Method in Java

In general, a **method** is a way to perform some task. Similarly, the **method in Java** is a collection of instructions that performs a specific task. It provides the reusability of code. We can also easily modify code using **methods**. In this section, we will learn **what is a method in Java, types of methods, method declaration,** and **how to call a method in Java**.

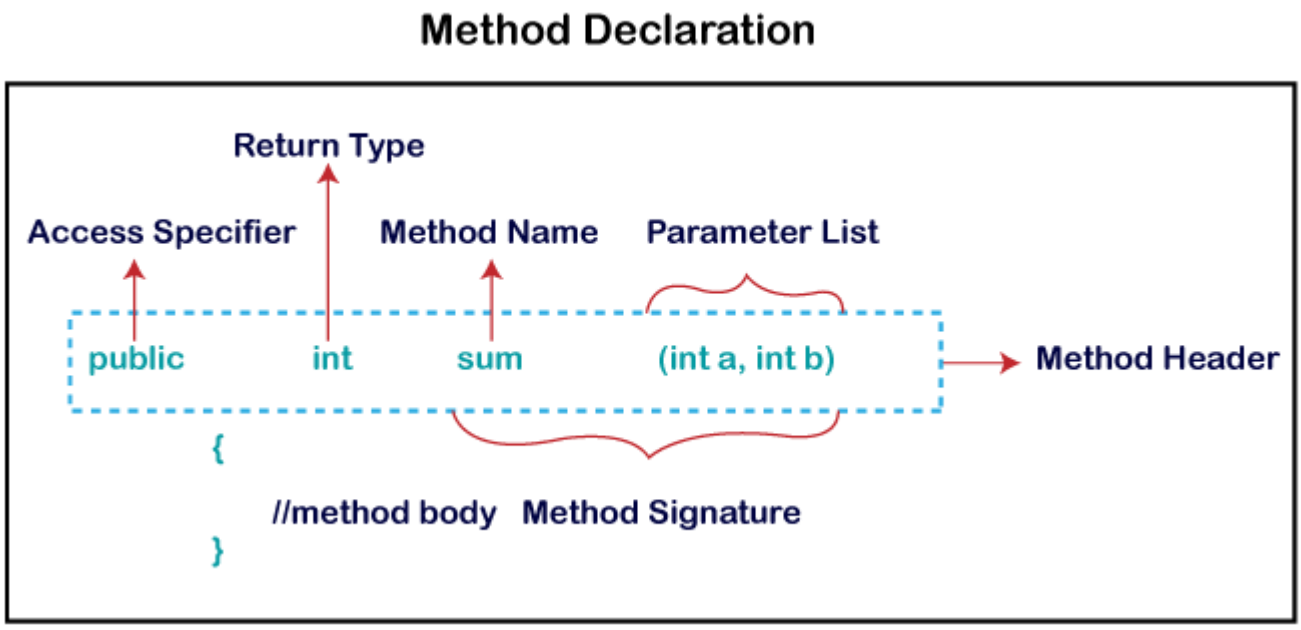
What is a method in Java?

A **method** is a block of code or collection of statements or a set of code grouped together to perform a certain task or operation. It is used to achieve the **reusability** of code. We write a method once and use it many times. We do not require to write code again and again. It also provides the **easy modification** and **readability** of code, just by adding or removing a chunk of code. The method is executed only when we call or invoke it.

The most important method in Java is the **main()** method.

Method Declaration

The method declaration provides information about method attributes, such as visibility, return-type, name, and arguments. It has six components that are known as **method header**, as we have shown in the following figure.



Method Signature: Every method has a method signature. It is a part of the method declaration. It includes the **method name** and **parameter list**.

Access Specifier: Access specifier or modifier is the access type of the method. It specifies the visibility of the method. Java provides **four** types of access specifier:

- **Public:** The method is accessible by all classes when we use public specifier in our application.
- **Private:** When we use a private access specifier, the method is accessible only in the classes in which it is defined.
- **Protected:** When we use protected access specifier, the method is accessible within the same package or subclasses in a different package.
- **Default:** When we do not use any access specifier in the method declaration, Java uses default access specifier by default. It is visible only from the same package only.

Return Type: Return type is a data type that the method returns. It may have a primitive data type, object, collection, void, etc. If the method does not return anything, we use void keyword.

Method Name: It is a unique name that is used to define the name of a method. It must be corresponding to the functionality of the method. Suppose, if we are creating a method for subtraction of two numbers, the method name must be **subtraction().** A method is invoked by its name.

Parameter List: It is the list of parameters separated by a comma and enclosed in the pair of parentheses. It contains the data type and variable name. If the method has no parameter, left the parentheses blank.

Method Body: It is a part of the method declaration. It contains all the actions to be performed. It is enclosed within the pair of curly braces.

Naming a Method

While defining a method, remember that the method name must be a **verb** and start with a **lowercase** letter. If the method name has more than two words, the first name must be a verb followed by adjective or noun. In the multi-word method name, the first letter of each word must be in **uppercase** except the first word. For example:

Single-word method name: sum(), area()

Multi-word method name: areaOfCircle(), stringComparision()

It is also possible that a method has the same name as another method name in the same class, it is known as **method overloading**.

Types of Method

There are two types of methods in Java:

- Predefined Method
- User-defined Method

Predefined Method

In Java, predefined methods are the method that is already defined in the Java class libraries is known as predefined methods. It is also known as the **standard library method** or **built-in method**. We can directly use these methods just by calling them in the program at any point. Some pre-defined methods are **length(), equals(), compareTo(), sqrt(),** etc. When we call any of the predefined methods in our program, a series of codes related to the corresponding method runs in the background that is already stored in the library.

Each and every predefined method is defined inside a class. Such as **print()** method is defined in the **java.io.PrintStream** class. It prints the statement that we write inside the method. For example, **print("Java")**, it prints Java on the console.

Let's see an example of the predefined method.

Demo.java

```
1. public class Demo
2. {
3. public static void main(String[] args)
4. {
5. // using the max() method of Math class
6. System.out.print("The maximum number is: " + Math.max(9,7));
7. }
8. }
```

Output:

```
The maximum number is: 9
```

In the above example, we have used three predefined methods **main(), print(),** and **max()**. We have used these methods directly without declaration

because they are predefined. The print() method is a method of **PrintStream** class that prints the result on the console. The max() method is a method of the **Math** class that returns the greater of two numbers.

User-defined Method

The method written by the user or programmer is known as **a user-defined** method. These methods are modified according to the requirement.

How to Create a User-defined Method

Let's create a user defined method that checks the number is even or odd. First, we will define the method.

```
1. //user defined method
2. public static void findEvenOdd(int num)
3. {
4. //method body
5. if(num%2==0)
6. System.out.println(num+" is even");
7. else
8. System.out.println(num+" is odd");
9. }
```

We have defined the above method named findevenodd(). It has a parameter **num** of type int. The method does not return any value that's why we have used void. The method body contains the steps to check the number is even or odd. If the number is even, it prints the number **is even**, else prints the number **is odd**.

How to Call or Invoke a User-defined Method

Once we have defined a method, it should be called. The calling of a method in a program is simple. When we call or invoke a user-defined method, the program control transfer to the called method.

EvenOdd.java

```
1. import java.util.Scanner;
2. public class EvenOdd
3. {
4. public static void main (String args[])
5. {
6. //creating Scanner class object
7. Scanner scan=new Scanner(System.in);
8. System.out.print("Enter the number: ");
9. //reading value from user
10. int num=scan.nextInt();
11. //method calling
12. findEvenOdd(num);
13. }
14. //user defined method
15. public static void findEvenOdd(int num)
16. {
17. //method body
18. if(num%2==0)
19. System.out.println(num+" is even");
20. else
21. System.out.println(num+" is odd");
22. }
23. }
```

In the above code snippet, as soon as the compiler reaches at line **findEvenOdd(num),** the control transfer to the method and gives the output accordingly.

Output 1:

```
Enter the number: 12
12 is even
```

Output 2:

```
Enter the number: 99
99 is odd
```

Let's see another program that return a value to the calling method.

In the following program, we have defined a method named **add()** that sum up the two numbers. It has two parameters n1 and n2 of integer type. The values of n1 and n2 correspond to the value of a and b, respectively. Therefore, the method adds the value of a and b and store it in the variable s and returns the sum.

Addition.java

```
1. public class Addition
2. {
3. public static void main(String[] args)
4. {
5. int a = 19;
6. int b = 5;
7. //method calling
8. int c = add(a, b);   //a and b are actual parameters
9. System.out.println("The sum of a and b is= " + c);
10. }
11. //user defined method
12. public static int add(int n1, int n2)   //n1 and n2 are formal parameters
13. {
14. int s;
15. s=n1+n2;
16. return s; //returning the sum
17. }
18. }
```

Output:

```
The sum of a and b is= 24
```

Static Method

A method that has static keyword is known as static method. In other words, a method that belongs to a class rather than an instance of a class is known as a static method. We can also create a static method by using the keyword **static** before the method name.

The main advantage of a static method is that we can call it without creating an object. It can access static data members and also change the value of it. It is used to create an instance method. It is invoked by using the class name. The best example of a static method is the **main()** method.

Example of static method

Display.java

```
1. public class Display
2. {
3. public static void main(String[] args)
4. {
5. show();
6. }
7. static void show()
8. {
9. System.out.println("It is an example of static method.");
10. }
11. }
```

Output:

```
It is an example of a static method.
```

Instance Method

The method of the class is known as an **instance method**. It is a **non-static** method defined in the class. Before calling or invoking the instance method, it is necessary to create an object of its class. Let's see an example of an instance method.

InstanceMethodExample.java

```
1. public class InstanceMethodExample
2. {
3. public static void main(String [] args)
4. {
5. //Creating an object of the class
6. InstanceMethodExample obj = new InstanceMethodExample();
7. //invoking instance method
8. System.out.println("The sum is: "+obj.add(12, 13));
9. }
10. int s;
11. //user-defined method because we have not used static keyword
12. public int add(int a, int b)
13. {
14. s = a+b;
15. //returning the sum
16. return s;
17. }
18. }
```

Output:

```
The sum is: 25
```