

Experiment No._____

Date___/___/2020

TITLE OF EXPERIMENT: - A program in Java to demonstrate the method and constructor overloading

DIVISION:_____ **BRANCH:** _____

BATCH:_____ **ROLL NO.:** _____

PERFORMED ON DATE: _____

SIGNATURE OF TEACHING STAFF:

EXPERIMENT NO. 4

Aim: Write a program in JAVA to demonstrate the method and constructor overloading

Software:

1. Eclipse
2. JDK 16

Theory:

Methods, Constructors can also be overloaded. Overloaded constructor is called based upon the parameters specified when new is executed.

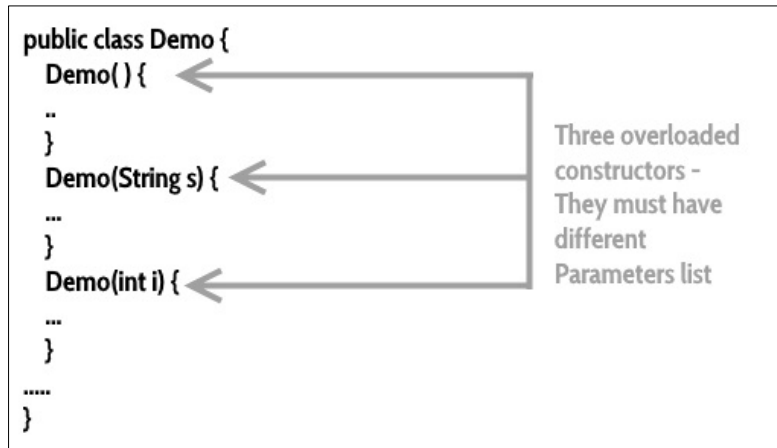
CONSTRUCTOR is a special method that is used to initialize a newly created object and is called just after the memory is allocated for the object. It can be used to initialize the objects to desired values or default values at the time of object creation. It is not mandatory for the coder to write a constructor for a class.

If no user-defined constructor is provided for a class, compiler initializes member variables to its default values.

- numeric data types are set to 0
- char data types are set to null character('\0')
- reference variables are set to null

Rules for creating a Java Constructor

1. It has the **same name** as the class
2. It should not return a value not even *void*
3. Constructor calling must be the **first** statement of constructor in Java.
4. If we have defined any parameterized constructor, then compiler will not create default constructor. and vice versa if we don't define any constructor, the compiler creates the default constructor(also known as no-arg constructor) by default during compilation



When do we need Constructor Overloading?

Sometimes there is a need of initializing an object in different ways. This can be done using constructor overloading. For example, Thread class has 8 types of constructors. If we do not want to specify anything about a thread then we can simply use default constructor of Thread class, however if we need to specify thread name, then we may call the parameterized constructor of Thread class with a String args like this:

```
Thread t= new Thread (" MyThread ");
```

Let us take an example to understand need of constructor overloading. Consider the following implementation of a class Box with only one constructor taking three arguments.

```
// An example class to understand need of constructor overloading.  
class Box  
{  
    double width, height,depth;  
  
    // constructor used when all dimensions specified  
    Box(double w, double h, double d)  
    {  
        width = w;  
        height = h;  
        depth = d;  
    }  
}
```

```
// compute and return volume
double volume()
{
    return width * height * depth;
}
}
```

As we can see that the Box() constructor requires three parameters. This means that all declarations of Box objects must pass three arguments to the Box() constructor. For example, the following statement is currently invalid:

```
Box ob = new Box();
```

Since Box() requires three arguments, it's an error to call it without them. Suppose we simply wanted a box object without initial dimension, or want to initialize a cube by specifying only one value that would be used for all three dimensions. From the above implementation of Box class these options are not available to us.

These types of problems of different ways of initializing an object can be solved by constructor overloading. Below is the improved version of class Box with constructor overloading.

```
// Java program to illustrate
// Constructor Overloading

class Box
{
    double width, height, depth;

    // constructor used when all dimensions specified

    Box(double w, double h, double d)
```

```
{

    width = w;

    height = h;

    depth = d;

}


// constructor used when no dimensions specified

Box()

{

    width = height = depth = 0;

}


// constructor used when cube is created

Box(double len)

{

    width = height = depth = len;

}


// compute and return volume

double volume()
```

```
{  
  
    return width * height * depth;  
  
}  
  
}
```

Program:

Filename: Test.java

```
class Rectangle {  
  
    private int length;  
    private int breadth;  
  
    public Rectangle(int side) {  
        length = side;  
        breadth = side;  
    }  
  
    public Rectangle(int l, int b) {  
        length = l;  
        breadth = b;  
    }  
  
    public int getArea() {  
        return length * breadth;  
    }  
}  
  
class Test {  
  
    public static void main(String[] args) {  
        Rectangle rect = new Rectangle(4, 5);  
        Rectangle sq = new Rectangle(5);  
  
        System.out.println(rect.getArea());  
        System.out.println(sq.getArea());  
    }  
}
```

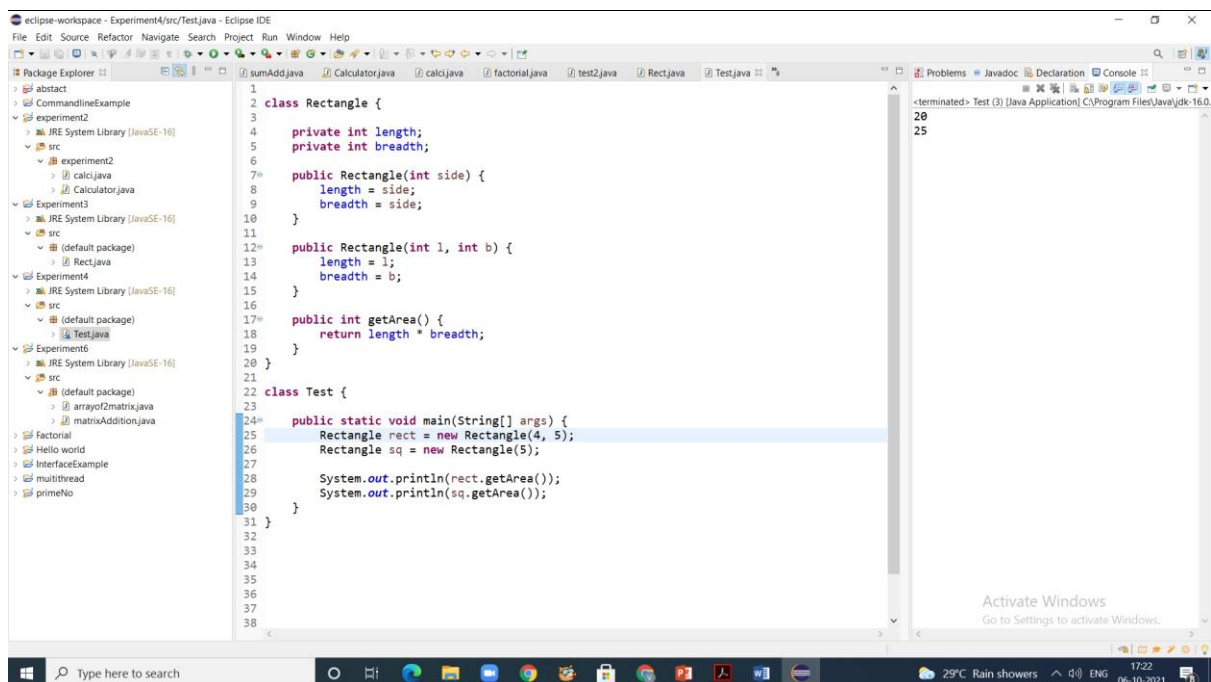
Output:

20

25

Conclusion:

Screenshot's of Program and Result:



The screenshot displays the Eclipse IDE interface. The Package Explorer on the left shows a project named 'Experiment4' with a source folder 'src' containing a file 'Test.java'. The main editor window shows the code for 'Test.java'. The code defines a 'Rectangle' class with private attributes 'length' and 'breadth', a constructor 'Rectangle(int side)', a constructor 'Rectangle(int l, int b)', and a method 'getArea()'. The 'Test' class contains a 'main' method that creates a 'Rectangle' object with side 4, a square object with side 5, and prints their areas. The Console window on the right shows the output of the program: '20' and '25'. The status bar at the bottom indicates the system temperature is 29°C, there are rain showers, and the date is 06-10-2021.

```
1 class Rectangle {
2
3
4     private int length;
5     private int breadth;
6
7     public Rectangle(int side) {
8         length = side;
9         breadth = side;
10    }
11
12    public Rectangle(int l, int b) {
13        length = l;
14        breadth = b;
15    }
16
17    public int getArea() {
18        return length * breadth;
19    }
20 }
21
22 class Test {
23
24     public static void main(String[] args) {
25         Rectangle rect = new Rectangle(4, 5);
26         Rectangle sq = new Rectangle(5);
27
28         System.out.println(rect.getArea());
29         System.out.println(sq.getArea());
30     }
31 }
32
33
34
35
36
37
38
```

20
25