Q1. Create multilevel inheritance for the following classes:

- Vehicle
- Four_wheeler
- Petrol_Four_Wheeler
- FiveSeater_Petrol_Four_Wheeler
- Baleno_FiveSeater_Petrol_Four_Wheeler

```
// Base class
class Vehicle {
   String type = "Generic Vehicle";

   Vehicle() {
      System.out.println("Vehicle constructor called");
   }

   void displayType() {
      System.out.println("Type: " + type);
   }
}

// First level
class Four_wheeler extends Vehicle {
   String wheels = "4 wheels";

   Four_wheeler() {
      super(); // Calls Vehicle constructor
      System.out.println("Four_wheeler constructor called");
   }

   void displayWheels() {
```

```
        super.displayType(); // Calls parent class method
        System.out.println("Has: " + wheels);
    }
}

// Second level
class Petrol_Four_Wheeler extends Four_wheeler {
    String fuel = "Petrol";

    Petrol_Four_Wheeler() {
        super(); // Calls Four_wheeler constructor
        System.out.println("Petrol_Four_Wheeler constructor called");
    }

    void displayFuel() {
        System.out.println("Fuel type: " + fuel);
    }
}

// Third level
class FiveSeater_Petrol_Four_Wheeler extends Petrol_Four_Wheeler {
    int seats = 5;

    FiveSeater_Petrol_Four_Wheeler() {
        super(); // Calls Petrol_Four_Wheeler constructor
        System.out.println("FiveSeater_Petrol_Four_Wheeler constructor called");
    }

    void displaySeats() {
```

```
        System.out.println("Seats: " + seats);
    }
}

// Fourth level
class Baleno_FiveSeater_Petrol_Four_Wheeler extends FiveSeater_Petrol_Four_Wheeler {
    String model = "Baleno";

    Baleno_FiveSeater_Petrol_Four_Wheeler() {
        super(); // Calls FiveSeater_Petrol_Four_Wheeler constructor
        System.out.println("Baleno_FiveSeater_Petrol_Four_Wheeler constructor called");
    }

    void displayDetails() {
        // Using super to call parent methods and variables
        super.displayWheels();
        super.displayFuel();
        super.displaySeats();

        // Access parent variable
        System.out.println("Vehicle type from parent: " + super.type);

        // Child-specific info
        System.out.println("Model: " + model);
    }
}

// Main class
public class MultiLevelInheritanceDemo {
```

```
    public static void main(String[] args) {
        Baleno_FiveSeater_Petrol_Four_Wheeler car = new Baleno_FiveSeater_Petrol_Four_Wheeler();
        car.displayDetails();
    }
}
```

Q3. Create a Hospital superclass and access this class inside the Patient child class, accessing properties from Hospital class.

```
// Superclass
class Hospital {
    String hospitalName = "City Care Hospital";
    String location = "Bangalore";
    int totalBeds = 200;

    void displayHospitalInfo() {
        System.out.println("Hospital Name: " + hospitalName);
        System.out.println("Location: " + location);
        System.out.println("Total Beds: " + totalBeds);
    }
}

// Child class
class Patient extends Hospital {
    String patientName;
    int age;
    String disease;

    // Constructor
    Patient(String patientName, int age, String disease) {
```

```
        this.patientName = patientName;
        this.age = age;
        this.disease = disease;
    }

    void displayPatientInfo() {
        // Accessing Hospital properties directly
        System.out.println("Hospital Name from parent: " + hospitalName);
        System.out.println("Hospital Location from parent: " + location);

        // Displaying Patient details
        System.out.println("Patient Name: " + patientName);
        System.out.println("Age: " + age);
        System.out.println("Disease: " + disease);

        // Calling Hospital's method
        System.out.println("\n--- Full Hospital Details ---");
        super.displayHospitalInfo();
    }
}

// Main class
public class HospitalDemo {
    public static void main(String[] args) {
        Patient p = new Patient("John Doe", 45, "Pneumonia");
        p.displayPatientInfo();
    }
}
```

4. Create Hierarchical inheritance

```
// Parent class
class After_12th {
   void courses() {
      System.out.println("Available streams after 12th.");
   }
}

// Engineering branch
class Engineering extends After_12th {
   void engCourses() {
      System.out.println("Engineering courses: IT, Mechanical, CS.");
   }
}

// IT specialization
class IT extends Engineering {
   void showIT() {
      System.out.println("Information Technology branch.");
   }
}

// Mechanical specialization
class Mechanical extends Engineering {
   void showMech() {
      System.out.println("Mechanical Engineering branch.");
   }
}
```

```
// CS specialization
class CS extends Engineering {
   void showCS() {
      System.out.println("Computer Science branch.");
   }
}

// Medical branch
class Medical extends After_12th {
   void medCourses() {
      System.out.println("Medical courses: MBBS, BDS.");
   }
}

// MBBS specialization
class MBBS extends Medical {
   void showMBBS() {
      System.out.println("MBBS - Bachelor of Medicine and Surgery.");
   }
}

// BDS specialization
class BDS extends Medical {
   void showBDS() {
      System.out.println("BDS - Bachelor of Dental Surgery.");
   }
}

// Other courses branch
```

```
class Other_Courses extends After_12th {
   void otherCourses() {
      System.out.println("Other courses: BCA, BBA.");
   }
}

// BCA specialization
class BCA extends Other_Courses {
   void showBCA() {
      System.out.println("BCA - Bachelor of Computer Applications.");
   }
}

// BBA specialization
class BBA extends Other_Courses {
   void showBBA() {
      System.out.println("BBA - Bachelor of Business Administration.");
   }
}

// Testing class
public class HierarchicalInheritanceDemo {
   public static void main(String[] args) {
      IT it = new IT();
      it.courses();
      it.engCourses();
      it.showIT();

      MBBS mbbs = new MBBS();
```

```
        mbbs.courses();
        mbbs.medCourses();
        mbbs.showMBBS();

        BCA bca = new BCA();
        bca.courses();
        bca.otherCourses();
        bca.showBCA();
    }
}
```

5. Create practice on this

1. Create a class Calculator with the following overloaded add() 1.add(int a, int b) 2.add(int a, int b, int c) 3.add(double a, double b)

```
class Calculator {
    // 1. Add two integers
    int add(int a, int b) {
        return a + b;
    }

    // 2. Add three integers
    int add(int a, int b, int c) {
        return a + b + c;
    }

    // 3. Add two doubles
    double add(double a, double b) {
        return a + b;
    }
```

```
    public static void main(String[] args) {
        Calculator calc = new Calculator();

        System.out.println("Sum of 2 integers: " + calc.add(5, 10));
        System.out.println("Sum of 3 integers: " + calc.add(5, 10, 15));
        System.out.println("Sum of 2 doubles: " + calc.add(5.5, 10.5));
    }
}
```

2. Create a base class Shape with a method area() that prints a message. Then create two subclasses Circleàoverride area() to calculator and print area of circle Rectangleà override area() to calculate and print area of a rectangle

```
// Base class
class Shape {
    void area() {
        System.out.println("This is the area method of Shape.");
    }
}

// Subclass for Circle
class Circle extends Shape {
    double radius;

    Circle(double radius) {
        this.radius = radius;
    }

    @Override
    void area() {
        double result = Math.PI * radius * radius;
```

```
        System.out.println("Area of Circle: " + result);
    }
}

// Subclass for Rectangle
class Rectangle extends Shape {
    double length, width;

    Rectangle(double length, double width) {
        this.length = length;
        this.width = width;
    }

    @Override
    void area() {
        double result = length * width;
        System.out.println("Area of Rectangle: " + result);
    }
}

// Main class to test
public class ShapeTest {
    public static void main(String[] args) {
        Shape shape1 = new Circle(5);     // Runtime polymorphism
        Shape shape2 = new Rectangle(4, 6);

        shape1.area();  // Calls Circle's overridden area()
        shape2.area();  // Calls Rectangle's overridden area()
    }
```

```
}
```

3. Create a Bank class with a method getInterestRate() create subclasses: SBIàreturn 6.7% ICICIàreturn 7.0% HDFCàreturn 7.5%

```
// Base class
class Bank {
  double getInterestRate() {
    return 0.0; // Default rate
  }
}

// SBI subclass
class SBI extends Bank {
  @Override
  double getInterestRate() {
    return 6.7;
  }
}

// ICICI subclass
class ICICI extends Bank {
  @Override
  double getInterestRate() {
    return 7.0;
  }
}

// HDFC subclass
class HDFC extends Bank {
  @Override
```

```
    double getInterestRate() {
        return 7.5;
    }
}

// Testing class
public class BankTest {
    public static void main(String[] args) {
        Bank b1 = new SBI();
        Bank b2 = new ICICI();
        Bank b3 = new HDFC();

        System.out.println("SBI Interest Rate: " + b1.getInterestRate() + "%");
        System.out.println("ICICI Interest Rate: " + b2.getInterestRate() + "%");
        System.out.println("HDFC Interest Rate: " + b3.getInterestRate() + "%");
    }
}
```

4. Runtime Polymorphism with constructor Chaining create a class vehicle with a constructor that prints “Vehicle Created”

Create a subclass Bike that override a method and uses super() in constructor

Combined question

Create an abstract class SmartDevice with methods like turnOn(), turnOff(), and performFunction(). Create child classes:

· SmartPhone: performs calling and browsing.

· SmartWatch: tracks fitness and time.

· SmartSpeaker: plays music and responds to voice commands.

.

· Write code to store all objects in an array and use polymorphism to invoke their performFunction().

Ans ;- Part 1 – Runtime Polymorphism with Constructor Chaining

```
class Vehicle {
  Vehicle() {
    System.out.println("Vehicle Created");
  }

  void run() {
    System.out.println("Vehicle is running");
  }
}

class Bike extends Vehicle {
  Bike() {
    super(); // Calls Vehicle's constructor
    System.out.println("Bike Created");
  }

  @Override
  void run() {
    System.out.println("Bike is running safely");
  }
}

public class VehicleTest {
  public static void main(String[] args) {
    Vehicle v = new Bike(); // Runtime polymorphism
    v.run();
  }
}
```

output

Vehicle Created

Bike Created

Bike is running safely

Part 2 – Abstract Class & Polymorphism

```
// Abstract base class
abstract class SmartDevice {
    abstract void turnOn();
    abstract void turnOff();
    abstract void performFunction();
}

// SmartPhone class
class SmartPhone extends SmartDevice {
    @Override
    void turnOn() {
        System.out.println("SmartPhone is turned ON");
    }

    @Override
    void turnOff() {
        System.out.println("SmartPhone is turned OFF");
    }

    @Override
    void performFunction() {
        System.out.println("SmartPhone: Making calls and browsing the internet");
    }
}
```

```
// SmartWatch class
class SmartWatch extends SmartDevice {
    @Override
    void turnOn() {
        System.out.println("SmartWatch is turned ON");
    }

    @Override
    void turnOff() {
        System.out.println("SmartWatch is turned OFF");
    }

    @Override
    void performFunction() {
        System.out.println("SmartWatch: Tracking fitness and showing time");
    }
}

// SmartSpeaker class
class SmartSpeaker extends SmartDevice {
    @Override
    void turnOn() {
        System.out.println("SmartSpeaker is turned ON");
    }

    @Override
    void turnOff() {
        System.out.println("SmartSpeaker is turned OFF");
```

```
    }

    @Override
    void performFunction() {
        System.out.println("SmartSpeaker: Playing music and responding to voice commands");
    }
}

// Testing
public class SmartDeviceTest {
    public static void main(String[] args) {
        SmartDevice[] devices = {
            new SmartPhone(),
            new SmartWatch(),
            new SmartSpeaker()
        };

        // Using polymorphism
        for (SmartDevice device : devices) {
            device.turnOn();
            device.performFunction();
            device.turnOff();
            System.out.println();
        }
    }
}
```

Q2.Design an interface Bank with methods deposit(), withdraw(), and getBalance(). Implement this in SavingsAccount and CurrentAccount classes.

· Use inheritance to create a base Account class.

· Demonstrate method overriding with customized logic for withdrawal (e.g., minimum balance in SavingsAccount).

```
// Step 1: Interface Bank
interface Bank {
   void deposit(double amount);
   void withdraw(double amount);
   double getBalance();
}

// Step 2: Base class Account
abstract class Account implements Bank {
   protected double balance;

   Account(double initialBalance) {
      this.balance = initialBalance;
   }

   @Override
   public void deposit(double amount) {
      if (amount > 0) {
         balance += amount;
         System.out.println("Deposited: " + amount);
      } else {
         System.out.println("Invalid deposit amount!");
      }
   }

   @Override
```

```
    public double getBalance() {
        return balance;
    }
}

// Step 3: SavingsAccount with minimum balance logic
class SavingsAccount extends Account {
    private static final double MIN_BALANCE = 500.0;

    SavingsAccount(double initialBalance) {
        super(initialBalance);
    }

    @Override
    public void withdraw(double amount) {
        if (balance - amount >= MIN_BALANCE) {
            balance -= amount;
            System.out.println("Withdrawn from SavingsAccount: " + amount);
        } else {
            System.out.println("Cannot withdraw! Minimum balance of Rs." + MIN_BALANCE + " must be
maintained.");
        }
    }
}

// Step 4: CurrentAccount with no minimum balance restriction
class CurrentAccount extends Account {
    CurrentAccount(double initialBalance) {
        super(initialBalance);
```

```
    }

    @Override
    public void withdraw(double amount) {
        if (amount <= balance) {
            balance -= amount;
            System.out.println("Withdrawn from CurrentAccount: " + amount);
        } else {
            System.out.println("Insufficient funds!");
        }
    }
}

// Step 5: Test the program
public class BankTest {
    public static void main(String[] args) {
        Bank savings = new SavingsAccount(2000);
        Bank current = new CurrentAccount(5000);

        System.out.println("\n--- Savings Account ---");
        savings.deposit(1000);
        savings.withdraw(2300); // Allowed
        savings.withdraw(2000); // Not allowed
        System.out.println("Final Savings Balance: " + savings.getBalance());

        System.out.println("\n--- Current Account ---");
        current.deposit(2000);
        current.withdraw(6000); // Allowed
        current.withdraw(2000); // Not allowed if insufficient
```

```
        System.out.println("Final Current Balance: " + current.getBalance());
    }
}
```

3

Create a base class Vehicle with method start(). Derive Car, Bike, and Truck from it and override the start() method.

· Create a static method that accepts Vehicle type and calls start().

· Pass different vehicle objects to test polymorphism.

```
// Base class
class Vehicle {
    void start() {
        System.out.println("Vehicle is starting...");
    }
}

// Subclass Car
class Car extends Vehicle {
    @Override
    void start() {
        System.out.println("Car is starting with key ignition.");
    }
}

// Subclass Bike
class Bike extends Vehicle {
    @Override
    void start() {
        System.out.println("Bike is starting with self start.");
    }
```

```
}

// Subclass Truck
class Truck extends Vehicle {
  @Override
  void start() {
    System.out.println("Truck is starting with heavy engine sound.");
  }
}

public class VehicleTest {

  // Static method accepting Vehicle type
  static void startVehicle(Vehicle v) {
    v.start(); // Polymorphic call
  }

  public static void main(String[] args) {
    Vehicle car = new Car();
    Vehicle bike = new Bike();
    Vehicle truck = new Truck();

    startVehicle(car);
    startVehicle(bike);
    startVehicle(truck);
  }
}
```

4.

Design an abstract class Person with fields like name, age, and abstract method getRoleInfo(). Create subclasses:

· Student: has course and roll number.

· Professor: has subject and salary.

· TeachingAssistant: extends Student and implements getRoleInfo() in a hybrid way.

· Create and print info for all roles using overridden getRoleInfo().

```
// Abstract class
abstract class Person {
    String name;
    int age;

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

    // Abstract method
    abstract void getRoleInfo();
}

// Student subclass
class Student extends Person {
    String course;
    int rollNumber;

    Student(String name, int age, String course, int rollNumber) {
        super(name, age);
        this.course = course;
        this.rollNumber = rollNumber;
```

```
    }

    @Override
    void getRoleInfo() {
        System.out.println("Student Name: " + name);
        System.out.println("Age: " + age);
        System.out.println("Course: " + course);
        System.out.println("Roll Number: " + rollNumber);
    }
}

// Professor subclass
class Professor extends Person {
    String subject;
    double salary;

    Professor(String name, int age, String subject, double salary) {
        super(name, age);
        this.subject = subject;
        this.salary = salary;
    }

    @Override
    void getRoleInfo() {
        System.out.println("Professor Name: " + name);
        System.out.println("Age: " + age);
        System.out.println("Subject: " + subject);
        System.out.println("Salary: " + salary);
    }
```

```
}

// TeachingAssistant subclass (Hybrid: Extends Student)
class TeachingAssistant extends Student {
  String subject;
  double stipend;

  TeachingAssistant(String name, int age, String course, int rollNumber, String subject, double stipend) {
    super(name, age, course, rollNumber);
    this.subject = subject;
    this.stipend = stipend;
  }

  @Override
  void getRoleInfo() {
    System.out.println("Teaching Assistant Name: " + name);
    System.out.println("Age: " + age);
    System.out.println("Course: " + course);
    System.out.println("Roll Number: " + rollNumber);
    System.out.println("Subject Assisted: " + subject);
    System.out.println("Stipend: " + stipend);
  }
}

// Main class to test
public class PersonTest {
  public static void main(String[] args) {
    Person student = new Student("Amit", 20, "Computer Science", 101);
```

```
        Person professor = new Professor("Dr. Sharma", 45, "Mathematics", 80000);
        Person ta = new TeachingAssistant("Rohit", 22, "Information Technology", 202, "Java Programming", 15000);

        // Using overridden getRoleInfo()
        student.getRoleInfo();
        System.out.println();
        professor.getRoleInfo();
        System.out.println();
        ta.getRoleInfo();
    }
}
```

5.Create:

· Interface Drawable with method draw()

· Abstract class Shape with abstract method area() Subclasses: Circle, Rectangle, and Triangle.

· Calculate area using appropriate formulas.

· Demonstrate how interface and abstract class work together.

```
// Step 1: Interface
interface Drawable {
    void draw();
}

// Step 2: Abstract Class
abstract class Shape implements Drawable {
    abstract double area(); // Abstract method for calculating area
}

// Step 3: Circle class
class Circle extends Shape {
```

```
    double radius;

    Circle(double radius) {
        this.radius = radius;
    }

    @Override
    public void draw() {
        System.out.println("Drawing a Circle...");
    }

    @Override
    double area() {
        return Math.PI * radius * radius;
    }
}

// Step 4: Rectangle class
class Rectangle extends Shape {
    double length, width;

    Rectangle(double length, double width) {
        this.length = length;
        this.width = width;
    }

    @Override
    public void draw() {
        System.out.println("Drawing a Rectangle...");
```

```
    }

    @Override
    double area() {
        return length * width;
    }
}

// Step 5: Triangle class
class Triangle extends Shape {
    double base, height;

    Triangle(double base, double height) {
        this.base = base;
        this.height = height;
    }

    @Override
    public void draw() {
        System.out.println("Drawing a Triangle...");
    }

    @Override
    double area() {
        return 0.5 * base * height;
    }
}

// Step 6: Testing
```

```
public class ShapeTest {
    public static void main(String[] args) {
        Shape[] shapes = {
            new Circle(5),
            new Rectangle(4, 6),
            new Triangle(3, 8)
        };

        for (Shape s : shapes) {
            s.draw();
            System.out.println("Area: " + s.area());
            System.out.println();
        }
    }
}
```