Java Methods

In this tutorial, we will learn about Java methods, how to define methods, and how to use methods in Java programs with the help of examples.

A method is a block of code that performs a specific task.

Suppose you need to create a program to create a circle and color it. You can create two methods to solve this problem:

- a method to draw the circle
- a method to color the circle

Dividing a complex problem into smaller chunks makes your program easy to understand and reusable.

In Java, there are two types of methods:

- **User-defined Methods**: We can create our own method based on our requirements.
- **Standard Library Methods**: These are built-in methods in Java that are available to use.

Let's first learn about user-defined methods.

Declaring a Java Method

The syntax to declare a method is:

```
returnType methodName() {
  // method body
}
```

Here,

- **returnType** - It specifies what type of value a method returns For example if a method has an `int` return type then it returns an integer value.

If the method does not return a value, its return type is `void`.

- **methodName** - It is an identifier that is used to refer to the particular method in a program.
- **method body** - It includes the programming statements that are used to perform some tasks. The method body is enclosed inside the curly braces `{ }`.

For example,

```
int addNumbers() {
// code
}
```

In the above example, the name of the method is `adddNumbers()`. And, the return type is `int`. We will learn more about return types later in this tutorial.

This is the simple syntax of declaring a method. However, the complete syntax of declaring a method is

```
modifier static returnType nameOfMethod (parameter1, parameter2, ...) {
  // method body
}
```

Here,

- **modifier** - It defines access types whether the method is public, private, and so on. To learn more, visit Java Access Specifier.
- **static** - If we use the `static` keyword, it can be accessed without creating objects.

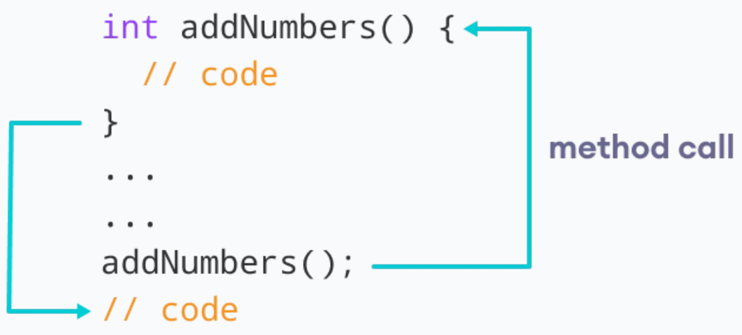
  For example, the `sqrt()` method of standard Math class is static. Hence, we can directly call `Math.sqrt()` without creating an instance of `Math` class.
- **parameter1/parameter2** - These are values passed to a method. We can pass any number of arguments to a method.

Calling a Method in Java

In the above example, we have declared a method named `addNumbers()`. Now, to use the method, we need to call it.

Here's is how we can call the `addNumbers()` method.

```
// calls the method
addNumbers();
```



Working of Java Method Call

Example 1: Java Methods

```
class Main {

  // create a method
  public int addNumbers(int a, int b) {
    int sum = a + b;
    // return value
    return sum;
  }

  public static void main(String[] args) {

    int num1 = 25;
    int num2 = 15;

    // create an object of Main
    Main obj = new Main();
    // calling method
    int result = obj.addNumbers(num1, num2);
    System.out.println("Sum is: " + result);
  }
}
```

Run Code

Output

```
Sum is: 40
```

In the above example, we have created a method named `addNumbers()`. The method takes two parameters `a` and `b`. Notice the line,

```
int result = obj.addNumbers(num1, num2);
```

Here, we have called the method by passing two arguments `num1` and `num2`. Since the method is returning some value, we have stored the value in the `result` variable.

> **Note**: The method is not static. Hence, we are calling the method using the object of the class.

Java Method Return Type

A Java method may or may not return a value to the function call. We use the **return statement** to return any value. For example,

```
int addNumbers() {
...
return sum;
}
```

Here, we are returning the variable `sum`. Since the return type of the function is `int`. The sum variable should be of `int` type. Otherwise, it will generate an error.

Example 2: Method Return Type

```
class Main {

// create a method
  public static int square(int num) {

    // return statement
    return num * num;
  }

  public static void main(String[] args) {
    int result;

    // call the method
    // store returned value to result
    result = square(10);

    System.out.println("Squared value of 10 is: " + result);
  }
}
```

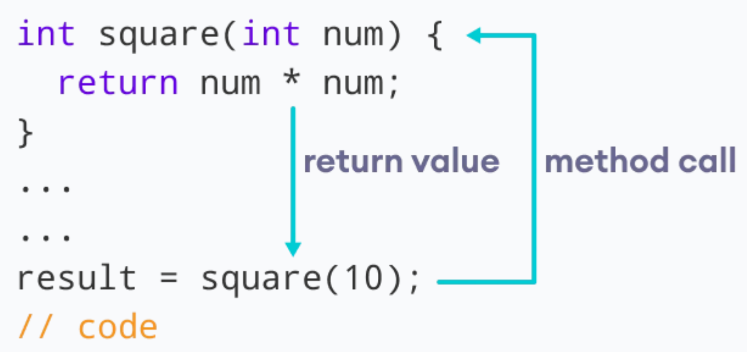
Run Code

Output:

```
Squared value of 10 is: 100
```

In the above program, we have created a method named `square()`. The method takes a number as its parameter and returns the square of the number.

Here, we have mentioned the return type of the method as `int`. Hence, the method should always return an integer value.



```
int square(int num) {
  return num * num;
}
...
...
result = square(10);
// code
```

Representation of the Java method returning a value

Note: If the method does not return any value, we use the void keyword as the return type of the method. For example,

```
public void square(int a) {
  int square = a * a;
  System.out.println("Square is: " + square);
}
```

Method Parameters in Java

A method parameter is a value accepted by the method. As mentioned earlier, a method can also have any number of parameters. For example,

```
// method with two parameters
int addNumbers(int a, int b) {
  // code
}

// method with no parameter
int addNumbers(){
  // code
}
```

If a method is created with parameters, we need to pass the corresponding values while calling the method. For example,

```
// calling the method with two parameters
addNumbers(25, 15);

// calling the method with no parameters
addNumbers()
```

Example 3: Method Parameters

```
class Main {

  // method with no parameter
  public void display1() {
    System.out.println("Method without parameter");
  }

  // method with single parameter
  public void display2(int a) {
    System.out.println("Method with a single parameter: " + a);
  }

  public static void main(String[] args) {

    // create an object of Main
    Main obj = new Main();

    // calling method with no parameter
    obj.display1();

    // calling method with the single parameter
    obj.display2(24);
  }
}
```

Run Code

Output

```
Method without parameter
Method with a single parameter: 24
```

Here, the parameter of the method is `int`. Hence, if we pass any other data type instead of `int`, the compiler will throw an error. It is because Java is a strongly typed language.

Note: The argument `24` passed to the `display2()` method during the method call is called the actual argument.

The parameter `num` accepted by the method definition is known as a formal argument. We need to specify the type of formal arguments. And, the type of actual arguments and formal arguments should always match.

Standard Library Methods

The standard library methods are built-in methods in Java that are readily available for use. These standard libraries come along with the Java Class Library (JCL) in a Java archive (*.jar) file with JVM and JRE.

For example,

- `print()` is a method of `java.io.PrintSteam`. The `print("...")` method prints the string inside quotation marks.
- `sqrt()` is a method of `Math` class. It returns the square root of a number.

Here's a working example:

Example 4: Java Standard Library Method

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

    // using the sqrt() method
    System.out.print("Square root of 4 is: " + Math.sqrt(4));
  }
}
```

Run Code

Output:

```
Square root of 4 is: 2.0
```

To learn more about standard library methods, visit Java Library Methods.

What are the advantages of using methods?

1. The main advantage is **code reusability**. We can write a method once, and use it multiple times. We do not have to rewrite the entire code each time. Think of it as, "write once, reuse multiple times".

Example 5: Java Method for Code Reusability

```
public class Main {

  // method defined
  private static int getSquare(int x){
    return x * x;
  }

  public static void main(String[] args) {
    for (int i = 1; i <= 5; i++) {

      // method call
      int result = getSquare(i);
      System.out.println("Square of " + i + " is: " + result);
    }
  }
}
```

Run Code

Output:

```
Square of 1 is: 1
Square of 2 is: 4
Square of 3 is: 9
Square of 4 is: 16
Square of 5 is: 25
```

In the above program, we have created the method named `getSquare()` to calculate the square of a number. Here, the method is used to calculate the square of numbers less than **6**.

Hence, the same method is used again and again.

2. Methods make code more **readable and easier** to debug. Here, the `getSquare()` method keeps the code to compute the square in a block. Hence, makes it more readable.