DEFINING CLASSES IN JAVA

A class is an entity that determines how an object will behave and what the object will contain. A class *is* the basic building block of an object-oriented language such as Java. It is acting as a template that describes the data and behavior associated with instances of that class.

When you instantiate a class means creating an object. The class contains set of variables and methods.

The data associated with a class or object is stored in variables; the behavior associated with a class or object is implemented with methods. A class is a blueprint from which individual objects are created.

```
class MyClass {
   // field,
   //constructor, and
   // method declarations
}
```

Example:

```
class Myclass{
   public static void main(String[] args)
   {
      System.out.println("Hello World!"); //Display the string.
   }
}
```

The keyword class begins the class definition for a class named name. The variables and methods of the class are embraced by the curly brackets that begin and end the class definition block. The "Hello World" application has no variables and has a single method named main.

In Java, the simplest form of a class definition is

```
class name {
   . . .
}
```

In general, class declarations can include these components, in order:

1. ***Modifiers*** *:* A class can be public or has default access.
2. ***Class name:*** The name should begin with a initial letter.

3. ***Superclass(if any):*** The name of the class's parent (superclass), if any, preceded by the keyword extends. A class can only extend (subclass) one parent.

4. ***Interfaces(if any):*** A comma-separated list of interfaces implemented by the class, if any, preceded by the keyword implements. A class can implement more than one interface.

5. ***Body:*** The class body surrounded by braces, { }.

CONSTRUCTORS

Every class has a constructor. If the constructor is not defined in the class, the Java compiler builds a default constructor for that class. While a new object is created, at least one constructor will be invoked. The main rule of constructors is that they should have the samename as the class. A class can have more than one constructor.

Constructors are used for initializing new objects. Fields are variables that provide the state of the class and its objects, and methods are used to implement the behavior of the class and its objects.

Rules for writing Constructor

- Constructor(s) of a class must have same name as the class name in which it resides.
- A constructor in Java cannot be abstract, final, static and synchronized.
- Access modifiers can be used in constructor declaration to control its access i.e which other class can call the constructor.

Following is an example of a constructor −

Example

```
public class myclass {
public myclass() { // Constructor
}
public myclass(String name) {
// This constructor has one parameter, name.
}
}
```

Types of Constructors

There are two type of constructor in Java:

1. No-argument constructor:

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

If the constructor is not defined in a class, then compiler creates default constructor (with no arguments) for the class. If we write a constructor with arguments or no-argument then compiler does not create default constructor. Default constructor provides the default values to the object like 0, null etc. depending on thetype.

```
// Java Program to illustrate calling a no-argument constructor
   import
   java.io.*;class
   myclass
   {
     int num;
   String
   name;
      // this would be invoked while object of that class
     created.myclass()
     {
       System.out.println("Constructor called");
     }
   }
    class myclassmain
   {
     public static void main (String[] args)
     {
       // this would invoke default
       constructor.myclass m1 = new
       myclass();
    // Default constructor provides the default values to the object like 0,
       nullSystem.out.println(m1.num);
       System.out.println(m1.name);
     }
   }
```

2. Parameterized Constructor

A constructor that has parameters is known as parameterized constructor. If we want to initialize fields of the class with your own values, then use parameterized constructor.

```
// Java Program to illustrate calling of  parameterized constructor.
      import
       java.io.*;class
       myclass
      {
         // data members of the
         class.String name;
         int num;
          // contructor with
         arguments.myclass(String
         name, int n)
         {
             this.name =
             name;this.num =
             n;
        }
      }
       class myclassmain{
         public static void main (String[] args)
         {
             // this would invoke parameterized
             constructor.myclass m1 = new
             myclass("Java", 2017);
             System.out.println("Name :" + m1.name + " num :" + m1.num);
         }
         }
```

There are no "return value" statements in constructor, but constructor returns current class instance. We can write 'return' inside a constructor.

CONSTRUCTOR OVERLOADING

Like methods, we can overload constructors for creating objects in different ways. Compiler differentiates constructors on the basis of numbers of parameters, types of the parameters and order of the parameters.

```
// Java Program to illustrate constructor overloading
      import
      java.io.*;class
      myclass
      {
          // constructor with one
          argumentmyclass (String
          name)
          {
              System.out.println("Constructor with one " + "argument - String : " + name);
          }
         // constructor with two
          argumentsmyclass (String
          name, int id)
          {
        System.out.print("Constructor with two arguments : " +" String and Integer : " +
        name
+ " "+ id);
          }
           // Constructor with one argument but with different type than
          previous.myclass (long num)
          {
              System.out.println("Constructor with one argument : " +"Long : " + num);
          }
      }
       class  myclassmain
      {
          public static void main(String[] args)
          {
             myclass m1 = new myclass ("JAVA");
             myclass m2 = new myclass ("Python",
             2017);myclass m3 = new myclass(3261567);
          }
      }
```

Constructors are different from methods in Java

- Constructor(s) must have the same name as the class within which it defined while it is not necessary for the method in java.
- Constructor(s) do not any return type while method(s) have the return type or **void** if does not return any value.
- Constructor is called only once at the time of Object creation while method(s) can be called any numbers of time.

Creating an Object

The class provides the blueprints for objects. The objects are the instances of the class. In Java, the new keyword is used to create new objects.

There are three steps when creating an object from a class −

- ***Declaration*** − A variable declaration with a variable name with an object type.
- ***Instantiation*** − The 'new' keyword is used to create the object.

Initialization − The 'new' keyword is followed by a call to a constructor. This callinitializes the new object