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.

Types of Method

There are two types of methods in Java:

- Predefined Method
- User-defined Method

JAVA METHODS-

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.

```
public class Main

{

static void myMethod()

 {

System.out.println("Java Programming !");

}

public static void main(String[] args)

 {
```

```
myMethod();

}

}
```

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:

```
public class Main {

  static void myMethod() {

    // code to be executed

  }

}
```

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:

```
public class Main {

  static void myMethod()

 {

    System.out.println("JAVA PROGRAMMING");

  }

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

```
public class Main
{
 static void myMethod()
 {
   System.out.println("JAVA PROGRAMMING!");
  }
 public static void main(String[] args)
 {
   myMethod();
   myMethod();
   myMethod();
  }
}
```

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.

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


  public static void main(String[] args)
{
   myMethod("Disha");
   myMethod("Ravet");
   myMethod("Pimpari");
  }
}
```

Multiple Parameters

```
public class Main {
  static void myMethod(String fname, int age)
 {
   System.out.println(fname + " is " + age);
  }
  public static void main(String[] args) {
   myMethod("Disha", 5);
   myMethod("Jonh", 8);
   myMethod("Anjali", 31);
  }
}
```

Return Values

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:

```
public class Main
 {
  static int myMethod(int x)
 {
    return 5 + x;
  }
  public static void main(String[] args)
{
    System.out.println(myMethod(3));
  }
}
```

This example returns the sum of a method's **two parameters**:

```
public class Main
 {
  static int myMethod(int x, int y)
 {
    return x + y;
  }
  public static void main(String[] args)
 {
    System.out.println(myMethod(5, 3));
  }
}
```

A Method with If...Else

It is common to use `if...else` statements inside methods:

```
public class Main
{
  static void Age(int age)
{
      if (age < 20)
          {
                System.out.println("Access denied - You are not old enough!");
           }
          else
            {
                System.out.println("Access granted - You are old enough!");
             }
           }
  public static void main(String[] args)
{
 Age(23);
  }
}
```

EvenOdd.java

```
import java.util.Scanner;
public class EvenOdd
{
public static void main (String args[])
{
```

```
Scanner scan=new Scanner(System.in);

System.out.print("Enter the number: ");

int num=scan.nextInt();

findEvenOdd(num);

}

public static void findEvenOdd(int num)

{

if(num%2==0)

System.out.println(num+" is even");

else

System.out.println(num+" is odd");

}

}
```

Addition.java

```
public class Addition

{

public static void main(String[] args)

{

int a = 19;

int b = 5;

int c = add(a, b);

System.out.println("The sum of a and b is= " + c);

}
public static int add(int n1, int n2)

{

int s;

s=n1+n2;

return s;

}

}
```

Static Method

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

```
public class Display

{

public static void main(String[] args)

{

show();

}

static void show()

{
```

```
System.out.println("It is an example of static method.");

}

}
```

Java 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.

Use recursion to add all of the numbers up to 10.

```
public class Main

 {

  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;
   }
  }
}
```