Java Methods

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.

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 { }.

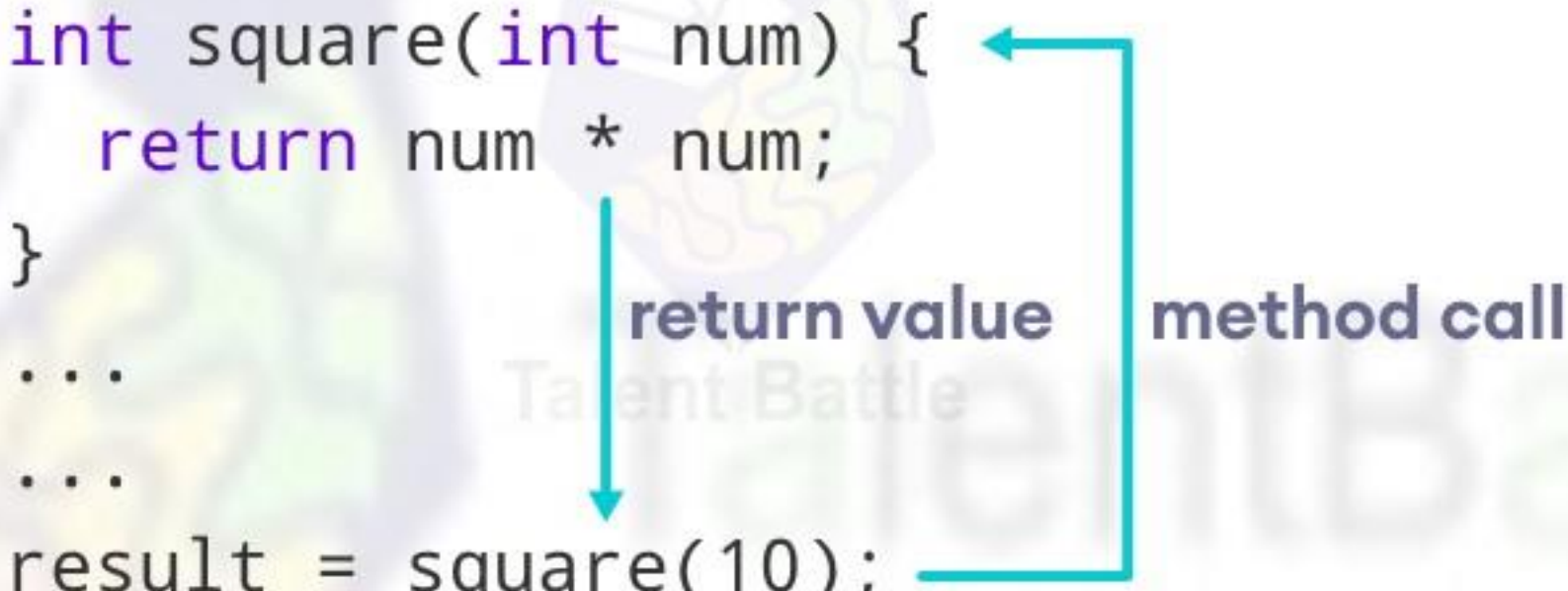
```
class Main {

 // 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);
 }
}
```



Method Parameters in Java

```
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);
  }
}
```

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.

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

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

What are the advantages of using methods?

1. The main advantage is **code reusability**
2. Methods make code more **readable and easier** to debug.

```
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);
  }
 }
}
```

Java Constructors

What is a Constructor?

A constructor in Java is similar to a method that is invoked when an object of the class is created.

Unlike Java methods, a constructor has the same name as that of the class and does not have any return type. For example,

```
class Test {
  Test() {
    // constructor body
  }
}
```

```
class Main {
  private String name;

  // constructor
  Main() {
    System.out.println("Constructor Called:");
    name = "Programming";
  }

  public static void main(String[] args) {

    // constructor is invoked while
    // creating an object of the Main class
    Main obj = new Main();
    System.out.println("The name is " + obj.name);
  }
}
```

Types of Constructor

In Java, constructors can be divided into 3 types:

1.No-Arg Constructor

2.Parameterized Constructor

3.Default Constructor

1. Java No-Arg Constructors

Similar to methods, a Java constructor may or may not have any parameters (arguments).

If a constructor does not accept any parameters, it is known as a no-argument constructor. For example,

```
private Constructor() {
   // body of the constructor
}
```

```
class Main {

  int i;

  // constructor with no parameter
  private Main() {
    i = 5;
    System.out.println("Constructor is called");
  }

  public static void main(String[] args) {

    // calling the constructor without any parameter
    Main obj = new Main();
    System.out.println("Value of i: " + obj.i);
  }
}
```

Example 1: Java program to create a private constructor

```
class Test {
  // create private constructor
  private Test () {
    System.out.println("This is a private constructor.");
  }
  // create a public static method
  public static void instanceMethod() {
    // create an instance of Test class
    Test obj = new Test();
  }
}
class Main {
 public static void main(String[] args) {
    // call the instanceMethod()
    Test.instanceMethod();
  }
}
```

2. Java Parameterized Constructor

A Java constructor can also accept one or more parameters. Such constructors are known as parameterized constructors (constructor with parameters).

```
class Main {
 String languages;
 // constructor accepting single value
 Main(String lang) {
  languages = lang;
  System.out.println(languages + " Programming Language");
 }
 public static void main(String[] args) {
  // call constructor by passing a single value
  Main obj1 = new Main("Java");
  Main obj2 = new Main("Python");
  Main obj3 = new Main("C");
 }
}
```

3. Java Default Constructor

If we do not create any constructor, the Java compiler automatically create a no-arg constructor during the execution of the program. This constructor is called default constructor.

```
class Main {
 int a;
 boolean b;
 public static void main(String[] args) {
  // A default constructor is called
   Main obj = new Main();
   System.out.println("Default Value:");
   System.out.println("a = " + obj.a);
   System.out.println("b = " + obj.b);
 }
}
```

Important Notes on Java Constructors

- Constructors are invoked implicitly when you instantiate objects.
- The two rules for creating a constructor are:

The name of the constructor should be the same as the class.

A Java constructor must not have a return type.

- If a class doesn't have a constructor, the Java compiler automatically creates a default constructor during run-time. The default constructor initializes instance variables with default values. For example, the int variable will be initialized to 0
- Constructor types:
- **No-Arg Constructor** - a constructor that does not accept any arguments
- **Parameterized constructor** - a constructor that accepts arguments
- **Default Constructor** - a constructor that is automatically created by the Java compiler if it is not explicitly defined.
- A constructor cannot be abstract or static or final.
- A constructor can be overloaded but can not be overridden.

Constructors Overloading in Java

Similar to Java method overloading, we can also create two or more constructors with different parameters. This is called constructors overloading.

```
class Main {
 String language;
 Main() {
   this.language = "Java";
 }
 Main(String language) {
   this.language = language;
 }
 public void getName() {
   System.out.println("Programming Langauage: " + this.language);
 }
 public static void main(String[] args) {
   Main obj1 = new Main();
   Main obj2 = new Main("Python");
   obj1.getName();
   obj2.getName();
 }
}
```

Java Strings

Java String

In Java, a string is a sequence of characters. For example, "hello" is a string containing a sequence of characters 'h', 'e', 'l', 'l', and 'o'.

We use double quotes to represent a string in Java. For example,

```
// create a string
String type = "Java programming";
```

Here, we have created a string variable named type. The variable is initialized with the string Java Programming.

Note: Strings in Java are not primitive types (like int, char, etc). Instead, all strings are objects of a predefined class named String.

And, all string variables are instances of the String class.

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

    // create strings
    String first = "Java";
    String second = "Python";
    String third = "JavaScript";

    // print strings
    System.out.println(first);   // print Java
    System.out.println(second);  // print Python
    System.out.println(third);   // print JavaScript
  }
}
```

Java String Operations

1. Get Length of a String

To find the length of a string, we use the length() method of the String. For example,

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

    // create a string
    String greet = "Hello! World";
    System.out.println("String: " + greet);

    // get the length of greet
    int length = greet.length();
    System.out.println("Length: " + length);
  }
}
```

2. Join two Strings

We can join two strings in Java using the concat() method. For example,

```
class Main {
  public static void main(String[] args) {
    // create first string
    String first = "Java ";
    System.out.println("First String: " + first);
    // create second
    String second = "Programming";
    System.out.println("Second String: " + second);
    // join two strings
    String joinedString = first.concat(second);
    System.out.println("Joined String: " + joinedString);
  }
}
```

3. Compare two Strings

In Java, we can make comparisons between two strings using the equals() method. For example,

```
class Main {
  public static void main(String[] args) {
   // create 3 strings
    String first = "java programming";
    String second = "java programming";
    String third = "python programming";
    // compare first and second strings
    boolean result1 = first.equals(second);
    System.out.println("Strings first and second are equal: " + result1);
    // compare first and third strings
    boolean result2 = first.equals(third);
    System.out.println("Strings first and third are equal: " + result2);
  }
}
```

Methods of Java String

Methods	Description
substring()	returns the substring of the string
replace()	replaces the specified old character with the specified new character
charAt()	returns the character present in the specified location
getBytes()	converts the string to an array of bytes
indexOf()	returns the position of the specified character in the string
compareTo()	compares two strings in the dictionary order
trim()	removes any leading and trailing whitespaces
format()	returns a formatted string
split()	breaks the string into an array of strings
toLowerCase()	converts the string to lowercase
toUpperCase()	converts the string to uppercase
valueOf()	returns the string representation of the specified argument
toCharArray()	converts the string to a char array

Creating strings using the new keyword

Since strings in Java are objects, we can create strings using the new keyword as well. For example,

```
// create a string using the new keyword
String name = new String("Java String");
```

In the above example, we have created a string name using the new keyword.

Here, when we create a string object, the String() constructor is invoked.

Example: Create Java Strings using the new keyword

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

    // create a string using new
    String name = new String("Java String");

    System.out.println(name);  // print Java String
  }
}
```

Create String using literals vs new keyword

In Java, the JVM maintains a string pool to store all of its strings inside the memory. The string pool helps in reusing the strings.

1. While creating strings using string literals,

String example = "Java";

Here, we are directly providing the value of the string (Java). Hence, the compiler first checks the string pool to see if the string already exists.

If the string already exists, the new string is not created. Instead, the new reference, example points to the already existed string (Java).

If the string doesn't exist, the new string (Java is created.

2. While creating strings using the new keyword,

String example = new String("Java");

Here, the value of the string is not directly provided. Hence, the new string is created all the time.