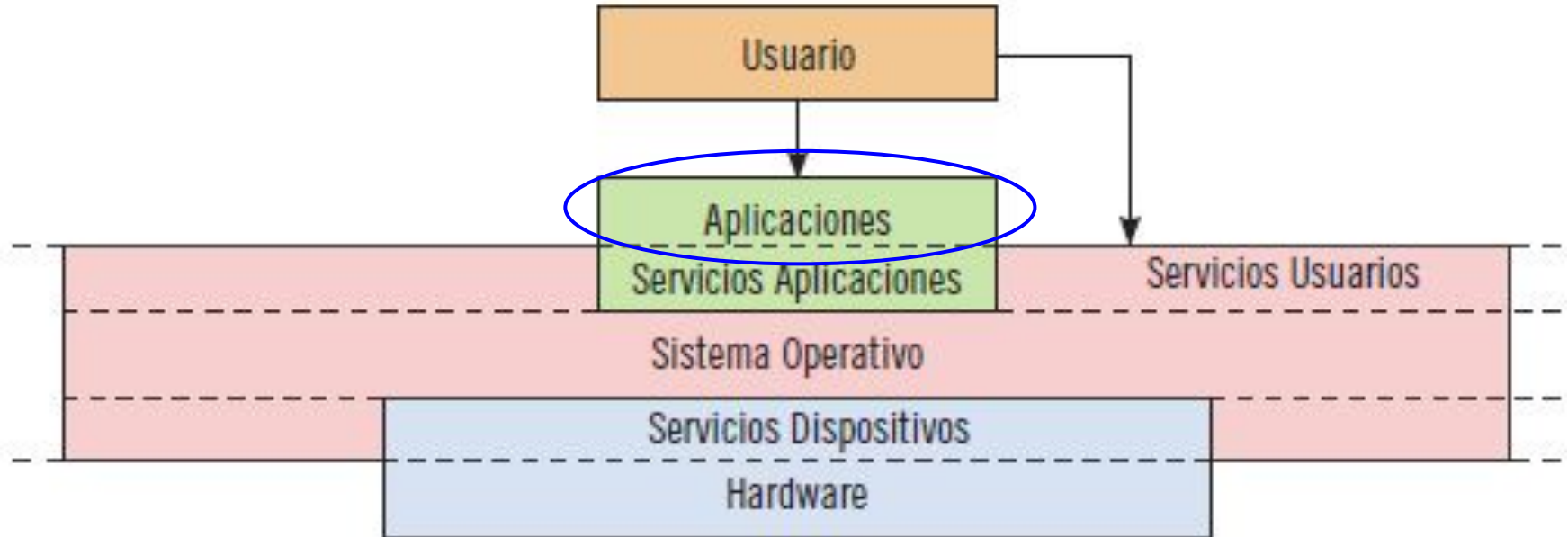




Programación 1

Programa, variables, constantes y operadores

¿Dónde vamos a trabajar?

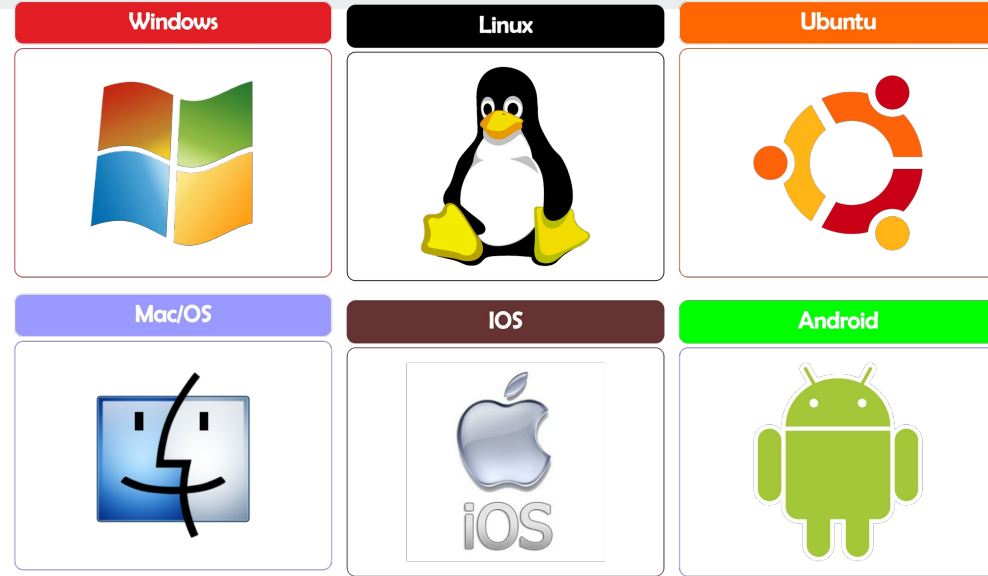


Sistema operativo

Programa base de la computadora.

Las funciones básicas son:

- administrar los recursos del sistema de computación,
- hacer de interfaz entre los dispositivos de hardware y los programas de usuario,
- organizar los archivos y directorios



Programa

Conjunto de instrucciones escritas en algún lenguaje de programación. Primero se diseña el **algoritmo** para resolver el problema (**papel y lápiz**) y luego escribimos el **programa** en algún lenguaje de programación para ejecutarlo en una **computadora**.

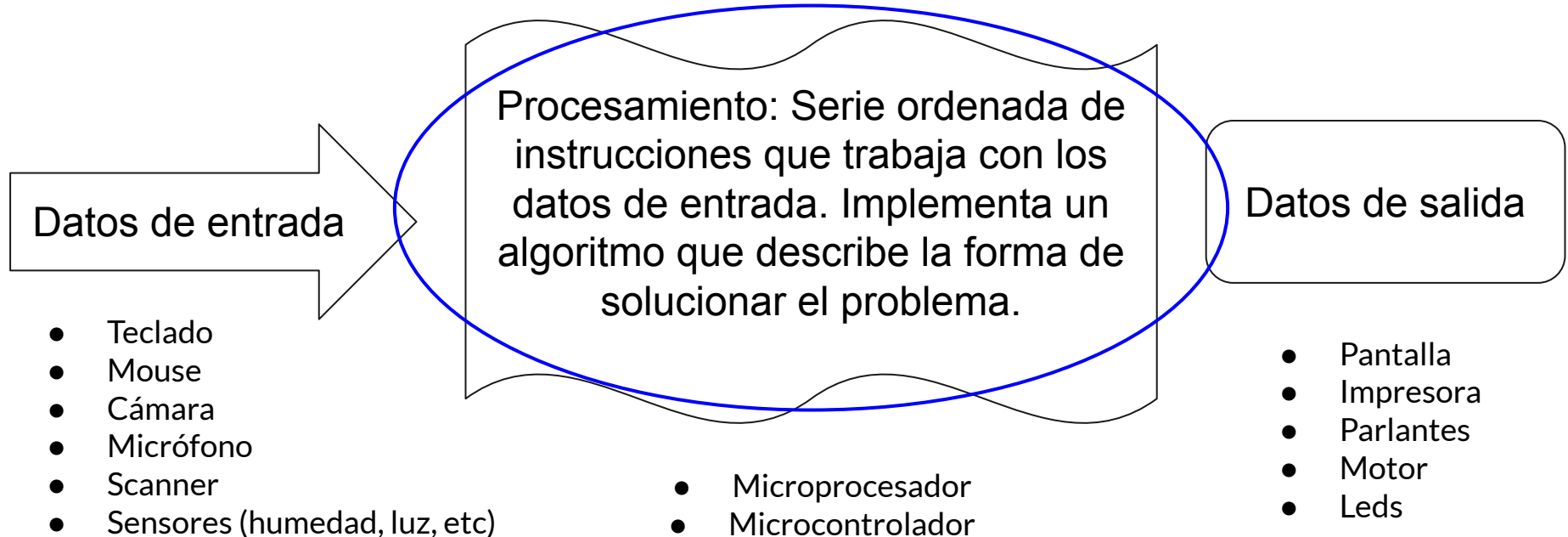
Los programas de **usuario** se utilizan para resolver problemas.

Ejemplo: un programa que calcula los cortes en una maderera para no desperdiciar material





Programa de usuario



Lenguajes de programación

Existen una gran cantidad de ellos y en general se los puede clasificar de bajo nivel o de alto nivel. También existe el lenguaje máquina. (0s y 1s)

Más usados: Java, JavaScript, Python, C/C++, PHP, Go, Ruby.

En este curso usaremos uno de alto nivel: **JAVA**

Es uno de los más usados en la industria



Lenguaje de programación JAVA

Orientado a objetos constituido por:

- Conjunto de instrucciones (println, for, while,...)
 - sumar, restar, mostrar un mensaje,...
- Conjunto de bibliotecas/librerías
- Conjunto de herramientas para el desarrollo de programas:
 - compilador de código binario que comprueba que esté bien escrito
 - generador de documentación JAVADOC





Algunos entornos que pueden usar

VS Code <https://code.visualstudio.com/docs/java/java-tutorial>

Codiva <https://www.codiva.io/> (On Line)

Eclipse <https://www.eclipse.org/downloads/>



Programación 1

Programa, variables, constantes y operadores

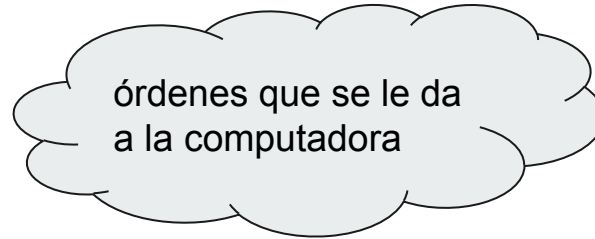
Los programas que vamos a escribir...

Todos los programas que vamos a escribir en Programación 1 tendrán la siguiente estructura:

```
public class Nombre_del_programa {  
    public static void main(String[] args) {  
        Sentencia 1;  
        Sentencia 2;  
        ...  
        Sentencia N;  
    }  
}
```

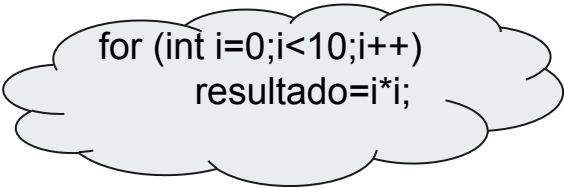


Orden secuencial



Instrucciones (primer programa)

```
public class Nombre_del_programa {  
    public static void main(String[] args) {  
        Sentencia 1;  
        Sentencia 2;  
        ...  
        Sentencia N;  
    }  
}
```



for (int i=0;i<10;i++)
 resultado=i*i;

En Java es obligatorio tener al menos dos bloques delimitados por “Llaves” => {}

1. `public class Nombre_del_programa` (será el nombre del archivo con extensión java, ej. programa.java)
2. `public static void main(String[] args)`: el método principal que se ejecuta primero

Estructura de un programa

Una única clase que contiene el método main() donde se resuelven los ejercicios.

Ejemplo:

```
// este programa permite resolver los ejercicios del práctico 1
/* Comentario, con inicio y final de marca
 * Los comentarios deberían contener sólo
 * información relevante para la lectura y
 * comprensión del programa.
 */
public class Practico1 {
    public static void main(String[] args) {
        System.out.println("Hola mundo");
    } // fin del método main
} // fin de la clase
```

Comentarios (no se tienen en cuenta al momento de la compilación y ejecución del programa)

Nombre de la clase y archivo: sin espacios ni caracteres especiales

Nombre del método

Estructura de un programa

```
public class Practico1 {  
    public static void main(String[] args) {  
        System.out.println("Hola mundo");  
    } // fin del método main  
} // fin de la clase
```

- **System.out.println**: provista por Java para imprimir (mostrar) un mensaje en la consola
- Lo que escribamos entre “ “ **dentro de los ()** se mostrará **textualmente** en la consola.
- El programa **siempre** comienza a ejecutarse **por la primera línea del bloque main**, continuando por las siguientes en orden secuencial.
- En este caso es la única, así que luego de imprimir por consola el mensaje **Hola mundo**, el programa finalizará.

¿Cómo harían para mostrar por consola otro texto debajo de Hola mundo?

Un problema real

Un programa para calcular el área de un rectángulo

Entrada

dato 1: altura 3 cm
dato 2: base 4 cm

Procesamiento

$\text{área} = \text{base} * \text{altura}$

Salida

$\text{área} = 12$

¿Dónde almacenar estos datos?



Variables

- Sirven para almacenar datos durante la ejecución del programa.
- Son sectores en la memoria de la computadora.
- El valor puede (debería) cambiar durante la ejecución del programa.

Ejemplo:

```
int altura=120;  
int base=208;  
float resultado=base*altura/2;  
altura=22;  
base=547;  
...
```



Nombre de variables en JAVA

- El nombre es lo que la identifica y permite referenciarla.
- Un nombre comienza por una letra, un carácter de subrayado (_) o un carácter de peso(\$). Ojo con los acentos!!!
- No hay límite máximo de caracteres para un nombre de variable.
- Java es *case sensitive*, se distinguen las mayúsculas de las minúsculas:
 - casa, CASA y cAsA son tres variables diferentes.

Tipo de una variable

Una **variable** además del **nombre**, tiene un **tipo** que define los valores que puede almacenar y las operaciones que se pueden realizar con ella.

Dato es todo con lo que opera un programa. Está asociado a una **variable** y ésta es la que se manipula para procesarlo.

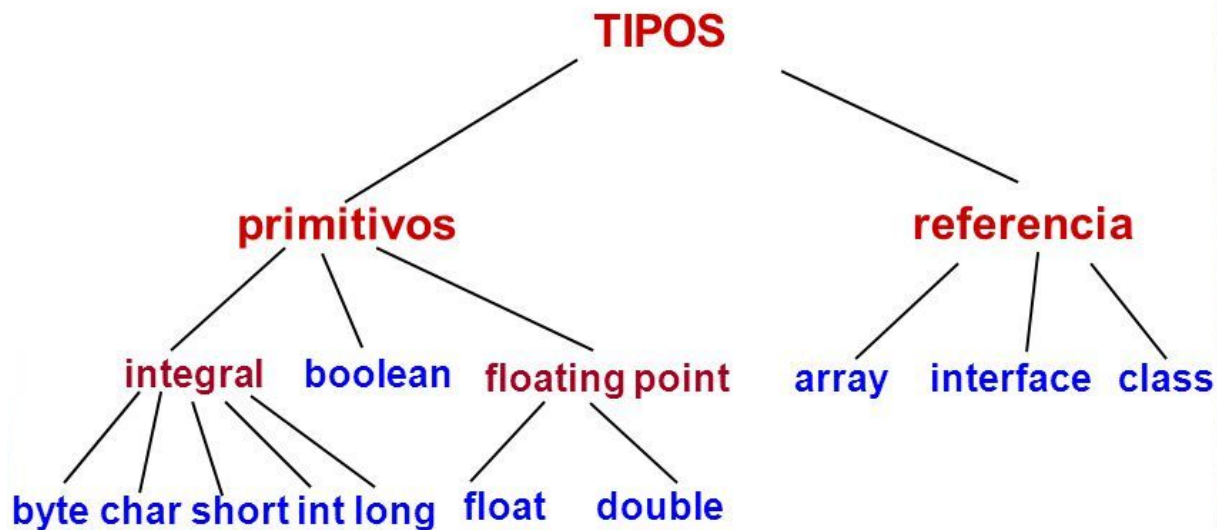
Ejemplo:

```
int edad=25;
```

Tipo primitivo Nombre de variable Dato / Valor

Tipos

Se dividen en dos grupos:





Tipos primitivos

Tipo	Representación / Valor	Tamaño (en bits)	Valor mínimo	Valor máximo	Valor por defecto
boolean	true o false	1	N.A.	N.A.	false
char	Carácter Unicode	16	\u0000	\uFFFF	\u0000
byte	Entero con signo	8	-128	128	0
short	Entero con signo	16	-32768	32767	0
int	Entero con signo	32	-2147483648	2147483647	0
long	Entero con signo	64	-9223372036854775808	9223372036854775807	0
float	Coma flotante de precisión simple Norma IEEE 754	32	$\pm 3.40282347E+38$	$\pm 1.40239846E-45$	0.0
double	Coma flotante de precisión doble Norma IEEE 754	64	$\pm 1.79769313486231570E+308$	$\pm 4.94065645841246544E-324$	0.0

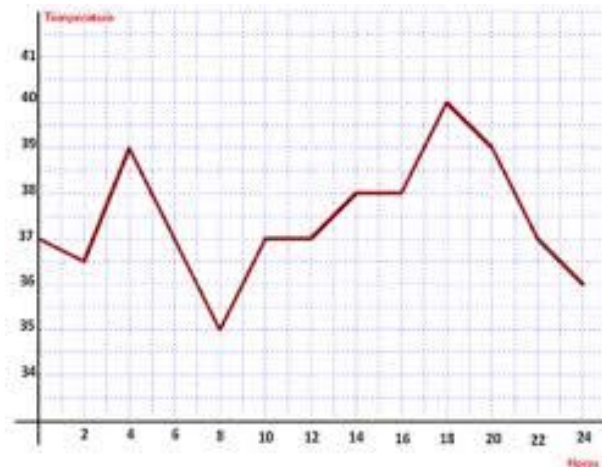
Declaración de variables

- Toda variable debe declararse antes de ser usada en el código de un programa en Java.
- En la declaración de una variable debe indicarse el identificador (**nombre**) y el **tipo** de dato asociado.

tipo_de_dato ident_1, ident_2, ..., ident_n;

Ej:

```
int nro;  
double x, y;  
int z;
```

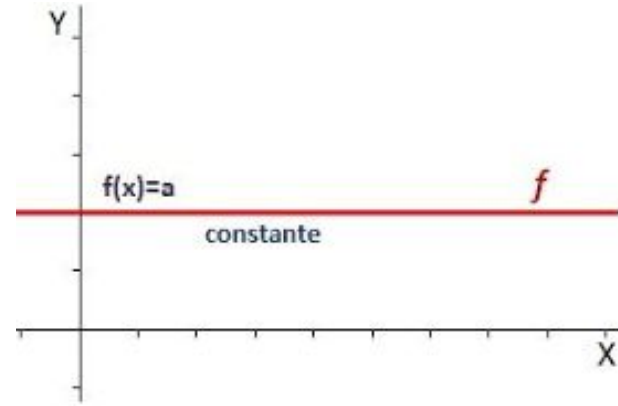


Declaración de constantes

- Constantes o variables finales: también sirven para almacenar datos pero no pueden modificarse posteriormente.
- Por ejemplo, el número **PI** y la aceleración de la gravedad **G**.
- Son similares a las constantes empleadas en otros lenguajes.
- Una vez inicializada su valor **no** puede ser modificado

Ej:

```
final double PI = 3.1415926;  
final double g = 9.81;
```



Literales

- Un literal es una constante que se puede asignar a una variable.
- Las constantes literales booleanas son **false** y **true**.
- Las constantes literales de tipo carácter aparecen entre comillas simples.
Letras mayúsculas ('A', 'B', 'C',...), minúsculas ('a', 'b', 'c',...), signos de puntuación (';', ':', '...' ...), dígitos, símbolos especiales ('#', '&', '%',...)
y literales que son caracteres de control:

Literal	Valor
\b	Retroceso o backspace
\t	Tabulación
\n	Nueva línea (enter)
\f	Salto de página



A codear

- Declarar variables de diferentes tipos
- Asignar valores y variables del mismo tipo
- Mostrar por consola un texto concatenado con una variable
- Uso del int y char
- Declarar constantes
- Asignar valores a constantes e intentar modificar
- Uso de literales de control

<Let's Code />

Carga de una variable entera por teclado

```
import java.io.BufferedReader;           //biblioteca que contiene operaciones de E/S
import java.io.InputStreamReader;       //biblioteca que contiene operaciones de E/S

public class programaCargaUnEntero {
    public static void main(String[] args) {
        int entero;
        try { //try define un bloque de manejo de posibles excepciones (errores)
            BufferedReader entrada = new BufferedReader(new InputStreamReader(System.in));
            System.out.println ("Ingrese un valor entero: ");
            entero = Integer.valueOf(entrada.readLine());
            System.out.println("Entero ingresado : " + entero);
        }
        catch (Exception exc ) { //se ejecuta si ocurre algún error de lectura
            System.out.println( exc );
        }
    }
}
```

Por ahora
solo vamos a
usarlas

bloque try-catch, se usa para
capturar excepciones cuando
ocurre un error.

Ejemplo parte 1/6

Vayamos por partes...



Entrada y salida desde consola, importación de bibliotecas

Una biblioteca es un conjunto de utilitarios que se acceden mediante sentencias para hacer operaciones predefinidas.

```
import java.io.BufferedReader;  
import java.io.InputStreamReader;
```

- Carga desde teclado de variables de distinto tipo a través de la sentencia **System.in**
`... entrada = new BufferedReader(new InputStreamReader(System.in));`
- Salida/impresión de variables a través de la sentencia **System.out**
`System.out.println ("Ingrese un valor entero: ");`

Ejemplo parte 2/6

Agregar las *bibliotecas* a utilizar y comenzar a definir las *variables*

```
import java.io.BufferedReader;  
import java.io.InputStreamReader;
```

```
public class programaCargaUnEntero {  
    public static void main(String[] args){  
        int entero;  
    }  
}
```

BUENAS PRÁCTICAS



Poner nombres representativos
que nos den una idea de que se
trata el programa o la variable

Ejemplo parte 3/6



*Una vez definidas las variables agregar el bloque **try-catch** para manejo de errores de ingreso de datos*

```
import java.io.BufferedReader;
import java.io.InputStreamReader;

public class programaCargaUnEntero {
    public static void main(String[] args){
        int entero;
        try { //try define un bloque de manejo de posibles excepciones
        }
        catch (Exception exc ) { //se ejecuta si ocurre algún error de lectura
            System.out.println(exc);
        }
    }
}
```

Ejemplo parte 4/6



*Crear el **buffer** donde se almacenarán los datos por teclado hasta que sean consumidos.*

```
import java.io.BufferedReader;
import java.io.InputStreamReader;

public class programaCargaUnEntero {
    public static void main(String[] args){
        int entero;
        try { //try define un bloque de manejo de posibles excepciones
            BufferedReader entrada = new BufferedReader(new InputStreamReader(System.in));
        }
        catch (Exception exc ) { //se ejecuta si ocurre algún error de lectura
            System.out.println(exc);
        }
    }
}
```

Ejemplo parte 5/6



Pedir el ingreso por teclado y consumir el buffer asignando lo ingresado a una variable.

```
import java.io.BufferedReader;
import java.io.InputStreamReader;
public class programaCargaUnEntero {
    public static void main(String[] args){
        int entero;
        try { //try define un bloque de manejo de posibles excepciones
            BufferedReader entrada = new BufferedReader(new InputStreamReader(System.in));
            System.out.println ("Ingrese un valor entero: ");
            entero = Integer.valueOf(entrada.readLine());
        }
        catch (Exception exc ) { //se ejecuta si ocurre algún error de lectura
            System.out.println(exc);
        }
    }
}
```

Ejemplo parte 6/6

Generar la **salida** del programa por consola

```
import java.io.BufferedReader;
import java.io.InputStreamReader;
public class programaCargaUnEntero {
    public static void main(String[] args){
        int entero;
        try { //try define un bloque de manejo de posibles excepciones
            BufferedReader entrada = new BufferedReader(new InputStreamReader(System.in));
            System.out.println ("Ingrese un valor entero: ");
            entero = Integer.valueOf(entrada.readLine());
            System.out.println ("Entero ingresado: " + entero);
        }
        catch (Exception exc ) { //se ejecuta si ocurre algún error de lectura
            System.out.println(exc);
        }
    }
}
```

BUENAS PRÁCTICAS



Indentar el código para mejor legibilidad usando la tecla Tab



Cargando más variables de otros tipos

```
import java.io.BufferedReader;
import java.io.InputStreamReader;
public class programaCargaNVariables { // programa carga más variables
    public static void main(String[] args){
        String cadenaCaracteres;
        float valorFlotante;
        double valorDouble;
        int entero;
        char unCaracter;
        try {
            BufferedReader entrada = new BufferedReader(new InputStreamReader(System.in));
            // ingresando datos por teclado
            System.out.println ("Ingrese una cadena de caracteres: ");
            cadenaCaracteres = entrada.readLine();
            System.out.println ("Ingrese un valor con decimal (float): ");
            valorFlotante = Float.valueOf(entrada.readLine());
            System.out.println ("Ingrese un valor con decimal (double): ");
            valorDouble = Double.valueOf(entrada.readLine());
```

BUENAS PRÁCTICAS



Poner comentarios en el código para saber lo que hace. Es muy fácil olvidarse después de un tiempo y de codear muchas líneas.

Cargando más variables de otros tipos (cont.)

```
System.out.println ("Ingrese un valor entero: ");
entero = Integer.valueOf(entrada.readLine());
System.out.println ("Ingrese un caracter: ");
unCaracter = entrada.readLine().charAt(0);
// mostrando por la consola lo que se ingresó
System.out.println("Cadena de caracteres ingresada: " + cadenaCaracteres);
System.out.println("Valor decimal ingresado (float): " + valorFlotante);
System.out.println("Valor decimal ingresado (double): " + valorDouble);
System.out.println("Entero ingresado: " + entero);
System.out.println("Caracter ingresado: " + unCaracter);
}
catch (Exception exc ) {
    System.out.println( exc );
}
}
}
```




Palabras reservadas

Palabras que no se pueden usar para nombres de **variables** ni **constantes** y son para otro uso.

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

¿Qué es un operador?



- *Los operadores son símbolos que indican cómo se deben manipular los operandos.*
- *Los operadores junto con los operandos forman una expresión*

“un operador es un elemento de programa que se aplica a uno o varios operandos en una expresión o instrucción”

¿Qué es un operador?

- *Los operadores son símbolos que indican cómo se deben manipular los operandos.*
- *Los operadores junto con los operandos forman una expresión*

“un operador es un elemento de programa que se aplica a uno o varios operandos en una expresión o instrucción”

Por ejemplo:

`n1 = 11 - 2 * 4;`

¿Cuántos operadores
vemos acá?



¿Qué es un operador?

- Los operadores son símbolos que indican cómo se deben manipular los operandos.
- Los operadores junto con los operandos forman una expresión

“un operador es un elemento de programa que se aplica a uno o varios operandos en una expresión o instrucción”

Por ejemplo:

n1=11-2*4;

¿Cuántos operadores
vemos acá?

3 Operadores



Operadores Aritméticos



Operador	Descripción	Ejemplo de expresión	Resultado
-	Cambio de signo (unario)	4	-4
+	Suma	2,5 + 7,1	9,6
-	Resta	235,6 - 103,5	132,1
*	Producto	1,2 * 1,1	1,32
/	División (entera o real)	0,05 / 0,2 7 / 2	0,25 3
%	Resto	20 % 7	6

Separadores



- Los separadores son fundamentales para cumplir con la sintaxis del lenguaje de programación y definir prioridades en la ejecución de operadores.

Separador	Descripción
()	Permiten modificar la prioridad de operadores en una expresión.
{ }	Permiten definir bloques de código: del programa, de sentencias como try...catch, etc.
;	Permite separar sentencias.
,	Permite separar identificadores en una declaración de una sola línea.

Operador de Asignación

```
public class Operadores {  
    public static void main(String args[]) {  
        int i, j;  
        i = 7;   
        j = 3;  
        System.out.println(" Operador suma: i + j= " + (i + j));  
        System.out.println(" Operador resto: i % j= " + (i % j));  
    } // fin del método main  
} // fin de la clase
```

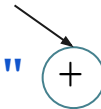
Le solemos decir "***i se vuelve 7***" ya que
no lo estamos comparando



Operador de Concatenación

```
public class Operadores {  
    public static void main(String args[]) {  
        int i, j;  
        i = 7;  
        j = 3;  
        System.out.println(" Operador suma: i + j= " + (i + j));  
        System.out.println(" Operador resto: i % j= " + (i % j));  
    } // fin del método main  
} // fin de la clase
```

Coloca las cadenas de caracteres
una al lado de la otra

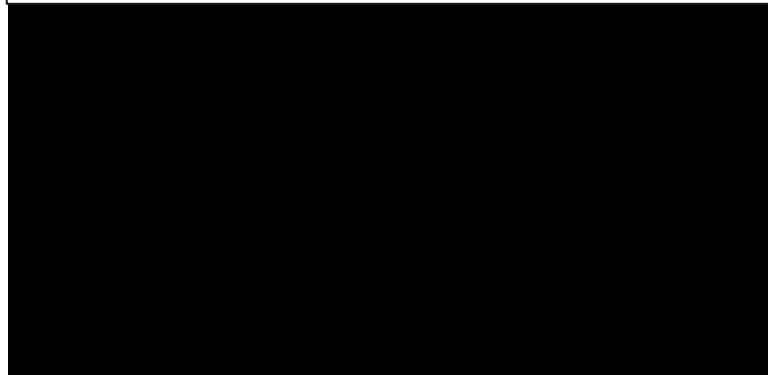


Veamos qué sucede paso a paso

Paso 1: Declaramos las variables *i* y *j* de tipo entero

```
public class Operadores {  
    public static void main(String args[]) {  
        → int i, j;  
        i = 7;  
        j = 3;  
        System.out.println(" Operador suma: i + j= " + (i + j));  
        System.out.println(" Operador resto: i % j= " + (i % j));  
    } // fin del método main  
} // fin de la clase
```

Consola

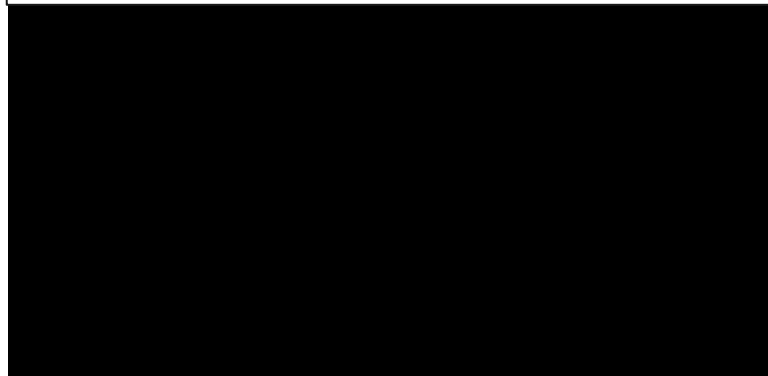


Veamos qué sucede paso a paso

Paso 2: la variable `i` ahora pasa a tener valor 7

```
public class Operadores {  
    public static void main(String args[]) {  
        int i, j;  
        → i = 7;  
        j = 3;  
        System.out.println(" Operador suma: i + j= " + (i + j));  
        System.out.println(" Operador resto: i % j= " + (i % j));  
    } // fin del método main  
} // fin de la clase
```

Consola

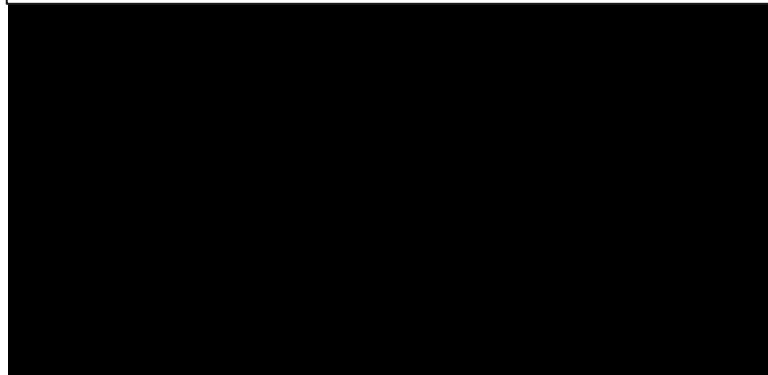


Veamos qué sucede paso a paso

Paso 3: la variable **j** ahora pasa a tener valor 3

```
public class Operadores {  
    public static void main(String args[]) {  
        int i, j;  
        i = 7;  
        → j = 3;  
        System.out.println(" Operador suma: i + j= " + (i + j));  
        System.out.println(" Operador resto: i % j= " + (i % j));  
    } // fin del método main  
} // fin de la clase
```

Consola



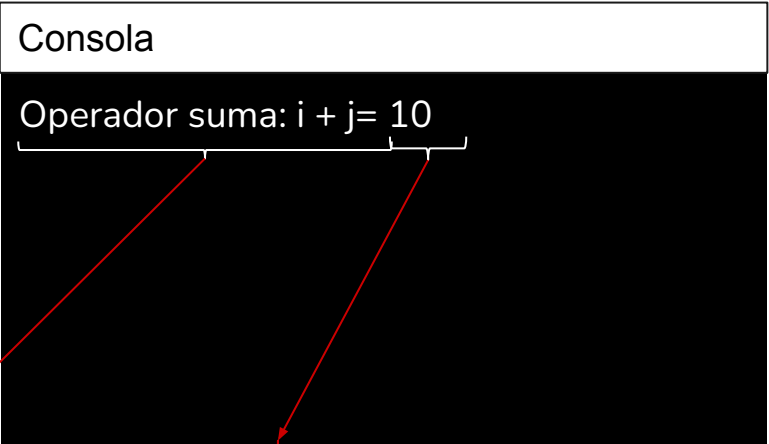
Veamos qué sucede paso a paso

Paso 4: muestra por pantalla lo que está entre “ ” concatenado con el resultado de $i + j$

```
public class Operadores {  
    public static void main(String args[]) {  
        int i, j;  
        i = 7;  
        j = 3;  
        → System.out.println(" Operador suma: i + j= " + (i + j));  
        System.out.println(" Operador resto: i % j= " + (i % j));  
    } // fin del método main  
} // fin de la clase
```

Consola

Operador suma: i + j= 10



Veamos qué sucede paso a paso

Paso 5: muestra por pantalla lo que está entre “ ” concatenado con el resultado de `i % j`

```
public class Operadores {  
    public static void main(String args[]) {  
        int i, j;  
        i = 7;  
        j = 3;  
        System.out.println(" Operador suma: i + j= " + (i + j));  
        → System.out.println(" Operador resto: i % j= " + (i % j));  
    } // fin del método main  
} // fin de la clase
```

Consola

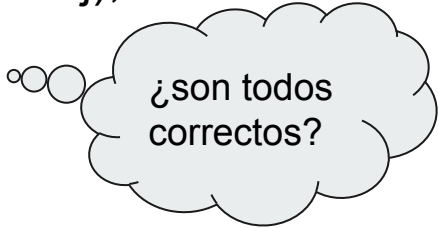
```
Operador suma: i + j= 10  
Operador resto: i % j= 1
```

Para probar...

```
public class Operadores {  
    public static void main(String args[]) {  
        int i, j;  
        i = 7;  
        j = 3;  
        System.out.println(" Operador suma: i + j= " + i + j);  
        System.out.println(" Operador resto: i % j= " + i % j);  
        System.out.println(" Operador multiplicación: i * j= " + i * j);  
        System.out.println(" Operador división: i / j= " + i / j);  
        System.out.println(" Operador resta: i - j= " + i - j);  
    } // fin del método main  
} // fin de la clase
```

Consola

```
Operador suma: i + j= ?  
Operador resto: i % j= ?  
Operador multiplicación: i * j= ?  
Operador división: i / j= ?  
Operador resta: i - j= ?
```



¿son todos correctos?

Operadores aritméticos incrementales

Operador	Descripción	Ejemplo de expresión	Resultado del ejemplo
++	<ul style="list-style-type: none">• i++. Primero se usa el valor de la variable y luego se incrementa.• ++i. Primero se incrementa el valor y después se utiliza.	a a = 5;b = a++; a = 5;b = ++a;	a vale 6 b vale 5 a vale 6 b vale 6
--	Decremento. Funciona de manera análoga al incremento.	a a = 5;b = a--; a = 5;b = --a;	a vale 4 b vale 5 a vale 4 b vale 4

Operadores aritméticos incrementales (ejemplo)

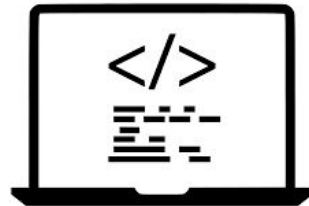
```
public class Operadores {  
    public static void main(String args[]) {  
        int a = 6;  
        System.out.println(a++); // 6  
        System.out.println(a); // 7  
        int b = 6;  
        System.out.println(b--); // 6  
        System.out.println(b); // 5  
    } // fin del método main  
} // fin de la clase
```

```
public class Operadores {  
    public static void main(String args[]) {  
        int a = 6;  
        System.out.println(++a); // 7  
        System.out.println(a); // 7  
        int b = 6;  
        System.out.println(--b); // 5  
        System.out.println(b); // 5  
    } // fin del método main  
} // fin de la clase
```


Operadores aritméticos incrementales (ejemplo)

```
public class Operadores {  
    public static void main(String args[]) {  
        int i = 7;  
        int j = i++;  
        System.out.println("i vale: " + i);  
        System.out.println("j vale: " + j);  
    } // fin del método main  
} // fin de la clase
```

Consola



Operadores aritméticos incrementales (ejemplo)

```
public class Operadores {  
    public static void main(String args[]) {  
        int i = 7;  
        int j = i++;  
        → System.out.println("i vale: " + i);  
        System.out.println("j vale: " + j);  
    } // fin del método main  
} // fin de la clase
```

Consola

i vale 8

Operadores aritméticos incrementales (ejemplo)

```
public class Operadores {  
    public static void main(String args[]) {  
        int i = 7;  
        int j = i++;  
        System.out.println("i vale: " + i);  
        → System.out.println("j vale: " + j);  
    } // fin del método main  
} // fin de la clase
```

Consola

```
i vale 8  
j vale 7
```

Operadores aritméticos incrementales (ejemplo)

```
public class Operadores {  
    public static void main(String args[]) {  
        int i = 7;  
        int j = ++i;  
        System.out.println("i vale: " + i);  
        System.out.println("j vale: " + j);  
    } // fin del método main  
} // fin de la clase
```

Consola

Operadores aritméticos incrementales (ejemplo)

```
public class Operadores {  
    public static void main(String args[]) {  
        int i = 7;  
        int j = ++i;  
        → System.out.println("i vale: " + i);  
        System.out.println("j vale: " + j);  
    } // fin del método main  
} // fin de la clase
```

Consola

i vale 8

Operadores aritméticos incrementales (ejemplo)

```
public class Operadores {  
    public static void main(String args[]) {  
        int i = 7;  
        int j = ++i;  
        System.out.println("i vale: " + i);  
        → System.out.println("j vale: " + j);  
    } // fin del método main  
} // fin de la clase
```

Consola

```
i vale 8  
j vale 8
```

Operadores aritméticos combinados

Operador	Descripción	Ejemplo de expresión	Resultado
<code>+=</code>	Suma	<code>a += b</code>	<code>a = a + b</code>
<code>-=</code>	Resta	<code>a -= b</code>	<code>a = a - b</code>
<code>*=</code>	Multiplicación	<code>a *= b</code>	<code>a = a * b</code>
<code>/=</code>	División	<code>a /= b</code>	<code>a = a / b</code>
<code>%=</code>	Resto	<code>a %= b</code>	<code>a = a % b</code>

Ejemplo:

```
int usuario=25; // usuario vale 25
```

```
usuario+=25; // es equivalente a usuario=usuario+25, usuario pasa a valer 50
```

Operadores aritméticos combinados (ejemplo)

```
public class Operadores {  
    public static void main(String args[]) {  
        int i = 7;  
        i+=3;  
        System.out.println("i vale: " + i);  
    } // fin del método main  
} // fin de la clase
```

Consola

Operadores aritméticos combinados (ejemplo)

```
public class Operadores {  
    public static void main(String args[]) {  
        int i = 7;  
        i+=3; // i=i+3  
        → System.out.println("i vale: " + i);  
    } // fin del método main  
} // fin de la clase
```

Consola

i vale 10

Operadores de relación

*Comparan dos expresiones y siempre dan como resultado un valor lógico (**true** o **false**)*

Operador	Descripción	Ejemplo de expresión	Resultado
==	Igual a	7 == 38	false
!=	Distinto de	'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

Operadores de relación (ejemplo)

```
public class Operadores {  
    public static void main(String args[]) {  
        int i = 7;  
        int j = 3;  
        char c1='g';  
        char c2='a';  
        System.out.println("¿i es distinto a j? " + (i != j));  
        System.out.println("¿c1 es menor que c2? " + (c1<c2));  
    } // fin del método main  
} // fin de la clase
```

Consola

Operadores de relación (ejemplo)

```
public class Operadores {  
    public static void main(String args[]) {  
        int i = 7;  
        int j = 3;  
        char c1='g';  
        char c2='a';  
        System.out.println("¿i es distinto a j? " + (i != j));  
        System.out.println("¿c1 es menor que c2? " + (c1<c2));  
    } // fin del método main  
} // fin de la clase
```

Consola

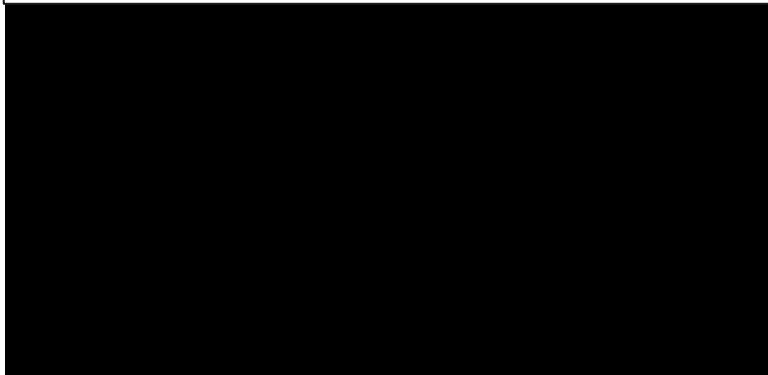
```
¿i es distinto a j? true  
¿c1 es menor que c2? false
```

Operadores de relación (ejemplo)



```
public class Operadores {  
    public static void main(String args[]) {  
        int i = 7;  
        int j = 3;  
        System.out.println("¿i es menor o igual a j? " + (i<=j));  
    } // fin del método main  
} // fin de la clase
```

Consola



Operadores de relación (ejemplo)



```
public class Operadores {  
    public static void main(String args[]) {  
        int i = 7;  
        int j = 3;  
        → System.out.println("¿i es menor o igual a j? " + (i<=j));  
    } // fin del método main  
} // fin de la clase
```

Consola

¿i es menor o igual a j? false

Operadores lógicos

Se utilizan entre operadores de relación y siempre dan un resultado **true** o **false**

Operador	Descripción	Ejemplo de expresión	Resultado
!	Negación – <u>Not</u>	!false !(5 == 5)	true false
	Suma lógica – <u>Or</u>	true false (5 == 5) (5 < 4)	true true
&&	Multiplicación lógica - And	false && true (5 == 5) && (5 < 4)	false false

Operadores lógicos (ejemplo)



```
public class Operadores {  
    public static void main(String args[]) {  
        int i = 7;  
        int j = 3;  
        int k = 3;  
        System.out.println("¿j es igual a i, o es igual a k? " + ((j==i) || (j==k)));  
    } // fin del método main  
} // fin de la clase
```

Consola

Operadores lógicos (ejemplo)



```
public class Operadores {  
    public static void main(String args[]) {  
        int i = 7;  
        int j = 3;  
        int k = 3;  
        → System.out.println("¿j es igual a i, o es igual a k? " + ((j==i) || (j==k)));  
    } // fin del método main  
} // fin de la clase
```

Consola

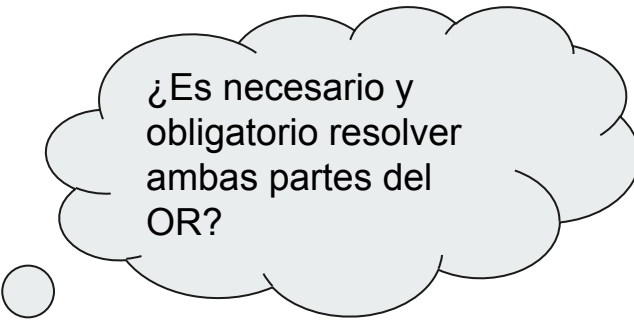
¿j es igual a i, o es igual a k? true

Ejemplo

```
public class Clase_1_Ejemplo_7{
    public static void main(String[] args) {
        int i, j, k;
        boolean resultado;
        i = 7;
        j = 3;
        k = 2;
        //(i==j)||(i==k) primero se resuelven los paréntesis (i==j) y (i==k), y luego ||
        resultado = (i==j)||(i==k);
        System.out.println("Operación: (i==j)||(i==k) " + resultado);

        //((i!=j)||(i==k)) primero se resuelven los paréntesis (i!=j) y (i==k), y luego ||
        System.out.println("Operación: ((i!=j)||(i==k)) " + ((i!=j)||(i==k)));

        //((!(i!=j))||(i==k)) primero se resuelven los paréntesis (!(i!=j))
        //(i==k), y luego ||. Para resolver (!(i!=j)) primero se resuelve
        //(i!=j) y luego se le aplica ! (negación)
        System.out.println("Operación: (!(i!=j))||(i==k)) " + (!(i!=j))||(i==k)));
    }
}
```



¿Es necesario y obligatorio resolver ambas partes del OR?

Algunos tips



- Aplicar buenas prácticas en nuestros programas:
 - Poner nombres representativos a los programas, variables y métodos (se verá más adelante)
 - Indentar el código para que sea más legible (en JAVA no es obligatoria como en Python)
 - Inicializar las variables
 - Poner comentarios breves en el código
- Recordar que el nombre de la clase tiene que ser el mismo nombre del archivo .java que se está definiendo (incluso respetar mayúsculas y minúsculas)
 - Ejemplo: `public class ProgramaNuevo {...}` ⇒ el archivo se llamará **ProgramaNuevo.java**
- Dado que Java es *case sensitive* prestar atención cuando declaran y luego usan
- Declarar las variables antes de su uso y en una sección que me sea fácil ubicarlas
- Usar tipos acordes a lo que se va a almacenar sin desperdiciar lugar. Cuidado con asignar a *int* un *char* porque no genera error sintáctico sino semántico.
- Recordar que la división de dos enteros retorna también un entero.
- Siempre que voy a usar valores que no cambian, declarar constantes con su valor