Methods in Java

What is Method?

- **Methods in Java** are the building block of a Java application. In Java, a method is a set of code used to write the logic of the applications which perform some specific tasks or operations.
- A method is execute when it is called from another methods.
- The main() method is the first method that is executed by JVM (Java Virtual Machine) in java program. If we want to want to execute any other method, we should call it from the main() method.

Why do we use methods in Java?

Java provide by default one method is called as main() and from main() execution will be started. Primarily we can write our logic of the application in main(). But if we have multiple task in program then we can write multiple logics in program. Writing multiple logic in single method is not good practices. So we can divide our big task in to small task and write a method for each small task. There are several advantages to write methods in program as

Advantages of Methods

1) It Increase readability, reusability and maintainability.
2) It also reduces the length of code and error.
3) Debugging is also easier.

Method Declaration:

A method must be declared inside the class. In general, a method has six fundamental parts such as modifiers, method name, return type, parameter list, exception list, and body.

Syntax:

```
method_modifier   return_type   method_name(Parameter_list) throws Exceptions
{
  // Method body
}
```

Method modifier: It defines the access type of a method but it is optional to use. It may be declared as static, final, abstract, synchronized, or one of the access specifiers private, default, protected, and public.

Note:

1. The public is the least restrictive access modifier and the private is the most.
2. If the method is abstract, the implementation is omitted and the method body is replaced with a single semicolon.

return type: It defines the type of data that comes back from the method after calling it. If the return type is void, the method can perform a task without returning any value. Such a method is called **void method**.
If the return type is non-void but a data-type like int, float, double, etc, the method will return a value. It will not be possible for execution to reach the end of the method body without executing a return statement.

Method Name: It defines the name of method. The method name should be functionality name related to logic.

Parameter list: The parameter list is a comma-separated list of zero or more parameter declaration. The basic form to declare the parameter list as
Syntax:

```
        Parameter-modifier   data-type parameter-name
```

- The parameter-modifier may be final. If the parameter is final, it cannot be modified inside the method. The data-type may be int, float, char, double, etc. Each parameter-name must be a distinct name.
- If there is more than one parameter, they are separated by commas. If there are no parameters, you include an empty pair of parentheses ().
- The scope of the parameter list is only inside the method body but these are optional. The method may also contain zero parameters.

Method Body: The body of the method defines the code or list of statements you need to be executed. It is enclosed within braces.

What is Method Signature in Java?

The method name with parameter list (number of parameters, type of parameters, and order of parameters) is called **method signature in java**.

Types of Methods

There are two types of Methods in Java

1) **Predefined Methods**
2) **User Defined Methods**

Predefined Methods:

Predefined methods in Java are those methods that are already defined in the Java API (Application Programming Interface) to use in an application.

There are numerous predefined methods, such as length(), sqrt(), max(), and print(), and each of them is defined inside their respective classes.

Example 1:

```
public class PredefinedMethods1 {
	public static void main(String args[]) {
		double n=16.4;
```

```
            System.out.println("Square Root:"+Math.sqrt(n));
      }
}
```

Example 2:

```
public class PredefinedMethods2 {
      public static void main(String args[]) {
            int x=13, y=7;
            System.out.println("Maximum:"+Math.max(x, y));
      }
}
```

User Defined Methods

User-defined methods in Java are those methods that are defined inside a class to perform a special task or function in an application. Such methods are called user-defined methods.

A user-defined method is also known as programmer-defined method. We can declare 4 types of user-defined methods in an application.

1. **Instance Method**
2. **Static Method**
3. Abstract Method
4. Factory Method

Instance Method:

An instance method is used to implement behaviours of each object/instance of the class. Since the instance method represents behaviours of the objects. Therefore, instance method is linked with an object.
An instance method is also known as **non-static method**. So, without creating an object of the class, the methods cannot exist to perform their desired behaviours or task. It is allocated in the **heap memory** during the object creation.

Calling Instance Method in Java:

There are two steps to call an instance method in java.
1. First, create an object of a class before calling an instance method of that class.
2. Second, call the method with a reference variable (object) using the dot operator.

```
public class InstanceMethods {

      public static void main(String[] args) {
            InstanceMethods instanceMethods = new InstanceMethods();
            instanceMethods.printMessage();
      }
      void printMessage() {
            System.out.println("Hello Friends..!!");
            System.out.println("Good Morning");
      }
}
```

Static Method (Class Method):

When you declare any method with a static modifier, it is called **static method in java**. A static method is linked with class.
Therefore, it is also known as a class method. It is used to implement the behaviour of the class itself. Static methods load into the memory during class loading and before object creation.

Calling Static Method in Java:

1) To call a static method, we can use dot operator with the class name (or) we can call directly by using name of method from static or non-static method of same class of same program.
2) But if we want to call static method from different program or package the we can use dot operator(.) with class name.

```
public class StaticMethods {
        static void display()
        {
                System.out.println("Hello Friends..!!");
                System.out.println("Good Morning");
        }
        public static void main(String[] args) {
                //StaticMethods.display();
                display();
        }
}
```

Note:
1. Instance method uses dynamic binding (late binding).
2. Class method uses static binding (early binding).

Parameter Passing Technique:

In java, there are 2 technique to pass parameter as
1) Pass by Value
2) Pass by Reference

Pass by Value:

- If we can pass primitive type value to method then it is called as "**Pass by Value**".
- Pass by value is also known as Call by Value.
- Basically, pass by value means that actual value of the variable is passed.

Examples:
Write a method factorial() that will print the factorial of given number.

Formal Parameter: The variable which is declare within Method Header is called as Formal Parameter.

Actual Parameter: The variable which we can pass at the time of method calling is called as Actual Parameter.

```
import java.util.Scanner;
public class CalculateFactorial {

    public static void main(String[] args) {
        int n;
        Scanner scanner = new Scanner(System.in);
        System.out.println("Enter Number:");
        n = scanner.nextInt();

        CalculateFactorial calculateFactorial = new CalculateFactorial();
        calculateFactorial.factorial(n);
        scanner.close();
    }
    void factorial(int x) {
        long f=1;
        while(x!=0)
        {
            f=f*x;
            x--;
        }
        System.out.println("Factorial="+f);
    }
}
```

The above program will be executed successfully and gives correct result but if want to return output value which is f in main function then we need to implement function returning value concept as

```
import java.util.Scanner;
public class CalculateFactorial {
    public static void main(String[] args) {
        int n;
        Scanner scanner = new Scanner(System.in);

        System.out.println("Enter Number:");
        n = scanner.nextInt();

        CalculateFactorial calculateFactorial = new CalculateFactorial();
        long r = calculateFactorial.factorial(n);
                                                          Actual Parameter
        System.out.println("Factorial="+r);
        scanner.close();
    }
    long factorial(int x) {
        long f=1;                        Formal Parameter
        while(x!=0) {
            f=f*x;
            x--;
        }
        return f;
    }
}
```

Pass by Reference:

- In C/C++, We can implement Pass by Reference concept using pointer.
- Java does not support Pass by Reference because java does not support pointer concept in java
- If we want to implement Pass by Reference concept in java then we can use to passed object or array.
- If we can pass non primitive type that includes object of any class to method then it is called as "**Pass by Reference**".
- Pass by reference is also known as Call by Address.
- Basically, pass by reference means the address of memory location is passed where the value of the variable is stored.

Example:

```
/*Write a method swap() to interchange the content of two variable*/
package com.shreesoft.Methods;

import java.util.Scanner;

public class SwapNumber {

	public static void main(String[] args) {
		int x, y;
		Scanner scanner = new Scanner(System.in);
		System.out.println("Enter 2 Number:");
		x = scanner.nextInt();
		y = scanner.nextInt();

		System.out.println("Before swap x="+x+", y="+y);

		swap(x,y);

		System.out.println("After swap x="+x+", y="+y);
	}
	static void swap(int a, int b)
	{
		int t;
		t=a;
		a=b;
		b=t;
	}
}
```

The above program compiled successfully but it gives output incorrect because whatever the changes made on formal parameter a and b will not be affected to actual parameter x and y.

One more problem is that we cannot passed more than one output value from methods. So if we want to achieve more than one result from methods then we can solve by using object passing parameter technique as

```
/*Write a method swap() to interchange the content of two variable*/
package com.shreesoft.Methods;

import java.util.Scanner;
public class SwapNumber {
	int x, y;
	public static void main(String[] args) {
		SwapNumber swapNumber = new SwapNumber();
		Scanner scanner = new Scanner(System.in);

		System.out.println("Enter 2 Number:");
		swapNumber.x = scanner.nextInt();
		swapNumber.y = scanner.nextInt();

		System.out.println("Before swap x="+swapNumber.x+", y="+swapNumber.y);

		swap(swapNumber); // we can pass swapNumber object of SwapNumber class

		System.out.println("After swap x="+swapNumber.x+", y="+swapNumber.y);

		scanner.close();
	}
	static void swap(SwapNumber sw) {
		int t;
		t = sw.x;
		sw.x = sw.y;
		sw.y = t;
	}
}
```

OR

```
/*Write a method swap() to interchange the content of two variable*/
package com.shreesoft.Methods;

import java.util.Scanner;
class Numbers {
	int x, y;
}
public class SwapNumber {

	public static void main(String[] args) {
		Numbers num = new Numbers();

		Scanner scanner = new Scanner(System.in);

		System.out.println("Enter 2 Number:");
		num.x = scanner.nextInt();
		num.y = scanner.nextInt();

		System.out.println("Before swap x="+num.x+", y="+num.y);
		swap(num);
		System.out.println("After swap x="+num.x+", y="+num.y);
		scanner.close();
	}
	static void swap(Numbers n) {
		int t;
		t = n.x;
		n.x = n.y;
		n.y = t;
	}
}

/*Write a program to calculate sum of first n number along with average*/
package com.shreesoft.Methods;

import java.util.Scanner;
class NumberData {
	int n;
	int sum;
	double average;
}
public class CalculateSumAverage {

	public static void main(String[] args) {
		NumberData numberData = new NumberData();

		Scanner scanner = new Scanner(System.in);

		System.out.println("Enter Number:");
		numberData.n = scanner.nextInt();

		sumAverage(numberData);

		System.out.println("Sum="+numberData.sum);
		System.out.println("Average="+numberData.average);

		scanner.close();

	}
	static void sumAverage(NumberData num) {
		int i=1;
		while(i<=num.n)
		{
			num.sum = num.sum + i;
			i++;
		}
		num.average = (double)num.sum/num.n;
	}

}
```