



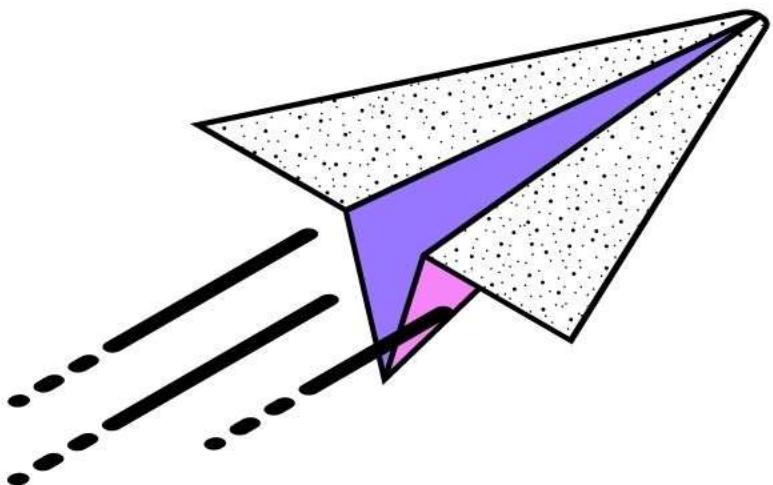
**Java Standard Edition**





# Sección 3

Trabajar con variables primitivas



Declarar y asignar valores a variables.

Uso de constantes.

Descripción de tipos de datos primitivos como int, float, double, boolean.

Utilizar operadores aritméticos para modificar valores.

Declaración e inicialización de variables de campo.



# Conceptos importantes

Las siguientes cincuenta palabras clave están reservadas para uso del lenguaje Java:

<code>abstract</code>	<code>double</code>	<code>int</code>	<code>super</code>
<code>assert</code>	<code>else</code>	<code>interface</code>	<code>switch</code>
<code>boolean</code>	<code>enum</code>	<code>long</code>	<code>synchronized</code>
<code>break</code>	<code>extends</code>	<code>native</code>	<code>this</code>
<code>byte</code>	<code>final</code>	<code>new</code>	<code>throw</code>
<code>case</code>	<code>finally</code>	<code>package</code>	<code>throws</code>
<code>catch</code>	<code>float</code>	<code>private</code>	<code>transient</code>
<code>char</code>	<code>for</code>	<code>protected</code>	<code>try</code>
<code>class</code>	<code>goto</code>	<code>public</code>	<code>void</code>
<code>const</code>	<code>if</code>	<code>return</code>	<code>volatile</code>
<code>continue</code>	<code>implements</code>	<code>short</code>	<code>while</code>
<code>default</code>	<code>import</code>	<code>static</code>	
<code>do</code>	<code>instanceof</code>	<code>strictfp*</code>	



# Conceptos importantes

Las siguientes cincuenta palabras clave están reservadas para uso del lenguaje Java:

---

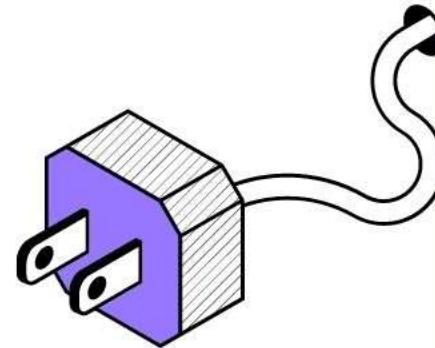
Carácter	Nombre	Descripción
{ }	Llaves de apertura y cierre	Denota un bloque para encerrar sentencias.
( )	Paréntesis de apertura y cierre	Se utiliza con métodos.
[ ]	Corchetes de apertura y cierre	Denota una matriz.
//	Doble slash	Precede una línea de comentario.
""	Comillas de apertura y cierre	Encierre una cadena (es decir, una secuencia de caracteres).
;	Punto y coma	Marcar el final de una declaración.



# ¿Qué es una variable?

## Definición de variable

- Las variables se utilizan para representar valores que pueden cambiar durante la ejecución de un programa.
- Las variables tienen un nombre, que es un identificador para hacer referencia a ellas en nuestros programas.



$x$   
 $\neq$   
 $y$

$f(x)$



## Reglas para definir el nombre de una variable en Java

- Un identificador es una secuencia de caracteres que consta de letras, dígitos, guiones bajos (\_) y signos de dólar (\$).
- Un identificador debe comenzar con una letra, un guion bajo (\_) o un signo de dólar (\$). No puede comenzar con un dígito.
- Un identificador no puede ser una palabra reservada.
- La sintaxis para declarar una variable es **tipo\_de\_dato nombre\_variable;**

## Notas y tips para definir nombres de variables.

- Dado que Java distingue entre mayúsculas y minúsculas, **area**, **Area** y **AREA** son identificadores diferentes.
- Los identificadores descriptivos hacen que los programas sean fáciles de leer.



# Declaración variables

Ejemplo donde se declaran tres variables.

```
● ● ●

public class Prueba{

    public static void main(String[] args){

        //Ejemplo DECLARACION de variables
        int contador;
        double radio;
        double tasaInteres;

    }
}
```



## Asignar valor a una variable

- Una declaración de asignación designa un valor para una variable. Una declaración de asignación se puede utilizar como una expresión en Java.
- Después de declarar una variable, puede asignarle un valor. En Java, el signo igual (=) se utiliza como operador de asignación.
- Un identificador no puede ser una palabra reservada.
- La sintaxis para asignar valor a una variable es *nombre\_variable = expresión o valor;*

## Notas y tips para expresiones de asignación.

- Una expresión representa un cálculo que involucra valores, variables y operadores que, tomándolos juntos, se evalúa como un valor.



# Ejercicio

**a = 10**

**b = 20**

**c = 5**

**a = a + 3**

**b = b + 4 - a**

**c = a + b + c**

**a = a + c**

**b = 4**

**c = c + 3 - b + 2**

**Qué valores quedan almacenados en las variables a, b y c ?**



# Conclusión

Las **variables** son espacios en la memoria temporal, dónde podemos guardar datos y accederlos a través de un nombre. También las podríamos pensar que son como cajas nombradas de la memoria.





# Asignar valor a una variable

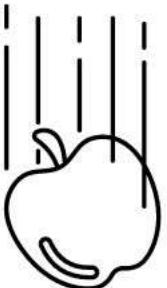
Ejemplo donde se declaran tres variables y se les asigna un valor.

```
public class Prueba{  
  
    public static void main(String[] args){  
  
        //Ejemplo ASIGNACION a variables  
        int contador = 1;  
        double radio = 1.0;  
        int x = 5 * (3 / 2);  
        double area;  
        contador = x + 1;  
        area = radius * radius * 3.14159;  
  
    }  
}
```



$\pi$

$e$



## Constantes

- Una constante es un identificador que representa un valor permanente.
- El valor de una variable puede cambiar durante la ejecución de un programa, pero una constante, representa datos permanentes que nunca cambian.
- La sintaxis para asignar valor a una variable es  
**final datatype NOMBRE\_CONSTANTE = valor;**

## Notas y tips para expresiones de asignación.

- Una constante debe declararse e inicializarse en la misma instrucción.



# Declaración Constante

Ejemplo donde se declaran cuatro constantes y sus valores.

```
public class Prueba{  
    public static void main(String[] args){  
        //Ejemplo declaracion de constantes  
        final double PI = 3.14159;  
        final double E = 2.7182;  
        final int VALOR_MINIMO = 1;  
        final int VALOR_MAXIMO = 20;  
    }  
}
```



# Tipo de datos primitivos

Diferencia entre tipos primitivos y objetos.



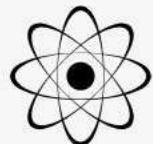
## No poseen atributos

Los tipos de datos no poseen atributo alguno, estos solo guardan un valor en alguna dirección en memoria.



## No poseen métodos

Los tipos de datos no poseen métodos, de hecho estos solo son utilizados por los métodos en partes del programa.



## Son indivisibles, atómicos

Con baja latencia hay menos retrasos; da una reacción genuina en tiempo real.



# Tipo de datos numéricos

Entre los tipos numéricos tenemos: **enteros** y de **punto flotante**

## byte

$-2^7$  to  $2^7 - 1$  (-128 to 127)

8 bits, con signo

## int

$-2^{31}$  to  $2^{31} - 1$  (-2147483648 to 2147483647)

32 bits, con signo

## float

32 bits, IEEE 754

Negative range:  $-3.4028235E+38$  to  $-1.4E-45$   
Positive range:  $1.4E-45$  to  $3.4028235E+38$

## short

16 bits, con signo

$-2^{15}$  to  $2^{15} - 1$  (-32768 to 32767)

## long

64 bits, con signo

$-2^{63}$  to  $2^{63} - 1$

(i.e., -9223372036854775808 to 9223372036854775807)

## double

64 bits, IEEE 754

Negative range:  $-1.7976931348623157E+308$  to  $-4.9E-324$   
Positive range:  $4.9E-324$  to  $1.7976931348623157E+308$





# Tipo de datos no numéricos

Entre los no numéricos, tenemos los de carácter y los booleanos.

## boolean

Un tipo de datos booleano declara una variable con el valor **true** (verdadero) o **false** (falso)



A

## char

Un tipo de datos de carácter representa un único carácter.



## "String"

Una cadena es una secuencia de caracteres.  
No es primitivo, pero ...



# Operadores aritméticos

Los operadores aritméticos se utilizan para modificar variables.

Simbolo	Nombre	Ejemplo	Resultado
+	Adición	34 + 1	35
-	Sustracción	34.0 - 0.1	33.9
*	Multiplicación	300 * 30	9000
/	División	1.0 / 2.0	0.5
%	residuo o modulo	20 % 3	2



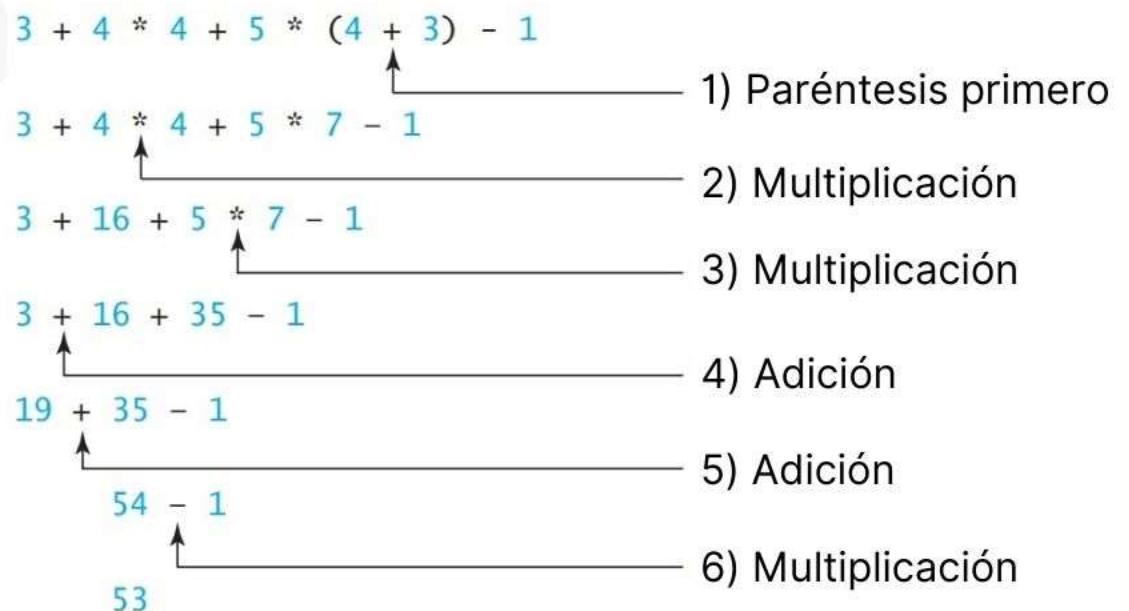


# Jerarquía de los operadores aritméticos

Los operadores aritméticos se utilizan para modificar variables.

## Constantes

- La jerarquía funciona tal y como es en la aritmética.
- Primero los paréntesis, pudiendo estos estar anidados.
- Después multiplicación, división y resto, modulo o residuo. Si una expresión contiene varios operadores de multiplicación, división y modulo , se aplican de izquierda a derecha.
- Por ultimo adición y sustracción. Si una expresión contiene varios operadores de suma y resta, se aplican de izquierda a derecha.





# Ejercicio

**1.- Hacer un programa en Java que dado un número entero de segundos calcule su equivalente en minutos con segundos.**

**Ejemplo: 400 segundos equivalen a 6 minutos con 40 segundos.**

**2.- Hacer un programa en Java que dada la altura y el peso de una persona calcule su índice de masa corporal.**



# Operadores de asignación aumentados.

Los operadores `+`, `-`, `*`, `/` y `%` se pueden combinar con el operador de asignación para formar operadores aumentados.

---

Operador	Nombre	Ejemplo	Equivalente
<code>+=</code>	Asignación de adición	<code>i += 8</code>	<code>i = i + 8</code>
<code>-=</code>	Asignación de resta	<code>i -= 8</code>	<code>i = i - 8</code>
<code>*=</code>	Asignación de multiplicación	<code>i *= 8</code>	<code>i = i * 8</code>
<code>/=</code>	Asignación de división	<code>i /= 8</code>	<code>i = i / 8</code>
<code>%=</code>	Asignación de modulo	<code>i %= 8</code>	<code>i = i % 8</code>



# Operadores de incremento y decrecimiento.

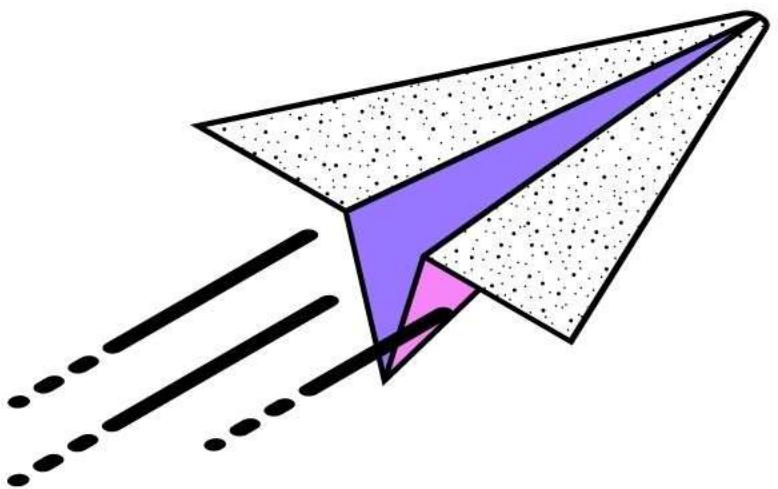
Los operadores de incremento (++) y decremento (--) son para incrementar y decrementar una variable en 1.

Operador	Nombre	Descripción	Ejemplo (i = 1)
<code>++var</code>	preincremento	Incrementa var en 1 y usa el nuevo valor de var en la declaración	<code>int j = ++i;</code> <code>// j es 2, i es 2</code>
<code>var++</code>	postincremento	Incrementa var en 1, pero use el valor original de var en la declaración	<code>int j = i++;</code> <code>// j es 1, i es 2</code>
<code>--var</code>	predecremento	Disminuye var en 1 y usa el nuevo valor de var en la declaración	<code>int j = --i;</code> <code>// j es 0, i es 0</code>
<code>var--</code>	postdecremento	Disminuye var en 1 y usa el valor original de var en la declaración	<code>int j = i--;</code> <code>// j es 1, i es 0</code>



# Sección 3

Operadores y construcción de sentencias lógicas



Creación de estructuras if, if/else.

Sentencias condicionales anidadas y encadenadas.

Igualdad entre cadenas (Strings).

Uso de operadores relacionales y condicionales.

Sentencia **switch**.

Evaluación de diferentes condiciones en un programa para determinar un algoritmo.



# Operadores relacionales.

Los operadores relacionales son de utilidad para realizar comparaciones y así determinar si se ejecuta o no cierta sentencia.

---

Operador	Nombre	Ejemplo var = 5	Resultado
<	Menor que	var < 0	false
<=	Menor o igual que	var <= 0	false
>	Mayor que	var > 0	true
>=	Mayor o igual que	var >= 0	true
==	Igual a	var == 0	false
!=	Distinto de	var != 0	true

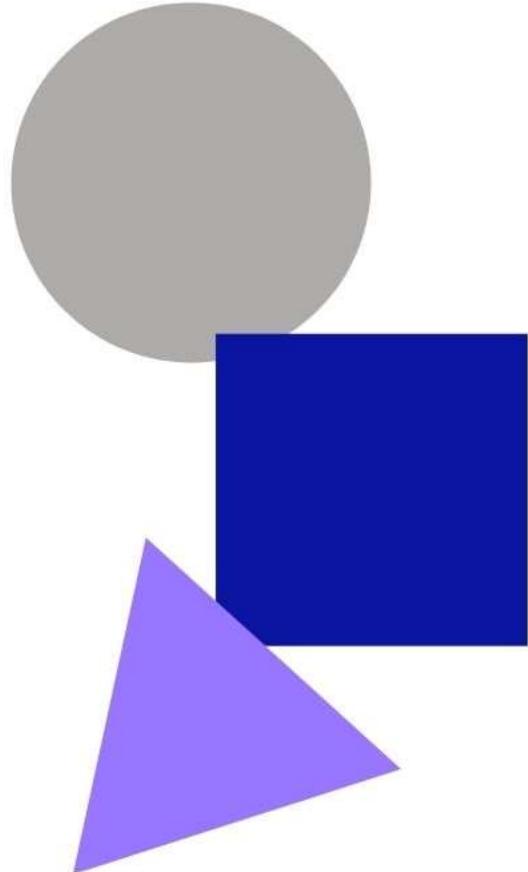


# Operadores lógicos.

Los operadores lógicos son de utilidad para realizar para evaluar expresiones booleana compuestas de mas de una expresión.

---

Operador	Nombre	Descripción
!	not	negación lógica
&&	and	conjunción lógica
	or	disyunción lógica
^	exclusive or	exclusión lógica



## Sentencia if

Una sentencia **if** ejecuta las instrucciones si la condición es verdadera.

## Que sentencias que permite Java

Java tiene varios tipos de sentencias de selección: sentencias if simples, sentencias if-else, sentencias if anidadas, sentencias switch y expresiones condicionales.

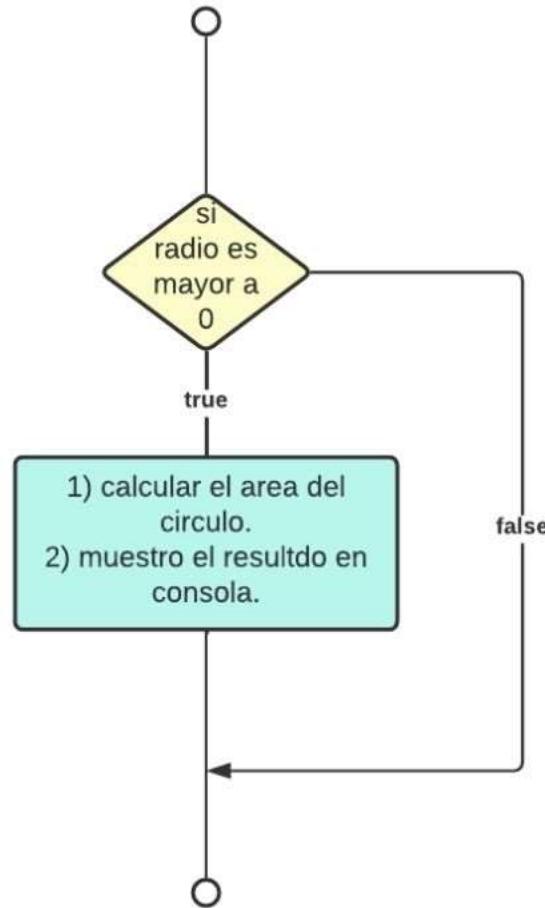
## Como luce una sentencia if

```
if (expresión-booleana) {  
    instrucción(es);  
}
```



# Diagrama de flujo

como se comporta una sentencia if





## Sentencia if-else

Una sentencia **if** ejecuta las instrucciones delimitadas por ella si la condición es verdadera, en caso contrario ejecutara las contenidas en el bloque delimitado por la palabra **else**.



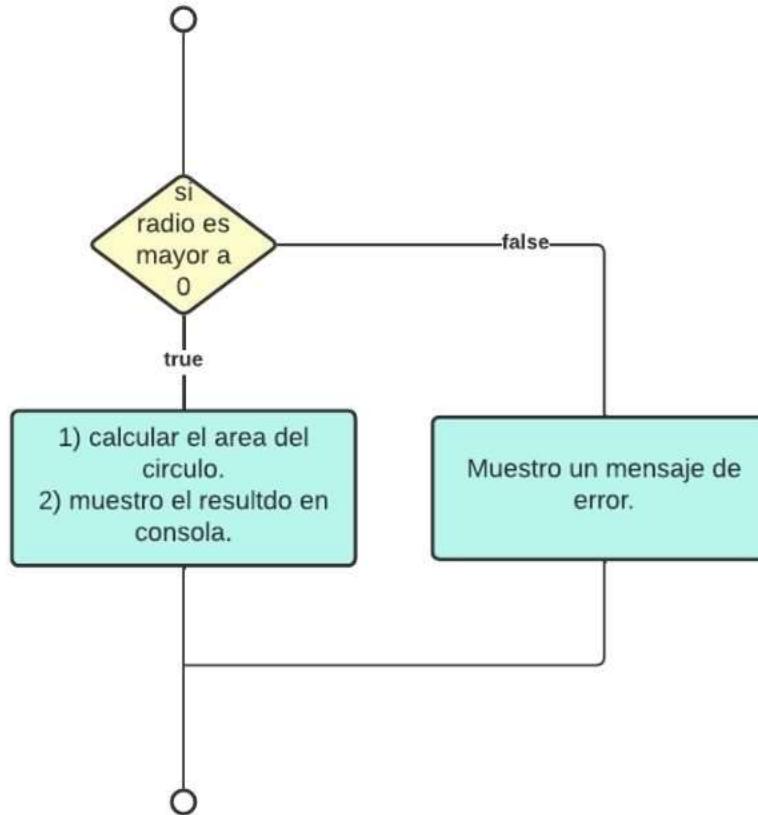
## Como luce una sentencia if-else

```
if (expresión-booleana) {  
    instrucción(es); //para el caso true  
}else{  
    instrucción(es); //para el caso false  
}
```



# Diagrama de flujo

como se comporta una sentencia if-else





### Como luce una sentencia if anidada.

```
if (expresión-booleana 1) {  
    if(expresión-booleana 2){  
        instrucción(es); // para el caso en que ambas condiciones  
        // resulten verdaderas  
    }  
}
```

### Como luce una sentencia if encadenada.

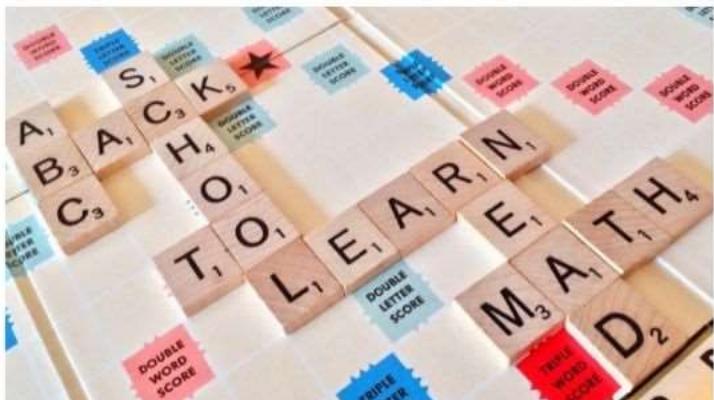
A graphic on the left side of the slide shows two simple 3D white human figures standing next to a grey signpost. One figure is pointing towards the signpost, which has three arrows pointing in different directions. This visual metaphor represents the flow of control or the multiple paths that can be taken in nested if statements.

```
if (expresión-booleana 1) {  
  
    instrucción(es); // para el caso verdadero  
  
}else if(expresión-booleana 2){  
  
    instrucción(es); // para el caso falso en 1 y verdadero en 2.  
}else{  
  
    instrucción(es); // para el caso falso tanto en 1 como en 2.  
}
```



## Comparación de cadenas con el operador ==

- Se utiliza para comparar cadenas y objetos en general.
- == es un **operador**
- == compara la referencia o dirección de memoria del objeto en el Heap, verifica que la dirección sea la misma o no.



## Comparacion de cadenas con el metodo .equals()

- Se utiliza también para comparar cadenas y objetos en general.
- .equals() es un método.
- Este no verifica dirección de memoria, este se enfoca en el contenido, en este caso en las letras que componen la cadena.



## Sentencia switch

Una sentencia **switch** ejecuta instrucciones basadas en el valor de una variable.

### Como luce una sentencia switch

```
switch (expresión-switch){  
    case valor1: instrucción(es)1;  
        break;  
  
    case valor2: instrucción(es)2;  
        break;  
  
    case valorN: instrucción(es)N;  
        break;  
  
    default: instrucción(es); // ningún caso se cumplió  
}
```



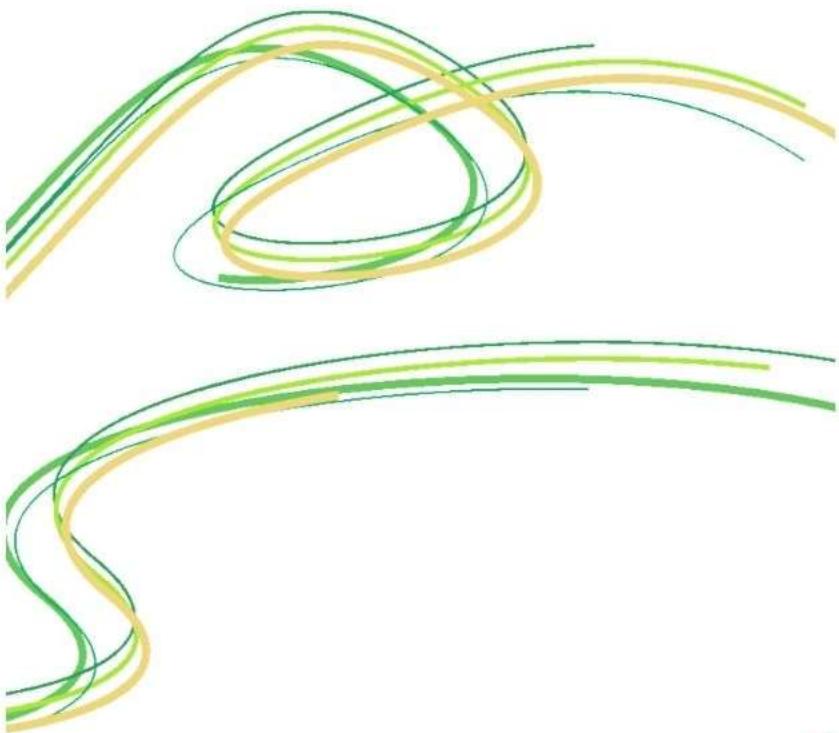
## Ejercicios:

- 1. Declarar una variable entera, asignarle un valor y determinar si es par.**
  
- 2. Declarar una variable entera, asignarle un valor y determinar si está en el conjunto [2,8].**
  
- 3. Declarar una variable tipo double y asignarle un valor, determinar si sus decimales son mayores o menores a 0.5.**  
**Ejemplo:** 3.8 su parte decimal es mayor a 0.5 mientras que 3.45 su parte decimal es menor a 0.5.



# Sección 3

Uso de ciclos



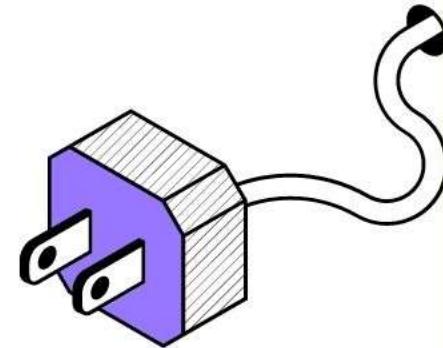
Crear ciclos while y ciclos while anidados.

Alcance de las variables.

Crear ciclos do-while

Crear ciclos for

Utilizar un ArrayList con el ciclo for.

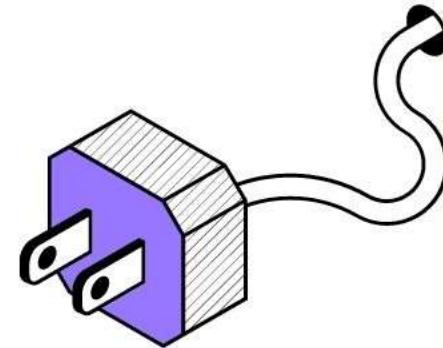


# Ciclo while

## Características

- Un bucle o ciclo while ejecuta declaraciones repetidamente mientras una condición es verdadera.
- La sintaxis del ciclo while es:  
**while** ( condición ){  
    //cuerpo del ciclo  
    Instrucción (es);  
}





# Ciclos while anidados

## Características

- Es posible definir un ciclo **while** dentro de otro.
- La sintaxis de ciclos while anidados es:  
**while** ( condición1 ){  
    **while** ( condición2 ){  
        //cuerpo del ciclo  
        Instrucción (es);  
    }  
}





# Alcance de las variables.

## Como funciona



- El alcance de una variable es la parte del programa donde se puede hacer referencia a la variable.
- Una variable definida dentro de un método se denomina variable local.
- El alcance de una variable local comienza desde su declaración y continúa hasta el final del bloque que contiene la variable.
- Una variable local debe declararse y asignarle un valor antes de que pueda ser utilizada.



# Alcance de las variables.

```
public static void method1() {  
    .  
    .  
    for (int i = 1; i < 10; i++) {  
        .  
        .  
        int j;  
        .  
        .  
    }  
}
```

Alcance de *i* →

Alcance de *j* →

A diagram illustrating variable scope. A vertical rectangle represents the scope of the variable *i*, which starts at the opening brace of the for-loop and ends at its closing brace. Inside this scope, the variable *j* is declared. Another vertical rectangle represents the scope of *j*, which starts at its declaration and ends at the closing brace of the entire method *method1()*. Arrows from the text "Alcance de i" and "Alcance de j" point to the start of these respective scopes.

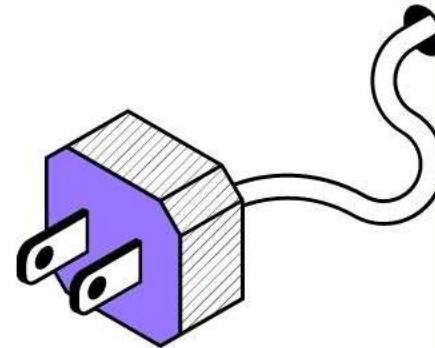
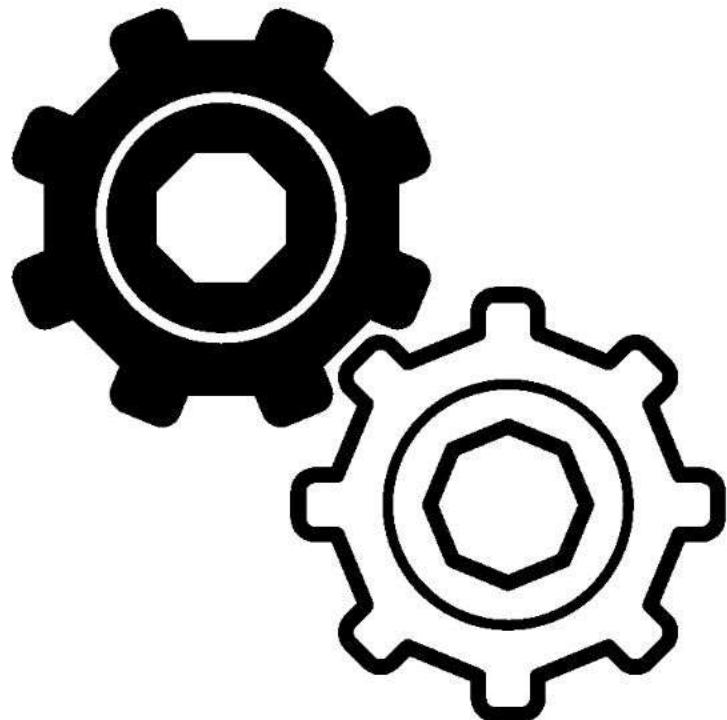


# Ciclo do while

## Características

- Un ciclo do-while es lo mismo que un ciclo while excepto que primero ejecuta el cuerpo del ciclo y luego verifica la condición de continuación del ciclo.
- La sintaxis del ciclo do while es:

```
do{  
    //cuerpo del ciclo  
    Instrucción (es);  
}while ( condición );
```



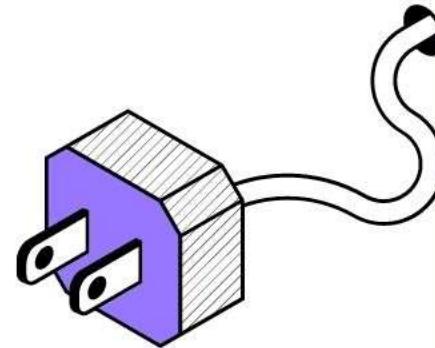
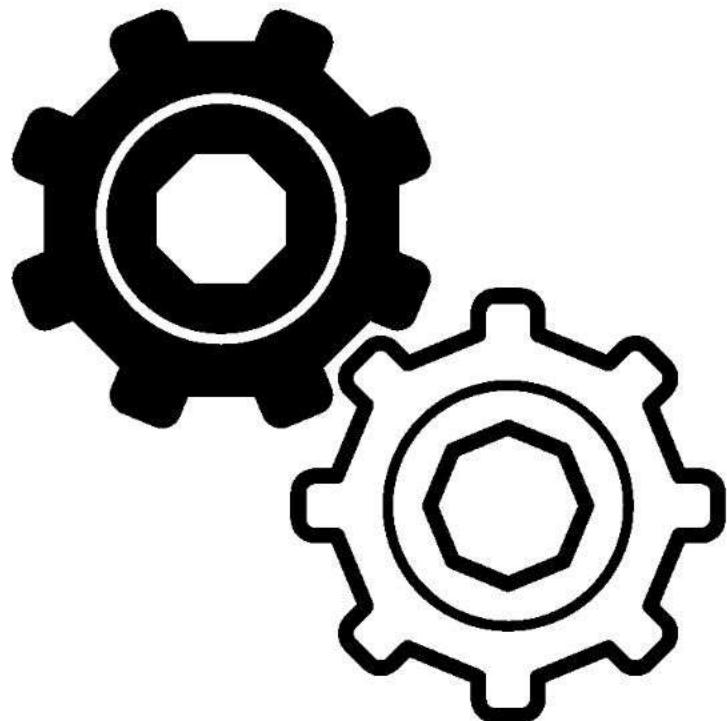


# Ciclo for

## Características

- Un ciclo for también tiene una condición, pero esta condición, se sabe desde un principio cuando dejará de ser verdadera.
- La sintaxis del ciclo for es:

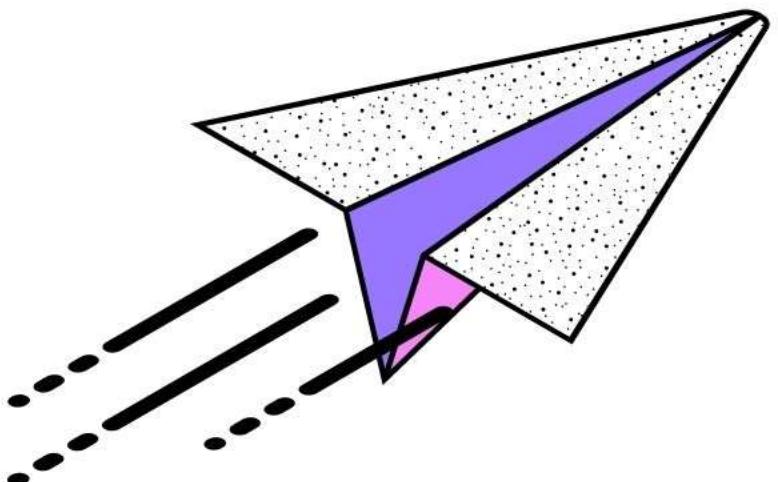
```
for (int i = valorInicial; i < valorFinal; i++) {  
    // Loop body ...  
}
```





# Sección 3

Creación y uso de arreglos.



Acceso a un valor en un arreglo o ArrayList.

Declarar, instanciar e inicializar un arreglo unidimensional.

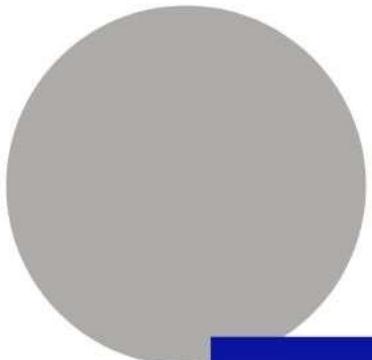
Uso de la sentencia import para trabajar con API's existentes de java.

Declarar, instanciar e inicializar un arreglo bidimensional.

Crear e inicializar un ArrayList.

Utilizar el vector de argumentos. (args Array)

Uso de un ciclo **for** para procesar un arreglo



## Que es un arreglo

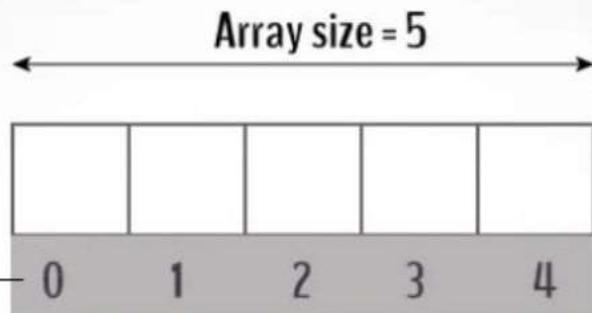
Es una colección de datos de un mismo tipo. Una sola variable de tipo arreglo o array puede hacer referencia a una gran colección de datos.

## Conceptos de arreglos

- Una vez que se crea un arreglo, su tamaño es fijo.
- Una variable de referencia de matriz se utiliza para acceder a los elementos de una matriz mediante un índice.

## Como se declara un arreglo en Java

**tipoElemento variableReferencia[ ] = new tipoElemento [tamaño]**





## Creación de un arreglo

```
double[] miArray = new double[10];
```

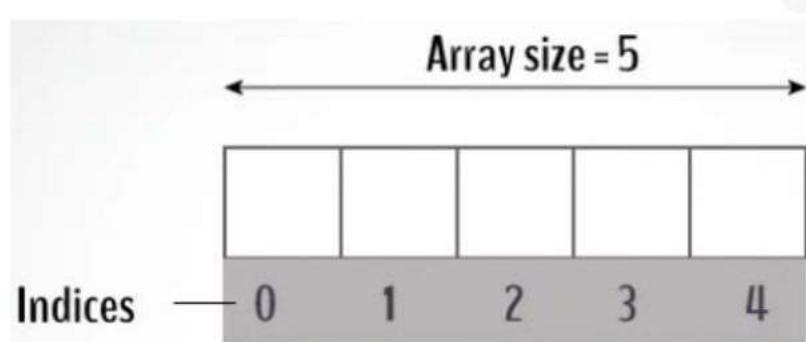
Se crea un arreglo, que guardara datos de tipo double, con una capacidad de 10. El arreglo tiene valores por defecto.

## Creación de un arreglo inicializado

```
double[] miArray= {1.9, 2.9, 3.4, 3.5};
```

Se crea un arreglo, que guarda los valores que están entre las llaves. El arreglo contiene valores específicos desde su creación.

## Asignar valores a un arreglo con valores por defecto.



```
miArreglo[0] = 4.0;  
miArreglo[1] = 34.33;  
miArreglo[2] = 34.0;  
miArreglo[3] = 45.45;  
miArreglo[4] = 99.993;  
miArreglo[5] = 11123;
```



## ArrayList

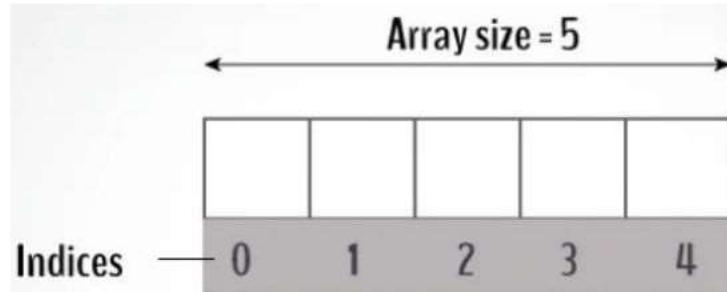
Una ArrayList se implementa usando un arreglo. En un arreglo convencional el tamaño de este era fijo una vez definido, y este no podía cambiar, en un ArrayList esto no es así, ya que el arreglo crece de manera dinámica, es decir su tamaño puede variar.

## Creación de un ArrayList

```
import java.util.*;  
  
ArrayList<String> miArrayList = new ArrayList<>();
```

## Agregar valores a un ArrayList

```
miArrayList.add("Arturo");  
miArrayList.add("Maria");  
miArrayList.add("Pedro");  
miArrayList.add("Fernanda");
```





## Arreglo de argumentos (`String[ ] args`)

Es un arreglo que guarda objetos de tipo cadena (String), este es pasado justo cuando ejecutamos nuestro programa en la linea de comandos.

## Pasando valores al arreglo (`String[ ] args`)

Cuando se ejecuta un programa, es posible pasar valores para que estos sean almacenados en el arreglo args.

```
java nombrePrograma [argumento1, argumento2 ... ]
```

## Ejemplo

Suponiendo que tenemos un programa que recibe nombres de personas mediante la linea de comandos, para después imprimir un saludo para estas, la ejecucion seria la siguiente.

```
java SaludoPersonas Arturo Sofia Jorge Pedro Camila
```





## Procesado de un arreglo de manera eficaz.

- Existen maneras de realizar tareas que son repetitivas.  
Para el procesado de un arreglo, tenemos al ciclo **for**.
- Con el ciclo **for** es posible recorrer los elementos de un arreglo, para acceder a sus datos o modificarlos de una manera eficaz.

## Ejemplo de uso del ciclo for con un arreglo.

Suponiendo que tenemos definido un arreglo de tamaño 5 y que la referencia se llama miArreglo, es posible recorrerlo con la siguiente instrucción:

```
for ( int i = 0; i < miArreglo.length; i++ ){  
    System.out.println(miArreglo[i]);  
}
```