18. Methods in Java

- A method is a block of code which only runs when it is called.
- You can pass data, known as parameters, into a method.
- Methods are used to perform certain actions, and they are also known as functions.
- Why use methods? To reuse code: define the code once, and use it many times.

Methods can be classified into two types.

1. User Defined Methods:
   These Methods are created by us depends upon the usage.
2. Pre Defined Methods:
   These Methods are nothing but built in Methods. These Methods are already in the JDK. So that we will use it directly.

18.1 Create a Method

A method must be declared within a class. It is defined with the name of the method, followed by parentheses (). Java provides some pre-defined methods, such as System.out.println(), but you can also create your own methods to perform certain actions:

Example:
Create a Method inside main:

```
public class Main
{
  static void myMethod()
  {
    // code to be executed
  }
  public static void main()
  {

  }
}
```

Example Explained

- `myMethod()` is the name of the method
- `static` means that the method belongs to the Main class and not an object of the Main class. You will learn more about objects and how to access methods through objects later in this tutorial.
- `void` means that this method does not have a return value. You will learn more about return values later in this chapter.

18.2 Call a Method

To call a method in Java, write the method's name followed by two parentheses () and a semicolon;

In the following example, myMethod() is used to print a text (the action), when it is called:

Example:
Inside main, call the myMethod() method:

```
public class Main
{
      static void myMethod()
      {
      System.out.println("I just got executed!");
      }

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

Output:

```
I just got executed!
```

Example2:

```
public class Main
{
      static void myMethod()
      {
      System.out.println("I just got executed!");
      }

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

Output:

```
I just got executed!
I just got executed!
I just got executed!
```

18.3 Types of Methods

1. Method with void and without parameters and arguments.
2. Method with void and with parameters and arguments.
3. Method with datatype and without parameters and arguments having return type.
4. Method with datatype and with parameters and arguments having return type.

18.3.1 Method with void and without parameters and arguments

We already seen the examples for this type of methods earlier. We will see some more examples for this type to understand it properly.

Example:

```
public class A
{
	static void myMethod()
	{
		int i=2;
		int j=3;
		System.out.println(i+j);
	}

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

Output:

```
5
5
5
```

18.3.2 Method with void and with parameters and arguments.

- Information can be passed to methods as parameter. Parameters act as variables inside the method.
- Parameters are specified after the method name, inside the parentheses. You can add as many parameters as you want, just separate them with a comma.

The following example has a method that takes a String called fname as parameter. When the method is called, we pass along a first name, which is used inside the method to print the full name:

Example1:

```
public class A
{
	static void myMethod(String fname)
	{
		System.out.println("fname is "+fname);
	}
```

```
	public static void main(String[] args)
	{
		myMethod("Liam");
		myMethod("Jenny");
		myMethod("Anja");
	}
}
```

Output:

```
fname is Liam
fname is Jenny
fname is Anja
```

Example2:

```
public class A
{
	static void myMethod(String fname, int age)
	{
		System.out.println("fname is "+fname+", age is "+age);
	}

	public static void main(String[] args)
	{
		myMethod("Liam",5);
		myMethod("Jenny",8);
		myMethod("Anja",23);
	}
}
```

Output:

```
fname is Liam, age is 5
fname is Jenny, age is 8
fname is Anja, age is 23
```

Note: that when you are working with multiple parameters, the method call must have the same number of arguments as there are parameters, and the arguments must be passed in the same order.

18.3.3 Method with datatype and without parameters and arguments having return type.

All the primitive data types in java must have return type when it is used in functions.

The void keyword, used in the examples above, indicates that the method should not return a value. If you want the method to return a value, you can use a primitive data type (such as int, char, etc.) instead of void, and use the return keyword inside the method:

Example:

```
public class MyClass
{
	static int add()
	{
		int i=2;
		int j=5;
		System.out.println("Total is "+(i+j));
		return i+j;
	}
	static float myfun()
	{
		int i=12;
		int j=23;
		int c=i-j;
		System.out.println(c);
		return c;
	}

	public static void main(String[] args)
	{
		System.out.println(add());
		int x=add();
		System.out.println(x);
		float y=myfun();
		System.out.println(y);
		System.out.println(myfun());

	}
}
```

Output:

```
Total is 7
7
Total is 7
7
-11
-11.0
-11
-11.0
```

18.3.4 Method with datatype and with parameters and arguments having return type.

Example:

```
public class MyClass
{
	static int add(int i, int j)
```

```
	{
		System.out.println("Total is "+(i+j));
		return i+j;
	}
	static double myfun(int i, double j)
	{

		double c=i-j;
		System.out.println(c);
		return c;
	}

	public static void main(String[] args)
	{
		System.out.println(add(2,4));
		int x=add(4,3);
		System.out.println(x);
		double y=myfun(8,5);
		System.out.println(y);
		System.out.println(myfun(3,4));

	}
}
```

Output:

```
Total is 6
6
Total is 7
7
3.0
3.0
-1.0
-1.0
```

18.4 Method Overloading

With **method overloading**, multiple methods can have the same name with different parameters:

Example1:

```
public class MyClass
{
	static int plusMethod(int x, int y)
	{
		return x + y;
	}
	static double plusMethod(double x, double y)
	{
```

```
        return x + y;
    }
    public static void main(String[] args)
    {
        int myNum1 = plusMethod(8, 5);
        double myNum2 = plusMethod(4.3, 6.26);
        System.out.println("int: " + myNum1);
        System.out.println("double: " + myNum2);
    }
}
```

Output:

```
int: 13
double: 10.559999999999999
```

Instead of defining two methods that should do the same thing, it is better to overload one.
In the example below, we overload the plusMethod method to work for both int and double:

Note: Multiple methods can have the same name as long as the number and/or type of parameters are different.

18.5 Scope of Variables

Java Scope:

In Java, variables are only accessible inside the region they are created. This is called **scope**.

Method Scope:

Variables declared directly inside a method are available anywhere in the method following the line of code in which they were declared:

Example:

```
public class MyClass
{
    public static void main(String[] args)
    {
        // Code here cannot use x

        int x = 100;

        // Code here can use x
        System.out.println(x);
    }
}
```

Output:

```
100
```

Block Scope:

A block of code refers to all of the code between curly braces {}.

Variables declared inside blocks of code are only accessible by the code between the curly braces, which follows the line in which the variable was declared:

Example:

```
public class MyClass
{
      public static void main(String[] args)
      {
          // Code here CANNOT use x
          { // This is a block
            // Code here CANNOT use x
            int x = 100;
            // Code here CAN use x
            System.out.println(x);
          } // The block ends here
         // Code here CANNOT use x
      }
}
```

Output:

```
100
```

Note: A block of code may exist on its own or it can belong to an if, while or for statement. In the case of for statements, variables declared in the statement itself are also available inside the block's scope.

18.6 Recursion

Recursion is the technique of making a function call itself. This technique provides a way to break complicated problems down into simple problems which are easier to solve. Recursion may be a bit difficult to understand. The best way to figure out how it works is to experiment with it.

Example:

Adding two numbers together is easy to do, but adding a range of numbers is more complicated. In the following example, recursion is used to add a range of numbers together by breaking it down into the simple task of adding two numbers:

```
public class MyClass
{
      public static void main(String[] args)
      {
            int result = sum(10);
            System.out.println(result);
      }
```

```
        public static int sum(int k)
        {
              if (k > 0)
              {
                    return k + sum(k - 1);
              }
             else
              {
                    return 0;
              }
        }
}
```

Output:

```
55
```