|  |  |  |  |
| --- | --- | --- | --- |
| **Course Name:** | **Object Oriented Programming** | **Semester:** | **III** |
| **Date of Performance:** | **07 / 08 / 2023** | **Batch No:** | **A - 3** |
| **Faculty Name:** | **Prof. Pragya Gupta** | **Roll No:** | **16014022050** |
| **Faculty Sign & Date:** |  | **Grade/Marks:** | **\_\_\_ / 25** |

|  |  |  |
| --- | --- | --- |
| **Writing Program (07)** | **Performance in lab**  **and Viva (05+03)** | **Post lab questions, conclusion and completion (03+02+05)** |
|  |  |  |

**Experiment No: 3**

**Title: Concept of static method and recursive function**

|  |
| --- |
| **Aim and Objective of the Experiment:** |
| Learn the concept of static method and recursive function in Java |

|  |
| --- |
| **COs to be achieved:** |
| **CO1**: Understand concepts of Object Oriented Programming and basic characteristics of Java. |

|  |
| --- |
| **Tools used:** |
| 1. Java Development Kit (JDK) 2. Visual Studio Code 3. [Difference between static and non-static method in Java - GeeksforGeeks](https://www.geeksforgeeks.org/difference-between-static-and-non-static-method-in-java/) 4. <https://www.javatpoint.com/command-line-argument> |

|  |
| --- |
| **Theory:** |
| 1. **(About static method)**   In Java, a static method is a method that belongs to a class rather than an instance of that class. This means you can call a static method on the class itself, without needing to create an object of that class. Here are some key points to understand about static methods in Java:   * **Declaration and Definition**: You define a static method using the static keyword before the method signature. The method can be accessed using the class name followed by the method name, like ClassName.methodName(). * **No Instance Required**: Since static methods belong to the class and not to instances of the class, you can call them without creating an object of the class. * **Accessing Members**: Within a static method, you can only access other static members (variables or methods) of the same class. You cannot access non-static (instance) members directly from a static context. However, you can access instance members through an object reference. * **Main Method**: The main method in Java is a common example of a static method. It's the entry point of a Java program, and you can call it without creating any objects. * **Utility Methods**: Static methods are often used for utility functions that perform operations related to the class but don't depend on specific instance data. * **Polymorphism**: Static methods cannot be overridden because they are associated with the class itself and not with instances. This means the behavior of a static method is determined at compile-time based on the reference type, not at runtime based on the object's type. * **Class Loading**: Static methods are loaded into memory when the class is loaded, and they remain in memory throughout the program's execution.   Here's a simple example of a class with a static method:  public class MathUtils {  public static int add(int a, int b) {  return a + b;  }  public static void main(String[] args) {  int result = MathUtils.add(5, 3);  System.out.println("Sum: " + result);  }  }  In this example, the add method is a static method that can be called using the class name (MathUtils.add()). The main method is also static, serving as the program's entry point.  Static methods are a fundamental concept in Java and are widely used for various purposes, including providing utility functions, performing operations at the class level, and initializing class-specific resources. |

|  |
| --- |
| **Code:** |
| 1. Write a recursive function ‘gcd’ to find the gcd of the given two numbers. Use this in main to find the gcd and LCM two given numbers.    1. Variation 1: Implementation with One class only   class exp3\_q1  {      int gcdlcm(int a, int b)      {          if (b == 0)              return a;          else              return gcdlcm(b, a % b);      }      public static void main(String args[])      {          System.out.println("enter two numbers as command-line arguments");          exp3\_q1 g2 = new exp3\_q1();          int num1 = Integer.parseInt(args[0]);          int num2 = Integer.parseInt(args[1]);          int gcd = g2.gcdlcm(num1, num2);          int lcm = (num1 \* num2) / gcd;          System.out.println("GCD: " + gcd);          System.out.println("LCM: " + lcm);      }  }   * 1. Variation 2: Accessibility with static and non-static methods within class and outside class.   class GCD  {      // static method to find GCD      static int gcd(int a, int b)      {          if (b == 0)              return a;          else              return gcd(b, a % b);      }      // non-static method to find LCM      int lcm(int a, int b)      {          return (a \* b) / gcd(a, b);      }  }  public class exp3\_q2  {      public static void main(String []args)      {          int num1 = Integer.parseInt(args[0]);          int num2 = Integer.parseInt(args[1]);          int gcd = GCD.gcd(num1, num2); // accessing static          GCD g2 = new GCD(); // creating an instance          int lcm = g2.lcm(num1, num2); // accessing non-static method          System.out.println("GCD is: " + gcd);          System.out.println("LCM is: " + lcm);      }  }  Note : For input use *command-line-argument* instead of *scanner* |

|  |
| --- |
| **Output:** |
|  |

|  |
| --- |
| **Post Lab Subjective/Objective type Questions:** |
| 1. **Write a recursive static method for calculation of factorial of n number.**   import java.util.\*;  public class exp3\_plq  {      static int factorial(int num)      {          if (num == 0 || num == 1) {              return 1;          } else {              return num \* factorial(num - 1);          }      }      public static void main(String[] args)      {          Scanner sc = new Scanner(System.in);          System.out.print("\nenter a number: ");          int num = sc.nextInt();          int result = factorial(num);            System.out.println("factorial: " + result);          sc.close();      }  }     1. **What is command-line-argument and how are they used?**   Command-line arguments are values provided to a program when it is executed from the command line or terminal. They allow you to pass specific inputs to your program without modifying the source code. Command-line arguments are commonly used to provide configuration settings, parameters, filenames, or any other information that a program might need to operate on.  When you run a program from the command line, you typically provide the program's name followed by one or more arguments. These arguments are separated by spaces. The program can then access and use these arguments during its execution.  Here's how command-line arguments are used:   * **Provide Arguments:** Run the program by typing its name followed by arguments, separated by spaces:   java MyProgram arg1 arg2 arg3   * **Access in Code:** Inside your program, you can access these command-line arguments using the args parameter of the main method. The args parameter is an array of strings where each element corresponds to a command-line argument.   public class MyProgram {  public static void main(String[] args) {  // args is an array of strings containing command-line arguments  for (int i = 0; i < args.length; i++) {  System.out.println("Argument " + i + ": " + args[i]);  }  }  }   * **Use as Needed:** Utilize the arguments in your program. Convert them if needed (e.g., to numbers) and use them in calculations, processing files, etc.   java Calculator 10 20  Command-line arguments enhance a program's flexibility by allowing users to customize its behavior without modifying its source code each time. |

|  |
| --- |
| **Conclusion:** |
| In these exercises, we've learned about static and non-static methods in Java, how they help organize code and work with objects. We've also used command-line arguments to make programs more flexible, improving our practical Java programming skills. |

|  |
| --- |
| **Signature of faculty in-charge with Date:** |