

Índice de contenido

Método main y “Hola Mundo” en Java

- 1. Crear un proyecto en IntelliJ*
- 2. Crear una clase usando IntelliJ*
- 3. Sintaxis básica para declarar una clase*
- 4. Archivos .java*
- 5. Sintaxis del método main*
- 6. Declaración del método main*
- 7. El método print de la clase system*
- 8. Imprimir Hola Mundo en Java*
- 9. print vs println*
- 10. Caracteres de escape*

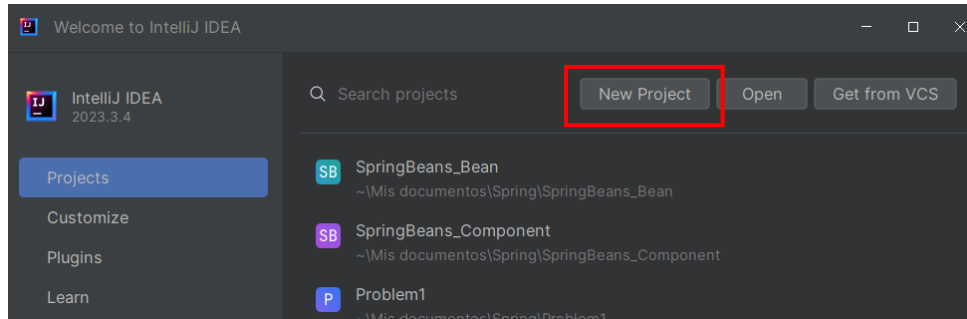
Declaración de una clase en Java

1. Crear un proyecto en IntelliJ

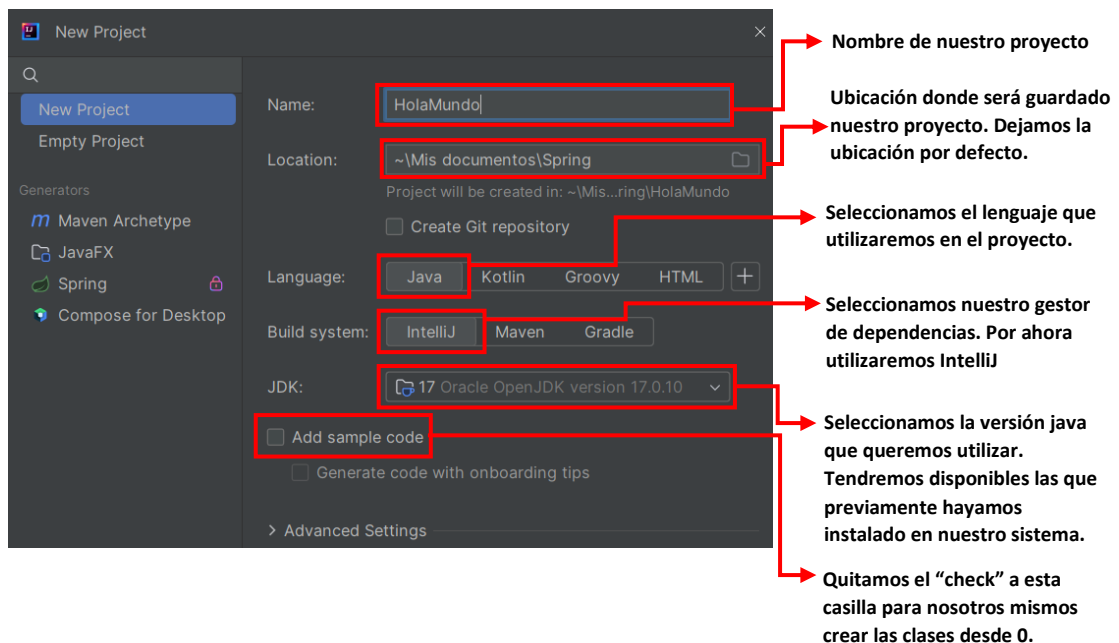
Utilizando IntelliJ vamos a crear nuestro primer proyecto java "HolaMundo".

Crear un proyecto java:

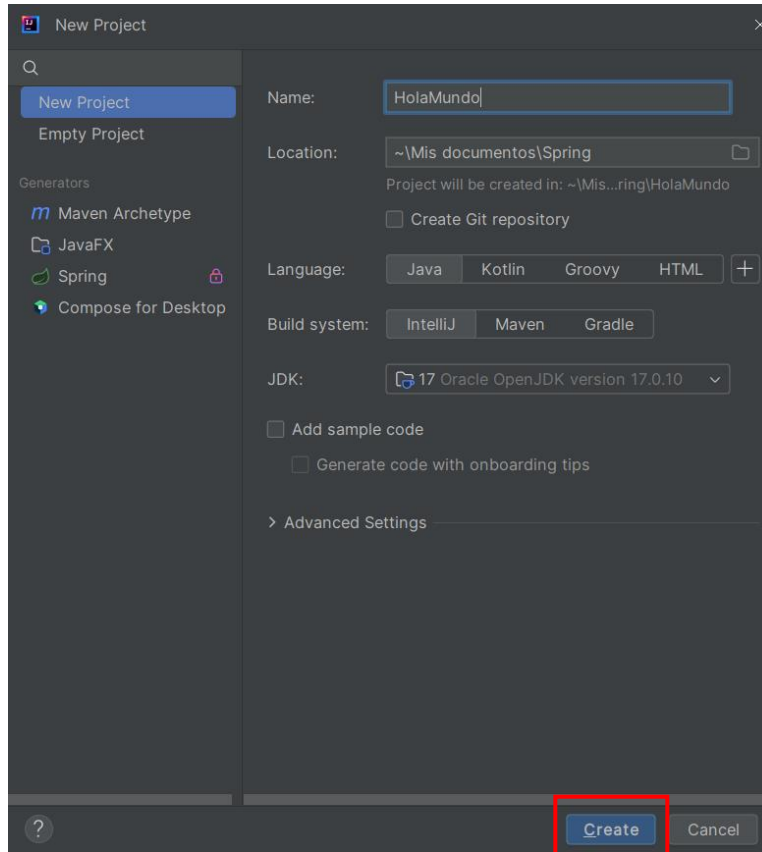
1. Abrimos IntelliJ Idea.
2. Nos aparecerá la página de inicio donde seleccionaremos "Nuevo Proyecto":



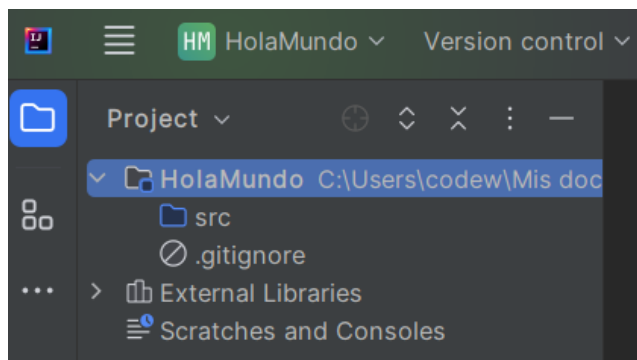
3. En la siguiente página de configuración de nuevo proyecto, tendremos las opciones para definir nuestro proyecto antes de crearlo:



4. Una vez definido las propiedades de nuestro proyecto, clickeamos en “Crear”

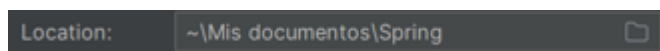


5. Una vez creado el proyecto veremos la estructura básica del proyecto:

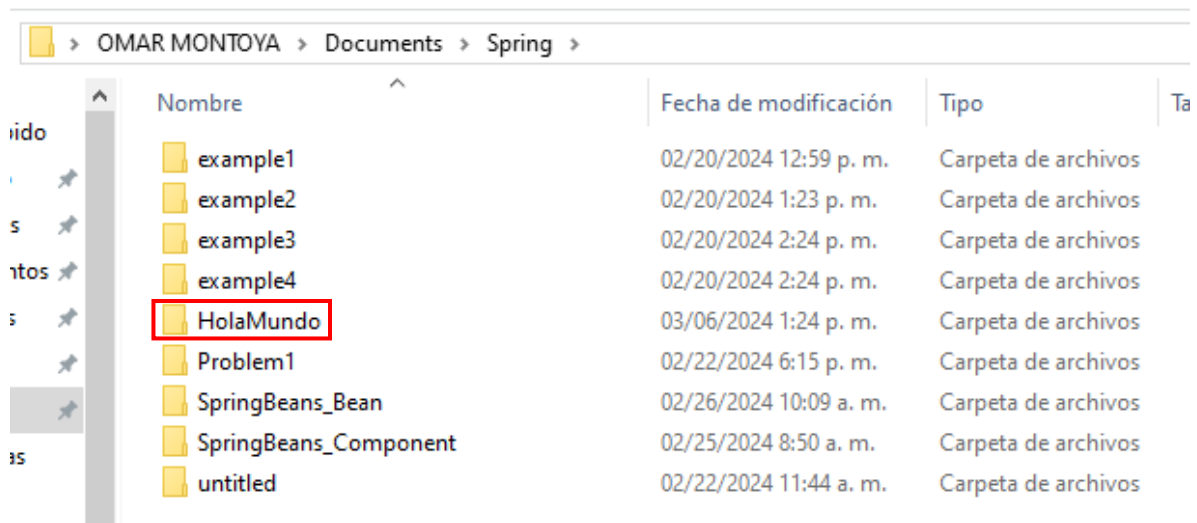


Con esto nuestro proyecto java ya estaría creado. Si bien siempre veremos los archivos desde dentro de IntelliJ, también es bueno saber y tener en cuenta que podemos ver, acceder, editar o crear estos mismos archivos directamente desde nuestro explorador de archivos.

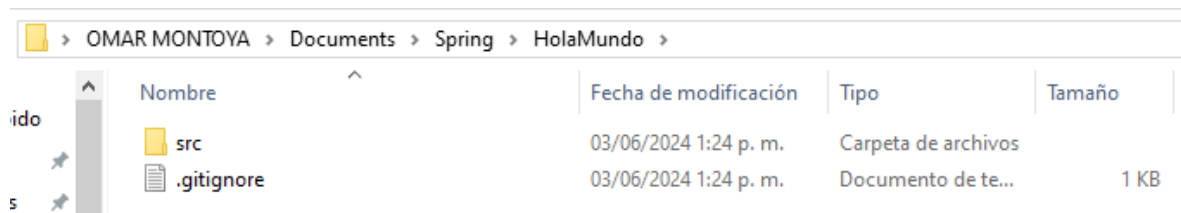
En mi caso cuando cree el proyecto utilicé la siguiente ruta:



Si vamos a esa localización en mi computadora podremos ver la carpeta “HolaMundo”



Abrimos la carpeta y veremos que coincide con los archivos que nos mostraba IntelliJ

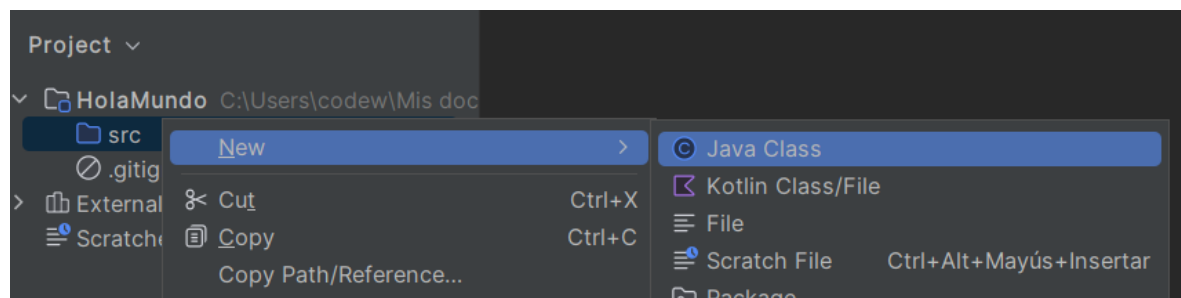


De esta manera sabemos que podemos ver dentro de nuestro explorador todos los archivos y documentos que vayamos creando en nuestro proyecto java. También es útil a la hora de exportar nuestro proyecto para compartirlo o llevarlo a otra ubicación dentro de nuestro computador.

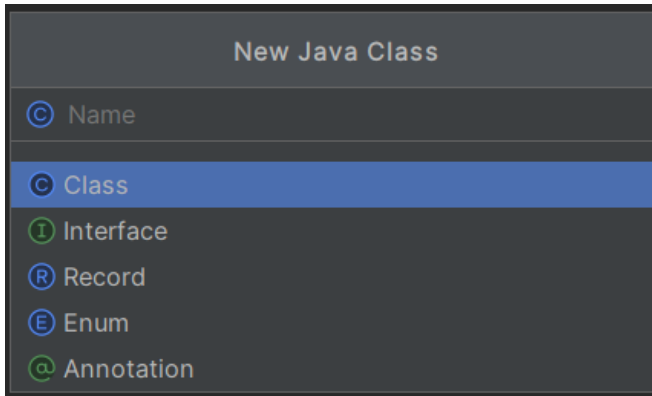
2. Crear una clase usando IntelliJ

Dentro de nuestro proyecto “HolaMundo” podremos ver que se crea por defecto la carpeta “src”. Dentro de esta carpeta tendremos guardado todas las clases java que iremos creando en nuestro proyecto.

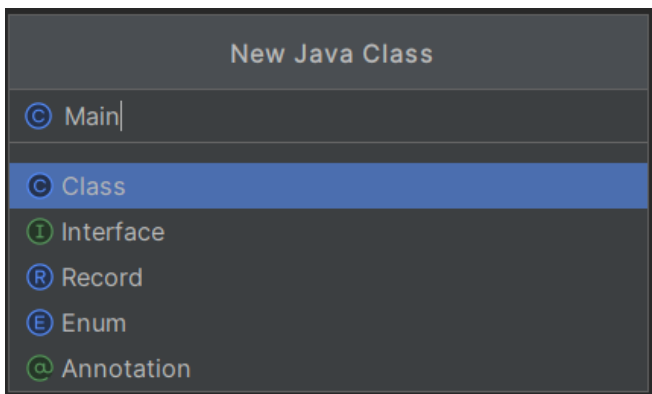
Para crear una clase en nuestro proyecto hacemos click derecho en nuestra carpeta “src” en la ventana de navegación -> New -> Java Class:



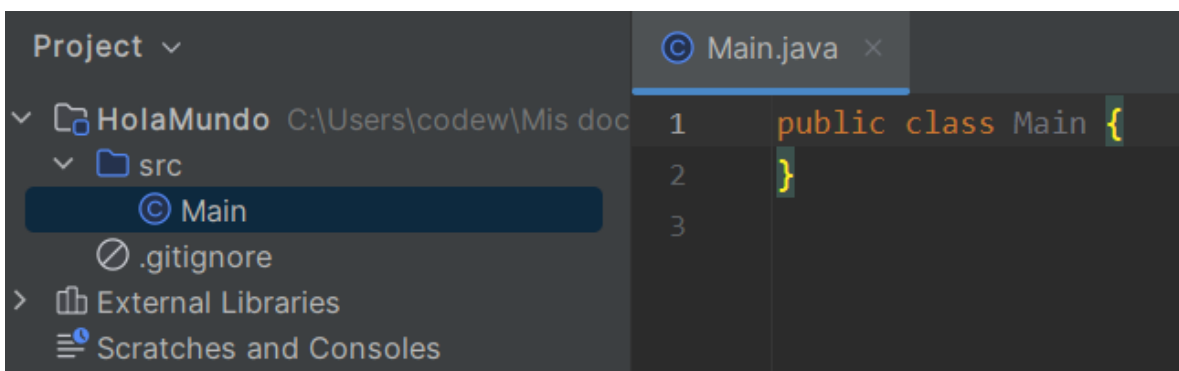
Una vez clickeado en “Java Class” nos pedirá que asignemos un nombre y que escojamos que tipo de archivo queremos crear, si es una clase, interface, enum, etc.



Como queremos crear una clase seleccionamos “Class” y le ponemos el nombre “Main” y damos enter.



Una vez damos enter, IntelliJ nos mostrará que se agrego una nueva clase “Main” dentro del paquete “src” y se abrirá el archivo mostrando lo siguiente:



3. Sintaxis básica para declarar una clase

En java podemos crear n cantidad de clases, la cantidad de clases que nuestro programa lo necesite.

Para declarar estas clases existe una sintaxis básica que debemos seguir:

```
[modificador_de_acceso] class [nombre_de_la_clase] { }
```

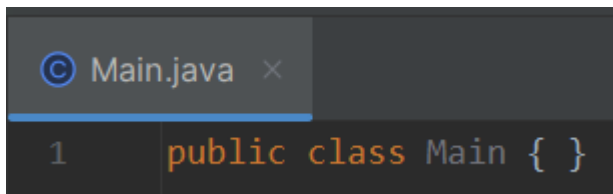
1. **modificador_de_acceso**: Se declara el tipo de visibilidad que tendrá la clase declarada con respecto al resto de clases. Lo mas común es que las clases sean declaradas como “**public**” para que puedan ser utilizadas por el resto del programa.

Los modificadores de acceso lo veremos a detalles en otra clase.

2. **class**: palabra reservada que se utiliza para indicarle a java que lo que estamos declarando es una clase.

3. **nombre_de_la_clase**: Debemos asignarle un nombre a nuestra clase. Para declarar el nombre de la clase debemos seguir las reglas de identificadores java (explicado en la clase 1) y también la técnica de nombramiento Upper Camel Case (explicado en la clase 1) que indica que las clases deben ser escritas con la primera letra en Mayúscula y de tener más de una palabra, éstas también serán escritas en mayúsculas.

En nuestro proyecto, Como le indicamos a IntelliJ que queríamos crear una clase llamada “Main”, nuestro IDE declaró automáticamente la sintaxis básica de una clase Java:



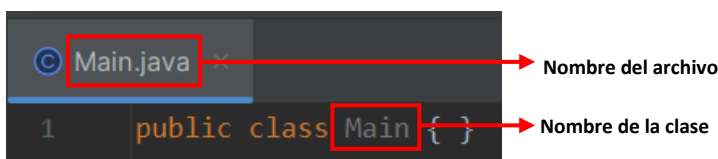
```
1 public class Main { }
```

4. Archivos .java

Cuando creamos una clase en java lo que realmente estamos creando es un archivo de texto plano con la extensión .java .

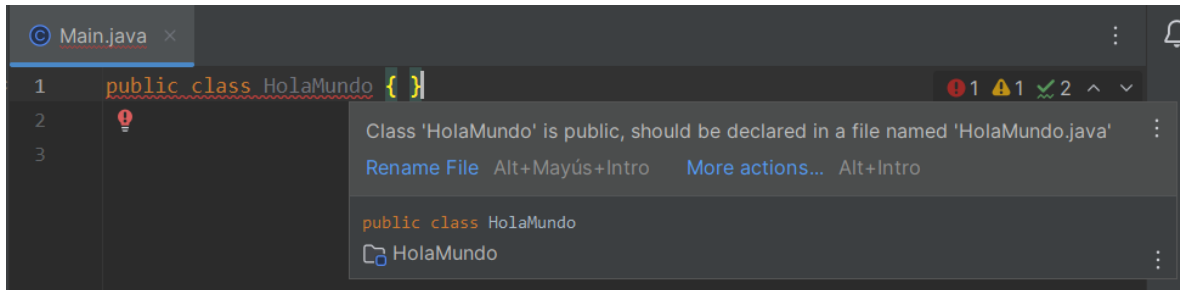
Por convención, dentro de cada archivo .java debe de existir mínimo una (1) clase pública con el mismo nombre del archivo .java .

En nuestro proyecto podemos ver que al crear la clase “Main”, se creó el archivo “Main.java” y además la clase publica que se declaró se le puso el nombre de “Main”; ya que **tanto el archivo y la clase publica deben coincidir teniendo exactamente el mismo nombre (con mayúsculas)**.



```
1 public class Main { }
```

Podemos ver mejor esto al tratar de cambiar el nombre de la clase. Si vamos y borramos la palabra “Main” y la reemplazamos por otro identificador válido de clase por ejemplo “HolaMundo” veremos el error que nos manda nuestro IDE:



“La clase ‘HolaMundo’ es pública, debería de estar declarada en un archivo llamado ‘HolaMundo.java’”

Este mensaje de error nos comunica que:

- En el archivo Main.java debería de existir mínimo 1 clase publica con el nombre “Main”. Como no encontró esta clase entonces nos marca error.
- Como tenemos declarado una clase pública “HolaMundo” en un archivo .java que no tiene ese nombre deberíamos de declarar esa clase en un archivo que tenga exactamente ese mismo nombre.

5. Sintaxis del método main

En nuestro proyecto podemos tener n cantidad de clases. De entre todas esas clases siempre habrá por lo menos una clase que contiene el método de ejecución del programa: el método main.

El método main es un método especial que marca el punto de entrada de un programa Java. Cuando ejecutas un programa Java, el sistema busca el método main para comenzar la ejecución del programa.

Sintaxis

El método main tiene una sintaxis concreta que siempre debe escribirse de la misma manera. Esta sintaxis o línea de Código es la que le indicará al sistema que aquí es donde tenemos declarado la ejecución principal del programa.

```
public static void main(String[] args) { }
```

- **public:** Indica que el método es accesible desde cualquier otra clase (modificadores de acceso lo veremos a detalles en la clase 6).
- **static:** Indica que el método pertenece a la clase en sí misma en lugar de a una instancia específica de la clase (static lo veremos a detalles en la clase 9).
- **void:** Indica que el método no devuelve ningún valor (veremos valores de retorno en la clase 7).
- **main:** nombre del método.

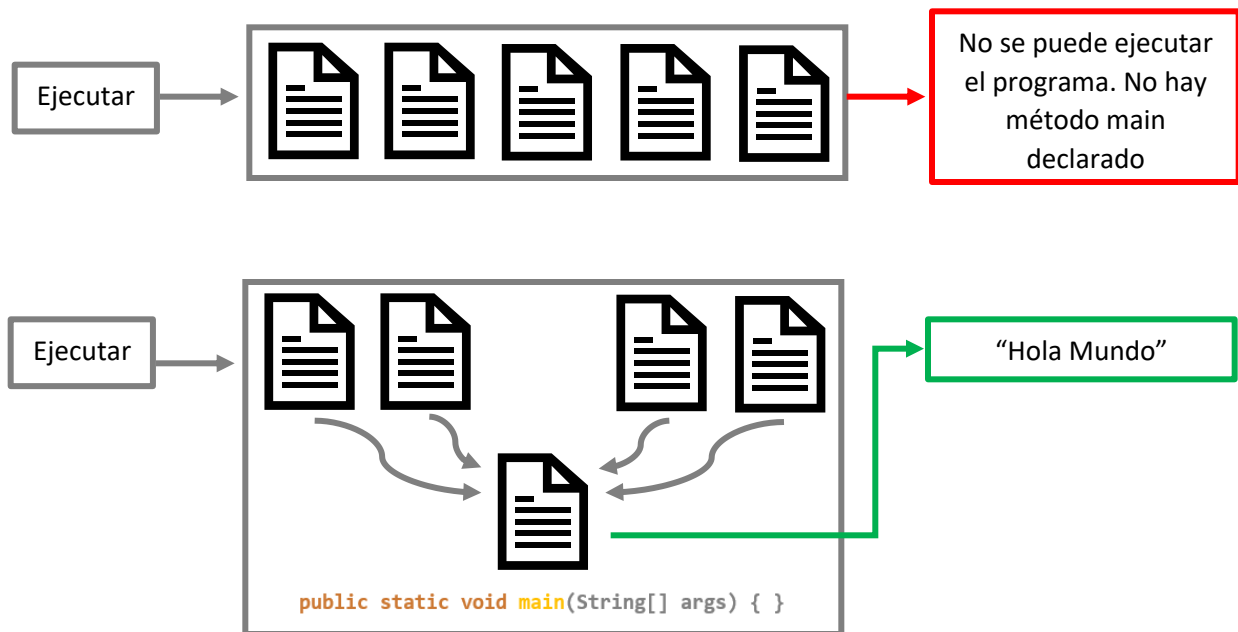


- `(String[] args)`: Es un arreglo de cadenas que puede contener argumentos pasados al programa cuando se inicia desde la línea de comandos. Lo mas probable es que ejecutemos los programas directamente desde el IDE pero los programas java también pueden ser ejecutados desde la línea de comandos. Cuando lo hacemos de esta manera es posible pasarle argumentos o valores al programa desde un inicio para que los utilice al momento de ejecutar el programa.

Veámoslo así:

Digamos que en nuestro proyecto tenemos declarado 5 clases. Para que nuestro programa se ejecute el sistema necesita que el método main (la sintaxis completa como mostramos anteriormente) esté declarado mínimo una vez en alguna clase.

Si el método main no esta declarado en ninguna clase, entonces el programa no puede ejecutarse. El método main es como una llave que enciende el programa. Sin esa llave no se puede ejecutar nada.



6. Declaración del método main

Ahora que sabemos la sintaxis del método main, su importancia y su funcionalidad, podemos ver cómo se declara dentro de nuestra clase.

En nuestro proyecto tenemos la clase “Main” que vendría siendo nuestra clase principal. Como es nuestra clase principal, debemos declarar el método main dentro de ella de la siguiente manera:

```
1 public class Main {  
2     public static void main(String[] args) {  
3  
4     }  
5 }
```

Ahora todo lo que queremos que se ejecute cuando corramos el programa debe estar declarado **dentro** del método main.

7. El método print de la clase system

En la clase 2 mencionábamos que dentro del JDK de java tenemos muchísimas librerías con clases predefinidas. Estas librerías tienen la finalidad de ayudarnos a crear software de manera más sencilla, evitando que los desarrolladores tengan que construir funcionalidades muy básicas de un programa desde cero. Una de esas funcionalidades es la de imprimir mensajes por consola a los usuarios: El método print.

El método print, como su nombre lo indica, funciona para imprimir mensajes a través de la consola de java. Para utilizarlo debemos seguir una sintaxis predefinida:

```
System.out.print( );
```

Si bien hay una explicación bastante técnica sobre el porqué se escribe de esa manera, qué significa “System” y “out”; por ahora solo aprendamos que esa es la sintaxis para imprimir por consola. En la clase 7 veremos la estructura de una clase en java y en el capítulo 8 veremos como manipular objetos.

Dentro de los paréntesis vamos a pasarle la información que queremos que imprima.

El método ya viene construido para que tenga la capacidad de imprimir diferentes tipos de datos sin necesidad de hacer algún cambio en la sintaxis.

Los datos que admite el método print son:

Tipos de datos primitivos:

1. int
2. double
3. float
4. char



5. boolean

Tipos de dato referencia (objetos)

1. String
2. Otros objetos (creados por desarrollador)

A lo largo de las siguientes clases usaremos muchísimo el método print por lo que podremos ver los diferentes casos de uso para distintos tipos de datos más adelante.

8. Imprimir Hola Mundo en Java.

Si queremos imprimir el mensaje “Hola Mundo”, dentro de java esto se consideraría como un tipo de dato String (cadena) ya que es una cadena de caracteres (más de un carácter).

Cuando utilizamos un String dentro de java siempre debe de estar declarado entre comillas dobles -> “ ”. Esto le indica al programa que todos los caracteres individuales junto con los espacios en blanco que estén dentro de las comillas dobles forman parte de una sola cadena.

Podemos imprimir una cadena declarándola dentro de los paréntesis del método print:

```
1 public class Main {  
2     public static void main(String[] args) {  
3  
4         System.out.print("Hola Mundo");  
5     }  
6 }
```

Resultado en consola

```
"C:\Program Files\Java\jdk-17\bin\java.exe" ...  
Hola Mundo  
Process finished with exit code 0
```

Podemos ver que al ejecutar el programa se nos muestra por consola la palabra Hola Mundo. De esta manera hemos creado nuestro primer programa java 😊.

9. print vs println

print se utiliza para imprimir mensajes por consola. println es otro método que también se utiliza para imprimir mensajes por consola. ¿En qué se diferencian?

La diferencia principal entre ellos es que print imprime el texto o el valor sin cambiar de línea al final, mientras que println imprime el texto o el valor y luego cambia de línea al final.

Veámoslo en un ejemplo:

```
1 public class Main {  
2     public static void main(String[] args) {  
3         //Impresión con print  
4         System.out.print("Hola");  
5         System.out.print("Mucho gusto");  
6         System.out.print("Mi nombre es Omar");  
7     }  
8 }
```

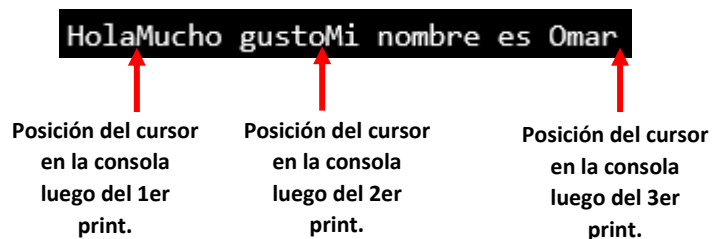
Resultado por consola

```
"C:\Program Files\Java\jdk-17\bin\java.exe" ...  
HolaMucho gustoMi nombre es Omar  
Process finished with exit code 0
```

Podemos notar que a pesar de que “Hola”, “Mucho gusto”, “Mi nombre es Omar” se encuentran declaradas en distintos métodos print, Todas quedaron impresas en la misma línea y sin ningún espacio entre ellas.

Esto se debe a que cuando utilizamos el método print, al finalizar la ejecución de cada print, el “carro” o “cursor” en la consola queda justo al final de la cadena y si tiene que imprimir otra cadena lo hará desde ese punto en que se encuentra.

```
System.out.print("Hola");  
System.out.print("Mucho gusto");  
System.out.print("Mi nombre es Omar");
```



En el caso de que queramos introducir saltos de líneas luego de cada impresión utilizando el método print, podemos agregar el carácter de escape “\n” que significa “Salto de Línea” para el sistema.

Podemos verlo en el siguiente ejemplo:

```
1 public class Main {  
2     public static void main(String[] args) {  
3         //Impresión con print  
4         System.out.print("Hola\n");  
5         System.out.print("Mucho gusto\n");  
6         System.out.print("Mi nombre es Omar\n");  
7     }  
8 }
```

Resultado por consola

```
"C:\Program Files\Java\jdk-17\bin\java.exe" ...  
Hola  
Mucho gusto  
Mi nombre es Omar  
  
Process finished with exit code 0
```

A pesar de que tenemos “\n” dentro de la declaración de la cadena, en nuestro resultado no se muestra estos caracteres. Esto es debido a que los caracteres de escape siempre serán ignorados por el sistema y no serán imprimidos ya que cada uno de los caracteres de escape tiene una funcionalidad específica (asi como la de \n es salto de línea).

Para evitar tener que utilizar \n al final de cada print, java tiene implementado el método println que incluye el salto de línea por defecto:

```
1 public class Main {  
2     public static void main(String[] args) {  
3         //Impresión con println  
4         System.out.println("Hola");  
5         System.out.println("Mucho gusto");  
6         System.out.println("Mi nombre es Omar");  
7     }  
8 }
```

Resultado por consola

```
"C:\Program Files\Java\jdk-17\bin\java.exe" ...  
Hola  
Mucho gusto  
Mi nombre es Omar  
  
Process finished with exit code 0
```

Podemos notar que a pesar de no utilizar “\n” en cada print, nuestro resultado por consola es exactamente el mismo. Esta es la finalidad de utilizar println en vez de print.

10. Caracteres de escape

Los caracteres de escape se pueden utilizar en cualquier parte de una cadena en el método print.

Aquí esta una lista de los caracteres de escape en java:

\': Comilla simple.

\": Comilla doble.

\\: Barra invertida.

\n: Salto de línea.

\r: Retorno de carro.

\t: Tabulación horizontal.

\b: Retroceso (borra un carácter hacia la izquierda).

\f: Avance de formulario (borra todo hasta el inicio de la siguiente línea).