Unit III

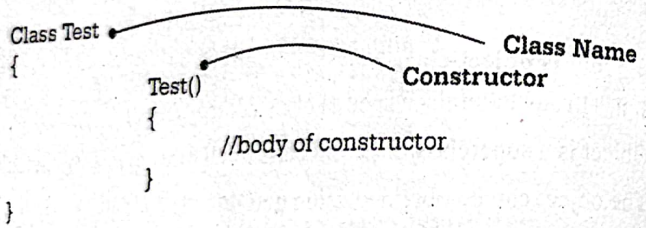
3 Constructors and Destructors

3.1 : Use of Constructor

Q.1 What is constructor ?

Ans. :

- **Definition : The constructor is a specialized method for initializing objects.**
- Name of the constructor is same as that of its class name. In other words, the name of the constructor and class name is same.
- For example -



3.2 : Characteristics of Constructor

Q.2 Enlist any four characteristics of constructor.

Ans. :

1. Name of constructor must be the same as the name of the class for which it is being used.
2. The constructor must be declared in the **public** mode.
3. The constructor gets invoked automatically when an object gets created.

4. The constructor should not have any return type. Even a **void** data type should not be written for the constructor.

3.3 : Types of Constructor

Q.3 What are the types of constructor ? ☞ [SPPU : June-22, Marks 3]

Ans. : There are two types of constructor - (1) No argument constructor and (2) Parameterized constructor

Q.4 Explain the concept of parameterized constructor with an illustrative example. ☞ [SPPU : June-22, Marks 6]

Ans. : Parameterized constructor is a constructor in which the parameters are passed to the constructor function.

Example Program

```
class Person
{
   int id;
   String city;
   Person(int i,String n)
   {
     id=i;
     name=c;
   }
   void display()
   {
    System.out.print("ID: "+id);
    System.out.println(" Name: "+name);
   }
   public static void main(String args[])
   {
    Person p1=new Person(10,"Ankita");
    Person p2=new Person(20,"Prajka");
    p1.display();
    p2.display();
   }
}
```

Output

```
ID: 10 Name: Ankita
ID: 20 Name: Prajka
```

3.4 : Constructor Overloading

Q.5 What is constructor overloading ? Explain.

Ans. :

- Constructor overloading in Java allows to have more than one constructor inside one Class.
- Multiple constructor with different signatures is called as **overloaded constructor.**
- Following is a simple Java program that illustrates the concept of constructor overloading.
- **Example Program**

```
public class Rectangle2
{
    int height,width;
    double ht,wd;
    Rectangle2(int h,int w)  //constructor with two integer //values
    {
     height=h;
     width=w;
    }
    Rectangle2(double h,double w) //constructor with two
    //double values
    {
     ht=h;
     wd=w;
    }
    Rectangle2(int val)  //constructor with single integer value
    {
     height=val;
    }
    void area1()
    {
     System.out.println("Now, The function is called...");
     int result=height*width;
     System.out.println("The area is "+result);
    }
    void area2()
```



```
    {
     System.out.println("Now, The function is called...");
     double result=ht*wd;
     System.out.println("The area is "+result);
    }
    void area3()
    {
     System.out.println("Now, The function is called...");
     int result=height*height;
     System.out.println("The area is "+result);
    }
}
class OverLoadConstr
{
public static void main(String args[])
 {
    Rectangle2 obj1=new Rectangle2(11,20);
    obj1.area1(); //call to the method
    Rectangle2 obj2=new Rectangle2(11.33,20.22);
    obj2.area2(); //call to the method
    Rectangle2 obj3=new Rectangle2(10);
    obj3.area3(); //call to the method
 }
}
```

Output

```
Now, The function is called...
The area is 220
Now, The function is called...
The area is 229.09259999999998
Now, The function is called...
The area is 100
```

3.5 : Dynamic Initialization of Objects

Q.6 In Java, dynamic initialization of objects is possible. Justify.

Ans. : Dynamic initialization of object means initializing the data members of class while creating the object. That means when we provide initial values to the data members of the class during creation of the object – we dynamically initialize the objects.

Following program illustrates the initialization of two objects.

Java Program

```
class Addition
{
    private int a;
    private int b;
    public Addition(int x,int y)
    {
        a=x;
        b=y;
    }
    public void display()
    {
        System.out.println("a = "+a);
        System.out.println("b = "+b);
    }
}
class Main {
    public static void main(String[] args) {
        Addition add1=new Addition(10,20);
        System.out.println("Initialization of first object");
        add1.display(); //output : a = 10, b = 20
        System.out.println("Initialization of second object");
        Addition add2=new Addition(100,200);
        add2.display(); //output : a = 100, b = 200
    }
}
```

3.6 : Constructor with Default Arguments

Q.7 Can we pass default argument to a constructor in Java ? Explain how.

Ans. : The constructor can be defined by passing the default arguments to it.

However, in C++ we can implement constructor with default arguments. Consider following example which illustrates the use of default argument constructor.

C++ Program

```
#include <iostream>
using namespace std;

class Test
{
private:
    int a;
    int b;
    int c;
public :
    Test(int x=10,int y=20); //declaration of constructor with default
arguments
    void display()
    {
        cout<<"\n a= "<<a;
        cout<<"\n b= "<<b;
        cout<<"\n c= "<<c;
    }
};
Test::Test(int x,int y)
{
    a=x;
    b=y;
    c=a+b;
}
int main()
{
    class Test obj1,obj2(100);
    obj1.display();
    cout<<"\n";
    obj2.display();
    return 0;
}
```

3.7 : Symbolic Constants

Q.8 Write a short note on - Symbolic constants.

Ans. :

- There are two problems that are associated with the constants and those are -
  1. **Modifiability :** The constants may need to be modified during the program. For example : The value of pi may be 3.14 at one place in the program and it may be required as 3.14159 at some other place in the same program for the accuracy.
  2. **Understandability :** The purpose of each constant need to be remembered. For example : The constant *INDEX* may be required to store the index, the constant *COUNT* may keep track of some count or there may be some constant for *SIZE*.
- To overcome these problems there is a use of **symbolic constants**. The keyword **final** is used to declare the symbolic constants.
- The syntax for declaring the symbolic constants is -

```
final datatype variable_name=value
```

- For example -

```
final int INDEX=1;
final int SIZE=10;
```

3.8 : Garbage Collection : Destructor and Finalizers

Q.9 How garbage collection is done in Java ?

Ans. :

- Garbage collection is a method of automatic memory management.
- It works as follows -
  1. When an application needs some free space to allocate the nodes and if there is no free space available to allocate the memory for these objects then a system routine called **garbage collector** is called.

  2. This routine then searches the system for the free space. The free space is then made available for reuse.
- Java used **finalize()** method for garbage collection.

END...