Experiment No. 4
Implement a program on method and constructor overloading.
Date of Performance:
Date of Submission:

Aim: Implement a program on method and constructor overloading.

Objective: To use concept of method overloading in a java program to create a class with same function name with different number of parameters.

Theory:
Method Overloading is a feature that allows a class to have more than one method having the same name, if their argument lists are different. It is similar to constructor overloading in Java, that allows a class to have more than one constructor having different argument lists.
Example: This example to show how method overloading is done by having different number of parameters for the same method name.

```
Class DisplayOverloading
{
   public void disp(char c)
   {
      System.out.println(c);
   }
   public void disp(char c, int num)
   {
      System.out.println(c + " "+num);
   }
}
Class Sample
{
  Public static void main(String args[])
  {
     DisplayOverloading obj = new DisplayOverloading();
     Obj.disp('a');
     Obj.disp('a',10);
  }
}
```

Output:

```
A
A 10
```

Java supports Constructor Overloading in addition to overloading methods. In Java, overloaded constructor is called based on the parameters specified when a new is executed.

Sometimes there is a need of initializing an object in different ways. This can be done using constructor overloading.

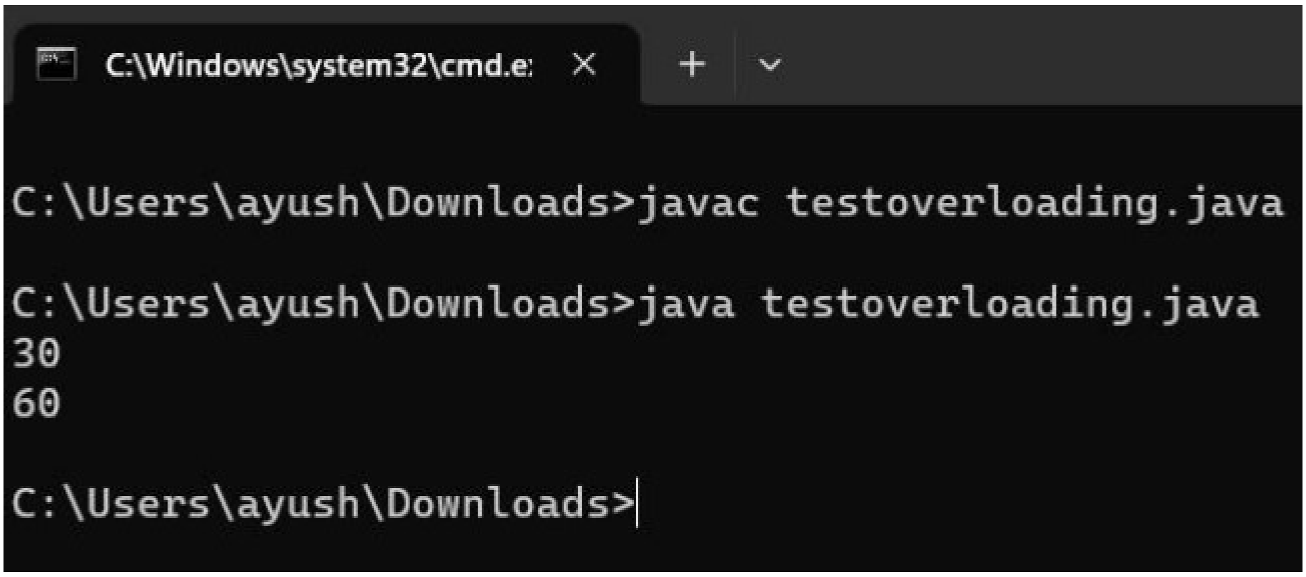
For example, the Thread class has 8 types of constructors. If we do not want to specify anything about a thread then we can simply use the default constructor of the Thread class, however, if we need to specify the thread name, then we may call the parameterized constructor of the Thread class with a String args like this:

Thread t= new Thread (" MyThread ");

Code:

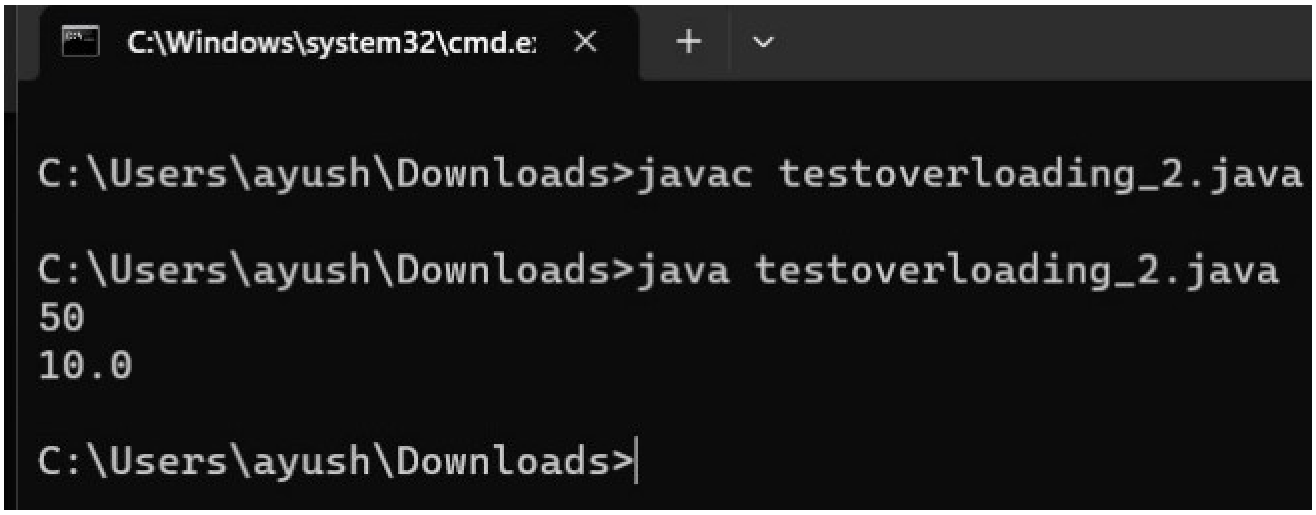
```
1. class testoverloading{
 public static void main(String args[]){
System.out.println(adder.add(10,20));
 System.out.println(adder.add(10,20,30));
 }
 }
 class adder{
 static int add(int a,int b){
return a+b;
}
 static int add(int a,int b,int c){
return a+b+c;
}
 }
```

OUTPUT:



```
C:\Windows\system32\cmd.e:

C:\Users\ayush\Downloads>javac testoverloading.java

C:\Users\ayush\Downloads>java testoverloading.java
30
60

C:\Users\ayush\Downloads>
```

2. class testoverloading_2{
 public static void main(String args[]){
System.out.println(operation.sub(100,50));
 System.out.println(operation.mul(2,5));
 }
 } class operation{
 static int sub(int a, int b){
return a-b;
}
 static double mul(int a, int b){
return a*b;
}
 }
 OUTPUT:

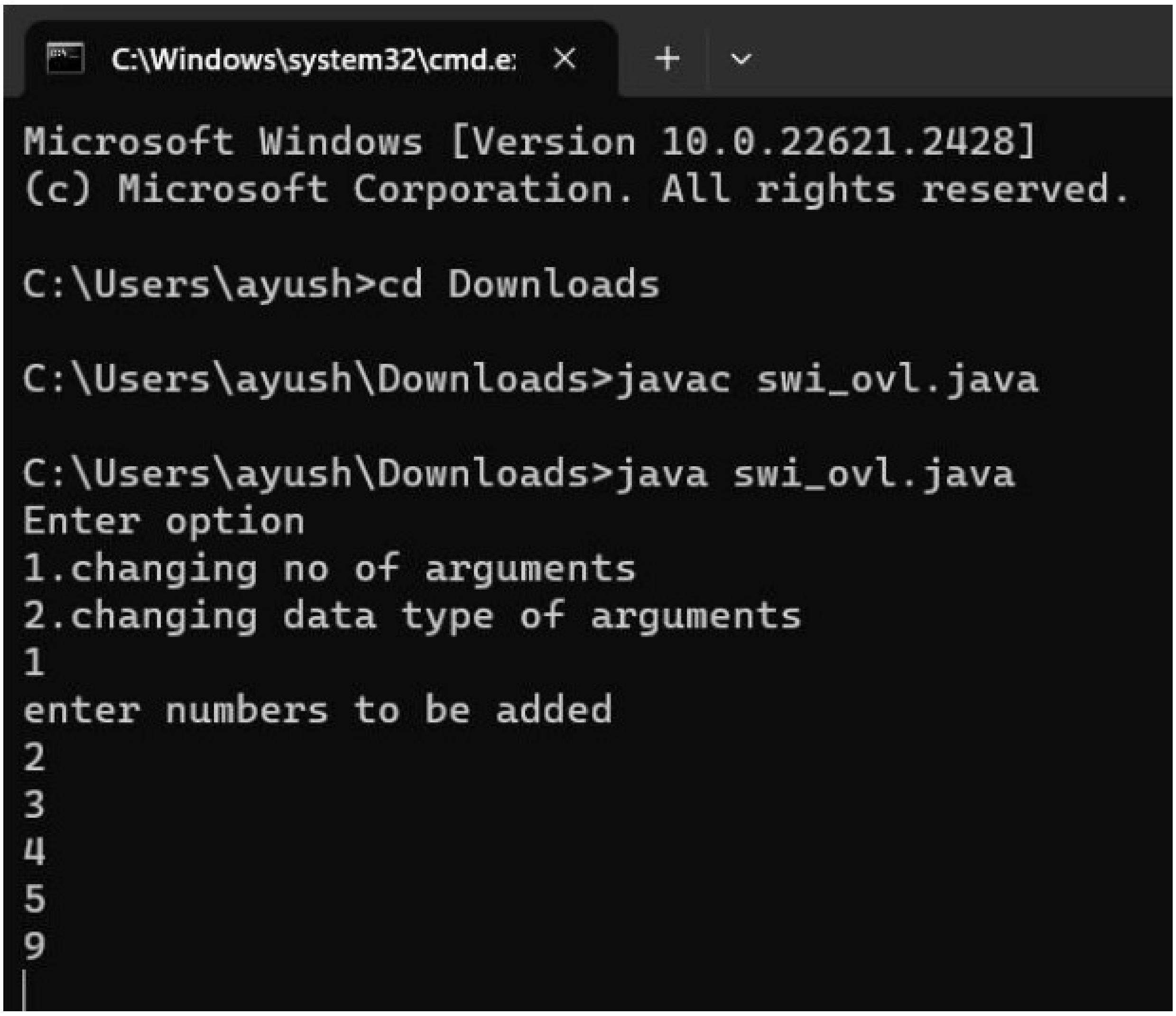


```
C:\Windows\system32\cmd.e:

C:\Users\ayush\Downloads>javac testoverloading_2.java

C:\Users\ayush\Downloads>java testoverloading_2.java
50
10.0

C:\Users\ayush\Downloads>
```

3. import java.util.*;

```
class swi_ovl {
 public static void main(String args []){
Scanner sc=new Scanner(System.in);
System.out.print("Enter option\n1.changing no of arguments\n2.changing data type of arguments\n");
int opt=sc.nextInt();
 switch(opt) {
 case 1: {
 System.out.println("enter numbers to be added");
int x=sc.nextInt();
 int y=sc.nextInt();
 int z=sc.nextInt();
 System.out.println(adder.add(x,y));
 System.out.println(adder.add(x,y,z));
 } case 2: {
 int a =sc.nextInt();
 int b =sc.nextInt();
 double n = sc.nextDouble();
 double m = sc.nextDouble();
 System.out.println(operation.add(a,b));
 System.out.println(operation.add(n,m));
}
}
}
} class adder {
 static int add(int x,int y){
return x+y;}
static int add(int x,int y,int z){
return x+y+z;
}
 }
 class operation {
 static int add(int a,int b){
return a+b;
} static double add(double n,double m){
return n+m;
}
 }
```

OUTPUT:



```
Microsoft Windows [Version 10.0.22621.2428]
(c) Microsoft Corporation. All rights reserved.

C:\Users\ayush>cd Downloads

C:\Users\ayush\Downloads>javac swi_ovl.java

C:\Users\ayush\Downloads>java swi_ovl.java
Enter option
1.changing no of arguments
2.changing data type of arguments
1
enter numbers to be added
2
3
4
5
9
```

Conclusion:

Comment on how function and constructor overloading used using java

In Java, function and constructor overloading involve creating multiple methods or constructors within a class, all sharing the same name but differing in the number or type of their parameters.

Function Overloading:
- Function overloading allows me to define several methods with the same name but distinct argument lists.
- This is useful when I want to perform similar operations on different types of input data.
- Java determines which method to execute based on the number or types of arguments passed during the function call.

Constructor Overloading:
- Constructor overloading applies the same concept to class constructors.
- I can create multiple constructors for a class, each accepting different sets of parameters.
- This enables flexibility when creating objects, as I can initialize them in various ways based on the constructor used.

In both cases, overloading enhances code reusability and readability, making it easier to work with different input scenarios while maintaining a consistent method or class name.