Java Methods

A Java method is a collection of statements that are grouped together to perform an operation

Creating Method

```
public static int methodName(int a, int b) {
   // body
}
```

- **public static** – modifier
- **int** – return type
- **methodName** – name of the method
- **a, b** – formal parameters
- **int a, int b** – list of parameters

- **modifier**:

  It defines the access type of the method and it is optional to use.

- **returnType** :

  Method may return a value.

- **nameOfMethod** :

This is the method name. The method signature consists of the method name and the parameter list.

- **Parameter List** :

  The list of parameters, it is the type, order, and number of parameters of a method. These are optional, method may contain zero parameters.

- **method body** :

  The method body defines what the method does with the statements.

Method Calling

r using a method, it should be called. There are two ways in which a method is called i.e., method returns a value or returning nothing (no return value).

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

      int a = 11;

      int b = 6;

      int c = minFunction(a, b);

      System.out.println("Minimum Value = " + c);

   }


   /** returns the minimum of two numbers */

   public static int minFunction(int n1, int n2) {

      int min;

      if (n1 > n2)
```

```
         min = n2;
      else
         min = n1;


      return min;
   }
```

```
Minimum value = 6
```

The void Keyword:

The void keyword allows us to create methods which do not return a value.

```
public static void main(String[] args) {
      methodRankPoints(255.7);
   }


   public static void methodRankPoints(double points) {
      if (points >= 202.5) {
         System.out.println("Rank:A1");
      }else if (points >= 122.4) {
         System.out.println("Rank:A2");
      }else {
         System.out.println("Rank:A3");
```

```
      }

   }
```

```
Rank:A1
```

Passing Parameters by Value

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

      int a = 30;

      int b = 45;

      System.out.println("Before swapping, a = " + a + " and b = " + b);



      // Invoke the swap method

      swapFunction(a, b);

      System.out.println("\n**Now, Before and After swapping values will
be same here**:");

      System.out.println("After swapping, a = " + a + " and b is " + b);

   }



   public static void swapFunction(int a, int b) {

      System.out.println("Before swapping(Inside), a = " + a + " b = " +
b);



      // Swap n1 with n2

      int c = a;
```

```
      a = b;

      b = c;

      System.out.println("After swapping(Inside), a = " + a + " b = " +
b);

   }
```

```
Before swapping, a = 30 and b = 45
Before swapping(Inside), a = 30 b = 45
After swapping(Inside), a = 45 b = 30

**Now, Before and After swapping values will be same here**:
After swapping, a = 30 and b is 45
```

Method Overloading

When a class has two or more methods by the same name but different parameters, it is known as method overloading. It is different from overriding. In overriding, a method has the same method name, type, number of parameters

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

      int a = 11;

      int b = 6;

      double c = 7.3;

      double d = 9.4;

      int result1 = minFunction(a, b);


      // same function name with different parameters
```

```
      double result2 = minFunction(c, d);
      System.out.println("Minimum Value = " + result1);
      System.out.println("Minimum Value = " + result2);
   }

   // for integer
   public static int minFunction(int n1, int n2) {
      int min;
      if (n1 > n2)
         min = n2;
      else
         min = n1;

      return min;
   }

   // for double
   public static double minFunction(double n1, double n2) {
     double min;
      if (n1 > n2)
         min = n2;
      else
         min = n1;
```

```
      return min;

   }
```

```
Minimum Value = 6
Minimum Value = 7.3
```

The Constructors

A constructor initializes an object when it is created. It has the same name as its class and is syntactically similar to a method. However, constructors have no explicit return type.

All classes have constructors, whether you define one or not, because Java automatically provides a default constructor that initializes all member variables to zero

```
// A simple constructor.

class Shah {

   int x;


   // Following is the constructor

   Shah () {

      x = 10;

   }

}
```

Parameterized Constructor

Most often, you will need a constructor that accepts one or more parameters. Parameters are added to a constructor in the same way that they are added to a method

```
// A simple constructor.

class Shah {

   int x;


   // Following is the constructor

   Shah (int i ) {

      x = i;

   }


public static void main(String args[]) {

      t1 = new Shah ( 10 );

      Shah t2 = new Shah ( 20 );

      System.out.println(t1.x + " " + t2.x);

   }


}
```

The this keyword

this is a keyword in Java which is used as a reference to the object of the current class, with in an instance method or a constructor.
Using *this* you can refer the members of a class such as constructors, variables and methods

he keyword *this* is used only within instance methods or constructors

Differentiate the instance variables from local variables if they have same names, within a constructor or a method.

all one type of constructor (parametrized constructor or default) from other in a class.

```
public class Shah {

   // Instance variable num

   int num = 10;


   Shah () {

      System.out.println("This is an example program on keyword this");

   }


   Shah (int num) {

      // Invoking the default constructor

      this();


      // Assigning the local variable num to the instance variable num
```

```
        this.num = num;
    }

    public void greet() {
        System.out.println("Hi Welcome to java");
    }

    public void print() {
        // Local variable num
        int num = 20;

        // Printing the local variable
        System.out.println("value of local variable num is : "+num);

        // Printing the instance variable
        System.out.println("value of instance variable num is : "+this.num);

        // Invoking the greet method of a class
        this.greet();
    }

    public static void main(String[] args) {
        // Instantiating the class
        Shah obj1 = new Shah ();
```

```
      // Invoking the print method
      obj1.print();

      // Passing a new value to the num variable through parametrized
constructor
      Shah obj2 = new Shah (30);


      // Invoking the print method again
      obj2.print();
   }
}
```

```
This is an example program on keyword this
value of local variable num is : 20
value of instance variable num is : 10
Hi Welcome to Tutorialspoint
This is an example program on keyword this
value of local variable num is : 20
value of instance variable num is : 30
Hi Welcome to java
```

The finalize( ) Method

It is possible to define a method that will be called just before an object's final destruction by the garbage collector. This method is called **finalize( )**

Inside the finalize( ) method, you will specify those actions that must be performed before an object is destroyed.

```
protected void finalize( ) {
   // finalization code here
}
```

he keyword protected is a specifier that prevents access to finalize( ) by code defined outside its class