1.
Introducción al Lenguaje JAVA

1. INTRODUCCIÓN AL LENGUAJE JAVA

- Características del Lenguaje Java.
- Hablar de las ventajas que ofrece la utilización de Java.
  - Lenguaje orientado a objetos.
  - Independencia de la Plataforma.
- Hablar de los códigos BYTE (bytecode).
- Concepto de Máquina Virtual Java.
- Tipos de Aplicaciones que se pueden crear con java:
  - Aplicaciones Java independientes (Stand Alone).
  - Aplicaciones de cliente. Applets.
  - Aplicaciones de Servidor. Servlets. (No se ven en el curso).
  - Otros elementos auxiliares.
- Utilización de las herramientas de JDK para:
  - Compilar (javac nombrepgm.java).
  - Ejecutar programas independientes (java nombrepgm.class).
  - Ejecutar Applets (appletviewer nombrepgm.htm).
  - Funcionamiento de las variables de entorno: class y classpath.

    classpath = .;c:\jdk1.3\lib\classes.zip

    path= .;c:\jdk1.3\bin
  - Ayudas sobre el Api de Java en:

    C:\jdk1.3\docs\api\index.html

    También en la web de Java:

    http://java.sun.com/j2se/1.3/docs/api/

MI PRIMER PROGRAMA EN JAVA

- **PRÁCTICAS**: Hacer el clásico programa de saludo, "**HolaMundo**".

- Y después de hacerlo. ¿Qué hay que conocer a este nivel?:

  - Que todos los pgms. están dentro de una CLASE.

  - Explicar brevemente el concepto de:
    - Clases.
    - Métodos y datos miembros.
    - Objetos. Crear objetos a partir de una clase.
    - Funcionamiento del método MAIN (Introducción a los argumentos de main).
    - Significado de los modificadores de acceso: PUBLIC y STATIC
    - Hablar de la clase System y de cómo se usa para enviar datos a salida (pantalla): **System.out.println("String")**

Introducción a la Programación Orientada a Objetos

Java es un lenguaje de programación orientado a objetos.

Los antiguos lenguajes de programación eran lineales por la que localizar errores en ellos era una tarea complicada cuando el código aumentaba, de ahí se pasó a la programación orientada a objetos.

Los objetos son unas entidades que tienen unas características (datos, atributos) y unas funciones determinadas (utilidades, tareas concretas, métodos).

En la p.o.o. cada problema se asocia a un objeto y uniendo todos esos objetos obtenemos las solución final. (Ej. Excel es un conjunto de objetos: el objeto botón, el objeto menú, el objeto celda...).

La p.o.o. se basa en ir creando diferentes objetos que vayan solucionando problemas y a partir de ahí los interrelacionamos para hacer la aplicación.

En Java ya existen objetos creados que junto con las que nosotros creamos desarrollaremos la aplicación. Precisamente lo complicado no es utilizar los objetos (clases) ya creados sino crear los nuestros propios.

Hay que hacer una distinción entre clase y objeto:

- Mientras un objeto es algo creado con características y unas funciones una clase es el molde a partir de la cual creamos dichos objetos.

- El código que define el comportamiento y características de los objetos esta definido dentro de las clases que se utilizan como base para crear esos objetos.

```
String   miString1="El Java me gusta";
String   miString2="El Java me gusta cada día más";
```

String = clase siempre la misma (molde para crear objetos).
miString1/2= objetos físicos creados (son diferentes entre si).

Tipos de programadores/usuarios en Java:

- Los sabios de Java: crearon las clases primeras.
- Programador: Crea aplicaciones basadas en esos objetos.
- Usuario: Utiliza esas aplicaciones.

Características del lenguaje Java.

- Lenguaje orientado a objetos.
- Portabilidad.
- Lenguaje seguro.

- **Lenguaje orientado a objetos**: Java es un lenguaje orientado a objetos semejante a c++.
- **La portabilidad** consiste en que un programa hecho en Java puede utilizarse en cualquier plataforma independientemente del sistema operativo. Por ese en tan importante Java, porque además está muy orientado a internet (arquitectura Java). El hecho de que los programas de Java sean interpretados, hace que se puedan ejecutar en una gran variedad de entornos, pues solo es necesario que la plataforma incluya el interprete de Java.
  En el mundo internet es fundamental tener programas en las que no tengamos que estar pensando en que plataforma (sistema operativo) se va a utilizar.

- **Seguridad**. Cuando se compila un programa Java, la salida del compilador no genera código ejecutable, sino código binario o **bytecodes independientes de la arquitectura**. Estos bytecodes únicamente pueden ser interpretados por un interprete de Java, el cual controlará la ejecución del Applet y evitará que provoque efectos no deseados. Por las características propias del lenguaje es muy difícil realizar violaciones de memoria (Ej. Punteros)

La dificultad de crear un programa java reside en pensar que clases utilizar y que clases crear, ya que todo programa en difinitiva en una clase.

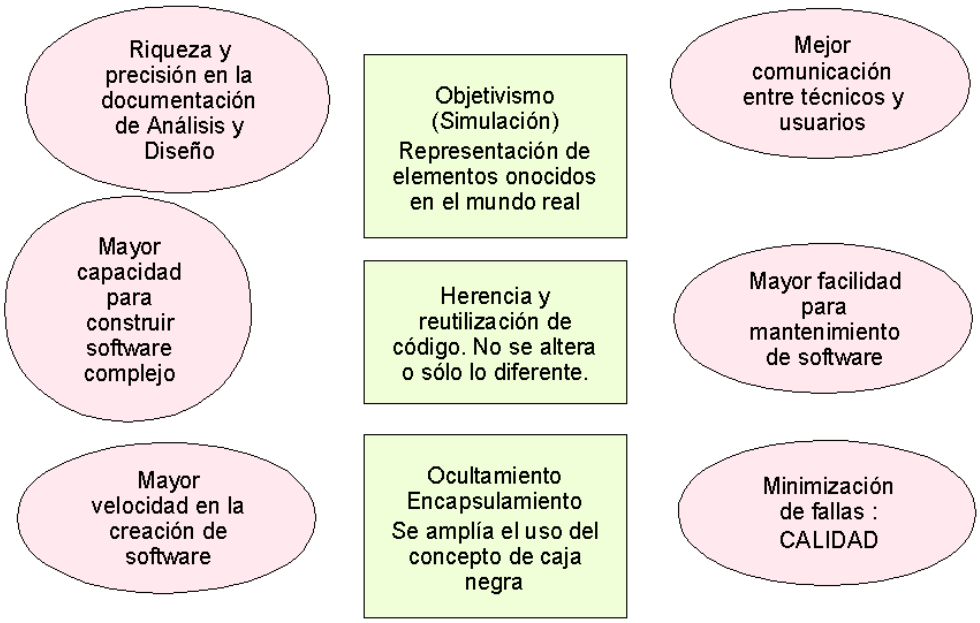
Tres son los tipos de programas que pueden crearse en Java:

- **APLICACIONES CLIENTE:**
  - **Aplicaciones Independientes**. Una aplicación es un programa que se ejecuta en la computadora, utilizando el sistema operativo de esta. Cuando a Java se le da esta utilidad no difiere de la utilización de otro leguaje.
  - **Applets**. Un Applet es una aplicación diseñada para ser transmitida por Internet y ejecutada en un navegador web compatible con Java. Como cualquier aplicación, el Applet es capaz de reaccionar ante las acciones del usuario y cambiar dinámicamente. La creación de Applets es la principal utilidad de Java. Las applets son mucho más potentes que los "scrips" de cliente.

- **APLICACIONES DE SERVIDOR :**
  - Servlets, JSP, Javabeans EJB. Son las aplicaciones del lado del servidor y sirven para dar soporte a las aplicaciones cliente.
  -

Programación Orientada a Objetos

Las ventajas



Programación Orientada a Objetos

Clases y Objetos

Programación Orientada a Objetos

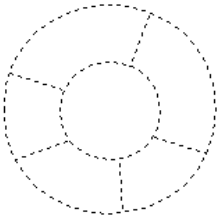
Encapsulamiento

Un objeto oculta sus datos y los mecanismos de su funcionamiento.

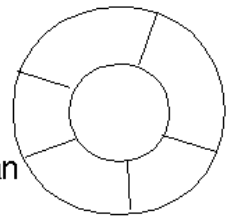
El mundo real

En OOP

CLASE
define datos privados y métodos privados y públicos



OBJETO
sus datos están protegidos, sólo sus métodos los deberían acceder

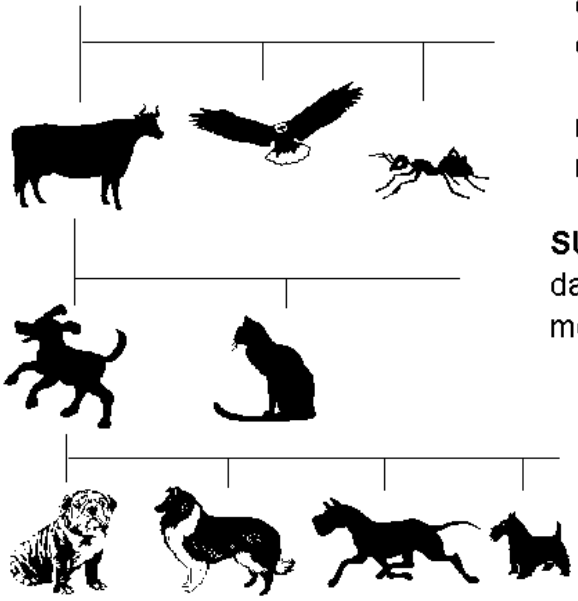


Programación Orientada a Objetos

Herencia

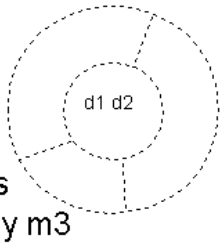
Propiedad que hace que una clase tenga de manera implícita las características de su "superclase" (clase superior en la jerarquía de clasificación)

El mundo real



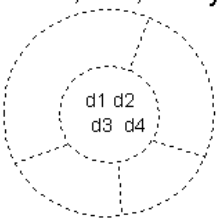
En OOP

CLASE
datos
d1 y d2



métodos
m1, m2 y m3

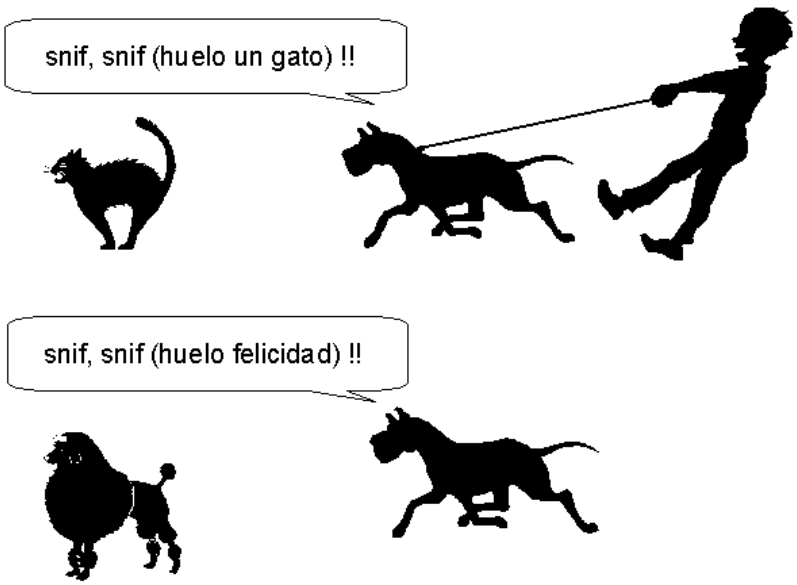
SUBCLASE
datos d1, d2, d3 y d4
métodos m1, m2, m3 y m4



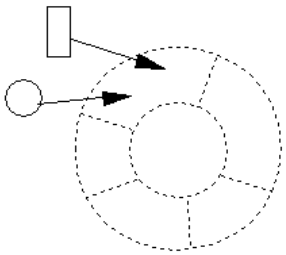
Programación Orientada a Objetos

Polimorfismo (en clases)

El mundo real ...



En OOP

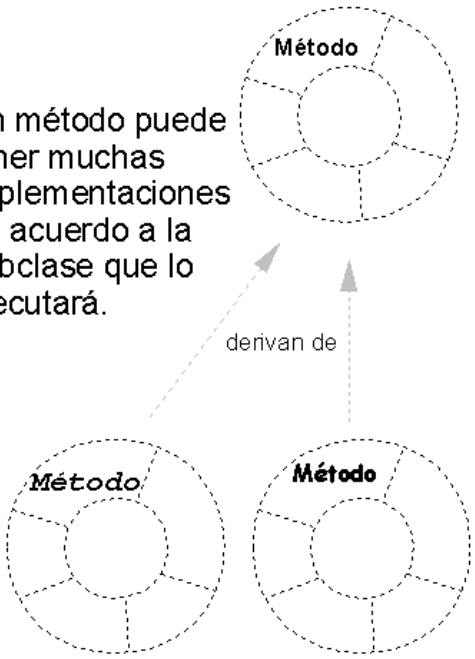


Un método puede tener muchas implementaciones que se seleccionan de acuerdo a qué tipo de objeto se le está comunicando.

Programación Orientada a Objetos

Polimorfismo (en subclases)

En OOP



Java como lenguaje de programación.

Clases, objetos.

Java es un lenguaje de programación totalmente orientado a objetos, por lo que **la base del lenguaje lo constituyen las clases**. De hecho, todo código escrito en Java debe estar dentro de una clase.
Una clase es un conjunto de datos y código que describe el comportamiento de los objetos de esa clase. Se puede decir que la clase es el molde a partir del que se crea el objeto.

Datos miembro y métodos.

Los objetos poseen características, que son implementadas en la clase mediante datos (llamados **datos miembro**), estos datos no son accesibles directamente desde el exterior, para que las características del objeto puedan ser establecidas y modificadas por el programa que utiliza el objeto, la clase proporciona una serie de procedimientos (**métodos**) que dan acceso a los datos, de forma que estén protegidos de accesos indebidos o indeseados.
Las clases, proporcionan otros métodos que no se limitan a meros intermediarios entre el exterior y los datos miembro si no que poseen funcionalidad propia y realizan determinadas acciones.

Estructura de un programa Java.

Todo programa en Java debe tener al menos una clase. Es común escribir cada clase en un archivo fuente, los archivos fuente tiene extensión **.java**.
La sintaxis para implementar una clase en Java, con sus datos miembro y métodos correspondientes, es la siguiente:

```
[public] class  NombreClase
{
        private tipo  dato1;
        private tipo  dato2;
                :
        [public] tipo  metodo1( parametros )
        {
                // código método1
        }
        [public] tipo  metodo2( parametros )
        {
                // código metodo2
        }
        :
}
```

Primer programa en Java.

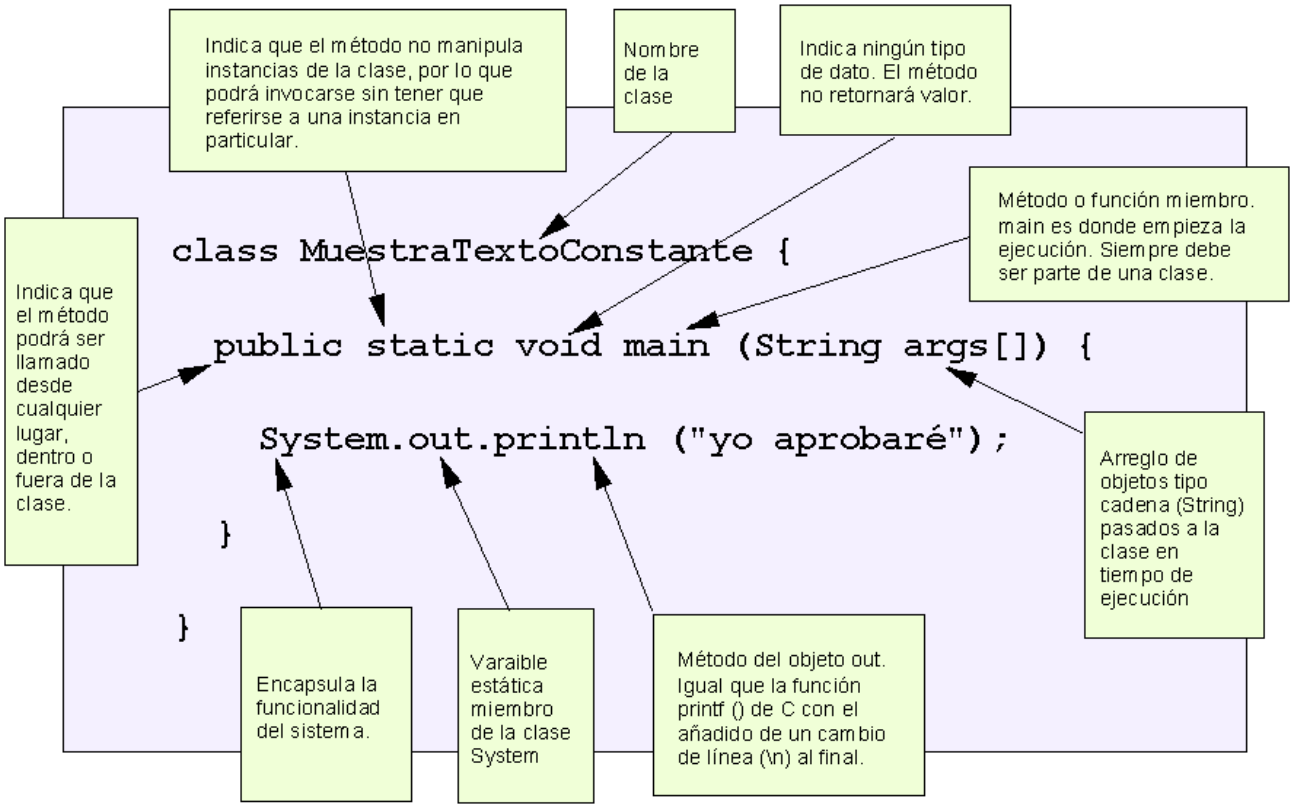
El programa más básico que se puede escribir en Java es aquel que presenta un mensaje en la consola. El código fuente será el siguiente:

```
public class PrimerPrograma
{
        public static void main (String [ ] args )
        {
                System.out.println ("Bienvenido al mundo de JAVA");
        }
}
```

Método main.

Toda aplicación Java (no Applet) debe contar con una clase que implemente un método llamado **main**. La Máquina Virtual ejecutará este método para iniciar el programa, representa por tanto el punto de inicio de la aplicación.

El parámetro **args** es un array de variables de texto que contienen los argumentos de la línea de comandos.



Modificadores.

La palabra main va precedida por los siguientes modificadores:

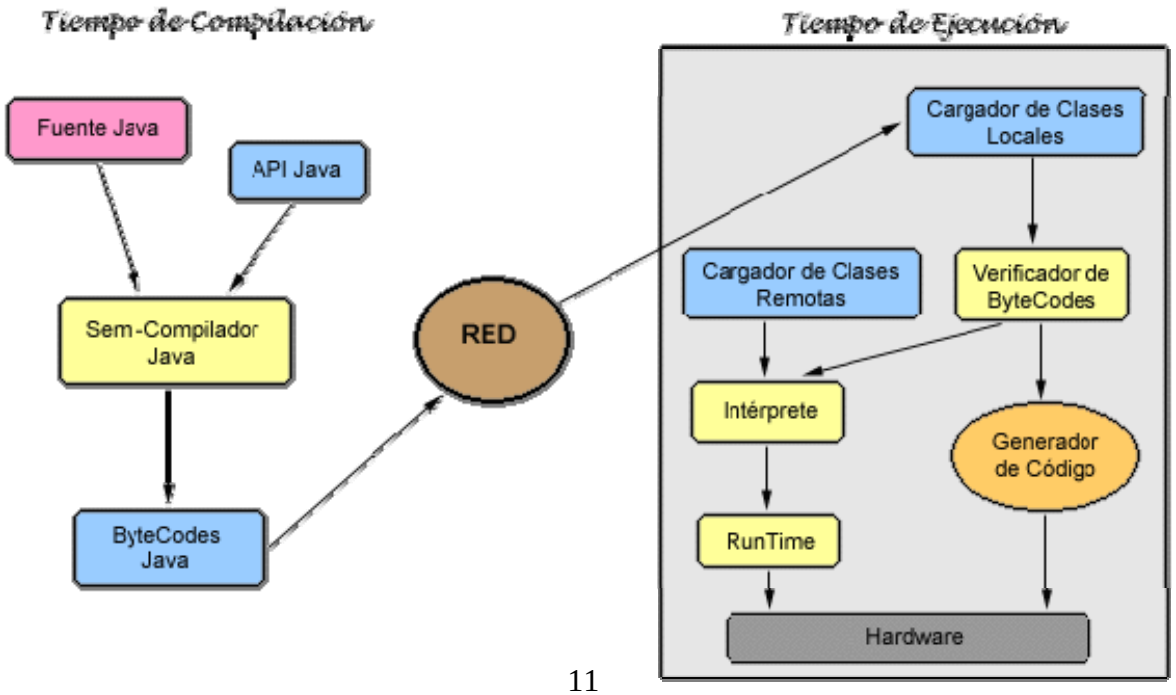
- **public**: modificador de acceso que permite que el método pueda ser llamado por código de fuera de la clase.
- **static**: indica que el método es estático, lo que significa que está asociado con la clase y no con un objeto particular. Todas las llamadas que se realicen al método utilizarán los mismos datos.
- **void**: indica que el método no devuelve ningún valor.

LA MÁQUINA VIRTUAL DE JAVA:

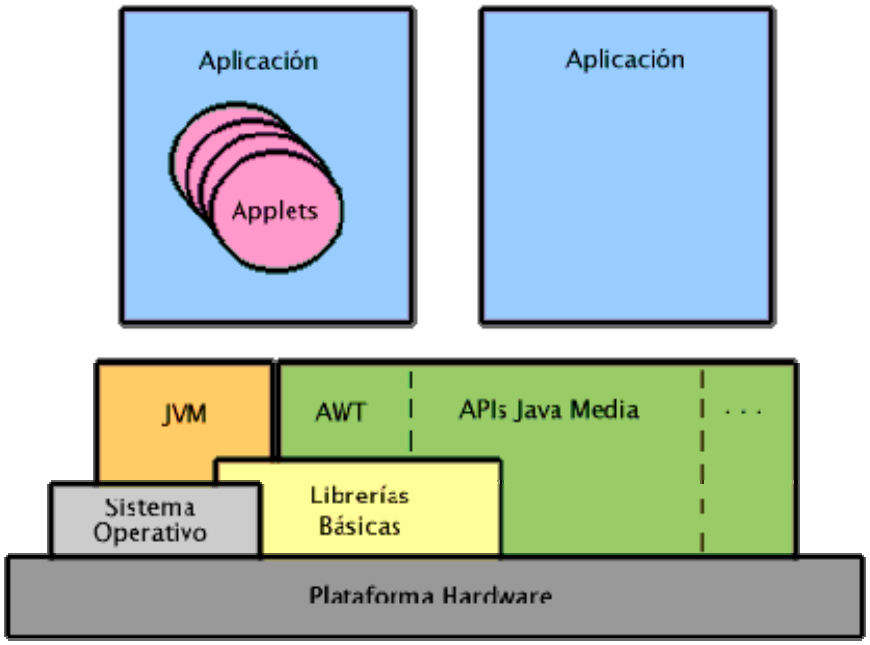
La máquina Virtual le permite a Java tener una **ARQUITECTURA NEUTRAL.**
Está intimamente relacionada con la portabilidad de java.

Todo programa Java se puede ejecutar en cualquier máquina gracias a que en cada equipo existe lo que se conoce como la MÁQUINA VIRTUAL DE JAVA. Visitar página http://java.sun.com/j2se/1.4/docs/api.

Para establecer Java como parte integral de la red, el compilador Java compila su código a un fichero objeto de formato independiente de la arquitectura de la máquina en que se ejecutará. Cualquier máquina que tenga el sistema de ejecución (run-time) puede ejecutar ese código objeto, sin importar en modo alguno la máquina en que ha sido generado. Actualmente existen sistemas run-time para Solaris 2.x, SunOs 4.1.x, Windows 95, Windows NT, Linux, Irix, Aix, Mac, Apple y probablemente haya grupos de desarrollo trabajando en el porting a otras plataformas.



En una representación en que tuviésemos que indicar todos los elementos que forman parte de la arquitectura de Java sobre una plataforma genérica, obtendríamos una figura como la siguiente:



El código fuente Java se "compila" a un código de bytes de alto nivel independiente de la máquina. Este código (**byte-codes**) está diseñado para ejecutarse en una máquina hipotética que es implementada por un sistema run-time, que sí es dependiente de la máquina.

Para programar en JAVA existes diferentes entornos de programación.

Cada fabricante de software ha creado su propio entorno de desarrollo de programas, que permite crear el código fuente de los programas, compilarlos, ejecutarlos, etc.

Así SUN Microsystem (creador de Java, http://java.sun.com/j2se/1.3/docs/) diseñó el entorno de programación JDK que no posee entorno gráfico. Algunos de los entornos más conocidos son:

- JDK 1.2, JDK 1.3, JDK 1.4 … … …
- PcGRASP
- JDeveloper
- JBuilder
- Visual J++
- Visual AGE
- Kawa
- Forte4i etc, etc...

Ejemplo:

```
// package fprgJAVA.t01;

// La clase HolaATodos simplemente
// dice "HolaATodos..." por la salida estandar

class HolaATodos {

    public static void main (String[] args) {

        System.out.println ("HOLA A TODOS BIENVENIDOS AL MUNDO DE JAVA!");
    }
}

/**************************************************************************
Instrucciones de configuracion del entorno java
-----------------------------------------------
Habra notado la sentencia  ====>         package    fprgJAVA.t01
                                                    en la cabecera del programa.

      Esto nos va a servir para estructurar todos los ejmplos del curso. NOS PERMITIRIA
      QUE CUANDO COMPILAMOS LOS FICHEROS .CLASS  QUEDEN EN ESE DIRECTORIO C:\fprg\t01


A continuacion le indicamos como SE PODRIAN ORGANIZAR sus ficheros DE PRACTICAS:

1. se elige un directorio para guardar los ( .java ) y los  ( .class )
    C:\fprgJAVA
   y un subdirectorio por tema
    C:\fprgJAVA\t01
   en el que est n los fuentes:
    C:\fprgJAVA\t01\HolaATodos.java

2. se mete dicho directorio en la variable de entorno CLASSPATH
    normalmente, en C:\autoexec.bat
    ejemplo:
      set PATH=%PATH%;C:\jdk1.3\bin
      set CLASSPATH=.;C:\;C:\jdk1.3\lib\classes.zip

3. COMPILAMOS UN PROGRAMA DE JAVA:
    javac   HolaATodos.java          <====  NO OLVIDAR LA EXTENSION ( .JAVA)
   y vemos que los .class aparecen en el mismo directorio
    C:\fprgJAVA\t01\HolaATodos.class

4. al EJECUTAR se indica el nombre del paquete
    java fprgJAVA.t01.HolaATodos      o desde el MISMO DIRECTORIO DONDE ESTA EL ( .CLASS)

**************************************************************************/
```

Configuración del JDK.

Para la configuración del jdk una vez realizada su instalación es necesario establecer en el sistema operativo donde se va ejecutar la variable de entorno PATH y con versiones anteriores del JDK 1.3 la variable de entorno CLASSPATH.

En un entorno **windows 95/98** estas variables se estableceran en el fichero **autoexec.bat**, en un entorno unix/linux en el fichero **.profile** y en un entorno windows NT en cuadro **entorno** de **sistema**.
En **windows NT/2000** , se accede a estas variables desde:

MiPc(botón derecho)/Propiedades/Avanzado/variables de entorno/edit

Variable de Entorno PATH

Esta variable apunta al directorio donde se encuentran los ejecutables del JDK como puede ser el compilador, interprete java, depurador, etc.

Este directorio es el directorio **/bin** que se encuentra en el directorio de instalación del jdk. En este ejemplo se muestran dos lineas extraidas del **autoexec.bat** con entornos jdk instalados:

Con un entorno jdk 1.2:

SET PATH=.;C:\WINDOWS;C:\WINDOWS\COMMAND;C:\;C:\DOS;**C:\JDK1.3\BIN**

⇧

Variable de Entorno CLASSPATH

Este variable de entorno es de uso obligatorio en entornos jdk1.x.x y apunta al directorio donde se localizan las clases java.

Opcionalmente tambien pueden apuntar a directorios donde se encuentran clases no incluidas con el jdk como pueden ser las clases del entorno jsdk (Servlets y JSP), las clases para interactuar con el navegador de Netscape, etc.

En este ejemplo se muestra la definicion de la variable de entorno classpath típica con un entorno jdk 1.x.x :

SET CLASSPATH=.;C:\;C:\jdk1.3\lib\classes.zip;C:\MisClases*.jar

⇧

API de java en dusco duro: C:\jdk1.3\docs\api\index.html
API de java en INTERNET: http://java.sun.com/j2se/1.3/docs/api/

ESTRUCTURA DE LOS PROGRAMAS EN JAVA:

Ya hemos visto nuestro primer programa en java. ¿Qué destacar?

Un programa Java se estructura en clases. Tendremos que crearnos al menos una clase.
CLASS (class) en una palabra clave de java que indica que vamos a crear una clase nueva. Todas las palabras reservadas de java se escriben en minusculas (JAVA es **casesensitive**). Dentro de una clase típica van las propiedades y los métodos. Si se quiere asignar un valor a una propiedad hay que hacerlo a traves de los métodos.
Los métodos siempre van dentro de una clase. Un método en realidad es una función que hace algo.

Así pues, en una clase existirán **variables** y **métodos**. Los métodos (son funciones) que se utilizan desde fuera de la clase para realizar acciones con los objetos.
Cuando utilizamos un objeto creado a partir de una clase, podemos acceder a sus métodos pero no a sus variables. Para poder manipular los valores de las variables propias de las clases se hará a traves de sus métodos.

A las variables de las clases se las conoce también como datos miembro (son las propiedades de los objetos):

borrador. color="verde"	En otros lenguajes.
Borrador.setColor("verde");	En java Cambiamos el valor de la propiedad a traves de un método que recibe el valor (que queremos asignar al dato miembro) como parámetro. El método controlará que el valor que se quiere asignar a la propiedad es correcto y si no no lo realizará.

En java todos los programas comienzan a ejecutarse por el método MAIN .
Cuando se quiere ejecutar un programa JAVA, la MAQUINA VIRTUAL de java hace una llamada al método MAIN de la clase, y a partir de ahí, le deja el control a la clase.

2.- Estructura de la Sintaxis del Lenguaje

2. ESTRUCTURA DE LA SINTAXIS DEL LENGUAJE

- El Lenguaje Java.
  - Variables
  - Tipos de datos.
  - Ambitos de utilización de las variables
  - Conversión de tipos: implicitas y explicitas (cast).
  - Conceptos generales sobre la Sintaxis de Java:
    - Comas, punto y coma, operador punto, llaves { }, etc...
    - Explicar que java es "case sensitive"
    - Tipos de comentarios
  - Operadores:
    - aritméticos,
    - relacionales
    - y lógicos.

Identificadores en Java

-Nombres para :

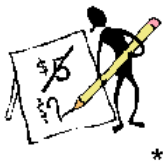
Variables : para almacenamiento de datos
Métodos : para especificación de funciones
Clases : para especificación de objetos
Interfaces : para especificación de protocolos
Paquetes : para agrupación de clases

-Mayúsculas y minúsculas

-1er caracter : letra
_ (caracter subrayado)
$ (dólar)

-siguientes : anteriores (letra, subraya y dólar) y dígitos (0 al 9)

Literales



- **Literales Enteros**
- **Literales de punto flotante**
- **Literales Booleanos**
- **Literales de un caracter ***
- **Literales de cadenas de caracteres**

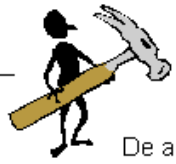
*** Literales especiales de un caracter**

Descripción	Representación
Barra invertida (backslash)	\\
Continuación	\
Retroceso (backspace)	\b
Retorno del carro (CR)	\r
Alimentación de página (FF)	\f
Tabulador (tab)	\t
Nueva línea	\n
Comilla simple	\'
Comilla doble	\"
Caracter Unicode	\udddd
Caracter Octal	\uddd

Identificadores reservados (keywords)

abstract	continue	for	new	if
boolean	default	goto	null	synchronized
break	do	swith	package	this
byte	double	implements	private	threadsafe
byvalue	else	import	protected	throw
case	extends	instanceof	public	transient
catch	false	int	return	true
char	final	interface	short	try
class	finally	long	static	void
const	float	native	super	while

Operadores



Aritméticos y de conversión			De comparación		Lógicos		De asignación
i,f	+	i	&	i,f	==	b	&	=
i,f	-	i	|	i,f	!=	b	|	+=
i,f	*	i	^	i,f	<	b	^	-=
i,f	/	i	<<	i,f	<=	b	&&	*=
i,f	%	i	>>	i,f	>	b	|	/=
i,f	++	i	>>>	i,f	>=	b	!	%=
i,f	--					b	==	&=
i	-					b	!=	|=
i	~					b	?:	^=

Tipos válidos de los operandos
i = Entero (byte, small, int, long)
f = Punto Flotante (float, double)
b = Lógico (boolean)

OPERADORES DE JAVA (continuación)

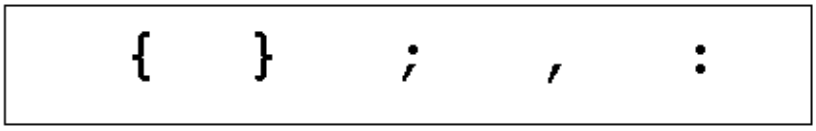
Descripción	Símbolo	Expresión de ejemplo	Resultado del ejemplo
Multiplicación	*	2*4	8
Resto de una división entera	%	5%2	1
Suma	+	2+2	4
Resta	-	7-2	5
Incremento	++	2++	3
Decremento	--	--2	1
Menos unario	-	-(2+4)	-6

Descripción	Símbolo	Expresión de ejemplo	Resultado del ejemplo
Igualdad	==	2==2	true
Desigualdad	!=	2!==2	false
Menor que	<	2<2	false
Mayor que	>	3>2	true
Menor o igual que	<=	2<=2	true
Mayor o igual que	>=	1>=2	false

Descripción	Símbolo	Expresión de ejemplo	Resultado del ejemplo
Negación	!	!(2==2)	false
Y lógico	&&	(2 == 2) && (2 >= 0)	true
O lógico	||	(2 == 2) || (2 != 2)	true

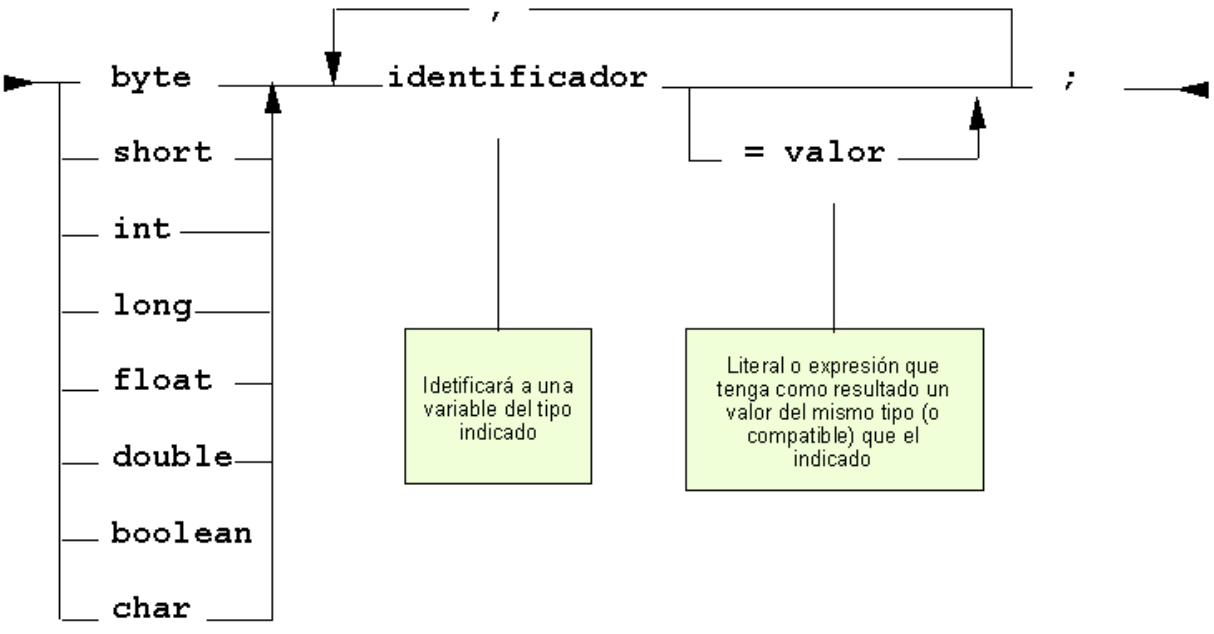
Separador	Descripción
()	Contienen listas de parámetros, tanto en la definición de un método como en la llamada al mismo. También se utilizan para modificar la precedencia en una expresión, contener expresiones para control de flujo y realizar conversiones de tipo.
{}	Se utilizan para definir bloques de código, definir ámbitos y contener los valores iniciales de los vectores.
[]	Se utiliza tanto para declarar vectores o matrices como para referenciar valores dentro de los mismos.
;	Separa sentencias.
,	Separa identificadores consecutivos en la declaración de variables y en las listas de parámetros. También se utiliza para encadenar sentencias dentro de una estructura `for`.
.	Separa un nombre de propiedad o método de una variable de referencia. También separa nombre de paquete de los de un subpaquete o una clase.

Separadores y Comentarios



```
/* comentarios al codigo del programa */
// comentarios hasta el final de la línea
/** comentarios a usarse por la facilidad de
    documentación  javadoc  */
```

Declaración de variables con tipos de datos primitivos



Tipo	Descripción	Tamaño	Clase equivalente
boolean	Valor lógico	1 bit	Boolean
char	Carácter	16 bit	Character
byte	Entero muy pequeño	8 bit	
short	Entero pequeño	16 bit	
int	Entero normal	32 bit	Integer
long	Entero grande	64 bit	Long
float	Número real de precisión simple	32 bit	Float
double	Número real de doble precisión	64 bit	Double
void	Tipo vacío		

Type	Size in bits	Values	Standard
boolean	8	**true** or **false**	
char	16	'**\u0000**' to '**\uFFFF**'	(ISO Unicode character set)
byte	8	–128 to +127	
short	16	–32,768 to +32,767	
int	32	–2,147,483,648 to +2,147,483,647	
long	64	–9,223,372,036,854,775,808 to +9,223,372,036,854,775,807	
float	32	–3.40292347E+38 to +3.40292347E+38	(IEEE 754 floating point)
double	64	–1.79769313486231570E+308 to +1.79769313486231570E+308	(IEEE 754 floating point)

Elementos del lenguaje.

Como todos los lenguajes de programación, Java cuenta con una serie de elementos y normas para construir los programas: VARIABLES, OPERADORES, ESTRUCTURAS DE CONTROL ,ARRAYS, etc.

Variables

- Una variable es una unidad básica de almacenamiento, en la que se puede almacenar un tipo definido por Java o un objeto de una clase.
- En Java las variables deben ser declaradas antes de utilizarlas:

```
tipo        nombre_variable [= valor] ;
```

ejemplos:

```
int  A;                 int B= 8;
int K, C, V=2, J=7;     boolean F=true;
char caracter= '@';     double D= 3.25f;
```

valor puede ser una constante o una expresión. Las variables globales a la clase (datos miembro) que no se inicializan explícitamente, son inicializadas a 0 , por otro lado, las variables locales deben ser siempre inicializadas explícitamente.

Una variable solo puede ser utilizada dentro del bloque en el que ha sido declarada, excepto los datos miembro, que pueden llevar el modificador de acceso public (no se recomienda) y ser utilizados desde fuera de la clase.
Una variable **no** puede tener el mismo nombre que otra de ámbito exterior.

Tipos de datos.

Se pueden dividir en cuatro grupos:

- **Enteros**: **byte** (8 bits), **short** (16 bits), **int** (32 bits) y **long** (64 bits).
- **Coma flotante**: **float** (32 bits) y **double** (64 bits).
- **Caracteres**: **char** (16 bit), representa caracteres **unicode**.
- **Lógicos**: **Boolean** (1 bit), admiten los valores **true** y **false**. No son compatibles con los numéricos.

Literales.

Un literal es un valor constante. Java sigue las siguientes normas respecto a los literales:

- Un **literal entero** es considerado como **int**, aunque, no existe ningún problema si este se asigna a un byte o un short si está dentro de su rango.
- Los **literales en coma flotante son considerados como double**, si queremos que sea considerado como float hay que añadir la letra **f** al final de la constante, pues no se puede asignar directamente el literal double a una variable tipo float (aunque esté dentro del rango):

```
float   f = 3.14f;          double  d = 3.14 ;
```

- Los literales booleanos son true y false, no pueden convertirse en ninguna representación numérica.
- Los literales tipo carácter se representan entre comilla simple '. Para los caracteres no imprimibles existen secuencias de escape (página 20).

1.-Números Enteros

En Java existen varios tipos de números enteros distinguiendose por el tamaño ocupado en memoria y por tanto el rango de valores que pueden contener.

Cuando declaramos sin darles un valor inicia cualquiera de estos tipos son inicializados al valor 0 de forma predeterminada.

Ejemplos:
```
int     var_a;
short   var_b=20;
long    var_c, var_d;
byte    var_e;
var_a = 015;        //Valor en octal
var_c = var_b * var_a;
var_d = 15L;
var_e = 0xDFA1;     //Valor en hexadecimal
```

Notar que al declarar un numero de tipo **long** se le añade el sufijo **L** no importando si este esta en mayúscula o minúscula.

2.- Numeros Reales con Coma Flotante

Cuando declaramos sin darles un valor inicia cualquiera de estos tipos son inicializados al valor 0.0 de forma predeterminada.

Ejemplos:
```
float     numero1 = 3.14F
double    numero2;
numero2 = 3.14d * numero1
```

Al igual que en los numeros de tipo **long** a los tipos **float** y **double** tambien se les añaden los sufijos **F** y **D** respectivamente no importando si esta en minúscula o mayúscula.

3.- Valores Booleanos

Toman valores de tipo booleano.

Ejemplos:
```
boolean   existe = false;
boolean   crear_otro;
crear_otro = true;
```

4.- Valores de tipo Caracter

Toman valores de tipo carácter (**char**). Su tamaño en memoria es de 16 Bits numero de bits con los que se representan los caracteres **unicode.** Los valores que van de oX0000h a oX00FFh representan los caracteres **ASCII**.

Ejemplos: char caracter1 = 'a'; char carácter2;
carácter2 = 'P';

Conversiones de tipos.

Cuando los tipos de datos son compatibles y el tipo de destino es más grande que el tipo de origen, Java realiza una conversión automática.

Conversiones automáticas.

Java realizará de forma automática las siguientes conversiones:
byte a cualquier numérico, short a todos los numéricos excepto byte, int a todos los numéricos excepto byte y short, long a float y double, float a double, char a todos los numéricos excepto byte y short.

Conversiones explicitas. (CASTING).

Cualquier conversión entre tipos numéricos y char es posible mediante la realización de un estrechamiento, esto se realiza utilizando el formato:

```
tipo  var_tipo_destino = (tipo_destino)  var_tipo_origen;
```

Ejemplos:
```
byte c;
float f=3.14;
c = (byte) f;
```

```
short  num=186;    //valores decimales  entre 0 y 255
System.out.print("Codigo(" + num + ") = Caracter " + (char) num + "\n");
```

```
System.out.println("VALOR= " + "A" );        se imprimirá ⇨ VALOR= A
System.out.println("VALOR= " + (int) 'A' );  se imprimirá ⇨ VALOR= 65
System.out.println("VALOR= " + (char) 65 );  se imprimirá ⇨ VALOR= A
```

El estrechamiento o casting puede provocar en algunos casos una pérdida de datos.

Promoción de tipos implícita.

Además de las conversión de tipos, Java define las siguientes reglas de promoción de tipos en operaciones aritméticas:

- Los tipos **byte y short se promocionan a int**.
- Si algún operando es long, la expresión completa se promociona a long.
- Si algún operando es float, la expresión se promociona a float.
- Si algún operando es double, la expresión se promociona a double.

- **Además de los operadores Aritmeticos, relacionales y lógicos existen otros operadores en java**:
  - ✓ **?** Equivale a la instrucción if ... else: **exp1 ? exp2:exp3**.
  - ✓ **instanceof** comprueba si una variable objeto contiene una instancia de una clase específica.
    variable instanceof nombre_clase
  - ✓ **new**. Es un operador para usos varios, fundamentalmente creación de objetos a partir de una clase:
    tipo variable = new clase(parametros)
    También se utiliza para crear matrices de datos.

Los operadores ++ y – – pueden aparecer delante o después de la variable sobre la que se aplican. **Si se colocan delante, la variable se incrementará o decrementará antes de ser utilizada en la expresión**.

Las variables se declaran en las clases:

```
class  prueba
{
        private int ValorX;  Las variables de la clase son los datos miembro  VALORX   SE INICIALIZAN POR DEFECTO CON EL VALOR CERO.
        public metodo( )
        {
                int  ValorY = 128;   Las  variables de los métodos son variablaes locales al método.   TENEMOS QUE INICIALIZARLAS SIEMPRE  SINO DA ERROR
                ValorX = ValorX + ValorY   Si no hubieramos inocializado ValorY daría un error  al  utilizar esta variable
        }
}
```

- **PRÁCTICAS:** Hacer algún/algunos programa sencillo/s como:
  -Calcular la suma de dos números cualquiera y sacar el resultado por pantalla.

3.- Sentencias del Control de Flujo

3. SENTENCIAS DE CONTROL DE FLUJO de JAVA

(cap.6)

- Sentencia: **if**
- Sentencia: **if ... else**
- Anidamiento de estructuras: **else if**
- Sentencia: **switch ...case** y **swith case ... default**
- Sentencia: **break**
- Sentencia: **while**
- Sentencia: **do ... while**
- Sentencia: **for**
- Bucles anidados y sentencias break y continue
- Etiquetas

Sentencias de Control.

Sirven para modificar el flujo de ejecución de un programa. Se dividen en dos grupos

- **Sentencias de selección.**
- **Sentencias de iteración** (bucles).

Sentencias de selección.

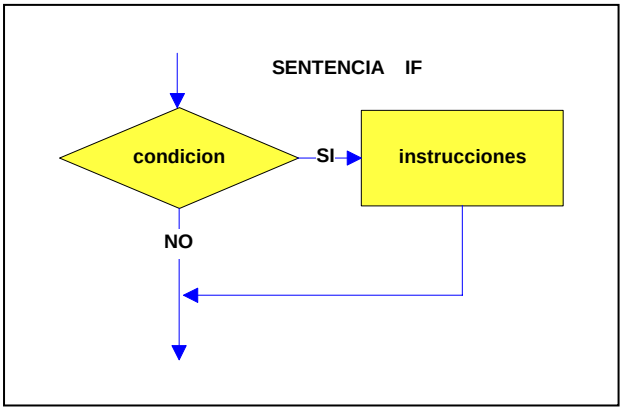
Controlan el flujo de ejecución del programa en función de una determinada condición o expresión. Java proporciona dos sentencias de este tipo: **if** y **switch**.

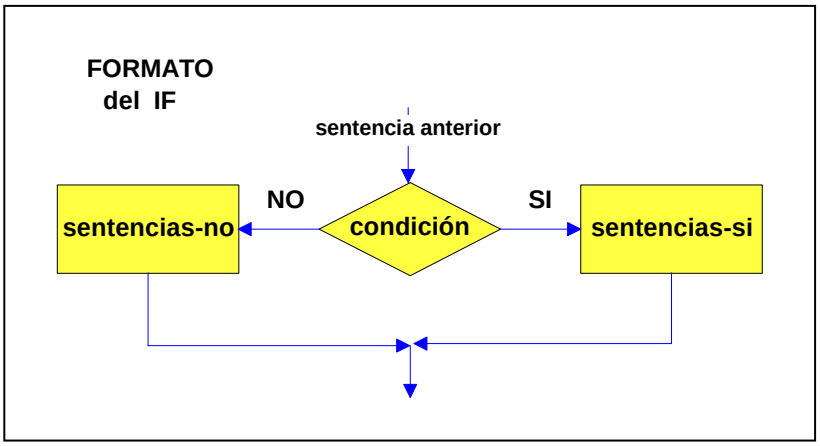
Formato de la sentencia if:

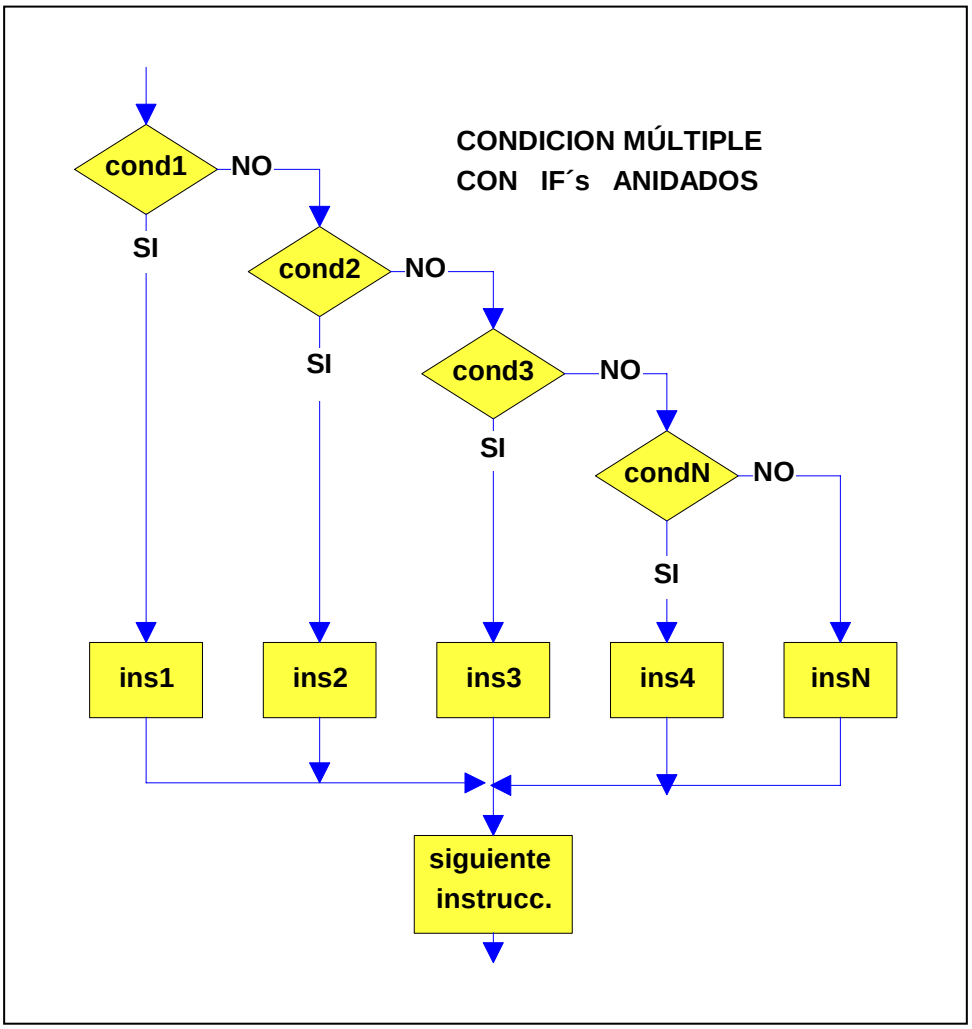
```
if (condicion)
{
        sentencia1
        sentencia2
              :
}
else
{
        sentencias
              :
}
```

Los if se pueden anidar, cada if puede o no llevar su else.
Si el bloque de sentencias solo contiene una instrucción no es necesario poner llaves

Representación Grafica de la sentencia IF:







EJEMPLO1:

```
.....
int    a;
int    b;
if (a > b)     a=0;
else           b=0;
```

EJEMPLO2:

```
.....
if (ia == 10) {
        if (j <  20)    a=b;
        if (k >  100)   c=d;     //este if
        else    a=c;             //está asociado con este else
} else a=d;     //este else está asociado con el primero
.....
```

* Para conseguir más claridad es conveniente tabular y poner llaves { } de comienzo y fin de bloque y poner una sola instrucción por línea .

Formato: **Sentencia if ... else if ... else if ...**

```
if (boolean condicion)
{
        //Sentencias
}
else if (boolean condicion)
{
        //Sentencias
}
else
{
        //sentencias
}
```

EJEMPLO PRÁCTICO: Hacer un programa con IF´s que a partir del número de més, imprima el literal de la estación del año en la que estamos.

Invierno = meses 12, 1, 2
Primavera = meses 3, 4, 5
Verano = meses 6, 7, 8
Otoño = meses 9, 10, 11

```
public class IfElseIf
{
        public  static  void  main(String [] args)
        {
                int Mes=4;   //  abril
                String Estacion;

                if      (Mes == 12 || Mes == 1 || Mes == 2 )
                          Estacion = "Invierno";
                else if (Mes == 3 || Mes == 4 || Mes == 5 )
                          Estacion = "Primavera";
                else if (Mes == 6 || Mes == 7 || Mes == 8 )
                          Estacion = "Verano";
                else if (Mes == 9 || Mes == 10 || Mes == 11 )
                          Estacion = "Otoño";
                else
                          Estacion = "Estación desconocida";
                System.out.print("EL NUMERO DE MES NOS INDICA QUE ESTAMOS EN" + "\n");
                System.out.println("LA ESTACION= " + Estacion );
        }
}
```

Sentencia Switch

En función del resultado de una expresión se establecen distintas salidas para el programa. El formato es:

```
switch (expresion)
{
        case valor1:
                sentencias
        break;
        case valor2:
                sentencias
        break;
        default:
                sentencias
}
```

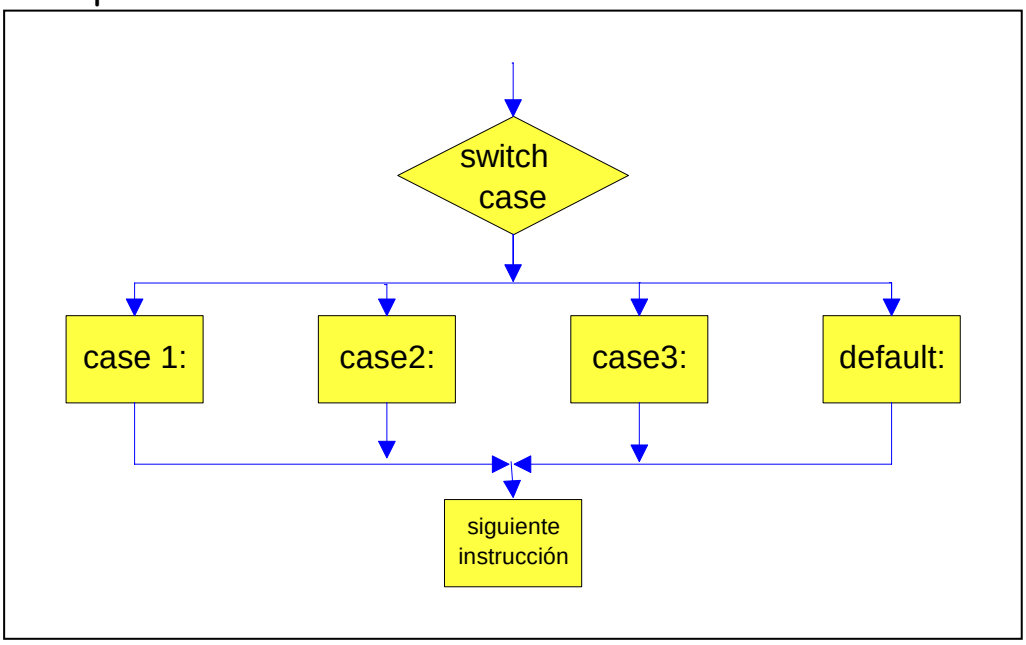
donde **valor** representa una **constante o literal numérico.**

PRÁCTICA: Hacer un programa que analice un número para ver si es par, impar ...

```
public class VerPar
{
 public static void main(String[] args)
 {
 int n=3435;
 System.out.println("El N\u00A3mero a analizar es=  " + n );
 if(n%2==0)
  System.out.println("el número "+n+" es par");
 else
  System.out.println("el número "+n+" no es par");

 switch(n%10)
  {
  case 2:
   System.out.println("acaba en 2");
   break;
  case 3:
   System.out.println("acaba en 3");
   break;
  case 4:
   System.out.println("acaba en 4");
   break;
  default:
   System.out.println("no acaba en 2,3,4");
  }
 }
}
```

Representación Grafica del **switch case**:



PRÁCTICA: Versión mejorada del programa de las estaciones del año.

Invierno = meses 12, 1, 2 Primavera = meses 3, 4, 5

Verano = meses 6, 7, 8 Otoño = meses 9, 10, 11

```
public class ClaSwitch {
  public static void main(String [] args)       {
        int Mes=4;  // abril
        String Estacion;

        switch ( Mes ) {
                case 12:  case 1:  case 2:
                        Estacion = "Invierno";
                        break;
                case 3:  case 4:  case 5:
                        Estacion = "Primavera";
                        break;
                case 6:  case 7:  case 8:
                        Estacion = "Verano";
                        break;
                case 9:  case 10:  case 11:
                        Estacion = "Otoño";
                        break;
                default:
                        Estacion = "Estación desconocida";
        }
        System.out.print("EL NUMERO DE MES NOS INDICA QUE ESTAMOS EN" + "\n");
        System.out.println("LA ESTACION= " + Estacion );
      }
  }
```

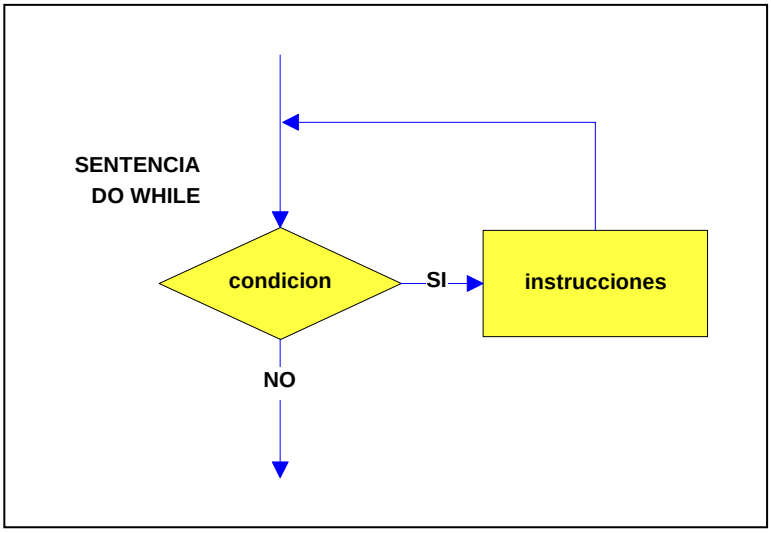
Sentencias de iteración.

Ejecutan un grupo de sentencias un número determinado de veces.

Sentencia While

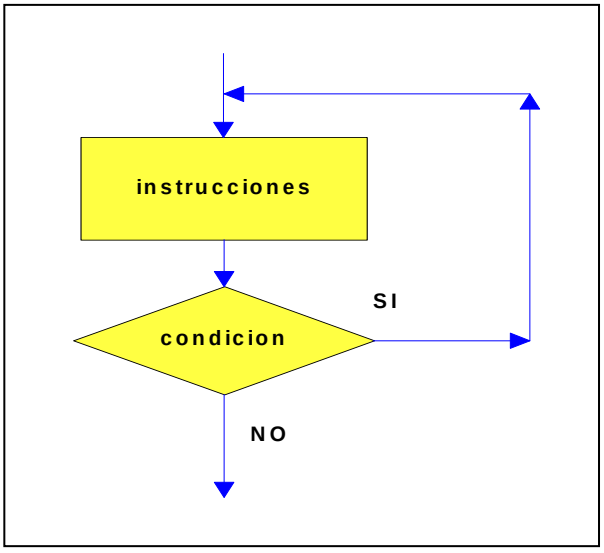
Ejecuta un grupo de sentencias mientras se cumpla una condición:

```
while (condición)
{
        sentencias
}
```



Existe una variante de while que es do – while:

```
do
{
        sentencias
}while (condición)
```



EJEMPLOS :

//El bucle WHILE

```
public  class  While  {
        public static void main(String [] args)          {
                int numero= 10 ;
                while (n > 0) {
                        System.out.println("TIC= " + n );
                        n--
                }
        }
}
```

```
       // El cuerpo de un bucle puede estar vacio.  En este ejemplo primero se analiza si la
       // condición se cumple, si la respuesta es "true" incrementa "i" y decrementa "j"  y se
       // repite el proceso mientras se cumple la condición del while
public  class  CuerpoWhileVacio  {
        public static void main(String [] args)          {
                int i= 100;
                int j=200;
                while (++i  <  --j) ;  //encuentra el centro entre i y j
                System.out.println("EL VALOR MEDIO ES: "  +  i );
        }
}
```

```
public  class  DoWhile  {
        public static void main(String [] args)          {
                int numero= 10 ;
                do  {
                        System.out.println("TIC= " + n );
                        n--
                } while (n > 0) ;
        }
}
```

```
//UTILIZACIÓN de la sentencia "break" para salir del un bucle
public  class  WhileConBreak  {
        public static void main(String [] args)          {
                int i=1
                while (n < 100) {
                        System.out.println("Vamos por el numero=> " + n );
                        n++
                        if ( i ==60 )  break;      // y el bucle se termina
                }
        }
}
```

Sentencia For.

El número de veces que se ejecutan las sentencias está definido en la propia instrucción:

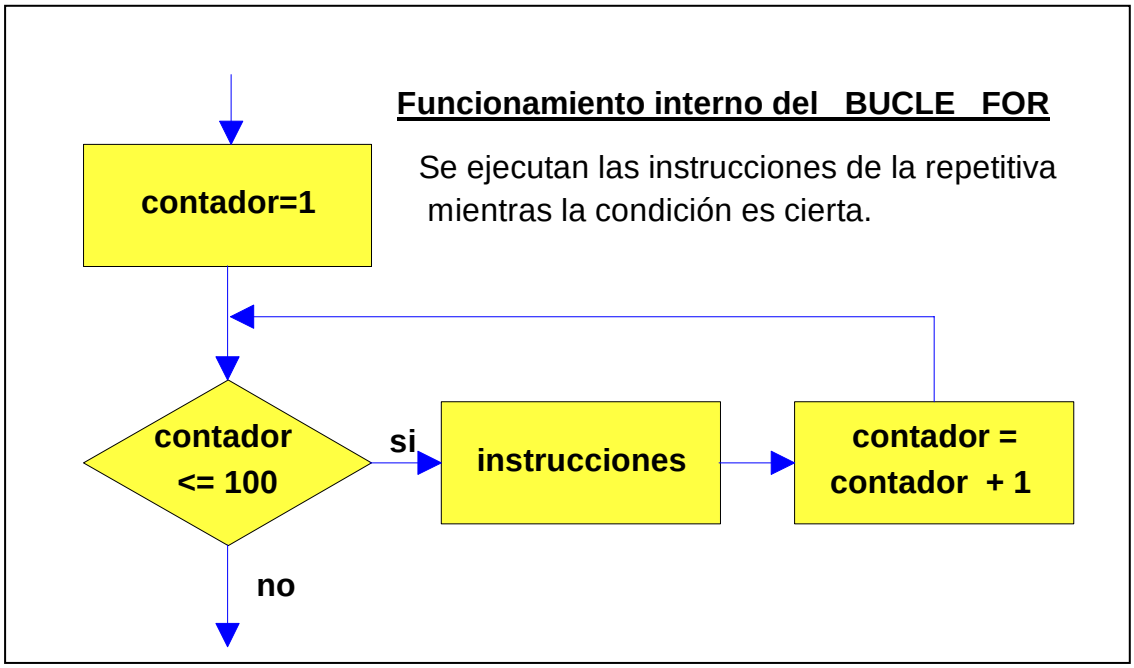
```
for (inicialización ; condición ; incremento)
{
        sentencias
        [continue;]
        [break;]
}
```

En primer lugar se ejecuta la inicialización, después se comprueba la condición y si es verdadera, se ejecutan las sentencias. Cuando se llega al final del bucle se ejecuta incremento y se vuelve a comprobar la condición.
La instrucción **continue** provoca un salto a la **siguiente iteración** y **break** una **salida forzada** del mismo:

```
for ( int i=1 ; i<=10 ; i++ )
{
        System.out.println ("tic" + i);
}
```

La variable de control puede ser inicializada dentro o fuera del for.
La inicialización, condición y acción son **opcionales** y no tienen por que limitarse a una única instrucción:

```
for (i=0, c=1; i<3;)
```



- **PRÁCTICAS:** Hacer los ejercicios siguientes relacionados con estructuras de control:

1- Hacer un programa que imprima todos los números del 1 al 20 inclusive, excepto los multiplos de de 5 , tampoco queremos que se imprima el número 13 ya que "dicen" que trae mala suerte.

2- Imprimir con un bucle al valor todos los caracteres ASCII y su correspondiente valor decimal.

3- Imprimir la suma de los diez primeros números naturales.

4- Imprimir los pares comprendidos entre el 1 y el 40, Se imprimirán en parejas simultaneas ascendente y descendente.
EJ: 2 y 40, 4 y 38, 6 y 36 etc.

5- Dados dos números el programa imprimirá los números comprendidos entre los dos además al final imprimirá la suma total de todos ellos.

6- Un programa que imprima la suma de los pares comprendidos entre el 1 y el 100.

7- Hallar e imprimir el factorial de un número dado (entero positivo) y si el número introducido es negativo no hará el cálculo e informará de que es negativo.

8- Comprobar si un año es bisiesto o no. (Un año es bisiesto si es múltiplo de 400. También es bisiesto si a la vez, es divisible por 4 y No es divisible por 100).

9- Comprobar si un año es primo o no. (Hay varias formas. Una de ellas es: un número será primo si no ha sido divisible por ninguno de los números menores menores que él empezando desde el dos y terminando cuando igual que número/2).

10- Imprimir el factorial de un número dado. Además, se irán imprimiendo los factoriales de todos los números menores.