INHERITANCE

1. Smart Home System with Multi-Device Control
Design a complex system that models various smart devices in a home. Devices have common characteristics but exhibit different behaviors.

- Create a **SmartDevice** base class with common properties like `brand`, `model`, and methods `turnOn()`, `turnOff()`, and `getStatus()`.
- Create subclasses:
  - **SmartLight** (adjust brightness and color temperature).
  - **SmartThermostat** (adjust temperature).
  - **SmartSpeaker** (play music, adjust volume).
- Each device should override `turnOn()` and `turnOff()` in a unique way.
- In the **Main** class, create a **SmartHome** that can add/remove devices and turn them on/off simultaneously.

---

2. Enhanced Payroll System with Tax Calculation
Design a payroll system where employees can either have a fixed salary or be paid based on commissions, and tax is deducted based on income.

- Create an abstract **Employee** class with fields `name`, `id`, and abstract method `calculateSalary()`.
- Implement two subclasses:
  - **SalariedEmployee**: Has a fixed salary and includes a `calculateTax()` method (e.g., 10% tax).

    - **CommissionedEmployee**: Earns commission based on sales and includes a `calculateTax()` method (e.g., 15% tax).

- Add a `payEmployee()` method in a **Payroll** class to calculate the final salary after tax and print salary details.

- In the **Main** class, demonstrate calculating and paying employees, applying tax deductions.

---

3. Advanced Vehicle Fleet Management System

Design a more complex vehicle management system where different types of vehicles perform delivery tasks with varying performance and fuel efficiency.

- Create a **Vehicle** base class with attributes like `vehicleId`, `capacity`, `fuelEfficiency`, and method `deliverGoods()`.

- Create subclasses:
    - **Truck**: Large capacity, low fuel efficiency, long delivery distance.
    - **Motorcycle**: Smaller capacity, high fuel efficiency, fast delivery.
    - **Drone**: Very small capacity, very high efficiency, fast but limited range.

- Each subclass should implement a specialized `deliverGoods()` method that calculates fuel consumption and time based on the delivery distance and capacity.

- In the **Main** class, create a **FleetManager** class to manage the fleet and track delivery performance.

COLLECTIONS

1. Project Task Dependency System
Design a system that tracks tasks in a project with dependencies between tasks (some tasks must be completed before others).

- Create a **Task** class with fields like `taskName`, `status`, and `dependencies` (a list of other tasks that must be completed before this one).

- Implement a **TaskManager** class that:
  - Adds tasks to a project.
  - Tracks task dependencies (using a `HashMap` to map each task to a list of dependent tasks).
  - Updates the status of each task, ensuring dependencies are completed before a task can be marked as "Completed".

- In the **Main** class, create a few tasks with dependencies and simulate a project workflow.

---

2. Student Course Enrollment System
Design a system to manage student enrollments in various courses, where each course has prerequisites.

- Create a **Student** class with attributes like `name`, `id`, and a list of courses they are enrolled in.

- Create a **Course** class with attributes like `courseName`, `courseCode`, and a list of prerequisite courses.

- Implement a **University** class that can:
  - Enroll students in courses.

    - Ensure that prerequisites are completed before enrolling in advanced courses.

- In the **Main** class, simulate course enrollments and check if students can enroll in advanced courses based on their completed prerequisites.

---

3. Shopping Cart with Discounts
Design a more advanced shopping cart system where users can add, remove, and apply discounts to items.

- Each **Item** has properties like `name`, `price`, and `quantity`.

- Implement a **ShoppingCart** class to:
    - Add items to the cart.
    - Remove items by name.
    - Apply a discount (percentage-based) to the total cart value.
    - Calculate the final cart price after discount.

- In the **Main** class, create a **DiscountManager** to apply different discount strategies (e.g., seasonal discount, loyalty discount).

INTERFACE & ABSTRACT CLASSES

1. Payment System with Multi-Currency Support
Design a complex payment system that supports multiple payment methods and currencies.

- Create an interface **PaymentMethod** with a method `processPayment(double amount, String currency)`.

- Create an abstract class **OnlinePayment** that implements the interface and has a `currency` field.

- Implement subclasses:
  - **CreditCardPayment**: Supports processing in different currencies.
  - **UPIPayment**: Supports processing in different currencies with a transaction fee.

- In the **Main** class, implement a **PaymentProcessor** to process payments and convert currencies based on exchange rates.

---

2. Home Automation System with Energy Management
Design an advanced system that controls home appliances with energy monitoring.

- Create an interface **EnergyEfficientDevice** with methods `turnOn()`, `turnOff()`, and `getEnergyConsumption()`.

- Create an abstract class **HomeAppliance** that implements this interface and includes `brandName` and `model`.

- Implement subclasses:
  - **SmartAirConditioner**: Can adjust the temperature and monitor energy consumption.
  - **SmartFridge**: Can adjust cooling and monitor energy consumption.
  - **SmartWashingMachine**: Can adjust washing cycles and monitor energy consumption.

- In the **Main** class, simulate energy usage across multiple devices and calculate the total energy consumption.

---

3. Employee Benefits System
Design an employee system that tracks attendance, leave days, and benefits.

- Create an interface **Employee** with methods `markAttendance()`, `applyLeave()`, and `getBenefits()`.

- Create an abstract class **FullTimeEmployee** that implements attendance tracking and leave application.

- Implement subclasses:
  - **PermanentEmployee**: Receives paid leave and full benefits.
  - **ContractEmployee**: Receives limited leave and benefits based on contract terms.

- In the **Main** class, track employee attendance, apply leave days, and calculate benefits based on employment type.

ENCAPSULATION

1. Bank Account System with Transaction History
Design a **BankAccount** class that manages account transactions and keeps a record of all deposits and withdrawals.

- The class should have private fields: `accountNumber`, `balance`, and a `transactionHistory` list.

- Provide getter methods for the account number and balance, and a setter for the balance (which ensures only valid, non-negative deposits/withdrawals).

- Add methods to:
    - Deposit and withdraw money.
    - Retrieve the transaction history.
- In the **Main** class, demonstrate multiple transactions and print the transaction history.

---

2. Advanced Employee Salary System with Deductions

Design an advanced **Employee** class to manage salary calculations with deductions for tax, insurance, and pension.

- The class should have private fields: `empId`, `name`, `baseSalary`, `taxRate`, `insuranceRate`, and `pensionRate`.
- Provide setters that validate input (no negative salary, tax, or pension rates).
- Add a method **calculateNetSalary()** that computes salary after applying all deductions (tax, insurance, pension).
- In the **Main** class, create employee objects, set their details, and calculate their net salaries.

---

3. Grade Management System with Weightage

Design a system for managing student grades, where each subject has different weightage.

- The **Student** class should have private fields: `name`, `rollNumber`, and `subjects` (a list of subjects with grades).
- The **Subject** class should have fields: `subjectName`, `grade`, and `weightage`.
- Implement a method to calculate the **weighted average grade**.
- In the **Main** class, create students and calculate their weighted average grades.
