



Introducción a Java



CARACTERÍSTICAS DE JAVA

- ❖ Simple
- ❖ Orientado a objetos (POO)
- ❖ Multiplataforma
- ❖ Interpretado
- ❖ Portable
- ❖ Multihilo
- ❖ Distribuido
- ❖ Sólido
- ❖ Seguro



Orientado a objetos (POO)

Es un paradigma de programación que propone resolver problemas a través de identificar objetos de la vida real, sus atributos (datos), su comportamiento (acciones) y las relaciones de colaboración entre ellos.

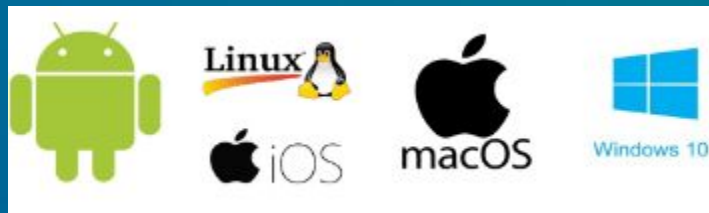


Multiplataforma

Java es independiente de la plataforma.

Depende de la JVM (Máquina Virtual de Java)

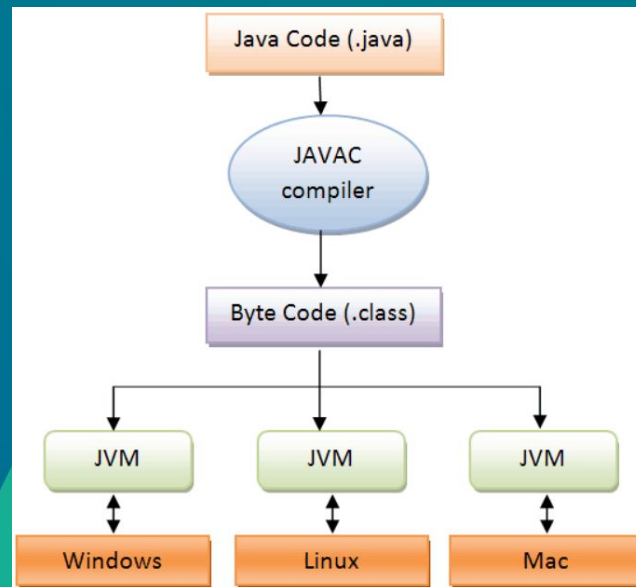
Esta independencia de la plataforma es posible gracias a se es un lenguaje interpretado.



Interpretado

El código en Java, pasa a través del compilador de Java que lo transforma en un Bytecode.

Este Bytecode es interpretado por la JVM, lo que lo hace independiente de la plataforma.





Portable

La multiplataforma es una de las cualidades que lo hacen portable.

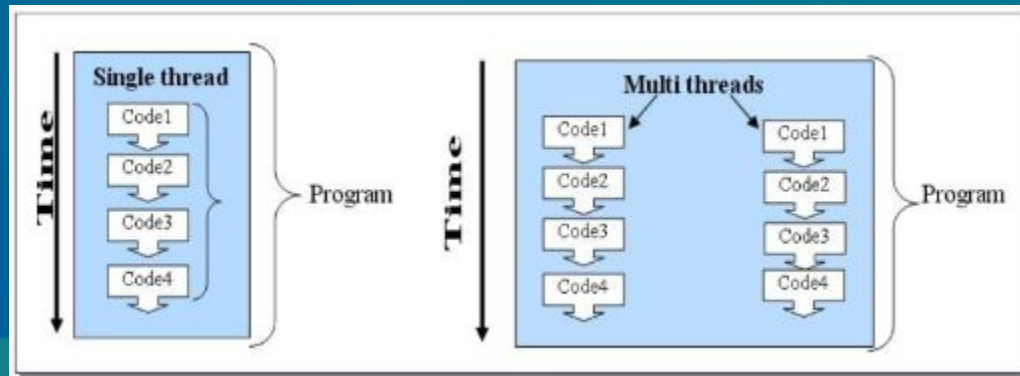
Java, además, especifica el tamaño de los tipos de datos básicos y las operaciones aritméticas.

Todos los programas son iguales en todas las plataformas.

Multihilo

Java puede llevar a cabo varias tareas simultáneamente dentro del mismo programa.

Mejorando el rendimiento y la velocidad de ejecución.





Distribuido



Contiene una gran biblioteca de clases para la utilización del protocolo TCP/IP.

El código de Java se puede manipular a través de recursos URL fácilmente.



Sólido

El código Java usa una sintaxis rigurosa, lo que evita que se quiebre fácilmente ante errores de programación.

Por ejemplo, Java no permite escribir en áreas arbitrarias de memoria, ni realizar operaciones que corrompan el código, como permite C y C++, entre otros.



Seguro

Java está diseñado para trabajar en ambientes de redes, por eso se busca que sea seguro.

No tiene punteros, por lo que no hay problema de destruir áreas internas del ordenador.

Por eso, las cualidades descritas en el punto anterior (Sólido), evitan además de la corrupción del código, su manipulación.

Desde el JDK 1.1, Java ofrece servicios criptográficos.

Admite firmas digitales.

Instalación

Instalar JDK, IDE y complementos.



Instalar el JDK (Java Development Kit)

Acceder a:



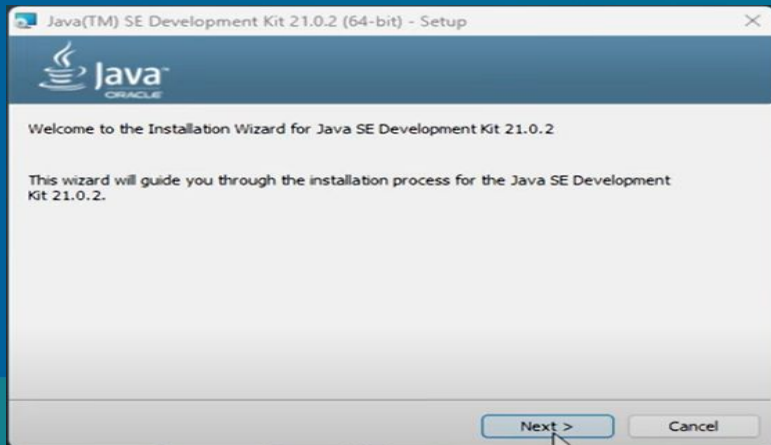
Seleccionar solapa JDK 21 (LTS)

Indicar el S.O. y descargar el instalador

The screenshot shows the Oracle Java Downloads page. At the top, there's an Oracle logo and navigation links like 'Productos', 'Sectores', 'Recursos', etc. The main heading is 'Java Downloads'. Below it, there's a section for 'Java 22, Java 21, and Java 17 available now', stating that JDK 21 is the latest long-term support release. There are buttons for 'OpenJDK Early Access Builds' and 'JRE for Consumers'. At the bottom, there's a section for 'JDK Development Kit 22.0.2 downloads'.

Instalar el JDK (Java Development Kit)

Ejecutar el instalador y completar los pasos con el clásico 'Siguiendo, siguiente... finalizar'.



Instalar la IDE (Integrated Development Environment)



Visual Studio Code





NetBeans

NetBeans es un entorno de desarrollo integrado (IDE) libre, gratuito y sin restricciones de uso, hecho principalmente para JAVA. Existe además un número importante de módulos para extenderlo. Posee, entre otros:

- ❖ Una terminal integrada.
- ❖ Intellisense.
- ❖ Consola de depuración.
- ❖ Gestor de módulos.
- ❖ ... y mucho más.

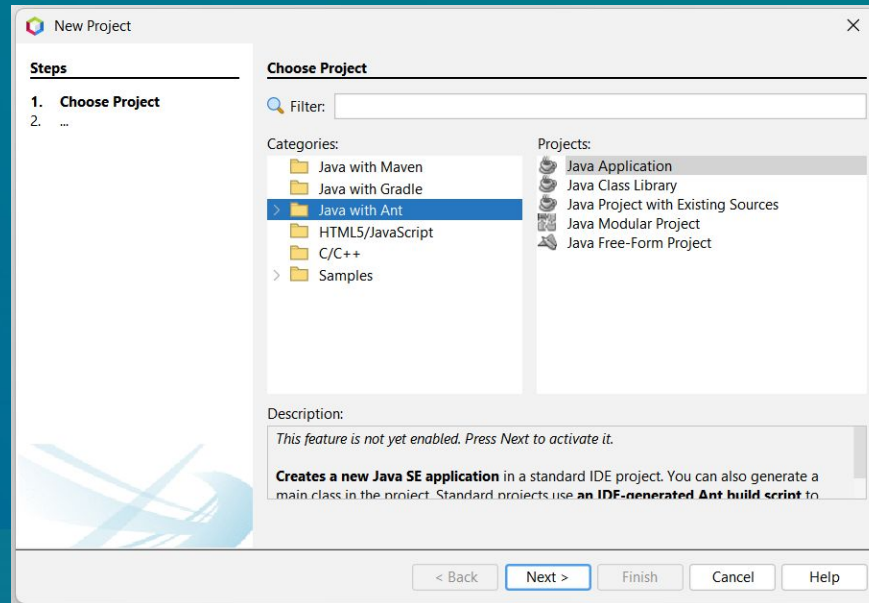




NetBeans

Una vez descargado el instalador,
seleccionar **Nuevo proyecto**:

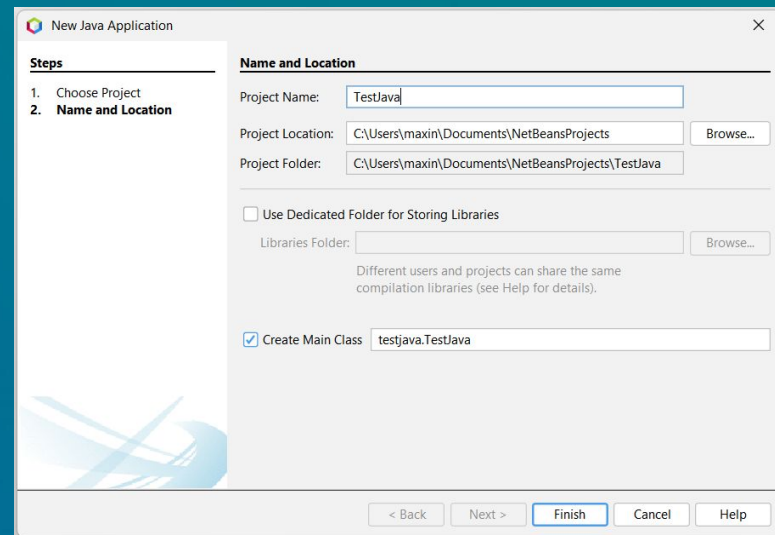
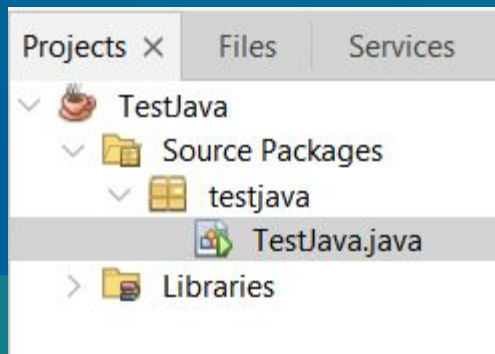
- ❖ Categoría -> *Java with Ant*
- ❖ Proyecto -> *Java Application*





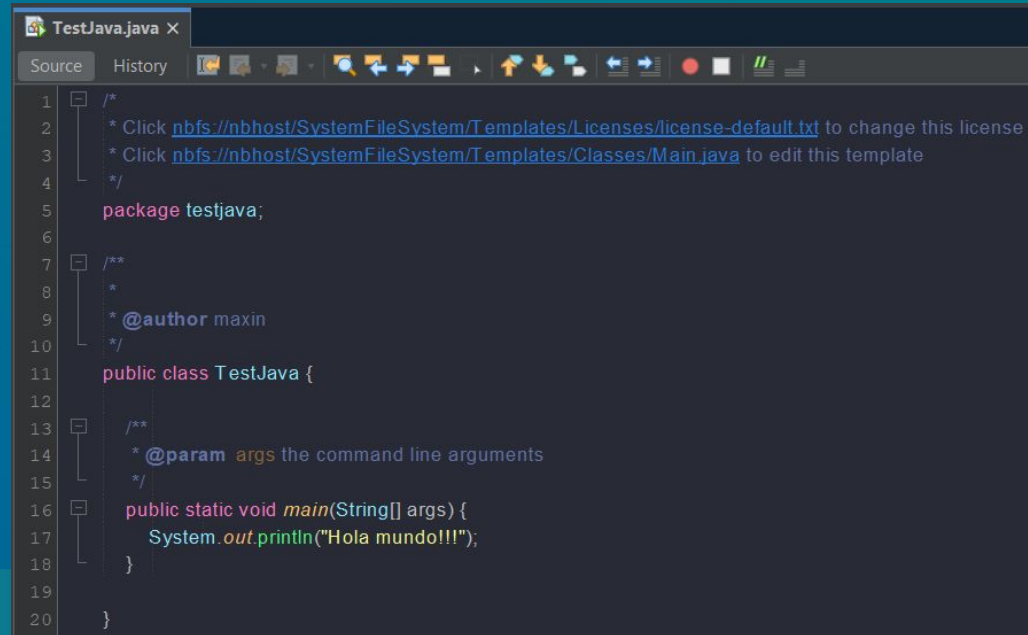
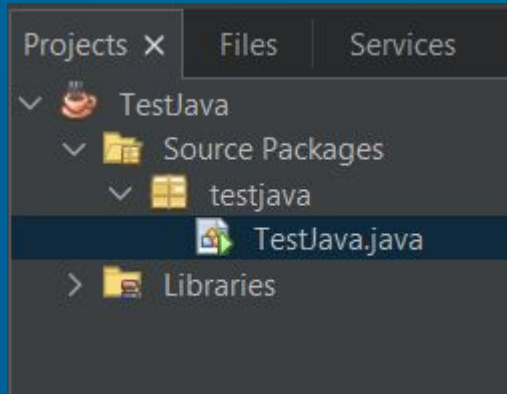
NetBeans

- ❖ Colocar nombre del proyecto.
- ❖ Establecer ubicación del mismo.
- ❖ Finalizar.
- ❖ ¡Listo!





NetBeans



Sintaxis básica

¿Cómo empiezo en Java?

Sintaxis básica

- ❖ Todo el código se escribirá en archivos con extensión *.java*.
- ❖ Los archivos contienen hasta tres estructuras de construcción:
 - Hasta una instrucción *package* seguida del nombre del paquete (siempre en la primera línea del archivo)
 - Cero o más sentencias *import* seguida del nombre cualitativo completo de las clases o '*' por cada paquete a importar.
 - Una definición de clase o interfaz seguida de su nombre y bloque.
- ❖ El nombre del archivo **debe** tener el mismo nombre de la clase pública o interfaz pública que contiene. Recordar que Java es *case sensitive*.



Convención de nombres

- ❖ **Clases:** Usar nombres descriptivos. Deben ser sustantivos con su primera letra en mayúscula y el resto minúscula (UpperCamelCase). Si es una palabra compuesta, las primeras letras de cada palabra en mayúsculas.
Ejemplo: MiClase, Auto, etc.
- ❖ **Métodos:** Usar verbos para los nombres. Comenzar en minúsculas. Si es una palabra compuesta, comenzar con mayúsculas la primera letra de la segunda palabra (camelCase).
Ejemplo: obtenerDato, acelerar, etc.
- ❖ **Paquetes:** Deben ser escritos en minúsculas y deben representar un nombre de dominio (similar a los namespaces de C#).

Convención de nombres

- ❖ **Variables:** Utilizar nombres cortos y descriptivos. Comenzar con minúsculas (si es compuesta, la primera letra de la segunda palabra, en mayúscula). Utilizar nombres que describen la intención de la variable.
Ejemplo: nombre, nombreCliente, edad, etc.
- ❖ **Constantes:** Similar a las variables pero todos sus caracteres en mayúsculas y si son compuestas, separadas por un guión bajo.
Ejemplo: CANTIDAD, EDAD_MAXIMA, etc.

Declaraciones

❖ Variables:

Tipo identificador;

Tipo identificador = valor;

Dónde:

Tipo: Indica el tipo de la variable.

Identificador: establece el nombre de la variable.

```
String nombre;  
nombre = "Juan";  
  
int edad = 22;  
  
float precioProducto = 25.33;
```

Declaraciones

❖ Constantes:

final Tipo IDENTIFICADOR = valor;

Dónde:

```
final int VALOR_ENTERO = 55;
```

final: es una palabra reservada que indica se definirá una constante.

Tipo: Indica el tipo de la constante.

IDENTIFICADOR: establece el nombre de la constante.

valor: establece el valor de la constante.

Variables escalares y no escalares

- ❖ Las **variables escalares** son variables (o constantes) que contienen un dato atómico y unidimensional.
- ❖ Las **variables no escalares** son: arrays (vectores), listas y objetos, que pueden tener almacenado en su estructura más de un valor.

```
//ESCALARES
double numero = 12.33;
final int CONSTANTE = 6;

//NO ESCALARES
int[] numeros = new int[25];
Padre objeto = new Padre();
ArrayList<String> lista = new ArrayList<>();
```

Sintaxis básica

Operadores: aritméticos y lógicos

Operadores aritméticos

Asignación	=
Adición	+
Sustracción	-
Multiplicación	*
División	/

Módulo	%
Mayor que	>
Menor que	<
Mayor o igual que	>=
Menor o igual que	<=

Operadores lógicos

Igualdad	==
Desigualdad	!=
AND	&&
OR	
Negación	!

Sintaxis básica

Estructuras de control: condicionales y repetitivas

Condicionales

- ❖ Estructuras **if** con varios formatos:

```
if(x < 10)
    x++;

if(x > 10)
{
    x--;
    y = 0;
}
```

```
if(y == 0)
{
    x = 3;
}
else
{
    x = 0;
}
```

```
if(x == y)
{
    y--;
    x++;
}
else if(x > y)
{
    y += x;
}
else
{
    x = y;
}
```

Condicionales

❖ Estructuras **switch**:

```
switch(x)
{
    case 6:
        //código
        break;
    case 2:
        //código
        break;
    default:
        //código defecto
        break;
}
```

```
String cadena = "hola";
switch(cadena)
{
    case "hola":
        //código
        break;
    case "mundo":
        //código
        break;
    default:
        //código defecto
        break;
}
```

Repetitivas

❖ Estructuras **for**:

```
//inicialización; prueba; acción
for(int index = 0; index < 5; index++)
{
    //código
}
```

```
//tipo elemento : colección de elementos
for(iterable_type iterable_element : iterable)
{
    //código
}
```

```
int[] coleccion = new int[2];
for (int i : coleccion)
{
    //código
}
```


Repetitivas

❖ Estructuras **while** y **do-while**:

```
while(x < 10)
{
    x++;
}
```

```
do
{
    x++;
} while (x < 10);
```

Entrada y salida de datos

Entry point



Entry point

```
public class App {  
  
    /**  
     * @param args the command line arguments  
     */  
    public static void main(String[] args)  
    {  
        System.out.println("Hola, mundo!!!");  
    }  
}
```

El punto de entrada a los programas de Java es el método público y estático **main**.

Dónde:

static: es el modificador que permite ejecutar un método sin tener que instanciar un objeto.

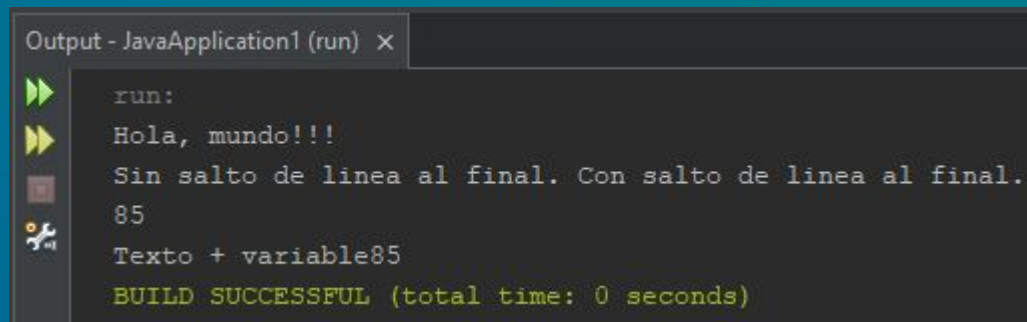
void: indica que este método no retornará ningún valor.

String[] args: es un array de tipo String que puede recibir el método.

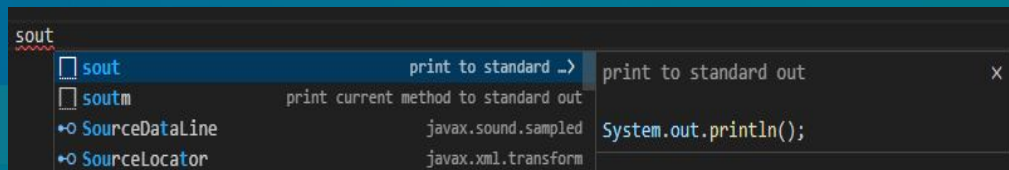
Salida por consola

- ❖ Para imprimir por la salida estándar se utilizan los métodos estáticos:

```
public class App {  
  
    /**  
     * @param args the command line arguments  
     */  
    public static void main(String[] args)  
    {  
        System.out.println("Hola, mundo!!!");  
  
        int variable = 85;  
  
        System.out.print("Sin salto de linea al final. ");  
  
        System.out.println("Con salto de linea al final.");  
  
        System.out.println(variable);  
  
        System.out.println("Texto + variable" + variable);  
    }  
}
```



Utilizar fragmentos de código:



Ingreso por consola

- ❖ Para ingresar desde la consola se utilizan los métodos de instancia:

Utilizando try con recursos:

```
//importar el paquete java.util.Scanner;
Scanner input = new Scanner(System.in);

String cadena = input.nextLine();
Double doble = input.nextDouble();
Integer entero = input.nextInt();

//NO olvidar cerrar el input
input.close();
```

```
String cadena;
Double doble;
Integer entero;
try ( Scanner input = new Scanner(System.in) )
{
    cadena = input.nextLine();
    doble = input.nextDouble();
    entero = input.nextInt();
    //NO olvidar cerrar el input
}
catch (Exception e)
{
    System.err.println(e);
}
```

Ejercitación