
OOP – Set 3: Intermediate Complexity

Exercise 1: Vehicle Rental System – Inheritance & Polymorphism

Problem:

Create a class hierarchy for different types of rental vehicles.

Instructions:

1. Create an abstract class `Vehicle` with:
    - Properties: `VehicleNumber`, `Brand`, `RatePerDay`
    - Method: `virtual CalculateRent(int days)`
2. Create derived classes: `Car`, `Bike`, `Truck`
    - Each overrides `CalculateRent()` with specific logic (e.g., base + surcharge)
3. In `Main()`, create a `List<Vehicle>` and display rent for each using polymorphism.

Exercise 2: E-Commerce Cart – Composition

Problem:

Build a simple shopping cart system using object composition.

Instructions:

1. Create class `Product` with: `Id`, `Name`, `Price`
2. Create class `CartItem` with: `Product`, `Quantity`, `GetTotalPrice()`
3. Create class `ShoppingCart` with:
    - List of `CartItem`s
    - Methods: `AddItem()`, `RemoveItem()`, `GetCartTotal()`
4. In `Main()`, add 2–3 products and print total cart value.

Exercise 3: Staff Management – Interface + Abstract Class

Problem:

Use both an interface and abstract class to model employees.

Instructions:

1. Create an interface `IAttendance` with method `MarkAttendance()`
2. Create an abstract class `Staff` with:
    - Properties: `Id`, `Name`, `Department`
    - Abstract method: `CalculateSalary()`
3. Create classes `PermanentStaff` and `ContractStaff`:

- Implement both `IAttendance` and `Staff`
- Implement `CalculateSalary()` differently

4. In `Main()`, use polymorphism to work with a list of staff.

Exercise 4: Library System – Aggregation and Search

Problem:

Create a mini-library that can search for books by author or title.

Instructions:

1. Create class `Book`: `Id`, `Title`, `Author`, `IsAvailable`
2. Create class `Library` with a `List<Book>`
3. Methods:
   - `AddBook(Book book)`
   - `SearchByAuthor(string author)`
   - `SearchByTitle(string title)`
4. In `Main()`, add sample books and perform both searches.

Exercise 5: Online Course Platform – Overloading + Interface

Problem:

Allow users to enroll in courses using different input combinations.

Instructions:

1. Create class `Course`: `Id`, `Title`, `Fee`
2. Create interface `IEnrollable` with method `Enroll()`
3. Create class `User` with overloaded `Enroll()`:
   - `Enroll(Course course)`
   - `Enroll(Course course, string couponCode)`
   - Use coupon to reduce fee
4. Print enrolled courses and final amount.

Exercise 6: Zoo Management – Abstract Class + Hierarchy

Problem:

Use an animal hierarchy to demonstrate behavior and feeding logic.

Instructions:

1. Create abstract class `Animal`:
   - Properties: `Name`, `Age`
   - Abstract method: `MakeSound()`
   - Concrete method: `Feed() => "Feeding the animal"`

2. Create classes `Lion`, `Elephant`, `Monkey`
    - Each overrides `MakeSound()`
3. Create a `Zoo` class that holds a list of animals and invokes all actions.
