1. A **Smart Device System** is being designed for an IoT (Internet of Things) ecosystem. The system includes various smart devices, such as **SmartPhones** and **SmartWatches**. All smart devices share some common attributes and behaviors but also have unique functionalities.

   - A **Device** has attributes: brand, model, and battery_Capacity. It also has a method turn-on().
   - A **SmartPhone** has an extra attribute OS (operating system) and a method makeCall ().
   - A **SmartWatch** has an additional attribute stepCounter and a method trackSteps ().
   - A **SmartPhone** can be linked to multiple **Wearable devices**, while a **SmartWatch** can connect to at most **one SmartPhone** but multiple **Wearables**. Each **Wearable** must be linked to exactly **one Device**

Task:

a) Draw a **UML Class Diagram** representing these relationships with proper **multiplicity notations** and explain the associations between classes.
b) Identify whether **Device** should be an **abstract class or a superclass** in an inheritance hierarchy.
c) Analyze how **Wearable** connects with SmartWatch. Should it be **inheritance or composition**?
d) Does **SmartPhone** inherit from **Device**, or is it an interface implementation?

2. A **Payment Processing System** is being developed for an online marketplace. The system supports different types of payments, such as **Credit Card Payments** and **Crypto Payments**.
   - A **Payment** class has attributes: amount, currency, and a method processPayment().
   - A **CreditCardPayment** class extends Payment and has attributes cardNumber, cardHolderName, and a method validateCard().
   - A **CryptoPayment** class extends Payment and has an attribute walletAddress with a method verifyTransaction().
   - There is also an **interface called Refundable**, which has a method issueRefund().

Task:

a) Draw a **UML Class Diagram** for this system, including all relationships.
b) Determine if **Payment** should be an **abstract class** or a **concrete class**.
c) Analyze whether Refundable should be an **interface or an abstract class**.
d) If both **CreditCardPayment** and **CryptoPayment** support refunds, how should they implement Refundable?

3. Create a **BankAccount** class with the following:
   - Private instance variables: accountNumber, balance
   - Public methods: deposit(double amount), withdraw(double amount), and getBalance()
   - Ensure that balance cannot be set directly and withdrawals cannot exceed the available balance.

Task:
Write a Java program that creates a BankAccount object, deposits 5000, withdraws 2000, and displays the remaining balance.

4. Create a **Product** class that has:
   - Private variables: productName, price, stockQuantity
   - Constructor overloading: One constructor initializes all attributes, another initializes only productName and price.
   - A method purchase(int quantity) that reduces stock and prevents purchasing more than available stock.

Task:
- Create multiple Product objects and simulate buying products using method calls.