



# Java

```
println("Diego Iván Oliveros Acosta")
```

# Centrarse en lo importante

```
public class Main {  
    public static void main(String[] args) {  
        System.out.println("Hola mundo");  
    }  
}
```

# What is Java?

Java es un lenguaje de programación popular, creado en 1995.

Es propiedad de Oracle y más de 3000 millones de dispositivos ejecutan Java.

Se utiliza entre otras cosas para:

- Aplicaciones móviles (especialmente aplicaciones de Android)
- Aplicaciones de escritorio
- aplicaciones web
- Servidores web y servidores de aplicaciones
- Juegos
- Conexión a la base de datos

<https://www.tiobe.com/tiobe-index/>

- [Lenguajes de programación más buscados al contratar desarrolladores • ENTER.CO](#)
- [https://www.enter.co/especiales/dev/herramientas-dev/python-superara-a-java-en-popularidad-muy-pronto/](#)
- [https://www.enter.co/especiales/dev/javascript-programacion-mas-popular-stack-overflow/](#)

Jul 2021	Jul 2020	Change	Programming Language	Ratings	Change
1	1		C	11.62%	-4.83%
2	2		Java	11.17%	-3.93%
3	3		Python	10.95%	+1.86%
4	4		C++	8.01%	+1.80%
5	5		C#	4.83%	-0.42%
6	6		Visual Basic	4.50%	-0.73%
7	7		JavaScript	2.71%	+0.23%
8	9	▲	PHP	2.56%	+0.68%
9	13	▲	Assembly language	2.40%	+1.46%
10	11	▲	SQL	1.53%	+0.13%
11	20	▲	Classic Visual Basic	1.39%	+0.73%
12	8	▼	R	1.32%	-1.08%
13	12	▼	Go	1.17%	-0.04%
14	50	▲	Fortran	1.12%	+0.90%
15	24	▲	Groovy	1.09%	+0.51%
16	10	▼	Swift	1.07%	-0.37%

# Why Use Java?

Java funciona en diferentes plataformas (Windows, Mac, Linux, Raspberry Pi, etc.)

Es uno de los lenguajes de programación más populares del mundo.

Es fácil de aprender y fácil de usar.

Es de código abierto y gratuito.

Es seguro, rápido y potente.

Tiene un gran apoyo de la comunidad (decenas de millones de desarrolladores)

Java es un lenguaje orientado a objetos que brinda una estructura clara a los programas y permite que el código se reutilice, lo que reduce los costos de desarrollo.

Como Java está cerca de C ++ y C #, facilita a los programadores cambiar a Java o viceversa.

# Java Syntax

```
public class Main {  
    public static void main(String[] args) {  
        System.out.println("Hola mundo");  
    }  
}
```

```
C:\Users\diego>java -version  
C:\Users\Your Name>javac Main.java  
C:\Users\Your Name>java Main  
Hello World
```

- The main Method
- System.out.println()

# The main Method

---

Se requiere el método **main ()** y lo verá en todos los programas Java:

---

Dentro del método **main ()**, podemos usar el método **println()** para imprimir una línea de texto en la pantalla:

---

**Nota:** Las llaves **{}** marcan el comienzo y el final de un bloque de código.

---

**Nota:** Cada declaración de código debe terminar con un punto y coma.

---

# Java Comments

- Los comentarios de una sola línea comienzan con dos barras diagonales (//).

*Java ignora cualquier texto entre // y el final de la línea (no se ejecutará).*

- Los comentarios de varias líneas comienzan con / \* y terminan con \* /.

*Java ignorará cualquier texto entre / \* y \* /.*

## ¿Comentarios de una o varias líneas?

- Depende de usted cuál desea/requiere utilizar.
- Normalmente, usamos // para comentarios cortos y / \* \* / ya saben.

# Ejemplo:

```
/ * El siguiente código imprimirá
las palabras Hello World
a la pantalla, y es asombroso * /
public class Main {
    public static void main(String[] args) {
        // This is a comment
        System.out.println ("Hola mundo"); // This is a comment
    }
}
```

# Java Variables

- **int**: almacena enteros (números enteros), sin decimales, como 1996 o -300
- **float**: almacena números de punto flotante, con decimales, como 19,09 o -19,09
- **char**: almacena caracteres individuales, como 'a' o 'B'.

*Los valores de caracteres están rodeados por comillas simples*

- **boolean**: almacena valores con dos estados: **true** o **false**
- **String** : almacena texto, como "**Hola**".

*Los valores de cadena están rodeados por comillas dobles*

# Final Variables

```
final int edad = 15;
edad = 20; /* generará un error */
edad = 5;
float unFlotante = 5.99f;
char unaLetra = 'D';
boolean unBoleano = true;
String unaCadena = "ya se java";
```

# Declaring (Creating) Variables

- **tipo variable = valor;**
- **System.out.println(Variabes);**

```
String nombre = "Diego";
int edad = 26;
int altura;
altura = 180;
String nombre = "Diego";
int edad = 26;
int altura;
altura = 180;
System.out.println(Variabes);
altura = 185; //¿está bien?
System.out.println(Variabes)
```

# Display Variables

```
• //declarar
• String nombre = "Diego";
• String nombre2 = "Iván";
• String apellido = "Oliveros";
• //operar
• String Nombrecompleto = nombre2 + apellido;
• int x = 40;
• int y = 20;
• // Imprimir
• System.out.println ("Hola" + nombre);
• System.out.println (Nombrecompleto);
• System.out.println (x + y); // Imprime el valor de x + y
```

# Declare Many Variables

```
int x = 5, y = 6, z = 50;  
System.out.println(x + y + z);
```

“Lo importante no es que tanto, si no que tan bien lo hagamos”

```
int m = 60; // comentado  
int minutosPorHora = 60; // autocomentado
```

# Java Data Types

Data Type	Size	Descripción
byte	1 byte	Almacena números enteros del -128 al 127
short	2 bytes	Almacena números enteros de -32,768 a 32,767
int	4 bytes	Almacena números enteros de -2,147,483,648 a 2,147,483,647
long	8 bytes	Enteros desde -9,223,372,036,854,775,808 a 9,223,372,036,854,775,807
float	4 bytes	Números fraccionarios. Suficiente para almacenar de 6 a 7 dígitos decimales
double	8 bytes	Números fraccionarios. Suficiente para almacenar 15 dígitos decimales
boolean	1 bit	Almacena valores verdaderos o falsos
char	2 bytes	Almacena un solo carácter / Letra o valores ASCII

# Integer Types

```
byte entero = 100; //Byte  
short entero = 5000; //Short  
int entero = 100000; //Int  
long entero = 15000000000L; //Long
```

# Floating Point Type

```
float flotante = 5.75f;  
double flotante = 19.99d;
```

## SCIENTIFIC NUMBERS:

```
float f1 = 35e3f; //Scientific Numbers  
double d1 = 12E4d; //Scientific Numbers
```

# Booleans and Characters

```
boolean seguimos = true; //Boolean  
boolean paramos = false; //Booleans  
char miNota = 'B';  
char a = 65, b = 66, c = 67; //ASCII values
```

# ¿Qué es una tabla ASCII?

b <sub>7</sub> b <sub>6</sub> b <sub>5</sub>				0	0	0	1	0	1	0	0	1	0	1	1	1	
b <sub>4</sub>	b <sub>3</sub>	b <sub>2</sub>	b <sub>1</sub>	Column	0	1	2	3	4	5	6	7					
↑	↑	↑	↑	Row	0	NUL	DLE	SP	0	@	P	'	p				
0	0	0	0	0	0	SOH	DC1	!	1	A	Q	a	q				
0	0	0	1	1	1	STX	DC2	"	2	B	R	b	r				
0	0	1	0	2	ETX	DC3	#	3	C	S	c	s					
0	1	0	0	4	EOT	DC4	\$	4	D	T	d	t					
0	1	0	1	5	ENQ	NAK	%	5	E	U	e	u					
0	1	1	0	6	ACK	SYN	&	6	F	V	f	v					
0	1	1	1	7	BEL	ETB	'	7	G	W	g	w					
1	0	0	0	8	BS	CAN	(	8	H	X	h	x					
1	0	0	1	9	HT	EM	)	9	I	Y	i	y					
1	0	1	0	10	LF	SUB	*	:	J	Z	j	z					
1	0	1	1	11	VT	ESC	+	;	K	[	k	{					
1	1	0	0	12	FF	FS	,	<	L	\	l	l					
1	1	0	1	13	CR	GS	-	=	M	]	m	}					
1	1	1	0	14	SO	RS	.	>	N	^	n	~					
1	1	1	1	15	SI	US	/	?	O	-	o	DEL					

# Strings

```
String saludo = "Hola a todos los Programadores";
//también hay métodos de clase.
System.out.println("la longitud de la cadena
es: " + saludo.length());
System.out.println(saludo.toUpperCase());
System.out.println(saludo.toLowerCase());
System.out.println(saludo.indexOf("Hola"));
System.out.println(unSaludo + " " + otroSaludo);
System.out.println(unSaludo.concat(otroSaludo));
```

# Non- Primitive Data Types **(reference types)**

Están predefinidos en Java.

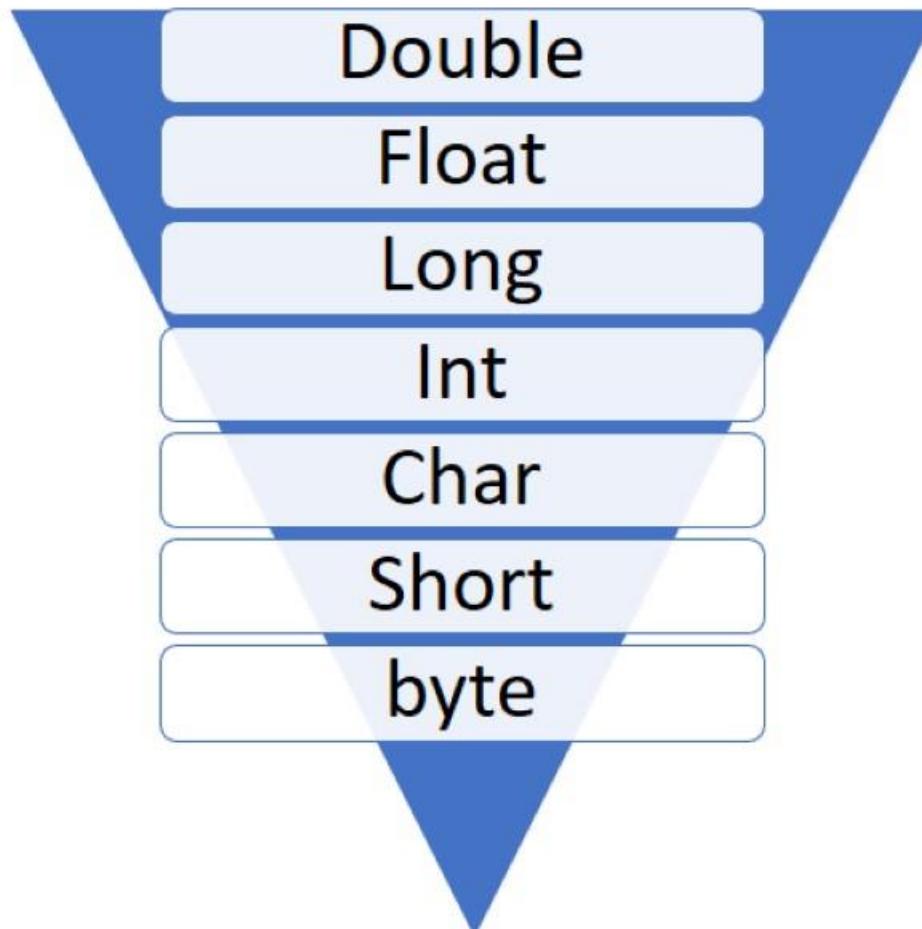
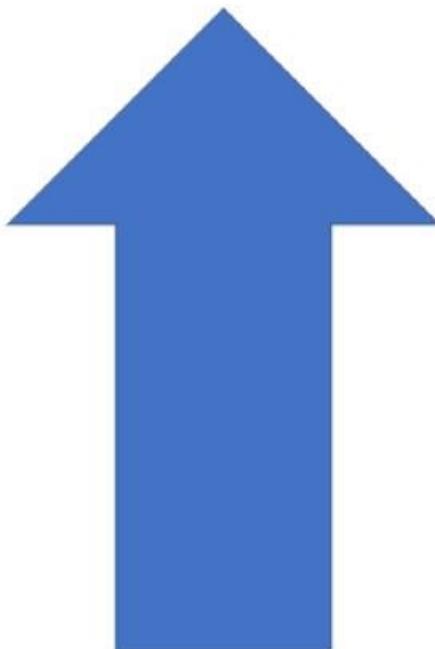
Los crea el programador y Java no los define (excepto **String**).

**Tipos NO primitivos son:**

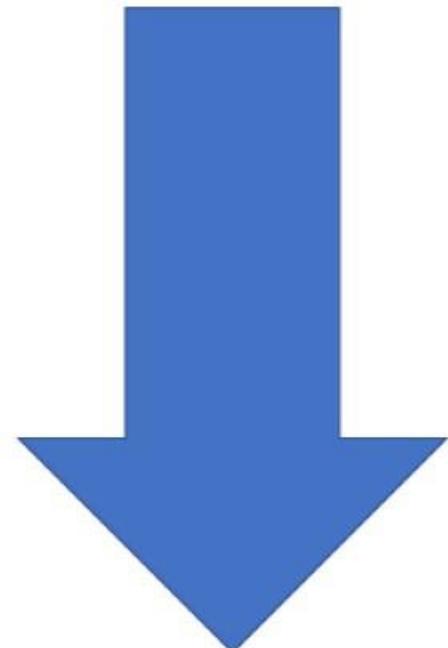
- Strings, Arrays, Classes, Interface, etc.

# Java Type Casting

**Widening  
Casting**  
(automatically)



**Narrowing Casting**  
•(manually)



# Java Type Casting

```
public class Main {  
    public static void main(String[] args) {  
        //Declaro:  
        int unEntero = 9;  
        double unDoble = unEntero; //opero? Automàtico  
        double otroDoble = 9.78d;  
        //Opero:  
        int otroEntero = (int) myDouble; // Casting Manual: doble a entero  
        //Muestro;  
        System.out.println(unEntero);    // Outputs 9  
        System.out.println(otroEntero);   // Outputs 9  
        System.out.println(unDoble);     // Outputs 9.0  
        System.out.println(otroDoble);    // Outputs 9.0  
    }  
}
```

# Java Arithmetic and logical Operators

Operador	Nombre	Descripción	Ejemplo
+	Adición	Suma dos valores	int x = x + y
-	Sustracción	Resta un valor de otro	x - y
*	Multiplicación	Multiplica dos valores	x * y
/	División	Divide un valor por otro	x / y
%	Modulo	Devuelve el resto de la división	x % y
++	Incremento	Aumenta el valor de una variable en 1	++x
--	Decremento	Disminuye el valor de una variable en 1	--x
&&	Logical and	Returns true if both statements are true	x < 5 && x < 10
	Logical or	Returns true if one of the statements is true	x < 5    x < 4
!	Logical not	Reverse the result, returns false if the result is true	!(x < 5 && x < 10)

## Ejercicio [ExpresionAritmética]

- Dada la expresión algebraica, escríbala en lenguaje de programación:
- $a = 5, b = 3; c = 8, d = 4, e = 2,$

$$a - \frac{b}{c} + \frac{b - \frac{c}{d}}{e}$$

# Java comparison and assignment operators

Comparison		
Operador	Descripción	Ejemplo
<code>==</code>	Igual a	<code>x == y</code>
<code>!=</code>	No es igual	<code>x != y</code>
<code>&gt;</code>	Mayor que	<code>x &gt; y</code>
<code>&lt;</code>	Menor que	<code>x &lt; y</code>
<code>&gt;=</code>	Mayor e igual a	<code>x &gt;= y</code>
<code>&lt;=</code>	Menor e igual a	<code>x &lt;= y</code>

assignment		
Operador	Ejemplo	Equivalencia
<code>=</code>	<code>x = 5</code>	<code>x = 5</code>
<code>+=</code>	<code>x += 3</code>	<code>x = x + 3</code>
<code>-=</code>	<code>x -= 3</code>	<code>x = x - 3</code>
<code>*=</code>	<code>x *= 3</code>	<code>x = x * 3</code>
<code>/=</code>	<code>x /= 3</code>	<code>x = x / 3</code>
<code>%=</code>	<code>x %= 3</code>	<code>x = x % 3</code>
<code>&amp;=</code>	<code>x &amp;= 3</code>	<code>x = x &amp; 3</code>
<code> =</code>	<code>x  = 3</code>	<code>x = x   3</code>
<code>^=</code>	<code>x ^= 3</code>	<code>x = x ^ 3</code>
<code>&gt;&gt;=</code>	<code>x &gt;&gt;= 3</code>	<code>x = x &gt;&gt; 3</code>
<code>&lt;&lt;=</code>	<code>x &lt;&lt;= 3</code>	<code>x = x &lt;&lt; 3</code>

Traduzca el código Ejercicio: operadores relaciones de Python a Java:

```
a = 16
b = True
c = 4*5
d = 8+1
e = 5/2
print(a > c and e <=d)
print(b or (d - e) > a / c)
print(not b)
print(not b and c < d or a/d <= e)
```

# Java program to illustrate bitwise

```
public class operators {  
    public static void main(String[] args){  
        int a = 5; // Initial values  
        int b = 7;  
        a &= b; // a=a&b  
        System.out.println("a&b = " + (a & b)); // 0101 & 0111=0101 = 5 bitwise and  
        System.out.println("a|b = " + (a | b)); // 0101 | 0111=0111 = 7 bitwise or  
        System.out.println("a^b = " + (a ^ b)); // 0101 ^ 0111=0010 = 2 bitwise xor  
        System.out.println("~a = " + ~a); // ~0101=1010 bitwise not  
        System.out.println("a= " + a);  
    }  
}
```

```
1
2     package triangulo;
3     import java.util.*;
4
5     public class Triangulo {
6
7         public static void main(String[] args) {
8             Scanner sc = new Scanner(System.in);
9             double a,b,c,p;
10            System.out.print("Introduzca longitud del primer lado del triángulo: ");
11            a = sc.nextDouble();
12            System.out.print("Introduzca longitud del segundo lado del triángulo: ");
13            b = sc.nextDouble();
14            System.out.print("Introduzca longitud del tercer lado del triángulo: ");
15            c = sc.nextDouble();
16            p = (a+b+c)/2;
17            System.out.println("Area -> " + Math.sqrt(p*(p-a)*(p-b)*(p-c)));
18        }
19    }
```

Output - triangulo (run) ×

	run:
	Introduzca longitud del primer lado del triángulo: 10
	Introduzca longitud del segundo lado del triángulo: 12
	Introduzca longitud del tercer lado del triángulo: 10
	Area -> 48.0
	BUILD SUCCESSFUL (total time: 41 seconds)

# If, else, else if

```
if (condición) {  
    // bloque de código a ejecutar si la condición es verdadera  
}
```

```
if (condición) {  
    // bloque de código a ejecutar si la condición es verdadera  
} else {  
    // bloque de código a ejecutar si la condición es falsa  
}
```

```
if (condition1) {  
    // bloque de código a ejecutar si condition1 es verdadera  
} else if (condición2) {  
    // bloque de código que se ejecutará si la condición1 es falsa y la condición2 es verdadera  
} else {  
    // bloque de código a ejecutar si la condición1 es falsa y la condición2 es falsa  
}
```

# CICLOS

*FOR, WHILE, DO WHILE*



# Objetivo

- Entender que son las estructuras de control cíclica.
- Entender la sintaxis y semántica de los ciclos en Java.
- Ejecutar una o varias líneas de código de acuerdo a sus necesidades.
- Realizar ejercicios de forma iterativa, incremental o repetitiva.
- Tener control y conocimiento sobre las iteraciones.

# Agenda

- Definición
- Sintaxis
- Patrones
- Ejercicios
  - 1. Números pares
  - 2. Múltiples definiciones
  - 3. Cuenta regresiva
  - 4. Contador
  - 5. Conversión grados Fahrenheit a Celsius
  - 6. Promedio de una lista enteros
- El juego alto bajo

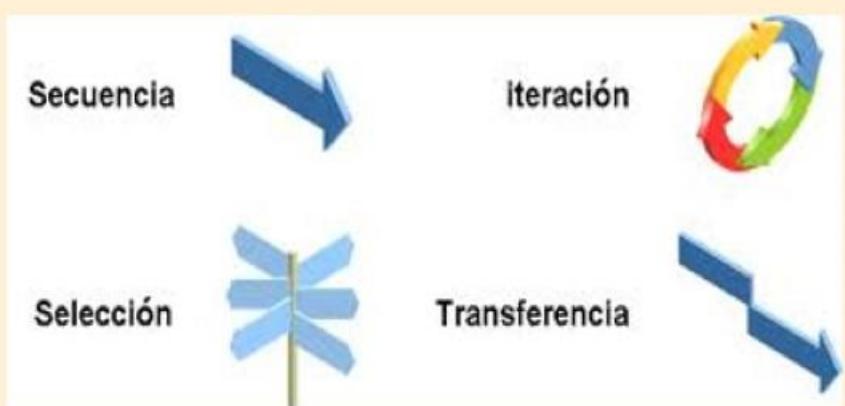
*Si lo logramos no necesitamos más.*

El teorema del programa estructurado, de *Böhm-Jacopini*, demuestra que todo programa puede escribirse utilizando únicamente las tres instrucciones de control siguientes:

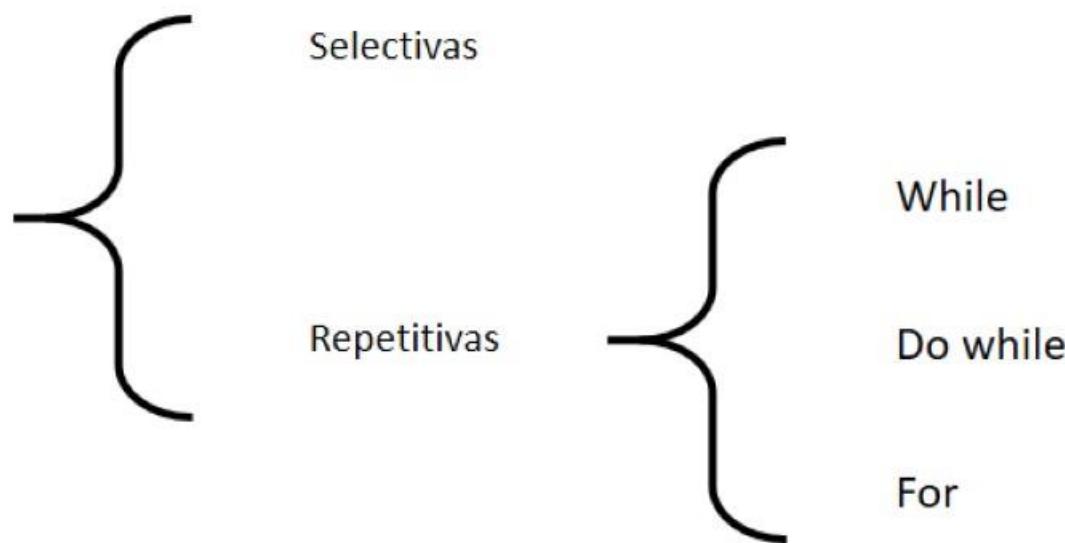
- **Secuencia**
- **Instrucción condicional.**
- **Iteración (bucle de instrucciones) con condición al principio.**

Solamente con estas tres estructuras o “patrones lógicos” se pueden escribir todos los programas y aplicaciones posibles.

Si bien los lenguajes de programación tienen un mayor repertorio de estructuras de control, éstas pueden ser construidas mediante las tres básicas.



# Estructuras de control



# Las estructuras de control repetitivas

Son aquellas en las que una sentencia o grupo de sentencias se repiten muchas veces. Este conjunto de sentencias se llama Bucle, también conocido como lazo o ciclo (Joyanes, 2003).

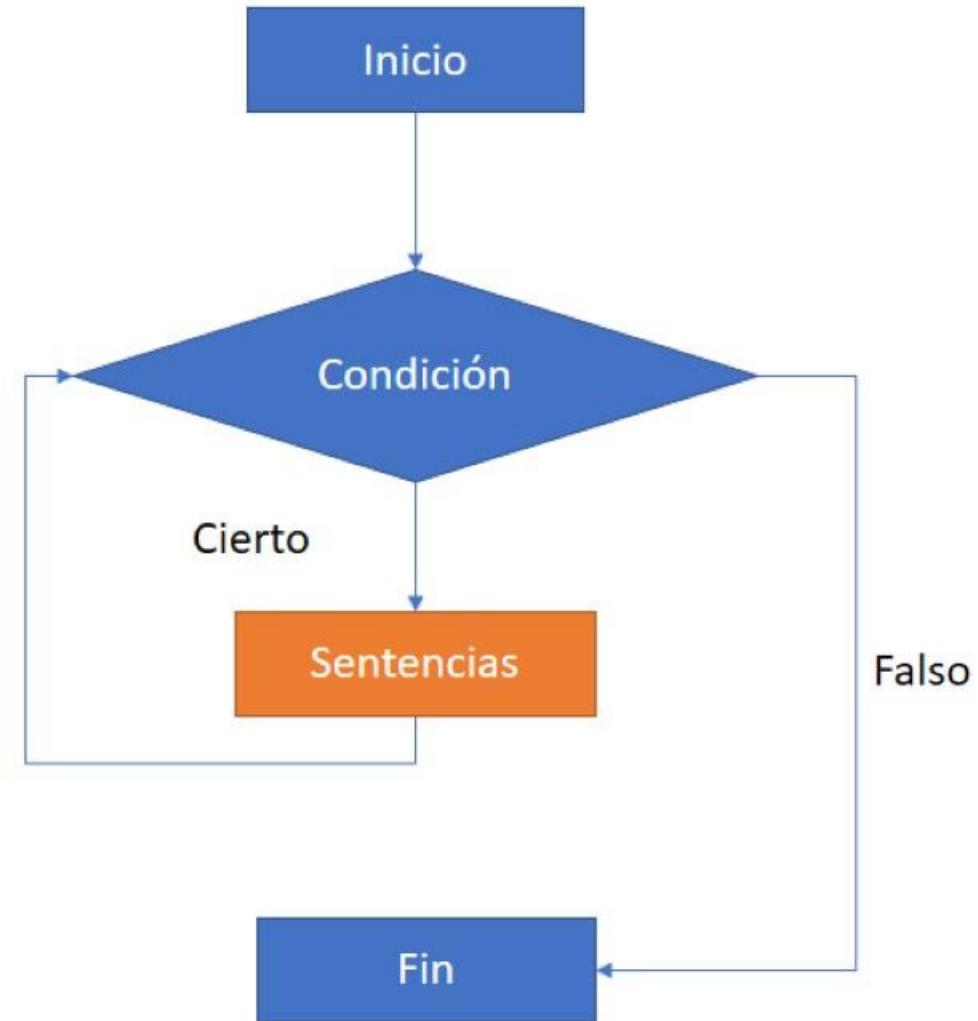


Un bucle es cualquier construcción de programa que repite una sentencia o secuencia de sentencias un número de veces (Borjas, 2012).

# Los ciclos o bucles

- Son una estructura de control
- Permite repetir una o varias instrucciones cuantas veces se necesite.
- Son una estructuras de control cíclica, nos permiten ejecutar una o varias líneas de código de forma iterativa (o repetitiva), pero teniendo cierto control y conocimiento sobre las iteraciones.

- For
- Do-while
- While



Las estructuras de control repetitivas controlan el número de veces que una sentencia o listas de sentencias se ejecutan, manteniendo un control, antes o después del cuerpo del ciclo.

Después del  
cuerpo del ciclo

### Estructura de control: Do while

Antes del cuerpo  
del ciclo

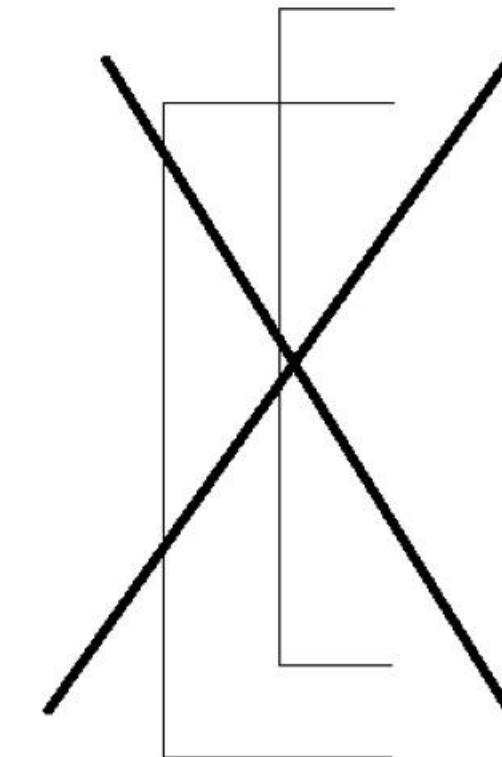
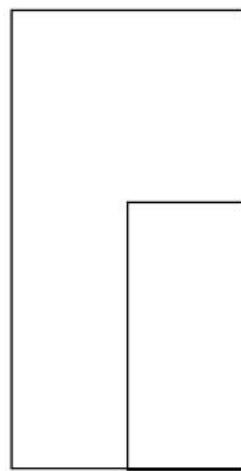
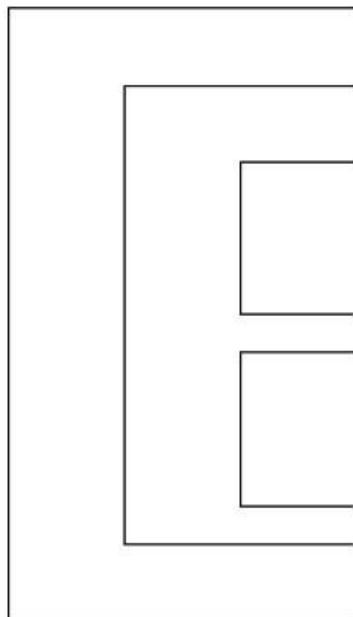
- **Estructura de control: while**
- **Estructura de control: for**

La diferencia radica en que la estructura que controla antes del ciclo no se ejecuta mientras no se cumpla la condición, sin embargo la del control después del ciclo, se ejecuta por lo menos una vez y evalúa, sólo se repite si se cumple la condición de lo contrario, termina el ciclo.

# Diseño de bucles.

## Bucles Anidados

PERMITIDAS Y PROHIBIDAS

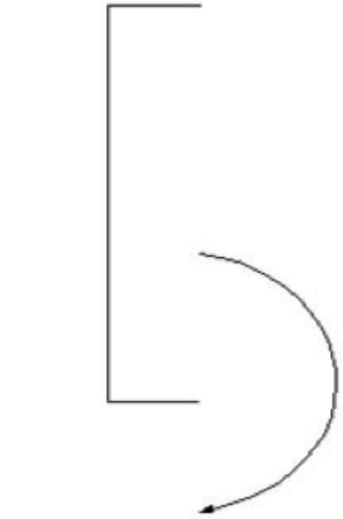


INDEPENDIENTES

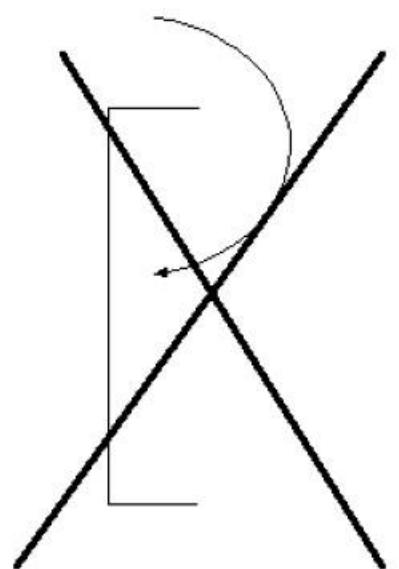
ANIDADAS

Diego Iván Oliveros Acosta, @scalapp [www.scalapp.co](http://www.scalapp.co)

NIDOS CRUZADOS



SALIR DEL BUCLE

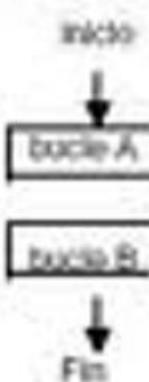


ENTRAR AL BUCLE

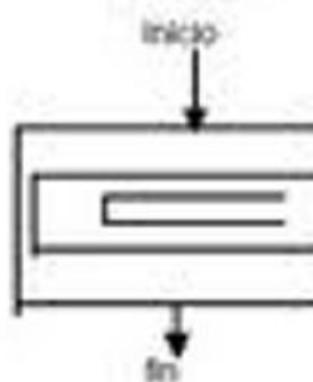
# Diseño de estructuras en fluajograma

- Independientes
- Anidados
- Cruzados
- Estructura repetitiva simple
- Estructura repetitiva anidada
- Estructura repetitiva compuesta múltiple.

a) Independientes



b) Anidados

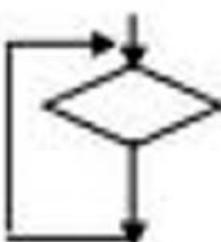


c) Cruzados

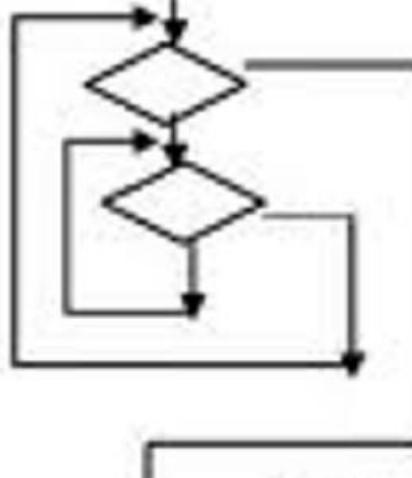


No es correcto su diseño

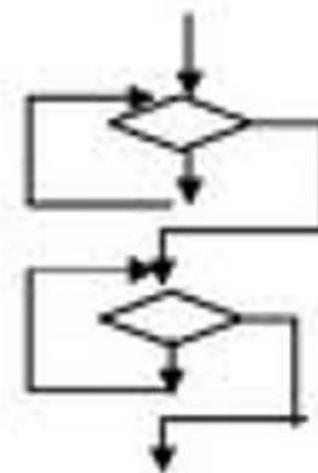
Diseño de las estructuras en fluajograma:



Estructura  
repetitiva simple



Estructura  
repetitiva  
Anidada



Estructura  
repetitiva  
Compuesta múltiple

# Ciclo for o ciclos para *Estructura, sintaxis y uso*

- Un ciclo for es una **estructura iterativa** para ejecutar **un mismo segmento** de código una cantidad de veces deseada; **conociendo previamente** un valor de inicio, un tamaño de paso y un valor final para el ciclo.
- **Ejercicio:**
- Queremos mostrar los **números pares** entre el 500 y el 1000.
- Tenemos:
  - ¿un valor inicial?
  - ¿un valor final?
  - ¿un tamaño de paso?

# Sintaxis del Ciclo For en Java:

```
for(int i = valor inicial; i <= valor final; i = i + paso)
{
    ....
    ....
    ....
    Bloque de Instrucciones....
    ....
    ....
}
```

- La sintaxis de un **ciclo for** es simple en Java, en realidad en la mayoría de los lenguajes de alto nivel es incluso muy similar
- Con tan solo tener bien claros los 3 componentes del ciclo tenemos prácticamente todo hecho.

# Ejemplo del Ciclo For en Java:

## *Ejemplo 1: Mostrar en pantalla los números pares*

- Vamos a retomar el ejemplo anterior, donde deseábamos sacar los números pares entre el numero 500 y el 1000
- Es un ejemplo sencillo con el que nos aseguraremos de haber comprendido bien lo anterior:

```
package javaciclosfor;  
public class JavaCiclosfor {  
    public static void main(String args[ ]) {  
        for(int i=500; i<=1000; i+=2) {  
            System.out.println(i);  
        }  
    }  
}
```

# switch

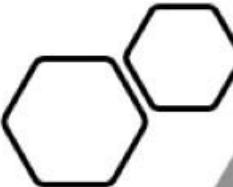
```
public class Main {  
    public static void main (String [] args) {  
        int día = 6;  
        switch (día) {  
            case 6:  
                System.out.println("Hoy es sábado");  
                break;  
            case 7:  
                System.out.println("Hoy es domingo");  
                break;  
            default:  
                System.out.println("Esperando el fin de semana");  
        }  
    }  
}
```

### Output - JavaCiclosfor (run)

	run:
	i= 1 j= 41
	i= 2 j= 4
	i= 3 j= 6
	i= 4 j= 8
	i= 5 j= 10
	i= 6 j= 12
	i= 7 j= 14
	i= 8 j= 16
	i= 9 j= 18
	i= 10 j= 20

- Ejemplo 2:

## Ejemplo 3: Cuenta regresiva en un ciclo for

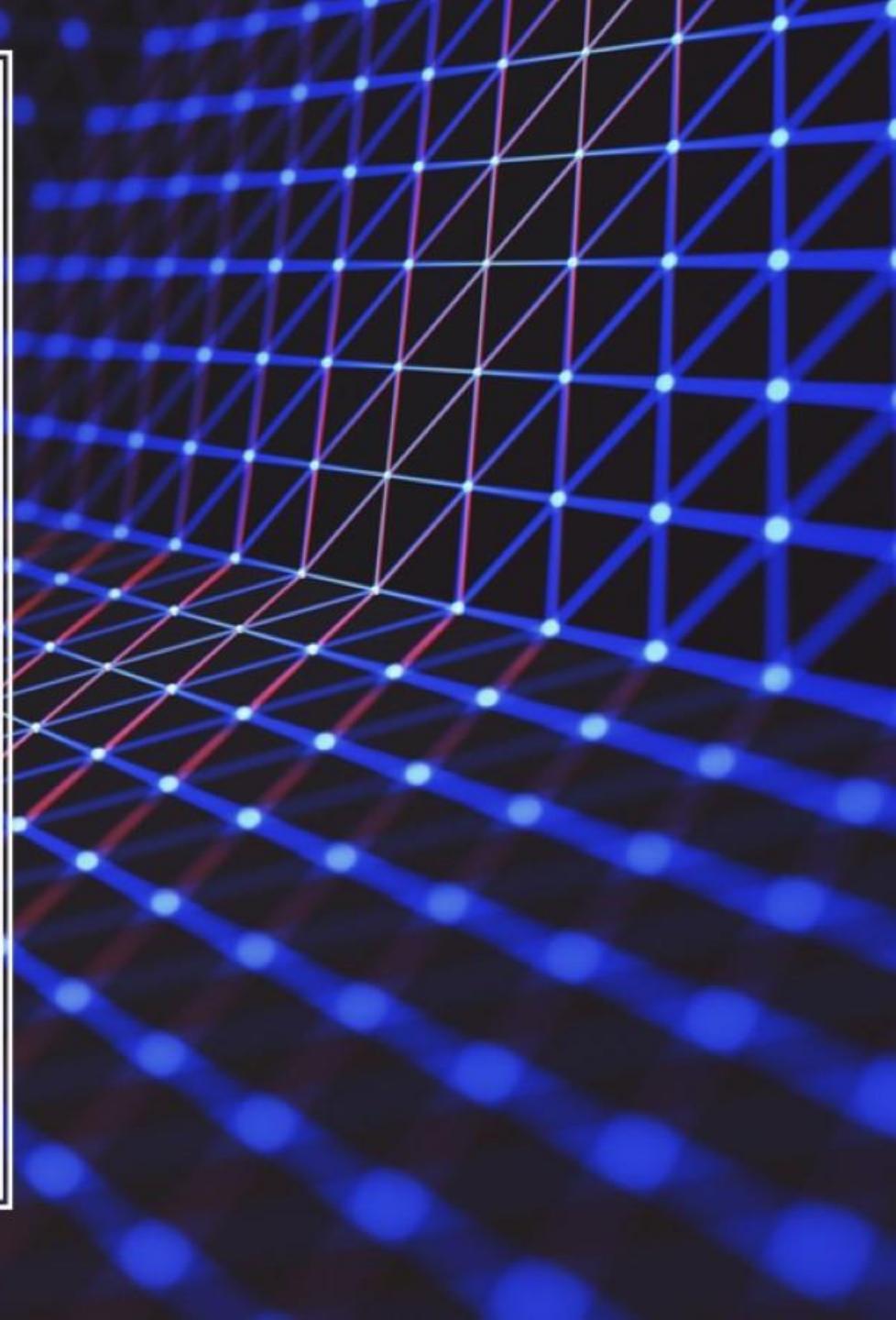


- Ahora veremos otro ejemplo sencillo en el cual haremos que el ciclo for también haga sus iteraciones en sentido inverso.
- Es decir disminuyendo el valor de la variable de control vamos a imprimir por pantalla una cuenta regresiva desde el número 100 hasta el 0, veamos:

## Ejemplo 4: Contador con un ciclo for

Para este ejemplo haremos algo un poco más complejo. El ejemplo consiste en contar al interior de un ciclo for, cuántos números entre el 0 y el 10.000 son múltiplos del 20. Para ello haremos uso del operador % (módulo) que obtiene el residuo de una división y también usaremos un pequeño condicional para verificar que el módulo sea cero al dividir por 20.

Nota: El operador de módulo (%) obtiene el residuo de una división, por tanto cuando el residuo es cero implica que la division es exacta y el dividendo es un múltiplo del divisor. Por ejemplo  $10 \% 3$  nos dará el residuo de dividir 10 entre 3, el cual es 1, si calculamos  $120 \% 20$  nos dará cero, pues 120 es múltiplo de 20 ( $20 * 6 = 120$ )



```
Package multiplos
public class Multiplos {
public static void main(String args[ ]){

    int contador = 0; //Iniciamos el contador en cero
    for(int i = 0; i <= 10000; i++){
        if(i % 20 == 0) //Preguntamos si el residuo es 0 (es múltiplo de 20) {
            contador++; //Si es múltiplo aumentamos el contador en 1
        }
        //Si no es múltiplo no hacemos nada
    }
    //Mostramos el valor del contador
    System.out.println(contador);
}
}
```

Este ciclo for nos permitirá saber que existen 501 múltiplos del número 20 en los números del 0 al 10000.

# El bucle while y el bucle do while

- presenta ciertas similitudes y ciertas diferencias con el bucle for. La repetición en este caso se produce no un número predeterminado de veces, sino mientras se cumpla una condición. Conceptualmente el esquema más habitual es el siguiente:
- El bucle **do while** es prácticamente igual al **while**, pero con la diferencia de que el código del bucle se ejecutara al menos una vez ya que la comprobación se hace después de cada iteración y no antes como en el caso del while.



## Ciclo while

```
1 package javaciclo;
2
3
4 public class Javaciclo {
5
6     public static void main(String[] args) {
7         int i = 0;
8         while (true) { //Condición trivial: siempre cierta
9             i++;
10            System.out.println ("El Valor de i: " + i);
11            if (i==15) { break;}
12        }
13    } //Cierre del main
14 } //Cierre de la clase
```

En este código hemos hecho algo un poco extraño. Como condición a evaluar hemos puesto “true”. Esto significa que la condición es siempre verdadera, lo que en teoría daría lugar a un bucle infinito y a un bloqueo del ordenador. Sin embargo, utilizamos un contador auxiliar que inicializamos en cero y en cada repetición del bucle aumentamos en una unidad. A su vez, introducimos una condición dentro del bucle según la cual cuando el contador alcanza el valor 9 se ejecuta la instrucción break.

Output - javaciclo (run) X

▶	El Valor de i: 1
▶	El Valor de i: 2
▶	El Valor de i: 3
▶	El Valor de i: 4
▶	El Valor de i: 5
▶	El Valor de i: 6
▶	El Valor de i: 7
▶	El Valor de i: 8
▶	El Valor de i: 9
▶	El Valor de i: 10
▶	El Valor de i: 11
▶	El Valor de i: 12
▶	El Valor de i: 13
▶	El Valor de i: 14
▶	El Valor de i: 15
✖	BUILD SUCCESSFUL (t)

```
2 package javaciclo;  
3  
4 public class Javaciclo {  
5  
6     public static void main(String[] args) {  
7  
8         int número = 788;  
9         int dígitos = 0;  
10        while ( número > 0 ) {  
11            número /=10;  
12            dígitos++;  
13        }  
14        System.out.println(dígitos);  
15    }  
16}
```

Realicemos un programa que cuente la cantidad de dígitos que posee un número. Para ello tendremos que dividir por diez el número que nos han dado, hasta que el resultado se vuelva cero. Entonces recurrimos al while para realice los ciclos necesarios.

```
1  
2 package javaciclo;  
3 public class Javaciclo {  
4     public static void main(String[] args) {  
5  
6         int número = 546823;  
7         int dígitos = 0;  
8         do {  
9             número /=10;  
10            dígitos++;  
11        }  
12        while ( número > 0 );  
13        System.out.println(dígitos);  
14    }  
15}
```

## Tabla de conversión grados Fahrenheit a Celsius

```
1
2 package javaciclo;
3 public class Javaciclo {
4     public static void main(String[] args) {
5
6         final double RANGO_INICIAL = 0; // limite inf. tabla
7         final double RANGO_FINAL = 100; // limite sup. tabla
8         final double PASO = 10 ; // tamaño paso
9
10        int fahrenheit;
11        double celsius;
12        fahrenheit = (int) RANGO_INICIAL;
13        System.out.printf("Fahrenheit \t Celsius \n");
14        while (fahrenheit <= RANGO_FINAL ) {
15            celsius = 5.* (fahrenheit - 32)/9;
16            System.out.printf("%7d \t %.3f \n", fahrenheit, celsius);
17            fahrenheit += PASO;
18        }
19    }
20 }
```

run:		
Fahrenheit	Celsius	
0	-17.778	
10	-12.222	
20	-6.667	
30	-1.111	
40	4.444	
50	10.000	
60	15.556	
70	21.111	
80	26.667	
90	32.222	
100	37.778	

BUILD SUCCESSFUL (total time: 0

## Ciclo do while

```
1
2 package javaciclo;
3
4 public class Javaciclo {
5
6     public static void main(String[] args) {
7         int contador = 0 ;
8
9         do { System.out.println ("Contando... " + (contador+1));
10            contador += 1;
11
12        } while (contador<10);      }
13
14 }
```



run:  
Contando... 1  
Contando... 2  
Contando... 3  
Contando... 4  
Contando... 5  
Contando... 6  
Contando... 7  
Contando... 8  
Contando... 9  
Contando... 10  
**BUILD SUCCESSFUL (t**

El bucle do ... while es muy similar al bucle while. La diferencia radica en cuándo se evalúa la condición de salida del ciclo. En el bucle while esta evaluación se realiza antes de entrar al ciclo, lo que significa que el bucle puede no llegar ejecutarse. En cambio, en un bucle do ... while, la evaluación se hace después de la primera ejecución del ciclo, lo que significa que el bucle obligatoriamente se ejecuta al menos en una ocasión

## Calcula promedio de una lista enteros

```
1 package javaciclo;
2 import java.util.Scanner;
3 public class Javaciclo {
4     public static void main(String[] args) {
5
6
7         int n, cont = 1; float x, promedio, suma = 0;
8         Scanner in = new Scanner( System.in );
9         System.out.printf("Promedio de numeros enteros\n");
10        System.out.printf("Cuantos numeros? ");
11        n = in.nextInt();
12        do {
13            System.out.printf("Ingresa numero %2d = ",cont);
14            x = in.nextFloat(); suma += x; ++ cont;
15        } while ( cont <= n );
16        promedio = suma / n;
17        System.out.printf("El promedio es: %.3f\n", promedio);
18
19    }
20 }
21
```

The screenshot shows an IDE interface with two main panes. On the left is the code editor containing the provided Java program. On the right is the debugger console window titled 'javacido (run) X'. The console output is as follows:

```
Debugger Console X javacido (run) X
run:
Promedio de numeros enteros
Cuantos numeros? 5
Ingresa numero 1 = 10
Ingresa numero 2 = 20
Ingresa numero 3 = 30
Ingresa numero 4 = 40
Ingresa numero 5 = 50
El promedio es: 30.000
BUILD SUCCESSFUL (total time:
```

# El juego alto bajo

- *Escriba una aplicación que permita jugar Alto-Bajo con el usuario.*
- *Se generarán números secretos aleatorios entre 1 y 100.*
- *Cuando el usuario propone un número, el programa responde Alto o Bajo dependiendo si es mayor o menor que el número secreto.*
- *El número máximo de intentos es 6.*



# Tarea

- 1. Escriba un método que devuelva true si el argumento es un número primo.
- 2. Utilizando el método del problema 1 escriba un método que liste todos los primos entre 1 y N , donde N es tecleado por el usuario.
- 3. Una terna de números  $a, b, c$  es cuadrada perfecta si  $a^2 + b^2 = c^2$ . Por ejemplo, 3, 4, 5 es una terna cuadrada perfecta. Escriba un programa para obtener todos las ternas cuadradas perfectas entre 1 y N, donde N es tecleado por el usuario. El programa no deberá repetir ternas, es decir, la terna 3, 4, 5 es la misma que la terna 4, 3, 5, y deberá imprimirse una sola vez.

# ¿Preguntas?



# Referencias

- [FUNDAMENTOS DE PROGRAMACION](#)
- **Oliveros Acosta. Diego Iván**
- Notas de clase.
  
- [FUNDAMENTOS DE PROGRAMACION](#)
- [Joyanes Aguilar, Luis](#)
- McGraw-Hill Interamericana de España S.L. / 978-607-15-1468-4
  
- [PROGRAMACION EN JAVA 6 ALGORITMOS](#)
- [PROGRAMACION ORIENTADA](#)
- [Joyanes Aguilar,Luis / Zahonero Martínez,Ignacio](#)
- McGraw-Hill Interamericana de España S.L. / 978-607-15-0618-4



# Referencias

- Albarrán, T. S. E. & Salgado, G. M. (2008). Programación Estructurada. Universidad Autónoma del Estado de México.
- Cairó, O. (2006). Fundamentos de Programación. Piensa en C. Pearson Educación.
- García, B., & Giner, J. R. (2008). Programación Estructurada en C. Madrid. Pearson Educación.
- Otros:
  - Balcázar, J. L. (1993). Programación metódica. España: McGraw-Hill.
  - Cairó, O. (1995). Metodología de Programación, Tomo I. Colombia. Computec.
  - Cairó, O. (1995). Metodología de Programación, Tomo II. Colombia. Computec.
  - Carrasco, R.; Patiño, I.; Santos, M. (2010). Fundamentos de Programación ( 2<sup>a</sup> ed). Ra-Ma, Alfaomega
  - Criado, A. (2006). Programación en lenguajes estructurados. México. Alfaomega-Ra ma.
  - Harel, D. (1987). Algorithmics, the Spirit of Computing. Massachusetts. Addison Wesley.
  - Joyanes A. L. (2008). Fundamentos de programación. Algoritmos, Estructura de datos y objetos (4<sup>a</sup> ed.). México. McGraw-Hill.
  - Joyanes A. L., Castillo, S. A., Sánchez, G. L, & Zahonero, M. I. (2002). Programación en C. Libro de problemas. México. McGraw-Hill.
  - Norton, P. (2006). Introducción a la Computación (6<sup>a</sup> ed.). México. McGraw-Hill.

- **Nueva certificación OCP Java SE 11 Developer: lo que debes saber para formarte y certificarte como Desarrollador Java**
- <https://blog.pue.es/oracle-nueva-certificacion-ocp-java-se-11-developer/>