

**Name:** Shaikh Salif Aminuddin

**CLASS:** S2    **ROLL NO:** 2201094

## **EXPERIMENT NO: 4**

**AIM:** Program on method and constructor overloading

### **THEORY:**

**Method Overloading** : If a class has multiple methods having same name but different in 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. Suppose you have to perform addition of the given numbers but there can be any number of arguments, if you write the method such as a(int,int) for two parameters, and b(int,int,int) for three parameters then it may be difficult for you as well as other programmers to understand the behavior of the method because its name differs. So, we perform method overloading to figure out the program quickly. Advantage of method overloading Method overloading increases the readability of the program. Different ways to overload the method There are two ways to overload the method in java-

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

### **Constructor Overloading :**

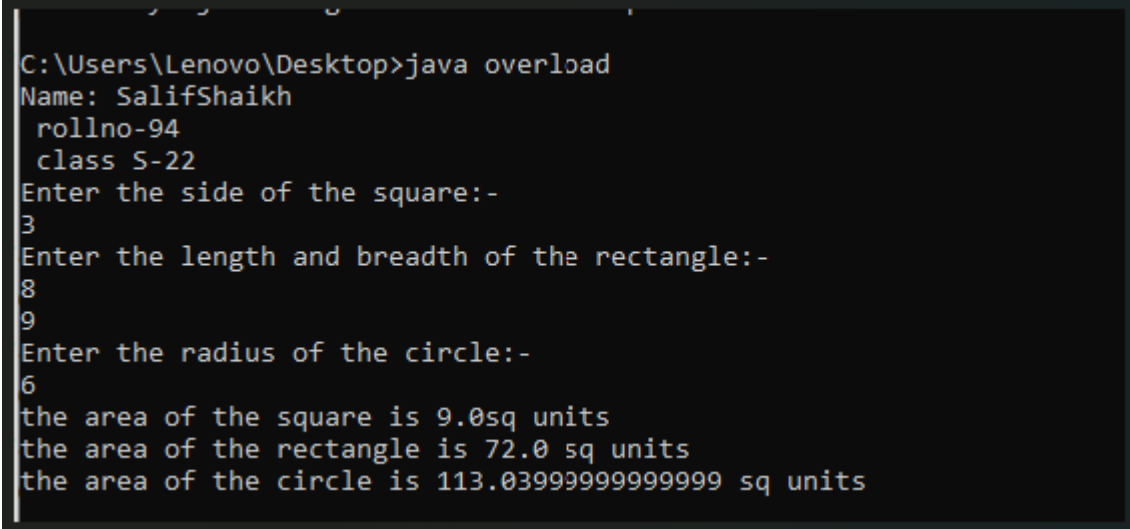
The constructor overloading can be defined as the concept of having more than one constructor with different parameters so that every constructor can perform a different task. Sometimes, we need to use multiple constructors to initialize the different values of the class. We must also notice that the java compiler invokes a default constructor when we do not use any constructor in the class. However, the default constructor is not invoked if we have used any constructor in the class, whether it is default or parameterized. In this case, the java compiler throws an exception saying the constructor is undefined.

## CODE-01: Method Overloading

```
class area
{
    area(int l, int b)
    {
        int arearect = l*b;
        System.out.println(arearect);
    }
    area(int s)
    {
        int areasq = s*s;
        System.out.println(areasq);
    }
    area(float r)
    {
        float pi = 3.14f;
        float areac= pi * r*r;
        System.out.println(areac);
    }
    public static void main(String[] args)
    {
        System.out.println("Nmae-SalifShaikh \n roll no-94\n class s-22");
        area c1 = new area(5, 4);
        area c2 = new area(5);
    }
}
```

```
area c3 = new area(6.0f);  
}  
}
```

## OUTPUT:



```
C:\Users\Lenovo\Desktop>java overload  
Name: SalifShaikh  
rollno-94  
class S-22  
Enter the side of the square:-  
3  
Enter the length and breadth of the rectangle:-  
8  
9  
Enter the radius of the circle:-  
6  
the area of the square is 9.0sq units  
the area of the rectangle is 72.0 sq units  
the area of the circle is 113.03999999999999 sq units
```

## CODE-02: Constructor Overloading

```
import java.util.Scanner;  
  
public class overload  
{  
    void area(float x)  
    {  
        System.out.println("the area of the square is "+ x*x +"sq units");  
    }  
    void area(float x, float y)  
    {  
        float k = x*y; System.out.println("the area of the rectangle is "+ k +" sq units");  
    }  
}
```

```
void area(double x) { double z = 3.14 * x * x;
System.out.println("the area of the circle is "+z+" sq units");
}
public static void main(String[] args)
{
System.out.println("Name: Salif Shaikh\n rollno-94\n class S-22");
Scanner s = new Scanner(System.in);
System.out.println("Enter the side of the square:-");
float x = s.nextFloat();
System.out.println("Enter the length and breadth of the rectangle:-");
float l = s.nextFloat();
float b = s.nextFloat();
System.out.println("Enter the radius of the circle:-");
double r = s.nextDouble();
overload ob = new overload();
ob.area(x);
ob.area(l,b);
ob.area(r);
}
}
```

**OUTPUT:**

```
C:\Users\Lenovo\Desktop>javac area.java
```

```
C:\Users\Lenovo\Desktop>java area
```

```
Nmae-Salif Shaikh
```

```
roll no-94
```

```
class s-22
```

```
20
```

```
25
```

```
113.04
```

**CONCLUSION:** Constructor overloading is a beneficial technique in object-oriented programming that permits the creation of multiple constructors within a class, each with different parameter combinations. This facilitates the creation of objects with varying initialization requirements while maintaining a unified interface.