Métodos en Java

Métodos en Java **1**
- ¿Qué aprenderás? 2
- Introducción 2
- Métodos 3
  - Creación de métodos 3
    - Definiendo un método 3
    - Ubicación del método dentro del código 5
    - Creando nuestro primer método 6
    - Parámetros 6
    - Sobrecarga de un método 7
    - Retorno 8

¡Comencemos!

¿Qué aprenderás?

- Comprender qué es un método para implementarlos en el lenguaje Java.
- Hacer uso de métodos para dividir y reutilizar códigos en Java.

Introducción

Hasta ahora hemos visto la implementación de algoritmos en Java para dar solución a ciertos problemas, no obstante esto lo hemos hecho de manera lineal y en un solo método: **El método main.**

Los métodos son fracciones de código que contienen las clases con instrucciones que se pueden utilizar más de una vez en todo el programa que estemos haciendo. Por ejemplo, si necesitamos realizar algún cálculo y este se repite varias veces dentro del programa. Para esto, debemos crear, por ejemplo, el método **cálculo,** y este podrá ser llamado varias veces en el código. Además, en programas complejos con muchas líneas de código, es necesario utilizar métodos ya que nos ayuda a organizar y entender mejor el programa completo.

¡Vamos con todo!

Métodos

Un método es una agrupación de instrucciones que realizan una determinada tarea, el cual permite reutilizar este código en distintas partes del programa.

Algunos métodos que ya hemos utilizado son:

- String.CompareTo();
- System.out.printf();
- Integer.parseInt();

Nosotros también podemos crear nuestros propios métodos, a esto llamaremos definir. Al utilizar un método ya creado por nosotros o por otras personas, le denominaremos llamar.

Creación de métodos

Hasta el momento hemos utilizado métodos que no han sido creados por nosotros, a los cuales les hemos entregado valores y hemos obtenido otros de vuelta. Ahora aprenderemos a codificar dichos métodos desde cero para luego utilizarlos. Esto nos permitirá simplificar y reutilizar código.

Definiendo un método

En Java un método se define usando la siguiente estructura.

```
[especificadores] tipoDevuelto nombreMetodo([lista parámetros]) [throws
listaExcepciones]
{
    // instrucciones
    [return valor;]
}
```

- **especificadores (opcional):** Determina el acceso a este método.
    - **private**: Solo es accesible desde la misma clase donde reside el método.
    - **public**: Accesible por la misma clase o cualquier clase.
    - **protected**: Se puede acceder solo desde clases heredadas.
    - **Por defecto**: Si no se indica, el método es público dentro de su mismo paquete.

- **tipoDevuelto:** Determina el tipo del valor que va a retornar el método. Para retornar un valor se debe utilizar la instrucción return. En caso de que no se retorne ningún valor, el método debe indicar que este tipo será void. En este caso, si se devuelve algo podría ser cualquier tipo de dato, tales como String, int, float o bien una clase personalizada, por ejemplo, Cuadrado o Notas.

- **nombreMetodo:** Es el nombre que se le da al método. Para crearlo hay que seguir las mismas normas que para crear nombres de variables.

- **lista parámetros (opcional):** Después del nombre del método y siempre entre paréntesis puede aparecer una lista de parámetros (también llamados argumentos) separados por comas.
    - Los parámetros son los datos de entrada que recibe el método para operar con ellos.
    - Un método puede recibir cero o más argumentos.
    - Se debe especificar para cada uno de ellos su tipo.
    - Los paréntesis son obligatorios aunque estén vacíos.

- **throws listaExcepciones (opcional):** Indica las excepciones que puede generar y manipular el método.

- **return:** Se utiliza para devolver un valor. En caso de ser void, no se utiliza.

Todas las instrucciones definidas dentro del método serán ejecutadas cuando hagamos un llamado a este.

Ubicación del método dentro del código

Ya teniendo definido cuál es la sintaxis de cómo se escribe un método, nos entra la duda de ¿dónde creamos el o los métodos?

Veamos cómo es la estructura de la clase donde estamos trabajando.

```
package nombre.package;
import java.util.Scanner;
public class NombreClase{
    public static void main(String[] args) {
        //INSTRUCCIONES
    }
}
```

Podemos ver que al inicio tenemos los paquetes de código que estamos utilizando. Luego viene la definición de nuestra clase (en nuestro caso el programa), y a continuación, tenemos un método llamado `main()`, donde todo el código que hemos escrito hasta ahora está dentro de éste.

```
package nombre.package;
import java.util.Scanner;
public class NombreClase{
    public static void main(String[] args) {
        //INSTRUCCIONES
    }
    static void nombreMetodo(){ //Listado de métodos
        //Lista de instrucciones
    }
}
```

Luego del método main, podemos ir agregando todos los métodos que queramos para nuestro programa.

Cabe destacar que el modificador static se debe utilizar en estos métodos ya que la llamada al método será desde otro método static (main).

Creando nuestro primer método

```
import package metodos;
public class Metodos{
    public static void main(String[] args) {
        imprimirMenu(); //llamando al método imprimirMenu();
    }

    static void imprimirMenu() {
        System.out.printf("MENU:\n");
        System.out.printf("1) Opción 1\n");
        System.out.printf("2) Opción 2\n");
        System.out.printf("3) Opción 3\n");
        System.out.printf("4) Salir\n");
    }
}
```

En el código anterior, se puede apreciar que se ha creado el método imprimirMenú, el cual al ser llamado imprimirá el menú correspondiente.

Parámetros

Crear un método que recibe un parámetro es sencillo. En la definición del método utilizaremos paréntesis para especificar los parámetros que debe recibir.

```
static void incrementar(int numero) {
    int total = numero +1;
    System.out.printf("%d\n",total);
}
```

Para utilizar el método, basta con llamarlo y entregarle un valor del tipo de dato que requiere.

```
incrementar(4); //5
```

Decimos que los parámetros se exigen porque si no los especificamos, obtendremos un error.

```
incrementar(); //5
```

```
Exception in thread "main" java.lang.Error: Unresolved compilation problem:
The method incrementar(int) in the type Metodos is not applicable for the
arguments ()
```

Además, cabe destacar que el método puede recibir variables como parámetros:

```
int a = 4;
incrementar(a); //5
```

Solamente debemos tener en consideración que la definición de los parámetros al momento de crear el método, define el tipo de datos de cada uno de sus parámetros, por lo que la variable que entre como parámetro debe ser del mismo tipo.

Sobrecarga de un método

La sobrecarga de un método corresponde a escribir el mismo nombre del método con una definición distinta, por ejemplo, podemos tener el `metodo1()` y el `metodo1(10)`. En este ejemplo podemos llamar al mismo método, con la salvedad que la primera vez se llamó sin parámetros y la segunda se le pasó el parámetro 10.

```
public static String holaMundo(){
    return "Hola Mundo!!";
}

public static String holaMundo(String nombre){
    return "Hola Mundo " + nombre;
}
```

En el ejemplo anterior, tenemos dos métodos con diferente definición de parámetros. Pues, si llamamos a los dos métodos tendremos un resultado distinto, no obstante ambos métodos se llaman de la misma manera. A esto llamamos sobrecarga de un método.

```
public class SobrecargaMetodo{

      public static void main(String[] args) {
            System.out.println(holaMundo());
            System.out.println(holaMundo("Pepito"));
      }

      public static String holaMundo(){
          return "Hola Mundo!!";
      }

      public static String holaMundo(String nombre){
          return "Hola Mundo " + nombre;
      }
}
```

Retorno

Los métodos pueden recibir parámetros y pueden devolver un valor. A este valor se le conoce como retorno. En Java, al definir un método, antes del nombre del método se especifica el tipo de dato de retorno.

```
tipoRetorno nombreMetodo(){
    //Declaración de variables locales
    //Cuerpo del método
    return valor;
}
```

Donde valor es el valor retornado por el método y tipo es el tipo del valor de retorno.

Los métodos pueden o no devolver un valor, en el caso de que se devuelva un valor la sentencia return debe ser obligatoria.

Por ejemplo:

```
public class SobrecargaMetodo{

	public static void main(String[] args) {
		System.out.println(holaMundo());
		System.out.println(holaMundo("Pepito"));
	}

	public static String holaMundo(){
	    return "Hola Mundo!!";
	}

	public static String holaMundo(String nombre){
	    return "Hola Mundo " + nombre;
	}
}
```

En este caso, el método `holaMundo` por su definición, dice que devolverá un dato tipo String.

Pero también podrían haberse creado dichos métodos sin retorno de valor:

```
public class SobrecargaMetodo{

	public static void main(String[] args) {
		holaMundo();
		holaMundo("Pepito");
	}

	public static void holaMundo(){
	    System.out.println("Hola Mundo!!");
	}

	public static void holaMundo(String nombre){
	    System.out.println("Hola Mundo " + nombre);
	}
}
```

En este caso, al definir el retorno como void, estamos diciendo que el método no devolverá algún resultado, de lo contrario hará el proceso dentro de el mismo. En este caso, la sentencia return no es obligatoria.

En Java, si se define un método con retorno es necesario que siempre haya un valor de retorno.

```
public static String mayorOMenor(int edad) {
    if (edad < 18) {
        return "Menor de edad";
    } else {
        return "Mayor de edad";
    }
}
```

El return no necesariamente debe estar en la última línea, puede utilizarse antes de llegar al final del código, haciendo que lo siguiente no se ejecute, pero debe haber al menos un return que se ejecute sí o sí dentro del método.

Si escribiéramos solamente esto:

```
public static String mayorOMenor(int edad) {
    if (edad < 18) {
        return "Menor de edad";
    }
}
```

Nos dará el siguiente error, donde nos dice que debe haber obligatoriamente un return de tipo String.

```
Exception in thread "main" java.lang.Error: Unresolved compilation problem:
This method must return a result of type String
```