Inheritance

1. Animal Hierarchy - Managing Different Animal Types

Design a zoo management system where animals share common attributes like name, age, and sound, but have specialized behaviors based on their type. Create an `Animal` base class with common attributes and a `makeSound()` method. Then, create subclasses like `Mammal`, `Bird`, and `Reptile`, each overriding `makeSound()` to produce a sound specific to that type (e.g., lion's roar, parrot's squawk, snake's hiss). Demonstrate by creating instances of these animals and showing their sounds.

---

2. Employee Salary System - Full-Time and Part-Time Employees

Create a payroll system that handles full-time and part-time employees with different salary structures. Define an `Employee` base class with properties for name and ID, and an abstract method `calculateSalary()`. Implement subclasses like `FullTimeEmployee` (with a fixed salary) and `PartTimeEmployee` (with an hourly rate). Add a method to display employee information, including their calculated salary, and demonstrate by creating both types of employees.

---

3. Vehicle Inheritance - Managing Different Vehicle Types

Build a transportation system for managing different vehicle types, where all vehicles can move but differ in fuel efficiency and maximum speed. Create a `Vehicle` base class with common attributes like model, fuel efficiency, and maximum speed. Then, implement subclasses like `Car`, `Truck`, and `Motorcycle`, each with its own behavior (e.g., overriding `move()`, adjusting fuel efficiency and speed). Show their unique behavior by creating instances of each vehicle.

Collections

1. Task Management System

Build a system to manage tasks with features to add, update, remove, and display tasks based on their status. Each task includes a name, deadline, and status. Implement a `Task` class with properties for task name, deadline, and status, and a `TaskManager` class to handle tasks with methods to add tasks, remove tasks, update status, and display tasks by status. Ensure that attempts to remove non-existent tasks are handled gracefully.

---

2. Inventory Management System

Create an inventory system to manage products in a store. The system should allow adding, deleting, modifying, and checking the availability of items. Implement an `InventoryItem` class with properties for item name and quantity, and an `Inventory` class to manage items with methods to add items, delete items, modify quantities, display all items, and check if an item exists. Ensure proper handling of item operations.

---

3. Shopping List Application

Design a shopping list application where users can add, remove, and check for items, and display the updated list. The system should allow users to add at least 5 items to the list, remove items by name, check if an item exists, and show the full list after each operation.

Inheritance and Abstract Classes

1. Vehicle Management System

Build a system for managing a fleet of vehicles, including cars, bikes, and trucks. All vehicles have basic functionality such as starting, stopping, and accelerating. Trucks have an extra feature for loading cargo, while cars and bikes do not.

Create an interface `Vehicle` with methods `start()`, `stop()`, and `accelerate()`. Then, create an abstract class `Truck` that implements `Vehicle` and adds an abstract method `loadCargo()`. Create concrete classes `Car` and `Bike` that implement `Vehicle`, and a concrete class `CargoTruck` that extends `Truck` and implements `loadCargo()`.

---

2. Music Player Example - Abstract Class and Interface

Design a music player application where there are different types of music players: **MP3Player** and **RadioPlayer**. Both should be able to play, pause, and stop music, but they will have different behaviors.

Create an interface `Playable` with methods `play()`, `pause()`, and `stop()`. Then, create an abstract class `AbstractPlayer` implementing `Playable`, which defines common properties like `volume` and `trackName`. Create two concrete classes: `MP3Player` (which can load tracks) and `RadioPlayer` (which plays radio stations).

---

3. Employee Management System

Design a system to manage employees, both full-time and part-time. Full-time employees have a fixed salary, while part-time employees have an hourly rate.

Create an interface `Employee` with methods `calculateSalary()` and `getDetails()`. Define two abstract classes: `FullTimeEmployee` (with a fixed salary) and `PartTimeEmployee` (with an hourly rate and hours worked). Then, create concrete classes `Manager` (a full-time employee) and `Intern` (a part-time employee) to calculate and print their respective salaries.

ENCAPSULATION

1. Employee Class - Managing Salary and Leaves

Design a **Human Resources system** that tracks employee details such as their name, ID, salary, and available leave days. The system needs to ensure that salaries are not set below a certain minimum, and leave days cannot be negative. Implement methods to manage employee leave requests and salary updates. When an employee applies for leave, the system should decrease the number of available leave days, ensuring there are enough days left. Additionally, salary increases must be validated to ensure they are reasonable.

2. Bank Account - Balance Management with Transaction History

Build a **Bank Account** class where each account has a balance, an account number, and a history of transactions. The system should prevent the direct setting of the account balance and only allow balance updates through deposit or withdrawal operations. Transactions should be tracked in a history list. The withdrawal operation should ensure the account has sufficient funds before proceeding, and each transaction should be recorded.

3.Student Management - Securing Grades Data

In a **Student Management System**, sensitive student data like grades need to be handled securely. Each student has personal information such as name and roll number, and an array of grades for various subjects. While the name and roll number should be accessible, the grades should be protected. Students' grades can be updated only through a method that ensures the values are within a valid range (0 to 100). Additionally, a method to calculate the average grade is required.
