Lenguaje de Programación Java

Tecnicatura Universitaria en Web

Características de Java

- ***Simple:*** sin unions, structs, typedefs, preprocesador, aritmética de punteros, sobrecarga de operadores, herencia múltiple, goto, funciones.
- ***Orientado a Objetos***.
- ***Tipado estáticamente:*** chequeo de tipos en tiempo de compilación.
- ***Compilado e interpretado:*** primero compilado a “byte-code”, después interpretado por intérprete Java.
- ***Independiente de la Arquitectura y Portable***.
- ***Recolector de Basura:*** libera al programador de desasignar memoria.

Características de Java

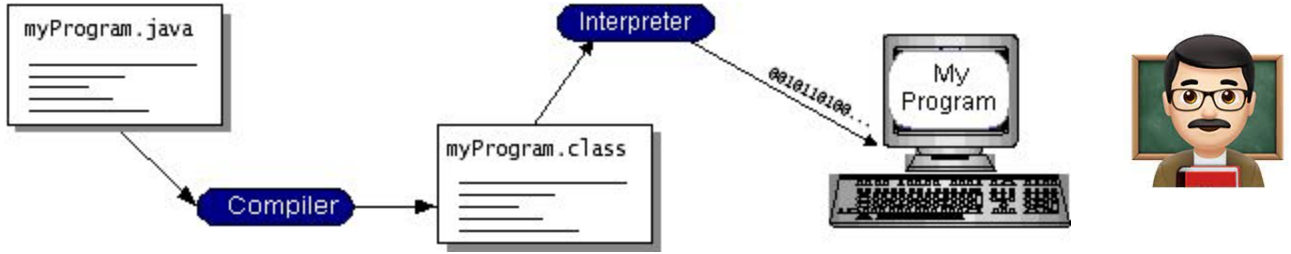
- ***Robusto:*** intérprete controla todos los accesos al sistema, no hay crash del sistema.
- ***Seguro:*** verifica todos los accesos a memoria principal. No hay acceso a áreas del sistema no autorizadas.
- ***Multi-Hilo:*** los programas pueden contener múltiples hilos de ejecución, lo que permite la concurrencia de tareas. Ejemplo: programa muestra una animación de una imagen mientras continúa aceptando entradas desde el teclado.
- ***Extensible:*** soporta métodos no nativos (métodos implementados en otro lenguaje). Son dinámicamente linkeados a la aplicación Java.

Características de Java

- Java puede ser usado en tres formas:
  - Aplicaciones stand-alone, es decir aplicaciones sin complementos.
  - Applets (aplicaciones que ejecutan en un browser). Un Applet es una aplicación que se encuentra embebida en código HTML y se envía junto con una página Web a un usuario.
  - Servlets (aplicaciones que ejecutan en un servidor).

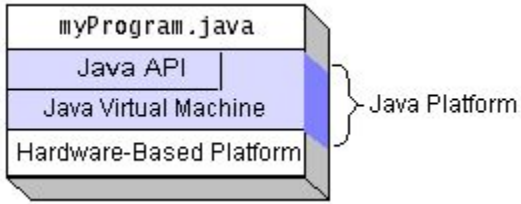
Plataforma de Java

- Una plataforma es el ambiente de hardware o software en el cual un programa ejecuta.
- Con el compilador, primero se traduce un programa en lenguaje Java a un programa en código intermedio llamado Java bytecodes (un código independiente de la plataforma interpretado por el intérprete Java sobre la plataforma Java).
- El intérprete analiza y ejecuta cada instrucción en bytecode.
- La compilación ocurre sólo una vez (sin errores); la interpretación ocurre cada vez que el programa es ejecutado.



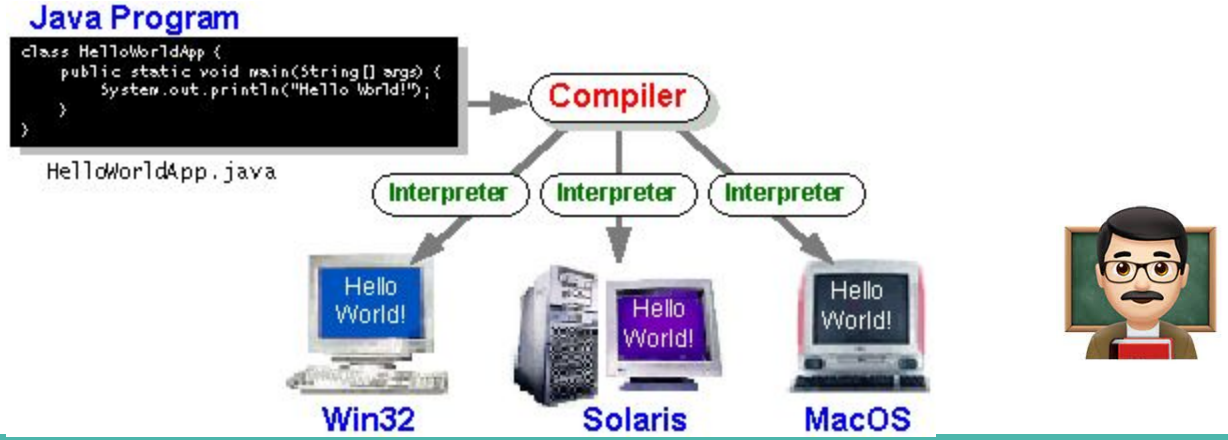
Plataforma de Java

- La plataforma Java tienen dos componentes:
  - La Java Virtual Machine (JVM)
  - La Java Application Programming Interface (Java API)
- La Java API es una gran colección de componentes de software listas para ser usadas que proveen muchas utilidades. Se encuentran agrupadas en paquetes.

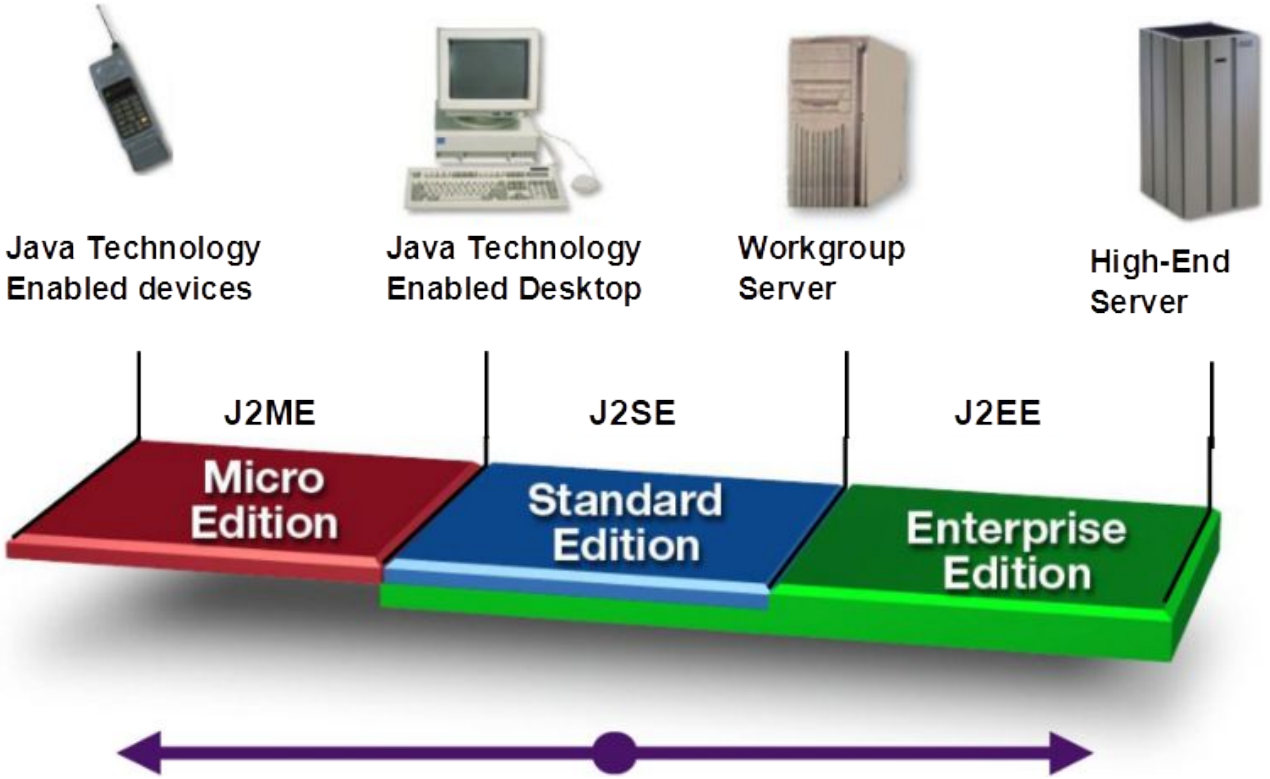


JVM (Java Virtual Machine)

- Se puede pensar en los bytecodes de Java como las instrucciones de código de máquina para una máquina particular: la Máquina Virtual Java (JVM).
- Cada intérprete Java es una implementación de la JVM.
- El bytecode de Java ayuda a hacer posible la idea de “escribir una vez, ejecutar donde quiera”.

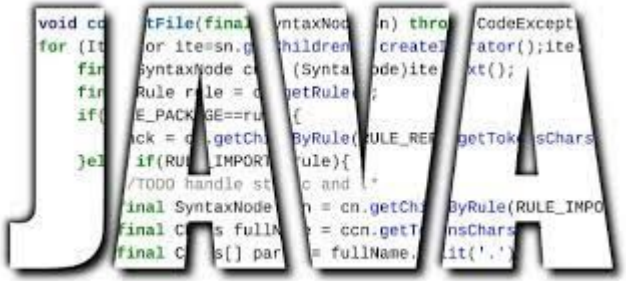


Ediciones de la Plataforma de Java

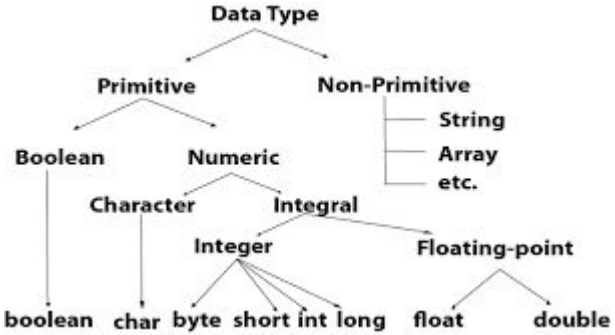


Entornos de Integrados de Desarrollo

- Permiten al usuario escribir, compilar, probar, depurar y editar programas escritos en lenguaje Java.
  - Intellij Idea
  - Sun Java Studio
  - Borland JBuilder
  - eclipse
  - JDEVELOPER
  - NetBeans
  - ...



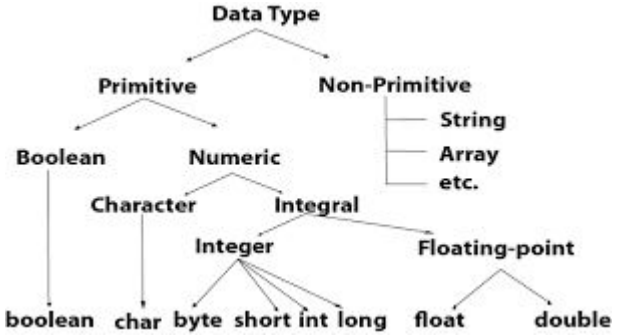
Tipos Numéricos - Enteros



Tipo	*Tamaño*	*Descripción*
byte	8 bits en complemento a 2	Entero de un byte
short	16 bits en complemento a 2	Entero short
int	32 bits en complemento a 2	Entero
long	64 bits en complemento a 2	Entero long

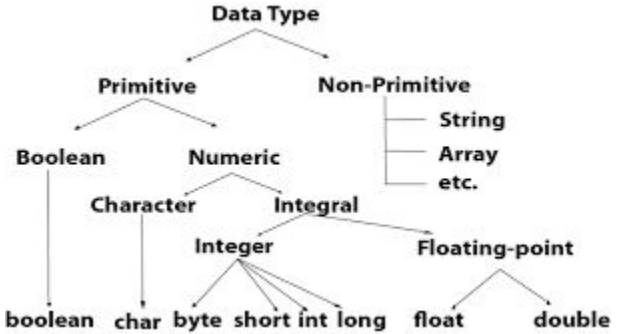
Tipos Numéricos - Flotantes

Tipo	*Tamaño*	*Descripción*
float	32 bits	flotante simple precisión
double	64 bits	flotante doble precisión

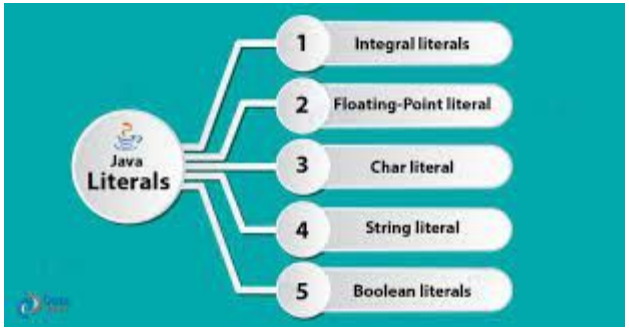


Tipos Numéricos - Otros Tipos

Tipo	*Tamaño*	*Descripción*
char	16 bits	Un caracter
boolean	True - False	Un valor de verdad



Literales



Literal	Data Type
178	int
8864L	long
37.266	double
37.266D	double
87.363F	float
26.77e3	double
'c'	char
true	boolean
false	boolean

Tipos de datos No Primitivos

- Es decir, el valor de una variable de referencia es una referencia (un puntero) hacia el valor real.
  - Strings
  - Arrays
  - Clases
  - Interfaces

Caracteres Especiales

Carácter	*Símbolo*
Backslash	\\
Backspace	\b
Carriage Return	r
Double Quote	\”
Form Feed	\f
Line Feed	\n
Single Quote	\’
Tab	\t

Sintaxis

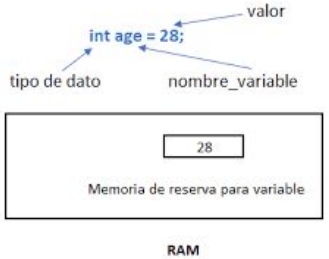
- ***Case sensitive***
  - *String* no es lo mismo que *string*
  - *MAIN* no es lo mismo que *main*
- ***Palabras reservadas en minúsculas***
  - Ej. *public*, *class*, *static*, *void*

Convenciones de Nombre

Aparte de unas pocas palabras reservadas, como por ejemplo *if*, *while*, etc., Java no impone restricciones sobre los nombres que se pueden usar para los identificadores. Sin embargo, existe una convención estándar para los nombres:

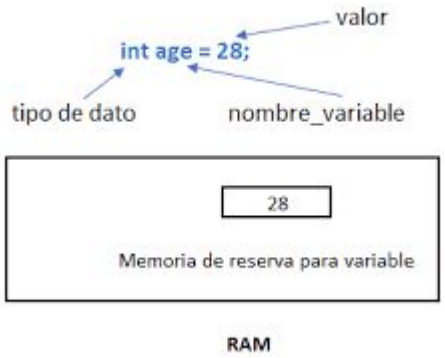
- Los métodos, variables y objetos comienzan con una letra minúscula: next, push, index, etc.
- Las clases comienzan con una letra mayúscula: String, StringBuffer, Vector, Calculator, etc.
- Las constantes son todas en mayúsculas: final double PI = 3.1415926;
- La separación de palabras en los identificadores es hecha empezando con mayúscula las palabras después de la primera, excepto para las constantes donde se usa underscore como separador de palabras. toString, addMouseListener, UNA_CONSTANTE

Nombres de variables



- Una variable es un espacio en memoria donde se almacena un dato.
- Un programa se refiere al valor de una variable por su nombre.
  - Por convención, en Java, los nombres de las variables empiezan con una letra minúscula (los nombres de las clases empiezan con una letra mayúscula).
  - Un nombre de variable Java debe ser un identificador legal de Java comprendido en una serie de caracteres Unicode.
  - No puede ser el mismo que una palabra clave o el nombre de un valor booleano (true or false)
  - No deben tener el mismo nombre que otras variables cuyas declaraciones aparezcan en el mismo ámbito.
  - Por convención, los nombres de variables empiezan por un letra minúscula. Si una variable está compuesta de más de una palabra, como *nombreDato* las palabras se ponen juntas y cada palabra después de la primera empieza con una letra mayúscula.

Declaración de variables



- Tipo nombreVariable;

int cantidad;

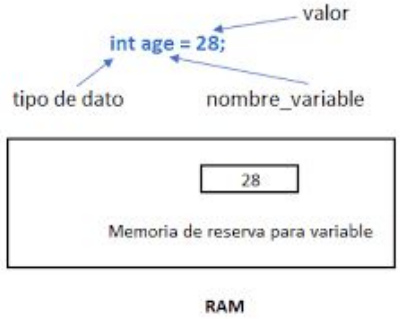
int edad=20; /*crea e inicializa la variable edad con 20*/

- ¿Desde donde se puede acceder a la variable?
  - Depende de donde se realizó la declaración de la misma.
  - Ámbito de una variable: parte del programa desde el cual es accesible la variable.

Ámbito de las variables

- ***Variables locales:*** se definen dentro de un método o dentro de cualquier bloque entre { } , se crean en el interior del bloque y se destruyen al finalizar dicho bloque.

- ***Variables miembro de una clase:*** se definen en una clase, fuera de cualquier método.



Declaración de constantes

- ¿Qué es una constante?
  - final int TAM=20;
- El valor de tamaño se mantiene y puede ser utilizado pero no modificado durante la ejecución del programa.

Operadores aritméticos

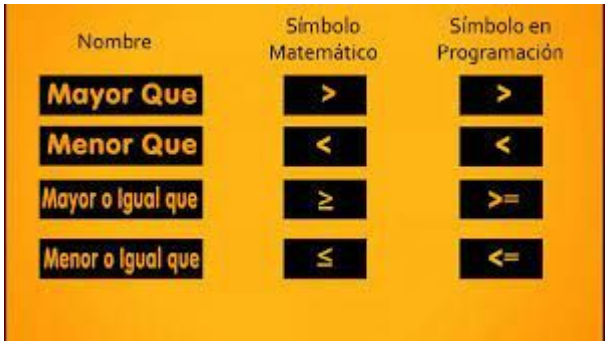
Operador	Significado	Ejemplo	Resultado
-	Resta	a - b	Resta de a y b
+	Suma	a + b	Suma de a y b
*	Multiplicación	a * b	Producto de a por b
/	División	a / b	Cociente de a entre b
%	Residuo	a % b	Residuo de a entre b

- + Suma de dos valores
- - Resta de dos valores
- - cambio de signo de un número (un solo operando: – 23)
- * Multiplicación de dos valores
- / División de dos valores
- % El resto de la división de dos números
- ++ Incremento en una unidad (un solo operando)
  - i++ pos incremento
  - ++i pre incremento
- -- Decremento en una unidad (un solo operando)
  - i-- pos decremento
  - --i pre decremento.

Operadores Condicionales

Operador	Descripción	Ejemplo de expresión	Resultado del ejemplo
==	igual que	`7 == 38`	`false`
!=	distinto que	`'a' != 'k'`	`true`
<	menor que	`'G' < 'B'`	`false`
>	mayor que	`'b' > 'a'`	`true`
<=	menor o igual que	`7.5 <= 7.38`	`false`
>=	mayor o igual que	`38 >= 7`	`true`

- == Comprueba si los dos operandos son iguales.
- != Comprueba si los dos operandos son distintos.
- > Mayor que, devuelve true si el primer operando es mayor que el segundo.
- < Menor que, es true si el primer operando es menor que el segundo.
- >= Mayor igual.
- <= Menor igual.



Operadores lógicos

- ! Operador NO o negación.
  - Boolean estado=true;
  - ! estado --> estado es false
- && Operador Y
  - estado && 2<3 ?
- || Operador O
  - estado || 2<3 ?.

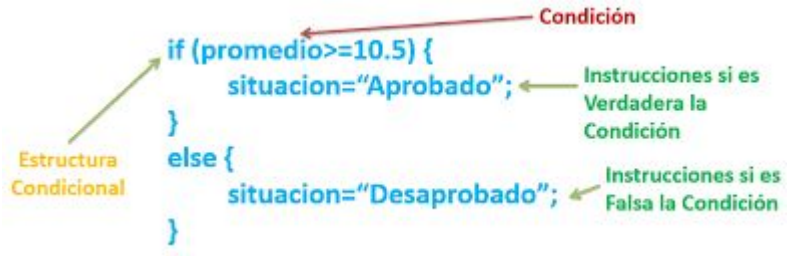
Operador	Descripción	Ejemplo de expresión	Resultado del ejemplo
!	Negación - NOT (unario)	`!false` `!(5==5)`	`true` `false`
|	Suma lógica – OR (binario)	`true | false` `(5==5)|(5<4)`	`true` `true`
^	Suma lógica exclusiva – XOR (binario)	`true ^ false` `(5==5)|(5<4)`	`true` `true`
&	Producto lógico – AND (binario)	`true & false` `(5==5)&(5<4)`	`false` `false`
||	Suma lógica con cortocircuito: si el primer operando es `true` entonces el segundo se salta y el resultado es `true`	`true || false` `(5==5)||(5<4)`	`true` `true`
&&	Producto lógico con cortocircuito: si el primer operando es `false` entonces el segundo se salta y el resultado es `false`	`false && true` `(5==5)&&(5<4)`	`false` `false`

Operadores de asignación

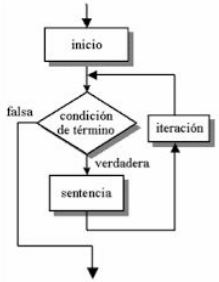
- = Asignación. Asigna la parte de la derecha del igual a la parte de la izquierda.
- += Asignación con suma. Realiza la suma de la parte de la derecha con la de la izquierda y guarda el resultado en la parte de la izquierda.
- -= Asignación con resta.
- *= Asignación de la multiplicación.
- /= Asignación de la división.
- %= Se obtiene el resto y se asigna.

Estructuras de control

- ***Toma de decisiones:*** Para realizar unas acciones u otras en función de una condición.
  - ***if***
  - ***switch***



- ***Bucles o iteraciones:*** Para realizar ciertas acciones repetidamente.
  - ***for***
  - ***while***
  - ***do while***



Estructuras de control: if - Sintaxis

if (expresión) { */* acciones a realizar en caso de que el resultado de evaluar la expresión sea verdadera */*

…

}

if (expresión) { */* acciones a realizar en caso de que el resultado de evaluar la expresión sea verdadera */*

…

} ***else*** { */*acciones a realizar en el caso que no se cumpla la expresión*/*

…

}

Estructuras de control: switch - Sintaxis

switch (expresión) {

case valor1: */* Sentencias a ejecutar si la expresión tiene como valor a valor 1*/*

break;

case valor2: */*Sentencias a ejecutar si la expresión tiene como valor a valor 2 */*

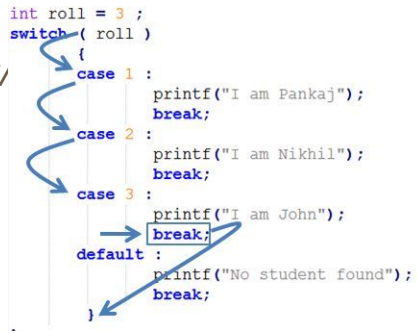
break;

case valor3: */*Sentencias a ejecutar si la expresión tiene como valor a valor 3 */*

break;

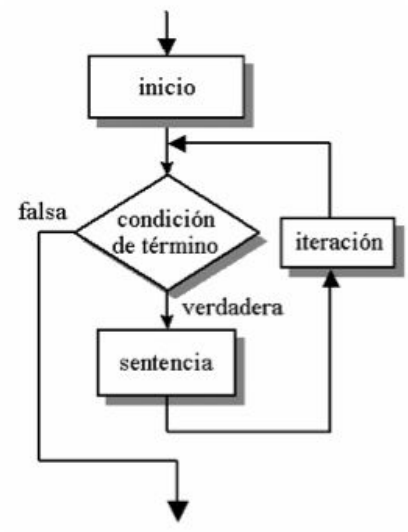
default: */*Sentencias a ejecutar si el valor no es ninguno de los anteriores*/*

}



Estructuras de control: bucles

Se utilizan para repetir una o más instrucciones un determinado número de veces.



Estructuras de control - Bucle - for

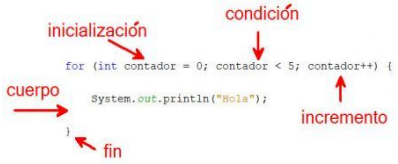
Se suele utilizar cuando sabemos seguro el número de veces que queremos que se ejecute las sentencias.

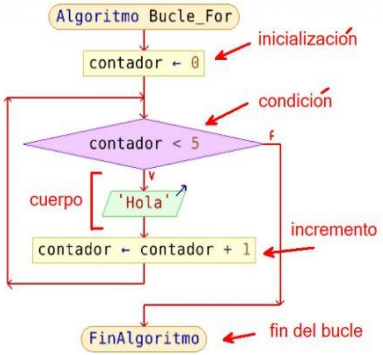
Sintaxis:

for (inicialización;condición;actualización) {

  sentencias a ejecutar en cada iteración

}





Estructuras de control - Bucle - while

Se utilizan cuando se quiere repetir la ejecución de una o más sentencias un número indefinido de veces (0 o más veces), según se cumpla una condición.

Sintaxis

while (condición) {

    sentencias a ejecutar en cada iteración

}

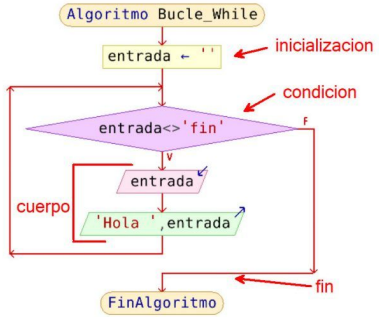
```
Scanner sc = new Scanner(System.in);
String entrada = "";
while (!entrada.equals("fin")) {
    System.out.println("Introduzca su nombre (fin para terminar): ");
    entrada = sc.nextLine();
    System.out.println("Hola " + entrada);
}
```

inicialización

condición

cuerpo

fin del bucle



Estructuras de control - Bucle - do-while

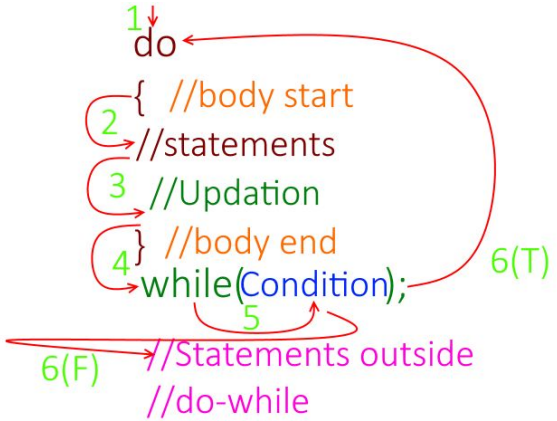
Se utilizan cuando se quiere repetir la ejecución de una o más sentencias al menos una vez.

Sintaxis

do {

    sentencias a ejecutar en cada iteración

} ***while*** (expresión)



Sentencias de ruptura

- La sentencia ***break*** hace que el control del flujo bifurque a la sentencia siguiente a la actual.
- La sentencia ***continue*** dentro de un bucle bifurque de la sentencia actual hacia el principio del bucle.
- La sentencia ***return*** se utiliza para salir del método actual y volver a la sentencia siguiente a la que originó la llamada en el método original.
- Existen dos formas de ***return***: una que devuelve un valor y otra que no lo hace.
- Para devolver un valor, simplemente se pone el valor (o una expresión que calcule el valor) seguido a la palabra ***return***.
    - ***return expresión***

Sentencias de ruptura

```
while (testExpression) {
    // codes
    if (condition to break) {
        break;
    }
    // codes
}
```

```
do {
    // codes
    if (condition to break) {
        break;
    }
    // codes
}
while (testExpression);
```

```
for (init; testExpression; update) {
    // codes
    if (condition to break) {
        break;
    }
    // codes
}
```

```
while (testExpression) {
    // codes
    if (testExpression) {
        continue;
    }
    // codes
}
```

```
do {
    // codes
    if (testExpression) {
        continue;
    }
    // codes
}
while (testExpression);
```

```
for (init; testExpression; update) {
    // codes
    if (testExpression) {
        continue;
    }
    // codes
}
```

Arreglos: arrays

- Estructura de datos que permite almacenar *n* elementos del mismo tipo.
- Declaración:
  - ***int*** [] edades; *o* ***int*** edades []; //No se le ha asignado espacio al arreglo
- Si intento asignar un valor a alguna posición del arreglo me dará error de compilación.

int[] edades = ***new int***[10]

//Es un arreglo de 10 enteros que se acceden desde la posición 0 a 9

- Acceso: edades[0]=2;
  - ……
  - edades[i]=22;

Cadenas: String

- ***String:*** secuencia de caracteres entre comillas
  - "Programación II"
  - "a"
  - "123"
  - ***String*** nombre= "Juanito";
  - ***String*** nombre= ***new String*** (10);
- Concatenar cadenas:
  - + concatena cadenas
    - "Hola" + "como estas" → "Holacomo estas"
    - "Hola " + nombre; → "Hola Juanito"

Estructura de un Programa JAVA

Para definir un Aplicación en Java, se debe definir, al menos, una clase que contenga un método especial que será el ***Programa Principal de la Aplicación***.

public class ClasePrincipal {

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

sentencia - 1;

sentencia - 2;

...

sentencia - N;

}

}

```java
public class HelloWorld {

    public static void main(String[] args)
    {
    System.out.println("Hello world");
    }
}
```

Una aplicación simple: “Hello World”

Hello.java

```
public class Hello{
   public static void main(String[] args){
      System.out.println(" Hola Programación II !!! ");
   }
}
```

C:\javac Hello.java *(compilación crea Hello.class)*
C:\java Hello *(Ejecución sobre la JVM local)*

- El archivo debe llamarse Hello.java (por el nombre de la clase pública).
- El comando javac invoca al compilador Java sobre el archivo fuente Hello.java
- Si todo va bien, creará un archivo llamado Hello.class, el cual contiene el código para la JVM.
- El comando java invoca al intérprete Java para ejecutar el código para la clase Hello.
- En una aplicación Java, la primera cosa que se ejecuta es el método llamado main de la clase indicada (en este caso Hello). Se pueden tener métodos main en todas las clases que se desee.