Lab Exercise: Mastering Exception Handling in C#

Objective:

Understand and practice the fundamentals of exception handling in C#, explore advanced concepts, and learn how to create and handle custom exceptions.

Prerequisites:

- Basic knowledge of C# programming
- Understanding of classes and inheritance in C#
- Visual Studio or any C# IDE installed

Exercise Sections:

Part 1: Basic Exception Handling

1. **Try-Catch Basics**
   - Create a simple console application that divides two numbers.
   - Enclose the division operation within a try block and catch any `DivideByZeroException`.
   - Print a friendly error message in the catch block.
2. **Using Finally**
   - Extend the above application to include a finally block.
   - Use the finally block to print a message indicating that the try-catch block is complete.
3. **Catching Multiple Exceptions**
   - Modify your division operation to include an array access operation (e.g., accessing an element by index).
   - Introduce a possible `IndexOutOfRangeException` by accessing an invalid index.
   - Use multiple catch blocks to handle both `DivideByZeroException` and `IndexOutOfRangeException` individually.

Part 2: Advanced Exception Handling

1. **Nested Try-Catch Blocks**
   - Create a method that uses a nested try-catch block. Inside the outer try block, attempt to parse a string to an integer using `int.Parse` and catch a `FormatException`.
   - Inside the inner try block, perform a division operation that could throw a `DivideByZeroException`.
2. **Exception Propagation**
   - Write a function that throws an `ArithmeticException` and call it from another function. Do not catch the exception in the called function.

- In the calling function, use a try-catch block to handle the propagated exception.

3. **Using `throw` and `throw ex`**
   - Demonstrate the difference between `throw` and `throw ex` within a catch block by rethrowing an exception and observing the stack trace in both cases.

Part 3: Custom Exceptions

1. **Creating a Custom Exception**
   - Define a custom exception named `InvalidInputException` that inherits from `ApplicationException`.
   - Add a constructor that allows the passing of an error message to the base class.
2. **Using Custom Exceptions**
   - Modify the division application to validate input before performing the division.
   - If the input is invalid (e.g., non-numeric), throw an `InvalidInputException`.
   - Use a try-catch block to catch the `InvalidInputException` and print a friendly error message.
3. **Advanced Custom Exception Usage**
   - Enhance `InvalidInputException` by adding additional properties such as `InputValue` and `ReasonForInvalidity`.
   - Catch the exception in your main program, and use these properties to provide detailed feedback to the user.

Deliverables:

- Source code for each part of the exercise.
- A short report detailing what you have learned about exception handling in C#, including how and when to use custom exceptions.

Assessment Criteria:

- Correct implementation of try-catch-finally blocks.
- Proper use of custom exceptions, including passing messages and additional data.
- Ability to handle multiple exceptions and explain the use of nested try-catch blocks.
- Understanding of exception propagation and the differences between `throw` and `throw ex`.

This exercise should help solidify your understanding of exception handling in C# by providing hands-on experience with a variety of scenarios ranging from basic to complex.