

Teoría JAVA I

¡Hola! 🙌 Te damos la bienvenida a *“Introducción a la programación backend con Java”*.

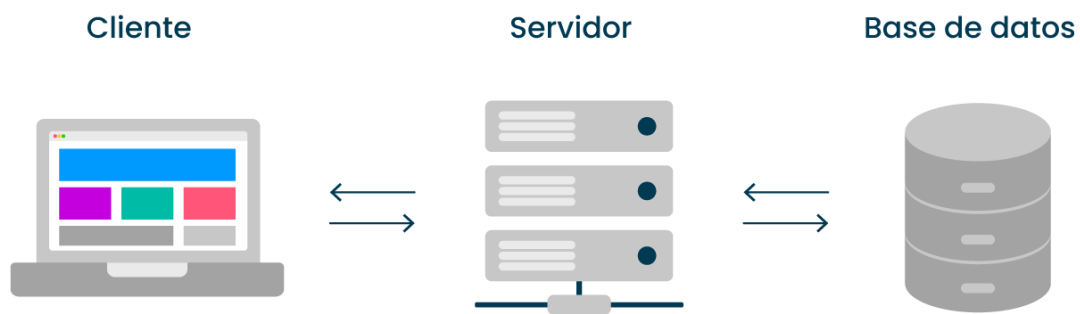
Durante esta etapa, iremos paso a paso para que puedas crear pequeños programas que te ayudarán a dominar el lenguaje Java y sentar las bases necesarias para luego construir un servicio backend.

Exploraremos qué hace un programador backend y nos adentraremos en el fascinante mundo de la programación con Java. Java es un lenguaje potente y versátil que se utiliza en una amplia variedad de sistemas y aplicaciones en todo el mundo. **Daremos los primeros pasos para familiarizarnos con el entorno de desarrollo de Java, descubriremos su sintaxis, aprenderemos los conceptos y estructuras de código fundamentales, y finalizaremos creando nuestro primer programa en Java.**

Es emocionante todo lo que aprenderemos, así que ¡empecemos esta increíble aventura!

Servicios backend

Un programador backend tiene la responsabilidad de recibir datos, procesarlos y almacenarlos en una *base de datos*. Si el cliente lo solicita, también se encarga de buscar y procesar esos datos, entregándolos en el formato requerido.



Por ejemplo, cuando enviamos información desde un formulario para que sea procesada y almacenada fuera del navegador, necesitamos un **servidor backend** capaz de recibir, procesar y almacenar los datos, y luego devolver una respuesta.

Sin un servidor backend, nuestras páginas HTML serían simplemente páginas estáticas sin funcionalidad.

En este curso, *aprenderemos a crear programas que proporcionen un servicio backend para una aplicación frontend.*

Introducción a Java

Java es un lenguaje de programación ampliamente utilizado en el desarrollo de **aplicaciones web backend**. Se ha ganado la confianza de la industria debido a su estabilidad, confiabilidad, robustez y escalabilidad.

Java fue diseñado bajo el **paradigma orientado a objetos**, lo que nos permite modelar y representar objetos del mundo real de manera eficiente en código. En Java, estos objetos se definen como "clases".

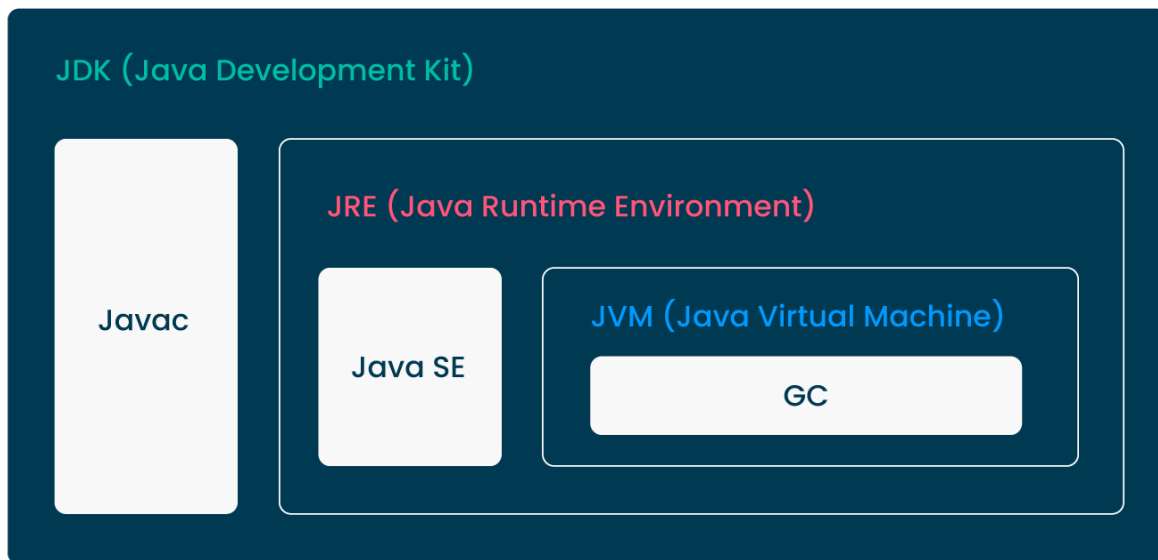
Por ejemplo, si queremos modelar un automóvil, podemos crear una clase "Automóvil" que tenga atributos como marca, modelo y color, y métodos que realicen acciones como acelerar o frenar. De esta manera, podemos organizar y estructurar nuestro código de manera lógica y comprensible.

Además, el enfoque orientado a objetos nos permite reutilizar código de manera efectiva. Por ejemplo, si tenemos una clase "Vehículo" con métodos y atributos comunes a varios tipos de vehículos, podemos heredar esa clase en clases más específicas como "Automóvil" o "Motocicleta", y así aprovechar el código existente sin necesidad de escribirlo nuevamente.

Java se destaca por su lema "Write Once, Run Anywhere" ("escribe una vez, corre dónde sea"). Esto significa que, a través de la *Java Virtual Machine* (JVM), los programas escritos en Java pueden ejecutarse en diferentes sistemas operativos sin necesidad de realizar modificaciones adicionales. La JVM interpreta y ejecuta el código Java en *bytecode*, que es un conjunto de instrucciones específicas de la máquina virtual.

Es importante mencionar que Java utiliza un conjunto de siglas y conceptos que son relevantes en su entorno. Algunos de ellos son:

- **JVM (Java Virtual Machine):** Es una máquina virtual que interpreta y ejecuta el código Java, proporcionando un entorno de ejecución independiente de la plataforma.
- **Java SE (Java Standard Edition):** Es el conjunto de clases y APIs base para desarrollar aplicaciones Java, proporcionadas por el propio lenguaje.
- **JRE (Java Runtime Environment):** Es un entorno de ejecución que contiene la JVM y las bibliotecas necesarias para ejecutar aplicaciones Java.
- **JDK (Java Development Kit):** Es un conjunto de herramientas que proporciona todo lo necesario para desarrollar, compilar y depurar aplicaciones Java, incluyendo el compilador de Java (javac) y el JRE.
- **Javac:** Es el compilador de Java, una herramienta proporcionada por el JDK que se utiliza para convertir el código fuente de Java en *bytecode*, que puede ser ejecutado por la JVM.
- **GC (Garbage Collector):** Es un componente en Java que administra automáticamente la memoria liberando objetos no utilizados. Evita fugas de memoria y mejora la eficiencia del programa. Identifica y elimina objetos no referenciados, liberando recursos para otros objetos. El GC se encarga de la recolección de basura sin intervención manual del programador.



Instalación de Java

Ahora, continuaremos con el proceso de instalación y configuración de Java, un paso fundamental para poder desarrollar aplicaciones en este lenguaje. Te guiaremos a través de los pasos necesarios para descargar Java, configurar las variables de entorno y verificar la instalación.

1. Para descargar Java (versión del openjdk provista por la Eclipse Foundation) de acuerdo a tu sistema operativo vas a poder ingresar al siguiente link:

👉 <https://adoptium.net/es/temurin/releases/?version=20>

Ver el video: 🧑🏻 [Descarga de JDK | JAVA | Egg](#)

💡 Nosotros usaremos la versión 20 de Java que salió en marzo de 2023 porque sus mejoras aparecerán también en la versión 21 próxima a salir. Puedes saber más sobre el soporte de las versiones de java entrando al siguiente 👉 [link](#).

2. Luego lo vamos a instalar y configurar las variables de entorno en Java:

Ver el video: 🧑🏻 [Instalación de JDK | JAVA | Egg](#)

💡 *Para chequear que todo esté correctamente instalado, puedes abrir la terminal en tu ordenador y colocar **"java -version"**.*

3. Para finalizar con el seteo del ambiente de trabajo, vamos a abrir *Visual Studio Code*, iremos a la parte de Aplicaciones y haremos clic en instalar **"Extension Pack for Java"**.

Ver el video:  [Instalar JAVA extension pack | JAVA | Egg](#)

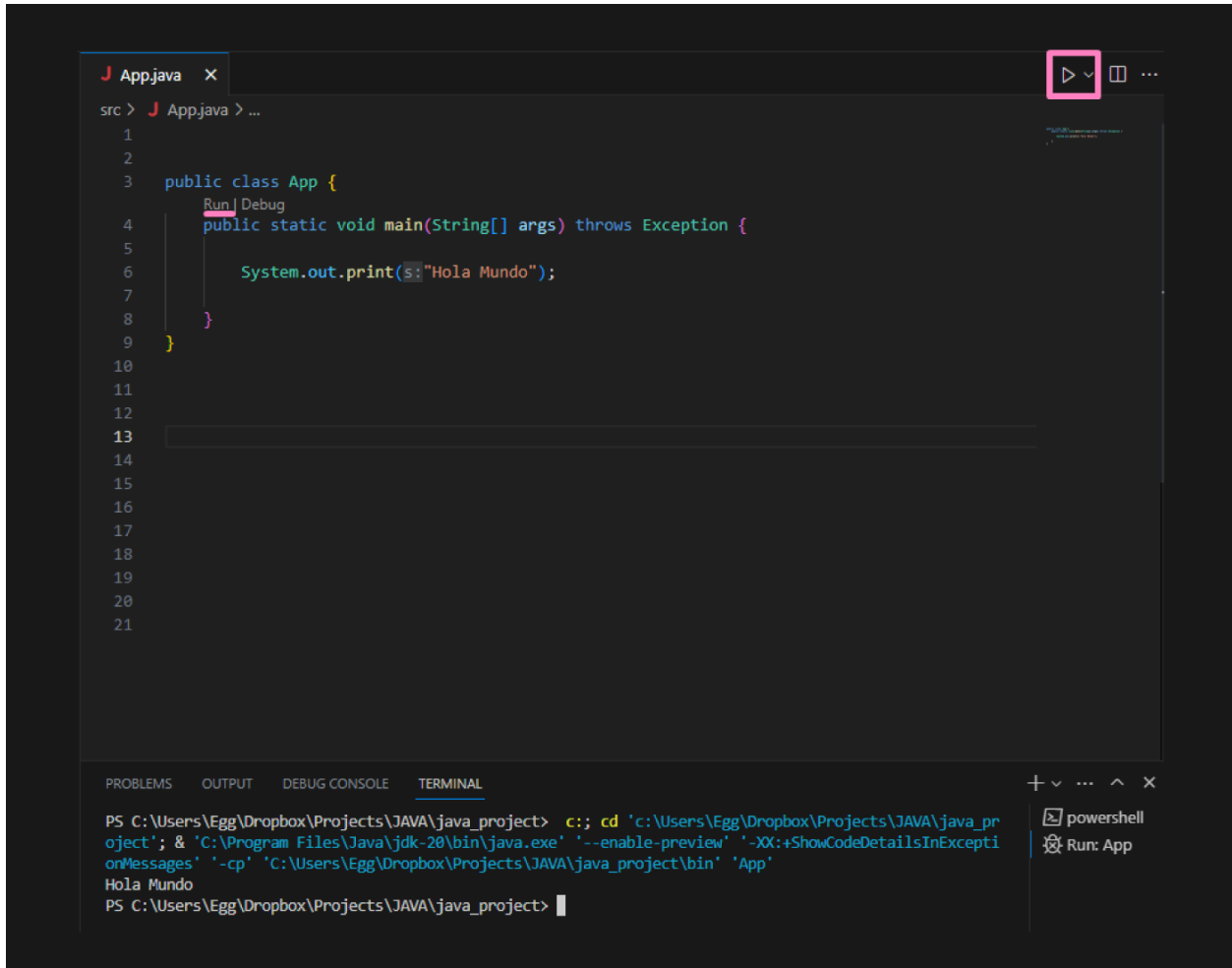
Una vez completada esta instalación, hemos configurado nuestro entorno de trabajo y ¡está todo listo para comenzar a programar!

Programación con Java

Llegamos al emocionante momento de entender cómo crear nuestro propio programa en Java. Para comenzar, necesitaremos crear un archivo llamado **"MiPrimerPrograma.java"** y compilarlo, tal como se mostró en el video de Visual Studio.

💡 *La compilación es un proceso en el que el código fuente escrito en lenguaje Java (como el que crearemos en el **"MiPrimerPrograma.java"**) se traduce a un lenguaje que la computadora pueda entender y ejecutar. Es como traducir un texto escrito en un idioma humano a un idioma que una máquina pueda entender.*

Luego, una vez que presionemos el botón de reproducción ("play"), podremos ver el resultado en la consola:



```
src > J App.java > ...  
1  
2  
3 public class App {  
4     public static void main(String[] args) throws Exception {  
5  
6         System.out.print(s:"Hola Mundo");  
7     }  
8  
9 }  
10  
11  
12  
13  
14  
15  
16  
17  
18  
19  
20  
21
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

```
PS C:\Users\Egg\Dropbox\Projects\JAVA\java_project> c:: cd 'c:\Users\Egg\Dropbox\Projects\JAVA\java_pr  
ject'; & 'C:\Program Files\Java\jdk-20\bin\java.exe' '--enable-preview' '-XX:+ShowCodeDetailsInExcepti  
onMessages' '-cp' 'C:\Users\Egg\Dropbox\Projects\JAVA\java_project\bin' 'App'  
Hola Mundo  
PS C:\Users\Egg\Dropbox\Projects\JAVA\java_project> |
```

Run: App

Un programa es una serie de instrucciones escritas que una computadora puede ejecutar para procesar datos y producir un resultado.

Para comprenderlo mejor, imaginemos una analogía con algo familiar, como una receta de cocina: Una receta de cocina es una serie de instrucciones escritas que un chef sigue para procesar ingredientes y crear un plato delicioso. Del mismo modo, la computadora sigue instrucciones línea por línea, pero en lugar de trabajar con ingredientes, trabaja con datos para procesarlos y generar un resultado.

Sin embargo, a diferencia de la cocina donde el resultado es un alimento, en el caso de las computadoras, el resultado es información. Por ejemplo, en este momento tu computadora está ejecutando un programa que muestra esta información en tu pantalla (en realidad, son varios programas trabajando juntos).

Ahora, **veamos la sintaxis básica de un programa en Java:**

```
public class App {  
    public static void main(String[] args) throws Exception {  
        System.out.println("Hola Mundo");  
    }  
}
```

Dentro de este código, hay varias partes que nos permiten ejecutar un programa. En este caso las vamos a separar en dos:

- Parte 1:

```
public class App {  
    public static void main(String[] args) throws Exception {  
    }  
}
```

La “*parte 1*” la abordaremos más adelante, pero por ahora, debemos saber que nuestros programas se ejecutarán dentro de este código.

- Parte 2:

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

En cuanto a la “*parte 2*”, la sintaxis “**System.out.println();**” indica que cualquier contenido entre comillas dentro de los paréntesis se imprimirá en la consola. “**println**” significa “*print line*”, lo que significa que lo impreso aparecerá en una nueva línea, mientras que “**print**” imprimirá en la misma línea.

Variables

Las **variables** desempeñan un papel fundamental en la programación, ya que **actúan como contenedores donde podemos almacenar información, conocida como datos.**

Siguiendo con nuestra analogía de las recetas, al igual que los ingredientes se almacenan en recipientes, los datos también necesitan un lugar donde se puedan guardar para poder manipularlos y darle instrucciones a la computadora sobre qué hacer con ellos. En programación, estos contenedores se llaman variables.

Imagina que cada variable es como un recipiente que contiene un dato específico, y además **tiene una etiqueta que indica qué tipo de dato puede almacenar.** Esta etiqueta es importante, ya que ayuda a la computadora a entender cómo tratar y manipular la información dentro de cada recipiente.

De esta manera, **al utilizar variables en nuestros programas, podemos almacenar y manejar diferentes tipos de datos de manera organizada y eficiente.**

💡 *Al programar en Java, es crucial asignar y almacenar los datos en variables del tipo correcto para garantizar un funcionamiento adecuado.*

El siguiente ejemplo ilustra esta limitación, destacando que *no se puede asignar un valor de texto a una variable destinada a números:*

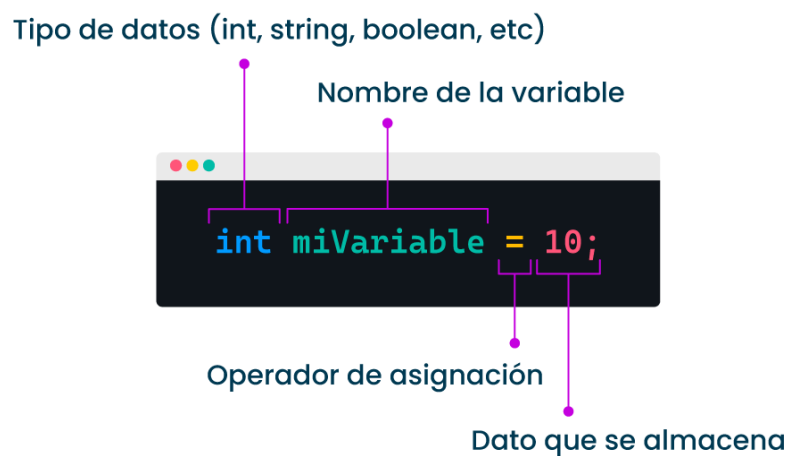


Sentencia de declaración de una variable

💡 *Las sentencias son unidades mínimas de ejecución en un programa y se componen de reglas gramaticales conocidas como sintaxis.*

La sintaxis define cómo se deben combinar las palabras, los símbolos y los elementos gramaticales para formar instrucciones y expresiones válidas.

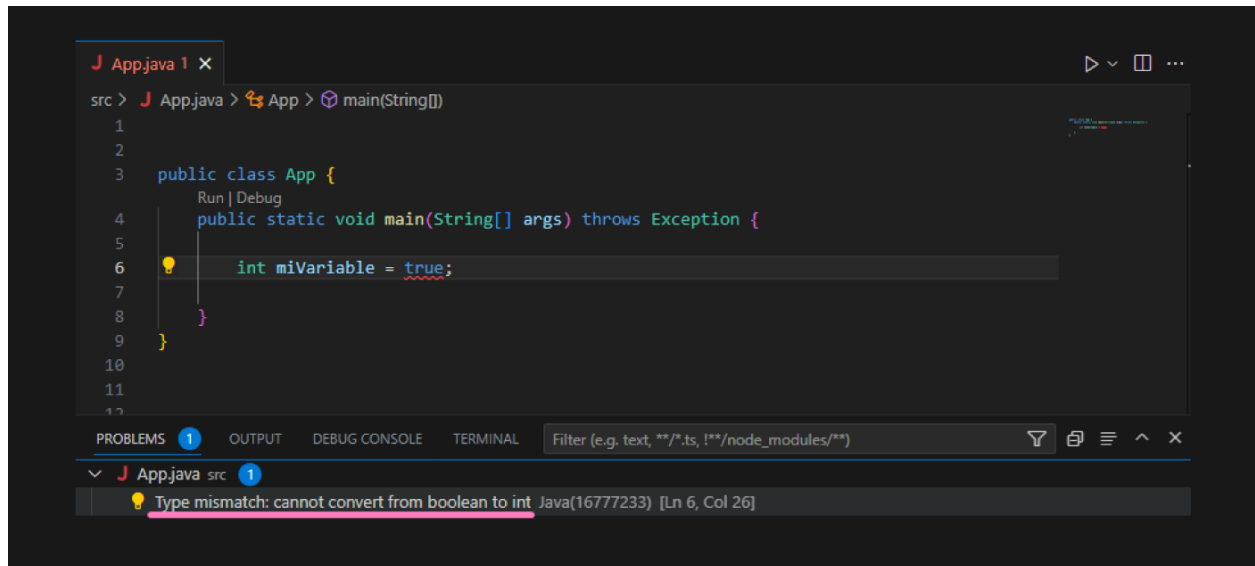
La declaración de una variable en un programa sigue una estructura específica. Por ejemplo, **una sentencia de declaración en Java se escribe de la siguiente manera:**



Como podemos observar en la imagen, esta sentencia consta de cinco partes:

- **int** → Indica el *tipo de dato* que deseamos almacenar en la variable.
- **miVariable** → Es el nombre que le asignamos a nuestra variable.
- **=** → Es el *operador de asignación* utilizado para asignar un valor a la variable.
- **15** → Es el valor que le asignamos a la variable.
- **;** → Indica el fin de la sentencia. *Es esencial incluir el punto y coma al final de cada sentencia para evitar errores.*

! Importante: Si intentamos asignar un valor de un tipo distinto a un **int** (**números enteros**) en Java, se generará un error. Puedes comprobar esto utilizando la siguiente sentencia: `"int miVariable = true;"`.



El error que podemos observar en la imagen de arriba indica que no se puede pasar de un tipo de dato booleano a uno entero. Esto es justamente porque le indicamos que el valor iba a ser **int** (número entero).

Declarar e inicializar una variable

En programación es importante entender la diferencia entre *declarar* e *inicializar una variable*. Veamos en qué consiste cada uno de estos conceptos.

- Para **declarar una variable** en Java, primero debemos especificar el tipo de dato que va a contener y luego asignarle un nombre. Por ejemplo:

```
int numero1;
```

En este caso, hemos creado una variable de tipo **int** (número entero), pero aún no le hemos asignado ningún valor. *Esta es la etapa de declaración.*

- La inicialización de una variable se refiere a darle un valor inicial cuando no tiene ninguno.

En Java, podemos **inicializar una variable** utilizando el *operador de asignación* "=" y proporcionando un *valor* que sea compatible con el *tipo de dato* declarado. Ejemplo:

```
numero1 = 17;
```

También **es posible combinar la declaración y la inicialización en una misma línea**, como vimos anteriormente. Por ejemplo:

```
int numero1 = 17;
```

Variables vs. constantes

Cuando definimos un dato como **constante**, le asignamos un valor por primera vez y luego no es posible cambiarlo. Una vez que el dato está inicializado, su valor no puede ser modificado de ninguna manera.

Para *declarar constantes en Java*, utilizamos la palabra reservada **"final"** y las inicializamos en la misma sentencia de la siguiente forma:

```
final int NUMERO_15 = 15;
```

Es importante tener en cuenta lo siguiente: **al declarar una constante, debemos asignarle un valor inmediatamente, es decir, debemos inicializarla de inmediato**. De lo contrario, nuestro código no funcionará correctamente.

Si estás observando detenidamente el código, es probable que hayas notado que **los nombres de las variables y las constantes se escriben de manera diferente**. Esto se debe a una convención seguida por la comunidad de programadores de Java. Estas convenciones son reglas que garantizan que el código sea legible y consistente, lo cual facilita la colaboración entre desarrolladores.

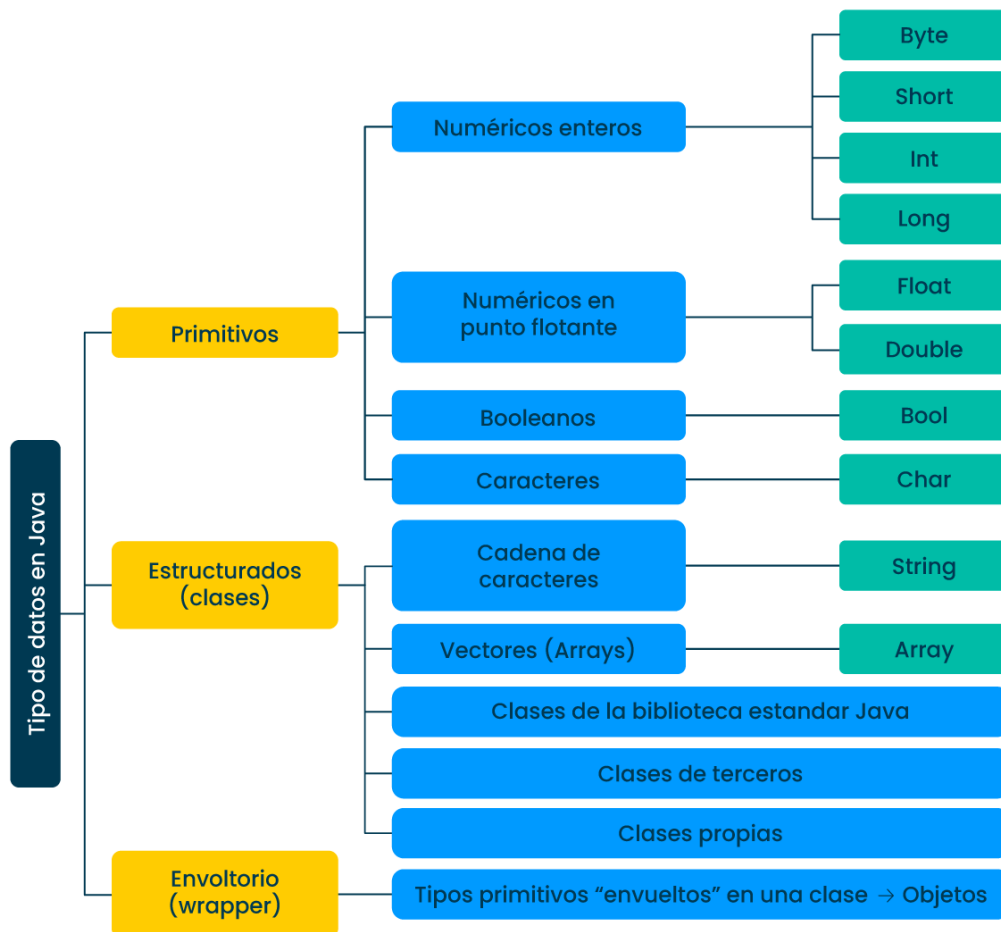
A continuación, mencionaremos dos convenciones comunes:

- **Las variables se escriben en camelCase.** Se utiliza una combinación de letras minúsculas y mayúsculas, donde la primera letra es minúscula y las palabras siguientes comienzan con una letra mayúscula. Por ejemplo: *miVariable*.
- **Las constantes se escriben en UPPER_CASE.** Se utilizan letras mayúsculas y las palabras se separan con guiones bajos "_". Por ejemplo: *VALOR_MAXIMO*.

! A partir de ahora, en el transcurso del curso, **utilizaremos la palabra "variable" para referirnos tanto a las variables como a las constantes**, con el fin de facilitar la lectura del material.

Tipos de datos

Hasta ahora hemos mencionado que las variables actúan como contenedores para almacenar diferentes tipos de datos, pero ¿qué tipos de datos existen exactamente? Veamos:



En el [material complementario](#), te proporcionamos una lista completa de todos los tipos de datos disponibles para que puedas consultarla en cualquier momento que lo necesites, y a continuación, nos enfocaremos en revisar en detalle los tipos de datos más comunes.

Repasemos en detalle los **tipos de datos primitivos** de Java. Son 8, y se encargan de representar en el lenguaje los tres tipos básicos que acabamos de nombrar, y también son *palabras reservadas*:

- **Datos booleanos:** *boolean*. Los únicos dos valores que acepta una variable booleana son *true* o *false*:

```
boolean booleana = true;
boolean booleana = false;
```

- **Datos de carácter de texto:** *char*. Los caracteres sólo pueden contener una letra, y la misma tiene que estar rodeado de comillas simples (las comillas dobles se usan para otro tipo de dato que veremos pronto):

```
char character = 'a';
```

- **Datos numéricos:** *byte*, *short*, *int*, *long*, *float*, *double*. Para los datos enteros seguiremos usando *int*. Java de forma predeterminada manipula esos números como tipos de datos *int*, en cambio cuando se usa la notación con "." para representar un número decimal, Java los manipula por defecto como "double".
 - Para forzar que los números enteros sean tratados como tipos de datos *float* o *double* se los puede escribir con una "D" o una "F" al final.

```
int miNumeroInt = 123;
int miNumeroFloat = 123.5;
int miNumeroDouble = 191.20000000000002;
```

- Para forzar que los números enteros sean tratados como tipos de datos *float* o *double* se los puede escribir con una "D" o una "F" al final.

```
int operacionFloat = 19f * 10f;
int operacionDouble = 19d * 10d;
```

La clase Scanner

La clase Scanner es utilizada en Java para obtener la entrada del usuario desde la consola. Al trabajar con esta clase, es importante comprender los conceptos de clase e instancia. Una clase se puede visualizar como un plano o una plantilla que define las características y comportamientos de un objeto. Al crear una instancia de una clase, se crea un objeto específico basado en esa plantilla.

! *No te preocupes si aún esto no queda tan claro, ahora vamos a ver cómo pedir un dato a un usuario y más adelante vamos a profundizar mucho más en clases y objetos.*

Con la ayuda de la clase Scanner vamos a poder solicitar que se ingresen datos por consola y el programa se detenga hasta que se ingrese la información:

Ver video:  [Scanner | JAVA | Egg](#)

La clase Scanner nos permite solicitar datos al usuario y detener la ejecución del programa hasta que se ingrese la información requerida. Para utilizar la clase Scanner, es necesario importarla al comienzo del archivo de Java mediante la instrucción:

```
import java.util.Scanner;
```

Después de importarla, se puede crear una instancia de la clase Scanner para comenzar a obtener la entrada del usuario. Usualmente, se crea una instancia asociada al flujo de entrada estándar, que es el teclado. Por ejemplo:

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

Una vez que se tiene una instancia de Scanner, se pueden utilizar sus métodos para leer los datos ingresados por el usuario. Algunos de los métodos más comunes son:

- **nextBoolean():** Lee un valor booleano (true o false) desde la entrada.
- **nextInt():** Lee un número entero desde la entrada.
- **nextDouble():** Lee un número de tipo double desde la entrada.
- **nextLine():** Lee una línea completa de texto desde la entrada.

A continuación, se muestra un ejemplo de cómo utilizar la clase Scanner para obtener un número entero ingresado por el usuario:

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

System.out.print("Ingresa un número entero: ");
int numero = miScanner.nextInt();

System.out.println("El número ingresado es: " + numero);
```

En este ejemplo, el programa solicita al usuario ingresar un número entero utilizando el método **nextInt()** de la instancia de Scanner. Luego, se asigna ese número a la variable "numero" y se imprime en pantalla.

A medida que avancemos, profundizaremos más en el tema de clases y objetos, lo que te permitirá comprender mejor el concepto de la clase Scanner y su utilización en la obtención de datos de entrada.

Operaciones y Operadores

Los operadores son símbolos que se utilizan para manipular los datos y producir nuevos valores. Los tipos más comunes de operadores que existen y explicaremos a continuación son:

- **Operadores de asignación** → Se utiliza para asignar un valor a una variable
- **Operadores aritméticos** → Se utilizan para hacer operaciones matemáticas
- **Operadores de comparación** → Se utilizan para hacer comparaciones entre números
- **Operadores lógicos** → Se utilizan para combinar y evaluar expresiones booleanas, y su resultado es un valor booleano.

Operador de asignación

Lo vimos al momento de asignarle el valor a una variable y es el símbolo "=".

```
int miNumero = 123;
```

Operadores aritméticos

- **Operador de suma "+":** Se utiliza para sumar dos números.
- **Operador de resta "-":** Se utiliza para restar dos números.
- **Operador de multiplicación "*":** Se utiliza para multiplicar dos números.
- **Operador de división "/":** Se utiliza para dividir dos números.
- **Operador de resto "%":** Se utiliza para obtener el resto de la división de dos números.

```
int operadorIgualdad = 1+2; // 3
int operacionResta = 2-1; // 1
int operacionMultiplicacion = 1*2; // 2
int operacionDivision = 6/2; // 3
int operacionModulo = 9%3; // 0
```

De los cuatro operadores anteriores, el último merece una explicación adicional. El operador "%", en lugar de devolver el resultado de dividir 9 entre 3 (que sería "3"), nos da el resto de esa operación, que en este caso es "0". Por ejemplo, si probamos con "11 % 3", obtendremos como resultado "2". Este operador es útil para determinar si un número es divisible por otro. Por ejemplo, podemos usar el operador de resto seguido de "2" para verificar si una variable almacena un número par. Si el resultado es "0", significa que la variable contiene un número par, mientras que si el resultado es "1", significa que la variable contiene un número impar.

Operadores de comparación

- **Operador de igualdad "=="**: Se utiliza para determinar si dos valores primitivos son iguales, si lo son el resultado de la operación devuelve un booleano true, si no lo son se devuelve false.
- **Operador distinto que "!="**: Se utiliza para determinar si dos valores son distintos, si son distintos el resultado de la operación devuelve un booleano true, si no lo son se devuelve false.

- **Operador de mayor que ">":** Se utiliza para determinar si un valor es mayor a otro.
- **Operador de menor que "<":** Se utiliza para determinar si un valor es menor a otro.
- **Operador de mayor igual que ">=":** Se utiliza para determinar si un valor es mayor e igual a otro.
- **Operador de menor igual que "<=":** Se utiliza para determinar si un valor es menor e igual a otro.

```
boolean operadorIgualdad = 1 == 2; // false
boolean operadorDistintoQue = 2 != 1; // true
boolean operadorMayorQue = 1 > 2; // false
boolean operadorMenorQue = 6 < 2; // false
boolean operadorMayoroIgualQue = 9 >= 3; // true
boolean operadorMenoroIgualQue = 3 <= 3; // true
```

Operadores lógicos

- **Operador Y "&&":** Este operador se usa con dos expresiones booleanas a ambos lados, si ambas son verdaderas devuelve true, si alguna de las dos es falsa devuelve false.
- **Operador O "||":** Este operador se usa con dos expresiones booleanas a ambos lados, si ambas son falsas devuelve false, si alguna de las dos es verdadera devuelve true
- **Operador de negación "!=":** Este operador se usa para negar una expresión booleana devolviendo el valor booleano contrario.

```
boolean operadorAnd = 5 > 4 && 5 < 10; // true
boolean operadorOr = 0 > 4 || 12 < 10; // false
boolean operadorNegacion = !(0 > 4 || 5 < 10); // false
```

Operador ternario

La estructura del operador ternario es la siguiente:


```
resultado = (condicion) ? valor1 : valor2;
```

Donde a la variable resultado recibirá el *valor1* en el caso de que la condición sea verdadera (true), o bien el *valor2* en el caso de que la condición sea falsa (false).

Así, si queremos calcular el mayor de dos números tendremos un código como el siguiente:

```
String mayor = (5 > 2) ? "mayor" : "menor o igual";  
  
System.out.print(mayor);
```

Es fundamental comprender que cualquier operación que genere un resultado puede ser almacenada en una variable para su posterior uso y manipulación.

En el  **material complementario**, encontrarás el resto de los operadores en Java. Puedes consultarlo más adelante cuando hagamos referencia a alguno de ellos y necesites recordar cómo funcionan.

Material complementario

Tipos de datos de Java:

Tipos de datos primitivos

Java cuenta con un pequeño conjunto de tipos de datos primitivos. Podríamos considerarlos fundamentales, ya que la mayor parte de los demás tipos, los tipos estructurados o complejos, son composiciones a partir de estos más básicos. Estos tipos de datos primitivos sirven para gestionar los tipos de información más básicos, como números de diversas clases o datos de tipo verdadero/falso (también conocidos como "valores booleanos" o simplemente "booleanos").

De estos tipos primitivos, ocho en total, seis de ellos están destinados a facilitar el trabajo con números. Podemos agruparlos en dos categorías:

- **Tipos numéricos enteros.**
- **Tipos numéricos en punto flotante.**

Los primeros permiten operar exclusivamente con números enteros, sin parte decimal, mientras que el segundo grupo contempla también números racionales o

con parte decimal.

- **Tipos numéricos enteros:** En Java existen cuatro tipos destinados a almacenar números enteros. La única diferencia entre ellos es el número de bytes usados para su almacenamiento y, en consecuencia, el rango de valores que es posible representar con ellos. Todos ellos emplean una representación que permite el almacenamiento de números negativos y positivos. El nombre y características de estos tipos son los siguientes (*byte, short, int, long*)
 - **byte:** como su propio nombre denota, emplea un solo byte (8 bits) de almacenamiento. Esto permite almacenar valores en el rango $[-128, 127]$.
 - **short:** usa el doble de almacenamiento que el anterior, lo cual hace posible representar cualquier valor en el rango $[-32.768, 32.767]$.
 - **int:** emplea 4 bytes de almacenamiento y es el tipo de dato entero más empleado. El rango de valores que puede representar va de -2^{31} a $2^{31}-1$.
 - **long:** es el tipo entero de mayor tamaño, 8 bytes (64 bits), con un rango de valores desde -2^{63} a $2^{63}-1$.
- **Tipos numéricos decimales:** Los tipos numéricos en punto flotante permiten representar números tanto muy grandes como muy pequeños además de números decimales. Java dispone de 2 tipos concretos en esta categoría (*float, double*).
 - **float:** conocido como tipo de precisión simple, emplea un total de 32 bits. Con este tipo de datos es posible representar números en el rango de 1.4×10^{-45} a 3.4028235×10^{38} .
 - **double:** sigue un esquema de almacenamiento similar al anterior, pero usando 64 bits en lugar de 32. Esto le permite representar valores en el rango de 4.9×10^{-324} a $1.7976931348623157 \times 10^{308}$.
- **Booleanos y caracteres:** Aparte de los 6 tipos de datos que acabamos de ver, destinados a trabajar con números en distintos rangos, Java define otros dos tipos primitivos más:
 - **boolean:** tiene la finalidad de facilitar el trabajo con valores "verdadero/falso" (booleanos), resultantes por regla general de evaluar expresiones. Los dos valores posibles de este tipo son true y false.
 - **char:** se utiliza para almacenar caracteres individuales (letras, para entendernos). En realidad está considerado también un tipo numérico, si bien su representación habitual es la del carácter cuyo código

almacena. Utiliza 16 bits y se usa la codificación UTF-16 de Unicode.

Tipos de datos estructurados

Los tipos de datos primitivos que acabamos de ver se caracterizan por poder almacenar un único valor. Salvo este reducido conjunto de tipos de datos primitivos, que facilitan el trabajo con números, caracteres y valores booleanos, todos los demás tipos de Java son objetos, también llamados tipos estructurados o "Clases". Los tipos de datos estructurados se denominan así porque en su mayor parte están destinados a contener múltiples valores tanto de tipos más simples, como los primitivos, como valores que son otros objetos. También se les llama muchas veces "tipos objeto" porque se usan para representar objetos. Puede que te suene más ese nombre.

- **Cadenas de caracteres:** Aunque las cadenas de caracteres no son un tipo simple en Java, sino una instancia de la clase String, el lenguaje otorga un tratamiento bastante especial a este tipo de dato, lo cual provoca que, en ocasiones, nos parezca estar trabajando con un tipo primitivo.

Aunque cuando declaramos una cadena estamos creando un objeto, su declaración no se diferencia de la de una variable de tipo primitivo de las que acabamos de ver:

```
String nombreCurso = "Iniciación a Java";
```

Y esto puede confundir al principio. Recuerda: Las cadenas en Java son un objeto de la clase String, aunque se declaren de este modo.

Las cadenas de caracteres se delimitan entre comillas dobles, en lugar de simples como los caracteres individuales. En la declaración, sin embargo, no se indica explícitamente que se quiere crear un nuevo objeto de tipo String, esto es algo que infiere automáticamente el compilador.

Las cadenas, por tanto, son objetos que disponen de métodos que permiten operar sobre la información almacenada en dicha cadena. Así, encontraremos métodos para buscar una subcadena dentro de la cadena, sustituirla por otra, dividirla en varias cadenas atendiendo a un cierto separador, convertir a mayúsculas o minúsculas, etc.

- **Arrays:** Los arrays son colecciones de datos de un mismo tipo. También son conocidos popularmente como "arreglos" (aunque se desaconseja esta última denominación por ser una mala adaptación del inglés).
Un array es una estructura de datos en la que a cada elemento le

corresponde una posición identificada por uno o más índices numéricos enteros.

También es habitual llamar vectores a los arrays de una dimensión y matrices a los arrays que trabajan con dos dimensiones.

Los elementos de un array se empiezan a numerar en el 0, y permiten gestionar desde una sola variable múltiples datos del mismo tipo.

Por ejemplo, si tenemos que almacenar una lista de 10 números enteros, declararíamos un array de tamaño 10 y de tipo entero, y no tendríamos que declarar 10 variables separadas de tipo entero, una para cada número.

- **Tipos definidos por el usuario:** Además de los tipos estructurados básicos que acabamos de ver (cadenas y arrays) en Java existen infinidad de clases en la plataforma, y de terceros, para realizar casi cualquier operación o tarea que se pueda ocurrir: leer y escribir archivos, enviar correos electrónicos, ejecutar otras aplicaciones o crear cadenas de texto más especializadas, entre un millón de cosas más.

Todas esas clases son tipos estructurados también.

Y por supuesto puedes crear tus propias clases para hacer todo tipo de tareas o almacenar información. Serían tipos estructurados definidos por el usuario.

Tipos envoltorio o wrapper

Java cuenta con tipos de datos estructurados equivalentes a cada uno de los tipos primitivos que hemos visto.

Así, por ejemplo, para representar un entero de 32 bits (int) de los que hemos visto al principio, Java define una clase llamada Integer que representa y "envuelve" al mismo dato pero le añade ciertos métodos y propiedades útiles por encima.

Además, otra de las finalidades de estos tipos "envoltorio" es facilitar el uso de esta clase de valores allí donde se espera un dato por referencia (un objeto) en lugar de un dato por valor (para entender la diferencia entre tipos por valor y tipos por referencia lee este artículo. Aunque está escrito para C#, todo lo explicado es igualmente válido para Java).

Estos tipos equivalentes a los primitivos pero en forma de objetos son: Byte, Short, Integer, Long, Float, Double, Boolean y Character (8 igualmente).

Operadores en Java:

Operadores Aritméticos

- **Adición (+):** Suma dos valores. Ejemplo: $a + b$.
- **Sustracción (-):** Resta un valor de otro. Ejemplo: $a - b$.
- **Multiplicación (*):** Multiplica dos valores. Ejemplo: $a * b$.
- **División (/):** Divide un valor por otro. Ejemplo: a / b .
- **Módulo (%):** Obtiene el residuo de una división. Ejemplo: $a \% b$.

Operadores de Asignación

- **Asignación (=):** Asigna un valor a una variable. Ejemplo: $a = b$.
- **Suma y asignación (+=):** Suma y luego asigna el valor. Ejemplo: $a += b$ (equivale a $a = a + b$).
- **Resta y asignación (-=):** Resta y luego asigna el valor. Ejemplo: $a -= b$ (equivale a $a = a - b$).
- **Multiplicación y asignación (*=):** Multiplica y luego asigna el valor. Ejemplo: $a *= b$ (equivale a $a = a * b$).
- **División y asignación (/=):** Divide y luego asigna el valor. Ejemplo: $a /= b$ (equivale a $a = a / b$).
- **Módulo y asignación (%=):** Aplica el módulo y luego asigna el valor. Ejemplo: $a \% = b$ (equivale a $a = a \% b$).

Operadores de Incremento y Decremento

- **Incremento (++):** Aumenta el valor de una variable en 1. Ejemplo: $a++$ o $++a$.
- **Decremento (--):** Disminuye el valor de una variable en 1. Ejemplo: $a--$ o $--a$.

Operadores Relacionales o de Comparación

- **Igual a (==):** Verifica si dos valores son iguales. Ejemplo: $a == b$.
- **No igual a (!=):** Verifica si dos valores no son iguales. Ejemplo: $a != b$.
- **Mayor que (>):** Verifica si un valor es mayor que otro. Ejemplo: $a > b$.
- **Menor que (<):** Verifica si un valor es menor que otro. Ejemplo: $a < b$.
- **Mayor o igual que (>=):** Verifica si un valor es mayor o igual que otro. Ejemplo: $a >= b$.
- **Menor o igual que (<=):** Verifica si un valor es menor o igual que otro. Ejemplo: $a <= b$.

Operadores Lógicos

- **AND lógico (&&):** Devuelve verdadero si ambos operandos son verdaderos.

Ejemplo: `a && b`.

- **OR lógico (||)**: Devuelve verdadero si al menos uno de los operadores es verdadero. Ejemplo: `a || b`.
- **NOT lógico (!)**: Invierte el valor de verdad del operando. Ejemplo: `!a`.

Operador Condicional (Operador Ternario)

- **(?:)**: Actúa como un if-else simplificado. Ejemplo: `a > b ? a : b` (si `a` es mayor que `b`, devuelve `a`, sino devuelve `b`).

Operadores de Bit

- **AND binario (&)**: Realiza una operación AND en cada par de bits. Ejemplo: `a & b`.
 - **OR binario (|)**: Realiza una operación OR en cada par de bits. Ejemplo: `a | b`.
 - **XOR binario (^)**: Realiza una operación XOR en cada par de bits. Ejemplo: `a ^ b`.
 - **Complemento binario (~)**: Invierte todos los bits. Ejemplo: `~a`.
 - **Desplazamiento a la izquierda (<<)**: Desplaza los bits a la izquierda, rellena con ceros a la derecha. Ejemplo: `a << 2`.
 - **Desplazamiento a la derecha (>>)**: Desplaza los bits a la derecha, rellena con el bit más significativo (signo) a la izquierda. Ejemplo: `a >> 2`.
 - **Desplazamiento a la derecha sin signo (>>>)**: Desplaza los bits a la derecha, rellena con ceros a la izquierda. Ejemplo: `a >>> 2`.
-