

sh1.edad=28;

sh1.mostrarCaracteristicas();

sh1.eresMayorEdad();

System.out.println("---------------------");

SerHumano sh2=new SerHumano ();

sh2.nombre="Rebeca";

sh2.colorOjos="verdes";

sh2.edad=27;

sh2.mostrarCaracteristicas();

sh2.esMayorEdad();

System.out.println("---------------------");

System.out.println("FIN DEL PROGRAMA");

}

}

## **2. Métodos static:**

Van **precedidos del modificador** **static**.

Para invocar a un método estático **no se necesita crear un objeto de la clase en la que se define**:

* Tiene acceso solo a los **atributos estáticos de la clase**.
* No es necesario instanciar un objeto para poder utilizarlo.

Si se le invoca desde una clase distinta, debe anteponerse a su nombre, el de la clase en la que se encuentra seguido del operador punto (.)

<NombreClase>.metodoEstatico

**Ejemplo:**

Vamos a escribir una clase Persona que contendrá un atributo contadorPersonas que indique cuántos objetos de la clase se han creado.

contadorPersonas debe ser un atributo de clase ya que no es un valor que se deba guardar en cada objeto persona que se crea, por lo tanto se debe declarar static:

public static int contadorPersonas;

Si lo declaramos como privado:

private static int contadorPersonas;

Desde fuera de la clase Persona solo podremos acceder al atributo a través de métodos static:

public static void incrementarContador(){

contadorPersonas++;

}

public static int getContadorPersonas() {

return contadorPersonas;

}

En este caso un ejemplo de uso puede ser:

System.out.println(Persona.getContadorPersonas());

Cada vez que se crea una persona se incrementará su valor.

Si no es private, desde otra clase podemos hacerlo así:

Persona.contadorPersonas++;

Si es private, desde otra clase debemos incrementarlo así:

Persona.incrementarContador();

**//Clase Persona**

public class Persona {

private String nombre;

private int edad;

private static int contadorPersonas;

public Persona() {

}

public Persona(String nombre, int edad) {

this.nombre = nombre;

this.edad = edad;

}

public void setNombre(String nom) {

nombre = nom;

}

public String getNombre() {

return nombre;

}

public void setEdad(int ed) {

edad = ed;

}

public int getEdad() {

return edad;

}

public static int getContadorPersonas() {

return contadorPersonas;

}

public static void incrementarContador() {

contadorPersonas++;

}

}

**//Clase Principal**

public class Estatico1 {

public static void main(String[] args) {

Persona p1 = new Persona("Tomás Navarra", 22);

Persona.incrementarContador();

Persona p2 = new Persona("Jonás Estacio", 23);

Persona.incrementarContador();

System.out.println("Se han creado: " + Persona.getContadorPersonas() + " personas");

}

}

En lugar de utilizar el método incrementarContador() cada vez que se crea un objeto, podemos hacer el incremento de la variable estática directamente en el constructor.

El código de la clase Persona y de la clase principal quedaría ahora así:

**//Clase Persona**

public class Persona {

private String nombre;

private int edad;

private static int contadorPersonas;

public Persona() {

contadorPersonas++;

}

public Persona(String nombre, int edad) {

this.nombre = nombre;

this.edad = edad;

contadorPersonas++;

}

public void setNombre(String nom) {

nombre = nom;

}

public String getNombre() {

return nombre;

}

public void setEdad(int ed) {

edad = ed;

}

public int getEdad() {

return edad;

}

public static int getContadorPersonas() {

return contadorPersonas;

}

}

**//Clase Principal**

public class Estatico1 {

public static void main(String[] args) {

Persona p1 = new Persona("Tomás Navarra", 22);

Persona p2 = new Persona("Jonás Estacio", 23);

System.out.println("Se han creado: " + Persona.getContadorPersonas() + " personas");

}

}

*No conviene usar muchos métodos estáticos, pues si bien se aumenta la rapidez de ejecución, se pierde flexibilidad, no se hace un uso efectivo de la memoria y no se trabaja según los principios de la Programación Orientada a Objetos*.

|  |
| --- |
| **NOTA:** muchas clases de la API disponen de métodos estáticos. Por ejemplo, la **clase Math** del paquete java.lang cuenta con multitud de ellos. Estos métodos se emplean para realizar operaciones matemáticas. La **clase Thread**, del mismo paquete, cuenta con varios: uno que se emplea para retardar la ejecución de código es “**void sleep(long retardo)**”. Consultar la API. Lo importante de estos métodos es que para su utilización no es necesario instanciar un objeto de las clases en las que se encuentran ya que son estáticos. |

public class MetodosEstaticosAPI{

public static void main(String args[]){

int num=100;

System.out.println("La raiz cuadrada de "+num+" es "+Math.sqrt(num));

//Bloque try ... catch. Se estudiará más adelante. A nivel de

//ejecución no afecta.

//Se introduce un retardo en la ejecución del código de 3 sg

try{

Thread.sleep(3000); //espera 3000ms

}catch(InterruptedException e){}

System.out.println("La potencia de 2 elevado a 8 es "+Math.pow(2,8));

}

}

|  |
| --- |
| **NOTA**: **this** no se puede utilizar en un método estático |

# UT4 Ejercicios básicos

Ejercicio 1.-

Define una clase Persona considerando los siguientes atributos de clase: nombre (String), apellidos (String), edad (int), casado (boolean), numeroDocumentoIdentidad (String). Define un constructor y los métodos para poder establecer y obtener los valores de los atributos.

Ejercicio 2.-

Considera que estás desarrollando un programa Java donde necesitas trabajar con objetos de tipo DiscoMusical. Define una clase DiscoMusical considerando los siguientes atributos de clase: titulo (String), autor (String), añoEdicion (int), formato (String), digital (boolean). Define un constructor y los métodos para poder establecer y obtener los valores de los atributos.

Ejercicio 3.-

Define una clase Profesor considerando los siguientes atributos de clase: nombre (String), apellidos (String), edad (int), casado (boolean), especialista (boolean). Define un constructor que reciba los parámetros necesarios para la inicialización y otro constructor que no reciba parámetros. Crea los métodos para poder establecer y obtener los valores de los atributos.

Ejercicio 4.-

Define una clase Bombero considerando los siguientes atributos de clase: nombre (String), apellidos (String), edad (int), casado (boolean), especialista (boolean). Define un constructor que reciba los parámetros necesarios para la inicialización y los métodos para poder establecer y obtener los valores de los atributos.

Ejercicio 5.-

Se pide definir una clase Medico (que representa a un médico de un hospital) con los siguientes atributos de clase: nombre (String), apellidos (String), edad (int), casado (boolean), numeroDocumentoIdentidad (String), especialidad (String). Definir un constructor que permita asignar valores de defecto a los atributos y los métodos para poder establecer y obtener los valores de los atributos. En cada método, incluye una instrucción para que se muestre por consola un mensaje informando del cambio. Por ejemplo si cambia la especialidad del médico, debe aparecer un mensaje que diga: "Ha cambiado la especialidad del médico de nombre … . La nueva especialidad es: …".

Ejercicio 6.-

Dada la clase Medico del ejercicio anterior, añade un método de nombre "calculoParaMultiploEdad" que no recibe parámetros y que permita determinar cuántos años faltan para que la edad del médico sea múltiplo de 5 y mostrar un mensaje informativo por pantalla. Por ejemplo si el médico tiene 22 años deberá en primer lugar obtener el resto de la división de 22 entre 5, que es 2. Ahora obtendrá los años que faltan para que el médico tenga una edad múltiplo de 5, que serán 5-2 = 3 años. A continuación deberá mostrar un mensaje por consola del tipo: "El médico de nombre … con especialidad … tendrá una edad múltiplo de 5 dentro de … años".

Ejercicio 7.-

Diseñar una clase Rueda que permita representar una rueda de un vehículo. Sus atributos de clase serán: tipo (String), grosor (double), diametro (double), marca (String). Define un constructor asignando unos valores de defecto a los atributos y los métodos para poder establecer y obtener los valores de los atributos. Crea un método denominado **comprobarDimensiones** donde a través de condicionales if realices las siguientes comprobaciones:

a) Si el diámetro es superior a 1.4 debe mostrarse por consola el mensaje "La rueda es para un vehículo grande". Si es menor o igual a 1.4 pero mayor que 0.8 debe mostrarse por consola el mensaje "La rueda es para un vehículo mediano". Si no se cumplen ninguna de las condiciones anteriores debe mostrarse por pantalla el mensaje "La rueda es para un vehículo pequeño".

b) Si el diámetro es superior a 1.4 con un grosor inferior a 0.4, ó si el diámetro es menor o igual a 1.4 pero mayor que 0.8, con un grosor inferior a 0.25, deberá mostrarse por consola el mensaje "El grosor para esta rueda es inferior al recomendado".

Ejercicio 8.-

Diseñar una clase Motor que representa el motor de una bomba para mover fluidos. Define la clase Motor considerando los siguientes atributos de clase: tipoBomba (int), tipoFluido (String), combustible (String). Define un constructor asignando unos valores de defecto a los atributos y los métodos para poder establecer y obtener los valores de los atributos.

Añade un método tipo procedimiento denominado **dimeTipoMotor()** donde a través de un condicional switch hagas lo siguiente:

a) Si el tipo de bomba es 0, mostrar un mensaje por consola indicando "No hay establecido un valor definido para el tipo de motor".

b) Si el tipo de bomba es 1, mostrar un mensaje por consola indicando "El motor es un motor de agua".

c) Si el tipo de bomba es 2, mostrar un mensaje por consola indicando "El motor es un motor de gasolina".

d) Si el tipo de bomba es 3, mostrar un mensaje por consola indicando "El motor es un motor de hormigón".

e) Si el tipo de bomba es 4, mostrar un mensaje por consola indicando "El motor es un motor de pasta alimenticia".

f) Si no se cumple ninguno de los valores anteriores mostrar el mensaje "No se puede clasificar el motor.

Ejercicio 9.-

Dada la clase del ejercicio anterior Motor, diseña un método tipo función que devuelva un booleano (true o false) denominado **dimeSiMotorEsParaAgua()** donde se cree una variable local booleana motorEsParaAgua de forma que si el tipo de motor tiene valor 1 tomará valor true y si no lo es tomará valor false. El método debe devolver la variable local booleana motorEsParaAgua.

# UT4 Ejercicios de clases

## 1. Ejercicios de clases 1

1.- Indica la salida al siguiente programa.

class Ejercicio{

public static void main(String[] args){

Clase1 obj1=new Clase1();

obj1.imprimir(24.3,5);

}

}

class Clase1{

//objeto

private double valor=9.8;

private int x=7;

public void imprimir(double valor, int x){

System.out.print(valor + ” ” + this.x);

//valor se refiere al de Ejercicio

//this.x se refiere al de la Clase1

}

}

*Salida Solucion:*

*24.3 7*

2.- Indicar la salida al siguiente programa.

class Ejercicio {

public static void main(String[] args){

Clase1 obj1=new Clase1(5,4);

System.out.print(obj1.modificar(4)+” “);

Clase1 obj2=new Clase1(5,4);

System.out.print(obj2.modificar(5)+” “);

obj2=obj1;

System.out.print(obj2.modificar(5)+” “);

}

}

class Clase1{

int p1,p2;

public Clase1(int i, int j){

p1=i;

p2=j;

}

public int modificar(int i){

p1=p1+i;

p2=p2+i;

System.out.print(p2+” “);

return p1;

}

//SERIA UTIL PARA DEPURAR

*Salida solucion:*

*1er print*

*P1 = 5+4*

*P2 =4+4*

*Print 8*

*Print 9*

*2do print*

*Print 9*

*Print 10*

*3er print*

*8+5=13  
9+5=14*

**3.-** Crea una clase que represente a un círculo. Debe tener tres atributos, las coordenadas x e y de su centro y su radio.

Tendrá tres constructores, sin parámetros (el centro será 0,0 y el radio 1), pasándole sólo el radio (el centro será 0,0) y pasándole el radio y las coordenadas X e Y.

También tendrá tres métodos uno que calcule el área, otro que calcule la longitud y otro que escriba los resultados.

Después, haz una clase principal en la que se creen 3 objetos círculos y se prueben los métodos.

**4.-.** Crea una clase Punto que modele un punto en un espacio bidimensional. Tendrá dos atributos, x e y, que guardan las coordenadas. Habrá un constructor sin parámetros que crea un punto en (0, 0) y otro al que se le pueden pasar las coordenadas del punto. También habrá métodos para obtener las coordenadas y para imprimir el punto con el formato (x,y).

**5.-** Realiza un programa en Java con la creación de una clase llamada Coche con los atributos color, marca, matricula, número de puertas. Crea un constructor que inicialice el objeto con estos valores y otro que funcione como un constructor por defecto. Escribe métodos para devolver cada uno de los valores de los atributos (dameMatricula, dameColor,...) y un método que simule la operación de pintar el coche cambiando su color. Crea un método main que implemente la solución

**6.-** Realiza un programa en Java que permita crear cuentas bancarias pidiendo la cantidad inicial al usuario, así como realizar operaciones de ingresar y sacar dinero de esas cuentas. Si la cantidad de dinero a sacar es superior a la que hay en la cuenta se mostrará un mensaje advirtiendo que no se puede realizar la operación. El programa irá contando el número de cuentas creadas y lo mostrará al final. El programa dispondrá de un método que imprima la cantidad de dinero que queda en la cuenta. Crea un método main que implemente la solución.

## 2. Ejercicios de clases 2

**1.-** Crea una clase llamada Libro que guarde la información de cada uno de los libros de una biblioteca. La clase debe guardar el título del libro, autor, número de ejemplares del libro y número de ejemplares prestados. La clase contendrá los siguientes métodos:

Constructor por defecto.

Constructor con parámetros.

Métodos Setters/getters

Método préstamo que incremente el atributo correspondiente cada vez que se realice un préstamo del libro. No se podrán prestar libros de los que no queden ejemplares disponibles para prestar. Devuelve true si se ha podido realizar la operación y false en caso contrario.

Método devolución que decremente el atributo correspondiente cuando se produzca la devolución de un libro. No se podrán devolver libros que no se hayan prestado. Devuelve true si se ha podido realizar la operación y false en caso contrario.

Método toString para mostrar los datos de los libros. Este método se heredada de Object y lo debemos modificar (override) para adaptarlo a la clase Libro.

Escribe un programa para probar el funcionamiento de la clase Libro.

**2.-** Creamos una clase llamada empleado con los atributos nombre, apellido, edad, salario de forma que no se puedan acceder de ninguna clase externa. Crea un constructor vacío y otro que reciba cuatro parámetros con los valores de sus cuatro atributos. Crea un método para cambiar cada uno de los atributos y otro para devolver cada uno de los atributos, todos deben poder ser usados desde cualquier clase externa.

Crea un método que permita que a un empleado se le sume un plus a su salario siempre y cuando su edad sea de 40 años o más.

Para probar la clase Empleado que has desarrollado crea una clase que cree varios empleados (pide el número al usuario) y los referencie en un array que también crearás. Sube a los empleados que tengan derecho 50 euros a su sueldo, indicando en el programa a qué empleados se les ha podido realizar la subida y a cuáles no (imprime los datos de cada empleado, a continuación si se les ha subido o no el sueldo y el nuevo sueldo que tendrán).

Si para hacer lo que te piden necesitas a algún método más de los que se te indican, añádelo.

## 3. Ejercicios de clases 3

**1.1.-** Escribe el código en Java de una clase para representar un empleado de una empresa. Del empleado vamos a querer tener los datos de su nombre, apellido, edad y salario.

Realiza dos métodos constructores, uno con los valores de los cuatro atributos y otro sin atributos.

Crea también los métodos que devuelvan los valores de los atributos y otros que permitan modificarlos, así como un método que si el empleado tiene más de 40 años se le aumente el sueldo una cantidad que se pasa por parámetro. A este último método le llamaremos comprobarPlus y devuelve true si se aumenta el sueldo y falso si no se aumenta.

Se debe crear también un método toString() para imprimir un objeto Empleado.

Utiliza la clase JOptionPane para recibir los valores del usuario para ir creando los empleados. Para ello, crearás un método denominado **leerEmpleado()** en la misma clase donde tengamos el método main(), este método pide los datos de un empleado y devuelve un objeto de la clase Empleado.

Crea tres empleados para probar los métodos descritos.

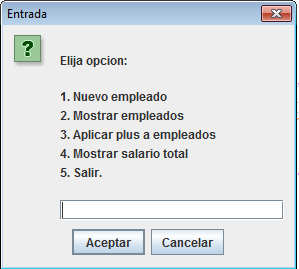
**1.2.-** Haz el ejercicio creando un array de empleados, donde se puedan acumular hasta 100 empleados (usa menos para probar el programa). Crea los empleados guardándolos en el array, para posteriormente comprobar si hay que aumentar el plus a los empleados.

Muestra los datos de cada empleado del array.

Muestra la suma del salario de todos los empleados antes y después de realizar el aumento que corresponda.

Haz el ejercicio con JOptionPane.

**1.3.-** Haz el ejercicio ahora con un menú como el siguiente:



Para ir creando los empleados usarás la **1ª opción** y un **atributo estático** que va contando los empleados según se van creando en el/los constructor/es.

En la **2ª opción** mostrarás todos los empleados creados hasta el momento en que se selecciona esa opción

En la **3ª** **opción**, se recorre el array y se sube la cantidad indicada como plus a los mayores de 40 años. Fíjate en que si se elige varias veces esta opción, habrá empleados a los que se les aplique la subida más de una vez. Busca una solución a este problema.

En la **4ª opción**, se muestra la suma de todos los empleados creados hasta el momento en que se selecciona la opción.

En esta versión del programa, deberás crear métodos estáticos en el fichero fuente del main, que reciban el array de empleados y lleven a cabo lo que se pide en cada opción. Es decir, **en el menú aparecerán solo las llamadas a los métodos**.

Si el usuario selecciona alguna opción y no hay empleados, el programa deberá notificarlo.

**2.-**  Implementa una clase en Java que permita realizar promedios. La clase debe tener dos métodos, uno para ingresar un nuevo número, llamado *agregarNumero(int numero)* y otro para obtener el promedio hasta el momento, llamado *obtenerPromedio()*. Determina qué atributos son necesarios para implementarla.

Implementa luego una clase de prueba que permita introducir algunos valores y que muestre el promedio.

**3.-** Implementa una clase en Java que represente una fracción de números enteros. Implementa asimismo los siguientes métodos en la clase Fracción:

* Suma de dos fracciones, que será una nueva fracción. Resta de dos fracciones, similar a la suma.
* División de dos fracciones.
* Multiplicación de dos fracciones.
* Calculo de la inversa de una fracción (cambiar numerador por denominador y viceversa).

Todos los métodos se realizan sobre un objeto Fracción pasándole por parámetro la segunda fracción cuando sea necesario.

Intenta simplificar los resultados usando un nuevo método.

**4.-** Realiza un programa en Java que, mediante un menú con varias opciones haga las siguientes tareas (hazlo con Scanner y con la clase JOptionPane):

**Opción 1:** Pide los datos de un alumno (nombre, apellido, curso, nota) y da de alta al alumno en nuestro programa.

**Opción 2:** Muestra los datos de los alumnos cuya nota media es mayor o igual a 5 y el número de ellos que hay que cumplan esa media.

**Opción 3:** Muestra los datos de los alumnos cuya nota media es menor a 5 y el número de ellos que hay que cumplan esa media.

Utiliza funciones (métodos estáticos) para llevar a cabo las tareas pedidas.

**5.-** Crea una clase en Java llamada Cubo con dos atributos:

int capacidad; // capacidad máxima en litros

int contenido; // contenido actual en litros

La clase tendrá un constructor que recibe la capacidad inicial del cubo. Una vez establecida la capacidad, ya no será posible modificarla.

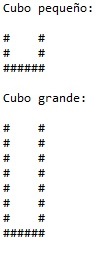
Habrá un método llamado **llena()**, que llenará el cubo hasta su capacidad máxima y otro **vacia()** que pone el contenido a 0 litros.

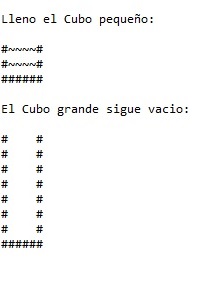
Haz un método llamado **vuelcaEn()** que vuelca el contenido de un cubo sobre otro. Antes de echar el agua se comprueba cuánto le cabe al cubo destino (será el que se manda por parámetro (void vuelcaEn(Cubo destino)).

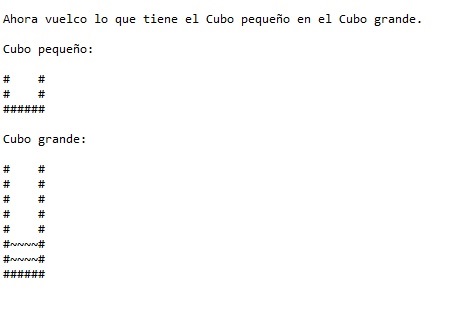
Se pide también un método llamado **pinta()** que pinta un cubo en la pantalla. Se muestran los bordes del cubo con el carácter # y el agua que contiene con el carácter ~.

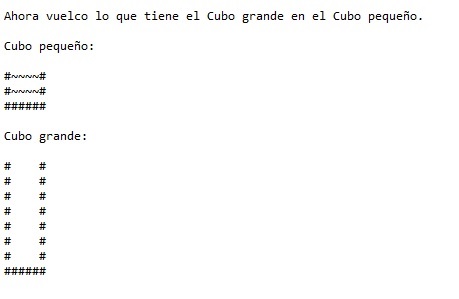
Cada litro se representa con una línea.

Para probar el ejercicio, ve haciendo las sentencias correspondientes a las siguientes salidas:









## 4. Ejercicios de clases 4

**1.-** Desarrolla una clase Cafetera con atributos:

**capacidadMaxima** (la cantidad máxima de café que puede contener la cafetera)

**cantidadActual** (la cantidad actual de café que hay en la cafetera)

Implementa, al menos, los siguientes métodos:

* **Constructor** predeterminado: establece la capacidad máxima en 1000 (c.c.) y la actual en cero (cafetera vacía).
* **Constructor** con la capacidad máxima de la cafetera; inicializa la cantidad actual de café igual a la capacidad máxima.
* **Constructor** con la capacidad máxima y la cantidad actual. Si la cantidad actual es mayor que la capacidad máxima de la cafetera, la ajustará al máximo.
* **Accedentes y mutadores (getter y setter)**.
* **llenarCafetera()**: hace que la cantidad actual sea igual a la capacidad máxima.
* **servirTaza(int)**: simula la acción de servir una taza con la capacidad indicada por parámetro. Si la cantidad actual de café en la cafetera “no llega” para llenar la taza, se sirve lo que quede.
* **vaciarCafetera()**: pone la cantidad de café actual en cero.
* **agregarCafetera(int)**: añade a la cafetera la cantidad de café indicada en el parámetro. Si esa cantidad excede el máximo, ajusta al máximo.

Escribe el código en el programa principal para probar lo siguiente:

**Capacidad máxima = 1000 Cantidad actual = 0**

**Agregamos 20 c.c. de cafe...**

**Capacidad máxima = 1000 Cantidad actual = 20**

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**Llenamos la cafetera...**

**Capacidad máxima = 1000 Cantidad actual = 1000**

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**Servimos una taza de 500 c.c....**

**Todavia quedan 500 c.c.**

**Capacidad máxima = 1000 Cantidad actual = 500**

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**Servimos una taza de 600 c.c....**

**Se sirve todo lo que quedaba (500 c.c.)**

**Capacidad máxima = 1000 Cantidad actual = 0**

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**Servimos una taza de 200 c.c....**

**Lo siento, pero no queda nada de cafe.**

**Capacidad máxima = 1000 Cantidad actual = 0**

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**Llenamos la cafetera con 700 c.c....**

**Capacidad máxima = 1000 Cantidad actual = 700**

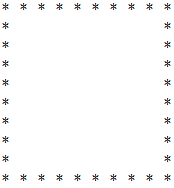
**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**Llenamos la cafetera con 400 c.c....**

**Capacidad máxima = 1000 Cantidad actual = 1000**

**2.-** Implementa una clase en Java que permita representar cuadrados. Cada objeto Cuadrado vendrá representado por sus cuatro vértices, que serán los atributos de la clase. Además de los métodos para modificar y devolver los valores de los vértices, se piden los siguientes métodos:

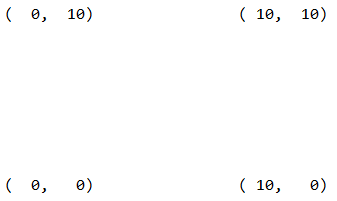
Cada cuadrado tendrá un método (dibujar) para representarse por medio de asteriscos. Por ejemplo, el cuadrado con vértices (0, 0) (0, 10) (10, 10) y (10, 0) se representaría:



También queremos un **método (dibRellenando)** que represente el cuadrado relleno de asteriscos.



Otro **método (dibujarVertices)** para escribir los 4 vértices del cuadrado de la forma siguiente para el ejemplo anterior:



Un **método (esCuadrado)** que devuelva true si se trata de los vértices de un cuadrado y false si no lo es.

La *distancia entre dos puntos* P1: (x1, y1) y P2: (x2, y2) viene dada por la fórmula:

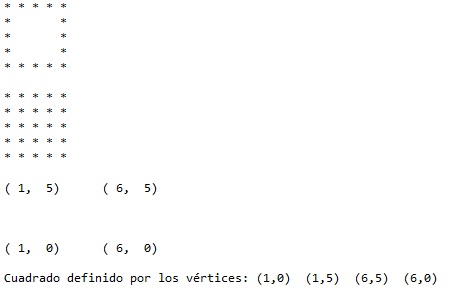
**D=**

Cada cuadrado tendrá un **método (lado)** que devuelve el valor del lado del cuadrado creado.

Escribe un **método (compara**) en la clase Cuadrado para poder comparar el área de dos cuadrados. Este método recibe un cuadrado y lo compara con el cuadrado que invoca el método, actuando igual que el método compare de la clase String.

Implementa luego una clase de prueba que permita crear objetos de la clase Cuadrado y probar todos sus métodos.

*Otro ejemplo:*



Observa que para dibujar los cuadrados vamos a ignorar la distancia al eje de las x y de las y, los pintamos siempre desde el punto P1(x1, y1), teniendo únicamente en cuenta la longitud del lado.

**3.-** Vamos a escribir un programa para representar el consumo de energía de una instalación eléctrica. Para ello, se hará una clase que representa los aparatos conectados en la instalación.

Cada aparato tiene un consumo eléctrico determinado. Al encender un aparato eléctrico, el consumo de energía se incrementa en la potencia de dicho aparato. Al apagarlo, se disminuye el consumo en dicha potencia.

Inicialmente, los aparatos están todos apagados.

Además, se desea consultar el consumo total de la instalación.

Haz un programa que declare dos aparatos eléctricos, una bombilla de 150 watios y una plancha de 2000 watios. El programa deberá imprimir el consumo nada más crear los objetos. Después, se enciende la bombilla y la plancha, y el programa imprime el consumo. Luego se apaga la bombilla, y se vuelve a imprimir el consumo.

**Ejemplo de salida:**

Inicialmente el consumo eléctrico es 0.0

Encendemos la bombilla Potencia 150.0

Encendemos la plancha Potencia 2000.0

El consumo eléctrico es 2150.0

Apagamos la plancha Potencia 2000.0

El consumo eléctrico es 150.0

## 5. Enunciado clase 1 de diciembre - Fraccion

Implementa una clase en Java que represente una fracción de números enteros. Implementa asimismo los siguientes métodos en la clase Fracción:

* Suma de dos fracciones, que será una nueva fracción. Resta de dos fracciones, similar a la suma.
* División de dos fracciones.
* Multiplicación de dos fracciones.
* Calculo de la inversa de una fracción (cambiar numerador por denominador y viceversa).

Todos los métodos se realizan sobre un objeto Fracción pasándole por parámetro la segunda fracción cuando sea necesario.

Intenta simplificar los resultados usando un nuevo método.

<http://www.objetos.unam.mx/matematicas/leccionesMatematicas/01/1_005/index.html>