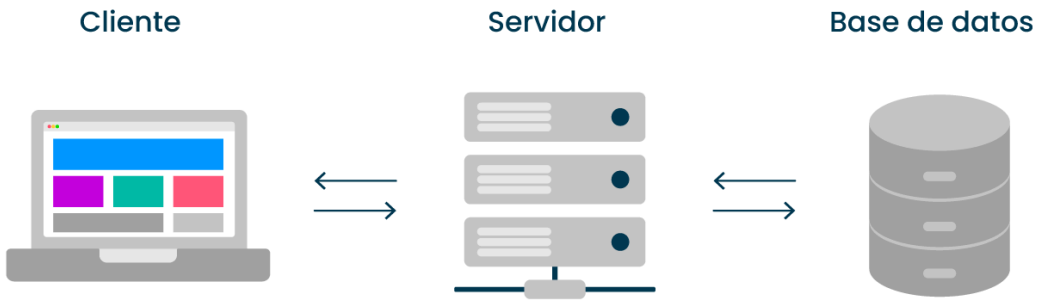
Material descargable

egg

Teoría JAVA I

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".

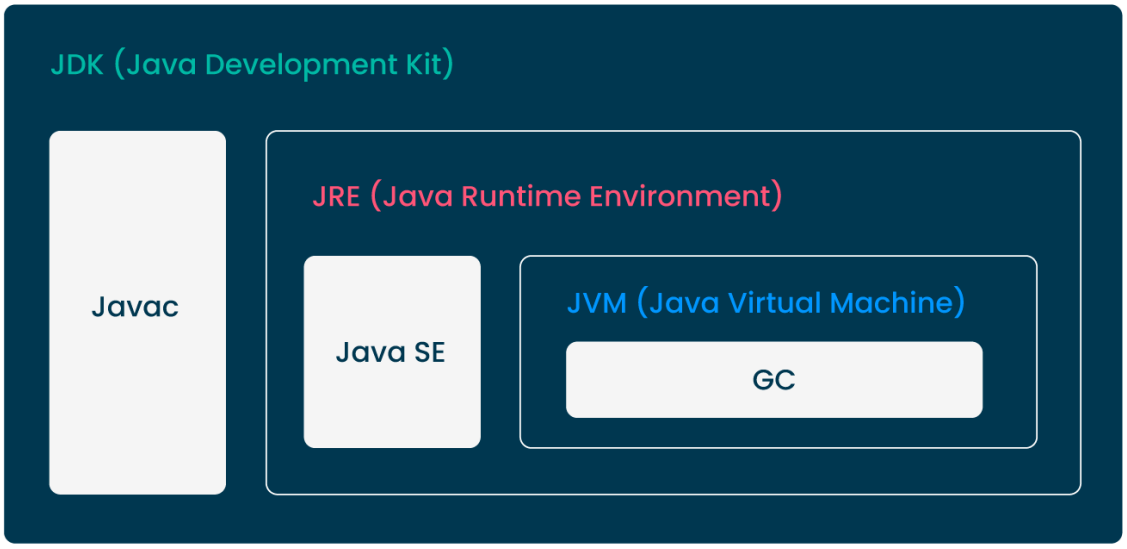
Por ejemplo, si queremos modelar un automóvil, podemos crear una clase "Automóvil" que tenga atributos como marca, modelo y color, y métodos que realicen acciones como acelerar o frenar. De esta manera, podemos organizar y estructurar nuestro código de manera lógica y comprensible.

Además, el enfoque orientado a objetos nos permite reutilizar código de manera efectiva. Por ejemplo, si tenemos una clase "Vehiculo" 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.

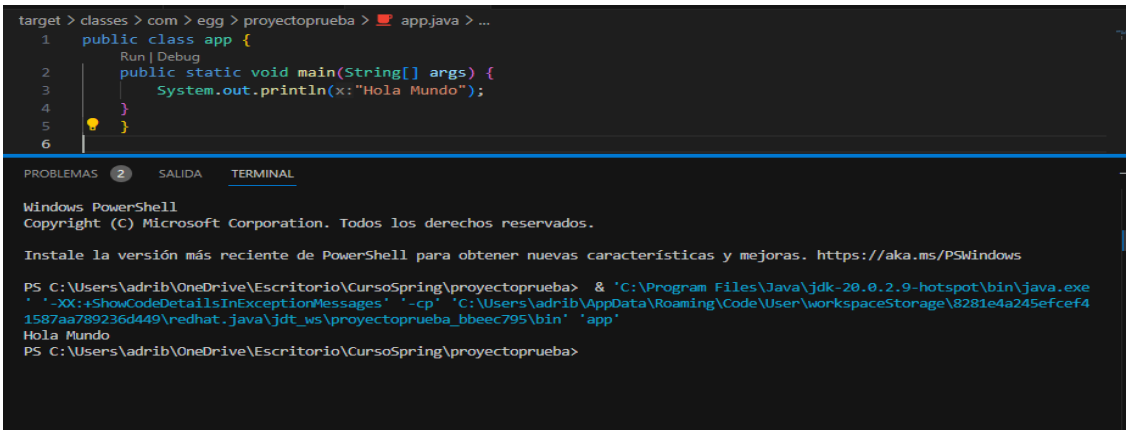


Programación con Java

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:



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.

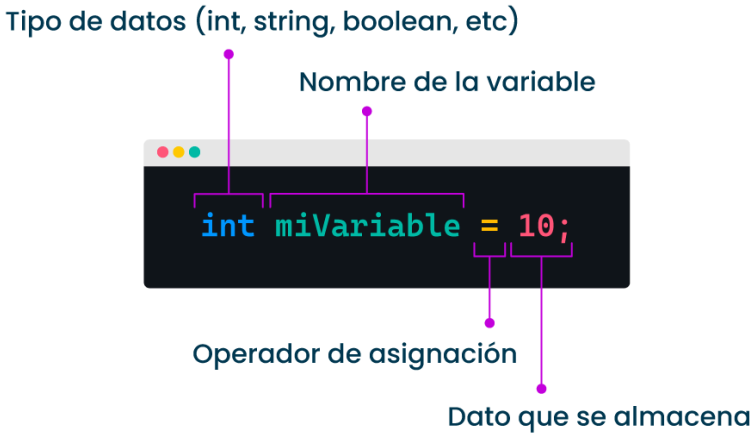
💡*Al programar en Java, es crucial asignar y almacenar los datos en variables del tipo correcto para garantizar un funcionamiento adecuado*.

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.

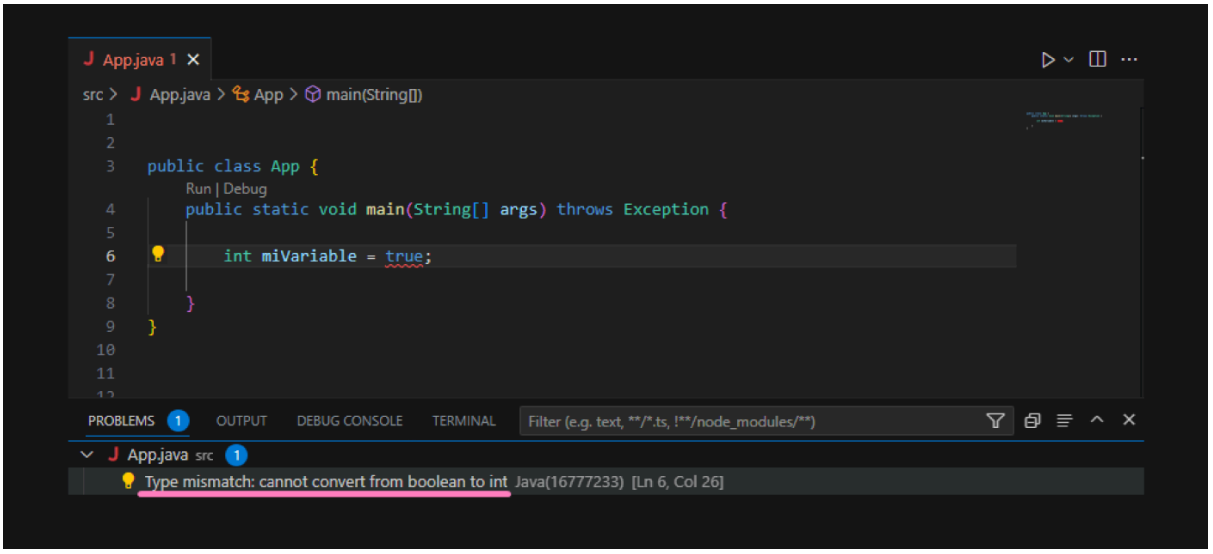
La declaración de una variable en un programa sigue una estructura específica. Por ejemplo, una sentencia de declaración en Java se escribe de la siguiente manera:



Como podemos observar en la imágen, 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.
- **10** → 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.

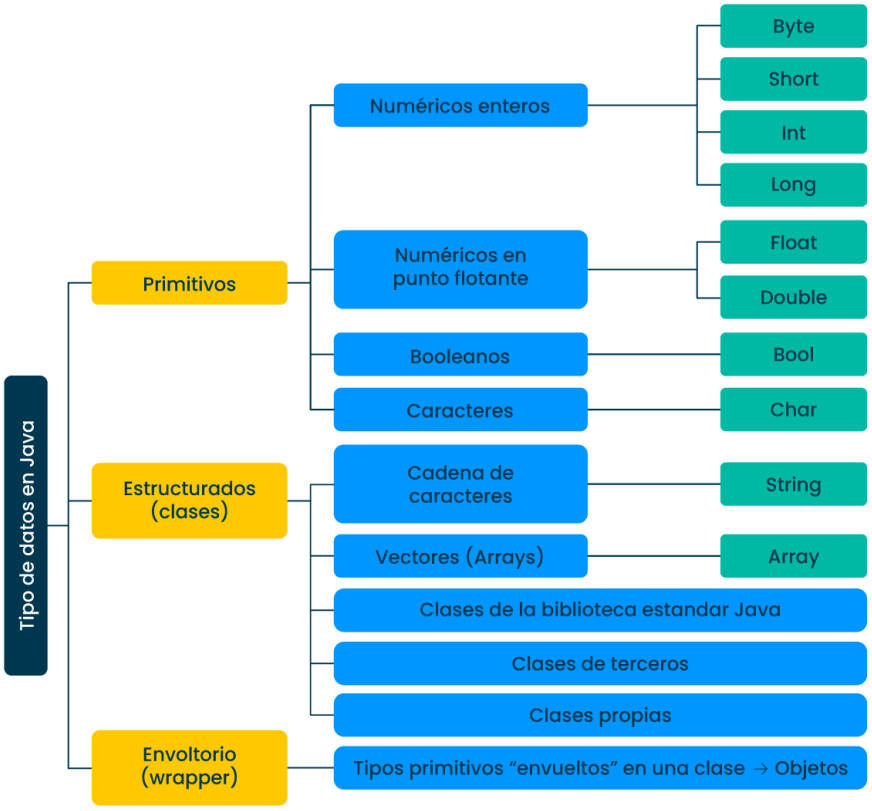
Si estás observando detenidamente el código, es probable que hayas notado que los nombres de las variables y las constantes se escriben de manera diferente. Esto se debe a una convención seguida por la comunidad de programadores de Java. Estas convenciones son reglas que garantizan que el código sea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

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:



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:** utiliza 4 bytes de almacenamiento (32 bits). El rango de valores que puede representar va de **-2^{31}** a **$2^{31} - 1$** (es decir, de -2,147,483,648 a 2,147,483,647).
- **long:** utiliza 8 bytes de almacenamiento (64 bits). El rango de valores que puede representar va de **-2^{63}** a **$2^{63} - 1$** (es decir, de -9,223,372,036,854,775,808 a 9,223,372,036,854,775,807).

- **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 1.4x10-45 a 3.4028235x1038.
  - **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 4.9x10-324 a 1.7976931348623157x10308.

- **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 únicos dos valores que acepta una variable booleana son true o false

```
boolean booleana = true;
boolean booleana = 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. la misma tiene que estar rodeado de comillas simples

```
char caracter = 'a';
```

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.

Datos 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).