|  |  |  |  |
| --- | --- | --- | --- |
| **Course Name:** | **Object Oriented Programming** | **Semester:** | **III** |
| **Date of Performance:** | **31 / 07 / 2023** | **Batch No:** | **Batch 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: 2**

**Title: First Java code use of Class and Object**

|  |
| --- |
| **Aim and Objective of the Experiment:** |
| Learn the concept of encapsulation 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. [Java Tutorial (w3schools.com)](https://www.w3schools.com/java/) |

|  |
| --- |
| **Theory:** |
| 1. **Modifiers in Java:** 2. In Java, modifiers are keywords that provide additional information about classes, fields, and methods. When declaring methods, modifiers can be used to control the accessibility, behavior, and visibility of the method. Here are some commonly used modifiers when declaring methods, along with explanations and examples:    1. **Access Modifiers:**       * public: The method can be accessed from any class.       * protected: The method can be accessed within the same package and by subclasses even if they are in a different package.       * default (no modifier): If no access modifier is specified, the method can be accessed only within the same package.       * private: The method can only be accessed within the same class.      1. **Non-Access Modifiers:**  * static: The method belongs to the class rather than instances of the class. * final: The method cannot be overridden by subclasses. * abstract: The method has no implementation and must be overridden by concretSSe subclasses. * synchronized: The method can be accessed by only one thread at a time. * native: The method's implementation is provided in a platform-specific manner, usually written in another programming language (like C or C++).      1. Modifiers can be combined as needed when declaring methods. For instance, a method can have both an access modifier and a non-access modifier like static, final, or synchronized.   **Constructors vs Methods:**   1. Constructors and methods in Java serve different purposes, although they share some similarities. Here's a breakdown of the key differences between constructors and methods:    1. **Constructors:**  * Purpose: Constructors are special methods used to initialize objects when they are created. They are automatically called when an instance of a class is instantiated. * Name: The name of a constructor is always the same as the class name. * Return Type: Constructors do not have a return type, not even void. * Modifiers: Constructors can have access modifiers (public, protected, private, or default) to control their visibility. * Overloading: Constructors can be overloaded, meaning a class can have multiple constructors with different parameter lists. * Invocation: Constructors are called implicitly when you create an object using the new keyword.      1. **Methods:**  * Purpose: Methods are functions that define the behavior of a class. They can perform operations, return values, and manipulate object properties. * Name: Methods have unique names within a class. Methods are used to define the behavior of the class, like performing calculations or operations. * Return Type: Methods have a return type (including void if they don't return anything). * Modifiers: Methods can have access modifiers and other non-access modifiers (like static, final, etc.) to control their behavior and visibility. * Overloading: Methods can be overloaded within a class, allowing you to define multiple methods with the same name but different parameter lists. * Invocation: Methods are explicitly called by their name, optionally with arguments, using the instance of the class they belong to.      1. In summary, constructors are special methods used for object initialization and are called automatically when creating an object, while methods are functions that define the behavior of a class and need to be explicitly invoked to execute their code. Both constructors and methods contribute to the overall functionality of a Java class. |

|  |
| --- |
| **Code:** |
| **Write a program to perform addition, subtraction, multiplication and division of two complex numbers.**   * **Variation 1: Implementation with One class only**   import java.util.Scanner;  public class exp2\_q1  {      public static void main(String[] args)      {          Scanner sc = new Scanner(System.in);          System.out.print("\nVARIATION 1 - ONE CLASS ONLY");          System.out.print("\nEnter real part of c1: ");          float c1Real = sc.nextFloat();          System.out.print("Enter imaginary part of c1: ");          float c1Img = sc.nextFloat();          System.out.print("\nEnter real part of c2: ");          float c2Real = sc.nextFloat();          System.out.print("Enter imaginary part of c2: ");          float c2Img = sc.nextFloat();          System.out.println("\nc1 is: ");          System.out.println(c1Real + " + " + c1Img + "i ");          System.out.println("\nc2 is: ");          System.out.println(c2Real + " + " + c2Img + "i ");          float addReal = c1Real + c2Real;          float addImg = c1Img + c2Img;          System.out.println("\naddition: ");          System.out.println(addReal + " + " + addImg + "i ");          float subReal = c1Real - c2Real;          float subImg = c1Img - c2Img;          System.out.println("\nsubtraction: ");          System.out.println(subReal + " + " + subImg + "i ");          float multReal = c1Real \* c2Real - c1Img \* c2Img;          float multImg = c1Real \* c2Img + c1Img \* c2Real;          System.out.println("\nmultiplication: ");          System.out.println(multReal + " + " + multImg + "i ");          float denominator = c2Real \* c2Real + c2Img \* c2Img;          if (denominator != 0)          {              float divReal = (c1Real \* c2Real + c1Img \* c2Img) / denominator;              float divImg = (c1Img \* c2Real - c1Real \* c2Img) / denominator;              System.out.println("\ndivision: ");              System.out.println(divReal + " + " + divImg + "i ");          }          else          {              System.out.println("\ndivision by zero");          }          sc.close();      }  }   * **Variation 2: Implementation with Two classes (One will be main class and Another will be class for complex) without constructors**   // variation 1  import java.util.Scanner;  class calc  {      float real, img;      calc()      {      }      calc(float r, float i)      {          real = r;          img = i;      }      void display()      {          System.out.println(real + " + " + img + "i ");      }      calc add(calc c2)      {          calc res = new calc();          res.real = real + c2.real;          res.img = img + c2.img;          return res;      }      calc sub(calc c2)      {          calc res = new calc();          res.real = real - c2.real;          res.img = img - c2.img;          return res;      }      calc multiply(calc c2)      {          calc res = new calc();          res.real = real \* c2.real - img \* c2.img;          res.img = real \* c2.img + img \* c2.real;          return res;      }      calc divide(calc c2)      {          calc res = new calc();          float denominator = c2.real \* c2.real + c2.img \* c2.img;          res.real = (real \* c2.real + img \* c2.img) / denominator;          res.img = (img \* c2.real - real \* c2.img) / denominator;          return res;      }  }  public class exp2\_q2  {      public static void main(String[] args)      {          Scanner sc = new Scanner(System.in);          System.out.print("\nVARIATION 2 - TWO CLASSES");          System.out.print("\n\nenter real part of c1: ");          float c1Real = sc.nextFloat();          System.out.print("enter imaginary part of c1: ");          float c1Img = sc.nextFloat();          calc c1 = new calc(c1Real, c1Img);          System.out.print("\nenter real part of c2: ");          float c2Real = sc.nextFloat();          System.out.print("enter imaginary part of c2: ");          float c2Img = sc.nextFloat();          calc c2 = new calc(c2Real, c2Img);          System.out.println("\nc1 is: ");          c1.display();          System.out.println("\nc2 is: ");          c2.display();          calc c3 = new calc();          System.out.println("\naddition: ");          c3 = c1.add(c2);          c3.display();          System.out.println("\nsubtraction: ");          c3 = c1.sub(c2);          c3.display();          System.out.println("\nmultiplication: ");          c3 = c1.multiply(c2);          c3.display();          System.out.println("\ndivision: ");          c3 = c1.divide(c2);          c3.display();          sc.close();      }  }   * **Variation 3: variation2 with use of constructors (But without this keyword)**   import java.util.Scanner;  class ComplexCalc  {      float real, img;      ComplexCalc(float r, float i)      {          real = r;          img = i;      }      void display()      {          System.out.println(real + " + " + img + "i ");      }      ComplexCalc add(ComplexCalc c2)      {          return new ComplexCalc(real + c2.real, img + c2.img);      }      ComplexCalc sub(ComplexCalc c2)      {          return new ComplexCalc(real - c2.real, img - c2.img);      }      ComplexCalc multiply(ComplexCalc c2)      {          return new ComplexCalc(real \* c2.real - img \* c2.img, real \* c2.img + img \* c2.real);      }      ComplexCalc divide(ComplexCalc c2)      {          float denominator = c2.real \* c2.real + c2.img \* c2.img;          return new ComplexCalc((real \* c2.real + img \* c2.img) / denominator, (img \* c2.real - real \* c2.img) / denominator);      }  }  public class exp2\_q3  {      public static void main(String[] args)      {          Scanner sc = new Scanner(System.in);          System.out.print("\n\nVARIATION 3 - USE OF CONSTRUCTORS");          System.out.print("\n\nenter real part of c1: ");          float c1Real = sc.nextFloat();          System.out.print("enter imaginary part of c1: ");          float c1Img = sc.nextFloat();          ComplexCalc c1 = new ComplexCalc(c1Real, c1Img);          System.out.print("\nenter real part of c2: ");          float c2Real = sc.nextFloat();          System.out.print("enter imaginary part of c2: ");          float c2Img = sc.nextFloat();          ComplexCalc c2 = new ComplexCalc(c2Real, c2Img);          System.out.println("\nc1 is: ");          c1.display();          System.out.println("\nc2 is: ");          c2.display();          ComplexCalc c3 = new ComplexCalc(0, 0);          System.out.println("\naddition: ");          c3 = c1.add(c2);          c3.display();          System.out.println("\nsubtraction: ");          c3 = c1.sub(c2);          c3.display();          System.out.println("\nmultiplication: ");          c3 = c1.multiply(c2);          c3.display();          System.out.println("\ndivision: ");          c3 = c1.divide(c2);          c3.display();          sc.close();      }  }   * **Variation 4: variation2 with use of constructors (But with use of this keyword)**   import java.util.Scanner;  class ComplexCalc  {      float real, img;      ComplexCalc(float real, float img)      {          this.real = real;          this.img = img;      }      void display()      {          System.out.println(real + " + " + img + "i ");      }      ComplexCalc add(ComplexCalc c2)      {          return new ComplexCalc(this.real + c2.real, this.img + c2.img);      }      ComplexCalc sub(ComplexCalc c2)      {          return new ComplexCalc(this.real - c2.real, this.img - c2.img);      }      ComplexCalc multiply(ComplexCalc c2)      {          return new ComplexCalc(this.real \* c2.real - this.img \* c2.img, this.real \* c2.img + this.img \* c2.real);      }      ComplexCalc divide(ComplexCalc c2)      {          float denominator = c2.real \* c2.real + c2.img \* c2.img;          if (denominator == 0)          {              System.out.println("Division by zero is not allowed.");              return null;          }          return new ComplexCalc(              (this.real \* c2.real + this.img \* c2.img) / denominator,              (this.img \* c2.real - this.real \* c2.img) / denominator          );      }  }  public class exp2\_q4  {      public static void main(String[] args)      {          Scanner sc = new Scanner(System.in);          System.out.print("\n\nVARIATION 4 - USE OF 'THIS' KEYWORD");          System.out.print("\n\nEnter real part of c1: ");          float c1Real = sc.nextFloat();          System.out.print("Enter imaginary part of c1: ");          float c1Img = sc.nextFloat();          ComplexCalc c1 = new ComplexCalc(c1Real, c1Img);          System.out.print("\nEnter real part of c2: ");          float c2Real = sc.nextFloat();          System.out.print("Enter imaginary part of c2: ");          float c2Img = sc.nextFloat();          ComplexCalc c2 = new ComplexCalc(c2Real, c2Img);          System.out.println("\nc1 is: ");          c1.display();          System.out.println("\nc2 is: ");          c2.display();          System.out.println("\naddition: ");          ComplexCalc c3 = c1.add(c2);          c3.display();          System.out.println("\nsubtraction: ");          c3 = c1.sub(c2);          c3.display();          System.out.println("\nmultiplication: ");          c3 = c1.multiply(c2);          c3.display();          System.out.println("\ndivision: ");          c3 = c1.divide(c2);          c3.display();          sc.close();      }  }   * **Variation 5: Add one more class which performs following mathematical operation on complex number.**   + **Inverse**   + **Complex conjugate**   + **Polar form**   // variation 1  import java.util.Scanner;  class calc  {      float real, img;      calc()      {      }      calc(float r, float i)      {          real = r;          img = i;      }      void display()      {          System.out.println(real + " + " + img + "i ");      }      calc add(calc c2)      {          calc res = new calc();          res.real = real + c2.real;          res.img = img + c2.img;          return res;      }      calc sub(calc c2)      {          calc res = new calc();          res.real = real - c2.real;          res.img = img - c2.img;          return res;      }      calc multiply(calc c2)      {          calc res = new calc();          res.real = real \* c2.real - img \* c2.img;          res.img = real \* c2.img + img \* c2.real;          return res;      }      calc divide(calc c2)      {          calc res = new calc();          float denominator = c2.real \* c2.real + c2.img \* c2.img;          res.real = (real \* c2.real + img \* c2.img) / denominator;          res.img = (img \* c2.real - real \* c2.img) / denominator;          return res;      }  }  class ComplexOperations  {      calc complex;      ComplexOperations(calc c)      {          complex = c;      }      calc getInverse()      {          float denominator = complex.real \* complex.real + complex.img \* complex.img;          return new calc(complex.real / denominator, -complex.img / denominator);      }      calc getConjugate()      {          return new calc(complex.real, -complex.img);      }      void displayPolarForm()      {          double magnitude = Math.sqrt(complex.real \* complex.real + complex.img \* complex.img);          double phase = Math.atan2(complex.img, complex.real);          System.out.println("polar form: " + magnitude + " \* (cos(" + phase + ") + i \* sin(" + phase + "))");      }  }  public class exp2\_q5  {      public static void main(String[] args)      {          Scanner sc = new Scanner(System.in);          System.out.print("\nVARIATION 2 - TWO CLASSES");          System.out.print("\n\nenter real part of c1: ");          float c1Real = sc.nextFloat();          System.out.print("enter imaginary part of c1: ");          float c1Img = sc.nextFloat();          calc c1 = new calc(c1Real, c1Img);          System.out.print("\nenter real part of c2: ");          float c2Real = sc.nextFloat();          System.out.print("enter imaginary part of c2: ");          float c2Img = sc.nextFloat();          calc c2 = new calc(c2Real, c2Img);          System.out.println("\nc1 is: ");          c1.display();          System.out.println("\nc2 is: ");          c2.display();          calc c3 = new calc();          System.out.println("\naddition: ");          c3 = c1.add(c2);          c3.display();          System.out.println("\nsubtraction: ");          c3 = c1.sub(c2);          c3.display();          System.out.println("\nmultiplication: ");          c3 = c1.multiply(c2);          c3.display();          System.out.println("\ndivision: ");          c3 = c1.divide(c2);          c3.display();          ComplexOperations operationsC1 = new ComplexOperations(c1);          ComplexOperations operationsC2 = new ComplexOperations(c2);          System.out.println("\ncomplex c1 operations:");          calc inverseC1 = operationsC1.getInverse();          calc conjugateC1 = operationsC1.getConjugate();          System.out.println("inverse: ");          inverseC1.display();          System.out.println("conjugate: ");          conjugateC1.display();          operationsC1.displayPolarForm();          System.out.println("\ncomplex c2 operations:");          calc inverseC2 = operationsC2.getInverse();          calc conjugateC2 = operationsC2.getConjugate();          System.out.println("inverse: ");          inverseC2.display();          System.out.println("conjugate: ");          conjugateC2.display();          operationsC2.displayPolarForm();          sc.close();      }  }  **NOTE:** Use the concept of parameterized method, non-parameterized method, different constructors (passing all arguments, passing few arguments, passing object and without passing arguments). |

|  |
| --- |
| **Output:** |
| * **Variations 1:**      * **Variations 2:**      * **Variations 3:**      * **Variations 4:**      * **Variations 5:** |

|  |
| --- |
| **Post Lab Subjective/Objective type Questions:** |
| 1. **Understand the following code and rewrite the code as per inform in following question.**   **class SalesTaxCalculator {**  **float amount=100.0f; //instance variable**  **float taxRate=10.2f; //instance variable**  **void calculateTax() {**  **float taxAmt = amount\*taxRate/100;**  **System.out.println(taxAmt); }**  **public static void main (String args[ ]) {**  **SalesTaxCalculator obj1 = new SalesTaxCalculator();**  **SalesTaxCalculator obj2 = new SalesTaxCalculator();**  **System.out.println("Amount in Object 1: "+ obj1.amount);**  **System.out.println("Tax Rate in Object 1: "+ obj1.taxRate);**  **System.out.println("Amount in Object 2: "+ obj2.amount);**  **System.out.println("Tax Rate in Object 2: "+ obj2.taxRate);**  **}}**   1. **Modify to accept instance variable using a constructor with no argument and execute it.**   class SalesTaxCalculator  {      float amount; // instance variable      float taxRate; // instance variable      SalesTaxCalculator()      {          amount = 100.0f;          taxRate = 10.2f;      }      void calculateTax()      {          float taxAmt = amount \* taxRate / 100;          System.out.println("Tax Amount: " + taxAmt);      }      public static void main(String args[])        {          SalesTaxCalculator obj1 = new SalesTaxCalculator();          SalesTaxCalculator obj2 = new SalesTaxCalculator();          System.out.println("\n\nAmount in Object 1: " + obj1.amount);          System.out.println("Tax Rate in Object 1: " + obj1.taxRate);          System.out.println("Amount in Object 2: " + obj2.amount);          System.out.println("Tax Rate in Object 2: " + obj2.taxRate);          obj1.calculateTax();          obj2.calculateTax();      }  }     1. **Overload the constructor in part (a) and execute it.**   class SalesTaxCalculator  {      float amount; // instance variable      float taxRate; // instance variable      SalesTaxCalculator()      {          amount = 100.0f;          taxRate = 10.2f;      }      SalesTaxCalculator(float amt, float rate)      {          amount = amt;          taxRate = rate;      }      void calculateTax()      {          float taxAmt = amount \* taxRate / 100;          System.out.println("Tax Amount: " + taxAmt);      }      public static void main(String args[])      {          SalesTaxCalculator obj1 = new SalesTaxCalculator();          SalesTaxCalculator obj2 = new SalesTaxCalculator(150.0f, 8.5f);          System.out.println("\n\nAmount in Object 1: " + obj1.amount);          System.out.println("Tax Rate in Object 1: " + obj1.taxRate);          System.out.println("Amount in Object 2: " + obj2.amount);          System.out.println("Tax Rate in Object 2: " + obj2.taxRate);          obj1.calculateTax();          obj2.calculateTax();      }  }     1. **Make the use of *this* keyword in part (b) and shows its usages.**   class SalesTaxCalculator  {      float amount; // instance variable      float taxRate; // instance variable      SalesTaxCalculator()      {          this(100.0f, 10.2f);      }      SalesTaxCalculator(float amount, float taxRate)      {          this.amount = amount;          this.taxRate = taxRate;      }      void calculateTax()      {          float taxAmt = amount \* taxRate / 100;          System.out.println("Tax Amount: " + taxAmt);      }      public static void main(String args[])      {          SalesTaxCalculator obj1 = new SalesTaxCalculator();          SalesTaxCalculator obj2 = new SalesTaxCalculator(150.0f, 8.5f);          System.out.println("\n\nAmount in Object 1: " + obj1.amount);          System.out.println("Tax Rate in Object 1: " + obj1.taxRate);          System.out.println("Amount in Object 2: " + obj2.amount);          System.out.println("Tax Rate in Object 2: " + obj2.taxRate);          obj1.calculateTax();          obj2.calculateTax();      }  }     1. **Explain the difference between instance variables and class variables.**   Instance variables and class variables are both types of variables in Java, but they serve different purposes and have distinct characteristics.  Instance Variables:   * Also known as instance fields or member variables. * Every instance of a class (object) has its own separate copy of instance variables. * They are declared within a class but outside of any method or constructor. * Instance variables represent the attributes or properties of an object. * They can have different values for different objects of the same class. * Their values are specific to each object and can be accessed and modified using object references. * Instance variables are initialized with default values if not explicitly initialized.   class Student {  String name; // Instance variable  int age; // Instance variable  }  Class Variables (Static Variables):   * Also known as static fields. * There is only one copy of a class variable shared among all instances of the class. * They are declared using the static keyword within a class, typically at the class level. * Class variables represent properties or attributes that are shared by all objects of the class. * They are usually used to store constants or values that are common to all instances. * Class variables can be accessed using the class name itself, without needing an object reference. * Their values are set when the class is loaded and can be modified by all instances, affecting all instances.   class MathUtils {  static final double PI = 3.14159; // Class variable (constant)  static int operationCount = 0; // Class variable (count of operations)  }  In summary, instance variables are specific to each object and represent their individual state, while class variables are shared among all instances and represent common attributes or constants related to the class as a whole.   1. **Explain conversion. How is it different from casting?**   Conversion involves changing the data type of a value from one type to another, often automatically done by the compiler. This is common when compatible types are involved, like converting an int to a double. Casting, on the other hand, is the explicit conversion of a value from one data type to another. Widening casting (implicit) handles conversions to larger types, like int to long, while narrowing casting (explicit) manages conversions to smaller types, like double to int. Casting is used when the compiler can't automatically perform the conversion or when precision concerns arise. |

|  |
| --- |
| **Conclusion:** |
| Understood and implemented of Java's object-oriented concepts, including class structure, constructors, method overloading, instance and class variables, conversions, and casting. |

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