**Banking System**

**Problem Statement:** Exception Handling in a Simple Banking System

Detailed Problem Statement for Case Study 1: Exception Handling in a Simple Banking System

**Objective**

As part of a training assignment, you are tasked with developing a console-based banking system in C# that demonstrates robust exception handling. The system will simulate basic banking operations, such as depositing and withdrawing money from a bank account, while ensuring errors are caught and handled gracefully to provide a user-friendly experience. This assignment focuses on applying exception handling concepts, including standard .NET exceptions, to enforce business rules and prevent invalid operations.

**Background**

In a real-world banking application, operations like deposits and withdrawals must adhere to strict rules to ensure data integrity and user trust. For example, users should not be allowed to deposit negative amounts or withdraw more than their available balance. This assignment simulates such a system at a beginner to intermediate level, emphasizing proper exception handling to manage invalid inputs and operational errors.

**Requirements**

You are required to create a C# console application that implements a simple banking system with the following components and functionality:

**1. BankAccount Class:**

- Create a `BankAccount` class to represent a bank account.

- Properties:

- `AccountHolder` (string): The name of the account holder (read-only after initialization).

- `Balance` (decimal): The current balance of the account (read-only to external classes, modifiable only via methods).

- Constructor:

- Accepts the account holder's name and an initial balance.

- Validates that the initial balance is non-negative.

- Methods:

- `Deposit(decimal amount)`: Adds the specified amount to the balance.

- `Withdraw(decimal amount)`: Subtracts the specified amount from the balance.

**2. Exception Handling Scenarios:**

- Invalid Initial Balance:

- If the initial balance provided during account creation is negative, throw an `ArgumentException` with a message like "Initial balance cannot be negative."

- Invalid Deposit Amount:

- If the deposit amount is zero or negative, throw an `ArgumentException` with a message like "Deposit amount must be positive."

- Invalid Withdrawal Amount:

- If the withdrawal amount is zero or negative, throw an `ArgumentException` with a message like "Withdrawal amount must be positive."

- Insufficient Balance:

- If the withdrawal amount exceeds the current balance, throw an `InvalidOperationException` with a message like "Insufficient balance."

- General Exceptions:

- Include a catch-all for unexpected errors using the base `Exception` class to ensure the application does not crash.

**3. Main Program:**

- In the `Main` method, create a `BankAccount` instance with an initial balance.

- Demonstrate the following test cases in a try-catch block:

- A valid deposit (e.g., deposit $500).

- An invalid deposit (e.g., deposit -$100).

- A valid withdrawal (e.g., withdraw $300).

- An invalid withdrawal (e.g., withdraw $2000 when the balance is insufficient).

- Display the account balance after each successful operation.

- Handle each exception type separately and display user-friendly error messages.

**Constraints**

- Use the `decimal` type for all monetary values to avoid floating-point precision issues.

- Ensure the `Balance` property is not directly modifiable outside the `BankAccount` class (use private setters or equivalent).

- Do not allow the account balance to go below zero under any circumstances.

- Use meaningful exception messages that clearly describe the error to the end user.

- The application must not crash on invalid inputs; all errors must be caught and handled.

- Keep the code simple and modular, suitable for beginner to intermediate learners.

**Expected Deliverables**

1. Source Code:

- A C# console application with a `BankAccount` class and a `Program` class containing the `Main` method.

- The code should be well-organized, with clear comments explaining the purpose of each class, method, and exception-handling block.

2. Console Output:

- Display the initial balance upon account creation.

- Show the balance after each successful operation (deposit or withdrawal).

- Display appropriate error messages for each exception scenario.

- Example output:

```

Initial Balance: $1000.00

After Deposit: $1500.00

Argument Error: Deposit amount must be positive.

After Withdrawal: $1200.00

Operation Error: Insufficient balance.

```

**Example Workflow**

1. Create a new `BankAccount` with an initial balance of $1000.

2. Attempt to deposit $500 (valid, balance becomes $1500).

3. Attempt to deposit -$100 (invalid, should throw `ArgumentException`).

4. Attempt to withdraw $300 (valid, balance becomes $1200).

5. Attempt to withdraw $2000 (invalid, should throw `InvalidOperationException`).

6. Catch and display appropriate error messages for each invalid operation.