Java Method is a collection of statements that are grouped together to perform an operation.

- Java methods are where you put the operations on data (variables) in your Java code / In other words, you group Java operations (code) inside Java methods.
- Java methods must be located inside a Java class.
- Similar to what is called functions or procedures in other programming languages
- A group of Java statements that perform some operation on some data, and may or may not return a result.
- **Method definition consists of a method header and a method body.**

THE SYNTAX:

```
modifier returnType nameOfMethod (Parameter List) {
  // method body
}
```

Examples:

```
public static int methodName(int a, int b) {

  // body

}
```

```
public MyClass{


  public void writeText(String text) {

    System.out.print(text);  //prints the text parameter to System.out

  }

}
```

```
class Calculation{
    void sum(int a,int b,int c){
        System.out.println(a+b+c);
    }

  public static void main(String args[]){
        Calculation obj=new Calculation();
        obj.sum(10,10,10);
  }
}

public class ExampleMinNumber {

   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;
  }
}
```

Method Overloading in Java

If a class has multiple methods by same name but different parameters, it is known as Method Overloading. If we have to perform only one operation, having same name of the methods increases the readability of the program.

Advantage: Increases the readability of the program

Different ways to overload the method:

1. By changing number of arguments
2. By changing the data type

SYNTAX:

```
public class DataArtist {
    ...
    public void draw(String s) {
        ...
    }
    public void draw(int i) {
        ...
    }
    public void draw(double f) {
        ...
    }
    public void draw(int i, double f) {
        ...
    }

}
```

Method Overloading by changing the no. of arguments

```
class Calculation{
        void sum(int a,int b){
                System.out.println(a+b);
        }
         void sum(int a,int b,int c){
                System.out.println(a+b+c);
        }
   public static void main(String args[]){
        Calculation obj = new Calculation();
        obj.sum(10,10,10);
        obj.sum(20,20);

  }
}
```

```
class DisplayOverloading{
   public void disp(char c){
       System.out.println(c);
   }
   public void disp(char c, int num){
       System.out.println(c + " "+num);
   }
}

class Sample{
  public static void main(String args[]){
     DisplayOverloading obj = new DisplayOverloading();
     obj.disp('a');
     obj.disp('a',10);
  }
}
```

```
class Overload
{
    void demo (int a)
    {
       System.out.println ("a: " + a);
    }
    void demo (int a, int b)
    {
       System.out.println ("a and b: " + a + "," + b);
    }
    double demo(double a) {
       System.out.println("double a: " + a);
       return a*a;
    }
}
class MethodOverloading
{
    public static void main (String args [])
    {
        Overload Obj = new Overload();
        double result;
        Obj .demo(10);
        Obj .demo(10, 20);
        result = Obj .demo(5.5);
        System.out.println("O/P : " + result);
    }
}
```

Method Overloading by changing data type of argument

//Overloading – Different Number of parameters in argument list

```
class Calculation2{
 void sum(int a,int b){
        System.out.println(a+b);
}
 void sum(double a,double b){
        System.out.println(a+b);
}

  public static void main(String args[]){
          Calculation2 obj=new Calculation2();
          obj.sum(10.5,10.5);
          obj.sum(20,20);

  }
}
```

//Overloading – Difference in data type of arguments

```
class DisplayOverloading2{
   public void disp(char c){
      System.out.println(c);
   }
   public void disp(int c){
     System.out.println(c );
   }
}

class Sample2{
   public static void main(String args[])    {
      DisplayOverloading2 obj = new DisplayOverloading2();
      obj.disp('a');
      obj.disp(5);
   }
```

```
}
```

//Overloading – Sequence of data type of arguments

```
class DisplayOverloading3
{
  public void disp(char c, int num)
  {
     System.out.println("I’m the first definition of method disp");
  }
  public void disp(int num, char c)
  {
      System.out.println("I’m the second definition of method disp" );
  }
}
class Sample3
{
  public static void main(String args[])
  {
     DisplayOverloading3 obj = new DisplayOverloading3();
     obj.disp('x', 51 );
     obj.disp(52, 'y');
  }
}
```

Note: ***In java, Method Overloading is not possible by changing the return type of the method.***

Why Method Overloaing is not possible by changing the return type of method?

In java, method overloading is **may / may not** possible by changing the return type of the method because there may occur ambiguity.

```
class Calculation3{
  int sum(int a, int b){
        return(a+b);
  }
  double sum(int a, int b){
        return(a+b);
  }

  public static void main(String args[]){
         Calculation3 obj=new Calculation3();
         int result1=obj.sum(20,20); //Compile Time Error
         double result2=obj.sum(20,20); //Compile Time Error
        ……
  }
}
```

Constructor in Java

Constructor in java is a *special type of method* that is used to initialize the object.

Java constructor is *invoked at the time of object creation*. It constructs the values i.e. provides data for the object that is why it is known as constructor.

Rules for creating java constructor

There are basically two rules defined for the constructor.

1. Constructor name must be same as its class name
2. Constructor must have no explicit return type

Types of java constructors

There are two types of constructors:

1. Default constructor (no-arg constructor)
2. Parameterized constructor

Java Default Constructor

A constructor that have no parameter is known as default constructor.

Syntax of default constructor: <class_name>(){}

```
class Bike1{
    Bike1(){
        System.out.println("Bike is created");
    }
    public static void main(String args[]){
        Bike1 b=new Bike1();
    }
}
```

Default constructor that displays the default values

```
class Student3{
        int id;
        String name;
        void display(){
              System.out.println(id+" "+name);
        }

   public static void main(String args[]){
        Student3 s1=new Student3();
        Student3 s2=new Student3();
        s1.display();
        s2.display();
   }
}
```

Java parameterized constructor

A constructor that have parameters is known as parameterized constructor.

Why use parameterized constructor?

Parameterized constructor is used to provide different values to the distinct objects.

Example of parameterized constructor

The constructor of Student class that have two parameters. We can have any number of parameters in the constructor.

```
class Student4{
   int id;
   String name;
        Student4(int i,String n){
          id = i;
          name = n;
        }
   void display(){
          System.out.println(id+" "+name);
   }

   public static void main(String args[]){
   Student4 s1 = new Student4(111,"Karan");
   Student4 s2 = new Student4(222,"Aryan");
   s1.display();
   s2.display();
  }
}
```

Constructor Overloading in Java

Constructor overloading is a technique in Java in which a class can have any number of constructors that differ in parameter lists. The compiler differentiates these constructors by taking into account the number of parameters in the list and their type.

Example of Constructor Overloading

```
class Student5{
    int id;
    String name;
    int age;

        Student5(int i,String n){
                id = i;
                name = n;
        }

        Student5(int i,String n,int a){
                id = i;
                name = n;
                age=a;
    }
    void display(){
        System.out.println(id+" "+name+" "+age);
    }

    public static void main(String args[]){
            Student5 s1 = new Student5(111,"Karan");
            Student5 s2 = new Student5(222,"Aryan",25);
            s1.display();
            s2.display();
   }
}
```

Difference between constructor and method in java

Java Constructor	Java Method
Constructor is used to initialize the state of an object.	Method is used to expose behaviour of an object.
Constructor must not have return type.	Method must have return type.
Constructor is invoked implicitly.	Method is invoked explicitly.
The java compiler provides a default constructor if you don't have any constructor.	Method is not provided by compiler in any case.
Constructor name must be same as the class name.	Method name may or may not be same as class name.

Java Copy Constructor

There is no copy constructor in java. But, we can copy the values of one object to another like copy constructor in C++.

There are many ways to copy the values of one object into another in java. They are:

- By constructor
- By assigning the values of one object into another
- By clone() method of Object class

```
class Student6{

   int id;

   String name;

   Student6(int i,String n){

        id = i;

        name = n;
```

```
    }

    Student6(Student6 s){
            id = s.id;
            name =s.name;
    }
    void display(){
        System.out.println(id+" "+name);
    }

   public static void main(String args[]){
        Student6 s1 = new Student6(111,"Karan");
        Student6 s2 = new Student6(s1);
        Student6 s3 = new Student6();
            s3.id = s2.id;
            s3.name = s2.name;

        s1.display();
        s2.display();
        s3.display();
 }
}
```