



Programación en Java
Inicio
Elementos de un programa informático
Programa y lenguajes de programación
El lenguaje Java
Variables, sentencias y paquetes
Tipos de datos
Operadores y expresiones
Comentarios en Java
Constantes y literales
Entrada y salida de información por consola
Introducción a la programación orientada a objetos
Estructuras básicas de control
@ Recursividad
Programación orientada a objetos POO
Arrays
POO avanzada
Colecciones
Programación avanzada
Interfaces de usuario
Entrada/Salida (I/O) de la información
Persistencia de la información
Atajos de teclado para IntelliJ
About me

## Elementos de un programa informático

Un programa es una secuencia de instrucciones que un ordenador ejecuta para realizar alguna tarea. Parece una idea bastante simple, pero para que el ordenador pueda hacer uso de las instrucciones, deben estar escritas de forma que las pueda usar. Esto significa que los programas deben estar escritos en lenguajes de programación. Los lenguajes de programación se diferencian de los lenguajes humanos ordinarios en que son completamente inequívocos y muy estrictos sobre lo que está y no está permitido en un programa. Las reglas que determinan lo que está permitido se denominan \*\*sintaxis del lenguaje\*\*. Las reglas de sintaxis especifican el vocabulario básico del lenguaje y cómo se pueden construir los programas.

## Lenguajes de programación

Se trata de un conjunto de instrucciones que permite la comunicación de los humanos con los ordenadores.

Aug 2021	Aug 2020	Change	Programming Language	Ratings	Change
1	1		C	12.57%	-4.41%
2	3	▲	Python	11.86%	+2.17%
3	2	▼	Java	10.43%	-4.00%
4	4		C++	7.36%	+0.52%
5	5		C#	5.14%	+0.46%

Sep 2022	Sep 2021	Change	Programming Language
1	2	▲	Python
2	1	▼	C
3	3		Java
4	4		C++
5	5		C#

1. C --> bases de datos, videojuegos, kernel linux, IOT, dispositivos inteligentes, etc.
2. Python --> inteligencia artificial, big data, etc.
3. Java --> puntos de ventas, aplicaciones de escritorio, cajeros automáticos, dispositivos móviles, ...



## ¿Con qué lenguaje de programación debo empezar?

En realidad, no existe el mejor lenguaje de programación para empezar, y el lenguaje que elijas prácticamente no tendrá un efecto duradero en tu carrera. Los lenguajes de programación pueden verse muy diferentes en la superficie, pero la mayoría de los conceptos fundamentales se transferirán de un lenguaje a otro. Aprender un nuevo idioma también será más fácil con el tiempo; un principiante puede tardar meses antes de que se sienta cómodo con su primer lenguaje; un programador experimentado puede familiarizarse con un nuevo idioma en cuestión de días.

## 7 Tips para aprender a programar con éxito

1. Learn by doing. Always play with the code while learning
2. Grasp the fundamentals for long-term benefits
3. Code by hand. It sharpens proficiency and you'll need it to get a job
4. Ask for help. You'll need it
5. Seek out more online resources. There's a wealth of content
6. Don't just read the sample code. Tinker with it!
7. Take breaks when debugging

Fuente: Coding dojo

## Tabla de contenidos

Lenguajes de programación

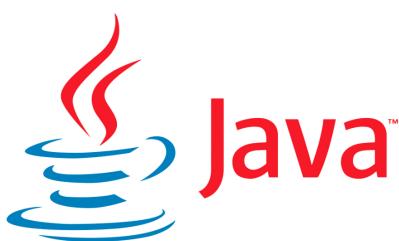
¿Con qué lenguaje de programación debo empezar?

7 Tips para aprender a programar con éxito

Sé persistente, no te rindas!

Programación en Java
Inicio
Elementos de un programa informático
Programa y lenguajes de programación
El lenguaje Java
Conceptos básicos del lenguaje
Setup
ProyectoHola Mundo
Variables, sentencias y paquetes
Tipos de datos
Operadores y expresiones
Comentarios en Java
Constantes y literales
Entrada y salida de información por consola
Introducción a la programación orientada a objetos
Estructuras básicas de control
@ Recursividad
Programación orientada a objetos POO
Arrays
POO avanzada
Colecciones
Programación avanzada
Interfaces de usuario
Entrada/Salida (I/O) de la información
Persistencia de la información
Atajos de teclado para IntelliJ
About me

## ¿Qué es JAVA ?



### Historia de Java

Java es un lenguaje de programación orientado a objetos. Fue creado por James Gosling en 1995. Oracle lo compró. Es un lenguaje independiente de la plataforma donde va a ser ejecutado, esto significa, que una vez que se compila (.class también llamado bytecode) puede ser transportado a diferentes plataformas (Windows, MacOS, Linux) para ser ejecutado. ¿Cómo se hace esto? Utilizando la JVM (Java Virtual Machine). La JVM coge los bytecode compilados y los interpreta dependiendo del OS para poder ser ejecutado. Por tanto, el bytecode siempre es el mismo, pero lo que genera la JVM cambiará para cada SO. También existen otros términos muy famosos relacionados con Java como JRE y JDK.

### Características de Java

- **Portabilidad:** Java es altamente portable y puede ejecutarse en diferentes sistemas operativos sin necesidad de modificaciones. Esto se debe a la Máquina Virtual de Java (JVM), que interpreta el código de byte Java en la plataforma de destino.
- **Orientación a Objetos:** Java es un lenguaje orientado a objetos, lo que significa que se basa en la creación y manipulación de objetos. Esto facilita la organización y reutilización del código.
- **Sintaxis Clara y Legible:** La sintaxis de Java se basa en gran medida en la de C++, pero se ha simplificado para mejorar la claridad y la legibilidad del código. Esto hace que Java sea más fácil de aprender y usar.
- **Administración Automática de Memoria:** Java utiliza un recolector de basura (garbage collector) para administrar la memoria automáticamente, lo que evita problemas comunes de administración de memoria, como fugas de memoria.
- **Seguridad:** Java se centra en la seguridad y utiliza un sistema de permisos y políticas de seguridad para proteger los sistemas contra código malicioso.
- **Multihilo (Multithreading):** Java admite la programación multihilo, lo que permite que las aplicaciones realicen múltiples tareas simultáneamente y mejoren la eficiencia.
- **Bibliotecas Estándar Ricas:** Java cuenta con una amplia biblioteca estándar que proporciona herramientas y funciones listas para usar, lo que acelera el desarrollo de aplicaciones.
- **Plataforma Independiente:** Java es una plataforma independiente, lo que significa que las aplicaciones Java pueden ejecutarse en cualquier sistema que tenga una JVM compatible.
- **Compatibilidad Hacia Atrás:** Java se preocupa por la compatibilidad hacia atrás, lo que significa que las versiones más nuevas del lenguaje generalmente son compatibles con el código existente.
- **Comunidad Activa:** Java cuenta con una comunidad de desarrollo activa y una amplia base de usuarios. Esto resulta en abundantes recursos y soporte en línea.
- **Escalabilidad:** Java es adecuado para proyectos de todos los tamaños, desde aplicaciones móviles hasta sistemas empresariales a gran escala.
- **Desarrollo Web y Empresarial:** Java se utiliza ampliamente en el desarrollo web, especialmente en aplicaciones empresariales y servidores web.

### ¿Por qué aprender Java?

- Nació en 1995 y todavía sigue siendo uno de los lenguajes más usados a nivel mundial.
- Su capacidad de escribirlo y ejecutarlo en cualquier plataforma: "Write once, run anywhere".
- Desde que nació han salido muchas releases de gran importancia, el lenguaje sigue evolucionando con nuevas funcionalidades.
- Tiene una comunidad muy grande de código abierto, documentación y tutoriales. Es usado por muchas compañías de software, por tanto se ofrecen muchos empleos de desarrolladores Java.

### Aplicaciones de Java

Java es ampliamente utilizado en una amplia gama de aplicaciones, como aplicaciones de banca en línea, aplicaciones de comercio electrónico, sistemas de gestión de bases de datos, sistemas de control de vuelo, aplicaciones de telefonía móvil y muchos más.

### Usos de Java

Java se utiliza en una variedad de dominios, incluyendo:

- Desarrollo de aplicaciones web.
- Aplicaciones móviles para Android.
- Aplicaciones de escritorio.
- Desarrollo de servidores empresariales.
- Sistemas embebidos.
- Aplicaciones científicas y financieras.
- Juegos y entretenimiento.

### Ventajas de Java

- Portabilidad.
- Lenguaje orientado a objetos

### Tabla de contenidos

Historia de Java
Características de Java
¿Por qué aprender Java?
Aplicaciones de Java
Usos de Java
Ventajas de Java
Importancia de Java
Principios de Java
Paradigma de programación Java
¿Cómo funciona Java?
Plataforma Java
Java Virtual Machine (JVM)
Sintaxis de Java
Otros conceptos relacionados con Java
JRE
JDK
Versiones Java

## Conceptos orientados a objetos.

- Seguridad.
- Facilidad de uso.
- Bibliotecas estándar ricas.
- Comunidad activa.
- Escalabilidad.

## Importancia de Java

Java es importante en la informática porque permite el desarrollo de aplicaciones y sistemas altamente funcionales y escalables. Su capacidad de ejecutarse en múltiples plataformas lo convierte en una herramienta valiosa para la creación de software en una amplia variedad de campos.

## Principios de Java

Los principios fundamentales de Java incluyen la portabilidad, la seguridad y la facilidad de uso. Java se basa en una sintaxis clara y una filosofía de "escribir una vez, ejecutar en cualquier lugar".

## Paradigma de programación Java

Java se basa en un paradigma de programación orientado a objetos. Esto significa que los programas Java están compuestos por objetos que interactúan entre sí. El enfoque orientado a objetos permite una organización y reutilización eficiente del código.

## ¿Cómo funciona Java?

Java funciona a través de un proceso de compilación e interpretación. El código fuente Java se compila en un archivo de bytecode, que luego se ejecuta en la Máquina Virtual de Java (JVM). La JVM se encarga de la ejecución del código y garantiza la portabilidad entre plataformas.

## Plataforma Java

La Plataforma Java es un conjunto de tecnologías que incluye el lenguaje de programación Java, la JVM y las bibliotecas estándar. La Plataforma Java proporciona un entorno completo para el desarrollo y la ejecución de aplicaciones Java.

## Java Virtual Machine (JVM)

La JVM es una parte fundamental de Java, ya que interpreta y ejecuta el bytecode generado por el compilador Java. La JVM garantiza la portabilidad del código, permitiendo que las aplicaciones Java se ejecuten en diferentes sistemas operativos.

## Sintaxis de Java

La sintaxis de Java se caracteriza por su claridad y legibilidad. El código Java utiliza reglas y estructuras específicas que facilitan la escritura y comprensión del código. Esto incluye la declaración de variables, estructuras de control, definición de clases y más.

## Otros conceptos relacionados con Java

### JRE

**Java Runtime Environment.** La máquina virtual de Java está incluida en el JRE. Se utiliza para ejecutar aplicaciones Java en el SO, por tanto si queremos desplegar una aplicación en Java, necesitamos tener instalado previamente el JRE para esa máquina y plataforma.

### JDK

**Java Development Kit.** Es más pesado que el JRE, contiene todas las herramientas para programar y compilar las clases Java en bytecode, por tanto cuando desarrollamos clases Java necesitamos el JDK. Además incluy el JRE.

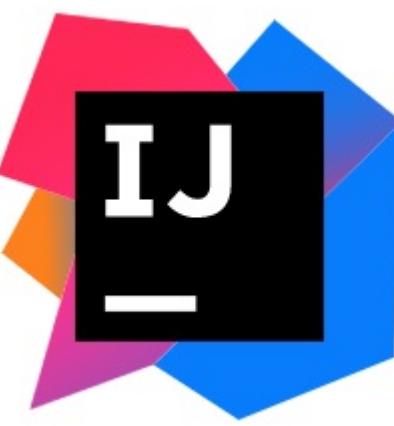
## Versiones Java

La última versión de Java se conoce como LTS Release (Long Term Support), y la podemos encontrar en su [página Oficial](#).

Programación en Java
Inicio
Elementos de un programa informático
Programa y lenguajes de programación
El lenguaje Java
Conceptos básicos del lenguaje
Setup
Proyecto Hola Mundo
Variables, sentencias y paquetes
Tipos de datos
Operadores y expresiones
Comentarios en Java
Constantes y literales
Entrada y salida de información por consola
Introducción a la programación orientada a objetos
Estructuras básicas de control
⊗ Recursividad
Programación orientada a objetos POO
Arrays
POO avanzada
Colecciones
Programación avanzada
Interfaces de usuario
Entrada/Salida (I/O) de la información
Persistencia de la información
Atajos de teclado para IntelliJ
About me

## ¿Qué se necesita para programar en Java?

- Instalar el JDK la última release, versión estable de Java. Lo podemos descargar desde la página oficial de Oracle.
- Instalar un IDE (Integrated Development Environment), es un programa que te ayudar a desarrollar aplicaciones. Hay muchas opciones disponibles de IDEs.



## Setup JDK Java y IntelliJ en Windows

1. Vamos a la página de Oracle y descargamos el JDK.
2. Instalamos el JDK.

En linux usamos el siguiente comando para instalar el jdk: sudo apt-get install openjdk-XX-jdk, donde XX es la versión del java.

1. Windows: Añadimos en las variables de entorno del sistema dentro de la variable Path, la ruta donde se ha instalado java en nuestra máquina (C:\Program Files\Java\jdk-VERSION\bin)
2. Windows: Agregamos también una nueva variable llamada JAVA\_HOME --> C:\Program Files\Java\jdk-VERSION (Ruta de nuestra máquina).
3. Verificamos que se ha instalado java, ejecutando desde línea de comandos:

```
java --version
```

1. Descargamos e instalamos IntelliJ Community

2. En File -> Settings, modificamos las siguientes opciones:

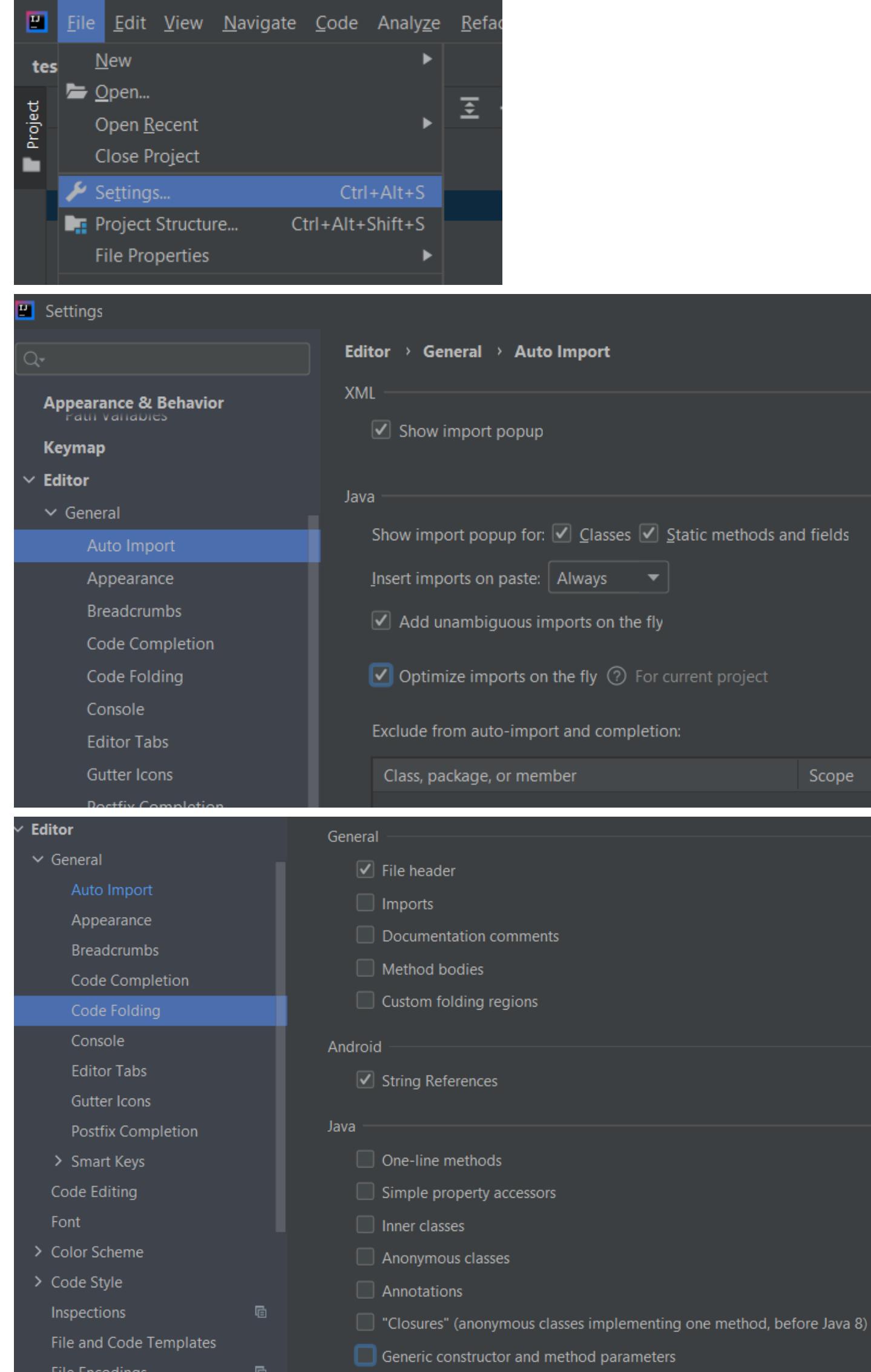


Tabla de contenidos
Setup JDK Java y IntelliJ en Windows



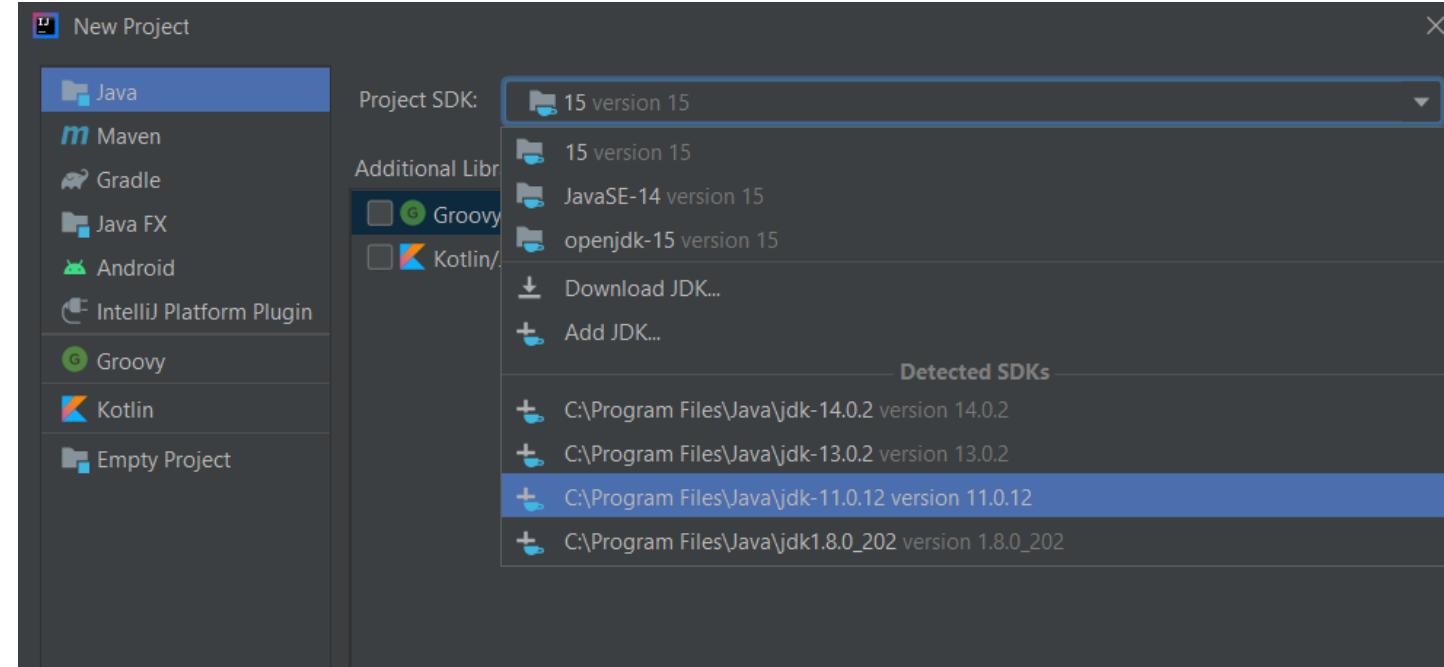
Programación en Java
Inicio
Elementos de un programa informático
Programa y lenguajes de programación
El lenguaje Java
Conceptos básicos del lenguaje
Setup
Proyecto Hola Mundo
Variables, sentencias y paquetes
Tipos de datos
Operadores y expresiones
Comentarios en Java
Constantes y literales
Entrada y salida de información por consola
Introducción a la programación orientada a objetos
Estructuras básicas de control
@ Recursividad
Programación orientada a objetos POO
Arrays
POO avanzada
Colecciones
Programación avanzada
Interfaces de usuario
Entrada/Salida (I/O) de la información
Persistencia de la información
Atajos de teclado para IntelliJ
About me

## Proyecto "Hola Mundo". Estructura de un programa

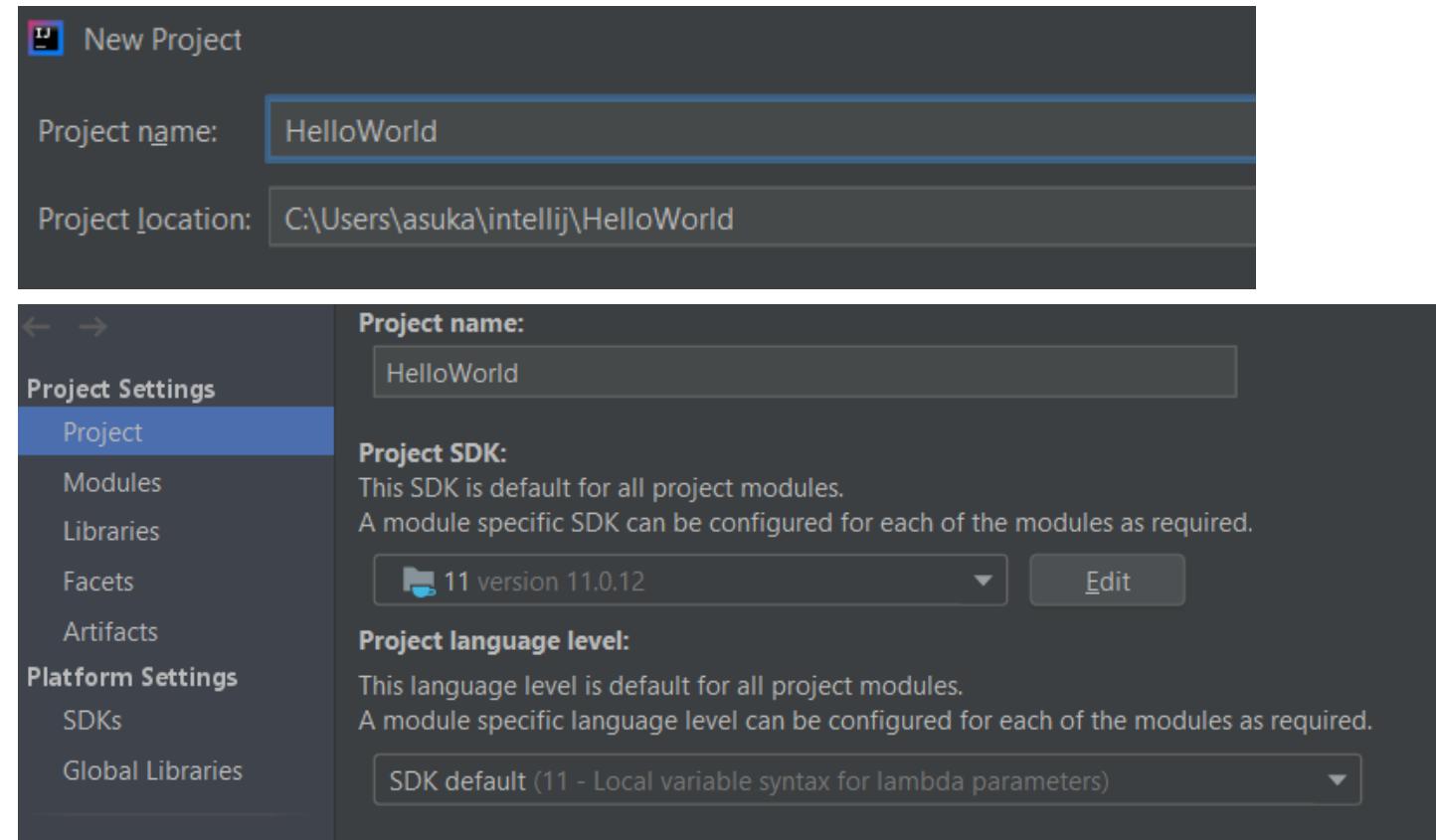
Tabla de contenidos  
Challenge

Cada vez que estás aprendiendo un nuevo lenguaje de programación es una *tradición* crear un nuevo programa muy simple que genera el texto **Hola mundo**. Veamos como podemos hacer esto en IntelliJ.

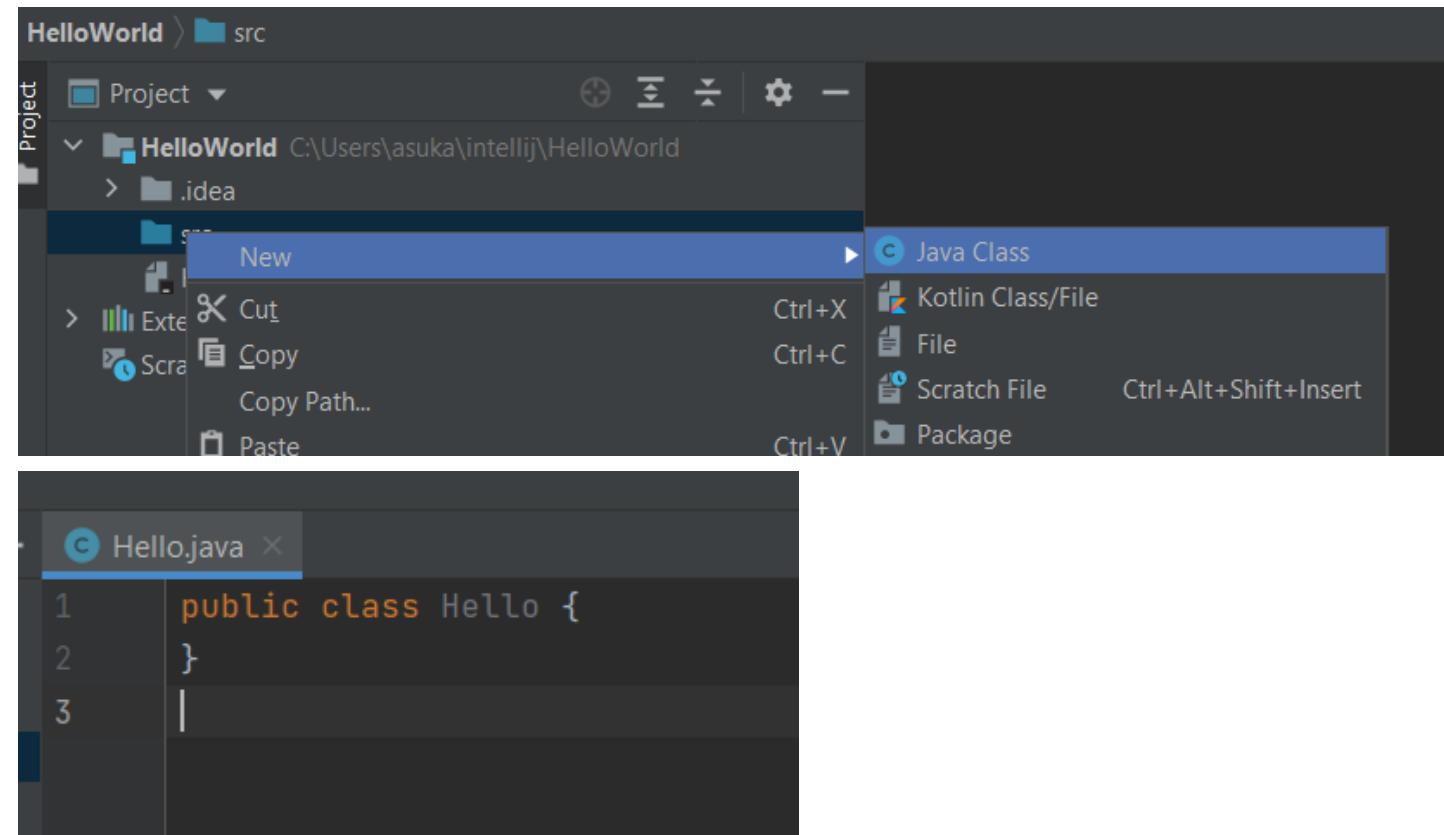
1. Hacemos click en crear nuevo proyecto y elegimos el JDK que hemos instalado.



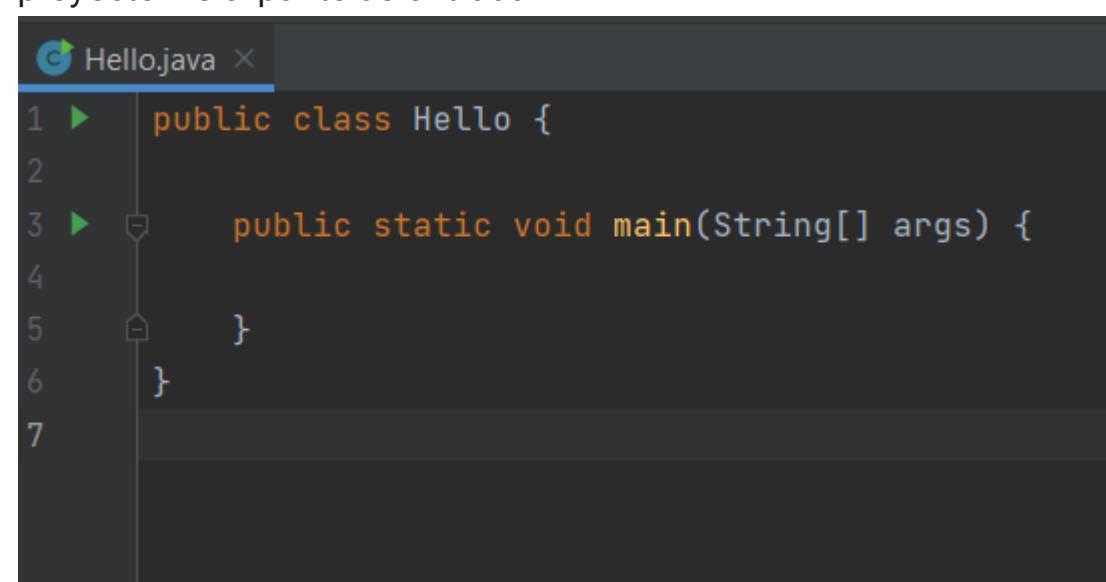
2. Luego en Next, Next, añadimos nombre al proyecto y Finish.



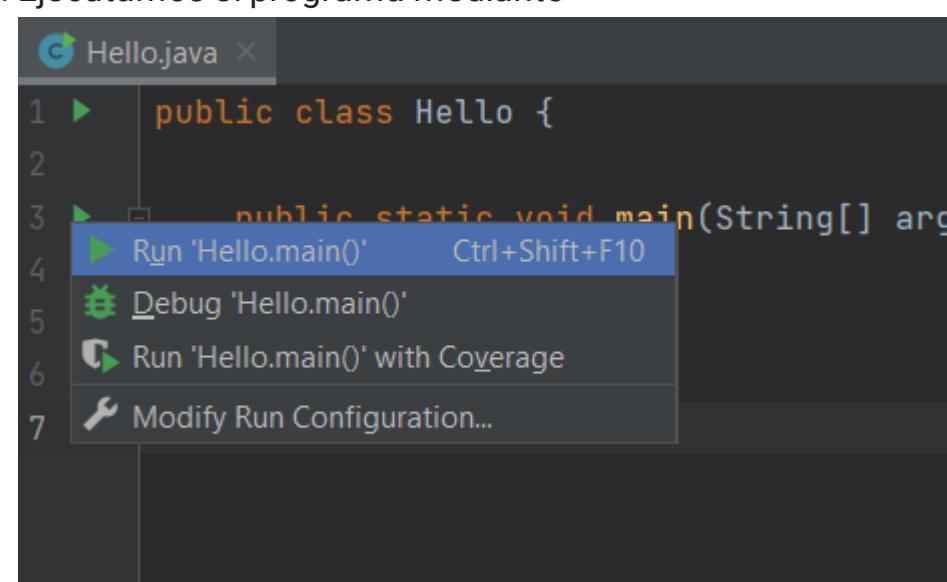
3. Creamos una nueva clase Java llamada Hello.java



4. Vamos a crear un método para imprimir el mensaje *Hola mundo*. Usaremos un método especial llamado **main**. El main es el primer método que Java busca y ejecuta en el proyecto. Es el punto de entrada.



5. Ejecutamos el programa mediante



6. Añadimos y ejecutamos la siguiente sentencia dentro del método main:

```
System.out.println("Hello World");
```

The screenshot shows a Java development environment. The top window, titled 'Hello.java', contains the following code:

```
1 public class Hello {  
2     public static void main(String[] args) {  
3         System.out.println("Hello World");  
4     }  
5 }  
6  
7
```

The bottom window, titled 'Hello', shows the terminal output:

```
"C:\Program Files\Java\jdk-11.0.12\bin  
Hello World  
Process finished with exit code 0
```

## Challenge

### Question

Modifica el programa para que imprima por pantalla *Hello Teacher*.

### Question

Realizar las actividades 1 y 2.

Programación en Java
Inicio
Elementos de un programa informático
Programa y lenguajes de programación
El lenguaje Java
Variables, sentencias y paquetes
Variables
Sentencias y paquetes
Tipos de datos
Operadores y expresiones
Comentarios en Java
Constantes y literales
Entrada y salida de información por consola
Introducción a la programación orientada a objetos
Estructuras básicas de control
@ Recursividad
Programación orientada a objetos POO
Arrays
POO avanzada
Colecciones
Programación avanzada
Interfaces de usuario
Entrada/Salida (I/O) de la información
Persistencia de la información
Atajos de teclado para IntelliJ
About me

## ¿Qué son las variables?

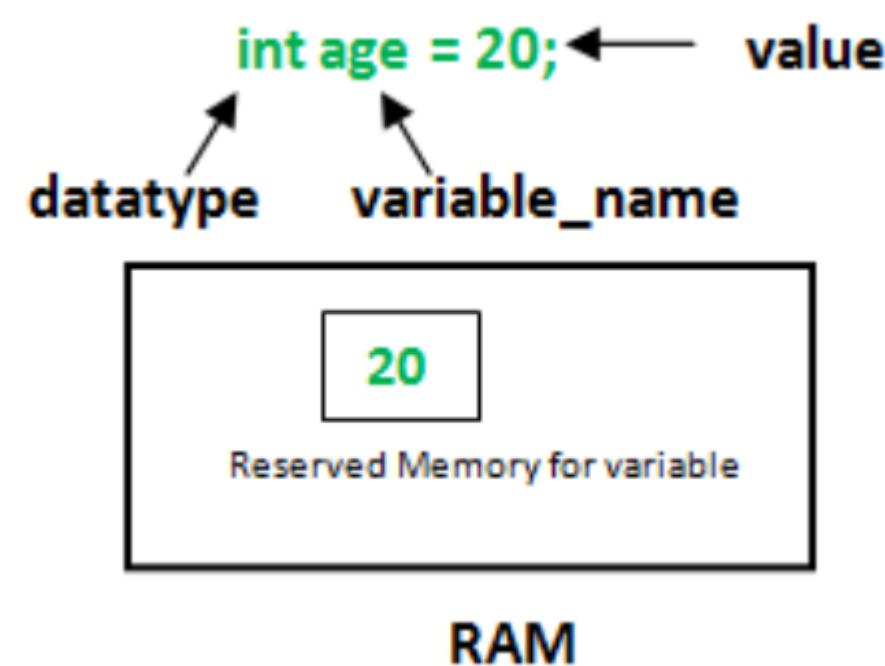
Las variables en Java es una forma de almacenar información en nuestro ordenador. Definimos variables asignándoles un nombre. De igual manera se puede acceder a la información que guardan las variables, simplemente accediendo por el nombre que le hemos dado. Es el ordenador el que se encarga de averiguar dónde se almacena internamente en la memoria RAM del ordenador.

Como su nombre indica "variable", se puede modificar el contenido que ésta almacena, es decir, es variable. Lo único que tenemos que hacer es decirle al PC qué tipo de información queremos guardar en nuestra variable y darle un nombre. Existe diferentes tipos de información que podemos utilizar para definir nuestras variables. Se les conoce como **tipos de datos** o data types. Los tipos de datos son palabras reservadas **keywords** en Java, es decir, no podemos utilizarlas fuera del contexto de tipo de datos.

Para definir una variable, necesitamos especificar el tipo de datos, luego darle un nombre a nuestra variable, y opcionalmente, podemos agregar una expresión para inicializar la variable con un valor.

### Reglas para el nombrado de variables

- Tiene que comenzar con una letra o '\_', nunca con números.
- Puede contener números. No debe contener espacios en blanco.
- No debe ser muy largo y debe expresar algo en el contexto.
- No se pueden usar palabras reservadas.
- Mayúsculas y minúsculas se tratan diferente.



Definimos nuestra primera variable en el programa de la siguiente forma:

```

1  public class Hello {
2
3      public static void main(String[] args) {
4          System.out.println("Hello World");
5
6          int miPrimerEntero = 7;
7      }
8
9

```

La línea que acabamos de escribir se conoce como **sentencia de declaración**.

### Keywords

Son palabras reservadas del lenguaje. Es decir, son palabras que tienen un significado especial en Java y no puedes usarlas fuera de ese contexto. Es decir, **no se puede declarar variables utilizando palabras reservadas como nombre**. Ejemplos: public, class, void, static, etc. Existen 61 **keywords** en Java.

### Vida de las variables

Las variables son memorias reservadas para almacenar valores en RAM. Estas posiciones de memoria se liberan tan pronto como termina la vida de la variable. Según la vida de las variables, hay tres tipos de Variables.

1. **Variables locales:** La vida permanece dentro de un bloque donde se ha declarado.
2. **Variables de instancia:** Declaradas dentro de la clase pero fuera de los métodos. No debería ser estático.
3. **Variables estáticas:** es como una variable global. Declarado como estático en la clase pero fuera de los métodos.

```
static int pi = 3.14;
```

### Visibilidad de las variables

En las variables locales, su ámbito y uso se encuentra dentro del método o bloque en el que se definió y se destruyen después de la ejecución del método. Es decir, no se puede usar una variable local fuera del método actual.

A las variables de instancia se puede acceder solo a través de objetos de la clase para la que se definió.

Un campo / variable estático pertenece a la clase y se cargará en la memoria junto con la clase.

Tabla de contenidos
Reglas para el nombrado de variables
Keywords
Vida de las variables
Visibilidad de las variables
Mutación o manipulación de las variables
Creación de una variable
Inicialización de las variables
Scope - Ámbito de las variables
Buenas prácticas con variables

Se invocan sin crear un objeto. (Usando el nombre de la clase como referencia). Solo hay una

copia de la variable estática disponible en toda la clase, es decir, el valor de la variable estática será el mismo en todos los objetos. Puede definir una variable estática utilizando la palabra clave **static**.

## Mutación o manipulación de las variables

Supongamos que hemos ejecutado la siguiente línea de código:

```
int number = 5;
```

¿Cómo podríamos sumar 6 al valor actualmente almacenado en `number`? Un enfoque ingenuo podría ser probar esta línea de código:

```
number + 6;
```

Sin embargo, esta línea de código es una expresión que da como resultado un valor: **no hemos alterado el valor de `number`**.

```
// Recuerda, number es una variable que contiene el valor 5
number + 6; // se evalúa como:
5 + 6; // y luego da como resultado:
11;

// Pero 11; no es una declaración que Java entienda,
// entonces el compilador lanza un error cuando ve: number + 6;
```

Para aumentar el valor de `number` en 6, necesitamos reasignar el valor de `number` para que sea el resultado de `number + 6`:

```
number = number + 6; // se evalúa como:
number = 5 + 6; // y luego se suman los valores
number = 11; // como resultado se asigna el valor 11 a number
```

Aquí, hemos usado el valor de `number` para calcular y almacenar un nuevo valor en la variable `number`; en este caso, 11.

## Creación de una variable

Para crear una variable se especifica el tipo de dato y se le da un nombre descriptivo que de información sobre esa variable. Ejemplo de creación de una variable:

```
int numero;
```

## Inicialización de las variables

La variable anterior `numero` no tiene un valor inicial asignado, es decir, no está inicializada. Inicializar una variable significa darle un valor inicial acorde al tipo de dato definido para esa variable en el momento de su creación. En Java además, las variables deben ser inicializadas antes de poder ser usadas.

Ejemplo:

```
int numero = 99; //Se crea y se inicializa
```

## Scope - Ámbito de las variables

El alcance o ámbito (**scope**) de una variable es la parte de un programa en la que existe. En Java, el alcance de una variable comienza donde se declara y termina cuando se alcanza la llave de cierre del bloque que la contiene.

Ejemplo de variables en programación Java

```
public static void main(String[] args) {
    int valor = 5;
    for (int i = 1; i <= 5; i++) {
        int y = 10;
        System.out.println(valor) // valor todavía está dentro del alcance aquí!
    }
    System.out.println(valor) // valor todavía está dentro del alcance aquí también
}
```

- `valor` está dentro del alcance entre su declaración en la línea 2 y la llave que la encierra en la línea 8.
- `y` está dentro del alcance entre su declaración en la línea 4 y la llave que la encierra en la línea 6.
- Las variables de bucle están dentro del alcance entre sus bucles `for {}`. Entonces, `i` está dentro del alcance entre las líneas 3 - 6. Nota: Dos variables con el mismo nombre no pueden existir dentro del mismo ámbito (scope).

## Buenas prácticas con variables

- **Utiliza nombres descriptivos** que reflejen el propósito de la variable. Evita nombres genéricos como "x" o "temp". Un nombre descriptivo como "numeroDeEstudiantes" es más claro y evita futuros comentarios en código. Recuerda que un código muy comentado significa que está mal escrito. Y un código bien escrito tendrá muy pocos comentarios, solo cosas significativas para aclarar código complejo.
- **Evita el uso de abreviaturas confusas** que puedan confundir a otros desarrolladores. Usa nombres completos y legibles en su lugar.
- **Convenio de nombres**. Sigue un convenio de nombres consistente, como CamelCase o snake\_case, según las convenciones de estilo de tu lenguaje de programación.
- **No uses palabras reservadas** del lenguaje como nombres de variables, ya que puede causar conflictos y errores.

- **Manten buen alcance (scope)**, limita el alcance de las variables al mínimo necesario. Declarar variables en el ámbito más cercano a su uso mejora la legibilidad y reduce la posibilidad de errores.
- **Inicialización oportuna**, inicializa las variables en el momento de la declaración o antes de su primer uso. Evita dejar variables sin inicializar, ya que puede causar comportamientos inesperados.
- **Comentarios significativos**. Agrega comentarios relevantes cuando el propósito de una variable no sea obvio. Esto ayuda a otros programadores a entender tu código.
- **Evita variables globales**. Limita el uso de variables globales, ya que pueden causar problemas de mantenimiento y depuración. Usa variables locales siempre que sea posible.
- **Evita variables reutilizadas** para diferentes propósitos en diferentes partes de tu código. Esto puede causar confusión y errores.
- **Usa constantes para valores fijos**. Si un valor es constante y no debe cambiar, décláralo como una constante en lugar de una variable.
- **Evita magia numérica**, es decir, evita usar valores numéricos sin explicación directa en el código. Usa constantes descriptivas o comenta el propósito de los números mágicos.
- **Refactoriza nombres cuando sea necesario**. Si el propósito de una variable cambia, renómbrala para reflejar el nuevo propósito en lugar de reutilizarla.
- **Evita variables muertas**. Elimina variables que ya no se utilizan en el código. Las variables inactivas dificultan la lectura y pueden llevar a confusiones.
- **No crear variables demasiado largas**. Si una variable tiene un nombre excesivamente largo, puede ser difícil de leer. Encuentra un equilibrio entre la claridad y la concisión.

Programación en Java
Inicio
Elementos de un programa informático
Programa y lenguajes de programación
El lenguaje Java
Variables, sentencias y paquetes
Variables
Sentencias y paquetes
Tipos de datos
Operadores y expresiones
Comentarios en Java
Constantes y literales
Entrada y salida de información por consola
Introducción a la programación orientada a objetos
Estructuras básicas de control
@ Recursividad
Programación orientada a objetos POO
Arrays
POO avanzada
Colecciones
Programación avanzada
Interfaces de usuario
Entrada/Salida (I/O) de la información
Persistencia de la información
Atajos de teclado para IntelliJ
About me

## Sentencias

Hemos comentado que un programa es un conjunto de instrucciones. Estas instrucciones se llaman sentencias o statements en inglés. Una sentencia es un segmento de código que realiza una acción en el programa. A medida que se ejecuta un programa, decimos que ejecuta sentencias, lo que significa que lleva a cabo las acciones especificadas por esas sentencias. En nuestro programa Hello World, tenemos un statement en la **Línea 4 y 6**. La regla en Java es que las sentencias deben terminar con un punto y coma. Si se olvida, se produciría un error de sintaxis.

### Sentencias de declaración

Se usan para definir una variable de un tipo de dato en particular. En Java, una variable debe declararse antes de que pueda usarse en un programa. De no hacerlo, se produciría un error de sintaxis. En su forma más simple una declaración incluye el tipo de datos y el nombre de la variable. Opcionalmente se puede establecer la variable a cierto valor. Es decir, se dice que se ha inicializado.

```
Ejemplos:  
int numero;  
int a = 3; //se crea la variable a y se le asigna el valor 3  
int dia;
```

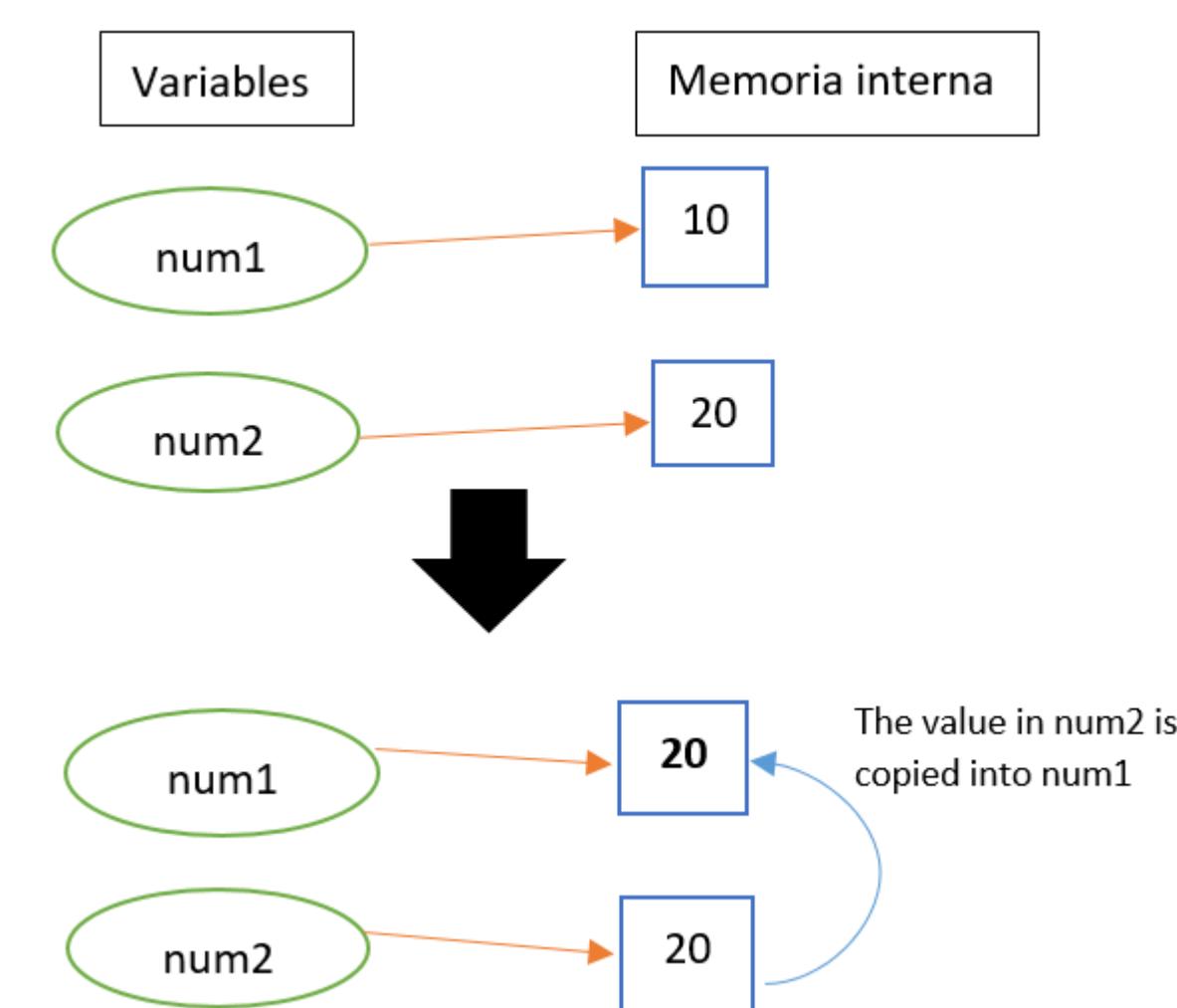
### Sentencias de asignación

Una sentencia de asignación es una sentencia que almacena (asigna) un valor en una variable. Una sentencia de asignación utiliza el signo igual (=) como operador de asignación. En su forma más simple, tiene una variable en el lado izquierdo del signo igual y algún tipo de valor en el lado derecho.

```
Ejemplos:  
numero = 6; //asigno el valor 6 a la variable numero  
a = 0;  
dia = 22;
```

En el siguiente ejemplo, hay variables tanto a la izquierda como a la derecha del operador de asignación (=). Pero tienen un significado muy diferente. La variable de la derecha (num2) se trata como un valor. Si esa variable almacena 20, entonces ese es su valor. De hecho, cualquier cosa que ocurra en el lado derecho de un operador de asignación se trata como un valor. La variable de la izquierda (num1) se trata como una ubicación de memoria. Es donde se almacenará el valor 20 como resultado de la ejecución de esta declaración. El efecto de esta declaración es copia el valor almacenado en num2 en num1, como se ilustra en la siguiente imagen.

```
Ejemplo  
//se crean las variables  
int num1 = 10;  
int num2 = 20;  
  
num1 = num2; //se copia el valor de num2 a la variable num1
```



## Challenge

### Question

En el programa Hello World, imprime por pantalla la variable miPrimerEntero que hemos creado.

## Challenge2

### Question

En el programa Hello World, crea las siguientes variables adicionales justo debajo de la declaración "int miPrimerEntero = 7;" en el programa:

- num1 de tipo int y con un valor de 10.
- num2 de tipo int y con un valor de 8.
- y una tercera total que sea la suma de las anteriores.
- imprime por pantalla la variable total.

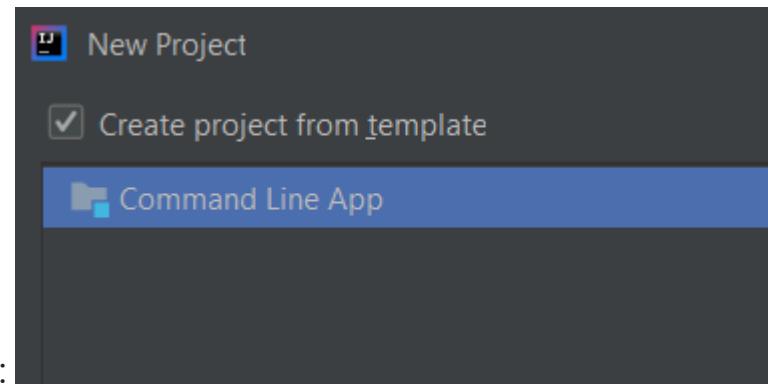
Tabla de contenidos
Sentencias de declaración
Sentencias de asignación
Challenge
Challenge2
Paquetes en Java - Java Packages

## Paquetes en Java - Java Packages

Un paquete es una forma de organizar nuestros proyectos Java. Se pueden ver como carpetas dentro de la estructuración interna del proyecto.

Los paquetes son el mecanismo que usa Java para facilitar la modularidad del código. Un paquete puede contener una o más definiciones de interfaces y clases, distribuyéndose habitualmente como un archivo. Para utilizar los elementos de un paquete es necesario importar este en el módulo de código en curso, usando para ello la sentencia ***import***.

IntelliJ nos ofrece la opción de automatizar esto cuando creamos un proyecto, simplemente

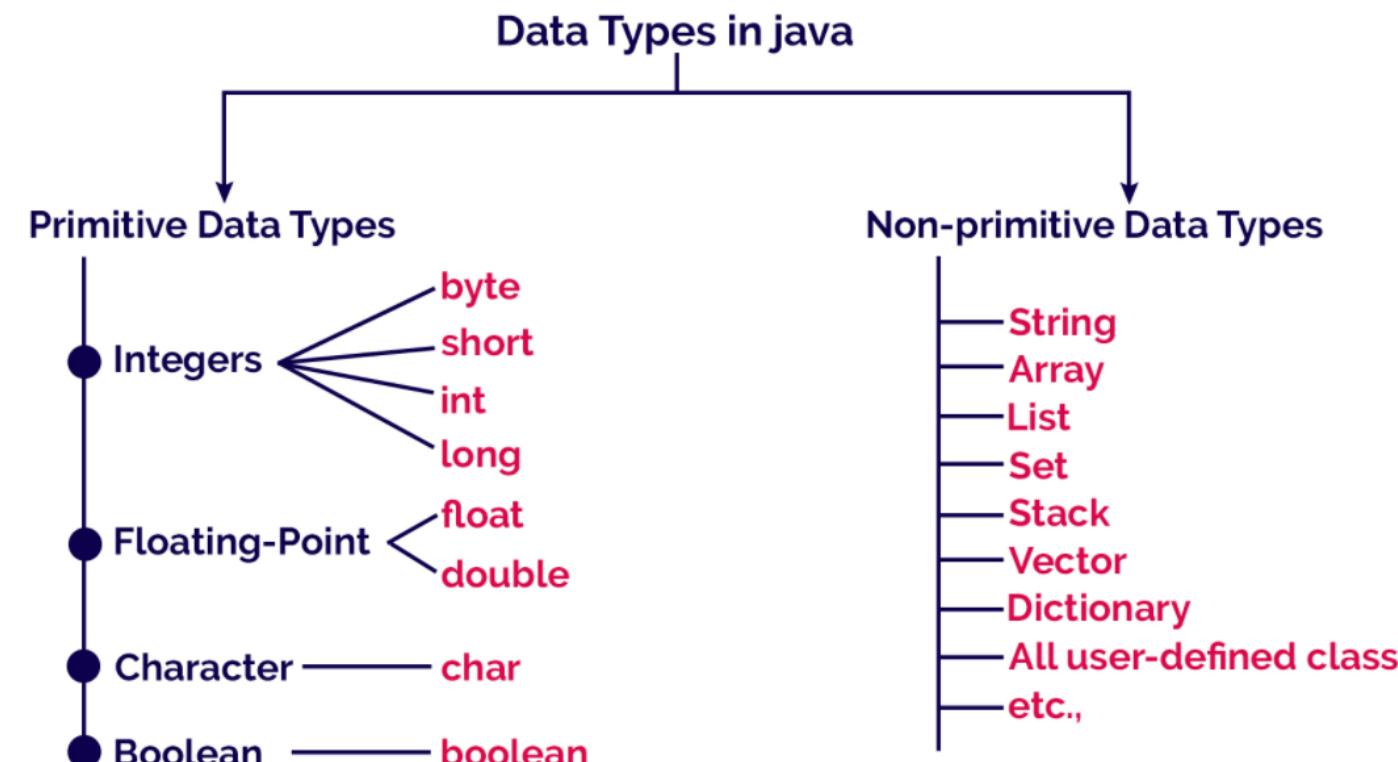


seleccionando la opción:

Programación en Java
Inicio
Elementos de un programa informático
Programa y lenguajes de programación
El lenguaje Java
Variables, sentencias y paquetes
Tipos de datos
<b>Datos primitivos</b>
Datos no primitivos
String
Operadores y expresiones
Comentarios en Java
Constantes y literales
Entrada y salida de información por consola
Introducción a la programación orientada a objetos
Estructuras básicas de control
⊗ Recursividad
Programación orientada a objetos POO
Arrays
POO avanzada
Colecciones
Programación avanzada
Interfaces de usuario
Entrada/Salida (I/O) de la información
Persistencia de la información
Atajos de teclado para IntelliJ
About me

## Tipos de datos

Un tipo de datos es un conjunto de valores y un conjunto de operaciones definidas en ellos. Se pueden clasificar en primitivos y objetos.



### Primitivos

Los primitivos son los más básicos y fundamentales, vienen integrados en Java. Especifican el tipo de valor almacenado en una variable y el tamaño de la memoria. Hay 8 tipos primitivos de datos integrados en el lenguaje Java. Estos incluyen: int, byte, short, long, float, double, boolean y char.

Primitive data type	Wrapper class
byte	Byte
short	Short
int	Integer
long	Long
float	Float
double	Double
boolean	Boolean
char	Character

#### byte

Como su propio nombre denota, emplea un solo byte (8 bits) de almacenamiento. Esto permite almacenar valores en el rango [-128, 127]. Raramente se usa. Ocupa menos memoria y puede ser más rápido accedido.

```
byte b = 2;
```

#### short

Usa el doble de almacenamiento que el anterior, es decir, ocupa 16 bits [-32.768, 32.767].

```
short s = 3467;
```

#### int

Emplea 4 bytes (32 bits) de almacenamiento y es el tipo de dato entero más empleado. Necesita cuatro veces el espacio que ocupa un byte. **Es el entero predeterminado usado en Java.**

```
int maxValor = 2147483647;
// after java 7 and higher
int maxValue = 2_147_483_647;
```

#### long

Es el tipo entero de mayor tamaño, 8 bytes (64 bits). Para definir un long, tenemos que hacerlo de la siguiente forma:

```
long myLongNumber = 500L;
```

#### float

Tiene una parte flotante que sirve para expresar números decimales. Es de simple precisión (formato y cantidad de espacio que ocupa) porque ocupa 32 bits. No se recomienda mucho su uso.

```
float f = 4;
float f = 4f; //también válida
```

#### double

Es un número de precisión doble y ocupa 64 bits. **Es el flotante predeterminado en Java.** Se

### Tabla de contenidos

Primitivos
byte
short
int
long
float
double
char
boolean
Wrapper classes (clases contenedoras)

recomienda su uso. Muchas librerías internas de Java, relacionadas con operaciones

matemáticas, usan double.

```
double d = 5;
double d = 5d; //también válida
```

### char

Se utiliza para almacenar caracteres (letras, números, signos, etc.) individuales. Ocupa 2 bytes en memoria (16 bits). Permite almacenar caracteres Unicode. Unicode es un estándar de codificación internacional que nos permite representar diferentes idiomas; y la forma en que funciona es usando una combinación de los dos bytes que un char ocupa en la memoria, que puede representar hasta 65535 diferentes tipos de caracteres. [Unicode table](#).

```
char c = 'P';
char u = '\u00A2';//print unicode character
```

Un carácter precedido por una barra invertida () es una secuencia de escape y tiene un significado especial para el compilador. La siguiente tabla muestra las secuencias de escape de Java:

**Escape Sequences**

Escape Sequence	Description
\t	Insert a tab in the text at this point.
\b	Insert a backspace in the text at this point.
\n	Insert a newline in the text at this point.
\r	Insert a carriage return in the text at this point.
\f	Insert a form feed in the text at this point.
\'	Insert a single quote character in the text at this point.
\"	Insert a double quote character in the text at this point.
\\\	Insert a backslash character in the text at this point.

### boolean

solo permite almacenar dos posibles valores que son *true* o *false*. Tiene la finalidad de facilitar el trabajo con valores "verdadero/falso" (booleanos), resultantes por regla general de evaluar expresiones.

Este tipo de datos representa un bit de información, pero su "tamaño" no es algo que esté definido con precisión.

```
boolean isMyNamePatri = true;
```

## Wrapper classes (clases contenedores)

Las clases contenedoras proporcionan una forma de utilizar tipos de datos primitivos como objetos. En Java, tenemos una Wrapper class para cada uno de los 8 tipos de datos primitivos. Gracias a esto, podemos realizar operaciones en un dato primitivo como por ejemplo en un int:

```
int minimoValorInt = Integer.MIN_VALUE;
```

Programación en Java
Inicio
Elementos de un programa informático
Programa y lenguajes de programación
El lenguaje Java
Variables, sentencias y paquetes
Tipos de datos
Datos primitivos
<b>Datos no primitivos</b>
String
Operadores y expresiones
Comentarios en Java
Constantes y literales
Entrada y salida de información por consola
Introducción a la programación orientada a objetos
Estructuras básicas de control
⊗ Recursividad
Programación orientada a objetos POO
Arrays
POO avanzada
Colecciones
Programación avanzada
Interfaces de usuario
Entrada/Salida (I/O) de la información
Persistencia de la información
Atajos de teclado para IntelliJ
About me

## No primitivos u objetos

En Java, los tipos de datos no primitivos son los tipos de datos de referencia o los tipos de datos creados por el usuario. Todos los tipos de datos no primitivos se implementan utilizando conceptos de objeto. Cada variable del tipo de datos no primitivo es un objeto. Los tipos de datos no primitivos pueden utilizar métodos adicionales para realizar determinadas operaciones. El valor predeterminado de la variable de tipo de datos no primitivos es nulo.

### String - Cadena de caracteres

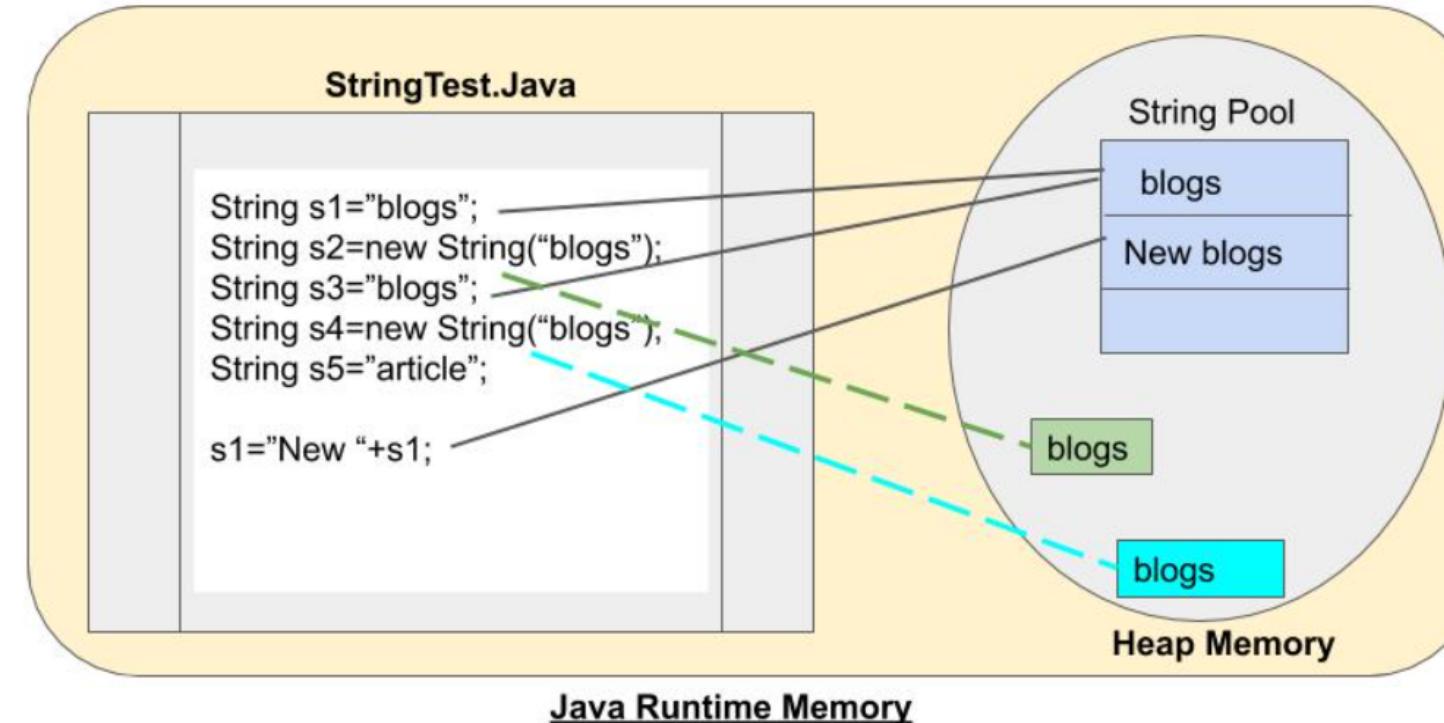
String es una clase integrada en el lenguaje Java ampliamente utilizada y definido en el paquete `java.lang`. Representa cadenas de caracteres y se utilizan para almacenar varios atributos como nombre de usuario, contraseña, etc. Los String son **inmutables**, es decir, no se pueden modificar una vez creados. Siempre que se modifica un objeto String, en realidad se crea uno nuevo String.

Existen varias formas para crear un String:

```
String texto = "Severo Ochoa";
String texto2 = new String("Severo Ochoa");
```

### Asignación de memoria de objetos String

La memoria se divide en dos partes, el String Pool y la memoria Heap.



Veamos como funcionaría para la imagen anterior.

1. Siempre que creamos un String con comillas dobles, se almacenan en String Pool. String Pool almacena el único valor en él. Es por eso que, String s1 = "blogs" se almacenan como el primer valor en el String Pool.
2. Siempre que creamos un objeto usando la palabra clave `new`, se almacena en la memoria Heap pero fuera del String Pool. Aquí se puede almacenar valores duplicados ya que pertenece a diferentes objetos. Entonces, para la declaración String s2 = new String("blogs"), aunque el valor de s1 y s2 es el mismo, s2 se almacenará fuera del String Pool.
3. Cuando creamos otro String usando comillas dobles, primero verifica todos los valores en el String Pool y si coincide con alguno se asigna la misma ubicación asignada a otro objeto de referencia. Por lo tanto, String s3 = "blogs" no agregará una nueva entrada en el String Pool.
4. Pero si creamos otro objeto con un valor existente usando la palabra clave `new`. Asignará nueva memoria al nuevo objeto en el Heap. Por lo tanto, a String s4 = new String ("blogs") se le asignará nueva memoria.
5. Ahora, si manipulamos el valor en s1 usando s1 = "New" + s1, no actualizará la entrada o referencia existente de s1. Este proceso creará una nueva entrada en el String Pool con el valor "New blogs" y la referencia de memoria cambiaría para el objeto s1.

### Java String Class Methods

La clase `java.lang.String` proporciona muchos métodos útiles para realizar operaciones en la secuencia de valores char.

### Date

En sus inicios se crearon las clases `java.util.Date` y `java.sql.Date` para almacenar y manejar fechas. Pero ambas clases son defectuosas en diseño e implementación. Por ejemplo, las clases existentes (como `java.util.Date` y `SimpleDateFormat`) no son thread-safe, lo que genera posibles problemas de concurrencia para los usuarios.

Por tanto, desde Java 8 y posteriores se ha desarrollado una nueva API `java.time` que resuelve los problemas que presentaban las librerías anteriores.

Tabla de contenidos
String - Cadena de caracteres
Asignación de memoria de objetos String
Java String Class Methods
Date
Enum en Java

Date-time types in Java & SQL	Legacy class	Modern class	SQL standard data type
Moment in UTC	java.util. <b>Date</b> java.sql. <b>Timestamp</b>	java.time. <b>Instant</b>	TIMESTAMP WITH TIME ZONE
Moment with offset-from-UTC (hours-minutes-seconds)	( lacking )	java.time. <b>OffsetDateTime</b> <small>JDBC 4.2</small>	TIMESTAMP WITH TIME ZONE
Moment with time zone (`Continent/Region`)	java.util. <b>GregorianCalendar</b> javax.xml.datatype. <b>XMLEGregorianCalendar</b>	java.time. <b>ZonedDateTime</b>	TIMESTAMP WITH TIME ZONE
Date & Time-of-day (no offset, no zone) Not a moment	( lacking )	java.time. <b>LocalDateTime</b>	TIMESTAMP WITHOUT TIME ZONE
Date only (no offset, no zone)	java.sql. <b>Date</b>	java.time. <b>LocalDate</b>	DATE
Time-of-day only (no offset, no zone)	java.sql. <b>Time</b>	java.time. <b>LocalTime</b>	TIME WITHOUT TIME ZONE
Time-of-day with offset (impractical, not used)	( lacking )	java.time. <b>OffsetTime</b>	TIME WITH TIME ZONE

[More information](#)

[Why do we need a new date and time library?](#)

Ejemplo de código para mostrar la fecha y la hora:

```
//mostrar fecha
LocalDate ld = LocalDate.now();
System.out.println(ld);

//mostrar hora
LocalTime lt = LocalTime.now();
System.out.println(lt);

//mostrar fecha y hora
LocalDateTime ldt = LocalDateTime.now();
System.out.println(ldt);

//formatear la fecha con un formato dado
DateTimeFormatter formatter = DateTimeFormatter.ofPattern("dd-MM-yyyy HH:mm:ss");
String formatted = formatter.format(ldt);
System.out.println(formatted);
```

## Enum en Java

El tipo enumerado es un tipo de datos especial que permite que una variable sea un conjunto de constantes predefinidas. La variable debe ser igual a uno de los valores que se han predefinido para ella.

Debido a que son constantes, los nombres de los campos del tipo enum deben estar en letras mayúsculas.

En Java, se define un enumerado utilizando la palabra clave **enum** seguido del nombre siguiendo la convención del nombrado de clases. Primera letra en mayúscula y CamelCase.

Para crear un enum en Java, botón derecho en el paquete --> new Java class y seleccionamos enum.

Ejemplo de enumerado:

```
public enum PuntosCardinales {
    NORTE, SUR, ESTE, OESTE
}

public class Main {

    public static void main(String[] args) {
        PuntosCardinales myVar = PuntosCardinales.ESTE;
        System.out.println(myVar);
    }
}
```

Programación en Java
Inicio
Elementos de un programa informático
Programa y lenguajes de programación
El lenguaje Java
Variables, sentencias y paquetes
Tipos de datos
Datos primitivos
Datos no primitivos
String
Operadores y expresiones
Comentarios en Java
Constantes y literales
Entrada y salida de información por consola
Introducción a la programación orientada a objetos
Estructuras básicas de control
⊗ Recursividad
Programación orientada a objetos POO
Arrays
POO avanzada
Colecciones
Programación avanzada
Interfaces de usuario
Entrada/Salida (I/O) de la información
Persistencia de la información
Atajos de teclado para IntelliJ
About me

## String en Java

Un String es un tipo de dato no primitivo que, en Java representa una cadena de caracteres no modificable. Todos los literales de la forma "cualquier texto", es decir, literales entre comillas dobles, que aparecen en un programa java se implementan como objetos de la clase String.

A diferencia de muchos objetos vimos que un String se puede crear sin la palabra new.

```
String text = "hola";
```

### Creación de String

Se puede crear un String de varias formas.

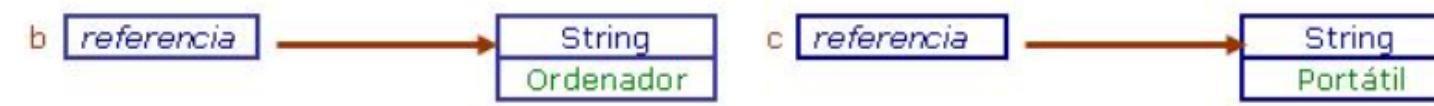
```
String texto = "Severo Ochoa";
//Utilizando new
String texto2 = new String("Severo Ochoa");

//Utilizando el operador concatenación +
String s2 = text + " 2021"; //s2 contiene "Severo Ochoa 2021"
```

### El operador concatenación

La clase proporciona el operador + (concatenación) para unir dos o más String. El resultado de aplicar este operador es un nuevo String concatenación de los otros. Por ejemplo, si tenemos dos String b y c:

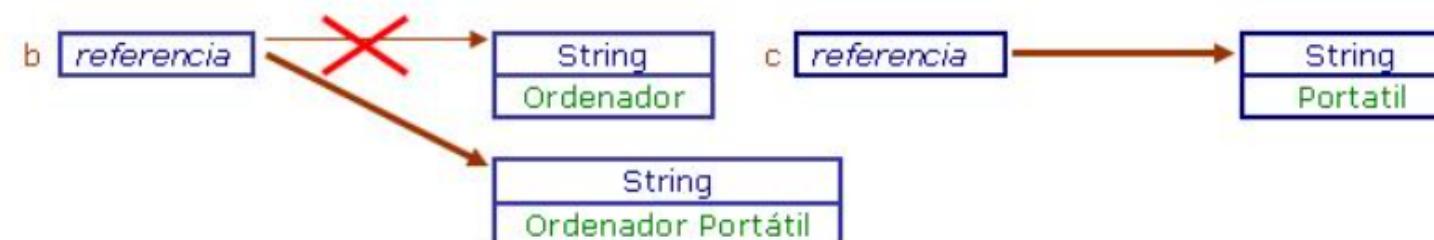
```
String b = "Ordenador";
String c = " Portátil";
```



### La operación

```
b = b + c;
```

Crea un nuevo String que se incluye en el String Pool:



## Índices

Cada uno de los caracteres que forman un String son del tipo primitivo **char**. Los caracteres de un string están numerados internamente con índices empezando desde el cero:

```
String name = "Ultimate";
```

index	0	1	2	3	4	5	6	7
character	U	l	t	i	m	a	t	e

El primer carácter tiene índice 0 y el último tiene la longitud del string menos 1.

### Métodos de la clase String

La clase String proporciona métodos para el tratamiento de las cadenas de caracteres: acceso a caracteres individuales, buscar y extraer una subcadena, copiar cadenas, convertir cadenas a mayúsculas o minúsculas, etc.

MÉTODO	DESCRIPCIÓN
length()	Devuelve la longitud de la cadena
indexOf('caracter')	Devuelve la posición de la primera aparición de <i>caracter</i> dentro del String. Devuelve -1 si no lo encuentra.
lastIndexOf('caracter')	Devuelve la posición de la última aparición de <i>caracter</i> dentro del String. Devuelve -1 si no lo encuentra.
charAt(n)	Devuelve el carácter que está en la posición n
substring(n1,n2)	Devuelve la subcadena desde la posición n1 hasta n2 - 1
toUpperCase()	Devuelve la cadena convertida a mayúsculas
toLowerCase()	Devuelve la cadena convertida a minúsculas
equals(otroString)	Compara dos cadenas y devuelve true si son iguales
equalsIgnoreCase(otroString)	Igual que equals pero sin considerar mayúsculas y minúsculas
compareTo(OtroString)	Devuelve 0 si las dos cadenas son iguales. <0 si la primera es alfabéticamente menor que la segunda ó >0 si la primera es alfabéticamente mayor que la segunda.
compareToIgnoreCase(OtroString)	Igual que compareTo pero sin considerar mayúsculas y minúsculas.
valueOf(N)	Convierte el valor N a String. N puede ser de cualquier tipo.

Para acceder a alguno de los métodos siguientes utilizamos la notación `.`

```
String texto = "Clase";
```

Tabla de contenidos
Creación de String
El operador concatenación
Índices
Métodos de la clase String
Comparar Strings
char dentro de String
char
Diferencias entre char y String

```

int longitud = texto.length(); //devuelve 5

// index      012345678901
String s1 = "Stuart Reges";
String s2 = "Marty Stepp";
System.out.println(s1.length());           // 12
System.out.println(s1.indexOf("e"));        // 8
System.out.println(s1.substring(7, 10));     // "Reg"
String s3 = s2.substring(1, 7);
System.out.println(s3.toLowerCase());        // "arty s"

```

#### Tip

Para más información consulta la documentación oficial de la clase `String`

## Comparar Strings

Los operadores relacionales como == o < > NO se utilizan para comparar Strings, aunque el código compile no es correcto, ya que == compara objetos, y devolvería falso aunque dos strings tuvieran el mismo texto puesto que son objetos diferentes.

Para comparar strings utilizamos el método `equals`.

```

String name = "Patri";
if (name.equals("Patri")) {
    System.out.println("Coincide.");
}

```

La siguiente tabla muestra los métodos que se utilizan para comparar Strings.

Method	Description
<code>equals(str)</code>	whether two strings contain the same characters
<code>equalsIgnoreCase(str)</code>	whether two strings contain the same characters, ignoring upper vs. lower case
<code>startsWith(str)</code>	whether one contains other's characters at start
<code>endsWith(str)</code>	whether one contains other's characters at end
<code>contains(str)</code>	whether the given string is found within this one

## char dentro de String

Como se ha comentado, un String está compuesto de caracteres tipo char.

Para acceder a los caracteres dentro de un String usamos el método `charAt`.

Se puede usar la concatenación + para concatenar char con String.

```

String food = "cookie";
char firstLetter = food.charAt(0); // 'c'
System.out.println(firstLetter + " is for " + food);

```

También podemos recorrer el String con un bucle for e imprimir cada uno de los caracteres que lo forman.

```

String major = "CSE";
for (int i = 0; i < major.length(); i++) {
    char c = major.charAt(i);
    System.out.println(c);
}

```

OUTPUT  
C  
S  
E

## char

A todos los valores char se les asigna un número internamente por el ordenador, son los llamados valores ASCII. Por ejemplo:

el carácter 'A' es 65 en código ASCII

el carácter 'a' es 97 en código ASCII

Mezclar tipos de datos char e int automáticamente cause una conversión en entero. Por ejemplo:

'a' + 10 --> devuelve 107.

Para convertir un entero en su equivalente a carácter (char) haríamos:

(char) ('a' + 2) --> devuelve 'c'.

## Diferencias entre char y String

- String es un objeto, por tanto, contiene métodos.
- char es un tipo de dato primitivo, no puedes llamar a métodos con él.
- String utiliza comillas dobles.
- char utiliza comillas simples.
- No se puede comparar un String usando operadores relacionales.
- Si se puede comparar un char usando operadores relacionales: 'a' < 'b', 'X' == 'X', ...

Programación en Java
Inicio
Elementos de un programa informático
Programa y lenguajes de programación
El lenguaje Java
Variables, sentencias y paquetes
Tipos de datos
Operadores y expresiones
Comentarios en Java
Constantes y literales
Entrada y salida de información por consola
Introducción a la programación orientada a objetos
Estructuras básicas de control
@ Recursividad
Programación orientada a objetos POO
Arrays
POO avanzada
Colecciones
Programación avanzada
Interfaces de usuario
Entrada/Salida (I/O) de la información
Persistencia de la información
Atajos de teclado para IntelliJ
About me

## Operadores

Los operadores son símbolos especiales en Java que realizan operaciones entre uno o varios operandos y devuelve un resultado. Uno de los más usados es el operador suma (+) como hemos visto en clases anteriores.

### Operando

Es cualquier término, que puede ser una variable o valor y que es manipulado por un operador.

```
int valor = 8;
int numero = valor + 12;
```

En el ejemplo anterior, + es el operador y valor y 12 son los operandos. valor + 12 es una expresión que devuelve el resultado de 20.

### Expresiones

Una expresión es una combinación de literales, operadores, nombres de variables y paréntesis que se utilizan para calcular un valor.

```
int miPrimerEntero = 7 + 5;
int resultado = 0;

resultado = (miPrimerEntero * 10) / (32 + 12);
```

Java examina la expresión de la derecha del signo igual y realiza el cálculo de una expresión matemática. Después asigna ese valor a la variable resultado. Podríamos complicar más la expresión utilizando **operadores** como paréntesis, multiplicaciones, divisiones, etc.

Las partes de una expresión deben estar ordenadas correctamente. Las reglas para las expresiones Java correctas son casi las mismas que las del álgebra:

1. Cada operador debe tener el número correcto de operandos.
  - Multiplicación \*, División /, Suma +, Resta -: debe tener dos operandos, uno en cada lado.
  - La negación - y unario más + deben ir seguidos de un operando.
2. Los paréntesis () pueden rodear una expresión legal para convertirla en operando.

Expression	Correct or Not Correct?	Expression	Correct or Not Correct?
25	CORRECT	25 - value	CORRECT
2( a - b )	NOT correct	(a-b) * (c-d)	CORRECT
A - b/c + D	CORRECT	-sum + partial	CORRECT
((x+y) / z) / (a - b)	CORRECT	((m-n) + (w-x-z) / (p % q))	NOT correct

### Expresiones mixtas con int y double

Si ambos operandos de un operador aritmético son de tipo int, entonces la operación es una operación entera. Si algún operando es de punto flotante, entonces la operación es de punto flotante.

## Tipos de operadores en Java

Java proporciona muchos tipos de operadores que se pueden usar según la necesidad. Se clasifican según la funcionalidad que brindan. Algunos de los tipos son los siguientes:

Operadores aritméticos, unarios, de asignación, relacionales, lógicos, etc.

### Operador de asignación (=)

Es uno de los operadores más usados. Se usa para asignar un valor a cualquier variable. Tiene una asociación de derecha a izquierda, es decir, el valor dado en el lado derecho del operador se asigna a la variable de la izquierda y, por lo tanto, el valor del lado derecho debe declararse antes de usarlo o debe ser una constante.

### Operadores aritméticos

Se utilizan para realizar operaciones aritméticas simples.

Símbolo	Operación	Descripción
+	Suma	Realiza la suma de los operandos.
-	Resta	Realiza la resta de los operandos.
*	Producto	Multiplica los operandos.
/	División	Realiza la división.
%	Módulo	Calcula el resto.

### Operadores unarios

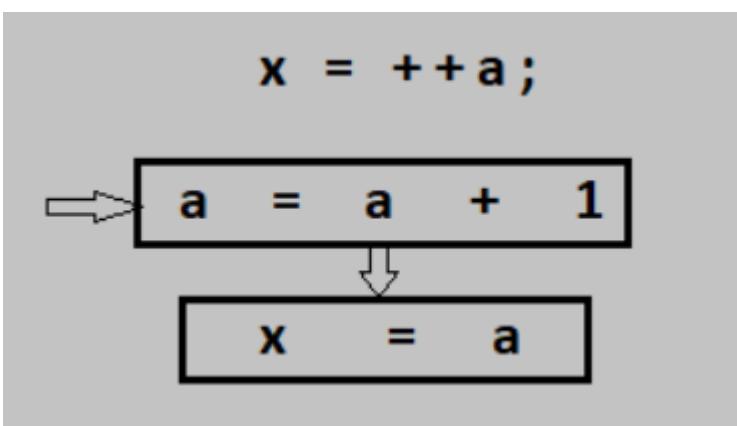
Los operadores unarios solo necesitan un operando. Se usan para incrementar, disminuir o negar un valor.

Tabla de contenidos
Operando
Expresiones
Expresiones mixtas con int y double
Tipos de operadores en Java
Operador de asignación (=)
Operadores aritméticos
Operadores unarios
Operadores relacionales
Operadores lógicos
Operadores de bits
Operador ternario (?:)
Abreviaciones
Precedencia de operadores
Conversiones de tipo
Conversiones por defecto
Conversiones forzadas (casting entre tipos nativos)
Sintaxis de las expresiones matemáticas

Símbolo	Operación	Descripción
<code>++</code>	Incremento	Incrementa el valor en 1 unidad.
<code>--</code>	Decremento	El valor disminuye en 1 unidad.
<code>!</code>	NOT lógico	Invierte un valor booleano.

Existen dos versiones de estos operadores:

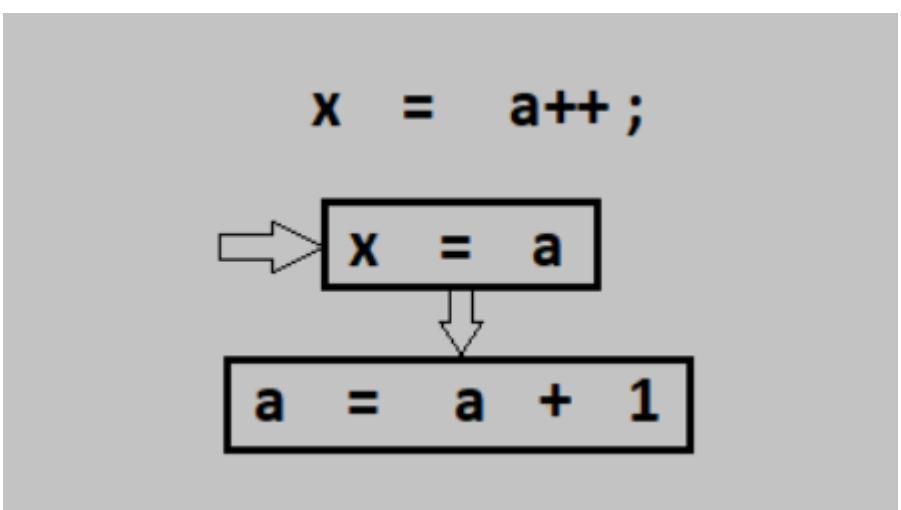
- Pre-incremento y pre-decremento. El valor se aumenta/disminuye primero y luego se calcula el resultado.



```
int a = 8, b = 1;
b = ++a; //b=9, a=9
```

- Post-Incremento y post-decremento: el valor se usa por primera vez para calcular el resultado y luego se incrementa/decremente.

```
int a = 8, b = 1;
b = a++; //b = 8, a = 9
```



## Operadores relacionales

Estos operadores se utilizan para verificar relaciones como igualdad, mayor que, menor que. Devuelven el resultado booleano después de la comparación.

Símbolo	Operación	Descripción
<code>==</code>	Igual a	Devuelve verdadero si el valor de la izquierda del símbolo es igual al de la derecha.
<code>!=</code>	Distinto a	Devuelve verdadero si el valor de la izquierda es distinto al de la derecha.
<code>&lt;</code>	Menor que	Devuelve verdadero si el valor de la izquierda es menor que el de la derecha.
<code>&lt;=</code>	Menor o igual que	Devuelve verdadero si el valor de la izquierda es menor o igual que el de la derecha.
<code>&gt;</code>	Mayor que	Devuelve verdadero si el valor de la izquierda es mayor que el de la derecha.
<code>&gt;=</code>	Mayor o igual que	Devuelve verdadero si el valor de la izquierda es mayor o igual al de la derecha.

```
int a = 20, b = 10;
System.out.println("a == b :" + (a == b)); //Devuelve falso, porque a no es igual
```

## Operadores lógicos

Estos operadores se utilizan para realizar operaciones lógicas **AND** y **OR**. Se usa ampliamente en sentencias if-then o bucles para verificar condiciones, establecer un punto de salida de un bucle o la toma de decisiones. Los operadores condicionales son:

Símbolo	Operación	Descripción
<code>&amp;&amp;</code>	AND lógico	Devuelve verdadero cuando ambas condiciones son ciertas.
<code>  </code>	OR lógico	Devuelve verdadero si al menos una condición es cierta.

INPUT		OUTPUT	INPUT		OUTPUT
A	B	A OR B	A	B	A AND B
0	0	0	0	0	0
0	1	1	0	1	0
1	0	1	1	0	0
1	1	1	1	1	1

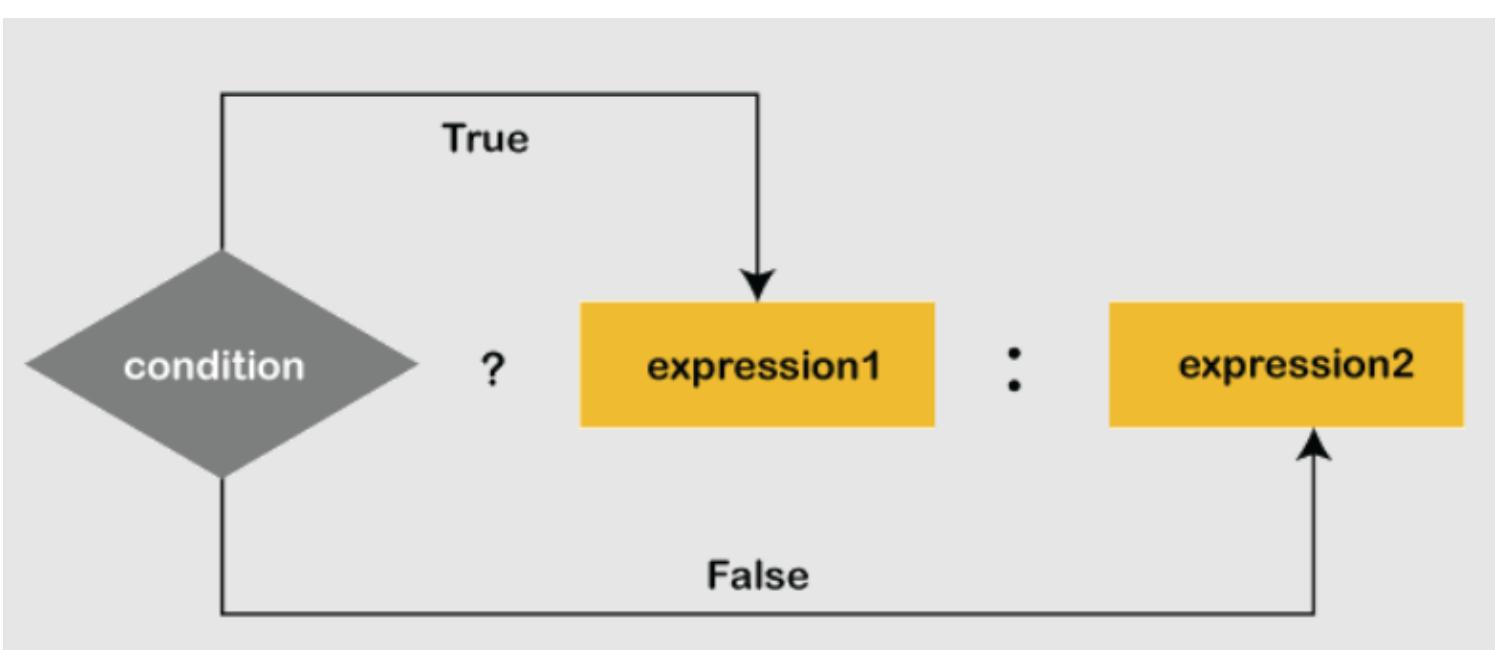
```
int a = 20, b = 10, c = 10;
System.out.println((b == c && a == c)); //False
System.out.println((a == c && b == c)); //False
System.out.println((a == b || b == c)); //True
```

## Operadores de bits

Símbolo	Operación	Descripción
&	AND	Si ambos bits de entrada son 1, establece como resultado 1. De lo contrario 0.
	OR	Si por lo menos uno de los bits de entrada es 1, establece como resultado 1. De lo contrario 0.
^	XOR	Si uno de los bits de entrada es 1 y el otro 0, establece como resultado 1. Si los bits son iguales establece 0.
~	NOT	Invierte todos los bits y devuelve el resultado en complemento a 2.

## Operador ternario (?:)

Ternario es una versión abreviada de la sentencia if-else. Tiene tres operandos y de ahí el nombre ternario. El formato general es:



La declaración anterior significa que si la condición se evalúa como verdadera, entonces ejecuta las instrucciones después del '?' de lo contrario, ejecuta las instrucciones después de ':'.

```
int nota = 7;
String notaFinal = (nota >= 5) ? "Aprobado" : "Suspendido";
System.out.println(notaFinal); //muestra Aprobado
```

## Abreviaciones

En muchos casos, el operador de asignación se puede combinar con otros operadores para construir una versión más corta de la declaración llamada Declaración Compuesta (Compound Statement).

- += , para sumar el operando izquierdo con el operando derecho y luego asignarlo a la variable de la izquierda.

```
int a = 5;
a += 5; // a = a + 5;
```

- -= , para restar el operando izquierdo con el operando derecho y luego asignarlo a la variable de la izquierda.

```
int a = 5;
a -= 5; // a = a - 5;
```

- \*= , para multiplicar el operando izquierdo con el operando derecho y luego asignándolo a la variable de la izquierda.

```
int a = 5;
a *= 5; // a = a * 5;
```

- /= , para dividir el operando izquierdo con el operando derecho y luego asignarlo a la variable de la izquierda.

```
int a = 5;
a /= 5; // a = a / 5;
```

- %= , para asignar el módulo del operando izquierdo con el operando derecho y luego asignarlo a la variable de la izquierda.

```
int a = 5;
a %= 5; // a = a % 5;
```

## Precedencia de operadores

El orden de precedencia, es decir, como Java decide la prioridad al evaluar los operadores en una expresión, se puede ver en [Java Operator Precedence Table](#)

## Conversiones de tipo

### Conversiones por defecto

Las reglas de Java para saber el tipo de datos resultante de una expresión se siguen las siguientes reglas:

Si algún operando es	El otro operando se transforma a
double	double
float	float
long	long
byte or short	int

Si se aplican dos regla, se elige la que aparece primero en la tabla.

#### Conversiones forzadas (casting entre tipos nativos)

Es una forma de convertir un número de un tipo a otro tipo de dato. Para hacerlo ponemos en paréntesis el tipo de dato al que queremos convertirlo.

```
byte miByte = (byte) (14 / 2); //conviero la operación división que devuelve un int a un byte

//Otra forma
float a = 8.0f;
int b = 10;
b = (int) a; //convierro el tipo float a int
```

#### Otros operadores

Existen más operadores que no se han mencionado en el curso. Si se desea consultar todos los operadores de Java se pueden ver en su documentación oficial. [More information](#)

### Sintaxis de las expresiones matemáticas

Método	Returns	Ejemplo
Math.abs	valor absoluto	Math.abs(-308) returns 308
Math.ceil	redondeo hacia arriba	Math.ceil(2.13) returns 3.0
Math.floor	redondeo hacia abajo	Math.floor(2.93) returns 2.0
Math.max	valor máx. de dos valores	Math.max(45, 207) returns 207
Math.min	valor min. de dos valores	Math.min(3.8, 2.75) returns 2.75
Math.pow	potencia	Math.pow(3, 4) returns 81.0
Math.round	redondear al entero más cercano	Math.round(2.718) returns 3
Math.sqrt	raíz cuadrada	Math.sqrt(81) returns 9.0



Programación en Java
Inicio
Elementos de un programa informático
Programa y lenguajes de programación
El lenguaje Java
Variables, sentencias y paquetes
Tipos de datos
Operadores y expresiones
Comentarios en Java
Constantes y literales
Entrada y salida de información por consola
Introducción a la programación orientada a objetos
Estructuras básicas de control
@ Recursividad
Programación orientada a objetos POO
Arrays
POO avanzada
Colecciones
Programación avanzada
Interfaces de usuario
Entrada/Salida (I/O) de la información
Persistencia de la información
Atajos de teclado para IntelliJ
About me

## Comentarios en Java

Los comentarios son ignorados por el ordenador y se usan en el programa para ayudar a describir alguna funcionalidad o explicación del código fuente. Existen dos tipos de comentarios.

### Comentarios de una línea

Usamos la doble barra // para realizar un comentario de una sola línea. Lo podemos usar en una línea en blanco o detrás del código.

```
int gravity; //variable para calcular la gravedad
```

### Comentarios multilínea

Usamos la barra con asterisco doble para realizar comentarios que afecten a más de una línea. De forma que cuando abramos un comentario usaremos /\*, para cerrarlo \*/.

```
/*
    Clase que almacena en la base de datos
    los datos de una persona.
*/
public class Persona {
    ...
}
```

### Tabla de contenidos

[Comentarios de una línea](#)

[Comentarios multilinea](#)





Programación en Java
Inicio
Elementos de un programa informático
Programa y lenguajes de programación
El lenguaje Java
Variables, sentencias y paquetes
Tipos de datos
Operadores y expresiones
Comentarios en Java
Constantes y literales
Entrada y salida de información por consola
Introducción a la programación orientada a objetos
Estructuras básicas de control
@ Recursividad
Programación orientada a objetos POO
Arrays
POO avanzada
Colecciones
Programación avanzada
Interfaces de usuario
Entrada/Salida (I/O) de la información
Persistencia de la información
Atajos de teclado para IntelliJ
About me

## Constantes y literales

Tabla de contenidos
Las constantes
Los literales

### Las constantes

Un programa puede contener ciertos valores que no deben cambiar durante su ejecución. Estos valores se llaman constantes. Definición: Una constante es una zona de memoria que se referencia con un identificador, conocido como nombre de la constante, donde se almacena un valor que no puede cambiar durante la ejecución del programa.

La nomenclatura para definir las constantes es la siguiente:

- Todas las letras de cada palabra deben estar en mayúsculas
- Se separa cada palabra con un \_
- Se declaran similar a las variables, con la diferencia de que el tipo de dato va después de la palabra reservada final

```
final double PI = 3.141591;
final int MIN_WIDTH = 4;
final double TASAS = 0.045;
```

Las constantes hacen que el programa sea más fácil de leer y verificar que sea correcto. Si es necesario cambiar una constante (por ejemplo, si cambian las tasas), todo lo que tendremos que hacer es cambiar la declaración de la constante. No será necesario buscar en todo el programa cada aparición de ese número específico.

### Los literales

Un literal Java es un valor de tipo entero, real, lógico, carácter, cadena de caracteres o un valor nulo (null) que puede aparecer dentro de un programa. Por ejemplo: 150, 12.4, "ANA", null, 't'. Los literales suelen aparecer en la asignación de valores a las variables o formando parte de expresiones aritméticas o lógicas.

```
int x = 25;
double precio = 120.99;
String mes = "enero";
descuento = precio - (precio * 5 / 100);
```



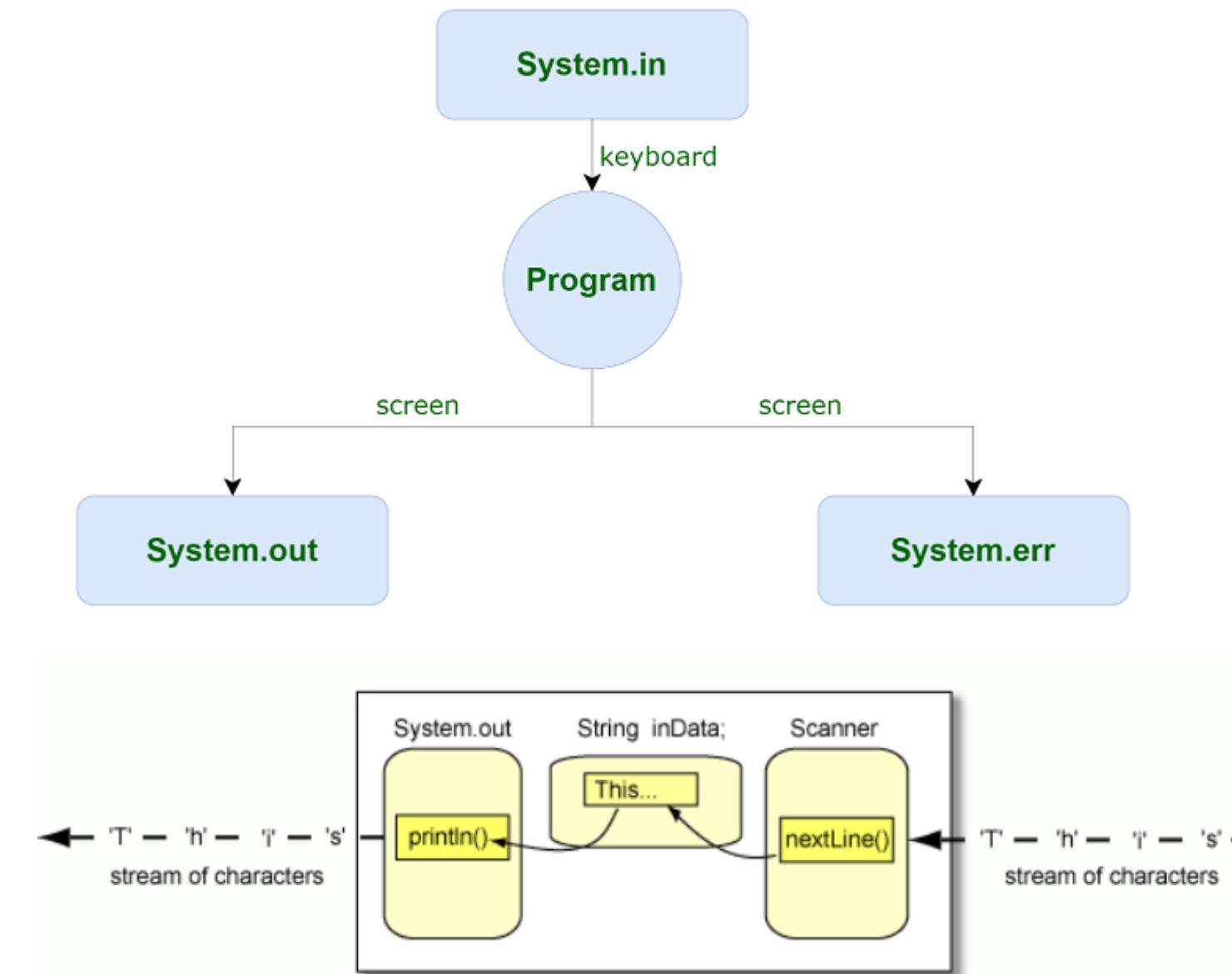
Programación en Java
Inicio
Elementos de un programa informático
Programa y lenguajes de programación
El lenguaje Java
Variables, sentencias y paquetes
Tipos de datos
Operadores y expresiones
Comentarios en Java
Constantes y literales
<b>Entrada y salida de información por consola</b>
Introducción a la programación orientada a objetos
Estructuras básicas de control
@ Recursividad
Programación orientada a objetos POO
Arrays
POO avanzada
Colecciones
Programación avanzada
Interfaces de usuario
Entrada/Salida (I/O) de la información
Persistencia de la información
Atajos de teclado para IntelliJ
About me

## Programación de la consola: entrada y salida de información (I/O input/output)

Java viene con una biblioteca de clases que se puede usar para realizar tareas comunes. La biblioteca de clases de Java está organizada en un conjunto de paquetes, donde cada paquete contiene una colección de clases relacionadas.

En esta sección presentamos las clases System y Scanner que se utilizan para imprimir la salida y leer la entrada de un programa.

El tipo más simple de interfaz de usuario es la interfaz de línea de comandos, en la que la entrada se toma de la línea de comandos a través del teclado y la salida se muestra en la consola. Algunas aplicaciones Java utilizan este tipo de interfaz otros usan interfaz gráfica como veremos más adelante.



### Salida de la información

En Java, cualquier origen o destino de I/O se considera un flujo de bytes o caracteres. Para realizar la salida, insertamos bytes o caracteres en la secuencia. Para realizar la entrada, extraemos bytes o caracteres del flujo (stream). Incluso los caracteres introducidos en un teclado, si se consideran como una secuencia de pulsaciones de teclas, se pueden representar como un stream.

En Java, la I/O se maneja a través de métodos que pertenecen a clases contenidas en el paquete java.io. Ya hemos visto cómo se usa el método de salida println() para enviar una cadena a la consola. Por ejemplo:

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

imprime el texto *Hola mundo* por la consola.

Los objetos **System.out** y **System.err** se pueden usar para escribir la salida en la consola. Como sugiere su nombre, el flujo de errores se usa principalmente para mensajes de error, mientras que el flujo de salida se usa para otras salidas impresas.

```
System.err.println("Falló al abrir el fichero");
```

De manera similar, como sugiere su nombre, el objeto System.in se puede usar para manejar la entrada, que se trata en el siguiente punto.

La única diferencia entre los métodos print() y println() es que println() también imprimirá un retorno de carro y un avance de línea después de imprimir sus datos, lo que permitirá que la salida posterior se imprima en una nueva línea. Por ejemplo:

```
System.out.print("Hola");
System.out.print("mundo");
System.out.println("Texto con salto de linea");
System.out.println("adiós");
```

```
Hola
mundo
Texto con salto de linea
adiós
```

### Entrada de la información java.util.Scanner

Se ha agregado la clase Scanner al paquete java.util que permite la entrada de teclado sin forzar el programador para manejar las excepciones. La clase Scanner está diseñada para ser una forma muy flexible de reconocer fragmentos de datos que se ajustan a patrones específicos de cualquier flujo de entrada.

Para usar la clase Scanner para la entrada de teclado, debemos crear una instancia de Scanner y asociarla con System.in. La clase tiene un constructor para este propósito, por lo que la declaración

```
Scanner sc = new Scanner(System.in);
```

declara y crea una instancia de un objeto que se puede utilizar para la entrada del teclado. Despues de crear un objeto Scanner, podemos hacer una llamada a nextInt(), nextDouble(), o next() para leer, respectivamente, un entero, un número real, o String del teclado.

Tabla de contenidos
Salida de la información
Entrada de la información
java.util.Scanner

```
public static void main(String[] args) {
    Scanner sc = new Scanner (System.in);
    System.out.print("Introduce un número: ");
    int num = sc.nextInt(); //Read the integer
    System.out.println("El número introducido es: " + num);
}
```

Cuando se ejecuta el método `nextInt()`, no se ejecutan más declaraciones hasta que el método devuelve un valor int. Normalmente, esto no sucede hasta que el usuario ha escrito los dígitos de un número entero y presiona la tecla Intro o Intro.

Para leer un String utilizamos el método `next()`.

```
public static void main(String[] args) {
    Scanner sc = new Scanner (System.in);
    System.out.print("Introduce una palabra: ");
    String str = sc.next();
    System.out.println(str);
}
```

Un objeto `Scanner` tiene un conjunto de cadenas de caracteres que separan o delimitan los fragmentos de datos que está buscando. De forma predeterminada, este conjunto de delimitadores consta de cualquier secuencia no vacía de caracteres en blanco, es decir, los caracteres de espacio, tabulación, retorno y nueva línea. Esto permitirá al usuario ingresar varios números enteros separados por espacios antes de presionar la tecla Enter. En código sería:

```
System.out.print("Introduce dos números: ");
int num = sc.nextInt();
int num2 = sc.nextInt();
```

Los espacios en blanco como delimitadores también significan que el método `next()` no puede devolver una cadena vacía ni puede devolver una cadena que contenga espacios. Por ejemplo, considere el código:

```
System.out.print("Introduce un texto separado por espacio en blanco: ");
String str = sc.next();
```

Si se escribe "Hola mundo" y se presiona la tecla enter, la cadena str almacenará sólo "Hola".

Para que un objeto `Scanner` lea cadenas que contienen espacios, debemos usar el método `nextLine()`:

```
String str = sc.nextLine();
```

Programación en Java
Inicio
Elementos de un programa informático
Programa y lenguajes de programación
El lenguaje Java
Variables, sentencias y paquetes
Tipos de datos
Operadores y expresiones
Comentarios en Java
Constantes y literales
Entrada y salida de información por consola
Introducción a la programación orientada a objetos
Estructuras básicas de control
Recursividad
Programación orientada a objetos POO
Arrays
POO avanzada
Colecciones
Programación avanzada
Interfaces de usuario
Entrada/Salida (I/O) de la información
Persistencia de la información
Atajos de teclado para IntelliJ
About me

## Introducción a la programación orientada a objetos

La programación modular es un paradigma que consiste en dividir un programa en módulos con el fin de hacerlo más legible y manejable. Enfatiza este concepto mediante la construcción de aplicaciones a partir de su división en componentes independientes que llevan a cabo tareas concretas.

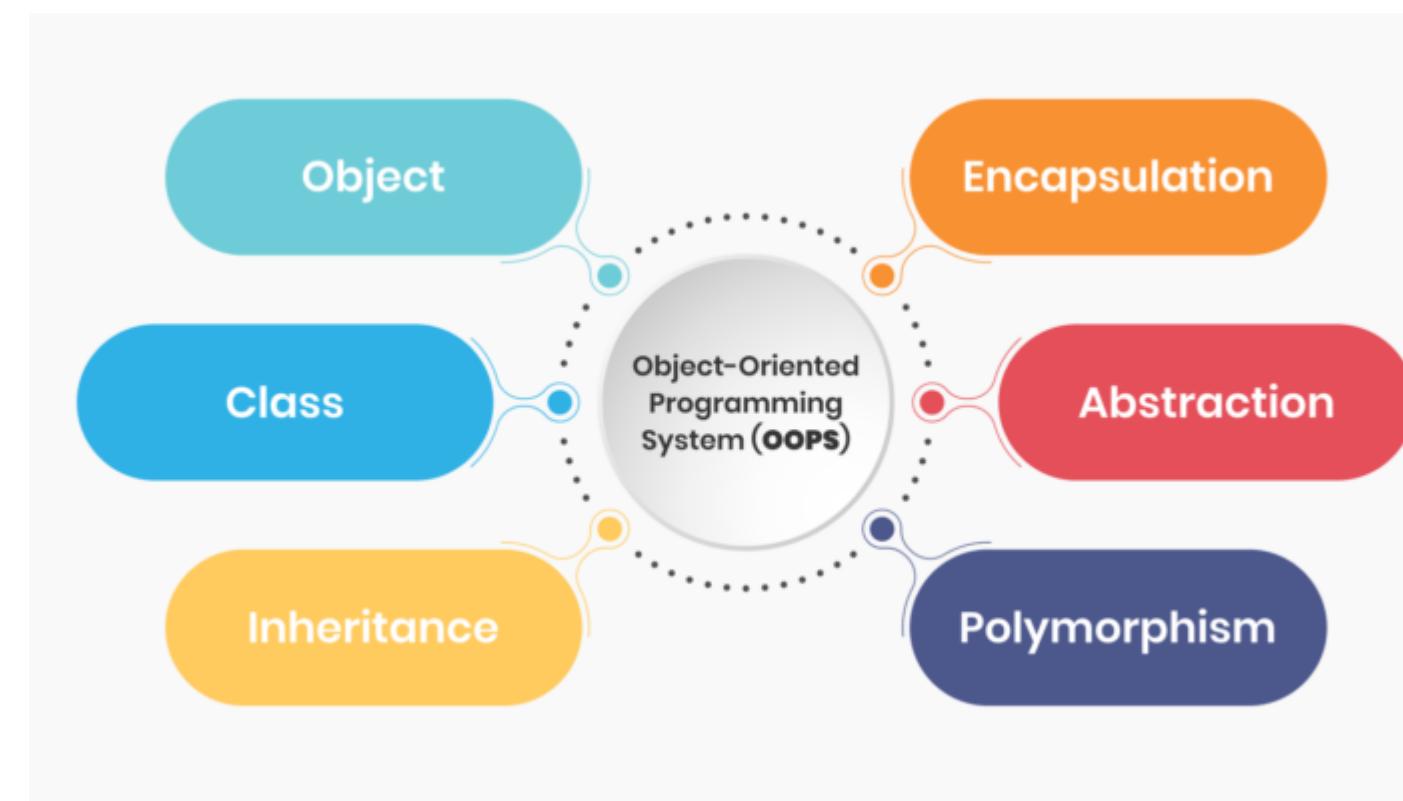
Al aplicar programación modular, un problema complejo debe ser dividido en otros subproblemas más simples aún.

Un módulo es cada una de las partes de un programa que resuelve uno de los subproblemas.

La programación orientada a objetos (POO) se basa en la programación modular, aunque supone una ruptura respecto a ésta al introducir el concepto de objeto, lo que supone un gran avance en términos de modularización y reutilización de código.

### Fundamentos de la POO

- **Abstracción:** es el pilar de la POO, un principio por el cual se aísla toda aquella información que no resulta relevante a un determinado nivel de conocimiento. Consiste en captar las características esenciales de un objeto, así como su comportamiento.
- **Encapsulación:** se centra en ocultar la complejidad de la clase. Significa proteger a los miembros de una clase de un acceso ilegal o no autorizado.
- **Herencia:** es el pilar más fuerte que asegura la reutilización de código. Permite la definición de nuevas clases a partir de otras ya existentes.
- **Polimorfismo:** posibilita que una misma operación pueda realizar tareas diferentes, dependiendo del tipo de objeto sobre el cual se ha invocado.



### Concepto de objeto

Entonces, ¿qué es un objeto? Al igual que en el mundo real, un objeto es cualquier cosa. Un objeto puede ser una cosa física, como un coche, o una cosa mental, como una idea. Puede ser algo natural, como un animal, o algo artificial hecho por el hombre, como un cajero automático. Un programa que administra un cajero automático involucraría cuentas bancarias y objetos de cliente. Un programa de ajedrez involucraría un objeto tablero y objetos piezas de ajedrez.

### Atributos y acciones

#### Atributos

Al igual que con los objetos reales, los objetos de nuestros programas tienen ciertos atributos o propiedades característicos. Por ejemplo, un objeto de cajero automático tendría una cantidad actual de efectivo que podría dispensar. Un objeto pieza ajedrez puede tener un par de atributos de fila y columna que especifiquen su posición en el tablero de ajedrez. Observe que los atributos de un objeto son en sí mismos objetos. El atributo de efectivo del cajero automático y los atributos de fila y columna de la pieza de ajedrez son números.

A veces nos referimos a la colección de atributos y valores de un objeto como su **estado**.

#### Acciones o Métodos

Además de sus atributos o propiedades, los objetos también tienen acciones o comportamientos característicos. Como ya dijimos, los objetos en los programas son dinámicos. Hacen cosas o les hacen cosas.

Por ejemplo, en un programa de ajedrez, `ChessPiece` tiene la capacidad de `moveTo()` a una nueva posición en el tablero de ajedrez. De manera similar, cuando un cliente presiona el botón "Saldo actual" en un cajero automático, esto le está diciendo al cajero automático que informe () el saldo bancario actual del cliente. (Observe cómo usamos paréntesis para distinguir acciones de objetos y atributos).

Las acciones asociadas con un objeto se pueden utilizar para enviar mensajes a los objetos y recuperar información de los objetos. Un mensaje es el paso de información o datos de un objeto a otro. En este ejemplo, le decimos a `peón1`: Pieza de ajedrez que se mueva a (3,4).

```
chessPiece.moveTo(3, 4);
```

Los números 3 y 4 en este caso son argumentos que le dicen al peón a qué casilla moverse. (Un tablero de ajedrez tiene 8 filas y 8 columnas y cada cuadrado se identifica por sus coordenadas de fila y columna). En general, un argumento es un valor de datos que especializa el contenido de un mensaje de alguna manera.

Tabla de contenidos
Fundamentos de la POO
Concepto de objeto
Atributos y acciones
Atributos
Acciones o Métodos
Características básicas
Creación y destrucción de objetos
Creación
Destrucción
Uso de objetos: acceso a atributos y métodos
Ejemplo completo de clase
Coché

Responder a un mensaje o realizar una acción a veces provoca un cambio en el estado de un objeto. Por ejemplo, después de realizar `moveTo (3,4)`, el peón estará en una casilla diferente. Su posición habrá cambiado.

Por otro lado, algunos mensajes (o acciones) no modifican el estado del objeto. Informar el saldo de la cuenta bancaria del cliente no cambia el saldo.

## Características básicas

- **Estado:** está representado por atributos de un objeto.
- **Comportamiento:** se representa mediante métodos de un objeto. También refleja la respuesta de un objeto con otros objetos.
- **Identidad:** le da un nombre único a un objeto y permite que un objeto interactúe con otros objetos.

## Creación y destrucción de objetos

### Creación

Para crear un objeto utilizamos la palabra reservada `new`, que asigna memoria del Heap. Se usa el nombre de la clase (constructor) seguido por paréntesis. Se le llama instanciar un objeto.

```
ATM atm = new ATM();
```

### Destrucción

En Java no es posible destruir objetos de forma explícita, los objetos se destruyen de forma automática por el recolector de basura. Java busca objetos inalcanzables y los destruye, normalmente cuando falta memoria. Los convierte de nuevo en memoria binaria no utilizada.

## Uso de objetos: acceso a atributos y métodos

Para acceder a los atributos y métodos de un objeto utilizamos la notación `"."` detrás del nombre del objeto. El objeto debe ser creado previamente sino dará error de compilación.

```
double cantidad = atm.efectivo;
atm.mostrarEfectivo();
```

Más adelante veremos la visibilidad de los métodos y atributos de los objetos.

## Ejemplo completo de clase Coche

```
public class Coche {
    //atributos, campos o estado
    private String marca;
    private String modelo;
    private int km;
    private LocalDate fabricacion;

    //acciones o comportamiento
    public void arrancar(){}
    public void frenar() {}
    public void repostar(){}
}
```