Experiment No. 4
Implement a program on method and constructor overloading.
Date of Performance: 22/08/2024
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:

```
import java.util.Scanner;

class Overload
{
   String name;
   int age;

   // Constructor Overloading
   Overload()
   {
      name = "Unknown";
      age = 0;
   }

   Overload(String name)
   {
      this.name = name;
      this.age = 0;  // Default age
   }

   Overload(String name, int age)
   {
      this.name = name;
      this.age = age;
   }

   void displayInfo()
   {

      System.out.println("Name: " + name + "\nAge: " + age);
   }
}

public class Overloading
{
   public static void main(String[] args)
   {
      Scanner scanner = new Scanner(System.in);

      System.out.println("Enter Name and Age for person 1:");
      String name1 = scanner.nextLine();
      int age1 = scanner.nextInt();
      scanner.nextLine();
```

```
        Overload person1 = new Overload(name1, age1);

        System.out.println("Enter Name and Age for person 2:");
        String name2 = scanner.nextLine();
        int age2 = scanner.nextInt();

        System.out.println();

        Overload person2 = new Overload(name2, age2);

        // Display information
        System.out.println("Information for person 1:");
        person1.displayInfo();
        System.out.println();
        System.out.println("Information for person 2:");
        person2.displayInfo();

        scanner.close();
    }
}
```

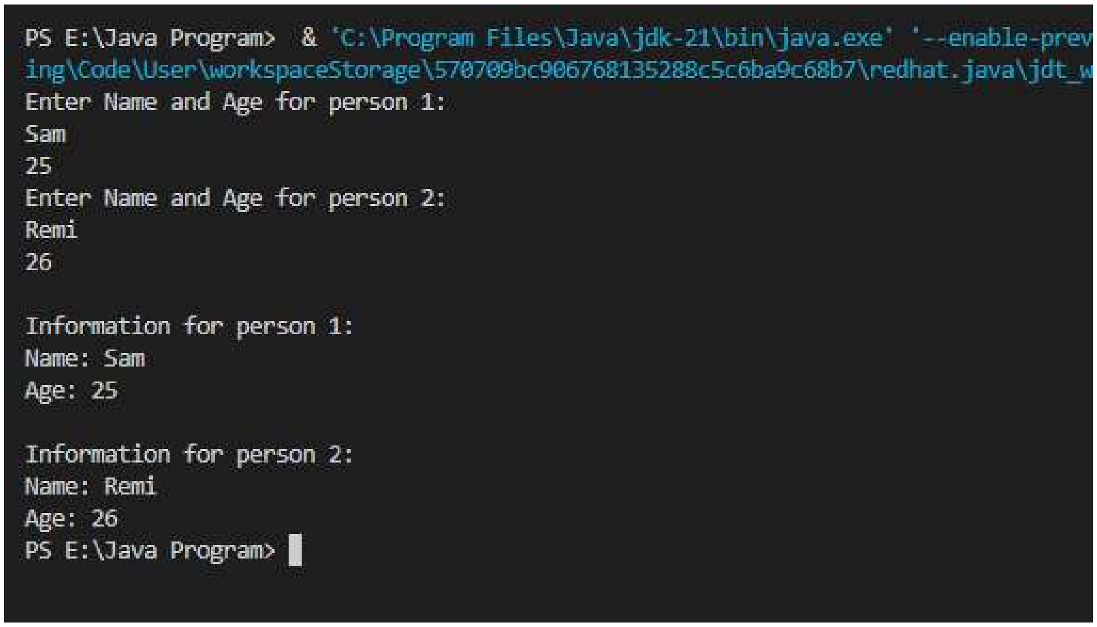
Output:

```
Enter Name and Age for person 1:
Sam
25
Enter Name and Age for person 2:
Remi
26

Information for person 1:
Name: Sam
Age: 25

Information for person 2:
Name: Remi
Age: 26
```

Screenshot:



```
PS E:\Java Program>  & 'C:\Program Files\Java\jdk-21\bin\java.exe' '--enable-prev
ing\Code\User\workspaceStorage\570709bc906768135288c5c6ba9c68b7\redhat.java\jdt_w
Enter Name and Age for person 1:
Sam
25
Enter Name and Age for person 2:
Remi
26

Information for person 1:
Name: Sam
Age: 25

Information for person 2:
Name: Remi
Age: 26
PS E:\Java Program>
```

Conclusion:

Comment on how function and constructor overloading used using java
Ans:

Function and constructor overloading in Java allow a class to have multiple methods or constructors with the same name but different parameter lists. Method overloading enables methods to perform similar tasks with varying inputs, improving code readability and reusability. Constructor overloading allows creating objects with different initializations, depending on the arguments provided. Both techniques enhance flexibility and help manage different use cases effectively within a single class.