COLEXIO VIVAS S.L.	RAMA:	Informática	CICLO:	DAM	
	MÓDULO	Programación		CURSO:	1º
	PROTOCOLO:	Apuntes clases	AVAL:	DATA:	
	AUTOR	Francisco Bellas Aláez (Curr			

Tema 1 – Introducción a Java.

Definiciones iniciales

Este “es un curso de programación en Java”; pero lo más importante es que es un “curso de programación” y por tanto el lenguaje y las herramientas usadas en principio no importan. El objetivo es obtener una mentalidad de programador, de pensar con las mismas estructuras en que se programa un ordenador y en resumen plantear soluciones a problemas mediante estas estructuras que se utilizan en el mundo de la programación. Por supuesto que se necesita un lenguaje y herramientas para programar, pero en el fondo todos los lenguajes (dentro de una categoría) son muy similares y simplemente cambian elementos sintácticos pero no de fondo, ya que lo que buscan los lenguajes es procesar datos de la forma más eficiente y cómoda posible.

Por tanto vamos a empezar a programar y como nuestra herramienta es el Java aprenderemos este lenguaje. Pero ¿qué se puede hacer con Java? Básicamente los siguientes tipos de aplicaciones:

- Aplicaciones de consola
- Aplicaciones en entorno gráfico (PC de escritorio o dispositivo móvil)
- Aplicaciones para el mundo Web (en cliente y servidor)

En esta materia nos quedaremos en los dos primero tipos de aplicaciones. Empezaremos por el primer caso en el que fabricaremos pequeñas aplicaciones que funcionarán en una consola del sistema operativo para luego evolucionar y terminar haciendo aplicaciones en un entorno gráfico de ventanas como puede ser GTK+, MS Windows, MacOS y otros.

En este primer tema veremos las bases de la programación, aquellos elementos o ladrillos que son necesarios manejar con absoluta fiabilidad antes de meterse a aprender estructuras más complejas como las clases, colecciones, streams, etc.

En cualquier caso Java, y en general cualquier lenguaje orientado a objetos es un lenguaje cuyo aprendizaje en principio puede parecer un poco extraño pues es necesario hacer un pequeño ejercicio de fe y creerse ciertas cosas que se irán entendiendo a medida que pasen los temas.

Durante los primeros temas empezaremos los programas de forma muy similar, con las líneas:

```
public class NombreDelPrograma {
      public static void main (String args[])
```

y no entenderemos qué es eso de *public* o de *static* hasta que lleguemos al tema de Orientación a Objetos. ¿Y por qué no empezamos por ese tema? Porque se haría muy tedioso y complicado entrar directamente en esa filosofía que se va a usar luego con total naturalidad.

Podríamos pararnos a continuación a contar un montón de aspectos sobre Java pero entiendo que en otro módulo de este ciclo profundizaremos en los distintos lenguajes de programación y será ahí donde el alumno obtendrá una visión más amplia del Java dentro del mundo de la programación.

Pero antes de entrar en materia conviene conocer algunos acrónimos y nombres del mundo de Java para tener en cuenta de qué hablamos en cada momento:

Java SE: Standar Edition. Versión estándar para programar en Java.

Java EE: Enterprise Edition. Enfocado al mundo empresarial. Para trabajo con arquitecturas cliente-servidor entre otros.

JRE: Java Runtime Environment. Es el conjunto de elementos básicos para poder ejecutar una aplicación Java. Al menos se compone de la JVM (Máquina virtual Java) y una serie de paquetes con clases (API). Tanto la JVM como la API deben estar en la misma versión para que no haya inconsistencias. Por supuesto con el JRE sólo se pueden ejecutar aplicaciones, para construirlas es necesario el JDK.

JDK: Java Development Kit. Conjunto de paquetes que permiten crear aplicaciones en Java. Además de incluir el JRE dispone también del compilador (javac), del depurador (jdb) y del creador de documentos(javadoc).

Resumiendo: si sólo quiero ejecutar programas Java me llega el JRE. Si además quiero programar, crear programas en Java necesito el JDK.

Applets: Aplicaciones incrustadas en clientes (navegadores) Web. Podrían incrustarse en otras aplicaciones.

AWT: Abstract Window Toolkit. El primer paquete de componentes gráficos para realizar aplicaciones en Java. Hoy se usa para tareas de bajo nivel de otras clases gráficas. Algunos programadores la siguen usando porque produce que las aplicaciones tengan el aspecto gráfico del sistema nativo. Pero sigue siendo muy limitado pues dispone de aquellos elementos comunes en todos los sistemas operativos. Además no funciona bien del todo.

Swing: El sistema actual que mediante Java 2D realiza sus propios dibujos (en lugar de hacerlo el SO con el awt). Al ser independiente del SO, las aplicaciones tienen el mismo aspecto independientemente del equipo en el que se ejecuten. Se supone que se irá sustituyendo por JavaFX que es el nuevo sistema gráfico.

Ficheros JAR: Es un fichero ZIP que contiene una serie de elementos que conforman una aplicación: imágenes, clases java, archivos de audio, etc... De esta forma se pueden tener toda una aplicación empaquetada aunque conste de muchos archivos independientes.

Se pueden ver más definiciones de distintos elementos de la arquitectura de Java en el Apéndice de este tema.

Instalación del JDK

A continuación se presentan los pasos que habría que hacer para preparar un equipo:

Linux (Ubuntu, Mint o similar)

Si ya tenemos el JRE instalado debemos comprobar la versión del mismo para instalar el mismo JDK. Para ello:

```
$ java -version
```

```
la salida será algo similar a esto:
openjdk version "1.8.0_131"
OpenJDK Runtime Environment (build 1.8.0_131-8u131-b11-2ubuntu1.16.04.3-b11)
OpenJDK 64-Bit Server VM (build 25.131-b11, mixed mode)
```

La primera línea es la versión genérica de Java.

La segunda la versión y "modelo" del JRE. En este caso se puede ver que no es el JRE de Oracle ("Propietario" de Java) si no que es el OpenJDK que es un "Java alternativo" que cumple todas las especificaciones de Java pero es Open Source.

Finalmente la tercera línea es la JVM usada. También existen varias tanto de código abierto como privativo.

Debemos instalar la misma versión del JDK, es decir, la 1.8 ó, quitando el 1, la versión 8 que es como se suelen denominar a las versiones de Java.

Los más probable es que no tengas nada instalado, en ese caso directamente al instalar el JDK ya instalas el JRE.

Nota: Se podría instalar la 1.9 o 1.10 que son más actuales pero aún no está en desarrollo completo (aunque sí la de Oracle) y no nos afectan los cambios de versión para este curso. Además la 1.8 es la que se sigue usando en Android.

Por tanto:

```
$ sudo apt-get update
$ sudo apt-get install openjdk-8-jdk
```

Si fuera otra versión habría que sustituir el número correspondiente.

Si por algún motivo no estuviera instalado el JRE, al instalar el JDK ya instalaría el JRE automáticamente. Si se quisiera instalar solo el JRE llegaría con cambiar el paquete a instalar por openjdk-8-jre

Nota: Si en un ordenador hubiera varias versiones de java se puede seleccionar la que se usa por defecto mediante el comando:

```
$ sudo update-alternatives --config java
```

*Si lo que quieres cambiar es la versión de javac, simplemente ejecuta lo anterior acabado en **javac** en lugar de **java***

```
$ sudo update-alternatives --config javac
```

Windows (válido también para Linux si se desea la versión de Oracle)

En la web http://java.com/es/download/ se puede bajar la versión recomendada del JRE o comprobar si ya se tiene instalado. También abriendo una consola y usando el comando:

```
c:\>java -version
```

nos dirá la versión instalada.

Si además queremos instalar el JDK podemos acceder a la página:

http://www.oracle.com/technetwork/java/javase/downloads/index.html

y bajarnos la versión en concordancia con el JRE que tenemos instalado. Si no tuviéramos ninguno desde esta Web también se puede descargar.

Una vez descargado se ejecuta (son necesarios permisos de administración) y se siguen los pasos de instalación.

Finalmente para poder ejecutar el compilador desde cualquier directorio del disco duro es necesario cambiar las variables de entorno. Para ello se debe ir al Panel de Control (en modo iconos, no categorías) y buscar el icono de Sistema.

Si estás en Windows 7, 8 o 10 además debes pulsar en Configuración Avanzada de Sistema (a la izquierda).

En la nueva ventana se pulsa en

Opciones Avanzadas → Variables de entorno

Se edita la variable de entorno *Path* del sistema (las inferiores) y se le añade:

```
C:\Program Files\Java\jdk1.8.0_101\bin
```

o el directorio equivalente según donde se haya instalado y la versión instalada. Es decir, cambiará la carpeta *jdk1.8.9_101* por otra, pero es importante que se indique el *bin*.

Mac OS X

No es necesario instalar nada. Ya viene por defecto tanto el JRE como el JDK

Creando, compilando y ejecutando un programa en Java

Una vez que tenemos el equipo preparado vamos a realizar nuestro primer programa en Java. Este simplemente mostrará una frase en la consola. Abre un editor de texto (**¡No procesador de texto!**) cualquiera, por ejemplo el *nano*, el *gedit* o el *mousepad* y escribe el siguiente código.

```
public class Hola {
      public static void main (String args[]){
            System.out.println("Welcome to de Java World");
      }
}
```

Guarda el programa con el nombre

Hola.java

ya que es imprescindible que el nombre del archivo coincida con el nombre de la clase (*class* indicará por ahora el nombre del programa aunque ya se verá en un futuro tema la utilidad de dicho concepto), si no nos dará un error al compilar. Este es un requisito de Java. La clase pública (en la mayoría de los casos la única clase en el archivo) tendrá el mismo nombre que el archivo.

Abre ahora un terminal y vete al directorio donde hayas grabado el código. Suponiendo que está en el directorio *programas* usa el comando *cd* de la siguiente manera.

```
nombre@equipo:~$ cd programas
```

A partir de ahora pondré solo el $ para indicar entrada de línea de comando.

Si ahora escribes el siguiente comando:

```
$ ls
```

o

```
$ ls -lh
```

verás los archivos que hay en la carpeta *programas*. En el segundo caso hay más información: permisos, propietario, tamaño, fecha,...

Tiene que aparecer, por supuesto, Hola.java.

*Nota: En el caso de Windows el comando para cambiar de directorio también es **cd** y el de listar los archivos del directorio es **dir**.*

Bien, a continuación vamos a compilar el código Java a un lenguaje (bytecodes) que comprenda la máquina virtual (JVM). Para ello ejecutamos:

```
$ javac Hola.java
```

Si hay algún error saldrá ahora algún tipo de mensaje. Los errores más típicos son el escribir una letra minúscula en mayúscula o viceversa, olvidarse de un punto y coma o de una llave o escribir mal un comando. Si no ha habido problema ejecutemos el comando *ls* y se puede ver que ha aparecido un nuevo archivo denominado *Hola.class*. Ese es el programa que puede ejecutar la JVM. Para ello:

```
$ java Hola
```

Se puede observar que no es necesario escribir el .class (de hecho no funciona si se hace). Como resultado aparece la frase que hemos puesto entre comillas en el comando *System.out.println()* de Java. Efectivamente mediante dicho comando sacaremos mensajes de distinto tipo a la consola.

En general compilar implica traducir el código fuente (Java, Pascal, C, ...) al código máquina (binario) que entiende la máquina en la que se ejecutará. En el caso de Java esa máquina es virtual, es decir, es un emulador de una máquina pero también tiene su propio juego de instrucciones en binario. Se profundizará en esto en otro módulo.

Más adelante iremos automatizando estas tareas con editores de código más potentes (Visual Studio Code) y finalmente mediante añadidos (Plugins) lo iremos convirtiendo en un entornos de desarrollo completo (como podría ser Netbeans, Eclipse, o IntelliJ, que también nos valdrían)

Veamos ahora un ejemplo de lo que sería una aplicación gráfica (muy simple) en Java. En un nuevo archivo escribe el siguiente código:

```
import javax.swing.*;

public class HolaWin{
      public static void main(String[] args){
            JOptionPane.showMessageDialog(null,
                  "Welcome to the Java World",
                  "Usando Gráficos",
                  JOptionPane.INFORMATION_MESSAGE);
      }
}
```

En futuros temas entenderemos el significado de todo lo aquí expuesto.

Conceptos básicos de programación y estructura de un programa en Java

Programa (o Algoritmo): Serie de operaciones detalladas y no ambiguas, a ejecutar paso a paso, y que conducen a la resolución de un problema.

Hay diversas formas de realizar un algoritmo. Simplemente hay que definirlo de forma no ambigua.

Dato: Información que procesa un algoritmo (y por tanto un futuro programa).

Tipo de dato: Tipo de información que se maneja. Es un concepto muy importante y que hay que ir teniendo muy claro al desarrollar programas.

Cuando se trabaja con un dato, hay que saber si dicho dato es un número entero, un número real, un carácter, una frase (una lista de caracteres), un valor lógico, etc… ya que el ordenador todo lo que guarda son 0s y 1s y por lo tanto es necesario indicar el significado de esos valores binarios.

Variable: Es un elemento que contiene un dato y su valor puede variar a lo largo del programa. Internamente en el ordenador es una zona de memoria reservada donde se guarda un dato de determinado tipo y cuyo valor cambia a lo largo de la ejecución.

Atributos:

- Nombre o identificador: Lo que la designa
- Tipo de dato: El tipo de información que puede manejar esa variable.

Se utilizarán para recoger datos, para manejar datos internos variables y para devolver datos.

En principio las variables se almacenan en una zona de memoria reservada para el programa denominada *heap* o *montículo*.

Literal: Elemento de valor invariable y de un tipo determinado. Al igual que una variable internamente en el ordenado ocupa una región de memoria pero su contenido no varía, esto permite un uso más eficiente en determinadas circunstancias.

Ejemplos:

- Entera: -3 , 5, 17
- Real: $6x10^8$ (se verá como 6e8), 5.34
- Carácter : 'A'
- Cadena de caracteres (String): “Hola”

Constante: Es un elemento que contiene un dato pero su valor NO varía a lo largo del programa. Es muy

cómodo utilizarlas por claridad o por hacer un programa más corto. Por ejemplo es más cómo usar la palabra PI que el número 3.14159265359.

Expresiones: Combinaciones de variables, constantes símbolos de operación, paréntesis y nombres de funciones especiales.

Ejemplo:

```
m*(n+5)-sqrt(p)
```

Veamos la **estructura de un programa** en Java con un nuevo ejemplo ligeramente más amplio que el primero.

```
import java.util.Scanner;

public class Hola2 {
	public static void main (String args[]) {
		// Declaración de variables
		String nombre;
		Scanner sc= new Scanner(System.in);

		// Código
		System.out.print("Dime tu nombre: ");
		nombre=sc.nextLine();
		System.out.println(nombre+", Welcome to the Java World");
		System.out.println("Nos vemos");
	}
}
```

Antes de explicar las distintas líneas veamos unos aspectos generales:

- Cada sentencia acaba en ; ya que es el indicador de fin de sentencia. Puedo escribir una sentencia en varias líneas como hicimos en el ejemplo de GUI ya que lo que marca el final es el ;
- Las llaves {} marcan inicio y fin de bloque. Cuando hay una serie de comandos o sentencias que queremos meter dentro de un bloque los meteremos entre estos símbolos.
- La identación, tabulación o sangrado es esencial en el mundo de la programación. Sirve para entender que comandos están dentro de que bloques de forma clara. En algunos lenguajes si no se identa el programa no funciona.

Los elementos que nos encontramos a lo largo de cualquier programa son:

- **Palabras reservadas:** Son palabras que pertenecen al lenguaje Java y por tanto no pueden ser usadas por nosotros para nombrar variables. La lista completa es:

```
abstract   default   if          private      this
boolean    do       implements  protected    throw
break      double   import      public       throws
byte       else     instanceof  return       transient
case       extends  int         short        try
catch      final    interface   static       void
```

```
char       finally   long         strictfp      volatile
class      float     native       super         while
const      for       new          switch
continue   goto      package      synchronized
```

- **Identificadores:** Son palabras que crea el programador para dar diferentes nombres a elementos como las variables, constantes, las clases, funciones, etc…

  Hay algunas limitaciones para poner nombres a las variables:

  - Se usan sólo letras, números, y los símbolos _ y $
  - No puede, por tanto, contener espacios.
  - No puede empezar por número.
  - Se distingue entre mayúsculas y minúsculas .
  - No se pueden usar palabras reservadas (class, import, etc…)

Actividad: Indica cuales de estos nombres son válidos para identificadores y cuales no:

nombre, $numero, dato1, 1dato, media total, kilometro/hora, kiloBitsPorSegundo, x8c, mi_dato.

De cara a la claridad de programación, vamos además a usar algunas reglas en cuanto a la creación de nombres de identificadores: Las clases siempre empezarán por mayúsculas y las variables por minúsculas. Si una clase o variable es una palabra compuesta, las siguientes palabras se escribirán comenzándolas en mayúscula: Clase *SistemaOperativo*, variable *datoTransferido*.

Esta forma de escribir es un estándar denominado CamelCase (en el caso particular de Java es lowerCamelCase).

- **Comentarios:** Son líneas de código que no afecta al funcionamiento del programa. No se compilan. Por tanto son usadas para aclarar ciertas partes del código que tienen un funcionamiento especial o para documentar de forma técnica el programa. Existen tres tipos de comentarios en java:
  - Comentario de línea: Comienza donde se escriba `//` hasta el fin de la línea.
  - Comentario multilínea: Comienza con `/*` y acaba con `*/` en la misma línea o en otra.
  - Comentario de documentación: el SDK de Java incorpora un programa denominado *javadoc* que facilita la creación de documentación técnica y cuyo funcionamiento veremos en otra asignatura. Los comentarios detectados por *javadoc* comienzan por `/**` y acaban por `*/`.
- **Constantes literales:** Datos explícitos como 3, “hola”, '$' ó -78.23

Teniendo en cuenta estos puntos anteriores vamos a explicar de forma aproximada el funcionamiento del programa anterior:

Importación

Lo primero que encontramos en un programa en Java son las líneas de importación. En estas líneas se indican los paquetes (**packages**) donde existen clases de Java que podemos necesitar y ya están programados. Una clase podemos verla como un contenedor. Por tanto todos los comandos de Java (salvo las instrucciones más básicas) aparecerán dentro de clases que a su vez se guardan en paquetes.

En este caso se importa la clase *Scanner* que vamos a usar para pedir datos al usuario. Esa clase se encuentra en el paquete *util* que a su vez está en el paquete genérico *java*. Ese es el significado de java.util.Scanner.

Puede haber varias líneas de importación para usar varias clases, pero en el caso de usar diversas clases que están en un paquete puede usarse el comodín *. Es decir, si en un programa tenemos que importar clases de esta manera:

```
import java.io.File;
import java.io.IOException;
import java.io.BufferedReader;
import java.io.InputStream;
import java.io.IOException;
```

podría ser más cómodo escribirlo en una única línea así:

```
import java.io.*;
```

Ese * indica "*todas las clases del paquete* io *que está en* java". Los puristas dicen que es mejor escribir sólo las clases que se van a usar por claridad, pero realmente no se pierde eficiencia y sí tiempo escribiendo.

Actividad: Prueba a quitar la línea import y comprueba el error que se produce.

La clase principal.

```
public class Hola2 {
}
```

Tras la zona de importación en el archivo viene la clase principal. Como ya se comentó una clase la veremos por el momento como un contenedor de elementos y ya veremos definiciones más precisas y correctas. Nos quedamos con la idea de que todo el código que vamos a meter hay que hacerlo dentro de la clase.

Una clase comienza por la palabra *class* y a continuación se le da el nombre deseado. Ya comentamos también que por claridad y estandarización vamos a nombrar las clases comenzando por letra mayúscula.

Todo lo que va dentro de la clase se mete entre llaves.

En un archivo se pueden meter varias clases pero por ahora meteremos sólo una que sirva para contener nuestros programas. Además el archivo ha de tener el mismo nombre que dicha clase (conservando mayúsculas y minúsculas) y precediendo a la palabra *class* irá le modificador *public* que indica que "cualquiera" puede acceder a la clase. Entenderemos mejor esto último en el tema de orientación a objetos.

Actividad: Si no lo has probado con algún ejemplo anterior prueba a nombrar la clase de forma distinta al archivo para ver el error que da. Prueba también a cambiar simplemente mayúsculas o minúsculas y haz la misma comprobación. Finalmente prueba a quitar alguna llave.

Programa principal (función principal).

```
public static void main (String args[]) {
}
```

Un apartado denominado por la palabra reservada *main* (principal) y precedido por las palabras reservadas *public*, *static y void* es el bloque donde meteremos las sentencias ejecutables que conformarán nuestro programa al menos al principio.

A este apartado se le denominará también función o método principal o función o método main.

En general una función o método es una zona del programa que contiene comandos ejecutables y que puede ser invocada desde otras partes del programa o sistema. Crearemos más adelante otros métodos distintos del principal.

El elemento especificado entre paréntesis (String args[]) no lo tendremos en cuenta al menos por el momento.

Cuando se construye un programa Java hay que indicar las variables que se van a usar y el tipo de dato que son. Realmente se puede hacer en cualquier parte de la función pero por claridad conviene separar la zona de declaración de variables (se puede poner al principio) del resto del código.

Vamos a ver dos grandes bloques de variables, por un lado las básicas que simplemente hay que declarar su nombre y el tipo de dato que contendrán:

```
String nombre; //Variable que guarda una cadena de caracteres
int numero; // Variable que guarda un valor entero
```

y los objetos o variables referenciadas que, sin entrar en detalle, son “variables” para las cuales hay que reservar memoria y por tanto se declaran e inicializan en la misma línea.

```
Scanner sc = new Scanner(System.in);
```

aunque podría hacerse en líneas distintas (una de declaración y otra de reserva de memoria):

```
Scanner sc;
sc = new Scanner(System.in);
```

En este caso se inicializa la variable *sc* que es el objeto que representa al teclado (*System.in*) y va a servir para pedirle datos al usuario.

La diferencia entre las simples y las referenciadas es que las simples básicamente manejan un valor único y las referenciadas son complejas en cuanto a que pueden guardar varios valores e incluso contener código ejecutable asociado a la variable (son los denominados objetos).

Finalmente describimos brevemente cada línea de código:

```
System.out.print("Dime tu nombre: ");
```

Muestra una frase en pantalla pero no pasa a la línea siguiente.

```
nombre=sc.nextLine();
```

Se queda esperando a que el usuario introduzca una cadena de caracteres y una vez que el usuario pulsa enter introduce el dato en la variable *nombre*.

```
System.out.println(nombre+", Welcome to de Java World");
```

Muestra en pantalla el contenido de la variable nombre seguido del texto , *Welcome to de Java World* sin las comillas. La operación + con cadenas de caracteres se denomina concatenación (coloca una cadena detrás de otra). Además una vez que termina de escribir pasa a la línea siguiente.

```
System.out.println("Nos vemos");
```

Muestra otra frase en pantalla y pasa a la línea siguiente.

Nota: si leéis algún documento en la que se explique una versión de Java anterior a la 1.5 la introducción de datos se realizaba con otro objeto de la siguiente manera:

BufferedReader br = new BufferedReader(new InputStreamReader(System.in));

nombre=br.readLine();

Además había que hacer el denominado control de excepciones que veremos más adelante pero que aumenta la complejidad de la entrada de datos.

Tipos de Datos primitivos en Java

En este apartado veremos qué tipos de datos podemos manejar y, sobre todo, guardar en las variables que usemos en nuestros programas.

Como se comentó en un apartado anterior, existen dos grandes grupos de tipos de datos, los sencillos o primitivos que serán la mayoría de los que veamos en este apartado, y los referenciados, compuestos u objetos, que necesitan más memoria y se reserva de una forma especial, no llega sólo la declaración.

byte: Entero de 1 byte en el rango de -128 a +127.

short: Enteros cortos de 2 bytes en el rango de -32768 a 32767.

int: Enteros de 4 bytes en el rango de -2147483648 a 2147483647

long: Enteros largos de 8 bytes en el rango de -9223372036854775808 a +9223372036854775807.

En general los valores numéricos se pueden expresar en decimal, hexadecimal, octal o binario. Para ello se usan lo siguientes códigos:

Decimal: 30

Octal: 036 (se antepone un 0)

Hexadecimal: 0x1E ó 0x1e (se antepone 0x)

Binario: 0b1110 (se antepone 0b **sólo válido desde Java 7**)

Si se desea forzar un número como entero largo se le puede poner una L al final.

float: Reales de 4 bytes según el estándar IEEE 754.

Cubre el rango de 1.40129846432481707e-45 a 3.40282346638528860e+38 tanto positivos como negativos.

Los literales con decimales en java por defecto son double, esto significa que al realizar esta operación:

```
float b=2.3;
```

obtenemos error de compilación. Para especificar que un número es float se le puede poner una F ó f al final.

```
float b=2.3F; // esto ya no da error
```

double: Reales de 8 bytes según el estándar IEEE 754.

Cubre el rango de 4.94065645841246544e-324 a 1.79769313486231570e+308 tanto positivos como negativos.

Si se deseara usar decimales con una gran precisión, no debe usarse ni *double* ni *float* si no la clase *java.math.BigDecimal*.

*Nota: desde **Java 7** se permite la separación de cifras mediante _. Por ejemplo se podría poner 666_123_456 como número de móvil por claridad y el programa entenderá 666123456.*

boolean: Valor lógico que implica verdadero o falso, por lo que toma los valores *false* o *true*. Teóricamente ocupa 1-bit pero no está especificado y puede ser dependiente de la máquina virtual. Se debe tener cuidado no usar como valores *True* y *False* o *TRUE* y *FALSE* que también existen pero no son tipo *boolean*.

char: Caracter unicode de 2 bytes que cubre de 0 al 65535 o expresados en modo caracteres de '\u0000' a '\ uffff' el código unicode está en hexadecimal.

En general un char suele contener algún símbolo que aparece en nuestro teclado y la representación más simple es dicho símbolo (letra, número, ...) entre comilla simple: 'a' , 'A', '7', '$', '@', ...

Se debe tener en cuenta que las minúsculas y mayúsculas son caracteres distintos ya que les corresponde distinto código unicode. Por ejemplo 'a' tiene el código 97 y 'A' el 65.

También hay que tener cuidado con los dígitos ya que el '7' no tiene código 7 si no que tiene código 55. Y además en un *char* sólo cabe un dígito. Por ejemplo '71' sería una incorrección.+

Además existen “secuencias de escape” que permite introducir ciertos caracteres que tienen un significado particular. Son los siguientes:

```
\t    tabulador
\n    linea siguiente
\"    comilla doble, "
\'    comilla simple, '
\\    backslash, \
```

String: A pesar de ponerlo aquí no es un tipo primitivo (por eso su primera letra va con mayúscula, porque es una clase) y permite representar cadenas de caracteres (textos) de cualquier longitud. En el caso que deseemos concatenar varias frases se puede usar el operador de concatenación +.

> *Actividad: Veamos como se puede mezclar los caracteres de escape en un String. ¿Qué sale en pantalla si se escribe el siguiente String?*
>
> *"S\u00ED se\u00F1or:\nPunto\t1"*
>
> *Pruébalo usando System.out.println.*

Existe en Java los equivalentes a los tipos primitivos (int, float, ...) pero en formato clase, es decir, su nombre

empieza por mayúscula (Byte, Float, Double,...) y tienen otras características. Son las llamadas clases wrapper y por el momento no las utilizaremos por lo que no debes confundirte al escribir los nombres.

Variables y constantes definidas por el programador.

Vimos ya que una variable es un identificador que da nombre a una zona de memoria donde se guarda un dato de determinado tipo. Precisamente como el nombre de las variables las da el programador antes de usar una variable hay que declararla. Esto implica que se le da un nombre, que se dice el tipo que se va a guardar en la misma y que el compilador va a reservar memoria para un dato de ese tipo. La declaración tiene este esquema:

```
Tipo_de_dato nombre_variable1;
```

Se pueden declarar varias variables del mismo tipo en la misma línea. Así estos ejemplo son válidos:

```
int a, num;
String nombre, direccion;
float temperatura;
```

Una variable sólo se declara una vez, a partir de ese momento se puede usar en el programa. La operación más básica con una variable es la asignación que consiste en guardar un dato en la variable. Se usa el símbolo = para realizar dicha operación.

```
a=0b11101;
nombre="Curro";
direccion="c/Príncipe 17";
temperatura=12.5F;
System.out.println(a);
System.out.println("Hola "+nombre+", en "+direccion+
            " hay una temperatura de "+temperatura+"°");
```

Actividad: Ejecuta el programa anterior y comprueba que lo entiendes todo.

Cuando se declara una variable es posible darle un valor al mismo tiempo. Es decir, estas líneas son válidas:

```
int a=23, num=0;
String nombre="Curro";
char bo='\u30dc'; //Caracter katakana japonés
```

Es posible obtener las denominadas constantes definidas por el programador. Son similares a las variables pero su valor no puede cambiar a lo largo del programa. Se declaran y se inicializan al mismo tiempo anteponiendo la palabra reservada final.

```
final char bo='\u30dc';
final double e=2.7182818284;
```

Está claro que no se le puede dar otros valores en otras partes del programa.

Operaciones aritméticas y de bit

Una operación es la realización de algún cálculo a partir de variables y constantes. El resultado de dicha operación tiene que tener algún destino: almacenarlo en una variable o mostrarlo por pantalla son dos ejemplos de lo que se puede hacer con el resultado de una expresión.

Para realizar operaciones con números disponemos de los siguientes operadores:

```
+ Suma
- Resta
* Multiplicación
/ División (cociente si los operandos son enteros)
% Resto de la división
```

De esta forma se puede hacer una operación como esta:

```
(12*n+dato)/3;
```

Pero en Java no puede existir esta línea porque sí, ya que su resultado se perdería. No es una sentencia válida. Veamos como ejemplo las dos posibilidades comentadas antes.

```
res=(12*n+dato)/3; //Almacenar resultado en la variable res
```

ó

```
System.out.println((12*n+dato)/3);  //Mostrar el resultado por pantalla
```

El primer caso interesa cuando el resultado va a usarse para más cosas. T

Lo que es muy importante resaltar es que una expresión **no es una ecuación**. No hay nada que despejar. Cuando aparece una asignación esta no es el igual de matemáticas si no que implica un movimiento de datos. Se opera con lo que hay a la derecha de la asignación (variables y constantes **de valores ya conocidos**) y cuando el cálculo está hecho se vuelca en la variable de la izquierda.

Así nunca vamos a tener expresiones como estas pues no tienen sentido:

res+5=n-23;

10=2*n+1;

> *Actividad: Comprobar los errores que da el compilador cuando se usan incorrectamente las expresiones.*

También tenemos los siguientes operadores que actúan a nivel de bit tal cual se conoce la lógica binaria (Algebra de Boole):

```
& and
| or
^ xor
~ not
a >> b desplaza a b bits a la derecha
a << b desplaza a b bits a la izquierda
```

Una operación válida sería:

```
resultado = 0x1F5 & 0xF00;
```

Conversión de tipos

En ocasiones es necesario realizar conversión de tipos de datos para guardar un dato en una variable distinta a la original. Por ejemplo puede darse el caso de tener un número como *String* y no podemos operar con él. En ese caso tenemos que convertirlo a entero o real para operar.

> *Actividad: Haz un programa en el que inicializas un String con el dato "231" y muestre ese dato más 5 ¿Cuál es el resultado?*

Hagámoslo ahora con la conversión indicada:

```
String dato="231";
int num=Integer.parseInt(dato);
System.out.println(num+5);
```

Otras conversiones de *String* a:

```
byte: Byte.parseByte(str)
double: Double.parseDouble(str)
float: Float.parseFloat(str)
int: Integer.parseInt(str)
long: Long.parseLong(str)
short: Short.parseShort(str)
```

siendo *str* siempre una variable o constante tipo *String*.

Para la conversión inversa (numérico a cadena) llega con sumar la cadena vacía al número o variable numérica.

```
dato=""+num;
```

Para conversiones entre tipos numéricos no se usan comandos si no una técnica denominada **casting** que consiste en indicar a que tipo de dato se convierte en la asignación. Veamos un ejemplo:

```
double n1=12.65;
int n2;
n2=n1;
System.out.println(n1+"\n"+n2);
```

si pruebas lo anterior da un error. Sin embargo cambiando la tercera línea con un casting a entero se soluciona:

```
n2=(int)n1;
```

Programación interactiva (entrada/salida de datos)

Resumamos brevemente los comandos vistos para mostrar datos y pedir datos al usuario.

Salida

Para mostrar datos usaremos habitualmente

```
System.out.println();
System.out.print();
```

La diferencia es que el primero introduce al final un retorno de carro, cosa que no hace el segundo.

Para mostrar varios datos simplemente se concatenan.

```
System.out.println("Valores de Pi");
System.out.println("-------------\n");
System.out.print("El numero Pi es "+ Math.PI);
System.out.print(" y su doble es "+ 2*Math.PI);
```

En ocasiones se necesita dar un formato especial a los datos de salida. Por ejemplo puede interesar mostrar solo dos decimales de cierto número o que un dato ocupe un número determinado de casillas. Para ello se usa la función *System.out.printf*. Veámoslo con un ejemplo:

```
int n1=100;
System.out.println("Valores de Pi");
System.out.println("-------------\n");
System.out.printf("El numero Pi es %.2f  y su doble es %.4f\
n",Math.PI, 2*Math.PI);
System.out.printf("%d\n%4d\n%10d\n", n1, n1, n1);
System.out.printf("%04d\n%010d\n", n1, n1, n1);
```

Después del carácter de formato (%d, %f) si se pone un valor numérico se refiere a la cantidad de casillas que ocupará el dato (Rellena con ceros si se antepone 0). Si se usa un punto y un valor numérico en el caso del %f se indica los decimales.

Algunos caracteres de formato que podemos encontrar son:

```
%f: Para valores reales.
%d: Para valores enteros
%c: para char
%s: para String
%o: Muestra en octal
%h: Muestra en hexadecimal
```

Se pueden ver todos en:

http://www.java2s.com/Code/JavaAPI/java.lang/System.out.printf.htm

Entrada

Para pedir datos al usuario usaremos la clase Scanner. Para ello hay que realizar la importación:

```
import java.util.Scanner;
```

Luego declarar la variable tipo *Scanner* que representa al teclado (denominado en Java *System.in*).

```
Scanner sc= new Scanner(System.in);
```

A partir de esta variable (realmente es un objeto) tenemos varios comandos dependiendo del dato que queramos leer. Algunos de ellos son:

```
nextInt()
nextLong()
nextDouble()
nextLine()
```

Cada uno se queda esperando a que el usuario meta un dato e intenta convertirlo al tipo especificado. En el caso de *nextLine()* se obtiene un String.

Esto puede provocar un problema. Prueba el código siguiente:

```
int num;
String nombre;

System.out.println("Introduce un numero");
num=sc.nextInt();
System.out.println("Introduce un nombre");
nombre=sc.nextLine();

System.out.println("Nombre: "+nombre+"\nNúmero: "+num);
```

Comprobarás que no deja meter el nombre. Esto se debe a que el nextInt() "ha recogido" el numero pero no el retorno de carro que pulsó el usuario. Esto provoca que el nextLine() recoja dicho retorno de carro previo y ya no espere a que el usuario meta algo.

Solución 1: Leer el retorno de carro después del número.

```
System.out.println("Introduce un numero");
num=sc.nextInt();
sc.nextLine(); // Esta línea lee el retorno de carro
System.out.println("Introduce un nombre");
nombre=sc.nextLine();
System.out.println("Nombre: "+nombre+"\nNúmero: "+num);
```

Solución 2: Leer todo con nextLine() para recoger el retorno de carro y luego hacer la conversión con parseInt().

```
System.out.println("Introduce un numero");
num=Integer.parseInt(sc.nextLine()); // lee cadena y lo
convierte a integer
System.out.println("Introduce un nombre");
```

```
nombre=sc.nextLine();
System.out.println("Nombre: "+nombre+"\nNúmero: "+num);
```

Puedes ver esta y otras formas de realizar entrada de datos en el siguiente enlace:

http://www.codejava.net/java-se/file-io/3-ways-for-reading-input-from-the-user-in-the-console

Ejercicios

1. *Haz un programa que pida al usuario dos números pero que los guarde en sendas variables tipo String. El programa realizará la suma de dichos números y mostrará el resultado por pantalla en decimal y en hexadecimal.*
2. *Escribe un programa que tenga una única instrucción System.out.print y con ella debes escribir en pantalla lo siguiente: Tu nombre en una línea, tus apellidos en la siguiente con una tabulación, tu edad entre comillas dobles (la misma que usas para Strings) en la tercera línea y en la cuarta las letras griegas Ω Φ Σ usando sus códigos Unicode (Busca el código unicode en el mapa de caracteres del sistema operativo).*
3. *Escribe un programa que pida al usuario su nombre, la edad (entero) y la temperatura actual (real). Debe mostrar luego una frase con los tres datos y además la temperatura debe mostrarla con 3 decimales.*
4. *Realiza un programa en el que se pida al usuario dos números reales y muestre por pantalla el resultado de la suma, la resta, la multiplicación y la división de los mismos. En el caso de la división el resultado se ha de mostrar siempre con 1 decimal.*

   ***(Avanzado)** Realiza además las siguientes operaciones:*

   - *Indica también si los números introducidos son múltiplos entre sí.*
   - *Añade las operaciones AND y OR a nivel de bit entre ambos números.*
   - *Busca la forma de hacer (aún no lo vimos: estructura selectiva) que si se produce una división entre cero que informe del problema al usuario.*
   - *Finalmente haz que indique al usuario que introduzca 1 para repetir el programa u otro número para terminar. En caso de pulsar 1, lógicamente, el programa empezará de nuevo (Tampoco lo hemos visto aún: estructura repetitiva).*

Apéndice: Otras definiciones

Java ME: Micro edition (dispositivos móviles). Reducido y muy optimizado para equipos con bajos recursos.

Hay otros: Java Card (tarjetas tipo SIM y otros dispositivos pequeños), JavaTV (Video digital), etc...

Se puede ver toda la familia en: http://www.oracle.com/technetwork/java/javase/overview/index.html

Servlets: Componentes de un servidor para servir peticiones (en Java EE)

JSP: Java Server Pages: Tecnología similar a los servlets pero más cómoda a la hora de generar páginas web dinámicamente del lado del servidor. Viene incluido también en el Java EE y requiere, lógicamente, de un servidor capaz de gestionar las JSP. Es una alternativa al ASP o al PHP.

MIDlet: Aplicación similar al applet pero realizada con el Java ME.

JavaBean: Componente Java.

EJB: Enterprise JavaBean. API de componentes para la construcción de aplicaciones empresariales del lado del servidor incluido en Java EE.

Java Web Start: Tecnología que permite a una aplicación en cuanto se ejecuta a comprobar su versión y si no es la más reciente, se conecta con un servidor, se baja la más reciente y se ejecuta en local de forma automática.

SWT: Standart Widgets Toolkit. Sistema gráfico que busca la eficacia y mejora de Swing pero a su vez intenta que en cada plataforma se vean las aplicaciones como nativas. Esto provoca que las SWT son distintas para cada plataforma. Además SWT no pertenece a Oracle, está desarrollado por el grupo de Eclipse y lleva un ritmo distintos y además funciona de forma más o menos eficiente dependiendo de la plataforma.

JIT: Just In Time compiler. Para compilar a código máquina nativo y que la aplicación sea más eficiente.

JVM: Máquina virtual de Java. Programa que emula un procesador virtual que hace que el lenguaje Java sea multiplataforma. Se profundizará sobre esto en otro módulo del ciclo.

Fuentes:

Wikipedia

Java for the Beginning Programmer. Jeff Heaton. Heaton Research Inc.

Java for programmers. Paul&Harvey Deitel. Prentice Hall

Oracle: http://www.oracle.com/technetwork/java/index.html

http://www.cafeaulait.org/course/

http://www.java2s.com/Code/JavaAPI/CatalogJavaAPI.htm

Aprende Java2 como si estuvieras en primero.

Java 2. Apuntes de Abraham Otero (Javahispano)