Scenario 1: Employee Promotion System

Context:

A company wants to track employee details and promote them based on their performance.

Task 1: Create `Employee` Class (BLC - Business Logic Class)

- Attributes:
  - `empId` (int) → Employee ID
  - `name` (String) → Employee name
  - `designation` (String) → Current designation
  - `salary` (double) → Employee salary
  - `performanceRating` (int) → Rating out of 5
- Implement:
  1. **Parameterized constructor** to initialize all attributes.
  2. **Getter methods** (`getEmpId()`, `getName()`, etc.) to fetch details.
  3. **Setter methods** (`setDesignation()`, `setSalary()`) to modify details.
  4. **Business logic method:**
     - `promoteEmployee()` :
       - If `performanceRating >= 4`, increase salary by 20% and promote to next level.
       - Else, no promotion.

Task 2: Create `EmployeeProcessor` Class (ELC - Executable Logic Class)

- Create an employee object using the constructor.
- Print current details using getters.
- Call `promoteEmployee()` method.
- Print the updated designation and salary.

Conditions:

- If rating is **4 or above**, employee is promoted and salary increased.
- If rating is **below 4**, no changes are made.

Example Output:

Case 1: Employee Gets Promoted

```
Before Promotion:
Employee ID: 201
Name: Bob
Designation: Developer
Salary: $50000.0
Performance Rating: 5

After Promotion:
New Designation: Senior Developer
Updated Salary: $60000.0
```

Case 2: Employee Not Promoted

```
Before Promotion:
Employee ID: 202
Name: Charlie
Designation: Developer
Salary: $50000.0
Performance Rating: 3

No promotion. Performance rating is below the threshold.
```

Scenario 2: Bank Account Management

Context:

A bank needs to manage customer accounts, allowing deposits and withdrawals.

Task 1: Create `BankAccount` Class (BLC - Business Logic Class)

- Attributes:
  - `accountNumber` (int) → Account number
  - `accountHolder` (String) → Account holder's name
  - `balance` (double) → Current account balance
- Implement:
  1. **Parameterized constructor** to initialize all attributes.
  2. **Getter methods** (`getBalance()` , `getAccountHolder()` , etc.).
  3. **Setter method** (`setBalance()`) to modify the balance.
  4. **Business logic methods:**
     - `deposit(double amount)` : Increases balance by `amount` .
     - `withdraw(double amount)` :
       - Deducts `amount` if **balance >= amount**.
       - Else, prints `"Insufficient balance"` and does not withdraw.

Task 2: Create `BankProcessor` Class (ELC - Executable Logic Class)

- Create a bank account object using the constructor.
- Print current balance.
- Perform deposit and withdrawal operations.
- Print updated balance after transactions.

Conditions:

- Withdrawal should **only be allowed if sufficient balance is available**.
- Deposit increases balance without restrictions.

Example Output:

Case 1: Successful Deposit and Withdrawal

```
Initial Balance: $5000.0

Depositing $2000...
New Balance: $7000.0
```

```
Withdrawing $3000...
New Balance: $4000.0
```

Case 2: Insufficient Balance

```
Initial Balance: $5000.0

Withdrawing $6000...
Insufficient balance. Withdrawal failed.
Balance remains: $5000.0
```

Scenario 3: Online Shopping Cart

Context:

An e-commerce website needs a system to manage product orders in a shopping cart.

Task 1: Create `Product` Class (BLC - Business Logic Class)

- Attributes:
  - `productId` (int) → Product ID
  - `productName` (String) → Name of the product
  - `price` (double) → Price of a single unit
  - `quantity` (int) → Available stock
- Implement:
  1. **Parameterized constructor** to initialize attributes.
  2. **Getter methods** (`getProductId()`, `getPrice()`, etc.).
  3. **Setter method** (`setQuantity()`) to modify stock.
  4. **Business logic method:**
     - `purchaseProduct(int purchaseQty)`:
       - Reduces stock by `purchaseQty` if **stock is available**.
       - Else, prints `"Not enough stock"` and does not process purchase.

Task 2: Create `CartProcessor` Class (ELC - Executable Logic Class)

- Create a product object using the constructor.
- Print available stock.
- Attempt to purchase an item.
- Print updated stock after purchase.

Conditions:

- Purchase should **only proceed if enough stock is available**.

Example Output:

Case 1: Purchase Successful

```
Product: Laptop
Price: $1200.0
Available Stock: 5
```

```
Purchasing 2 units...

Updated Stock: 3
```

Case 2: Purchase Fails Due to Low Stock

```
Product: Laptop
Price: $1200.0
Available Stock: 5

Purchasing 7 units...

Not enough stock available!
```

Scenario 4: Student Grading System

Context:

A school wants to store student records and assign grades based on their marks.

Task 1: Create `Student` Class (BLC - Business Logic Class)

- Attributes:
  - `studentId` (int) → Student ID
  - `name` (String) → Student's name
  - `marks` (double) → Marks obtained
- Implement:
  1. **Parameterized constructor** to initialize attributes.
  2. **Getter methods** (`getStudentId()` , `getName()` , `getMarks()`).
  3. **Setter method** (`setMarks()`) to modify marks.
  4. **Business logic method:**
     - `calculateGrade()` :
       - `marks >= 90` → Grade A
       - `marks >= 80` → Grade B
       - `marks >= 70` → Grade C
       - `marks < 70` → Grade D

Task 2: Create `StudentProcessor` Class (ELC - Executable Logic Class)

- Create a student object using the constructor.
- Print student details and grade.
- Modify marks and check the updated grade.

Example Output:

```
Student: Alex
Marks: 85
Grade: B
```

```
Updating Marks to 95...

New Grade: A
```
