Jahanvi Varshney

Assignment 5

Q1: Create a C# program that intentionally throws a DivideByZeroException when dividing by zero. Catch the exception and handle it gracefully.

```
using System;


class Program
{
   static void Main()
   {
      try
      {
         int numerator = 10;
         int denominator = 0;


         int result = DivideNumbers(numerator, denominator);
         Console.WriteLine("Result: " + result);
      }
      catch (DivideByZeroException ex)
      {
         Console.WriteLine("Error: " + ex.Message);
      }
   }


   static int DivideNumbers(int numerator, int denominator)
   {
      if (denominator == 0)
      {
         throw new DivideByZeroException("Division by zero is not allowed.");
```

```
        }
        return numerator / denominator;
    }
}
```

Write a program that attempts to access an array element at an index that is out of bounds. Use a try-catch block to handle the IndexOutOfRangeException.

```
using System;


class Program
{
    static void Main()
    {
        int[] numbers = { 1, 2, 3, 4, 5 };
        int index = 7; // Attempt to access an out-of-bounds index


        try
        {
            int element = numbers[index];
            Console.WriteLine("Element at index " + index + ": " + element);
        }
        catch (IndexOutOfRangeException ex)
        {
            Console.WriteLine("Error: " + ex.Message);
        }
    }
}
```

Create a C# program that uses a try-catch block to handle an exception when converting a string to an integer using int.Parse(). Handle the FormatException that may occur.

```
using System;


class Program
{
   static void Main()
   {
      string input = "123abc"; // This string cannot be parsed to an integer


      try
      {
         int number = int.Parse(input);
         Console.WriteLine("Parsed integer: " + number);
      }
      catch (FormatException ex)
      {
         Console.WriteLine("Error: " + ex.Message);
      }
   }
}
```

4. Implement a C# program that uses a custom exception class. Create a custom exception and throw it in your code when a specific condition is met.

```
using System;
```

```
// Custom exception class
class MyCustomException : Exception
{
    public MyCustomException(string message) : base(message)
    {
    }
}

class Program
{
    static void Main()
    {
        int age = 15;

        try
        {
            if (age < 18)
            {
                throw new MyCustomException("Age must be 18 or older to perform this action.");
            }

            Console.WriteLine("Action performed successfully!");
        }
        catch (MyCustomException ex)
        {
            Console.WriteLine("Error: " + ex.Message);
        }
    }
}
```

Build a C# program that demonstrates the use of multiple catch blocks for different exception types. Handle exceptions such as IndexOutOfRangeException, FormatException, and InvalidOperationException.

```
using System;


class Program
{
   static void Main()
   {
      try
      {
         // Simulate different exceptions
         // Uncomment one of the following lines to trigger the specific exception


         // IndexOutOfRangeException
         //int[] numbers = { 1, 2, 3 };
         //int value = numbers[3]; // Access an out-of-bounds index


         // FormatException
         //string input = "abc123";
         //int number = int.Parse(input); // Attempt to parse a non-numeric string


         // InvalidOperationException
         //string operation = null;
         //if (operation.Length == 0) // Attempt to access a property of a null string
         //    Console.WriteLine("This won't be executed");
      }
      catch (IndexOutOfRangeException ex)
```

```
        {
            Console.WriteLine("IndexOutOfRangeException: " + ex.Message);
        }
        catch (FormatException ex)
        {
            Console.WriteLine("FormatException: " + ex.Message);
        }
        catch (InvalidOperationException ex)
        {
            Console.WriteLine("InvalidOperationException: " + ex.Message);
        }
        catch (Exception ex)
        {
            Console.WriteLine("General Exception: " + ex.Message);
        }
        finally
        {
            Console.WriteLine("Finally block executed.");
        }
    }
}
```

6. Create a C# program that includes nested try-catch blocks. Throw an exception in an inner block and catch it in the outer block. Explain the flow of execution.

```
using System;

class Program
{
    static void Main()
```

```
    {
        try
        {
            Console.WriteLine("Outer Try Block: Start");
            try
            {
                Console.WriteLine("Inner Try Block: Start");

                // Simulate an exception in the inner block
                int result = 10 / 0; // This will throw a DivideByZeroException

                Console.WriteLine("Inner Try Block: End");
            }
            catch (DivideByZeroException innerException)
            {
                Console.WriteLine("Inner Catch Block: Caught Exception - " + innerException.Message);
            }

            Console.WriteLine("Outer Try Block: End");
        }
        catch (Exception outerException)
        {
            Console.WriteLine("Outer Catch Block: Caught Exception - " + outerException.Message);
        }

        Console.WriteLine("After Nested Try-Catch Blocks");
    }
}
```

7. Implement a program that divides two numbers entered by the user. Handle exceptions like division by zero and invalid input. Continue to prompt the user for valid input until a valid division is performed.

```
using System;


class Program
{
   static void Main()
   {
      while (true)
      {
         try
         {
            Console.Write("Enter the numerator: ");
            int numerator = int.Parse(Console.ReadLine());

            Console.Write("Enter the denominator: ");
            int denominator = int.Parse(Console.ReadLine());

            if (denominator == 0)
            {
               Console.WriteLine("Error: Division by zero is not allowed. Please try again.");
               continue; // Continue to the next iteration of the while loop
            }

            int result = numerator / denominator;
            Console.WriteLine("Result: " + result);
            break; // Exit the loop if the division is successful
         }
```

```
        catch (FormatException)
        {
            Console.WriteLine("Error: Invalid input. Please enter valid numbers.");
        }
        catch (DivideByZeroException)
        {
            Console.WriteLine("Error: Division by zero is not allowed. Please try again.");
        }
        catch (Exception ex)
        {
            Console.WriteLine("An unexpected error occurred: " + ex.Message);
        }
    }
  }
}
```

8. Develop a C# program that demonstrates how to use the throw statement to rethrow an exception. Catch the rethrown exception and handle it appropriately.

```
using System;

class Program
{
  static void Main()
  {
    try
    {
      DivideByZero();
    }
```

```
    catch (DivideByZeroException ex)
    {
        Console.WriteLine("Caught original exception: " + ex.Message);

        // Rethrow the exception with additional information
        throw new Exception("Rethrowing the exception with additional information", ex);
    }
    catch (Exception ex)
    {
        Console.WriteLine("Caught rethrown exception: " + ex.Message);
    }
}

static void DivideByZero()
{
    int numerator = 10;
    int denominator = 0;

    try
    {
        int result = numerator / denominator; // This will throw a DivideByZeroException
    }
    catch (DivideByZeroException ex)
    {
        Console.WriteLine("Exception in DivideByZero method: " + ex.Message);

        // Rethrow the exception
        throw;
    }
```

```
    }
}
```

9. Develop a program that simulates a simple calculator with basic arithmetic operations (addition, subtraction, multiplication, and division). Use exception handling to catch and handle various type of exceptions that may occur.

```
using System;


class Calculator
{
    static void Main()
    {
        while (true)
        {
            Console.WriteLine("Simple Calculator");
            Console.WriteLine("1. Addition");
            Console.WriteLine("2. Subtraction");
            Console.WriteLine("3. Multiplication");
            Console.WriteLine("4. Division");
            Console.WriteLine("5. Quit");


            Console.Write("Select an operation (1-5): ");
            string choice = Console.ReadLine();


            if (choice == "5")
            {
                Console.WriteLine("Goodbye!");
                break;
            }
```

```
if (choice != "1" && choice != "2" && choice != "3" && choice != "4")
{
   Console.WriteLine("Invalid choice. Please select a valid operation (1-5).");
   continue;
}

try
{
   double result = 0;

   Console.Write("Enter the first number: ");
   double num1 = double.Parse(Console.ReadLine());

   Console.Write("Enter the second number: ");
   double num2 = double.Parse(Console.ReadLine());

   switch (choice)
   {
      case "1": // Addition
         result = num1 + num2;
         break;
      case "2": // Subtraction
         result = num1 - num2;
         break;
      case "3": // Multiplication
         result = num1 * num2;
         break;
      case "4": // Division
```

```
            if (num2 == 0)
            {
                Console.WriteLine("Error: Division by zero is not allowed.");
                continue;
            }
            result = num1 / num2;
            break;
        }

        Console.WriteLine("Result: " + result);
      }
      catch (FormatException)
      {
        Console.WriteLine("Error: Invalid input. Please enter valid numbers.");
      }
      catch (OverflowException)
      {
        Console.WriteLine("Error: Overflow occurred. The numbers are too large.");
      }
      catch (Exception ex)
      {
        Console.WriteLine("An unexpected error occurred: " + ex.Message);
      }
    }
  }
}
```

10. Scenario You are developing a simple e-commerce application in C#. One of the features is a shopping cart that allows users to add items to their cart. The cart is represented as an array of

integers, where each integer corresponds to an item's price. Users can input the price of an item they want to add to the cart. You want to handle exceptions gracefully to ensure a smooth user experience. If the user enters an invalid price, your code should catch and handle the exception appropriately. Question: Write a C# program that simulates adding items to a shopping cart. The program should take user input for the price of items and store them in an array. Implement exception handling with multiple catch blocks to handle various scenarios. Specifically, you should handle the following exceptions: • If the user enters a negative price, catch and handle the exception as a "NegativePriceException." Display a message indicating that the price entered is invalid. • If the user enters a non-numeric value (e.g., a string), catch and handle the exception as a "FormatException." Display a message indicating that the input is not a valid price. • If the user enters a price that exceeds a predefined maximum value (e.g., 10000), catch and handle the exception as a "PriceTooHighException." Display a message indicating that the price entered is too high. Note: The program should continue to prompt the user for prices until a valid price is entered. After each valid price is entered, add it to the shopping cart array. Once the user is done adding items, display the total price of the items in the cart. Ensure that the program uses multiple catch blocks to handle the specific exceptions mentioned above and provides informative error messages.

```
using System;

class NegativePriceException : Exception
{
    public NegativePriceException(string message) : base(message) { }
}

class PriceTooHighException : Exception
{
    public PriceTooHighException(string message) : base(message) { }
}

class Program
{
    static void Main()
    {
        int[] shoppingCart = new int[10];
```

```
int cartIndex = 0;

while (true)
{
   try
   {
      Console.Write("Enter the price of the item (or '0' to finish): ");
      string input = Console.ReadLine();

      if (input == "0")
      {
         break;
      }

      int price = int.Parse(input);

      if (price < 0)
      {
         throw new NegativePriceException("Error: Negative price entered. Please enter a valid price.");
      }
      else if (price > 10000)
      {
         throw new PriceTooHighException("Error: Price is too high. Please enter a valid price.");
      }

      shoppingCart[cartIndex] = price;
      cartIndex++;
```

```
            Console.WriteLine("Item added to the cart.");

        }
        catch (FormatException)
        {
            Console.WriteLine("Error: Invalid input. Please enter a valid price.");
        }
        catch (NegativePriceException ex)
        {
            Console.WriteLine(ex.Message);
        }
        catch (PriceTooHighException ex)
        {
            Console.WriteLine(ex.Message);
        }
        catch (Exception ex)
        {
            Console.WriteLine("An unexpected error occurred: " + ex.Message);
        }
    }

    int total = 0;
    for (int i = 0; i < cartIndex; i++)
    {
        total += shoppingCart[i];
    }

    Console.WriteLine("Cart total price: " + total);
}
```

}