1. Inheritance and Polymorphism: Vehicle Hierarchy
java

```
// Base class Vehicle
class Vehicle {
    protected String make;
    protected String model;
    protected int year;
    protected String color;

    public Vehicle(String make, String model, int year, String color) {
        this.make = make;
        this.model = model;
        this.year = year;
        this.color = color;
    }

    // Method to be overridden by derived classes
    public void startEngine() {
        System.out.println("Starting vehicle engine...");
    }

    public void stopEngine() {
        System.out.println("Stopping vehicle engine...");
    }

    public void displayInfo() {
        System.out.printf("%s %s %d (%s)%n", make, model, year, color);
    }

    // Getters
    public String getMake() { return make; }
    public String getModel() { return model; }
    public int getYear() { return year; }
    public String getColor() { return color; }
}

// Derived class Car
class Car extends Vehicle {
    private int doors;
    private String transmission;
    private boolean isTurbo;

    public Car(String make, String model, int year, String color,
               int doors, String transmission, boolean isTurbo) {
        super(make, model, year, color);
        this.doors = doors;
        this.transmission = transmission;
        this.isTurbo = isTurbo;
    }

    @Override
    public void startEngine() {
        System.out.println("ðŸ”‘ Inserting key and turning ignition...");
        System.out.println("ðŸš— Car engine roaring to life!");
        if (isTurbo) {
            System.out.println("ðŸ’¨ Turbocharger activated!");
        }
    }

    @Override
    public void displayInfo() {
```

```
        super.displayInfo();
        System.out.printf("Doors: %d, Transmission: %s, Turbo: %s%n",
                          doors, transmission, isTurbo ? "Yes" : "No");
    }

    public void openTrunk() {
        System.out.println("Opening car trunk...");
    }
}

// Derived class Motorcycle
class Motorcycle extends Vehicle {
    private String type; // Sport, Cruiser, Touring, etc.
    private boolean hasFairing;
    private int engineCC;

    public Motorcycle(String make, String model, int year, String color,
                      String type, boolean hasFairing, int engineCC) {
        super(make, model, year, color);
        this.type = type;
        this.hasFairing = hasFairing;
        this.engineCC = engineCC;
    }

    @Override
    public void startEngine() {
        System.out.println("ðŸ□□ï¸□  Kick-starting motorcycle...");
        System.out.println("VROOOOM! Motorcycle engine revving!");
        System.out.println("ðŸ”Š Distinctive motorcycle exhaust sound!");
    }

    @Override
    public void displayInfo() {
        super.displayInfo();
        System.out.printf("Type: %s, Fairing: %s, Engine: %dcc%n",
                          type, hasFairing ? "Yes" : "No", engineCC);
    }

    public void wheelie() {
        System.out.println("ðŸš€ Performing a wheelie!");
    }
}

// Demonstration class
public class InheritancePolymorphismDemo {
    public static void main(String[] args) {
        System.out.println("=== INHERITANCE AND POLYMORPHISM DEMO ===\n");

        // Create instances
        Vehicle car = new Car("Toyota", "Camry", 2023, "Red", 4, "Automatic", false);
        Vehicle motorcycle = new Motorcycle("Harley-Davidson", "Sportster", 2022, "Black",
"Cruiser", false, 1200);

        // Demonstrate polymorphism
        Vehicle[] vehicles = {car, motorcycle};

        for (Vehicle vehicle : vehicles) {
            System.out.println("\n--- Vehicle Information ---");
            vehicle.displayInfo();
            System.out.print("Starting engine: ");
            vehicle.startEngine(); // Polymorphic method call
            vehicle.stopEngine();
```

```
            // Demonstrate specific behaviors using instanceof
            if (vehicle instanceof Car) {
                ((Car) vehicle).openTrunk();
            } else if (vehicle instanceof Motorcycle) {
                ((Motorcycle) vehicle).wheelie();
            }
        }

        // Additional demonstration with more vehicles
        System.out.println("\n" + "=".repeat(40));
        System.out.println("ADDITIONAL VEHICLE DEMONSTRATION:");

        Vehicle sportsCar = new Car("Porsche", "911", 2024, "Yellow", 2, "Manual", true);
        Vehicle touringBike = new Motorcycle("BMW", "R1250GS", 2023, "Blue", "Adventure", true,
1250);

        Vehicle[] moreVehicles = {sportsCar, touringBike};

        for (int i = 0; i < moreVehicles.length; i++) {
            System.out.printf("%nVehicle %d:%n", i + 1);
            moreVehicles[i].startEngine();
        }
    }
}
2. Abstraction and Interfaces: Shape System
java
// Shape interface
interface Shape {
    double calculateArea();
    double calculatePerimeter();
    void displayShapeInfo();
    String getShapeType();
}

// Circle class implementing Shape interface
class Circle implements Shape {
    private double radius;
    private String name;

    public Circle(double radius, String name) {
        if (radius <= 0) {
            throw new IllegalArgumentException("Radius must be positive");
        }
        this.radius = radius;
        this.name = name;
    }

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

    @Override
    public double calculatePerimeter() {
        return 2 * Math.PI * radius;
    }

    @Override
    public void displayShapeInfo() {
        System.out.println("â—‹ Circle: " + name);
        System.out.printf("   Radius: %.2f units%n", radius);
```

```
        System.out.printf("   Diameter: %.2f units%n", radius * 2);
        System.out.printf("   Area: %.2f square units%n", calculateArea());
        System.out.printf("   Circumference: %.2f units%n", calculatePerimeter());
    }

    @Override
    public String getShapeType() {
        return "Circle";
    }

    // Circle-specific methods
    public double getRadius() { return radius; }
    public String getName() { return name; }
}

// Rectangle class implementing Shape interface
class Rectangle implements Shape {
    private double length;
    private double width;
    private String name;

    public Rectangle(double length, double width, String name) {
        if (length <= 0 || width <= 0) {
            throw new IllegalArgumentException("Length and width must be positive");
        }
        this.length = length;
        this.width = width;
        this.name = name;
    }

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

    @Override
    public double calculatePerimeter() {
        return 2 * (length + width);
    }

    @Override
    public void displayShapeInfo() {
        System.out.println("â– Rectangle: " + name);
        System.out.printf("   Length: %.2f units%n", length);
        System.out.printf("   Width: %.2f units%n", width);
        System.out.printf("   Area: %.2f square units%n", calculateArea());
        System.out.printf("   Perimeter: %.2f units%n", calculatePerimeter());
        System.out.printf("   Is Square: %s%n", isSquare() ? "Yes" : "No");
    }

    @Override
    public String getShapeType() {
        return "Rectangle";
    }

    // Rectangle-specific methods
    public boolean isSquare() {
        return length == width;
    }

    public double getLength() { return length; }
    public double getWidth() { return width; }
```

```
    public String getName() { return name; }
}

// Triangle class implementing Shape interface
class Triangle implements Shape {
    private double base;
    private double height;
    private double sideA;
    private double sideB;
    private String name;

    public Triangle(double base, double height, double sideA, double sideB, String name) {
        if (base <= 0 || height <= 0 || sideA <= 0 || sideB <= 0) {
            throw new IllegalArgumentException("All dimensions must be positive");
        }
        this.base = base;
        this.height = height;
        this.sideA = sideA;
        this.sideB = sideB;
        this.name = name;
    }

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

    @Override
    public double calculatePerimeter() {
        return base + sideA + sideB;
    }

    @Override
    public void displayShapeInfo() {
        System.out.println("â–³ Triangle: " + name);
        System.out.printf("   Base: %.2f units%n", base);
        System.out.printf("   Height: %.2f units%n", height);
        System.out.printf("   Side A: %.2f units%n", sideA);
        System.out.printf("   Side B: %.2f units%n", sideB);
        System.out.printf("   Area: %.2f square units%n", calculateArea());
        System.out.printf("   Perimeter: %.2f units%n", calculatePerimeter());
    }

    @Override
    public String getShapeType() {
        return "Triangle";
    }
}

// Shape Calculator utility class
class ShapeCalculator {
    public static void displayAllShapes(Shape[] shapes) {
        System.out.println("\n=== ALL SHAPES SUMMARY ===");
        for (Shape shape : shapes) {
            shape.displayShapeInfo();
            System.out.println(); // Empty line for separation
        }
    }

    public static double calculateTotalArea(Shape[] shapes) {
        double totalArea = 0;
        for (Shape shape : shapes) {
```

```
            totalArea += shape.calculateArea();
        }
        return totalArea;
    }

    public static void displayShapeStatistics(Shape[] shapes) {
        System.out.println("=== SHAPE STATISTICS ===");
        System.out.printf("Total Shapes: %d%n", shapes.length);
        System.out.printf("Total Area: %.2f square units%n", calculateTotalArea(shapes));

        // Count by shape type
        java.util.Map<String, Integer> shapeCount = new java.util.HashMap<>();
        for (Shape shape : shapes) {
            shapeCount.put(shape.getShapeType(),
                           shapeCount.getOrDefault(shape.getShapeType(), 0) + 1);
        }

        System.out.println("Shape Distribution:");
        for (java.util.Map.Entry<String, Integer> entry : shapeCount.entrySet()) {
            System.out.printf("  %s: %d%n", entry.getKey(), entry.getValue());
        }
    }
}

// Demonstration class
public class AbstractionInterfacesDemo {
    public static void main(String[] args) {
        System.out.println("=== ABSTRACTION AND INTERFACES DEMO ===\n");

        // Create shapes using interface reference
        Shape circle = new Circle(5.0, "Large Circle");
        Shape rectangle = new Rectangle(8.0, 6.0, "Office Desk");
        Shape square = new Rectangle(4.0, 4.0, "Square Tile");
        Shape triangle = new Triangle(6.0, 4.0, 5.0, 5.0, "Right Triangle");

        // Array of shapes demonstrating polymorphism
        Shape[] shapes = {circle, rectangle, square, triangle};

        // Display individual shapes
        for (Shape shape : shapes) {
            shape.displayShapeInfo();
            System.out.println("--------------------");
        }

        // Use ShapeCalculator utility
        ShapeCalculator.displayShapeStatistics(shapes);
        ShapeCalculator.displayAllShapes(shapes);

        // Demonstrate interface flexibility
        System.out.println("\n" + "=".repeat(40));
        System.out.println("DYNAMIC SHAPE PROCESSING:");

        processShapesDynamically(shapes);
    }

    private static void processShapesDynamically(Shape[] shapes) {
        for (int i = 0; i < shapes.length; i++) {
            System.out.printf("%nProcessing %s %d:%n", shapes[i].getShapeType(), i + 1);
            System.out.printf("Area: %.2f%n", shapes[i].calculateArea());
            System.out.printf("Perimeter: %.2f%n", shapes[i].calculatePerimeter());
        }
    }
```

```
}
3. Abstract Classes and Polymorphism: Animal System
java
// Abstract base class Animal
abstract class Animal {
    protected String name;
    protected int age;
    protected String habitat;

    public Animal(String name, int age, String habitat) {
        this.name = name;
        this.age = age;
        this.habitat = habitat;
    }

    // Abstract method - must be implemented by subclasses
    public abstract void makeSound();

    // Abstract method for animal movement
    public abstract void move();

    // Concrete method in abstract class
    public void sleep() {
        System.out.println(name + " is sleeping... ðŸ’¤");
    }

    public void eat() {
        System.out.println(name + " is eating... ðŸ½ï¸");
    }

    // Template method pattern
    public final void dailyRoutine() {
        System.out.println("\n--- " + name + "'s Daily Routine ---");
        wakeUp();
        eat();
        move();
        makeSound();
        sleep();
    }

    protected void wakeUp() {
        System.out.println(name + " is waking up... ðŸŒ…");
    }

    // Getters
    public String getName() { return name; }
    public int getAge() { return age; }
    public String getHabitat() { return habitat; }

    @Override
    public String toString() {
        return String.format("%s [Age: %d, Habitat: %s]", name, age, habitat);
    }
}

// Concrete class Dog
class Dog extends Animal {
    private String breed;
    private boolean isTrained;

    public Dog(String name, int age, String habitat, String breed, boolean isTrained) {
        super(name, age, habitat);
```

```
        this.breed = breed;
        this.isTrained = isTrained;
    }

    @Override
    public void makeSound() {
        System.out.println(name + " says: Woof! Woof! ðŸ•");
    }

    @Override
    public void move() {
        System.out.println(name + " is running and wagging tail! ðŸ¾");
    }

    @Override
    protected void wakeUp() {
        System.out.println(name + " the " + breed + " is stretching and waking up...");
    }

    // Dog-specific methods
    public void fetch() {
        System.out.println(name + " is fetching the ball! ðŸŽ¾");
    }

    public void barkAtMailman() {
        System.out.println(name + " is barking at the mailman! ðŸ“¬");
    }

    public String getBreed() { return breed; }
    public boolean isTrained() { return isTrained; }
}

// Concrete class Cat
class Cat extends Animal {
    private String furType;
    private boolean isIndoor;

    public Cat(String name, int age, String habitat, String furType, boolean isIndoor) {
        super(name, age, habitat);
        this.furType = furType;
        this.isIndoor = isIndoor;
    }

    @Override
    public void makeSound() {
        System.out.println(name + " says: Meow! Purrr... ðŸ˜º");
    }

    @Override
    public void move() {
        System.out.println(name + " is gracefully walking and stretching... ðŸ§˜");
    }

    @Override
    protected void wakeUp() {
        System.out.println(name + " is waking up and doing cat yoga... ðŸ§˜");
    }

    // Cat-specific methods
    public void climbTree() {
        System.out.println(name + " is climbing a tree! ðŸŒ³");
    }
```

```
    public void chaseLaser() {
        System.out.println(name + " is chasing the laser pointer! ðŸ”´");
    }

    public String getFurType() { return furType; }
    public boolean isIndoor() { return isIndoor; }
}

// Concrete class Bird
class Bird extends Animal {
    private double wingspan;
    private boolean canFly;

    public Bird(String name, int age, String habitat, double wingspan, boolean canFly) {
        super(name, age, habitat);
        this.wingspan = wingspan;
        this.canFly = canFly;
    }

    @Override
    public void makeSound() {
        System.out.println(name + " says: Tweet! Chirp! ðŸ¦");
    }

    @Override
    public void move() {
        if (canFly) {
            System.out.println(name + " is flying through the air! ðŸ•Šï¸");
        } else {
            System.out.println(name + " is hopping on the ground! ðŸ¦˜");
        }
    }

    // Bird-specific methods
    public void buildNest() {
        System.out.println(name + " is building a nest! ðŸªº");
    }

    public double getWingspan() { return wingspan; }
    public boolean canFly() { return canFly; }
}

// Animal Shelter class to manage animals
class AnimalShelter {
    private java.util.List<Animal> animals;

    public AnimalShelter() {
        this.animals = new java.util.ArrayList<>();
    }

    public void addAnimal(Animal animal) {
        animals.add(animal);
        System.out.println("âœ… Added " + animal.getName() + " to the shelter");
    }

    public void makeAllSounds() {
        System.out.println("\n=== ANIMAL CONCERT ===");
        for (Animal animal : animals) {
            animal.makeSound();
        }
    }
```

```
    public void startDailyRoutines() {
        System.out.println("\n=== DAILY ANIMAL ROUTINES ===");
        for (Animal animal : animals) {
            animal.dailyRoutine();
        }
    }

    public void displayAllAnimals() {
        System.out.println("\n=== ANIMAL SHELTER INVENTORY ===");
        for (int i = 0; i < animals.size(); i++) {
            System.out.printf("%d. %s%n", i + 1, animals.get(i));
        }
    }
}

// Demonstration class
public class AbstractClassesPolymorphismDemo {
    public static void main(String[] args) {
        System.out.println("=== ABSTRACT CLASSES AND POLYMORPHISM DEMO ===\n");

        // Create animals using abstract class reference
        Animal dog = new Dog("Buddy", 3, "House", "Golden Retriever", true);
        Animal cat = new Cat("Whiskers", 2, "Apartment", "Short Hair", true);
        Animal bird = new Bird("Tweety", 1, "Forest", 12.5, true);
        Animal penguin = new Bird("Pingu", 4, "Arctic", 8.0, false);

        // Array of animals demonstrating polymorphism
        Animal[] animals = {dog, cat, bird, penguin};

        // Demonstrate polymorphism
        System.out.println("POLYMORPHIC METHOD CALLS:");
        for (Animal animal : animals) {
            System.out.println("\n--- " + animal.getName() + " ---");
            animal.makeSound(); // Polymorphic call
            animal.move();      // Polymorphic call
        }

        // Animal Shelter demonstration
        System.out.println("\n" + "=".repeat(50));
        AnimalShelter shelter = new AnimalShelter();

        // Add animals to shelter
        for (Animal animal : animals) {
            shelter.addAnimal(animal);
        }

        // Shelter operations
        shelter.displayAllAnimals();
        shelter.makeAllSounds();
        shelter.startDailyRoutines();

        // Demonstrate specific behaviors using instanceof
        System.out.println("\n" + "=".repeat(50));
        System.out.println("SPECIFIC ANIMAL BEHAVIORS:");

        for (Animal animal : animals) {
            if (animal instanceof Dog) {
                ((Dog) animal).fetch();
                ((Dog) animal).barkAtMailman();
            } else if (animal instanceof Cat) {
                ((Cat) animal).chaseLaser();
```

```
                ((Cat) animal).climbTree();
            } else if (animal instanceof Bird) {
                ((Bird) animal).buildNest();
            }
        }
    }
}
```

4. Encapsulation and Interfaces: Logging System

```
java
// Logger interface
interface Logger {
    void logInfo(String message);
    void logError(String message);
    void logWarning(String message);
    void logDebug(String message);
    void setLogLevel(String level); // INFO, ERROR, WARN, DEBUG
}

// FileLogger implementation
class FileLogger implements Logger {
    private String filename;
    private String logLevel;
    private java.io.PrintWriter writer;

    public FileLogger(String filename) {
        this.filename = filename;
        this.logLevel = "INFO"; // Default log level
        initializeWriter();
    }

    private void initializeWriter() {
        try {
            this.writer = new java.io.PrintWriter(new java.io.FileWriter(filename, true));
        } catch (java.io.IOException e) {
            System.err.println("Failed to initialize file logger: " + e.getMessage());
        }
    }

    @Override
    public void logInfo(String message) {
        if (shouldLog("INFO")) {
            String logEntry = createLogEntry("INFO", message);
            writer.println(logEntry);
            writer.flush();
            System.out.println("[FILE] " + logEntry);
        }
    }

    @Override
    public void logError(String message) {
        if (shouldLog("ERROR")) {
            String logEntry = createLogEntry("ERROR", message);
            writer.println(logEntry);
            writer.flush();
            System.out.println("[FILE] " + logEntry);
        }
    }

    @Override
    public void logWarning(String message) {
        if (shouldLog("WARN")) {
            String logEntry = createLogEntry("WARN", message);
```

```
            writer.println(logEntry);
            writer.flush();
            System.out.println("[FILE] " + logEntry);
        }
    }

    @Override
    public void logDebug(String message) {
        if (shouldLog("DEBUG")) {
            String logEntry = createLogEntry("DEBUG", message);
            writer.println(logEntry);
            writer.flush();
            System.out.println("[FILE] " + logEntry);
        }
    }

    @Override
    public void setLogLevel(String level) {
        this.logLevel = level.toUpperCase();
        System.out.println("FileLogger log level set to: " + level);
    }

    private boolean shouldLog(String messageLevel) {
        java.util.Map<String, Integer> levels = java.util.Map.of(
            "DEBUG", 1,
            "INFO", 2,
            "WARN", 3,
            "ERROR", 4
        );

        return levels.get(messageLevel) >= levels.getOrDefault(logLevel, 2);
    }

    private String createLogEntry(String level, String message) {
        return String.format("[%s] %s: %s",
                             java.time.LocalDateTime.now().toString(),
                             level, message);
    }

    public void close() {
        if (writer != null) {
            writer.close();
        }
    }
}

// DatabaseLogger implementation
class DatabaseLogger implements Logger {
    private String connectionString;
    private String logLevel;
    // Simulated database connection
    private java.util.List<String> logBuffer;

    public DatabaseLogger(String connectionString) {
        this.connectionString = connectionString;
        this.logLevel = "INFO";
        this.logBuffer = new java.util.ArrayList<>();
        System.out.println("DatabaseLogger connected to: " + connectionString);
    }

    @Override
    public void logInfo(String message) {
```

```
    if (shouldLog("INFO")) {
        String logEntry = createLogEntry("INFO", message);
        logBuffer.add(logEntry);
        System.out.println("[DATABASE] " + logEntry);
        // In real implementation, this would insert into database
    }
}

@Override
public void logError(String message) {
    if (shouldLog("ERROR")) {
        String logEntry = createLogEntry("ERROR", message);
        logBuffer.add(logEntry);
        System.out.println("[DATABASE] " + logEntry);
    }
}

@Override
public void logWarning(String message) {
    if (shouldLog("WARN")) {
        String logEntry = createLogEntry("WARN", message);
        logBuffer.add(logEntry);
        System.out.println("[DATABASE] " + logEntry);
    }
}

@Override
public void logDebug(String message) {
    if (shouldLog("DEBUG")) {
        String logEntry = createLogEntry("DEBUG", message);
        logBuffer.add(logEntry);
        System.out.println("[DATABASE] " + logEntry);
    }
}

@Override
public void setLogLevel(String level) {
    this.logLevel = level.toUpperCase();
    System.out.println("DatabaseLogger log level set to: " + level);
}

private boolean shouldLog(String messageLevel) {
    java.util.Map<String, Integer> levels = java.util.Map.of(
        "DEBUG", 1,
        "INFO", 2,
        "WARN", 3,
        "ERROR", 4
    );

    return levels.get(messageLevel) >= levels.getOrDefault(logLevel, 2);
}

private String createLogEntry(String level, String message) {
    return String.format("[%s] %s: %s",
                         java.time.LocalDateTime.now().toString(),
                         level, message);
}

// Simulate getting logs from database
public java.util.List<String> getLogs() {
    return new java.util.ArrayList<>(logBuffer);
}
```

```
    public void clearLogs() {
        logBuffer.clear();
        System.out.println("Database logs cleared");
    }
}

// ConsoleLogger implementation
class ConsoleLogger implements Logger {
    private String logLevel;

    public ConsoleLogger() {
        this.logLevel = "INFO";
    }

    @Override
    public void logInfo(String message) {
        if (shouldLog("INFO")) {
            System.out.println("ðŸ’¡ INFO: " + message);
        }
    }

    @Override
    public void logError(String message) {
        if (shouldLog("ERROR")) {
            System.out.println("â Œ ERROR: " + message);
        }
    }

    @Override
    public void logWarning(String message) {
        if (shouldLog("WARN")) {
            System.out.println("âš ï¸  WARN: " + message);
        }
    }

    @Override
    public void logDebug(String message) {
        if (shouldLog("DEBUG")) {
            System.out.println("ðŸ› DEBUG: " + message);
        }
    }

    @Override
    public void setLogLevel(String level) {
        this.logLevel = level.toUpperCase();
        System.out.println("ConsoleLogger log level set to: " + level);
    }

    private boolean shouldLog(String messageLevel) {
        java.util.Map<String, Integer> levels = java.util.Map.of(
            "DEBUG", 1,
            "INFO", 2,
            "WARN", 3,
            "ERROR", 4
        );

        return levels.get(messageLevel) >= levels.getOrDefault(logLevel, 2);
    }
}

// Application class with encapsulated logging
```

```
class Application {
    private Logger logger; // Encapsulated logger dependency
    private String appName;

    // Constructor injection for dependency inversion
    public Application(String appName, Logger logger) {
        this.appName = appName;
        this.logger = logger;
        logger.logInfo("Application '" + appName + "' initialized");
    }

    // Setter injection for runtime logger changes
    public void setLogger(Logger logger) {
        this.logger = logger;
        logger.logInfo("Logger changed for application '" + appName + "'");
    }

    public void performApplicationTask() {
        logger.logInfo("Starting application task...");

        try {
            // Simulate some work
            logger.logDebug("Step 1: Initializing components");
            simulateWork(100);

            logger.logDebug("Step 2: Processing data");
            simulateWork(200);

            // Simulate a potential issue
            if (Math.random() > 0.7) {
                logger.logWarning("Performance degradation detected");
            }

            logger.logDebug("Step 3: Finalizing operations");
            simulateWork(150);

            logger.logInfo("Application task completed successfully");

        } catch (Exception e) {
            logger.logError("Task failed: " + e.getMessage());
        }
    }

    public void performCriticalTask() {
        logger.logInfo("Starting critical task...");

        try {
            logger.logDebug("Critical operation step 1");
            simulateWork(300);

            // Simulate error
            if (Math.random() > 0.8) {
                throw new RuntimeException("Critical system failure");
            }

            logger.logDebug("Critical operation step 2");
            simulateWork(200);

            logger.logInfo("Critical task completed successfully");

        } catch (Exception e) {
            logger.logError("CRITICAL FAILURE: " + e.getMessage());
```

```
        }
    }

    private void simulateWork(int milliseconds) {
        try {
            Thread.sleep(milliseconds);
        } catch (InterruptedException e) {
            Thread.currentThread().interrupt();
            logger.logError("Work simulation interrupted");
        }
    }

    public void shutdown() {
        logger.logInfo("Application '" + appName + "' shutting down");
    }
}

// Demonstration class
public class EncapsulationInterfacesDemo {
    public static void main(String[] args) {
        System.out.println("=== ENCAPSULATION AND INTERFACES DEMO ===\n");

        // Create different loggers
        Logger fileLogger = new FileLogger("app.log");
        Logger databaseLogger = new DatabaseLogger("jdbc:mysql://localhost:3306/logs");
        Logger consoleLogger = new ConsoleLogger();

        // Set different log levels
        fileLogger.setLogLevel("DEBUG");
        databaseLogger.setLogLevel("WARN");
        consoleLogger.setLogLevel("INFO");

        // Create application with different loggers
        System.out.println("\n" + "=".repeat(50));
        System.out.println("TESTING WITH FILE LOGGER:");
        Application app1 = new Application("DataProcessor", fileLogger);
        testApplication(app1);

        System.out.println("\n" + "=".repeat(50));
        System.out.println("TESTING WITH DATABASE LOGGER:");
        Application app2 = new Application("WebService", databaseLogger);
        testApplication(app2);

        System.out.println("\n" + "=".repeat(50));
        System.out.println("TESTING WITH CONSOLE LOGGER:");
        Application app3 = new Application("CLI Tool", consoleLogger);
        testApplication(app3);

        // Demonstrate runtime logger switching
        System.out.println("\n" + "=".repeat(50));
        System.out.println("DEMONSTRATING RUNTIME LOGGER SWITCHING:");
        Application dynamicApp = new Application("DynamicApp", consoleLogger);
        dynamicApp.performApplicationTask();

        System.out.println("\n--- Switching to File Logger ---");
        dynamicApp.setLogger(fileLogger);
        dynamicApp.performCriticalTask();

        System.out.println("\n--- Switching to Database Logger ---");
        dynamicApp.setLogger(databaseLogger);
        dynamicApp.performApplicationTask();
```

```
        // Cleanup
        app1.shutdown();
        app2.shutdown();
        app3.shutdown();
        dynamicApp.shutdown();

        if (fileLogger instanceof FileLogger) {
            ((FileLogger) fileLogger).close();
        }
    }

    private static void testApplication(Application app) {
        app.performApplicationTask();
        app.performCriticalTask();
    }
}
```

5. Exception Handling: Data Processor

```
java
// Custom exception classes
class InvalidDataException extends Exception {
    public InvalidDataException(String message) {
        super(message);
    }

    public InvalidDataException(String message, Throwable cause) {
        super(message, cause);
    }
}

class DataValidationException extends InvalidDataException {
    public DataValidationException(String message) {
        super(message);
    }
}

class DataProcessingException extends Exception {
    public DataProcessingException(String message) {
        super(message);
    }

    public DataProcessingException(String message, Throwable cause) {
        super(message, cause);
    }
}

// DataProcessor class with comprehensive exception handling
class DataProcessor {
    private String processorName;
    private int processingCount;

    public DataProcessor(String processorName) {
        this.processorName = processorName;
        this.processingCount = 0;
    }

    // Main method with exception handling
    public double processData(int[] data) throws InvalidDataException, DataProcessingException {
        processingCount++;

        System.out.printf("%n--- Processing Attempt #%d ---%n", processingCount);
        System.out.println("Processor: " + processorName);
```

```
        try {
            // Validate input data
            validateData(data);

            // Process data
            double result = calculateAverage(data);

            System.out.printf("âœ… Successfully processed %d data points%n", data.length);
            System.out.printf("ðŸ“Š Result: %.2f%n", result);

            return result;

        } catch (InvalidDataException e) {
            // Log the specific validation error
            System.err.println("â Data Validation Error: " + e.getMessage());
            throw e; // Re-throw for caller to handle

        } catch (Exception e) {
            // Handle unexpected errors
            String errorMsg = String.format("Unexpected error during processing: %s",
e.getMessage());
            System.err.println("â " + errorMsg);
            throw new DataProcessingException(errorMsg, e);
        }
    }

    // Overloaded method with default values
    public double processDataWithDefault(int[] data, double defaultValue) {
        try {
            return processData(data);
        } catch (InvalidDataException | DataProcessingException e) {
            System.out.printf("âš ï¸  Using default value: %.2f%n", defaultValue);
            return defaultValue;
        }
    }

    // Batch processing with multiple datasets
    public void processMultipleDatasets(int[][] datasets) {
        System.out.println("\n" + "=".repeat(40));
        System.out.println("BATCH PROCESSING " + datasets.length + " DATASETS");

        int successCount = 0;
        int failureCount = 0;

        for (int i = 0; i < datasets.length; i++) {
            try {
                double result = processData(datasets[i]);
                successCount++;
                System.out.printf("Dataset %d: âœ… Success (Average: %.2f)%n", i + 1, result);

            } catch (InvalidDataException e) {
                failureCount++;
                System.out.printf("Dataset %d: â Failed - %s%n", i + 1, e.getMessage());

            } catch (DataProcessingException e) {
                failureCount++;
                System.out.printf("Dataset %d: ðŸ’¥ Processing Error - %s%n", i + 1,
e.getMessage());
            }
        }

        System.out.printf("%nBatch Processing Summary:%n");
```

```
        System.out.printf("âœ… Successful: %d%n", successCount);
        System.out.printf("â Failed: %d%n", failureCount);
        System.out.printf("ðŸ“ˆ Success Rate: %.1f%%%n", (successCount * 100.0 /
datasets.length));
    }

    // Data validation method
    private void validateData(int[] data) throws InvalidDataException {
        if (data == null) {
            throw new InvalidDataException("Data array cannot be null");
        }

        if (data.length == 0) {
            throw new InvalidDataException("Data array cannot be empty");
        }

        // Check for reasonable array size
        if (data.length > 1000) {
            throw new DataValidationException("Data array too large: " + data.length + "
elements (max: 1000)");
        }

        // Validate each data point
        for (int i = 0; i < data.length; i++) {
            if (data[i] < 0) {
                throw new DataValidationException(
                    String.format("Negative value at index %d: %d", i, data[i])
                );
            }

            // Check for unreasonably large values
            if (data[i] > 1_000_000) {
                throw new DataValidationException(
                    String.format("Unreasonably large value at index %d: %d", i, data[i])
                );
            }
        }

        System.out.printf("âœ“ Validated %d data points%n", data.length);
    }

    // Calculate average with additional checks
    private double calculateAverage(int[] data) throws DataProcessingException {
        try {
            long sum = 0;

            for (int value : data) {
                // Check for integer overflow
                if (sum > 0 && value > Long.MAX_VALUE - sum) {
                    throw new DataProcessingException("Integer overflow detected during
summation");
                }
                sum += value;
            }

            double average = (double) sum / data.length;

            // Check for mathematical validity
            if (Double.isNaN(average) || Double.isInfinite(average)) {
                throw new DataProcessingException("Invalid average calculation result");
            }
```

```
            return average;

        } catch (ArithmeticException e) {
            throw new DataProcessingException("Arithmetic error during average calculation", e);
        }
    }

    // Statistical methods with exception handling
    public double calculateMedian(int[] data) throws InvalidDataException {
        validateData(data);

        try {
            int[] sortedData = data.clone();
            java.util.Arrays.sort(sortedData);

            int middle = sortedData.length / 2;
            if (sortedData.length % 2 == 0) {
                return (sortedData[middle - 1] + sortedData[middle]) / 2.0;
            } else {
                return sortedData[middle];
            }
        } catch (Exception e) {
            throw new DataProcessingException("Error calculating median", e);
        }
    }

    // Utility method to display data statistics
    public void displayDataStatistics(int[] data) {
        System.out.println("\n--- Data Statistics ---");

        try {
            double average = processData(data);
            double median = calculateMedian(data);

            System.out.printf("Data Points: %d%n", data.length);
            System.out.printf("Average: %.2f%n", average);
            System.out.printf("Median: %.2f%n", median);

            // Additional stats
            int min = java.util.Arrays.stream(data).min().getAsInt();
            int max = java.util.Arrays.stream(data).max().getAsInt();
            System.out.printf("Range: %d to %d%n", min, max);

        } catch (InvalidDataException e) {
            System.out.println("Cannot compute statistics: " + e.getMessage());
        } catch (DataProcessingException e) {
            System.out.println("Error computing statistics: " + e.getMessage());
        }
    }

    // Getters
    public String getProcessorName() { return processorName; }
    public int getProcessingCount() { return processingCount; }
}

// Demonstration class
public class ExceptionHandlingDemo {
    public static void main(String[] args) {
        System.out.println("=== EXCEPTION HANDLING DEMO ===\n");

        DataProcessor processor = new DataProcessor("Main Data Processor");
```

```
// Test cases with various scenarios
int[][] testDatasets = {
    {10, 20, 30, 40, 50},           // Valid data
    {},                              // Empty array
    null,                            // Null array
    {1, 2, 3, 4, 5},                // Valid small dataset
    {-1, 2, 3},                     // Negative numbers
    {Integer.MAX_VALUE, 1},         // Potential overflow
    new int[1001],                  // Too large array
    {0, 0, 0, 0},                   // All zeros
    {100, 200, 300, 400, 500}       // Another valid dataset
};

// Test individual processing with exception handling
System.out.println("INDIVIDUAL DATA PROCESSING TESTS:");
for (int i = 0; i < testDatasets.length; i++) {
    System.out.printf("%nTest Case %d:%n", i + 1);

    try {
        double result = processor.processData(testDatasets[i]);
        System.out.printf("ðŸŽ‰ Processing successful! Result: %.2f%n", result);

    } catch (InvalidDataException e) {
        System.out.printf("âš ï¸  Data validation failed: %s%n", e.getMessage());

    } catch (DataProcessingException e) {
        System.out.printf("ðŸ’¥ Processing error: %s%n", e.getMessage());

    } catch (Exception e) {
        System.out.printf("âŒ Unexpected error: %s%n", e.getMessage());
    }
}

// Test batch processing
System.out.println("\n" + "=".repeat(50));
processor.processMultipleDatasets(testDatasets);

// Test with default value fallback
System.out.println("\n" + "=".repeat(50));
System.out.println("TESTING WITH DEFAULT VALUE FALLBACK:");

int[] problematicData = {}; // Empty array
double result = processor.processDataWithDefault(problematicData, -1.0);
System.out.printf("Final result with fallback: %.2f%n", result);

// Demonstrate statistics
System.out.println("\n" + "=".repeat(50));
System.out.println("DATA STATISTICS DEMONSTRATION:");

int[] sampleData = {15, 25, 35, 45, 55, 65};
processor.displayDataStatistics(sampleData);

// Test edge cases
System.out.println("\n" + "=".repeat(50));
System.out.println("EDGE CASE TESTING:");

try {
    // Single element array
    double singleResult = processor.processData(new int[]{42});
    System.out.printf("Single element: %.2f%n", singleResult);

    // Large numbers
```

```
            double largeResult = processor.processData(new int[]{1000000, 2000000, 3000000});
            System.out.printf("Large numbers: %.2f%n", largeResult);

        } catch (Exception e) {
            System.out.println("Edge case failed: " + e.getMessage());
        }

        // Display final processing statistics
        System.out.println("\n" + "=".repeat(50));
        System.out.println("FINAL PROCESSOR STATISTICS:");
        System.out.printf("Processor: %s%n", processor.getProcessorName());
        System.out.printf("Total processing attempts: %d%n", processor.getProcessingCount());
    }
}
```