

# Algoritmos y Estructuras de Datos

## Introducción a Java

Curso 3

Andrés Juárez

## Clase 1: Introducción

### Java: características

Java es un lenguaje de programación como Python, C, C++, C#, TypeScript, etcétera. Java es uno de los lenguajes más utilizados, encontrándose entre los lenguajes más demandados desde hace varios años.

Algunas características son:

- **Lenguaje de alto nivel.** Esto significa que el lenguaje está más cerca de la comprensión de los seres humanos que de una máquina. Un lenguaje de más bajo nivel es, por ejemplo, C. Y uno de aún más alto nivel sería Python.
- **Lenguaje versátil, multipropósito.** Se utiliza para una variedad de aplicaciones, como desarrollos web, aplicaciones para telefonía móvil, aplicaciones de escritorio, etc.
- **Lenguaje portable.** Esto significa que los programas escritos en Java pueden ejecutarse en diferentes máquinas sin necesidad de hacer ninguna adaptación o con ajustes mínimos.
- **Lenguaje Orientado a Objetos.** La Programación Orientada a Objetos (POO) es un paradigma de programación que veremos en el curso. Parte de la suposición que los problemas a solucionar están conformados por **objetos que interactúan entre sí mediante mensajes**. Por ejemplo, cuando tomo un control remoto (objeto) y presiono el botón de encendido, el objeto control remoto le envía un mensaje al televisor (otro objeto) para que se encienda. Luego puedo enviar otro mensaje a través del control indicando que suba el volumen o lo baje, etc.
- **Lenguaje robusto.** Por robustez nos referimos a la capacidad de resolver fallos o situaciones anómalas. Java exige el manejo de excepciones para porciones de código crítico que pueden desencadenar en un fallo en la ejecución. Por ejemplo, querer recuperar información de un archivo que no existe porque fue borrado o se le cambió el nombre.
- Hay otras características, como **multihilos** y **programación funcional**, entre otras, que no veremos en este curso dado que el objetivo no es aprender un lenguaje de programación, sino ver estructuras de datos y algoritmos para trabajar con dichas estructuras.

### JDK

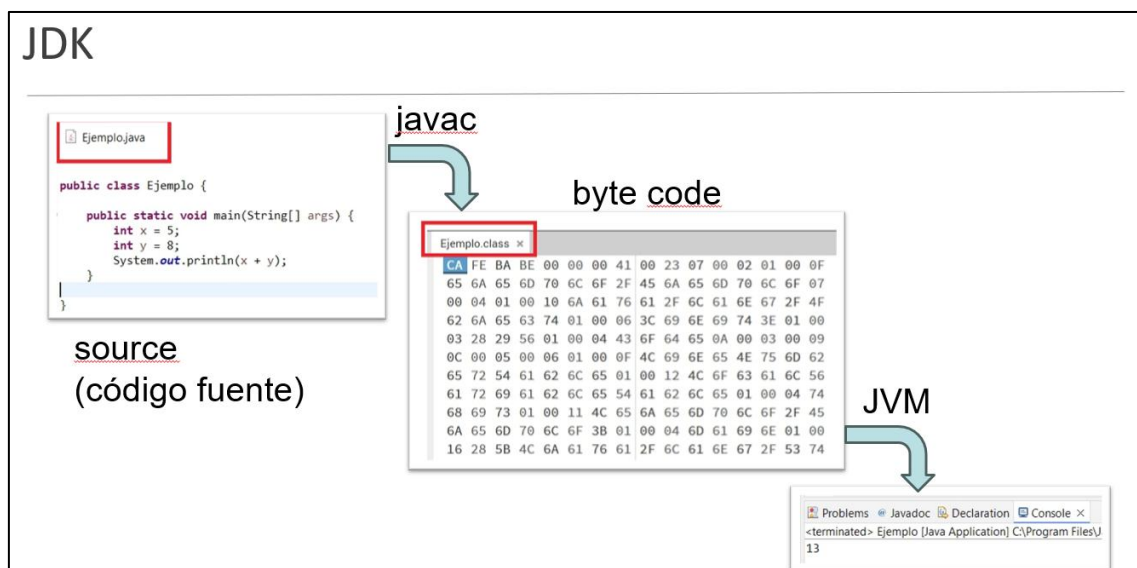
Java no solo es un lenguaje de programación, sino que es una **plataforma para el desarrollo de aplicaciones**. Esta plataforma, que se llama JDK (Java Development Kit o Herramientas de Desarrollo de Java), contiene:

- El **compilador** que se encarga de traducir las instrucciones en Java a lo que se llama **bytecode**, un lenguaje escrito en código binario: unos y ceros.

- La **Java Virtual Machine (JVM)** que se encarga de **interpretar** el archivo *bytecode* que genera el compilador para que la máquina lo pueda ejecutar.
- La **API (Application Programming Interface** o Interfaz de Aplicaciones de Programación). Es un conjunto de **bibliotecas** que contiene código que podemos utilizar facilitando el desarrollo de nuestras aplicaciones. Por ejemplo, bibliotecas para cálculos matemáticos o para manejo de archivos o estructuras de datos, etc.

El diagrama de cómo interactúa el compilador y la JVM lo vemos en la figura que está debajo:

- En primer lugar, se escribe el **código fuente** (source) en un editor de texto. Este código se guarda en archivos de extensión **.java**.
- Luego, se **compila** con la instrucción **javac**, esto genera, para cada archivo fuente, un archivo **.class**: son los archivos binarios: **bytecodes**.
- Por último, la **JVM** interpreta dicho código y se **ejecuta**.



## IDEs

IDE, una sigla más para aprender: Integrated Development Environment (Entorno de Desarrollo Integrado). Son entornos que nos permitirán desarrollar programas complejos porque nos brindan ayuda con algunas herramientas, por ejemplo:

- Marcan las palabras reservadas con cierto color.
- Nos marcan los errores sintácticos y nos sugieren algunas maneras de solucionarlos.
- Cuentan con un debug (depurador) que nos permitirá seguir el código, línea por línea, observando los valores de las variables. Esta herramienta es muy útil a la hora de encontrar errores en el código.

### ¿Qué IDEs podemos utilizar?

La respuesta es: cualquiera con el que te sientas cómodo. Listaremos algunos de los que hay disponibles. Nosotros utilizaremos Eclipse, pero eso no impide que puedas utilizar algún otro.

- Eclipse
- NetBeans
- IntelliJ IDEA
- BlueJ
- MyEclipse
- Oracle JDeveloper

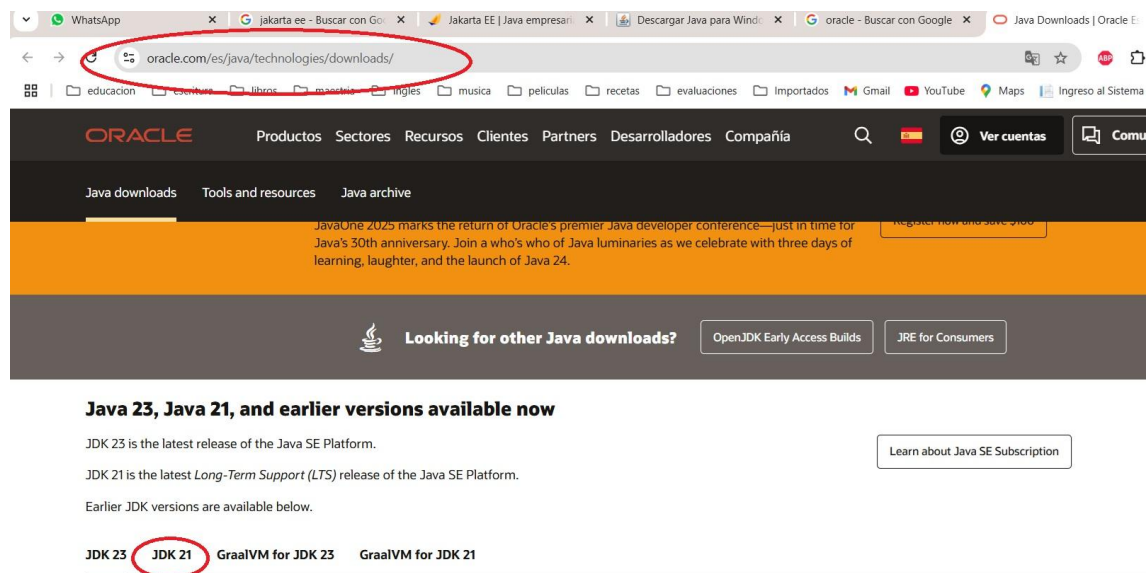
### Instalación del JDK

Si instalan algún IDE, por ejemplo, Eclipse, además instala el JDK, por lo que este paso no sería necesario. Sin embargo, vamos a hacerlo para verificar que no es necesario tener instalado un IDE para poder hacer una aplicación en Java, pero sí es muy recomendable.

Deben dirigirse a este sitio web:

<https://www.oracle.com/es/java/technologies/downloads/>

y elegir JDK 21.



Una vez en ese sitio, eligen la pestaña según el Sistema Operativo que tengan.

Windows:

Eligen la pestaña Windows y bajan el archivo que dice “x64 Installer”.

Java 19 Java 17

**Java SE Development Kit 19.0.2 downloads**

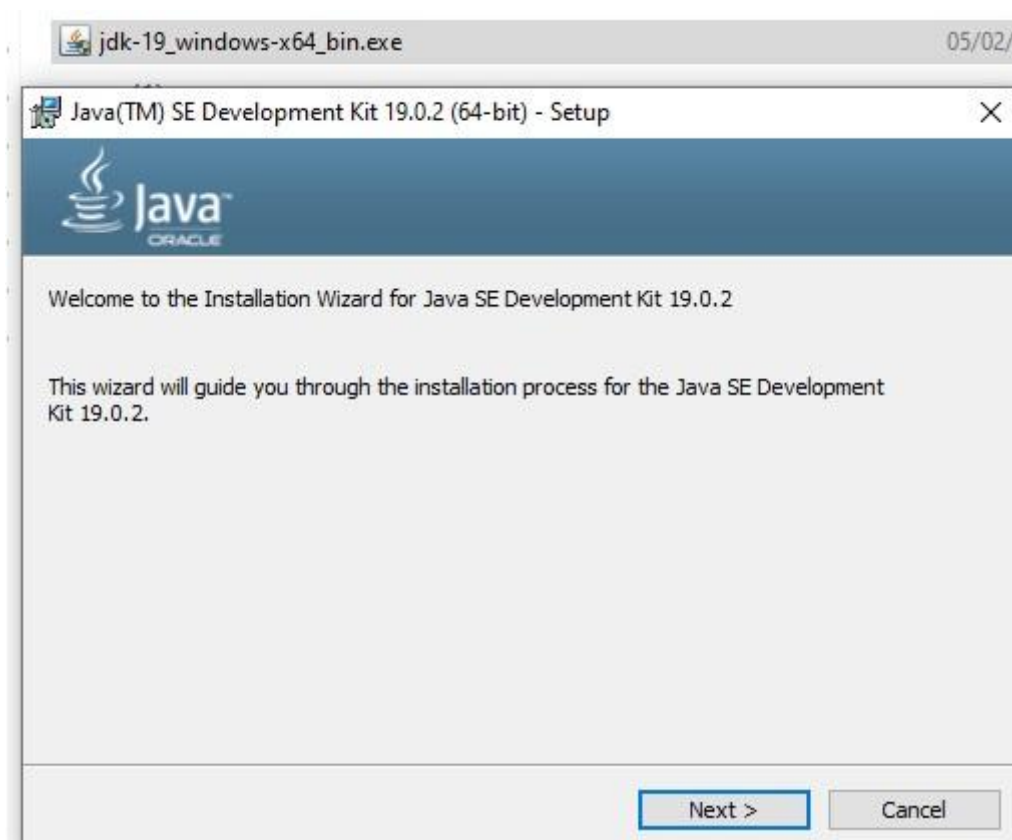
Thank you for downloading this release of the Java™ Platform, Standard Edition Development Kit (JDK™). The JDK is a development environment for building applications and components using the Java programming language.

The JDK includes tools for developing and testing programs written in the Java programming language and running on the Java platform.

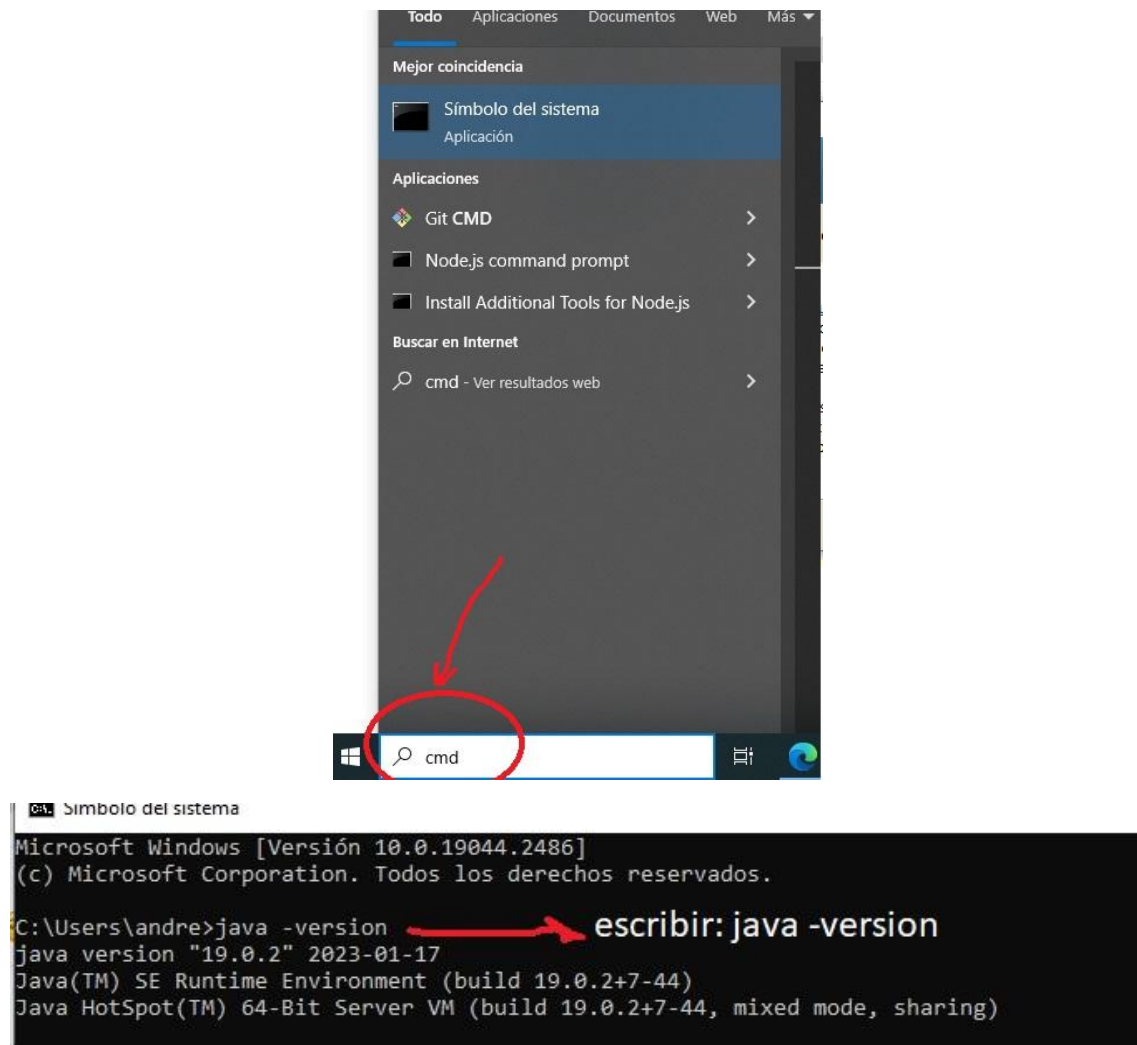
Linux macOS **Windows**

| Product/file description | File size | Download  |
|--------------------------|-----------|---|
| x64 Compressed Archive   | 179.13 MB | <a href="https://download.oracle.com/java/19/latest/jdk-19_windows-x64_bin.zip">https://download.oracle.com/java/19/latest/jdk-19_windows-x64_bin.zip</a> ( sha256) |
| x64 Installer            | 158.91 MB | <a href="https://download.oracle.com/java/19/latest/jdk-19_windows-x64_bin.exe">https://download.oracle.com/java/19/latest/jdk-19_windows-x64_bin.exe</a> ( sha256) |
| x64 MSI Installer        | 157.76 MB | <a href="https://download.oracle.com/java/19/latest/jdk-19_windows-x64_bin.msi">https://download.oracle.com/java/19/latest/jdk-19_windows-x64_bin.msi</a> ( sha256) |

Una vez que termina de bajar, le dan doble clic y aceptan todo. Les preguntará si desea instalarlo y hacer cambios en su máquina, indican que sí.



Al finalizar ya estará instalado. ¿Cómo comprobarlo? Abres una ventana de escritorio colocando “cmd” en la ventana del buscador y escribes: java -version.



Al presionar enter verás la versión que está instalada.

Linux:

El procedimiento es similar.

También se puede instalar desde una terminal. Abres una terminal presionando Ctrl + Alt + T al mismo tiempo, y ejecutas:

```
sudo apt-get update
```

```
sudo apt-get upgrade
```

```
sudo apt-get install default-jre
```

En la primera instrucción te pedirá la contraseña y en todas, te preguntará si deseas continuar.

### Primer programa: ¡Hola mundo!

Vamos a probar nuestro primer programa en Java. En todos los lenguajes es una tradición que el primer programa sea una aplicación que nos muestra en pantalla un cartel que dice “¡Hola mundo!”. No vamos a romper esa tradición y haremos eso.

Abres algún directorio de tu máquina, recomendamos crear uno específico para este curso, y escribes en un archivo de texto las siguientes líneas, respetando las mayúsculas y minúsculas:

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

Este archivo debe llamarse “Saludo.java”. Hay que habilitar la configuración en Windows para poder ver las extensiones de los archivos y, luego, poder modificarlas.

Luego, en la ventana de consola, que abriste con cmd, te diriges al directorio en donde está ese archivo, con la instrucción

```
cd carpeta/subcarpeta/
```

Una vez ahí, escribes:

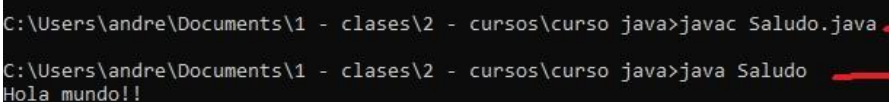
```
javac Saludo.java
```

Con esa instrucción compilamos el archivo. Si está todo bien, en unos pocos segundos, debería aparecer nuevamente el *prompt* a la espera de un nuevo comando sin indicar nada. De lo contrario, marcará algún error.

Finalmente, escribes

```
java Saludo
```

y deberás observar el cartel que dice “Hola mundo!”.



The screenshot shows a Windows command prompt with the following text:

```
C:\Users\andre\Documents\1 - clases\2 - cursos\curso java>javac Saludo.java  
C:\Users\andre\Documents\1 - clases\2 - cursos\curso java>java Saludo  
Hola mundo!!
```

Red arrows point from the text "compilación" to the first command line, and from the text "ejecución" to the second command line.

### Analicemos el código

Todo el código que escribamos debe pertenecer a alguna clase. Y, cada clase, está en un archivo que debe tener el mismo nombre que la clase, pero con extensión *java*. Más adelante veremos esto con mayor profundidad. Sin embargo, podemos adelantar que una clase contendrá:

- **atributos** (variables)
- **métodos** (funciones)

En el caso de nuestro ejemplo, la clase se llama *Saludo* y es una clase pública, para que se pueda acceder. Por convención la clase debe comenzar con un carácter en mayúscula. Lo único que contiene nuestra clase es un método (función) que se llama *main*.

En cualquier aplicación que desarrollemos debemos tener una clase con un método *main* que será el método desde donde comenzará a ejecutarse nuestra aplicación. El método *main* debe ser *public*, *static* y *void*, más adelante veremos qué significa.

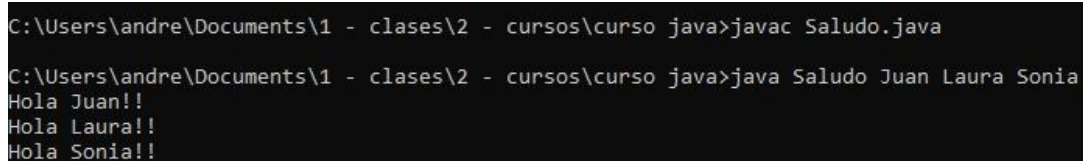
Nuestro método, contiene una sola sentencia que es la impresión de “¡Hola mundo!”. Para lograr imprimir algo en pantalla debemos llamar a un método de otra clase: el método *println* del campo *out* de la clase *System*.

Observando el método *main*, vemos que recibe un parámetro que es un vector de *String* (*String [ ]*) llamado *args* por argumentos. Estos argumentos son los que recibe nuestra aplicación cuando se llama en modo consola.

Si modificamos un poco nuestro código de ejemplo de la siguiente manera:

```
public class Saludo {  
    public static void main(String[] args) {  
        for (int i = 0; i < args.length; i++)  
            System.out.println("Hola " + args[i] + "!!!");  
    }  
}
```

Lo compilamos de la misma forma que el anterior y lo ejecutamos, pero esta vez colocando algunos nombres, veremos la salida que se muestra en la siguiente imagen.



```
C:\Users\andre\Documents\1 - clases\2 - cursos\curso java>javac Saludo.java  
C:\Users\andre\Documents\1 - clases\2 - cursos\curso java>java Saludo Juan Laura Sonia  
Hola Juan!!  
Hola Laura!!  
Hola Sonia!!
```

¿Qué es lo que hace este código?

Cada uno de los argumentos pasados ocupará una posición en nuestro vector de argumentos *args*.

Por ejemplo, *Juan* estará en la posición cero, por lo tanto, será *args [0]*.

Laura, en la posición 1 y así, sucesivamente. El objeto *args* tiene varios métodos y atributos que son de la clase *String*. Uno de estos atributos guarda la longitud del vector (o cantidad de elementos que contiene).

Por lo tanto *args.length* en nuestro ejemplo valdrá 3, ya que se pasaron tres argumentos. Hay que tener cuidado porque el tamaño del vector es 3 pero las posiciones que se ocupan son la 0, 1 y 2. Por este motivo, el ciclo empieza en 0 y finaliza cuando es



menor que la cantidad de argumentos (y no menor o igual). Dentro del ciclo se imprimen cada uno de los argumentos.

Si no se comprende bien cómo funciona esto no hay ningún problema, enseguida comenzaremos desde cero utilizando algún IDE.

## Eclipse

### Instalación de Eclipse

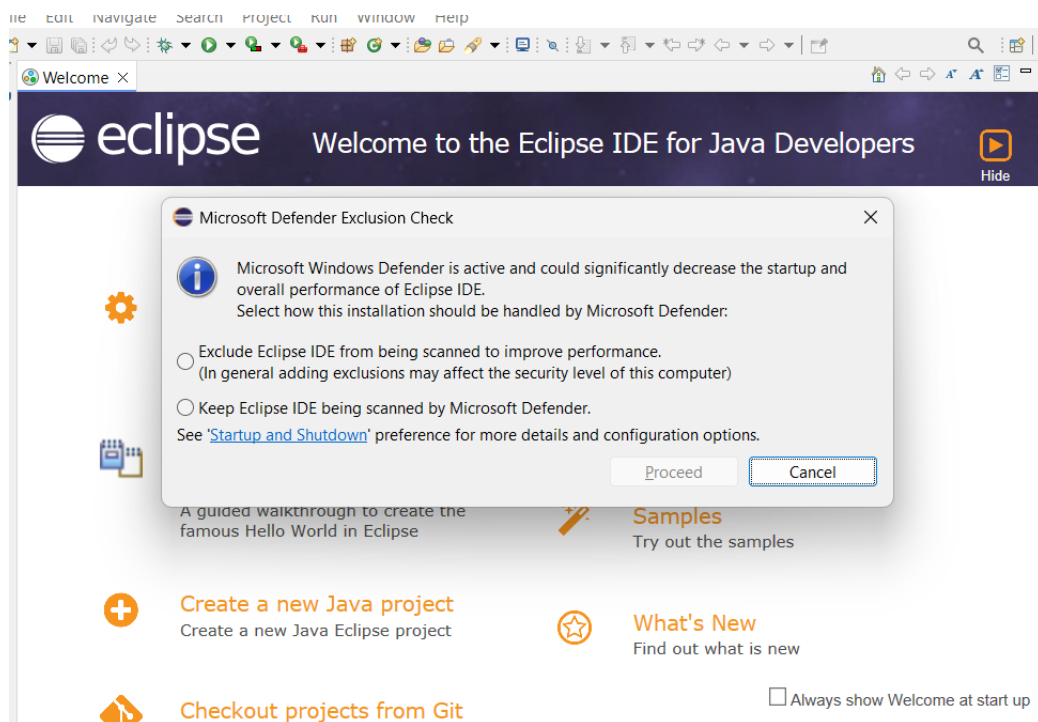
Para instalar el IDE Eclipse, hay que dirigirse al sitio oficial: <https://eclipseide.org/>

Y descargar el IDE en el link que se ve en la figura.

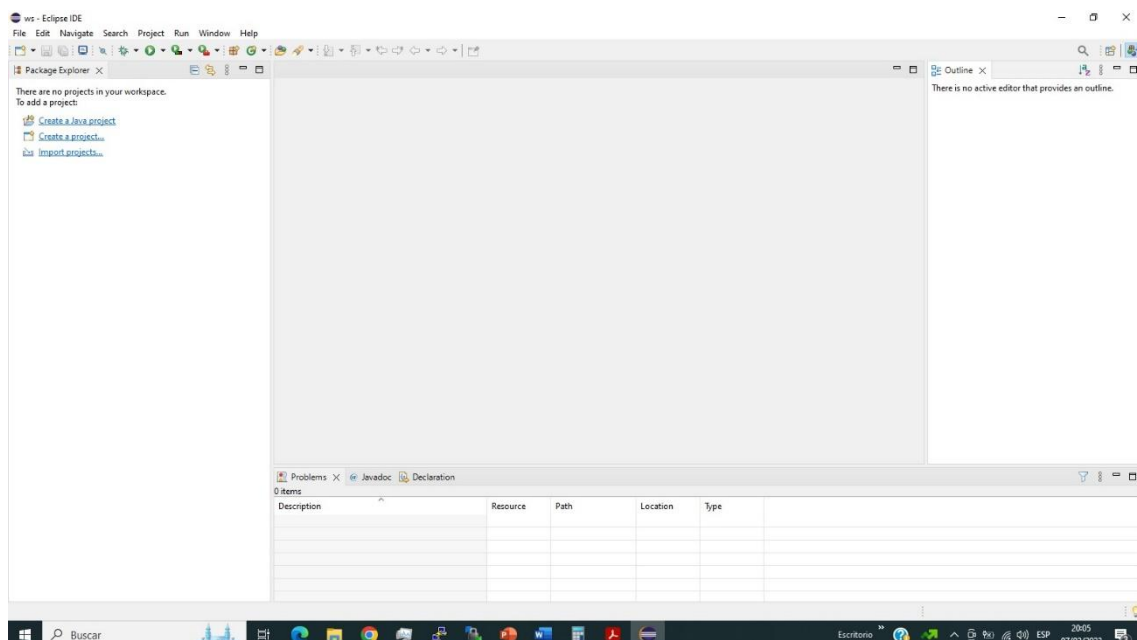


Luego, descomprimen el archivo que bajaron y, dentro de la carpeta eclipse, tendrán el logo de eclipse en azul, que dirá *eclipse.exe*, doble clic en ese archivo y se les abrirá el editor.

Les preguntará en qué carpeta quieren guardar sus proyectos y también saldrá un cartel consultando si quieren excluirlo del escaneo de Microsoft Windows Defender (si están en Windows), les recomiendo decir que lo excluya del escaneo y listo.

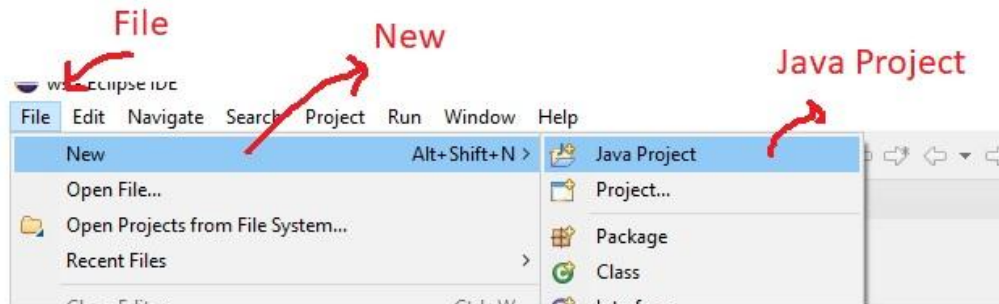


Verán una ventana que les ofrece un recorrido por la aplicación, pueden cerrarla y marcar el cuadro de “no volver a mostrar”. Finalmente, nos encontraremos con la siguiente pantalla:

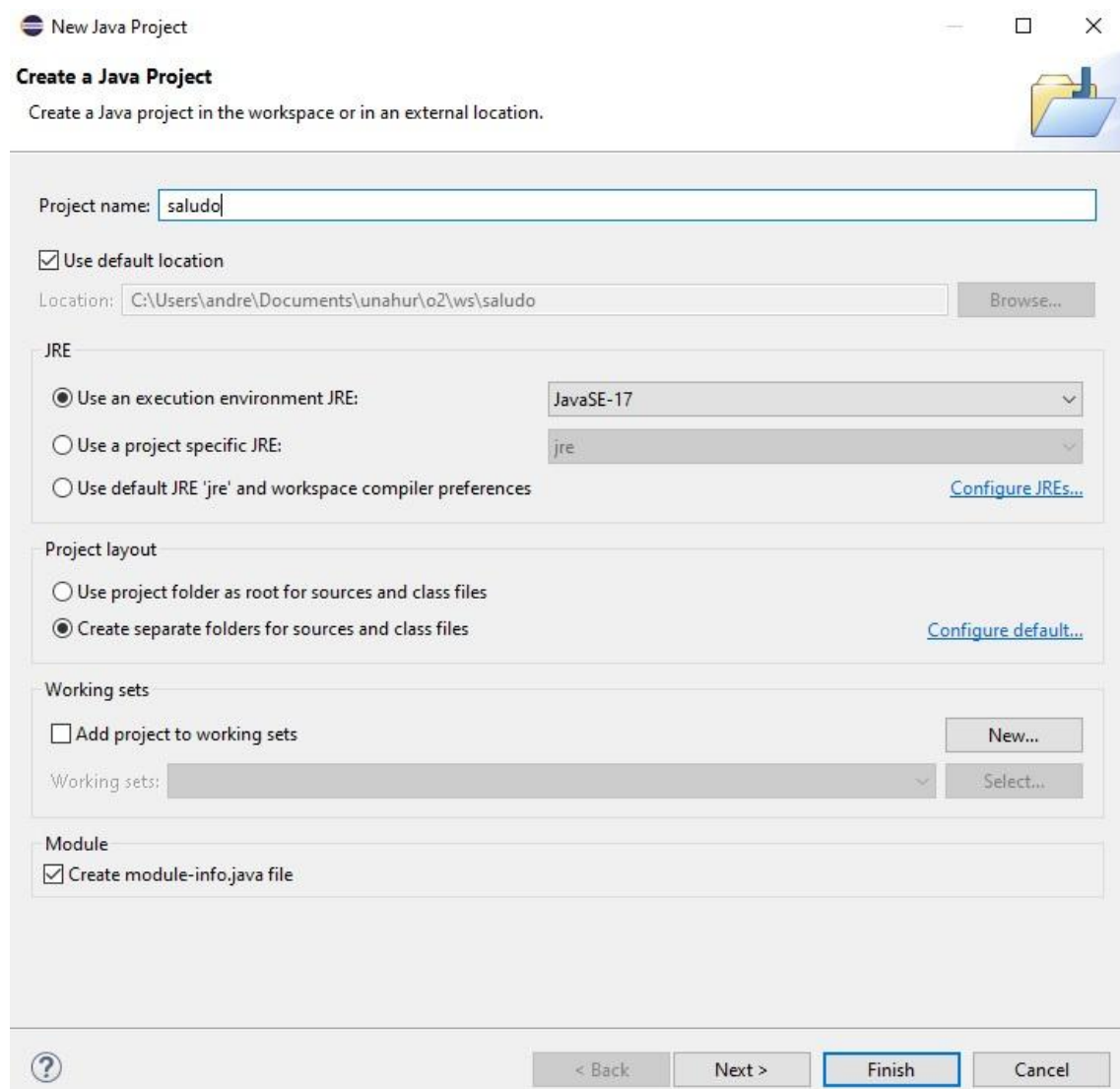


### Primer programa en Eclipse

Para hacer una aplicación debemos crear un nuevo proyecto: presionamos File, New, Java Project.



Cada proyecto debe tener un nombre, colocamos “saludo” o “ejemplo\_1” o el nombre que te parezca mejor, pero que luego te sirva para reconocerlo.

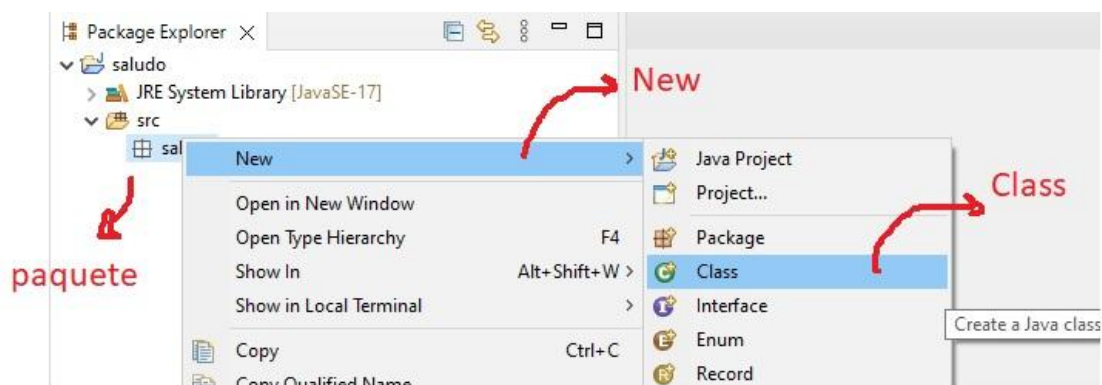


Una vez creado nuestro nuevo proyecto, en la parte superior izquierda, en la ventana src, hacemos click derecho, New, Package (creamos un nuevo paquete).



El paquete también debe tener un nombre, por el momento no es importante, podemos ponerle el mismo nombre del proyecto u otro.

Luego, presionamos clic derecho nuevamente, pero esta vez en el ícono del paquete, le indicamos New, Class (nueva clase).



El nombre de la clase debe ir con mayúsculas en su primera letra. Colocamos “Saludo” y marcamos la casilla “public static void main(String[] args)”.

Finalmente, nos aparece el archivo para escribir nuestro código. Escribimos el mismo código que en el ejemplo de la aplicación por consola.

Y llegó el momento de ejecutar nuestro archivo. Del lado derecho de la flecha verde de “play”, hay otra flecha, la presionamos y le indicamos Run As (correr como) Java Application.

**New Java Class**

**Java Class**  
Create a new Java class.

Source folder:

Package:

☐ Enclosing type:

Name:  **Nombre clase**

Modifiers:  
☒ public ☐ package ☐ private ☐ protected  
☐ abstract ☐ final ☐ static  
☒ none ☐ sealed ☐ non-sealed ☐ final

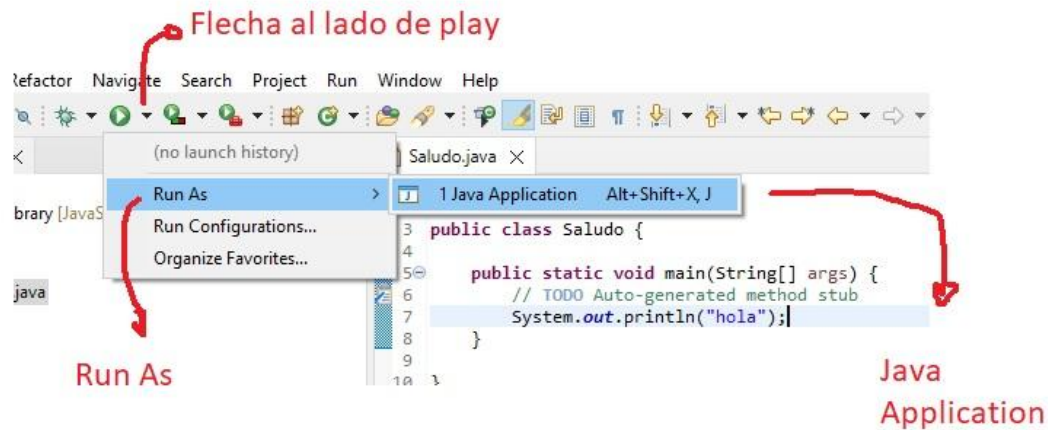
Superclass:

Interfaces:

Which method stubs would you like to create?  
☒ public static void main(String[] args)  
☐ Constructors from superclass  
☒ Inherited abstract methods

Do you want to add comments? (Configure templates and default value [here](#))  
☐ Generate comments **marcar**

```
Saludo.java X
1 package saludo;
2
3 public class Saludo {
4
5     public static void main(String[] args) {
6         // TODO Auto-generated method stub
7     }
8 }
9
10
11
```



En la ventana de consola, en la parte inferior, debemos ver el cartel “hola”.

Para las siguientes ejecuciones alcanza con presionar el botón “play” directamente.

### Ejercicios

- 1) Probar el ejemplo anterior. Confirmar la salida.
- 2) Cambiar el mensaje “hola” por un saludo personalizado, por ejemplo “Hola Juan” u “Hola María”.
- 3) Escribir varios println con varios mensajes.
- 4) Probar qué sucede si alguna instrucción se escribe mal (sin punto y coma o en lugar de “println”, pueden probar con “pritln” o algo similar).

## Tipos de datos, variables y entrada de datos

### Tipos de datos

¿Qué es un tipo de dato? Podríamos decir que es el **atributo que indica la clase de información que guardaremos**.

Veamos un ejemplo, supongamos que nos dan el dato “1”, y me piden guardarlo, entonces lo tengo que guardar con un tipo que describa el comportamiento indicado. ¿Pero cómo sé cuál es el indicado? Eso está determinado por la finalidad de ese dato: qué necesitas hacer o qué comportamiento necesitas que tenga.

**Los tipos de datos básicos o primitivos son los únicos que no necesitan pertenecer a una clase para trabajar con ellos.** Estos son:



- Números
  - `byte`
  - `short`
  - `int`
  - `long`
  - `float`
  - `double`
- Texto
  - `char` (un carácter)
- Otros
  - `boolean`

Ahora vamos a ir uno por uno, explicando un poco el comportamiento que tienen y qué pueden guardar.

- `int`: en este tipo de datos podemos almacenar `números enteros`, y podemos hacer cualquier tipo de operación matemática con ellos (suma, resta, multiplicación, etc.). `El rango de valores va desde  $-2^{31}$  hasta  $2^{31}-1$ .`
- `short`: este tipo de datos tiene las mismas características que un `int` pero ocupa menos espacio, por lo que `el rango de valores que podemos almacenar es más corto: desde -32.768 hasta 32.767.`
- `long`: similar al `int` pero al contrario del `short`, ocupa más espacio, por lo que `podemos almacenar números enteros más largos: desde  $-2^{63}$  hasta  $2^{63}-1$ .`
- `byte`: es un tipo de dato que `puede almacenar números enteros de -128 hasta 127.`
- `float`: este tipo de dato `nos permite almacenar números decimales con una precisión simple` y podemos hacer cualquier tipo de operación matemática con ellos (suma, resta, multiplicación, etc.).
- `double`: este tipo de dato `nos permite almacenar números decimales con una precisión doble y con un rango mayor que los de float`, por lo que necesita ocupar más espacio. También podemos hacer cualquier tipo de operación matemática con ellos (suma, resta, multiplicación, etc.).
- `char`: es un tipo de dato que `representa a un carácter`, como una letra mayúscula o minúscula, un símbolo como \$ o #, un dígito, por ejemplo "3", etc. Cuando almacena un dígito lo hace como un carácter, es decir, no podemos hacer operaciones matemáticas con ellos.
- `boolean`: es un tipo de dato que puede tener dos valores, `verdadero o falso.`

## Variables

Imaginemos que nos vamos a mudar y tenemos que guardar nuestras cosas en cajas. Para luego encontrarlas, a cada caja le ponemos un nombre, por ejemplo "libros", "platos y cubiertos", etc.

Esto es muy parecido a guardar un dato en un programa. Nuestros datos serían los libros, los platos y los cubiertos, y las cajas con las etiquetas, son las variables.

Entonces ¿Qué es una variable? **Es un espacio que tomamos de la memoria de nuestra computadora para almacenar un dato.** Una variable, puede justamente variar, esto significa que el dato que está almacenado en esa “caja” se puede cambiar.

Un ejemplo práctico:

```
int numero = 10; // La variable 'numero' contiene un dato de tipo int.  
  
System.out.println(numero); // -> 10  
  
numero = numero + 1; // 11 = 10 + 1  
  
// Estoy reasignando el valor de  
// la variable 'numero' sumandole 1 al valor anterior.  
  
System.out.println(numero); // -> 11
```

Intenta analizar el código de arriba antes de seguir.

Lo que este código hace es lo siguiente: en primer lugar, crea una variable con tipo de dato `int` y guarda dentro de esa variable un 10. Luego, imprime el valor que está dentro de la variable `numero` con el método de impresión visto anteriormente. Podemos notar algo nuevo: cuando quiero imprimir el valor guardado en una variable, simplemente escribo el nombre de la variable dentro de los paréntesis sin comillas.

En la siguiente instrucción, le indicamos a la variable `numero` que, a partir de esa línea en adelante, su valor será el que tenía hasta el momento, más uno.

Por último, imprime el valor que está dentro de la variable con el mismo método de antes.

Noten, que para **crear y asignar una variable** se escribe lo siguiente:

```
tipo nombre = dato;
```

**Siendo el dato del tipo correspondiente.**

Si quisiéramos **asignar muchas variables, que son del mismo tipo**, podríamos expresarlo de la siguiente forma:

```
tipo nombreUno = datoUno, nombreDos = datoDos;
```

Noten que **las variables están separadas por una coma, y el punto y coma están al final de la línea.** Aunque esto no está recomendado por una cuestión de claridad. Se prefiere definir una variable por cada línea.

**Si a una variable no queremos asignarle un valor en el momento de su declaración, debemos hacerlo así:**



```
tipo nombre;
```

Y después, para asignarle el valor:

```
nombre = dato;
```

Noten que, al asignar más tarde el valor, no debemos indicar de qué tipo es. Esto nos daría un error porque Java interpretaría que quieres crear otra variable con el mismo nombre que una ya existente.

Con respecto a los nombres, vamos a hablar sobre cómo nombrar variables. Primero que nada, los nombres tienen que ser descriptivos ¿Qué quiere decir esto? Supongamos que, en cierta aplicación, necesito guardar algunos datos del usuario, como ser el nombre, la edad y la altura. ¿Cuál sería la manera correcta de nombrar las variables? Mira las opciones, piénsalo, después continúa leyendo.

```
a) String nombre = "Pedro";
```

```
    int edad = 20;
```

```
    float altura = 1.7;
```

```
b) String usuario = "Pedro";
```

```
    int anios = 20;
```

```
    float alt = 1.7;
```

Bien, la opción correcta, es la a). Pero alguno podría argumentar que la variable para el nombre podría llamarse "usuario", o la variable de edad podría llamarse "anios". Pero esto no es correcto. ¿Por qué?

Porque usuario podría representar la cadena de caracteres que una persona utiliza para ingresar a la aplicación y no, necesariamente el nombre. Por ejemplo, el usuario podría ser: "jo2020".

¿Y la edad? ¿Por qué no se puede llamar "anios"? Bueno, porque anios, podría representar muchas cosas, como los años que pasaron desde que el usuario se registró en la aplicación, por ejemplo.

Los nombres de las variables tienen que empezar con una letra en minúscula. Es decir, no debe declararse una variable que empiece con un símbolo o con un número.

Por último, cuando el nombre de una variable tiene más de una palabra, por ejemplo, "nombre del usuario", se escribe así:

nombreDelUusuario

Escribimos todo junto, y en lugar de dejar los espacios, ponemos la primera letra de la siguiente palabra con mayúscula.

## Palabras reservadas

Antes de cerrar con el tema de variables, vamos a ver algunas palabras que no se pueden usar como nombres de variables ni de métodos. Estas son las palabras reservadas de Java y tienen una función específica que veremos más adelante:

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

## Impresiones en pantalla y entradas por teclado

Veamos un poco más sobre impresiones en pantalla y entradas por teclado.

Habíamos visto que una impresión en pantalla podrá hacerse así:

```
System.out.println(";Hola mundo!");
```

O, si necesitáramos imprimir una variable:

```
int numero = 1;
```

```
System.out.println(numero);
```

También vimos que podíamos utilizar *print* y *println*. Donde *println*, hace un salto de línea, y *print*, no.

A todo esto, vamos a sumar la entrada de datos, es decir, que el usuario pueda guardar y utilizar datos que ingrese.

Para esto, vamos a necesitar importar nuestra primera librería: *Scanner*. Por ahora el concepto de librería no importa entenderlo. Escribimos el siguiente código:

```
package prueba;
import java.util.Scanner;

public class Ppal {

    public static void main(String[] args) {
        Scanner entrada = new Scanner(System.in);

        System.out.print("Ingrese su nombre: ");
        String nombre = entrada.nextLine();

        System.out.print("Ingrese su edad: ");
        int edad = entrada.nextInt();

        System.out.println("Hola " + nombre + " tienes " + edad + " años");

        entrada.close();
    }
}
```

La librería que importamos se llama **Scanner** y nos permite almacenar el texto que se ingresa por teclado en la consola. Noten que cuando nosotros usamos ese escáner a través de la palabra “entrada” vamos a escribir **entrada.nextLine()** para las entradas de tipo *String* (texto) y **entrada.nextInt()** para los ingresos de tipo entero.

Luego, a través de los métodos que ya conocemos, podemos imprimir los datos en la pantalla.

Fíjense que para unir el dato de una variable al texto que se está imprimiendo se utiliza el operador **+**.

## Operadores

### De Asignación

El operador de asignación es el **=**. Asigna la expresión del lado derecho al izquierdo. Por supuesto, los tipos deben ser compatibles.

Hay formas abreviadas de expresiones de asignación, por ejemplo:

```
acumulador = acumulador + cuenta; // Forma expandida
```

```
acumulador += cuenta; // Forma abreviada
```

### Aritméticos

Los operadores aritméticos son los siguientes:

- `+` suma
- `-` resta
- `*` multiplicación
- `/` división
- `%` resto

El operador `-` también se puede utilizar de forma unitaria para cambiar el signo a un número, por ejemplo:

```
x = -y;
```

### Enteros

En la división de enteros se truncan los resultados. Ejemplo:

```
byte x = 20 / 3; // x queda con valor 6
```

Si aplicamos el operador resto:

```
byte x = 20 % 3; // x queda con valor 2, porque 6 x 3 = 18, resto 2
```

### Coma flotante

Siguiendo el ejemplo anterior, si hacemos:

```
double x = 20 / 3; // x queda con valor 6.0
```

¿Por qué no toma los valores decimales? Porque 20 y 3 son enteros, por lo tanto, hace la división entera, luego, el resultado, lo promociona (convierte) a *double*. Si quisiéramos que el resultado sea con decimales, deberíamos castear los números a *double* o bien, agregarles un punto cero para que los tome como *doubles*, o, también una *d* al final.

Ejemplos:

```
double x = (double)20 / (double)3; // Con casteo
```

```
double x = 20.0 / 3.0; // División de dos doubles
```

```
double x = 20d / 3d; // División de dos doubles
```

Nota: alcanza con que un solo número sea flotante para que aplique la división flotante. En los ejemplos anteriores podíamos haber casteado solamente el 20 o el 3, con eso sería suficiente.

### Incremento y Decremento

Son los operadores ++ y -- (doble raya), los cuales incrementan o decrementan en uno a la variable, y pueden ser prefijos o sufijos. En el caso de los prefijos, el operador se aplica antes de devolver el valor de la expresión. En el caso sufijo, primero se devuelve el valor de la expresión y, luego, se aplica el operador.

#### Ejemplo

```
int x = 5;

int y = x++;      // Sufijo: x queda con valor 6 e y con valor 5
// Primero asigna el valor a la variable y, y luego incrementa x
```

#### En cambio

```
int x = 5;

int y = ++x;      // Prefijo: ambas variables quedan con valor 6
// Primero incrementa x, y luego asigna el valor a la variable y
```

Estos operadores, también pueden aplicarse a variables de tipo char de modo de obtenerse el siguiente (o anterior) carácter según Unicode.

### Relacionales y de Igualdad

Los operadores relaciones e igualdad producen resultados de tipo boolean (verdaderos o falsos), y son:

- > mayor que
- >= mayor o igual que
- < menor que
- <= menor o igual que
- == igual a
- != distinto a

### Lógicos

Comparan expresiones booleanas y devuelven un resultado booleano.

- && AND (conjunción)
- || OR (disyunción)
- ! NEGACIÓN

### Tablas de verdad

Si *A* y *B* son dos expresiones booleanas que pueden tener cada una el valor verdadero o falso, las siguientes tablas nos indican el resultado de aplicar las expresiones lógicas indicadas arriba.

| A | !A (negación) |
|---|---------------|
| V | F             |
| F | V             |

El operador de negación cambia el valor de la expresión por su contrario: si es verdadera, la considera falsa, y si es falsa, la considera verdadera.

| A | B | A && B (conjunción) | A    B (disyunción) |
|---|---|---------------------|---------------------|
| V | V | V                   | V                   |
| V | F | F                   | V                   |
| F | V | F                   | V                   |
| F | F | F                   | F                   |

El and (conjunción) solo devuelve verdadero si las dos expresiones son verdaderas, en cualquier otro caso, devuelve falso.

Por el contrario, el or (disyunción) devuelve siempre verdadero, salvo que ambas expresiones sean falsas.

Nota: se pueden formar combinaciones de estos operadores con tres o más expresiones. Se recomienda agruparlas con paréntesis.

#### Ejemplo 1:

```
int x = 5;
boolean cierto = true;
((x < 10) && (! cierto))
```

La expresión comienza analizando el primer paréntesis:  $x < 10$ , como *x* vale 5, esa expresión devuelve verdadero (true). Luego, analiza el segundo paréntesis, como *cierto* tiene asignado un valor de verdadero (true), su negación dará falso.

Por último aplica el operador AND (&&) entre un verdadero y un falso, el resultado final será falso.

### Ejemplo 2:

```
int x = 5;
int y = 8;
boolean cierto = false;
((x < 5) || (! cierto)) && (y > 6)
```

En primer lugar, verifica  $x < 5$ , como  $x$  vale 5, el resultado da falso.

En segundo lugar, niega la variable *cierto*, que tiene un valor de falso. Al negarla, el resultado es verdadero. Como hay un or entre estas dos expresiones, da resultado de verdadero, ya que una de las dos es verdadera.

En tercer lugar, evalúa  $y > 6$ , como  $y$  tiene el valor 8, esta expresión da verdadera.

Por último, el resultado del and, es también verdadero.

## Strings (cadenas de caracteres)

La clase String no es un tipo de dato básico, como lo es un *int* o un *double*, pero haremos una primera aproximación a esta clase ya que la utilizaremos bastante para almacenar datos en forma de texto, como puede ser un nombre y apellido, una dirección, etc.

Lo primero que tenemos que distinguir es que cuando nos encontremos utilizando Strings estaremos trabajando con objetos, tema que veremos más adelante. Por ese motivo solo haremos unas pocas menciones sobre este tipo de dato.

Si quisiéramos almacenar un nombre y apellido nos conviene utilizar un String, por ejemplo:

```
String nombre = "Juan";
String apellido = "Pérez";
```

Con el operador "+" podemos concatenar (unir) Strings.

```
String nombreYApellido = nombre + " " + apellido;
```

Como String es una clase, tiene asociados varios métodos (funciones) para manipular los objetos de este tipo. Veremos unos pocos ahora y el resto más adelante.

### Longitud

`length()` : devuelve un entero (int). Este método nos devuelve la cantidad de caracteres del String. Ejemplo:

```
String nombre = "Juan";  
System.out.println(nombre.length()); // imprime 4
```

### Comparación

Hay métodos para saber si un objeto de tipo String es igual a otro. Por ahora veremos solo dos.

#### Igualdad

Para saber si dos Strings son iguales, no nos sirve el operador `==`, debemos utilizar el método `equals`.

Ejemplos:

```
String nombre = "Juan";  
nombre.equals("Juan"); // devuelve verdadero (true)  
nombre.equals("Pedro"); // devuelve false  
nombre.equals("JUAN"); // devuelve false
```

En el último caso, a pesar de que los nombres son iguales, la devolución es falsa porque uno está completamente en mayúsculas, y los códigos de los caracteres en minúsculas son diferentes a los de las mayúsculas.

Si quisiéramos ignorar la diferencia entre mayúsculas y minúsculas, debemos utilizar otro método.

#### Igualdad sin distinguir mayúsculas de minúsculas

El método es `equalsIgnoreCase`. Ejemplo:

```
String nombre = "Juan";  
nombre.equals("JUAN"); // devuelve false  
nombre.equalsIgnoreCase("JUAN"); // devuelve verdadero
```