GUJARAT UNIVERSITY
5 Year integrated M. Sc. (Computer Science)
Semester: V
Java Programming
Practical Assignment – 3

Q-1 : Write a Java program using the Object-Oriented Programming (OOP) concept to find the Greatest Common Divisor (GCD) of two positive integers using recursion.
The program should:

1. Define a class GCDExample that contains:
    - Two instance variables to store the numbers.
    - A constructor to initialize these numbers.
    - A public method calculateGCD() that calls a private recursive method gcdRecursive(int a, int b) implementing Euclid's Algorithm.
2. In the main() method:
    - Create an object of GCDExample by passing two positive integers.
    - Display the GCD of the two numbers.

Q-2 : Write a Java program using the Object-Oriented Programming (OOP) approach to compute the sum of the following series:

$S=1/1+1/2+1/3+\cdots+1/n$

$S1 = 1/1 + 1/2 + 1/2^2 + \ldots.. 1/2^n$

where n is a positive integer entered by the user.

1. Create a class Rational to represent a rational number (fraction) with:
    - Two instance variables: numerator and denominator.
    - A constructor to initialize the rational number and simplify it.
    - A method add(Rational other) to add two rational numbers and return the result as a new Rational object.
    - A toString() method to return the rational number in numerator/denominator format.
2. In the main class Summation Series:
    - Prompt the user to enter the value of n.
    - Compute the sum of the series using Rational objects without converting fractions to decimals.
    - Display the sum as a simplified fraction.

3. Bank Account Management System

Write a Java program using inheritance and polymorphism to model a banking system:

- Create an abstract class BankAccount with attributes accountNumber, accountHolderName, and balance.
- Include abstract methods deposit(double amount) and withdraw(double amount).
- Create subclasses SavingsAccount and CurrentAccount with specific withdrawal rules.
- In the main method, store multiple accounts in an array and perform deposits/withdrawals using dynamic method dispatch.

4. Library Management with Interfaces

Design a system for managing books in a library:

- Create an interface LibraryOperations with methods addBook(Book b), removeBook(int bookId), and searchBook(String title).
- Implement this interface in a Library class that maintains an ArrayList<Book>.
- Use encapsulation to keep book details private.
- Demonstrate adding, removing, and searching books in the main method.

5. Employee Payroll Calculation

Using inheritance and method overriding:

- Create a base class Employee with fields name, id, and salary.
- Create subclasses FullTimeEmployee and PartTimeEmployee with different salary calculation methods.
- Store multiple employees in an ArrayList and use polymorphism to calculate salaries.

6. Student Result Processing with Abstract Class

- Create an abstract class Student with fields rollNumber, name and abstract method calculateResult().
- Create a subclass UndergraduateStudent and PostgraduateStudent with different pass percentage criteria.
- Take input for multiple students and display pass/fail results using runtime polymorphism.

7. Online Shopping Cart with Generics

- Create a generic class Cart<T> that stores items of type T.
- Add methods addItem(T item), removeItem(T item), and displayItems().
- Create a Product class with name and price.
- Use the cart with Product objects and display the total cost.

8. Matrix Operations using OOP
   - Create a class Matrix with a 2D array and methods for addition, subtraction, multiplication of matrices.
   - Implement method overloading for scalar multiplication.
   - Overload the toString() method to display matrices neatly.
   - Write a main program to demonstrate all operations.
9. Shape Area & Perimeter Calculation
   - Create an abstract class Shape with abstract methods calculateArea() and calculatePerimeter().
   - Create subclasses Circle, Rectangle, and Triangle.
   - Use method overriding to implement each formula.
   - Store multiple shapes in an array and calculate areas/perimeters using a single loop (polymorphism).
10. Polynomial Operations
   - Create a Polynomial class that represents a polynomial equation (e.g., 3x2+2x+53x^2 + 2x + 5).
   - Use an ArrayList to store coefficients.
   - Implement methods to add, subtract, and multiply two polynomials.
   - Override toString() to display them in algebraic form.
   - In main(), take two polynomials and perform all operations.
11. University Course Enrollment
   - Create classes Course and Student.
   - A student can enroll in multiple courses, and a course can have multiple students (many-to-many relationship).
   - Use composition with ArrayList to store enrollments.
   - Implement methods to add/remove students from a course and display enrolled students.
12. Vehicle Rental System

    Design a Java program to manage a vehicle rental system using OOP principles.

    1. Create an abstract class Vehicle with:
       - Fields: registrationNumber, brand, dailyRentalRate.
       - Constructor to initialize these fields.
       - Abstract method calculateRentalCost(int days).
       - toString() method to display vehicle details.
    2. Create subclasses:
       - Car (extra field: seatingCapacity).
       - Motorbike (extra field: engineCapacity).
       - Truck (extra field: loadCapacity).
    3. Each subclass overrides calculateRentalCost() to apply different rules:
       - Car: base cost = days × dailyRentalRate. If days > 7, apply 10% discount.
       - Motorbike: base cost = days × dailyRentalRate. If engineCapacity > 500cc, add ₹200/day.
       - Truck: base cost = days × dailyRentalRate. If loadCapacity > 5000kg, add 15% surcharge.
    4. In the main() method:
       - Store multiple vehicles in an ArrayList<Vehicle>.
       - Let the user input days for each rental.
       - Display the total rental cost for each vehicle using runtime polymorphism.

Example Output:
Vehicle: Car, Reg No: GJ05AB1234, Brand: Toyota, Days: 10, Cost: ₹13500.0

13. Inventory System with Method Overloading
   - Create an Inventory class to store products.
   - Implement overloaded methods addProduct() that can:
     - Add by name and price.
     - Add by name, price, and quantity.
     - Add by a Product object.
   - Ensure no duplicate products (use equals() and hashCode() in Product).
14. Complex Number Calculator
   - Create a ComplexNumber class with real and imaginary parts.
   - Implement methods for addition, subtraction, multiplication, and division of complex numbers.
   - Override toString() for proper display (e.g., 3 + 4i).
   - Demonstrate in main() with multiple operations.
15. Bank Loan Eligibility System
   - Create an abstract class Loan with method isEligible().
   - Create subclasses HomeLoan, CarLoan, and EducationLoan with different eligibility rules.
   - Take customer details and display whether they are eligible for a loan type.

- Use runtime polymorphism for the check.

16. Online Exam Grading System
    - Create classes Question and Exam.
    - Store multiple choice questions and answers in ArrayList.
    - Use encapsulation for storing the correct answer.
    - In main(), allow the user to attempt the exam and display their score.
17. Travel Booking Fare Calculator
    - Create an abstract class TravelBooking with method calculateFare().
    - Subclasses: BusBooking, TrainBooking, and FlightBooking.
    - Each has a different fare formula (base fare + tax + discount rules).
    - In main(), create different bookings and calculate fares using polymorphism.
18. E-Commerce Discount System
    - Create an interface Discountable with method getDiscountPrice().
    - Implement it in Electronics, Clothing, and Grocery classes.
    - Each category has different discount rules.
    - Use interface polymorphism to display final prices for different products.
19. Sports Tournament Points Table
    - Create a Team class with fields name, matchesPlayed, wins, losses, points.
    - Use methods to update match results and auto-update points.
    - Store teams in an ArrayList and sort them by points using Comparator.

20. Bank Transaction History (Without Files)
    - Create a Transaction class with fields type (Deposit/Withdraw), amount, and date.
    - Create a BankAccount class that maintains an ArrayList<Transaction>.
    - Implement deposit/withdraw methods that automatically record each transaction in history.
    - Display account details and transaction history in main().