Q1. Create a Java program to model a basic educational system using classes, constructors, and inheritance. Define three classes: Person, Student, and Teacher. The Student and Teacher classes should inherit from the Person class. Implement constructors in each class to initialize their attributes.
The Person class should have the following attributes and methods:
name (String)
age (int)
gender (String)
displayDetails(): Display the name, age, and gender of the person.
The Student class should be a subclass of Person and have an additional attribute and method:
studentID (int)
study(): Display a message indicating that the student is studying.
The Teacher class should also be a subclass of Person and have an additional attribute and method:
teacherID (int)
teach(): Display a message indicating that the teacher is teaching.
--Write a Java program that creates instances of Student and Teacher, initializes their attributes, and demonstrates their methods.

Q2. Design a scenario related to a university with students, courses, and classrooms.

Implement the "Has-a" relationship by designing classes for the following entities:

Student: Represents a student with attributes like studentID, name, and coursesEnrolled.
Course: Represents a course with attributes like courseCode, courseName, and teacher.
Teacher: Represents a teacher with attributes like teacherID and name.
-Demonstrate "Association" by creating relationships between these classes. A student can be associated with multiple courses. A course can have one teacher.

Show "Aggregation" by allowing a course to contain multiple students. In this scenario, students can be added and removed from courses.

Illustrate "Composition" by allowing a university to contain classrooms. A classroom can have students. When a classroom is destroyed, the students are removed from it.

Write a Java program that creates instances of these classes, establishes relationships, and performs operations like enrolling students in courses and destroying classrooms.

Q3. Design a Java program that demonstrates the concept of abstraction by creating an abstract class **Employee** and implementing it in two concrete subclasses: **Manager** and **Technician**.

Create an abstract class Employee with the following abstract methods:

calculateSalary(): To calculate the salary of the employee.
displayDetails(): To display the details of the employee.
Create concrete subclasses **Manager** and **Technician** that extend the Employee class and implement the abstract methods.

In the Main class, create instances of both **Manager** and **Technician**, calculate and display their respective salaries and details.

Q4. Create an interface for a **simple messaging system** and implement it in classes representing different messaging services.

a. Define the **MessagingService interface** with methods for sending and receiving messages.
b. Create three classes (**EmailService, SMSService**, and **ChatService**) that implement the **MessagingService interface**. Each class provides its own implementation of the send and receives methods.
c. In the Main class, create instances of the above classes, representing different messaging services. Demonstrate how to send and receive messages using these services.

Q5. Create a base **class Shape** and two derived classes, **Circle** and **Rectangle**.

a. Perform **up-casting** by assigning instances of **Circle** and **Rectangle** to reference variables of type **Shape**. This allows calling the **display method**, which is overridden in the derived classes, but we can't access the specific methods of the derived classes (e.g., **calculateArea**) without down-casting.

b. Demonstrate **down-casting** by checking the type of the object using the instance of operator and then explicitly casting to the appropriate derived class. This allows us to access the specific methods of the derived classes (calculateArea).

Q6. ‘ShoppingCart" application:

1. The **ShoppingCart class** contains a **non-static inner class Item**, which represents items in the shopping cart. It also contains a **static inner class Discount**, which provides a method to apply discounts.
2. The **applyCoupon method** of the **ShoppingCart class** demonstrates an anonymous inner class that implements a **functional interface DiscountCalculator** to calculate discounts.
3. In the Main class, create an instance of the **ShoppingCart** and use the inner classes and anonymous inner class to perform various shopping cart operations, including item display and discount calculations.