**Constructors in java**

* Introduction

OOP languages like java, c++ provide the facility to treat a composite data type or a user-defined data type like **class** to behave like in-build data types. i.e. we can read the related data of a class directly (like we can read the data of a predefined data type or in-build data type like: - integer (int), float, double, character (char) etc.

Note: - By saying that the in-build or pre-defined datatypes can read the data directly, we mean:

int x=50;

float y=12.05;

we can write these types of statements where the values of those variables of in-build datatypes can be assigned to them directly.

Now, constructor of a class is a special function of a class, which enables the facility to assign the values of the data members of a class directly (to treat use-defined datatype class in the same way we can treat an in-build datatype). So, the need of functions like readdata in a class is eliminated.

* Definition

Constructor is a special member function (method) of a class which is used to initialize data members of an object when a new instance of the object is created or instantiated.

* Constructor is the first method called (must be called) in the lifetime of an object. Construction of an object involves memory allocation of data members and initialization for the object.
* Constructors are called special functions because:
* Constructor has same method name as of the class. (i.e. consider a class “Student”. The constructor of the class will also have the same name. (i.e. “Student”)
* Constructor has no return type
* Constructor is called (invoked) at the time of the creation of the object.
* Constructor is invoked as many times (the default constructor will be invoked or called) as the member of the elements in array of objects.
* Constructor can be overloaded.(function or method overloading)
* A constructor can be of four types: Default constructor, parameterized constructor, copy constructor and private constructor.
* Default constructor

Default constructor is a constructor is a constructor which takes no arguments.

**An example:**

class Person

{

private String Name;

private int Age;

private char Gender;

public Person()//default constructor of the base class

{

Name=null;

Age=0;

Gender='\0';

}

……………………………………………

//these dots indicate other methods and their implementations in the class

……………………………………………

}

class Mainsclass

{

public static void main(String args[])

{

Person person1=new Person();

//creation of the new object named “person1” an initialization of it via default constructor

//Note: this is explicit call of the constructor. Implicit calling of constructor is not possible in //java

…………………………………………….

…………………………………………….

}

}

We create a class “Person”. Now, the “Person” class has the data members as **Name, Age, and Gender.**

Suppose, we create a class named **“Person”** which has data members as Name, Age and Gender. Now, we mention the default constructor and how it is invoked in the example. Another thing to note that, default constructor is included automatically in the class definition if not defined. But, if we define parameterized constructor of a class and then tries to initialize it via automatically included default constructor then it is not possible. In this situation, we need to define the default constructor manually.

import java.io.\*;

class Person

{

private String Name;

private int Age;

private char Gender;

public Person(final String Name,final int Age,final char Gender)

//parameterized constructor of the base class

{

this.Name=Name;

this.Age=Age;

this.Gender=Gender;

}

}

class Mainsclass

{

public static void main(String args[])

{

Person person1=new Person();

//This is not possible in this case. We have to define the default constructor in that case;

}

}

* Parameterized constructor

Parameterized constructor takes arguments and it initializes the data member of the object with the help of the values of the argument passed in the constructor.

* Parameterized constructor

Parameterized constructor takes parameters and initializes the data members through the parameters.

import java.io.\*;

class Person

{

private String Name;

private int Age;

private char Gender;

public Person(final String Name,final int Age,final char Gender)

//parameterized constructor of the base class

{

this.Name=Name;

this.Age=Age;

this.Gender=Gender;

}

}

class Mainsclass

{

public static void main(String args[])

{

Person person1=new Person(“Sayak”,21,’M’);

//This is creation and initialization of object via parameterized constructor

}

}

* Copy Constructor

There are situations when we need to initialize an object’s data member through another object.

Don’t consider the example of “Person” (Because, this would be a faulty example. In the previous example Person class only consists of three data members: Name, Age and Gender. Now, two persons can be same according to these three parameters. But if we consider more parameters like: Father’s Name, Mother’s Name, House Location etc. then two persons cannot be same.) Now, consider the class Complex. Two complex nos can be same (according to their parameters)

class complex

{

private int real;

private int imaginary;

public complex()//this is default constructor

{

real=0;

imaginary=0;

}

public complex(int real,int imaginary)//this is paramterized constructor

{

this.real=real;

this.imaginary=imaginary;

}

public complex(complex c)//this is copy constructor

{

real=c.real;

imaginary=c.imaginary;

}

}

//

class Mainsclass

{

//driver function of class complex

public static void main(String args[])

{

//instantiation of an object via default constructor

complex c1=new complex();

//instantiation of an object via parameterized constructor

complex c2=new complex(1,-2);

//instantiation of object c3 via another object c2, i.e. via copy constructor

complex c3=c2;

}

}

* Private constructor

Private constructor is needed mainly for three purposes:

## When we want to restrict the toil no. of instantiation of a class (i.e. we want to restrict total no. of objects created by the class

* When we want to create only one instance of the class

An example (here, we want to create only one instance of the class)

class Robot

{

private static Robot self;

//Now, java automatically initializes the object reference to null

private String name;

//Hides the default constructor

private Robot(){}

public static Robot Instance()

{

if(self==null)

{

//the Robot object will only be null (automatically initialized) when we create the first instance of robot

//through it

self=new Robot();

}

return self;

}

public void Name(final String name)

{

this.name=name;

}

public String DisplayName()

{

return name;

}

**And now the driver function of it:-**

class Mainsclass

{

public static void main(String args[])

{

Robot a=Robot.Instance();

a.Name(“Jim”); //we name the a instance of the Robot as “jim”

System.out.println(“Robot name for object a:”+a.DisplayName());

Robot b=Robot.Instance();

System.out.println(“Robot name for object b:”+b.DisplayName());

Robot c=Robot.Instance();

System.out.println(“Robot name for object c:”+c.DisplayName());

}

}

**The output of the function:**

**This is because the first instance is actually created. When we try to create instance via b and c, the first instance is got copied to them.**

Robot name for object a: jim

Robot name for object b: jim

Robot name for object c: jim

This is because only the first instance is actually created. When we create more instances of the class via objects b and c, first instance is copied to them.

Now, this is quite tough concept. You may write a driver function like:

class Mainsclass

{

public static void main(String args[])

{

Robot a=Robot.Instance();

a.Name(“Jim”); //we name the a instance of the Robot as “jim”

System.out.println("Robot name for object a:"+a.DisplayName());

Robot b=Robot.Instance();

b.Name("Jack");

System.out.println("Robot name for object b:"+b.DisplayName());

System.out.println("Robot name for object c:"+c.DisplayName());

//you may think that three instances of the class are created

//so, display it again

System.out.println("Displaying again");

System.out.println("Robot name for object a:"+a.DisplayName());

System.out.println("Robot name for object b:"+b.DisplayName());

System.out.println("Robot name for object c:"+c.DisplayName());

}

}

Then check the display. The display/output of the function will be:

Robot name for object a: jim

Robot name for object b: jack

Robot name for object c: jill

Robot name for object a: jill

Robot name for object b: jill

Robot name for object c: jill

i.e. when we do Robot b=Robot.Instance();

then b.Name(“jack”)

then actually rewrites the name of the only instance created (from jim to jack)

Similarly when we do Robot c=Robot.Instance();

Then c.Name(“jill”);

Then again the only instance’s name is changed from jack to jill

And then no changes take place to the name.

So, the instance’s name is remained to jill.

That’s the logic behind such output.

**Another example:**

public class MyClass {

private static final int LIMIT = 10; //Set this to whatever you want to restrict

private static int count = 0;

private MyClass() {}

public static synchronized MyClass getInstance() {

if (count < LIMIT) {

MyClass myClass = new MyClass();

count++;

return myClass;

}

return null;

}

}

Here, total instances of the Myclass is limited to 10.