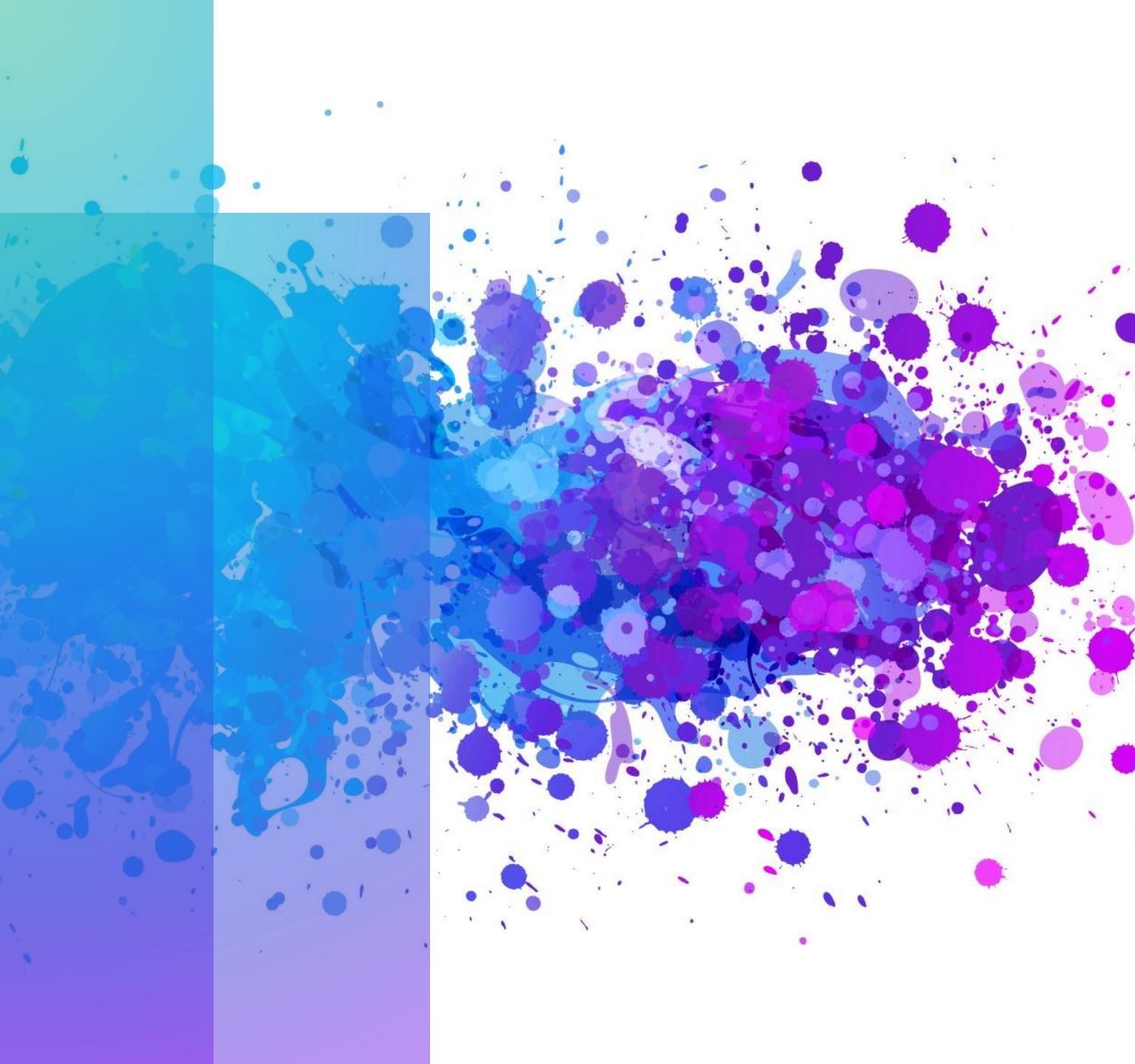




# Introducción a la Programación y Computación 1 Sección G

Max Rodrigo Durán Canteo

<b>Día, Fecha:</b>	Miércoles, 5 de febrero
<b>Hora de inicio:</b>	4:30 – 6:10 pm

The background of the slide features a abstract design on the left side. It consists of a grid of small, semi-transparent blue and purple dots. Overlaid on this grid are larger, semi-transparent circles in shades of blue, purple, and magenta, creating a sense of depth and motion. The right side of the slide is a solid black color.

# Clase 3

Introducción a JAVA

# Contenido



# ¿Qué es JAVA?

## (Definición/Reseña Histórica)



- Java es una tecnología que se usa para el desarrollo de aplicaciones que convierten a la Web en un elemento más interesante y útil. Java no es lo mismo que JavaScript, que se trata de una tecnología sencilla que se usa para crear páginas web y solamente se ejecuta en el explorador.
- Es un lenguaje de programación y una plataforma informática comercializada por primera vez en 1995 por Sun Microsystems. Hay muchas aplicaciones y sitios web que no funcionarían a menos que se disponga de Java instalado, y cada día se crean más. Java es rápido, seguro y fiable. Se utiliza en portátiles, centros de datos, consolas para juegos, super computadoras, teléfonos móviles e Internet. Java está en todas partes, y es ejecutado en una plataforma que no tiene que ser recompilado para correr en otra. Java es, a partir de 2012, uno de los lenguajes de programación más populares en uso, particularmente para aplicaciones de cliente-servidor de web, con unos diez millones de usuarios reportados.

# Versiones y Entorno de JAVA

## (Definición/Reseña Histórica)

### JAVA 1.0 A 1.2

- 1.0: Versión de lanzamiento, incluye las clases principales, máquina virtual, y AWT.
- 1.1: Incorpora Readers/Writers, Calendars y Bundles. Notable por la inclusión de JavaBeans y JDBC.
- 1.2: Incorpora Collections y Swing.

### JAVA 1.3 A 1.5

- 1.3: Máquina virtual con compilación JIT.
- 1.4: APIs de soporte XML, RegEx, y criptografía.
- 1.5: Java reconoce estas versiones como SE. Inclusión de tipado genérico y anotaciones.

## JAVA 6 A 8

- 1.6: Compilación “on the fly” para servicios web.
- 1.7: Mejoras en la máquina virtual y recolector de basura.
- 1.8: Expresiones lambda, streams.

## JAVA 9 en Adelante

Estas versiones incluyen cambios que en su mayoría refieren únicamente a corrección de errores y mejoras en el rendimiento de APIs.

- 9: Mejoras en seguridad, y corrección de errores. Soporte para un heap de tamaño multi-gigabyte.
- 10: Corrección de errores y mejoras de seguridad.
- 11: Se eliminaron Java Applets y Java Web Start.
- 12: Corrección de errores y mejoras de seguridad.
- 13: Corrección de errores y mejoras de seguridad.

Version	Type	Class file format version <sup>[7]</sup>	Release date	End of public updates (free)	End of extended support (paid)
JDK 1.0		45 <sup>[8]</sup>	23rd January 1996	May 1996	—
JDK 1.1		45	18th February 1997	October 2002	—
J2SE 1.2		46	4th December 1998	November 2003	—
J2SE 1.3		47	8th May 2000	March 2006	—
J2SE 1.4		48	13th February 2002	October 2008	—
J2SE 5.0 (1.5)		49	30th September 2004	October 2009	—
Java SE 6 (1.6)		50	11th December 2006	April 2013 for Oracle December 2018 for Azul <sup>[9]</sup>	December 2016 for Red Hat <sup>[10]</sup> October 2018 for Oracle <sup>[11]</sup> December 2027 for Azul <sup>[9]</sup> March 2028 for BellSoft Liberica <sup>[12]</sup>
Java SE 7 (1.7)		51	28th July 2011	July 2015 for Oracle July 2022 for Azul <sup>[9]</sup>	June 2020 for Red Hat <sup>[10]</sup> July 2022 for Oracle <sup>[13]</sup> December 2027 for Azul <sup>[9]</sup> March 2028 for BellSoft Liberica <sup>[12]</sup>
Java SE 8 (1.8)	LTS	52	18th March 2014	April 2019 for Oracle November 2026 for Eclipse Temurin <sup>[14]</sup> November 2026 for Red Hat <sup>[10]</sup> November 2026 for Azul <sup>[9]</sup> December 2030 for Amazon Corretto <sup>[15]</sup>	December 2030 for Oracle <sup>[4]</sup> December 2030 for Azul <sup>[9]</sup> March 2031 for BellSoft Liberica <sup>[12]</sup>
Java SE 9 (1.9)		53	21st September 2017	March 2018	—
Java SE 10 (1.10)		54	20th March 2018	September 2018	—

Java SE 11	LTS	55	25th September 2018	April 2019 for Oracle September 2027 for Microsoft Build of OpenJDK <sup>[16]</sup> October 2024 for Red Hat <sup>[10]</sup> October 2027 for Eclipse Temurin <sup>[14]</sup> October 2027 for Azul <sup>[9]</sup> January 2032 for Amazon Corretto <sup>[15]</sup> January 2032 for Azul <sup>[9]</sup>	January 2032 for Azul <sup>[9]</sup> March 2032 for BellSoft Liberica <sup>[12]</sup> October 2027 for Red Hat <sup>[10]</sup> January 2032 for Oracle <sup>[4]</sup>
Java SE 12		56	19th March 2019	September 2019	—
Java SE 13		57	17th September 2019	March 2020	—
Java SE 14		58	17th March 2020	September 2020	—
Java SE 15		59	16th September 2020	March 2021	—
Java SE 16		60	16th March 2021	September 2021	—
Java SE 17	LTS	61	14th September 2021	September 2024 for Oracle <sup>[4]</sup> September 2027 for Microsoft Build of OpenJDK <sup>[16]</sup> October 2027 for Eclipse Temurin <sup>[14]</sup> October 2027 for Red Hat <sup>[10]</sup> October 2029 for Amazon Corretto <sup>[15]</sup> September 2029 for Azul <sup>[9]</sup>	September 2029 for Oracle <sup>[4]</sup> March 2030 for BellSoft Liberica <sup>[12]</sup>
Java SE 18		62	22nd March 2022	September 2022	—
Java SE 19		63	20th September 2022	March 2023	—
Java SE 20		64	21st March 2023	September 2023	—
Java SE 21	LTS	65	19th September 2023	September 2028 for Oracle <sup>[4]</sup> September 2028 for Microsoft Build of OpenJDK <sup>[16]</sup> December 2029 for Red Hat <sup>[10]</sup> December 2029 for Eclipse Temurin <sup>[14]</sup> October 2030 for Amazon Corretto <sup>[15]</sup> September 2031 for Azul <sup>[9]</sup>	September 2031 for Oracle <sup>[4]</sup> March 2032 for BellSoft Liberica <sup>[12]</sup>
Java SE 22		66	19th March 2024	September 2024	—
Java SE 23		67	17th September 2024	March 2025 for Oracle September 2032 for Azul <sup>[9]</sup>	—
Java SE 24		68	March 2025	September 2025	—
Java SE 25	LTS	69	September 2025	September 2030 for Oracle <sup>[4]</sup>	September 2033 for Oracle <sup>[4]</sup> March 2034 for BellSoft Liberica <sup>[12]</sup>

Legend:  Unsupported version  Old version, still maintained  Latest version  Future release

## JVM

- Es una máquina virtual de proceso nativo, es decir, ejecutable en una plataforma específica, capaz de interpretar y ejecutar instrucciones expresadas en un código binario especial (el bytecode Java), el cual es generado por el compilador del lenguaje Java.

## JRE

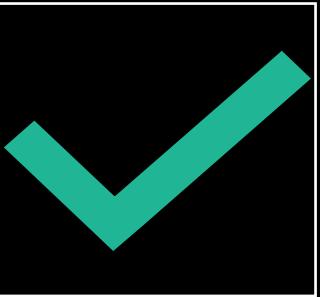
- Significa Java Runtime Environment, y son un conjunto de utilidades que permiten la ejecución de aplicaciones Java.
- Está constituido por:
  - Máquina Virtual de JAVA
  - Librerías de JAVA

## JDK

- Significa Java Development Kit, y son un conjunto de utilidades que permiten el desarrollo de aplicaciones Java.
- Está constituido por:
  - Máquina Virtual de JAVA
  - Librerías de JAVA
  - Compilador de JAVA

# Características de JAVA

## ¿Por qué JAVA?



### Simple

Sintaxis similar a la de C y C++, pero sin características de bajo nivel:

- Memoria dinámica automática
- Sin aritmética de punteros
- Sin structs
- Sin typedefs
- Sin pre-procesamiento



### Independencia de la Arquitectura

Los programas corren en la JVM, que es independiente del dispositivo, dando soporte multiplataforma a un lenguaje compilado.

---

**Portable y Distribuido**

Implementa estándares que permiten portabilidad adicional. Además, posee extensiones con soporte de red, protocolo HTTP, TCP/IP y FTP.

---

**Robusto y Seguro**

El compilador y el intérprete manejan los errores internamente, administrando la memoria y verificando todo tipo de excepción.

---

No permite accesos ilegales a la memoria, ni a clases internas. Se puede limitar entre aplicaciones y clases, validando internamente el código compilado.

---

**Alto Rendimiento, Multihilo y Dinámico**

El código objeto que produce es altamente eficiente.

---

Permite multihilo para un comportamiento interactivo con los usuarios.

---

La memoria crece conforme se accede a los módulos solicitados en la ejecución.

---

# Tipos de Programas en JAVA

**¿Qué tipo de programas se pueden hacer con JAVA?**

## Tipos de Aplicaciones en JAVA

- Aplicaciones de consola: Desde una CLI, se muestra como texto toda salida de la aplicación, o se ejecutan en el trasfondo instrucciones que procesan el programa.
- Aplicaciones de interfaz gráfica: Hacen uso de librerías de interfaz de usuario para mostrar información en ventanas y otros componentes interactivos.

## Tipos de Aplicaciones en JAVA

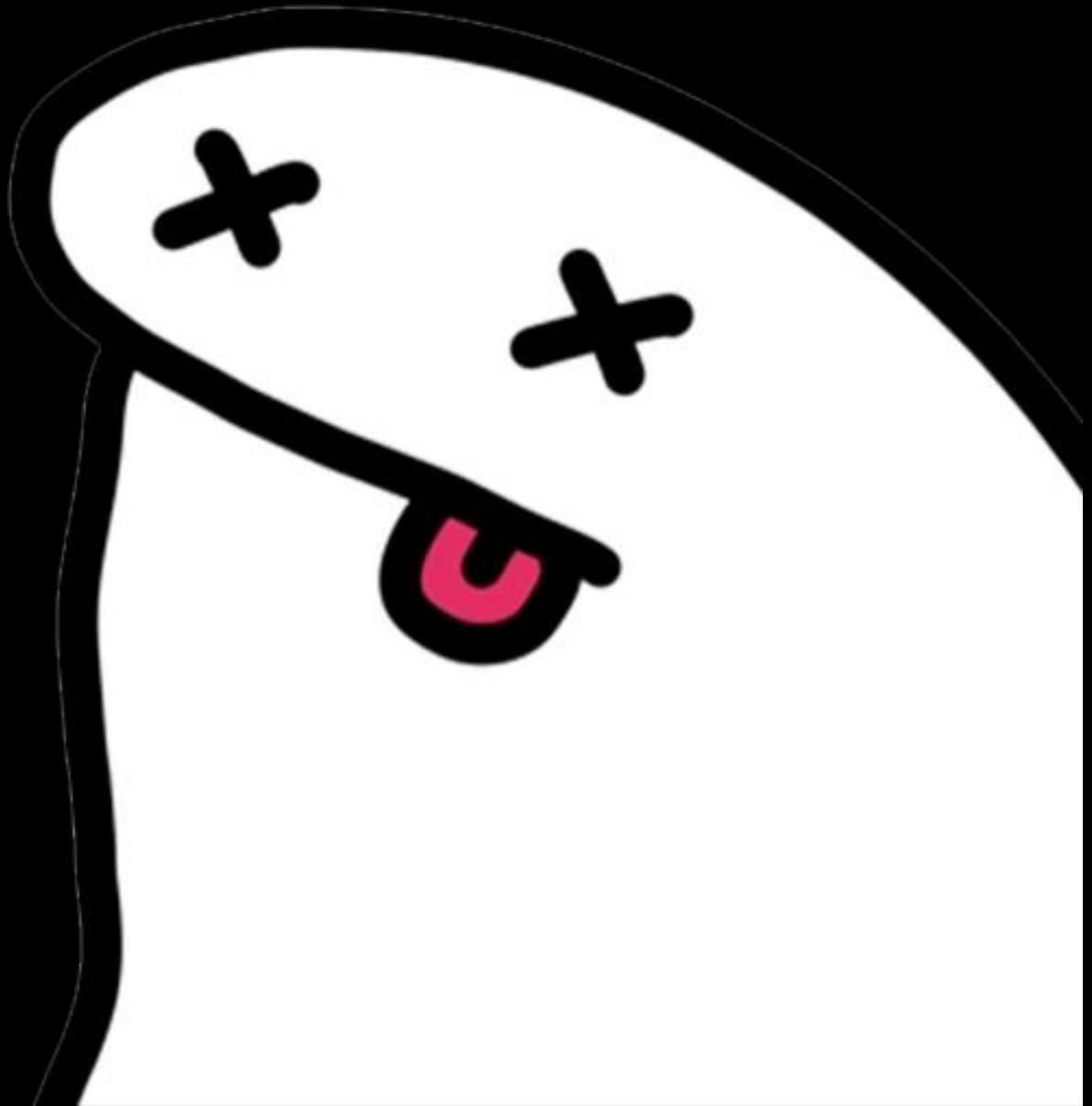
- Sistemas embebidos: Aquellos que consisten en chips de computadoras especializadas para otros sistemas electromecánicos que realizan tareas especializadas (reproductores multimedia, tarjetas SIM, televisores, etc.) hacen uso de tecnología Java.
- Aplicaciones web: Aplicaciones web, JSP y sistemas de seguridad especializados del lado del servidor.
- Servidores y aplicaciones de servidores: Aplicaciones y sistemas dedicados al alojamiento de páginas web.

## Tipos de Aplicaciones en JAVA

- Aplicaciones empresariales: Sistemas comerciales están desarrollados enteramente en Java y reciben soporte especializado del fabricante. Se utilizan como un sistema centralizado de información
- Aplicaciones científicas: Aplicaciones dedicadas para la investigación y experimentación científica.
- Aplicaciones móviles: Aplicaciones que se desarrollan específicamente para dispositivos que soportan Java Platform Micro Edition.

# Versión para usar en Laboratorio

- JAVA 23.0.2



# Descarga JAVA

- OpenJDK (Instalación Manual)

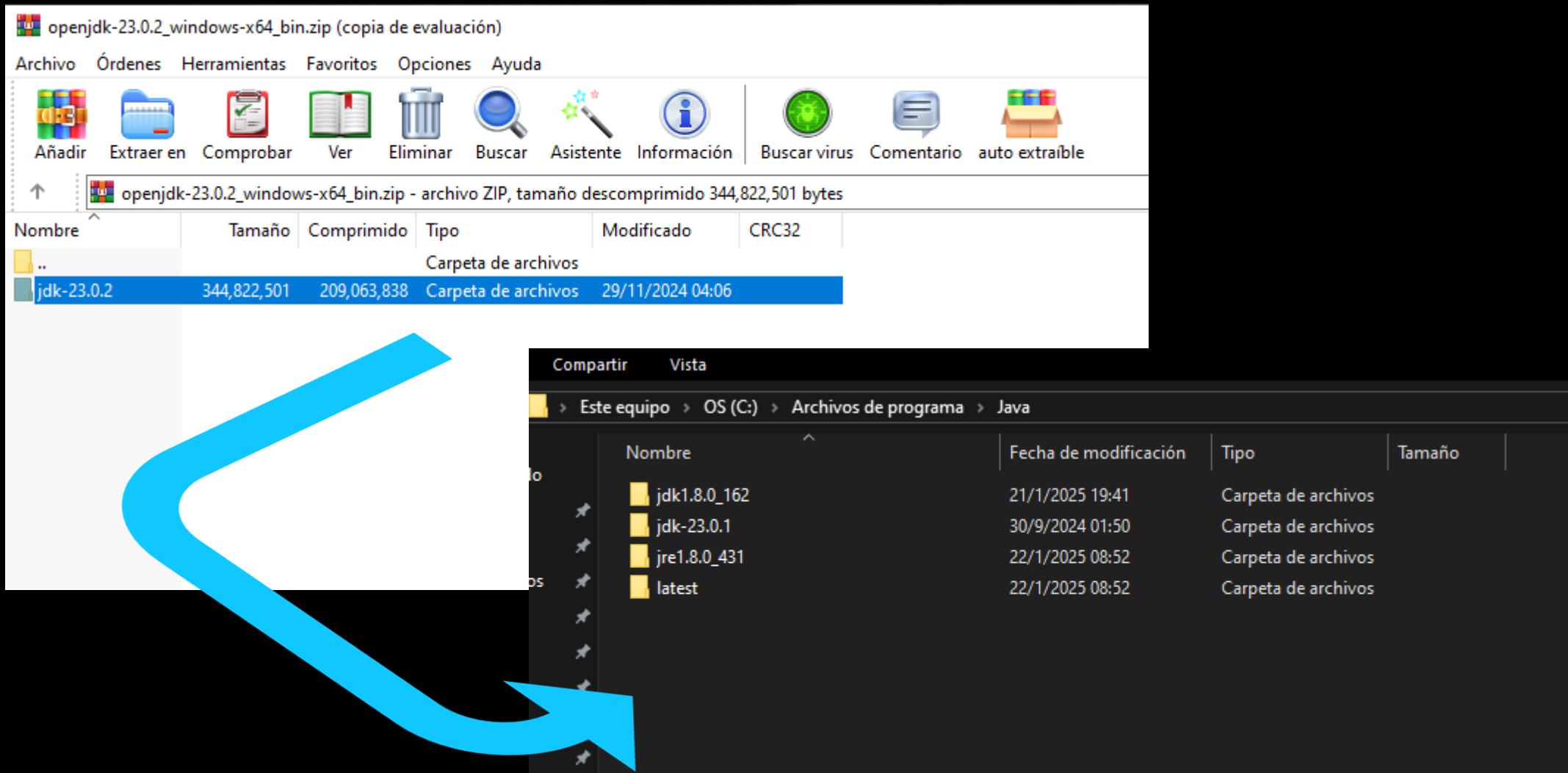
<https://jdk.java.net/23/>

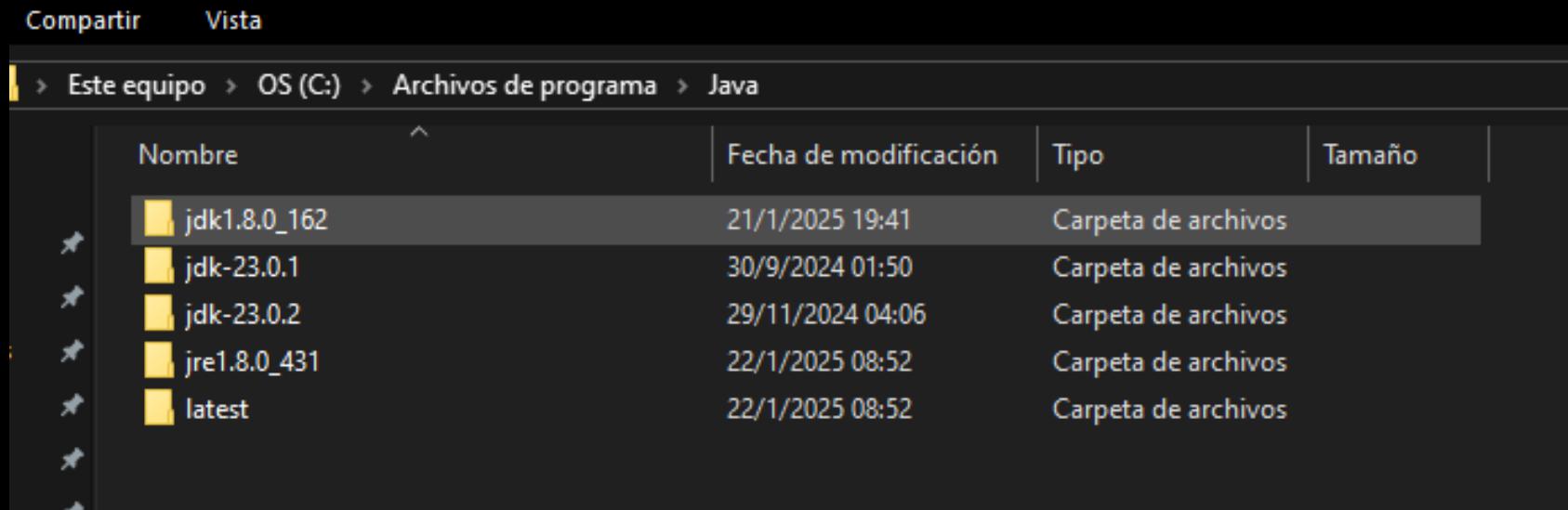
- Página Oficial de Oracle

<https://www.oracle.com/java/technologies/downloads/#jdk23-windows>



# Ajustes con OpenJDK (Windows)



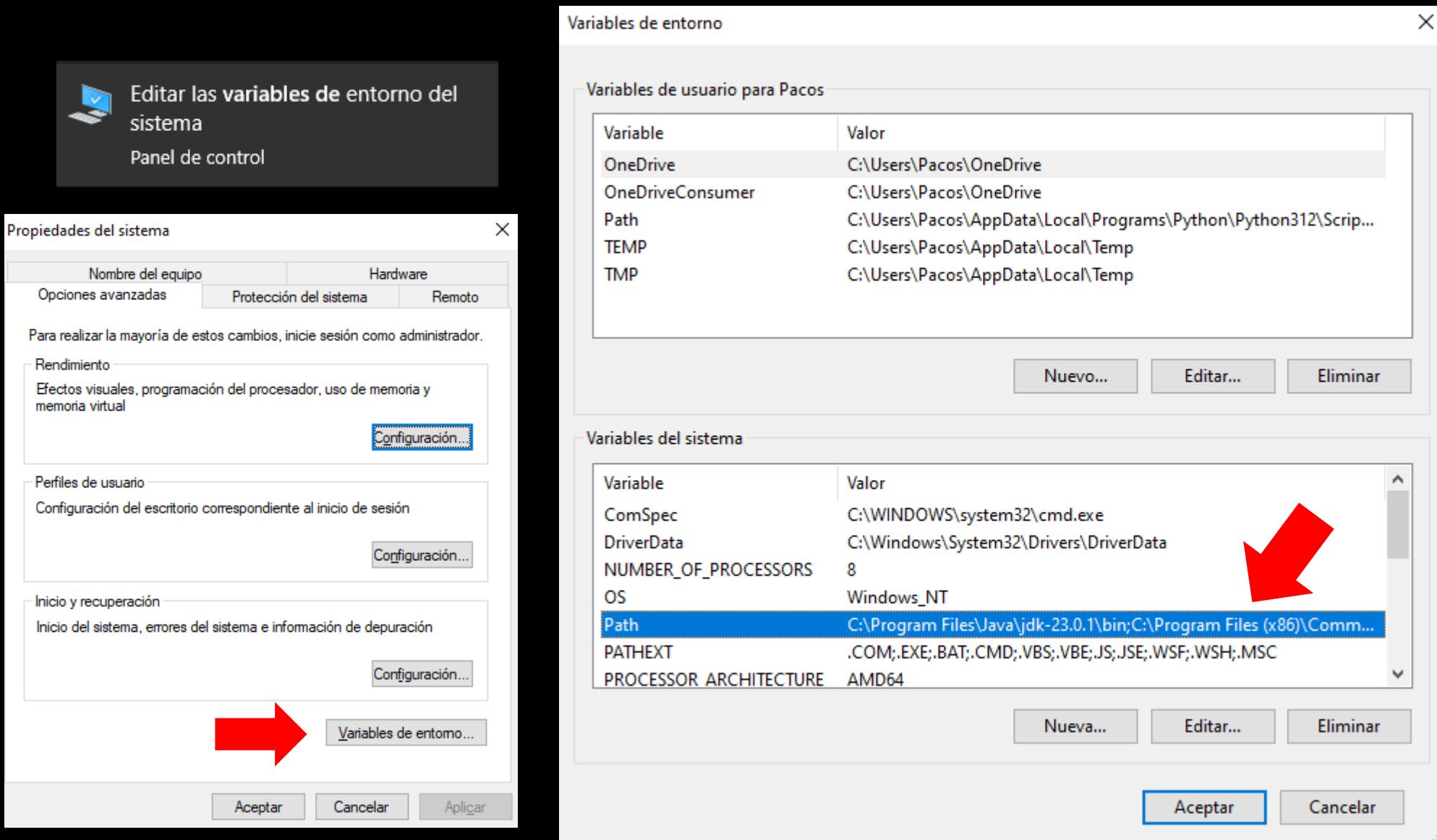


The screenshot shows a Windows File Explorer window with the following path: Este equipo > OS (C:) > Archivos de programa > Java. The table lists several Java installations:

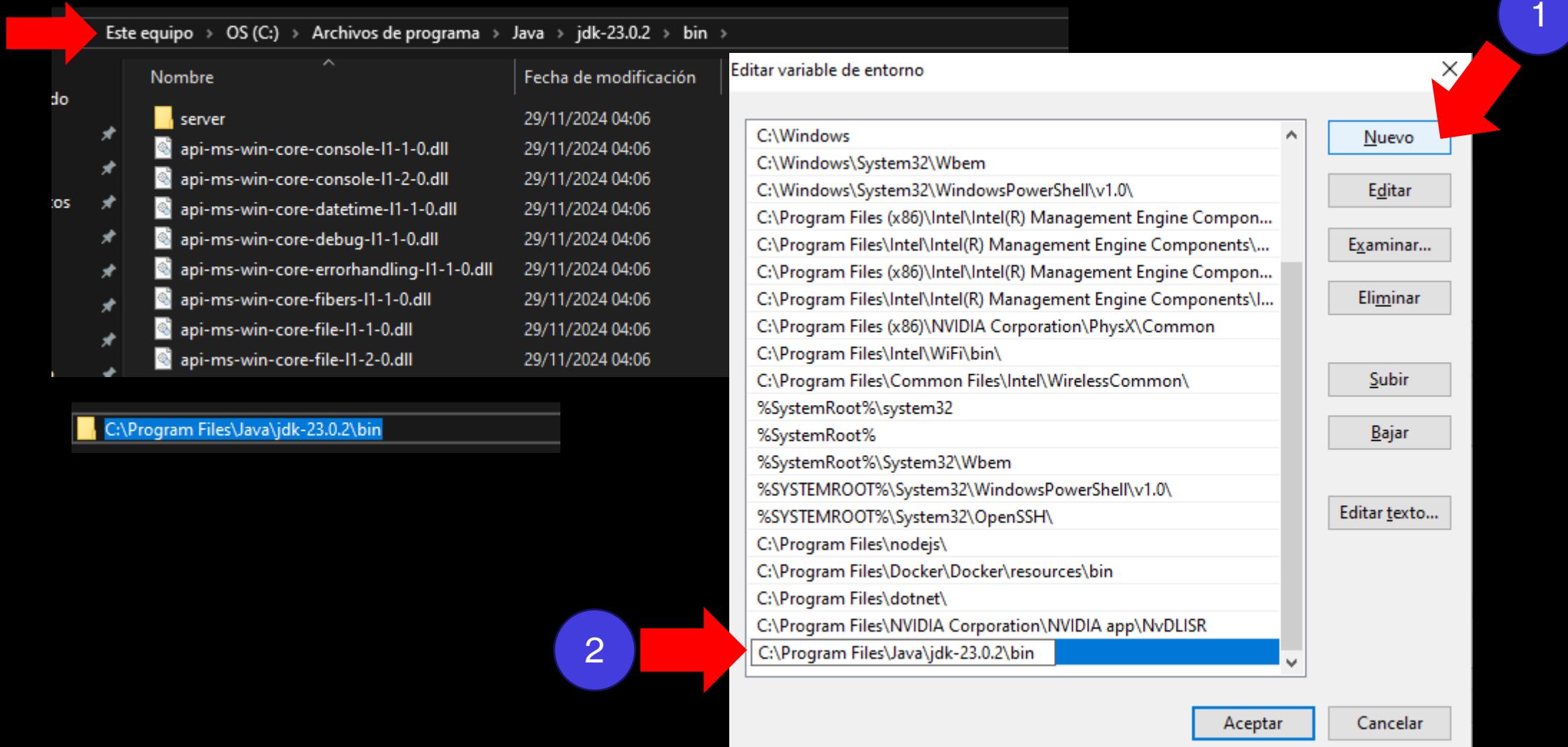
Nombre	Fecha de modificación	Tipo
jdk1.8.0_162	21/1/2025 19:41	Carpeta de archivos
jdk-23.0.1	30/9/2024 01:50	Carpeta de archivos
jdk-23.0.2	29/11/2024 04:06	Carpeta de archivos
jre1.8.0_431	22/1/2025 08:52	Carpeta de archivos
latest	22/1/2025 08:52	Carpeta de archivos

# Ajustes con OpenJDK (Windows)

# Editar Variables de Entorno

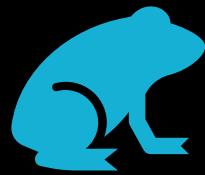


# Editar Variables de Entorno



# Instalación Versiones Linux

## OpenJDK

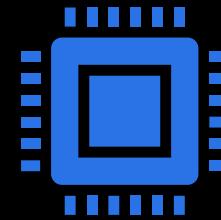


### Ubuntu/Debian

```
sudo apt update
```

```
sudo apt upgrade
```

```
sudo apt install openjdk-23-jdk
```



### Arch Linux

```
sudo pacman -Syyu
```

```
sudo pacman -S jdk23-openjdk
```

```
PS C:\Users\Pacos> java -version
openjdk version "23.0.2" 2025-01-21
OpenJDK Runtime Environment (build 23.0.2+7-58)
OpenJDK 64-Bit Server VM (build 23.0.2+7-58, mixed mode, sharing)
PS C:\Users\Pacos> javac -version
javac 23.0.2
PS C:\Users\Pacos> |
```

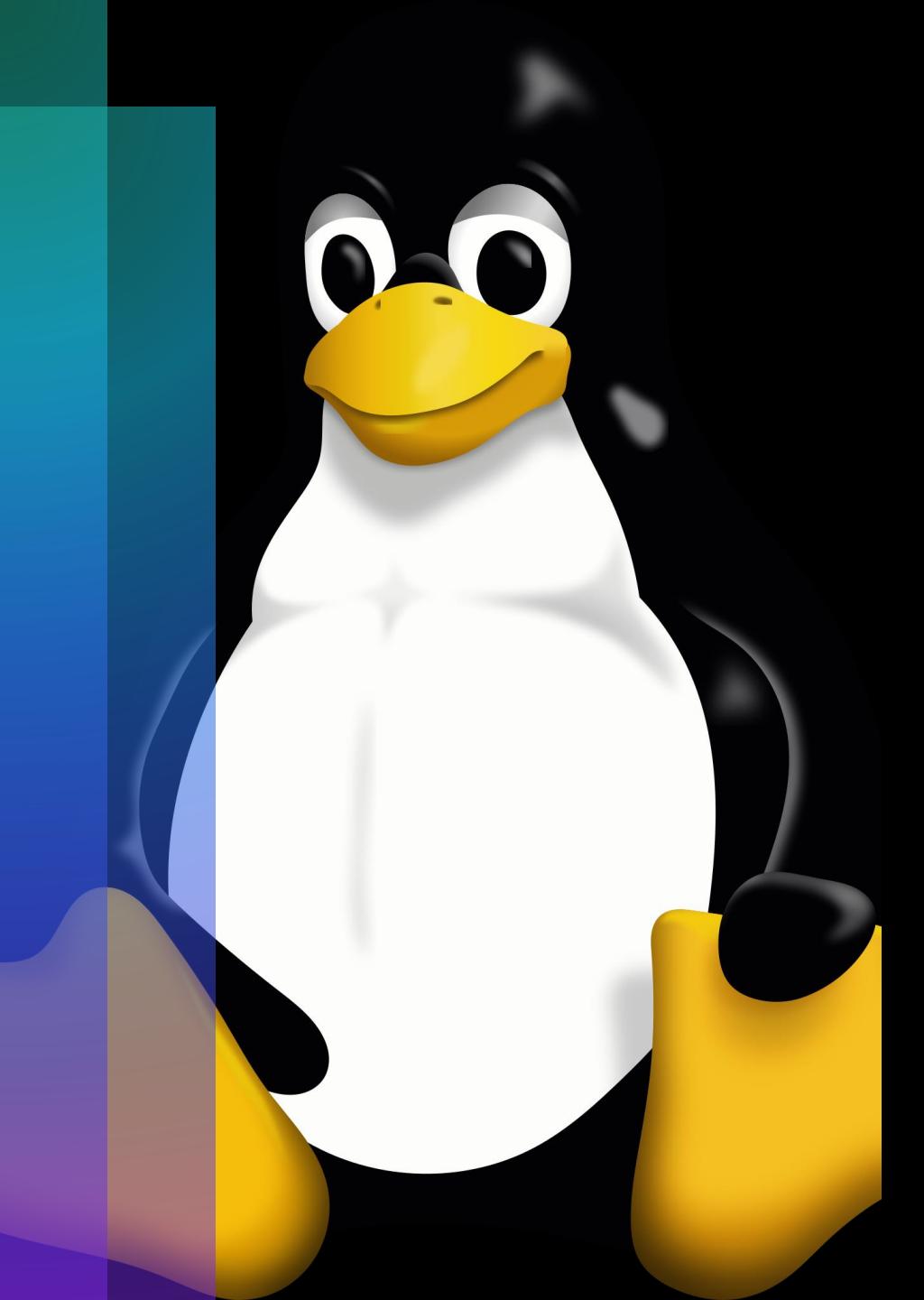
# Verificación que esté instalado

- java –versión
- javac -version

# Instalación NetBeans (Windows)

- Página oficial

[https://netbeans.apache.org/  
front/main/download/](https://netbeans.apache.org/front/main/download/)



# Instalación NetBeans (Linux)

Ubuntu/Debian

- Descargar de la página oficial

<https://netbeans.apache.org/front/main/download>

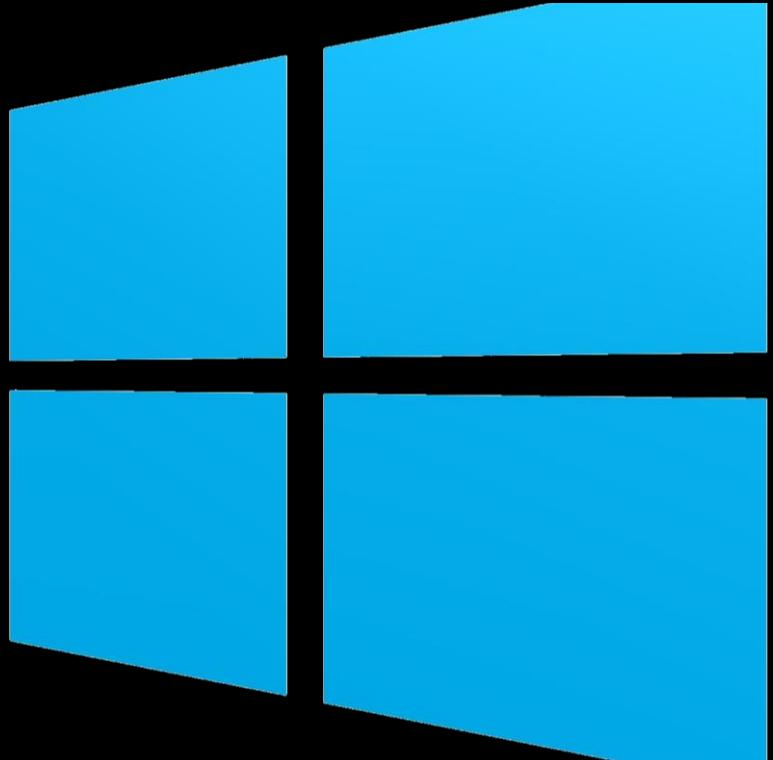
- cd ~/Descargas
- sudo dpkg –i apache-netbeans\_XX-X\_all.deb

Arch Linux

- sudo pacman –S netbeans

# Instalación IntelliJ IDEA (Windows)

- Pagina Oficial (Community)
  - <https://www.jetbrains.com/idea/download/other.html>
- Pagina Oficial (Ultimate)
  - <https://account.jetbrains.com/licenses>



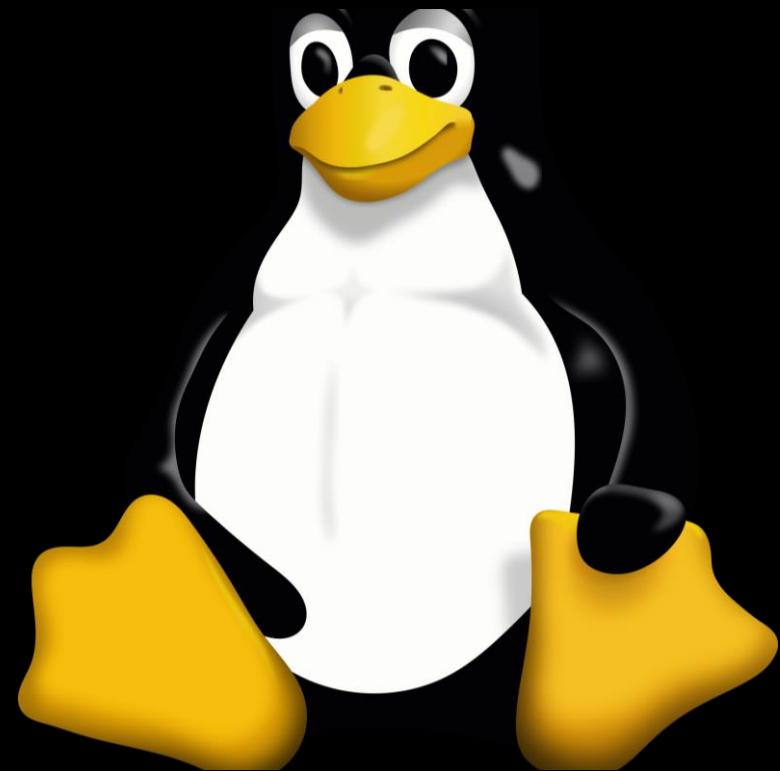
# Instalación IntelliJ IDEA (Linux)

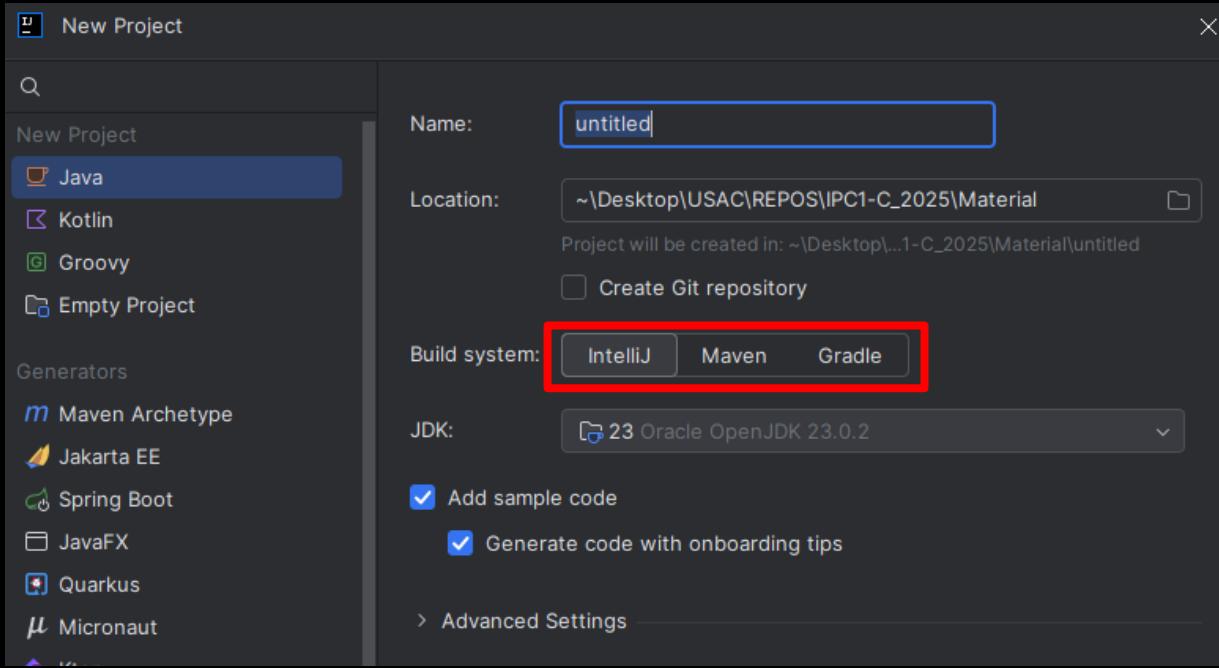
## Ubuntu/Debian

- sudo apt update && sudo apt upgrade -y
- sudo apt install snapd -y
- sudo snap install intellij-idea-community --classic
- Sudo snap install intellij-idea-ultimate –classic

## Arch Linux

- paru -S intellij-idea-community-edition
- paru -S intellij-idea-ultimate-edition

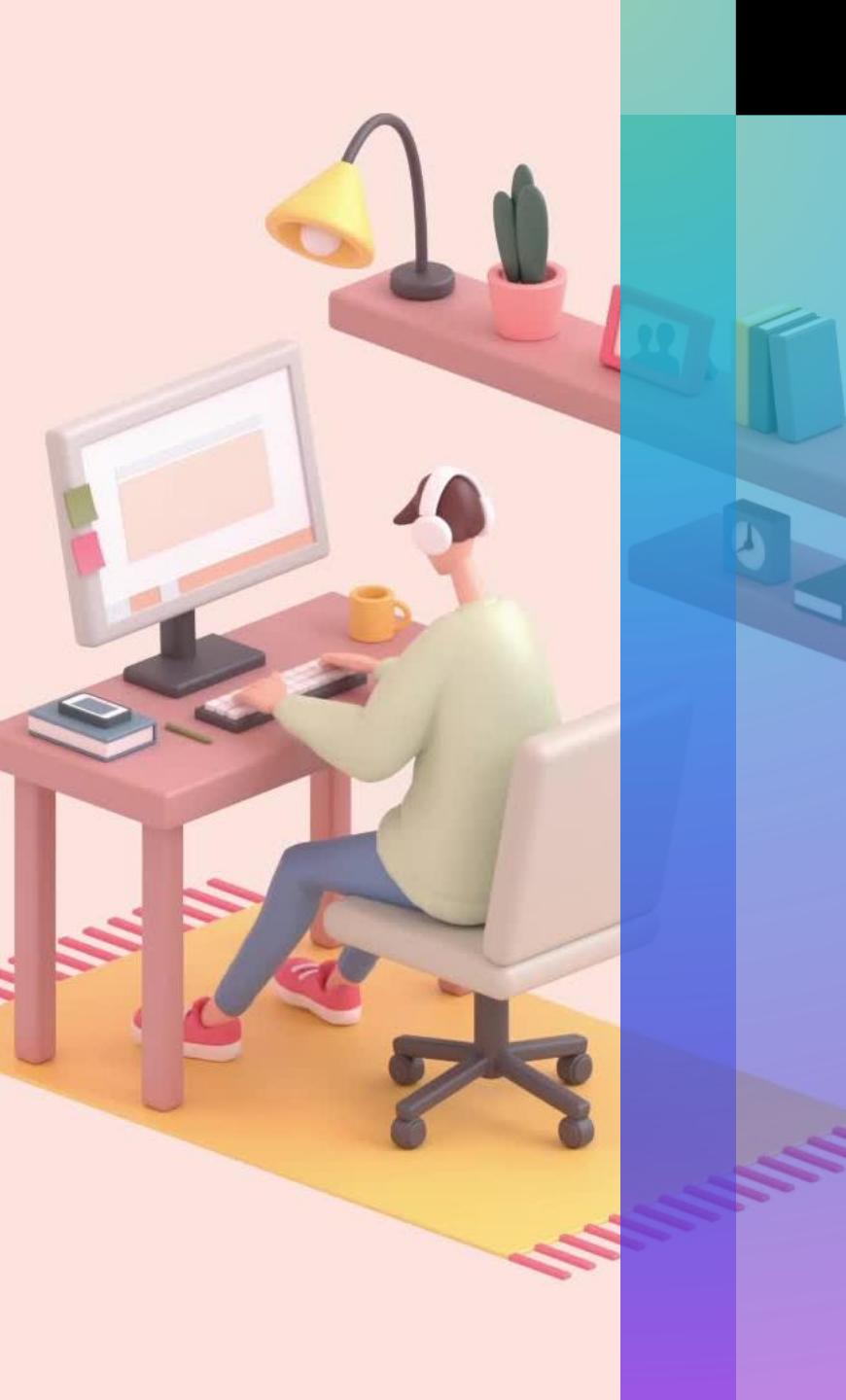




# Creación Proyectos (IntelliJ IDEA)

Al crear un nuevo proyecto aparecen las siguientes opciones:

- IntelliJ
- Maven
- Gradle



# IntelliJ (Proyecto simple)

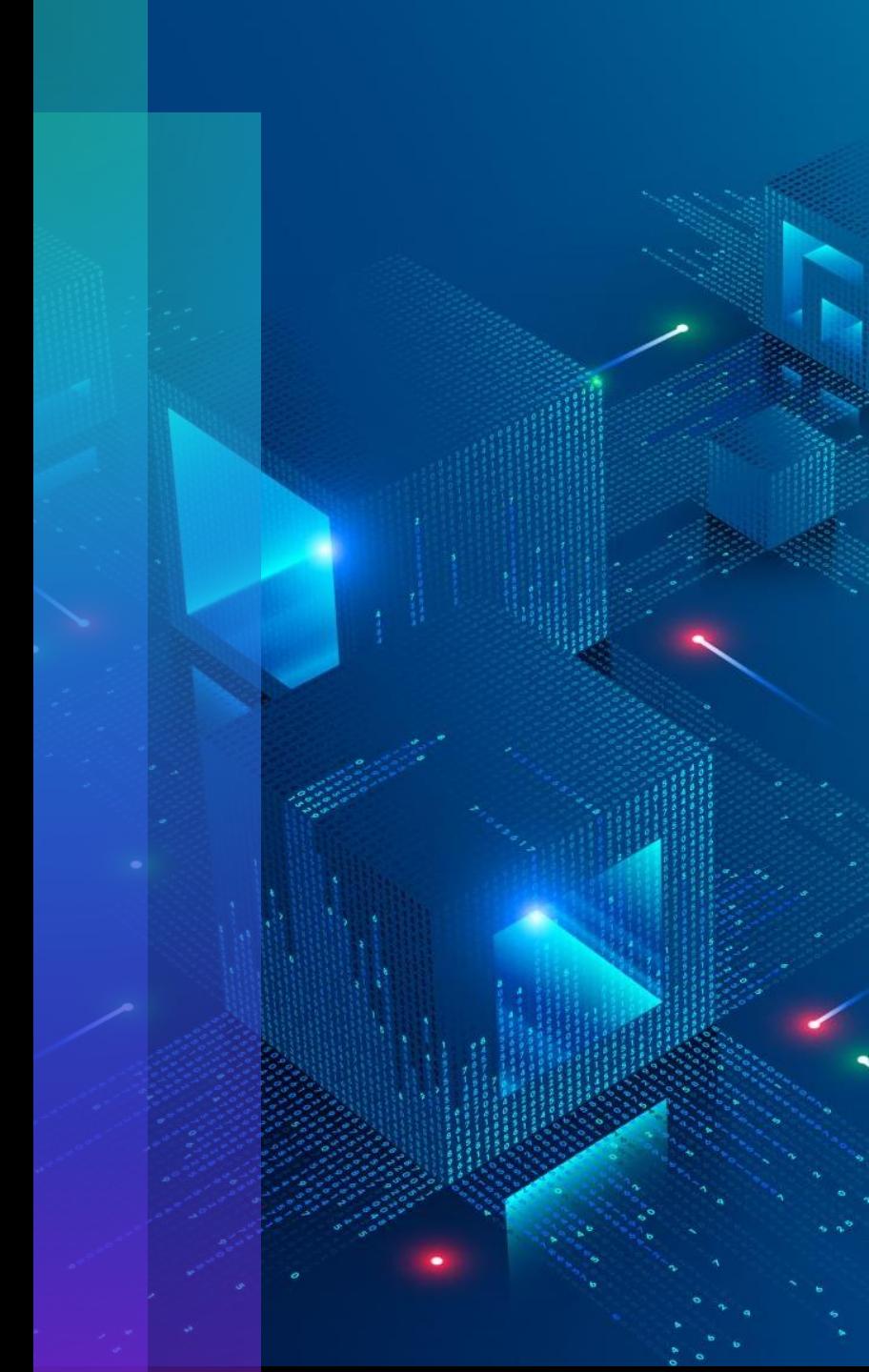
Es un sistema de construcción integrado en IntelliJ IDEA, se utiliza para proyectos pequeños, sin la necesidad de configuraciones externas.

- Cuando usarlo:
  - Trabajos simples como aplicaciones de consola o aplicaciones pequeñas de escritorio.
  - En caso de no necesitar gestionar dependencias externas o automatizar tareas de construcción.
- Ventajas:
  - Entorno fácil y rápido de configurar.
  - Ideal para personas que empiezan un lenguaje como JAVA o Kotlin.
- Desventajas:
  - Difícil de escalar a proyectos más grandes.
  - Carece de soporte para gestión avanzada de dependencias.

# Maven

Es un sistema de construcción basado en XML, facilita la gestión de proyectos y dependencias.

- Cuándo usarlo:
  - En proyectos de JAVA o Kotlin con muchas dependencias externas.
  - Desarrollo de aplicaciones empresariales (Spring Boot) o bibliotecas compartidas.
  - A la hora de construir un proyecto con comandos estandarizados como *mvn clean install*.
- Ventajas:
  - Gestión robusta de dependencias.
  - Amplia compatibilidad con proyectos de terceros.
  - Configuración estándar del archivo *pom.xml*.
- Desventajas:
  - Sintaxis más verbosa (se basa en XML).
  - Menos flexible en comparación con Gradle.



# Gradle

Es un sistema de construcción flexible basado en el lenguaje de scripting (Groovy y Kotlin).

- Cuando usarlo:
  - Para proyectos que requieren un sistema de construcción altamente personalizable.
  - Desarrollo de aplicaciones como Spring Boot, aplicaciones móviles en Android o microservicios.
  - Para construcción de un proyecto que requiera un mayor rendimiento.
- Ventajas:
  - Configuración más concisa en comparación con Maven.
  - Extensible y adaptable a necesidades específicas.
  - Compatible con proyectos grandes y complejos.
- Desventajas:
  - Curva de aprendizaje más pronunciada (en caso de no conocer Groovy o Kotlin).
  - Menor documentación a diferencia de Maven.



# ¿Cuál Elegir?

## IntelliJ

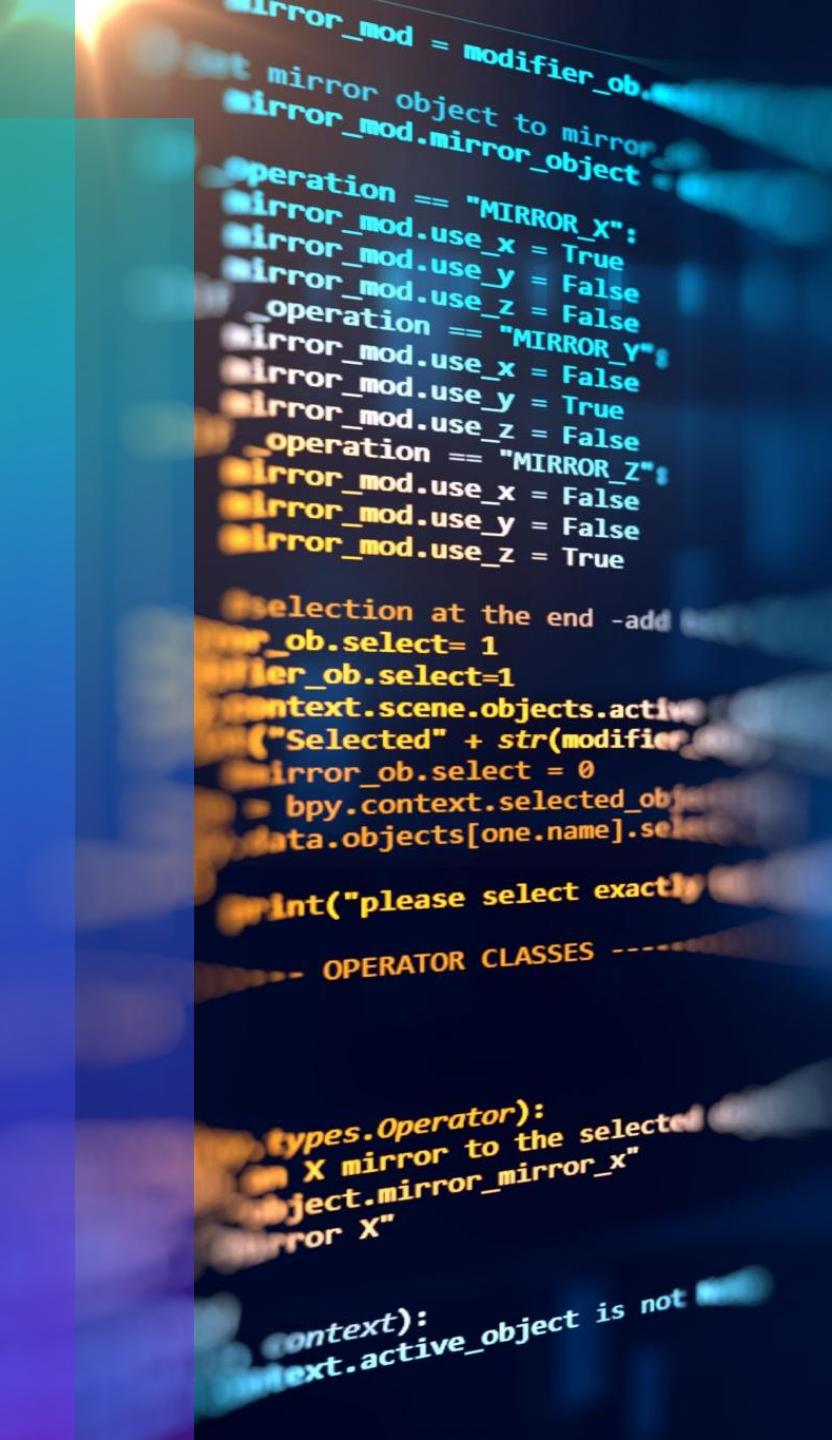
- Se utilizará para proyectos simples, donde no se requiera gestionar dependencias ni realizar configuraciones avanzadas.

## Maven

- Para proyectos empresariales o estándar (Java/Kotlin), se usa si se requiere compatibilidad con estándares establecidos y se prefiere un enfoque basado en XML.

## Gradle

- Para proyectos más modernos o complejos, en caso que se requiera flexibilidad y rendimiento; más si se trata de aplicaciones de gran escala o Android.





# Componentes del Lenguaje JAVA

¿En qué Consiste el Lenguaje?

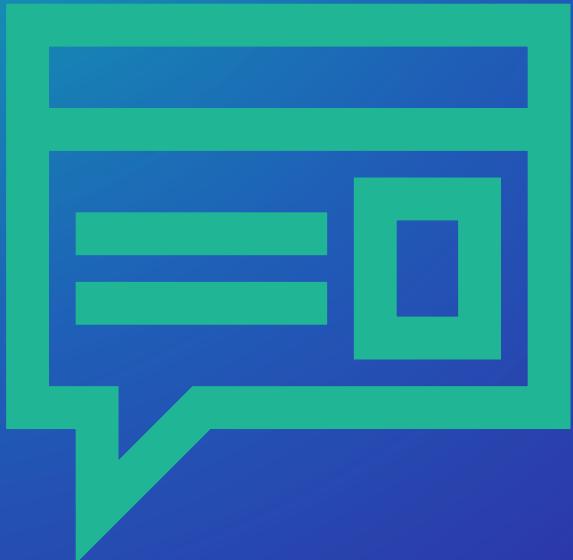
# Identificadores



Un identificador es un nombre que identifica a una variable, a un método o función miembro, a una clase. Todos los lenguajes tienen ciertas reglas para componer los identificadores:

1. Todos los identificadores han de comenzar con una letra, el carácter subrayado ( \_ ) o el carácter dollar ( \$ ).
2. Puede incluir, pero no comenzar por un número.
3. No puede incluir el carácter espacio en blanco.
4. Distingue entre letras mayúsculas y minúsculas.
5. No se pueden utilizar las palabras reservadas como identificadores.

# Comentarios



Un comentario es un texto adicional que se añade al código para explicar su funcionalidad, ya sea a otras personas que lean el programa, o al propio autor como recordatorio. Los comentarios son una parte importante de la documentación de un programa. Los comentarios son ignorados por el compilador, por lo que no incrementan el tamaño del archivo ejecutable; se pueden por tanto, añadir libremente al código para que pueda entenderse mejor.

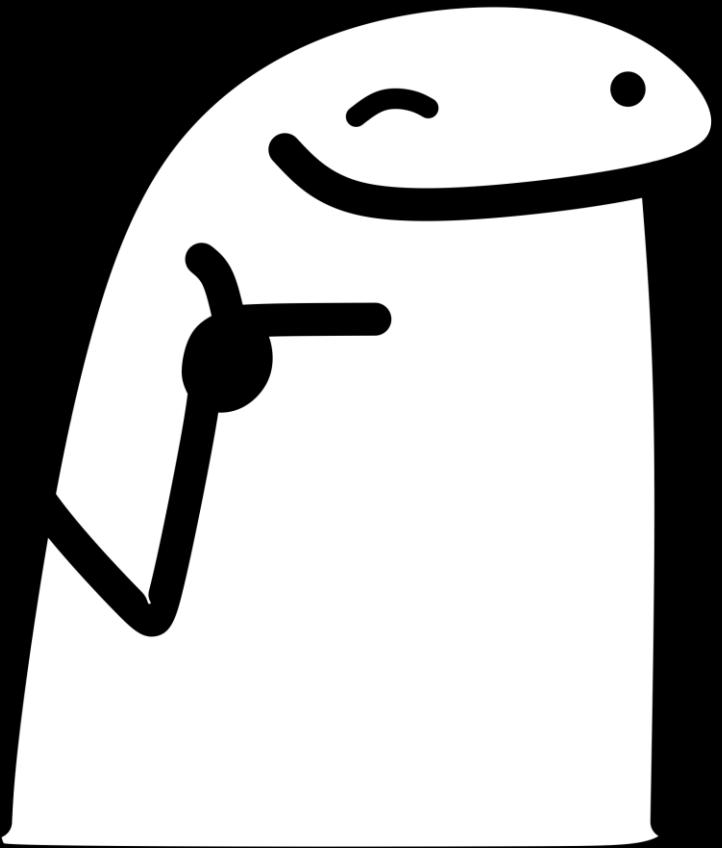
- Comentarios en una sola línea (//).
- Comentarios de varias líneas (/\*\*/).
- Comentarios de documentación, hacen uso de etiquetas.

# Sentencias

Una sentencia es una orden que se le da al programa para realizar una tarea específica, esta puede ser: mostrar un mensaje en la pantalla, declarar una variable (para reservar espacio en memoria), inicializarla, llamar a una función, etc. Las sentencias acaban con “;”. este carácter separa una sentencia de la siguiente. Normalmente, las sentencias se ponen unas debajo de otras, aunque sentencias cortas pueden colocarse en una misma línea.

# Bloques de Código

Un bloque de código es un grupo de sentencias que se comportan como una unidad. Un bloque de código está limitado por las llaves de apertura “{“ y cierre “}”. Como ejemplos de bloques de código tenemos la definición de una clase, la definición de una función miembro, una sentencia iterativa for, los bloques try ... catch, para el tratamiento de las excepciones, etc.



# Tipos de Variables

Ámbitos, Variables en JAVA,  
Valor y Referencia

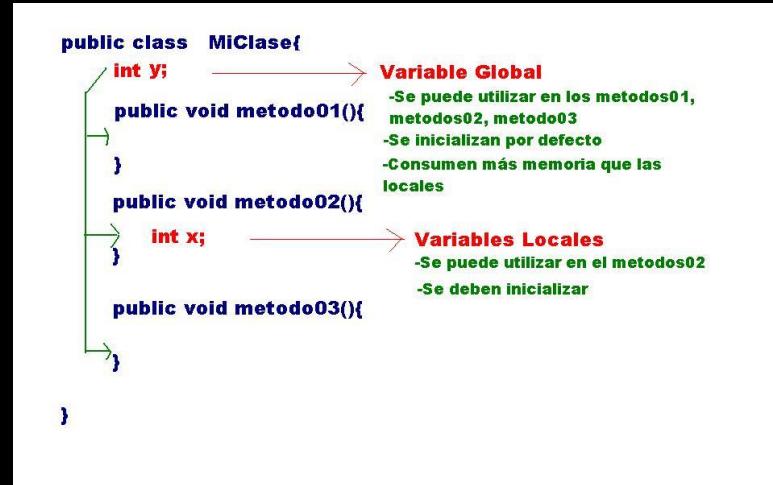
# Ámbito en Programación

En programación de computadoras, el ámbito (referido mediante su denominación en inglés “scope”) es el contexto que pertenece a un nombre dentro de un programa. El ámbito determina en qué partes del programa una entidad puede ser usada. Esto sirve para que se pueda volver a definir una variable con un mismo nombre en diferentes partes del programa sin que haya conflictos entre ellos.



# Variables Globales

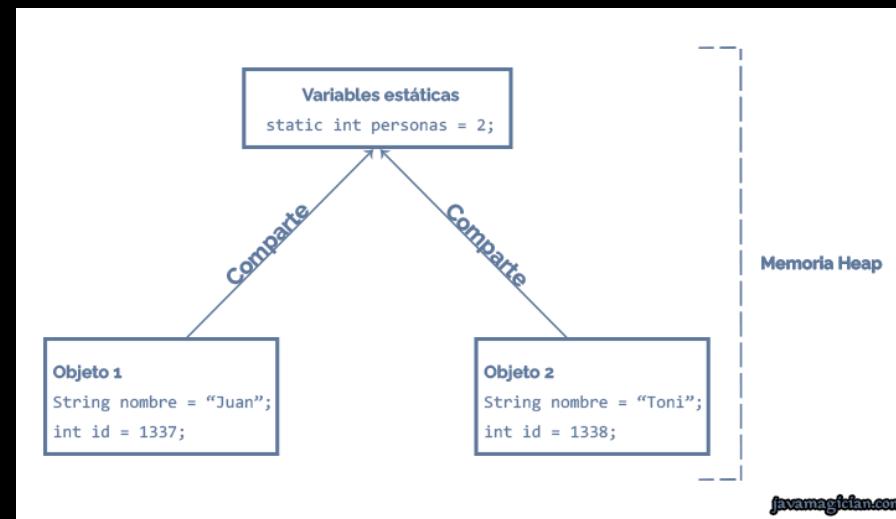
Una variable global es una variable accesible en todos los ámbitos de un programa informático. Los mecanismos de interacción con variables globales se denominan mecanismos de entorno global. El concepto de entorno global contrasta con el de entorno local donde todas las variables son locales sin memoria compartida (y por ello todas las iteraciones pueden restringirse al intercambio de mensajes).



# Variables Estáticas

## De Instancia

Una variable estática es una variable que ha sido ubicada estáticamente y cuyo tiempo de vida se extiende durante toda la ejecución del programa. Normalmente una variable estática tiene un ámbito más amplio que otras variables. Los valores de variables estáticas se pueden establecer una vez (durante el tiempo de ejecución) o se pueden cambiar en múltiples ocasiones durante la ejecución del programa. La terminología "variable estática" se basa en C y C++, pero también se usa en muchos lenguajes de programación derivados.



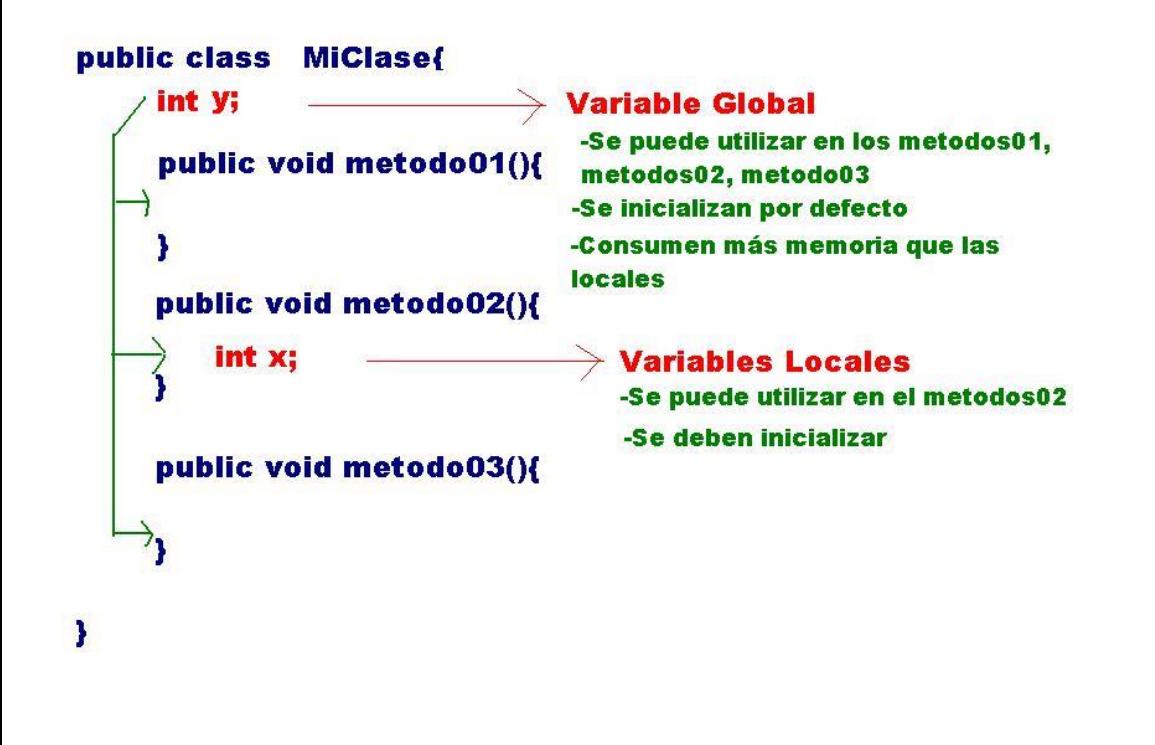
# Variables Finales

Una variable final se utiliza para definir una entidad que puede asignarse una única vez, y permanecerá siempre con el mismo valor. Si una variable final referencia a un objeto, los atributos de la instancia del objeto pueden cambiar, pero la variable final siempre hará referencia a la misma instancia.

```
class FinalClass
{
    final int Num = 5;

    class InheritedClass extends FinalClass
    {
        {
            ss.Num cannot be assigned
            Num = 6;
        }
    }
};
```

# Variables Locales



Una variable local es la variable a la que se le otorga un ámbito local. Tales variables sólo pueden accederse desde la función o bloque de instrucciones en donde se declaran. Las variables locales se contraponen a las variables globales.

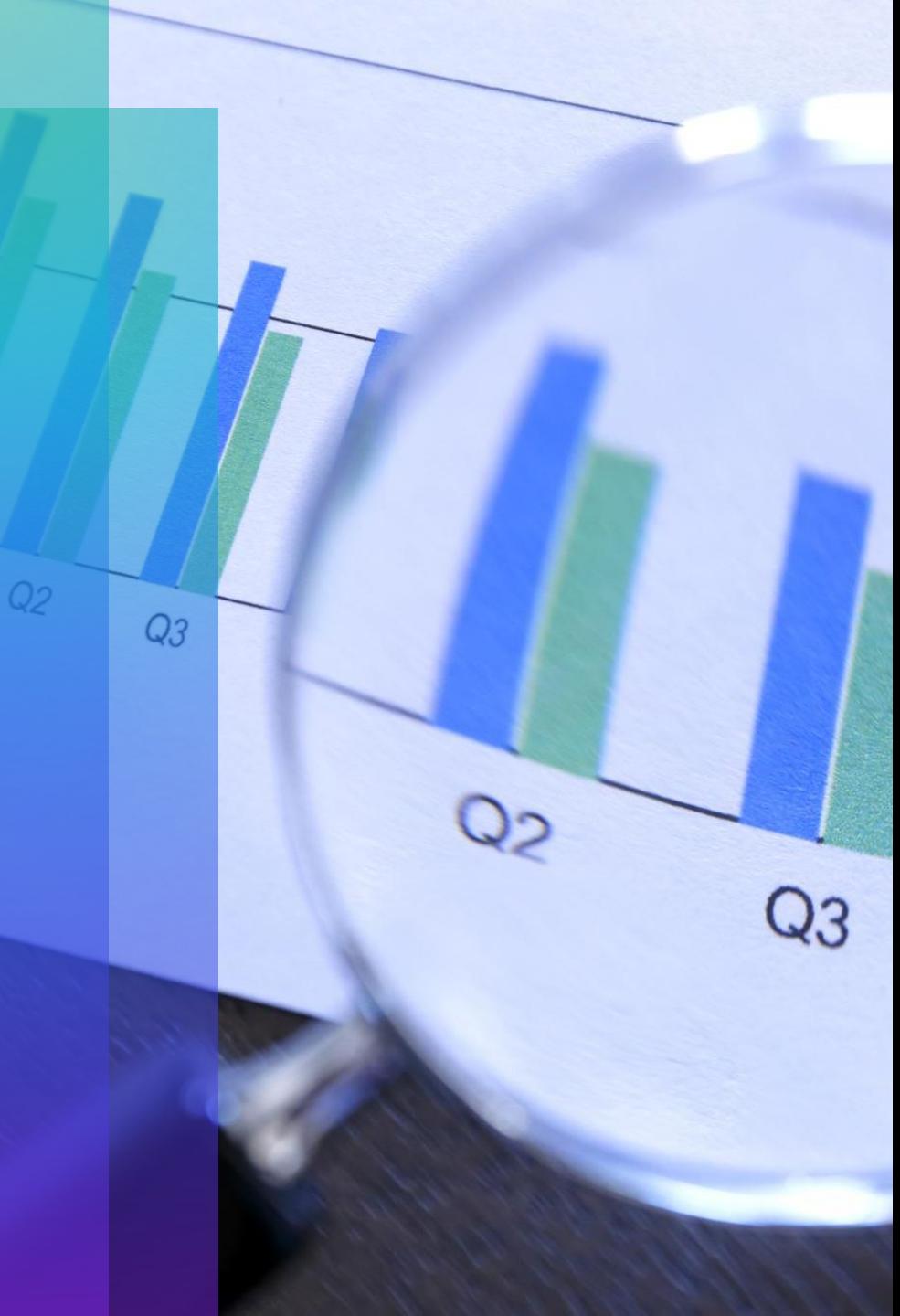
```
public class Counter {  
    ! usage  
    private volatile long counter = 1;  
  
    ! usage  
    void increment() {  
        for (int i = 0; i < Integer.MAX_VALUE; i++) {  
            counter++;  
        }  
    }  
}
```

# Variables Volátiles

Son variables que se marcan en Java para ser almacenadas en la memoria principal. Significa que cada lectura a una variable volátil será desde la memoria principal de la computadora, y no del caché del CPU, y toda escritura de una variable sucederá en la memoria principal de la computadora, y no en el caché del CPU. Garantiza que las variables volátiles sean escritas y leídas desde la memoria principal.

# Variables Trascendentes

Son variables se utilizan para evitar la serialización. Si una variable está definida como trascendente, esta no será serializada. Se llama "serializar un objeto" al proceso de convertirlo a bytes, para poder enviarlo por una red, y reconstruirlo luego a partir de esos bytes.



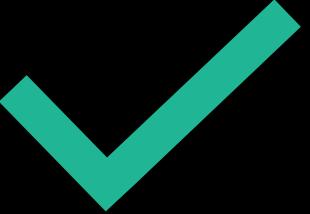
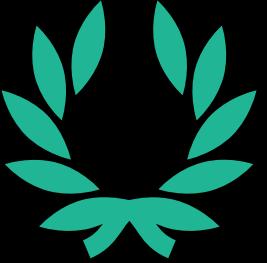
# Tipos Enumerativos

Tipo de dato que consiste en una serie de valores con nombres propios, llamados numerables. Los numerables suelen tener valores constantes, y se utilizan para tareas de comparación propia. Son declarados para optimizar tareas de identificación.



# Valor y Referencia

En Java, una variable adquiere su valor por medio de una asignación. Depende del tipo de la variable, el valor que adquiere la misma como una copia o una referencia hacia el valor obtenido. Los tipos de datos primitivos y las cadenas en Java se manejan por valor, mientras que los objetos se manejan por referencia.



## Asignación por Valor

```
Int a = 5;  
Int b = a;  
Print(b);  
• Resultado: >> 5
```

## Asignación por Referencia

```
int[] a = {1, 2, 3};  
int[] b = a;  
a[0] = 100;  
print(b);  
• Resultado: >> 100, 2, 3
```

# Ejemplo (Valor y Referencia)

# Expresiones y Variables

Una expresión es todo aquello que se puede poner a la derecha del operador asignación “=”.

Una variable es un nombre que se asocia con una porción de la memoria del ordenador, en la que se guarda el valor asignado a dicha variable. Hay varios tipos de variables que requieren distintas cantidades de memoria para guardar datos.

Todas las variables han de declararse antes de usarlas, la declaración consiste en una sentencia en la que figura el tipo de dato y el nombre que asignamos a la variable. Una vez declarada se le podrá asignar valores. Java tiene 3 tipos de variables:

De instancia  
(Globales)

De clase

Locales

# Declaración de Variables



Para declarar una variable en JAVA se utiliza la siguiente sintaxis:

TIPO\_DATO IDENTIFICADOR;

TIPO\_DATO IDENTIFICADOR = EXPRESION;

## Ejemplo

```
int valor1 = 50;
```

```
string nombre; //se inicializa con cadena vacía
```

```
string curso = "IPC 1";
```



# Asignación de Variables

Para asignar valores a una variable (ya declarada anteriormente) se utiliza la siguiente sintaxis:

**IDENTIFICADOR = EXPRESION;**

## Ejemplo

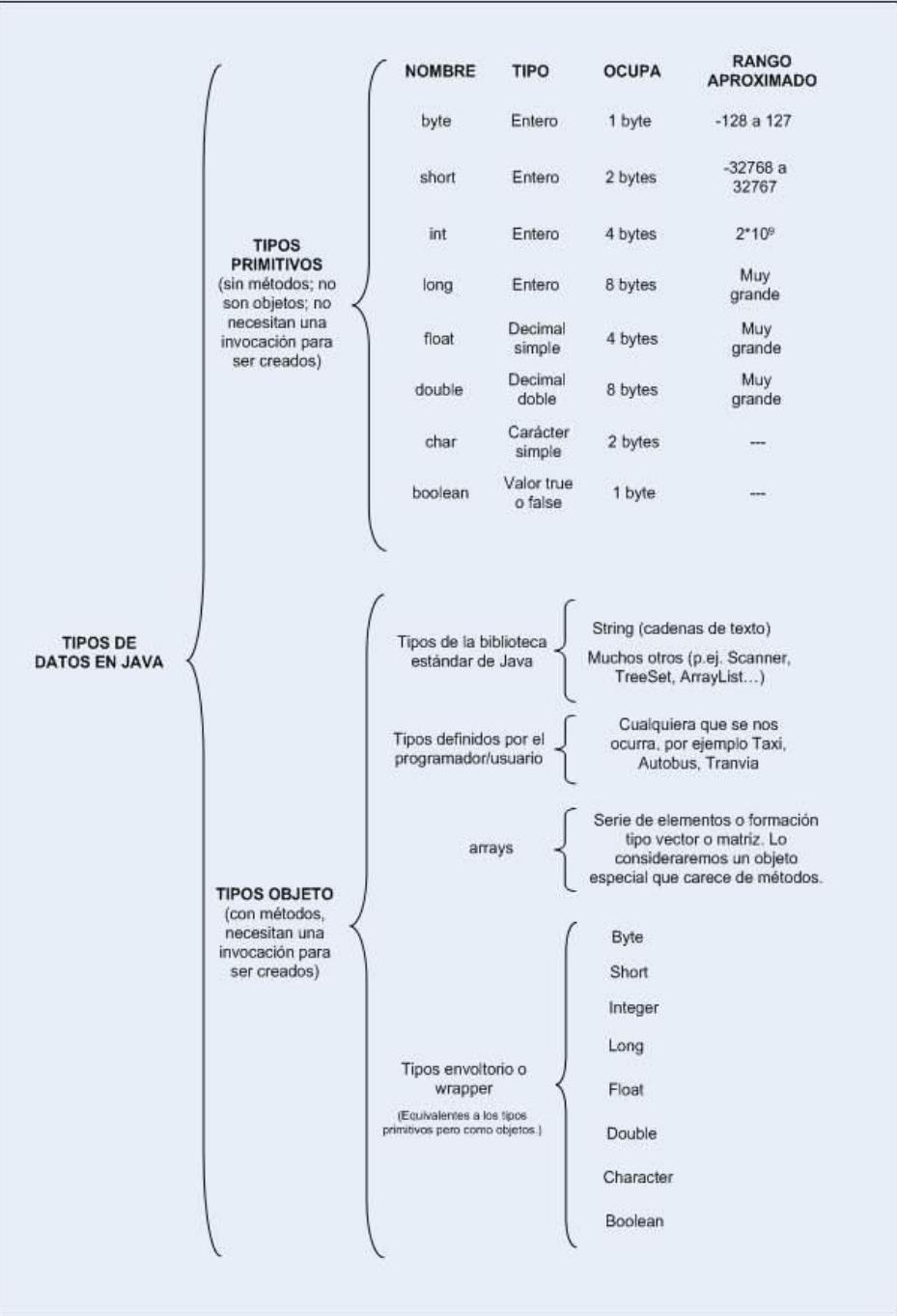
`int valor;`

`valor = 50;`

# Tipo de Dato

Nombre	Declaración	Memoria requerida	Intervalo	Descripción
Booleano	boolean	-	true - false	Define una bandera que puede tomar dos posibles valores: true o false.
Byte	byte	1 byte (8 bits)	[-128 .. 127]	Representación del número de menor rango con signo.
Entero pequeño	short	2 byte (16 bits)	[-32,768 .. 32,767]	Representación de un entero cuyo rango es pequeño.
Entero	int	4 byte (32 bits)	[-2 <sup>31</sup> .. 2 <sup>31</sup> -1]	Representación de un entero estándar. Este tipo puede representarse sin signo usando su clase <i>Integer</i> a partir de la Java SE 8.
Entero largo	long	8 byte (64 bits)	[-2 <sup>63</sup> .. 2 <sup>63</sup> -1]	Representación de un entero de rango ampliado. Este tipo puede representarse sin signo usando su clase <i>Long</i> a partir de la Java SE 8.
Real	float	4 byte (32 bits)	[±3,4·10 <sup>-38</sup> .. ±3,4·10 <sup>38</sup> ]	Representación de un real estándar. Recordar que al ser real, la precisión del dato contenido varía en función del tamaño del número: la precisión se amplia con números más próximos a 0 y disminuye cuanto más se aleja del mismo.
Real largo	double	8 byte (64 bits)	[±1,7·10 <sup>-308</sup> .. ±1,7·10 <sup>308</sup> ]	Representación de un real de mayor precisión. Double tiene el mismo efecto con la precisión que float.
Carácter	char	2 byte (16 bits)	['\u0000' .. '\uffff'] o [0 .. 65.535]	Carácter o símbolo. Para componer una cadena es preciso usar la clase <i>String</i> , no se puede hacer como tipo primitivo.

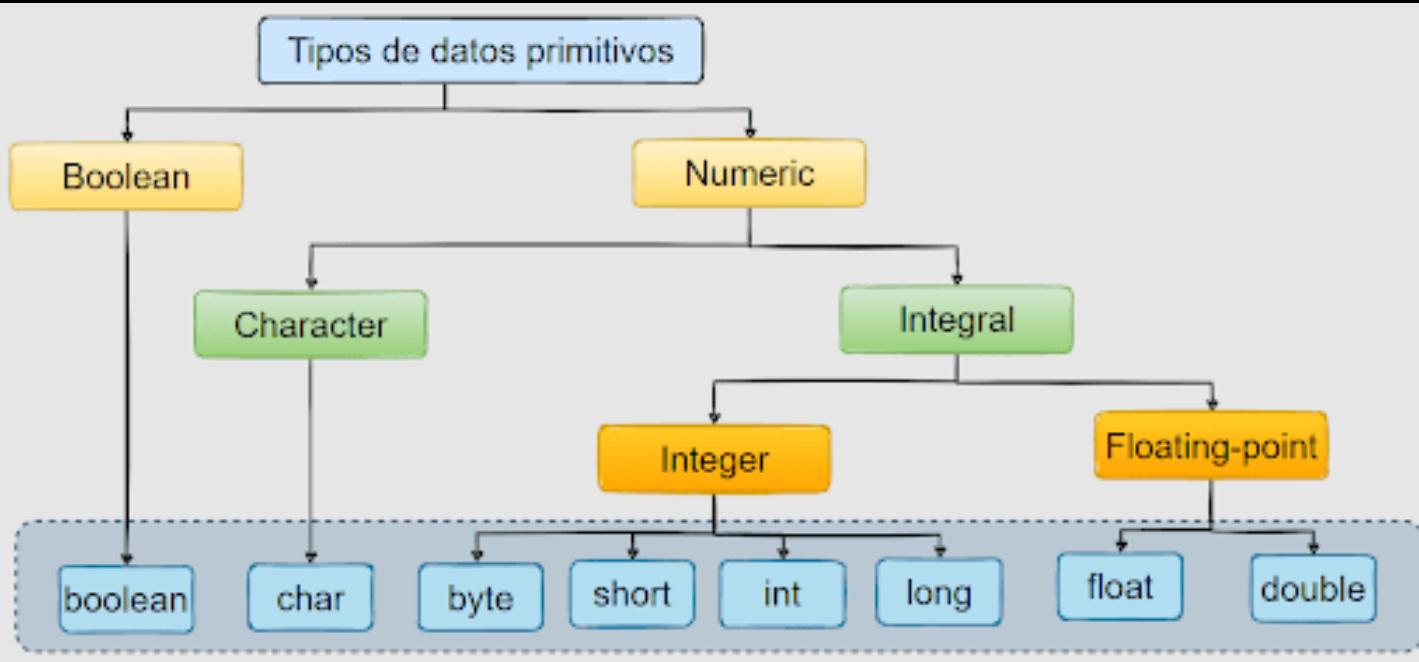
# Tipos de Datos



En programación, los tipos de datos son fundamentales para definir las características de las variables y cómo se almacenan, manipulan y procesan los datos en memoria. Se dividen en primitivos y no primitivos.

# Tipo de Datos Primitivos

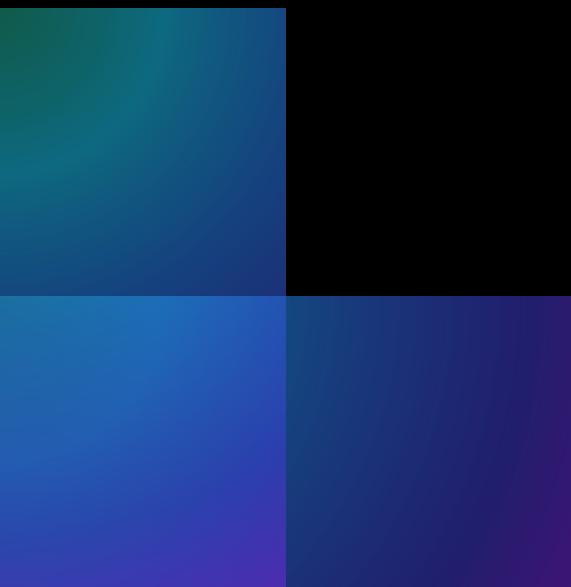
Descripción, Características, Clasificación (JAVA), Importancia



# Tipo de Datos Primitivos

Los tipos de datos primitivos son los más básicos y simples. Representan valores atómicos (no están compuestos por otros datos) y se manejan directamente por el lenguaje. Son inmutables (su valor no puede cambiar, aunque las variables que los referencian sí).

# Características



No tienen métodos ni propiedades.



Están definidos por el lenguaje de programación.



Tienen un tamaño fijo o predefinido.



Se almacenan en stack (memoria estática).

# Clasificación (JAVA)

## Numéricos Enteros

- Byte (1 byte): -128 a 127
- Short (2 bytes): -32,768 a 32,767
- Int (4 bytes): -2,147,483,648 a 2,147,483,647
- Long (8 bytes)

## Números con Punto Flotante (Decimales)

- Float (4 bytes)
- Double (8 bytes) // Se usa más para cálculos científicos/financieros

## Carácter

- Char (2 bytes): Caracteres Unicode

## Booleano

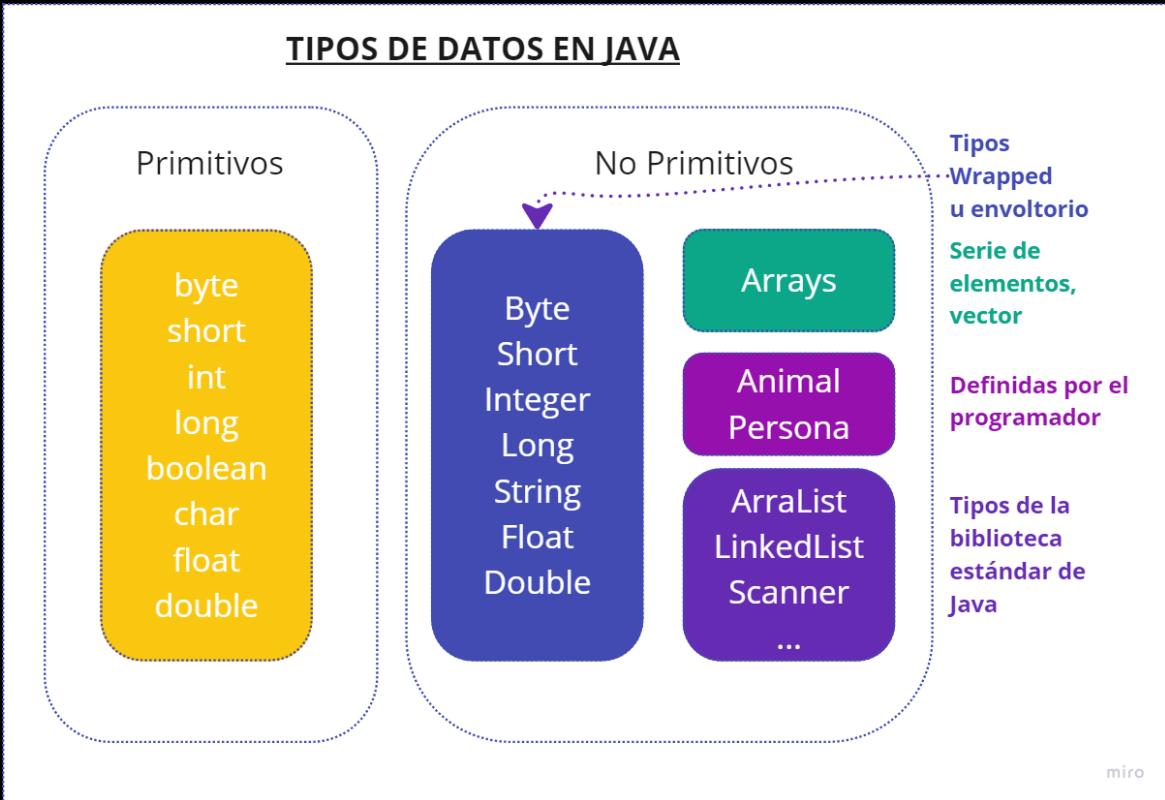
- Boolean (1 byte): true, false

# Tipos de Datos no Primitivos

Descripción, Características, Clasificación (JAVA)

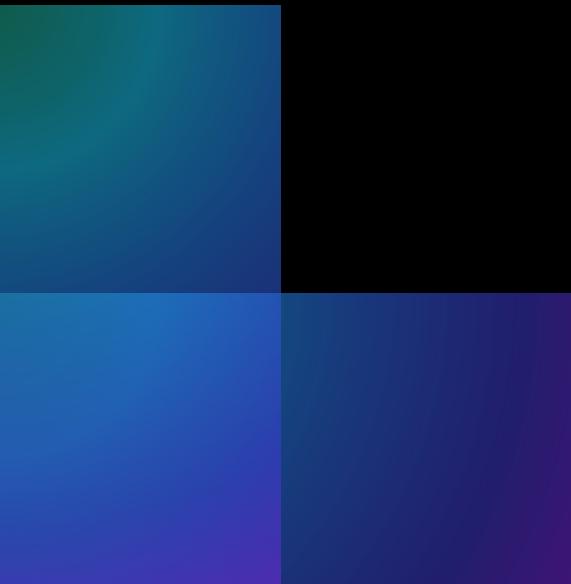


# Tipos de Datos no Primitivos



Los tipos de datos no primitivos (o referenciados) son aquellos que están definidos por el usuario o son estructuras más complejas que los primitivos. Incluyen objetos, arreglos y otros tipos avanzados.

# Características



Pueden tener métodos y propiedades.



Almacenan una referencia en la memoria heap (en lugar de datos directamente).

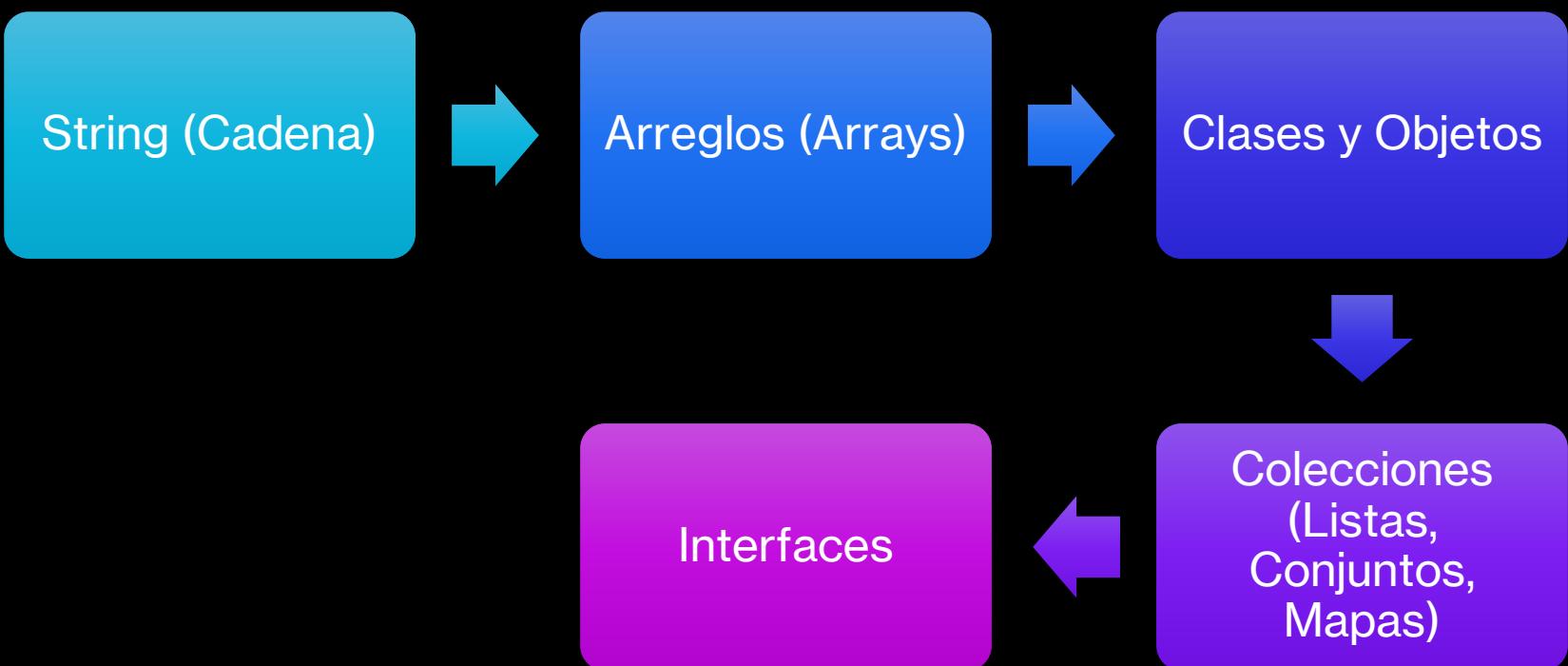


Incluyen estructuras más complejas, como clases, arreglos, interfaces, etc.



Se pueden personalizar (definir nuevos tipos mediante clases o interfaces).

# Clasificación (JAVA)



<b>Características</b>	<b>Primitivos</b>	<b>No Primitivos</b>
Almacenamiento	Memoria Stack	Memoria Heap
Inmutabilidad	Inmutables	Generalmente mutables
Métodos/Propiedades	No tienen	Tienen
Definición	Predefinidos por el lenguaje	Definidos por el usuario
Referencia/Valor	Contienen valores directos	Contienen referencias

# Tabla Primitivos & No Primitivos

# Importancia de los Tipos de Datos



## Uso eficiente de memoria

Escoger el tipo adecuado reduce el consumo de recursos.



## Evitar errores

Declarar tipos claros para evitar asignaciones indebidas (Ejemplo: almacenar un texto en una variable numérica).



## Lectura y mantenimiento del código

Un código con tipos claros es más fácil de entender y mantener.



## Compatibilidad y Operaciones

Permiten realizar operaciones seguras y válidas (sumar números, concatenar cadenas, etc.).

	<b>Byte</b>	<b>Short</b>	<b>Char</b>	<b>Int</b>	<b>Long</b>	<b>Float</b>	<b>Double</b>
<b>Byte</b>		Implícito	Char	Implícito	Implícito	Implícito	Implícito
<b>Short</b>	Byte		Char	Implícito	Implícito	Implícito	Implícito
<b>Char</b>	Byte	Short		Implícito	Implícito	Implícito	Implícito
<b>Int</b>	Byte	Short	Char		Implícito	Implícito	Implícito
<b>Long</b>	Byte	Short	Char	Int		Implícito	Implícito
<b>Float</b>	Byte	Short	Char	Int	Long		Implícito
<b>Double</b>	Byte	Short	Char	Int	Long	Float	

# Casteo

Consiste en la conversión de tipos de datos similares (compatibles) entre sí, generalmente a través de la herencia, en Java se pueden castear los siguientes tipos de datos (en filas) a los otros tipos de datos (columnas).

# Casteo implícito

El casteo implícito ocurre automáticamente cuando convertimos un tipo de dato de menor precisión (menor tamaño en memoria) a uno de mayor precisión (mayor tamaño en memoria). Este proceso no requiere intervención del programador porque no hay riesgo de pérdida de datos.

```
int valor;  
valor = 200;  
double conversion = valor; // Conversión implícita de int a double  
System.out.println(conversion); // Salida: 200.0
```

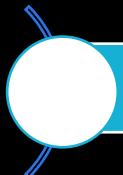
# Reglas (Casteo implícito)

Ocurre cuando no hay riesgo de pérdida de datos.

Ejemplos comunes:

- byte → short → int → long → float → double
- char → int

# Casteo explícito



tipoDestino variableDestino = (tipoDestino) variableOrigen;

El casteo explícito ocurre cuando convertimos un tipo de dato de mayor precisión a uno de menor precisión. Dado que esto puede implicar una pérdida de datos , el programador debe indicar explícitamente que desea realizar esta conversión.

```
double decimal = 54697.12035;
int entero = (int) decimal; // Conversión explícita de double a int
System.out.println(entero); // Salida: 54697
```

# Reglas (Casteo explícito)

Debe realizarse manualmente.

Puede provocar pérdida de datos.

Ejemplos comunes:

- double → float → long → int → short → byte
- int → char

<b>Característica</b>	<b>Casteo Implícito</b>	<b>Casteo Explícito</b>
Dirección	De menor a mayor precisión	De mayor a menor precisión
Riesgo de pérdida	No hay	Puede haber
Intervención	Automática	Manual

# **Casteo Implícito vs Explícito**

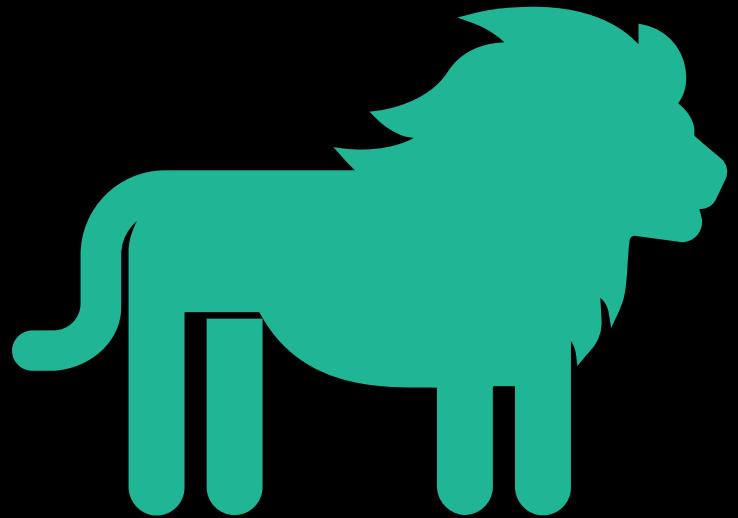
# Casteo entre tipos primitivos

De valores pequeños a más grandes (Implícito)

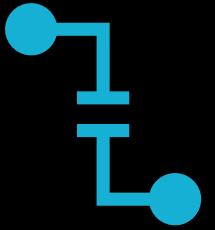
- int valor1 = 100;
- long largo = 100;
- double decimales = largo; // out >> 100.0

De valores más grandes a pequeños (Explícito)

- double decimales = 589.21135;
- int entero = (int) decimales; // out >> 123

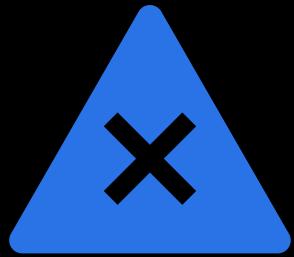


# Casteo de objetos en jerarquía de clases



**Upcasting**  
**subclase a superclase**

Automático  
Sin pérdida de información



**Downcasting**  
**superclase a subclase**

Debe hacerse de forma explícita  
Debe comprobarse el tipo para evitar errores en su ejecución

# Upcasting

```
1 < class Animal {  
2   < public void hacerRuido() {  
3     System.out.println("Soy un animal");  
4   }  
5 }  
6  
7 < class Gato extends Animal {  
8   < public void hacerRuido() {  
9     System.out.println("Miau");  
10   }  
11 }  
12  
13 < public class Main {  
14   < public static void main(String[] args) {  
15     Gato gatito = new Gato();  
16     Animal gato1 = gatito; // Upcasting implícito  
17     gato1.hacerRuido(); // out: Miau  
18   }  
19 }
```

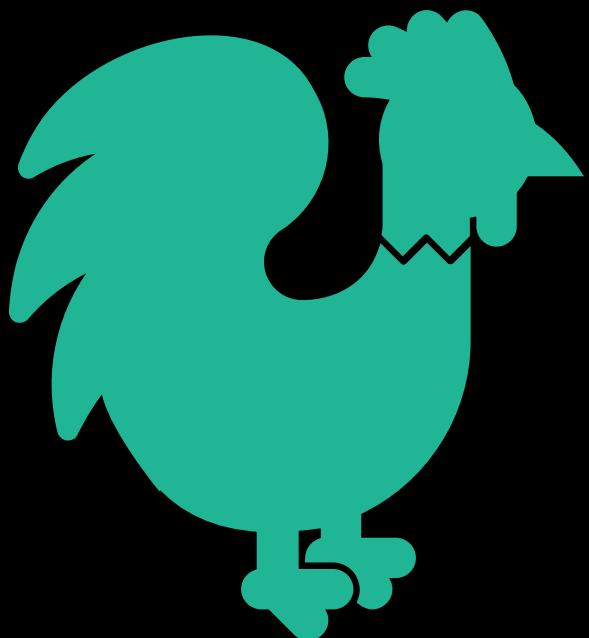


# Downcasting

```
1 ✓ class Animal {  
2   ✓   public void hacerRuido() {  
3     System.out.println("Soy un animal");  
4   }  
5 }  
6  
7 ✓ class Gato extends Animal {  
8   ✓   public void hacerRuido() {  
9     System.out.println("Miau");  
10  }  
11 }  
12  
13 ✓ public class Main {  
14   ✓   public static void main(String[] args) {  
15     Animal animal1 = new Gato(); // Upcasting implícito  
16   ✓   if (animal1 instanceof Gato) {  
17     Gato gato1 = (Gato) animal1; // Downcasting explícito  
18     gato1.hacerRuido(); // out: Miau  
19   }  
20 }  
21 }
```



# Casteo entre tipos de datos envueltos (Wrapper Classes)

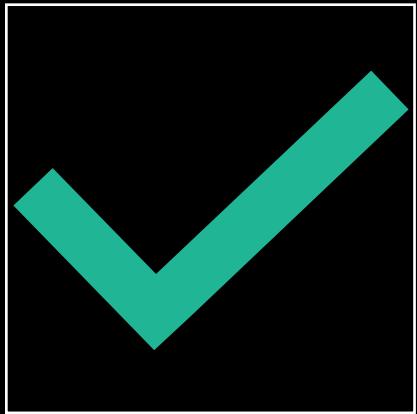


Java proporciona clases envolventes para los tipos primitivos, como Integer para int o Double para double.

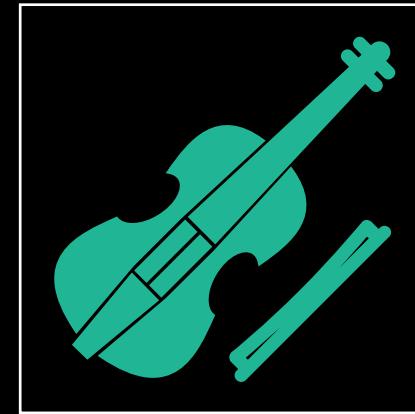
## Ejemplo

```
Integer entero = 50;  
int primitiveInt = entero; // Auto-unboxing  
// Conversión explícita  
double primitiveDecimal = entero.doubleValue();
```

# Casteo de cadenas a números y viceversa



String a tipo primitivo



Numero a string

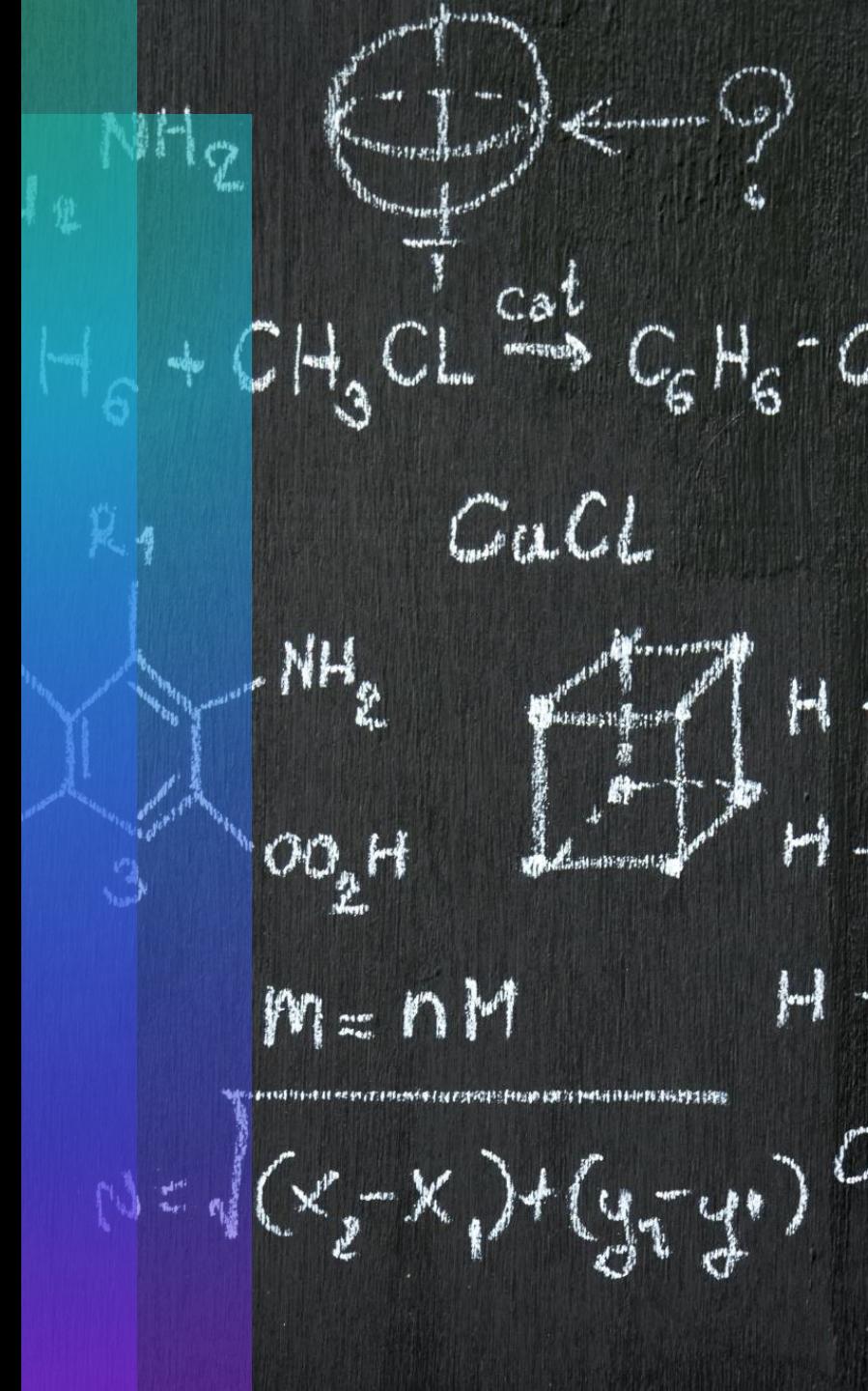


# String a tipo primitivo

```
String cadena = "5468";  
int entero = Integer.parseInt(cadena);  
>> 5468  
  
double decimal = Double.parseDouble(cadena);  
>> 5468.0
```

# Numero a string

```
int numero = 123;  
String valorcadena = Integer.toString(numero);  
>> "123"
```



# Casteo con interfaces

En caso de que una clase implemente una interfaz, se puede castear entre la interfaz y la clase.

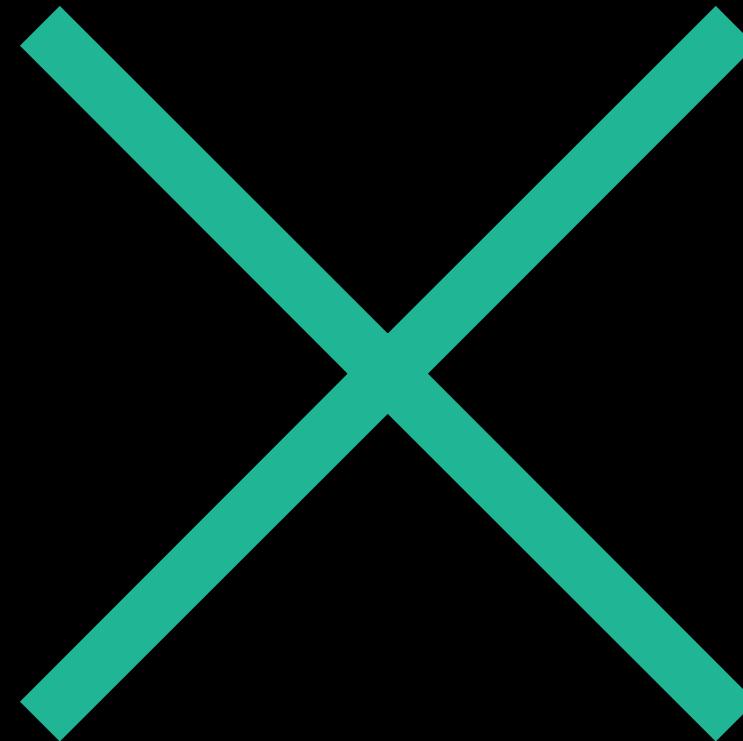
```
1 interface Mamifero {  
2     void sonido();  
3 }  
4  
5 class Gato implements Mamifero {  
6     public void sonido() {  
7         System.out.println("Miau");  
8     }  
9 }  
10  
11 public class Main {  
12     public static void main(String[] args) {  
13         Mamifero animal1 = new Gato(); // Upcasting  
14         Gato gato1 = (Gato) animal1; // Downcasting explícito  
15         gato1.sonido(); // out: Miau  
16     }  
17 }
```

# Casteo entre tipos no compatibles

En caso de que los tipos de datos no sean directamente compatibles, se deben usar métodos adicionales.

## Ejemplo

```
Object objeto1 = "456";  
// Downcasting y conversión  
int numero = Integer.parseInt((String) objeto1);
```

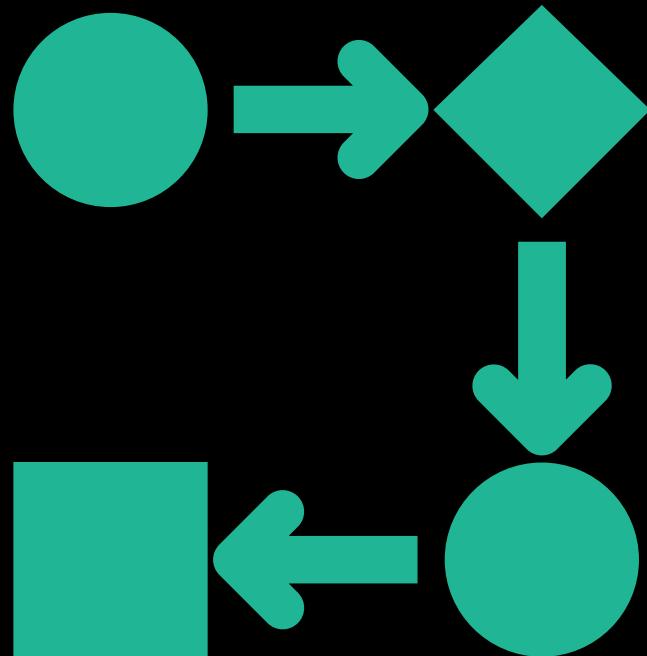


# Parseo

El método Parse no es simplemente el cambio del "tipo" , sino una adaptación y análisis sintáctico al nuevo tipo de dato deseado.

## Ejemplo

```
String cadena = "975";  
int entero = Integer.parseInt(cadena);
```



# Cadenas y Caracteres



Además de los ocho tipos de datos primitivos, las variables en Java pueden ser declaradas para guardar una instancia de una clase.



En Java los caracteres no están restringidos a los ASCII sino son Unicode. Un carácter está siempre rodeado de comillas simples como 'A', '9', 'ñ', etc. El tipo de dato char sirve para guardar estos caracteres.



Un tipo especial de carácter es la secuencia de escape, similares a las del lenguaje C/C++, que se utilizan para representar caracteres de control o caracteres que no se imprimen. Una secuencia de escape está formada por la barra invertida (\) y un carácter. En la siguiente tabla se dan las secuencias de escape más utilizadas.

# Palabras Reservadas

Las palabras reservadas se pueden clasificar en las siguientes categorías:

1. Tipos de datos: boolean, float, double, int, char.
2. Sentencias condicionales: if, else, switch.
3. Sentencias iterativas: for, do, while, continue.
4. Tratamiento de las excepciones: try, catch, finally, throw.
5. Estructura de datos: class, interface, implements, extends.
6. Modificadores y control de acceso: public, private, protected, transient.
7. Otras: super, null, this.

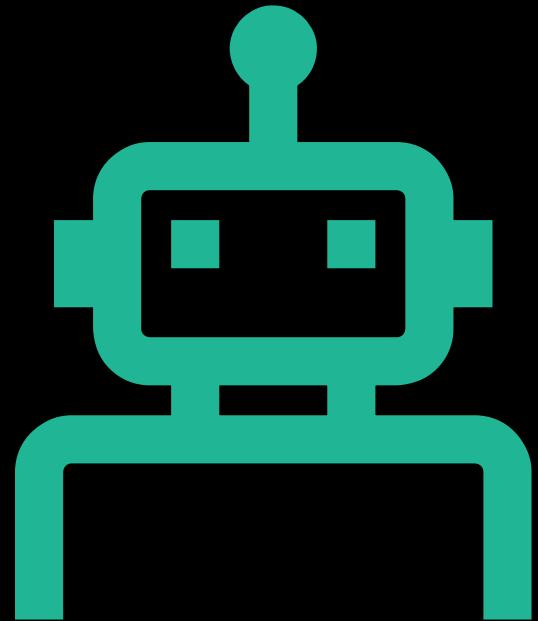
# Funciones y Procedimientos

Funciones, Procedimientos, Métodos



# Modularidad del Software

La modularidad es, en programación modular y más específicamente en programación orientada a objetos, la propiedad que permite subdividir una aplicación en partes más pequeñas (llamadas módulos), cada una de las cuales debe ser tan independiente como sea posible de la aplicación en sí y de las restantes partes. Estos módulos que se puedan compilar por separado, pero que tienen conexiones con otros módulos. Al igual que la encapsulación, los lenguajes soportan la Modularidad de diversas formas. La modularidad debe seguir los conceptos de acoplamiento y cohesión.



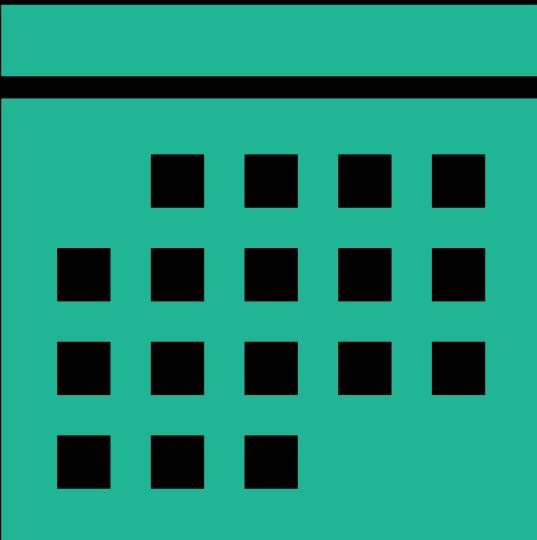
# Procedimientos

En informática, una subrutina o subprograma (también llamada procedimiento, función , rutina o método), como idea general, se presenta como un subalgoritmo que forma parte del algoritmo principal, el cual permite resolver una tarea específica. Algunos lenguajes de programación, como Visual Basic .NET o Fortran, utilizan el nombre función para referirse a subrutinas que devuelven un valor.



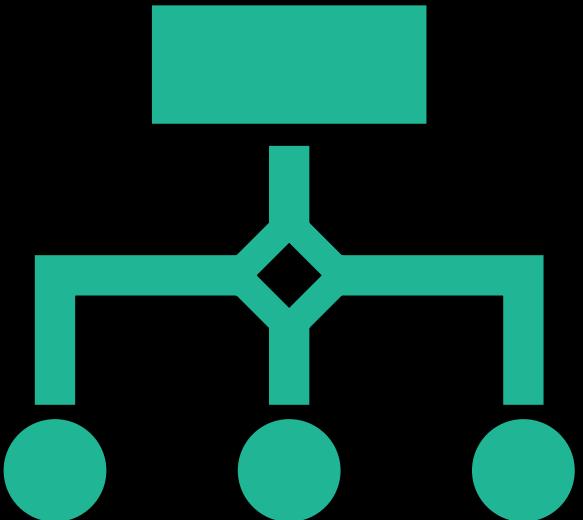
# Métodos

En la programación orientada a objetos, un método es una subrutina cuyo código es definido en una clase y puede pertenecer tanto a una clase, como es el caso de los métodos de clase o estáticos, como a un objeto, como es el caso de los métodos de instancia. Análogamente a los procedimientos en lenguajes imperativos, un método consiste generalmente de una serie de sentencias para llevar a cabo una acción, un juego de parámetros de entrada que regularán dicha acción.



# Funciones

En programación orientada a objetos, una función es un conjunto de líneas de código que realizan una tarea específica y retorna un valor. Las funciones pueden tomar parámetros que modifiquen su funcionamiento. Las funciones son utilizadas para descomponer grandes problemas en tareas simples y para implementar operaciones que son comúnmente utilizadas durante un programa y de esta manera reducir la cantidad de código. Cuando una función es invocada se le pasa el control a la misma, una vez que esta finalizó con su tarea el control es devuelto al punto desde el cual la función fue llamada.



# Dudas o Comentarios

