# ENCAPSULATION

1. **Bank Account System with Multiple Transactions**

    - Design a `BankAccount` class with private fields: `accountNumber`, `balance`, and `transactionHistory`.
    - Use getter and setter methods for `balance` and `accountNumber`.
    - Ensure the setter for balance allows only positive values.
    - Create methods:
        - `deposit(double amount)` to add money.
        - `withdraw(double amount)` to subtract money (if balance is sufficient).
        - `getTransactionHistory()` to return a list of all transactions made.
    - In the `Main` class, simulate a sequence of transactions and display the transaction history.

2. **Employee Salary Management with Complex Validation**

    - Create an `Employee` class with private fields: `empId`, `name`, `baseSalary`, and `bonusPercentage`.
    - Use setters with validation to ensure no negative values are set for salary or bonus.
    - Create a method `getGrossSalary()` to compute the total salary as: `baseSalary + (baseSalary * bonusPercentage / 100)`.
    - Add a `displaySalaryDetails()` method that prints the employee's salary details.
    - In the `Main` class, create employee objects, set their details, and print their salary details after validation.

3. **Student Grade Management with Exception Handling**

    - Design a `Student` class that uses encapsulation to manage grades.
    - Fields: `name`, `rollNumber`, `grades[]` (array of integers).
    - Create a setter method for grades that throws an exception if any grade is outside the valid range (0–100).
    - Implement a `calculateAverage()` method to compute the average grade.
    - In the `Main` class, handle exceptions gracefully when setting invalid grades and compute the average.

---

# INHERITANCE

1. **Employee System with Role Hierarchy**

    - Create an abstract class `Employee` with common fields: `empId`, `name`, and `salary`.
    - Add an abstract method `calculateSalary()`.
    - Create subclasses:
        - `Manager`: Inherits from `Employee`, adds a field `department`, and overrides `calculateSalary()` to include department-specific allowance.
        - `Salesman`: Inherits from `Employee`, adds a field `salesAmount`, and overrides `calculateSalary()` to compute commission-based salary.
    - In the `Main` class, create objects of `Manager` and `Salesman` and demonstrate the polymorphic behavior of the `calculateSalary()` method.

2. **Library Management System with Multiple Item Types**

- Create a base class `LibraryItem` with fields: `title`, `author`, and a method `displayInfo()`.

- Create subclasses `Book` and `Magazine` that inherit from `LibraryItem`:
  - `Book`: Adds a `pages` field and overrides `displayInfo()` to include pages.
  - `Magazine`: Adds an `issueMonth` field and overrides `displayInfo()` to include the month.

- In the `Main` class, create several objects of `Book` and `Magazine` and demonstrate method overriding and polymorphism.

3. **Vehicle System with Overriding and Dynamic Binding**

   - Create a base class `Vehicle` with fields `vehicleId`, `capacity`, and method `move()`.

   - Create subclasses:
     - `Car`: Overrides `move()` to simulate a car's movement.
     - `Truck`: Overrides `move()` to simulate a truck's movement.

   - In the `Main` class, create instances of both `Car` and `Truck` and use dynamic method binding to demonstrate polymorphic behavior when calling `move()`.

---

# INTERFACE & ABSTRACT CLASSES

1. **Payment System with Multiple Payment Methods**

- Create an interface `PaymentMethod` with the method `processPayment(double amount)`.

- Create an abstract class `OnlinePayment` that implements `PaymentMethod` and includes fields like `accountId` and a concrete method `connect()`.

- Subclasses of `OnlinePayment`:
  - `CreditCardPayment`: Implements `processPayment()` to handle payments via credit card.
  - `UPIPayment`: Implements `processPayment()` to handle payments via UPI.

- In the `Main` class, demonstrate processing different types of payments.

2. **Smart Home Device Management System**

- Create an interface `SmartDevice` with methods: `turnOn()`, `turnOff()`, and `getStatus()`.

- Create an abstract class `Appliance` that implements `SmartDevice` with a field `brand`.

- Subclasses of `Appliance`:
  - `SmartLight`: Implements `turnOn()` and `adjustBrightness()`.
  - `SmartThermostat`: Implements `turnOn()` and `setTemperature()`.

- In the `Main` class, simulate controlling the devices by turning them on and adjusting settings.

3. **Employee Attendance System with Multiple Employee Types**

- Create an interface `Employee` with methods: `markAttendance()` and `getDetails()`.

- Create an abstract class `FullTimeEmployee` that implements `Employee` and includes fields like `monthlySalary`.

- Subclasses of `FullTimeEmployee`:
  - `Manager`: Adds additional responsibility fields and overrides `markAttendance()` to include manager-specific details.
  - `Intern`: Adds fields like `workingHours` and overrides `markAttendance()` for part-time attendance tracking.

- In the `Main` class, create instances of `Manager` and `Intern`, and track their attendance.

---

# COLLECTIONS

1. **Task Management System Using a List**

   - Create a `Task` class with fields: `taskName`, `deadline`, and `status`.

   - Use an `ArrayList` to manage tasks.

   - Methods to:
     - Add a task.
     - Update task status (e.g., "To Do", "In Progress", "Completed").
     - Remove tasks.

- In the `Main` class, demonstrate task management by adding, updating, and removing tasks.

2. **Library System Using a HashMap**

   - Design a system to manage a library's book inventory using a `HashMap`, where the key is the ISBN number.
   - Each `Book` object should have fields: `title`, `author`, and `isbnNumber`.
   - Provide methods to add, update, and delete books.
   - In the `Main` class, simulate the management of books using the HashMap and demonstrate basic CRUD operations.

3. **Shopping Cart System with Custom Classes**

   - Create a `CartItem` class with fields: `itemName`, `quantity`, and `price`.
   - Use an `ArrayList` to represent the shopping cart.
   - Methods to:
     - Add an item to the cart.
     - Remove an item by name.
     - Display the total price.
   - In the `Main` class, add at least 5 items to the cart, remove some, and display the cart contents.

---