**📝 Exercise 1: Interface and Abstract Class - Smart Home Devices**

**💡 Problem Statement:**

Create a system to manage **smart home devices** using interfaces and abstract classes.

**Instructions:**

1. Create an **abstract class** **SmartDevice** with:
   * An **abstract method**: **TurnOn()**
   * A **concrete method**: **ShowStatus()** to print **"Device status: Active"**
2. Create an **interface** **IRemoteControl** with methods:
   * **IncreaseVolume()**
   * **DecreaseVolume()**
3. Create derived classes:
   * **SmartLight**:
     + Implements **TurnOn()** to print **"Smart Light turned on"**
     + Implements **IncreaseVolume()** and **DecreaseVolume()** to print **"Lights do not support volume control"**
   * **SmartSpeaker**:
     + Implements **TurnOn()** to print **"Smart Speaker turned on"**
     + Implements **IncreaseVolume()** and **DecreaseVolume()** to print **"Volume increased/decreased"**
4. In the **Main()** method:
   * Create objects of **SmartLight** and **SmartSpeaker** using **SmartDevice** reference.
   * Call **ShowStatus()**, **TurnOn()**, **IncreaseVolume()**, and **DecreaseVolume()** for both.

**📝 Exercise 2: Collections - Employee Salary Management**

**💡 Problem Statement:**

Manage **employee salary records** using a **List<T>** and **Dictionary<TKey, TValue>**.

**Instructions:**

1. Create a class **Employee** with fields:
   * EmployeeID (int)
   * Name (string)
   * Salary (decimal)
2. In the **Main()** method:
   * Create a **List<Employee>** to store multiple employees.
   * Add at least **5 employees** to the list.
   * Print the **name and salary** of the employee with the **highest salary**.
   * Store employee **ID and Name** in a **Dictionary**.
   * Display the dictionary contents.
   * Search for an employee by **ID** and print the **name** if found.

**📝 Exercise 3: Queue and Stack - Library Book Management**

**💡 Problem Statement:**

Implement a **library book management system** using **Queue<T>** and **Stack<T>**.

**Instructions:**

1. Create a class **Book** with fields:
   * Title (string)
   * Author (string)
2. Use a **Queue<Book>** to manage **borrowed books**:
   * Add **5 borrowed books** to the queue.
   * Display the **next book to be returned**.
   * Remove the book after it is returned.
3. Use a **Stack<Book>** to manage **newly purchased books**:
   * Add **3 new books** to the stack.
   * Display the **last book purchased**.
   * Remove a book after it is processed.
4. Print the **remaining books** in both the **queue** and **stack**.

**📝 Exercise 4: Abstract Class and Polymorphism - Payment Processing**

**💡 Problem Statement:**

Create an abstract class to handle **multiple payment methods** with polymorphism.

**Instructions:**

1. Create an **abstract class** **PaymentMethod** with:
   * An **abstract method**: **ProcessPayment(decimal amount)**
   * A **concrete method**: **ShowPaymentMethod()** to print **"Payment Method: [Method]"**
2. Create derived classes:
   * **CreditCardPayment**:
     + Implements **ProcessPayment()** to print **"Processing credit card payment of ₹amount"**
   * **UPIPayment**:
     + Implements **ProcessPayment()** to print **"Processing UPI payment of ₹amount"**
3. In the **Main()** method:
   * Create objects of **CreditCardPayment** and **UPIPayment** using **PaymentMethod** reference.
   * Call **ShowPaymentMethod()** and **ProcessPayment()** for both objects.
   * Demonstrate **polymorphism** by calling the **ProcessPayment()** method using the **base class reference**.