Java Abstract Classes & Interfaces - Industry Scenario Based Assignments

1. Payment Gateway Integration

Problem Statement:
You are developing an e-commerce application for a retail company that supports multiple payment methods. Customers can pay through Stripe or PayPal. Each payment method needs to connect to its respective API, process a payment, and process a refund.

Requirements:

- Create an abstract class PaymentGateway with:
    - Method connect() → prints "Connected to <gateway> API".
    - Method disconnect() → prints "Disconnected from <gateway> API".
    - Common attributes like API keys or connection URL.
- Create two interfaces:
    - PaymentProcessor → method processPayment(double amount)
    - RefundProcessor → method processRefund(double amount)
- Create classes:
    - StripeGateway → extends PaymentGateway, implements both interfaces.
    - PayPalGateway → extends PaymentGateway, implements both interfaces.
- In the main method: connect, process payment, process refund, disconnect.

Hint:
Abstract class should handle common API connection logic. Interfaces ensure consistent payment and refund method signatures.

Expected Output:

Connected to Stripe API

Processing payment of $500 via Stripe...

Processing refund of $100 via Stripe...

Disconnected from Stripe API

Connected to PayPal API

Processing payment of $250 via PayPal...

Processing refund of $50 via PayPal...

Disconnected from PayPal API

---

2. Vehicle Service Management System

Problem Statement:
A vehicle service company needs a system to manage servicing and repairing different vehicle types like cars and bikes.

Requirements:

- Create an abstract class Vehicle with:
    - Properties: brand, model.
    - Abstract method service().
- Create an interface Repairable with method repair().
- Create classes Car and Bike that:
    - Extend Vehicle.
    - Implement Repairable.
- In the main method: Create objects, call service() and repair().

Hint:
Put common properties in the abstract class. Let each subclass decide its own service logic.

Expected Output:

Servicing car: Toyota Corolla

Repairing car engine...

Servicing bike: Yamaha FZ

Repairing bike brakes...

---

3. Online Learning Platform

Problem Statement:
An online learning platform offers programming and design courses. Students can enroll, start the course, and receive a certificate upon completion.

Requirements:

- Create an abstract class Course with:
    - Properties: courseName, duration.
    - Abstract methods: enrollStudent(String studentName), startCourse().
- Create an interface CertificateProvider with method generateCertificate(String studentName).
- Create classes ProgrammingCourse and DesignCourse:
    - Extend Course.
    - Implement CertificateProvider.

- In the main method: Enroll students, start course, generate certificate.

Hint:
Use abstract class for course details and common behavior, interface for certificate generation.

Expected Output:

Enrolling John in Java Programming Course

Starting Java Programming Course

Certificate generated for John in Java Programming Course

Enrolling Alice in Graphic Design Course

Starting Graphic Design Course

Certificate generated for Alice in Graphic Design Course

---

4. Banking System Loan Processing

Problem Statement:
A bank offers home and car loans. Each loan type shares EMI calculation logic but has different approval criteria.

Requirements:

- Create an abstract class Loan with:
    - Method calculateEMI(double principal, double rate, int tenureMonths).
- Create an interface ApprovalProcess with method approveLoan().
- Create classes HomeLoan and CarLoan:
    - Extend Loan.
    - Implement ApprovalProcess.
- In the main method: Calculate EMI, approve loans.

Hint:
Abstract class for EMI formula; interface for loan approval rules.

Expected Output:

Home Loan EMI for $50000 is $1200/month

Home Loan Approved

Car Loan EMI for $20000 is $450/month

Car Loan Approved

---

5. Smart Home Automation

Problem Statement:
A smart home app controls devices like lights and AC via normal switch and voice commands.

Requirements:

- Create an abstract class SmartDevice with:
    - Property: deviceName.
    - Abstract method turnOn().
- Create an interface VoiceControl with method controlByVoice(String command).
- Create classes SmartLight and SmartAC:
    - Extend SmartDevice.
    - Implement VoiceControl.
- In the main method: Turn on devices and control via voice.

Hint:
Abstract class for shared device attributes, interface for voice control capability.

Expected Output:

Turning on Smart Light

Smart Light controlled by voice: 'Turn on the lights'

Turning on Smart AC

Smart AC controlled by voice: 'Set temperature to 22°C'

---

6. Restaurant Order Management

Problem Statement:
A restaurant manages orders digitally for items like pizza and burgers. Each item has preparation steps and delivery steps.

Requirements:

- Create an abstract class MenuItem with:
    - Property: itemName.
    - Abstract method prepare().
- Create an interface Deliverable with method deliver(int tableNumber).
- Create classes Pizza and Burger:

    - Extend MenuItem.
    - Implement Deliverable.
- In the main method: Prepare and deliver orders.

Hint:
Abstract class for preparation steps, interface for delivery process.

Expected Output:

Preparing Pizza: Margherita

Delivering Pizza to table 5

Preparing Burger: Veggie Burger

Delivering Burger to table 2