14.0-Reading and writing to files

Program

```
using System;

using System.IO;

using System.Runtime.Serialization;

using System.Runtime.Serialization.Formatters.Binary;


class Program

{

    [Serializable]

    class Score

    {

        public int Value { get; set; }

    }


    static void Main()

    {

        Score score;


        try

        {

            using (FileStream fs = new FileStream("score.dat", FileMode.OpenOrCreate))

            {

                if (fs.Length > 0)

                {

                    BinaryFormatter formatter = new BinaryFormatter();
            score = (Score)formatter.Deserialize(fs);
            Console.WriteLine("Successfully deserialized score.");
        }
        else
        {
            score = new Score();
            Console.WriteLine("File is empty. Initializing new score.");
        }
    }
}
catch (SerializationException ex)
{
    Console.WriteLine($"SerializationException: {ex.Message}");
    score = new Score();
}
catch (FileNotFoundException)
{
    score = new Score();
    Console.WriteLine("File not found. Initializing new score.");
}

Console.WriteLine($"Current Value: {score.Value}");

score.Value++;

using (FileStream fs = new FileStream("score.dat", FileMode.Create))
{
```

```
            BinaryFormatter formatter = new BinaryFormatter();

            formatter.Serialize(fs, score);

            Console.WriteLine("Successfully serialized score.");

        }

    }

}
```

Exercise

Write a C# program that demonstrates a simple scoring system implemented using file serialization. It begins by defining a Score class with a single property Value to represent the score value. Within the Main method, the program attempts to open a file named "score.dat" for reading. If the file exists and is not empty, it deserializes the Score object from the file using a BinaryFormatter, effectively restoring the previous score. If the file does not exist or is empty, it initializes a new Score object with a default value of 0. Any encountered exceptions during file handling or deserialization are caught, with appropriate messages printed to the console, and a new Score object is created.

After loading or initializing the Score object, the program displays the current score value. It then increments the score by one and proceeds to serialize the updated Score object back into the "score.dat" file using a BinaryFormatter. Finally, it prints a success message to indicate that the score serialization process was completed.

Hint

The program opens or creates the file "score.dat" and reads from or writes to it using FileMode.OpenOrCreate and FileMode.Create modes, respectively.

Serialization: The BinaryFormatter class is utilized for both serialization and deserialization. It serializes the Score object to write it into the file and deserializes it to read the object from the file.

Exception Handling: The program employs exception handling to manage potential errors during file operations and serialization/deserialization. It catches SerializationException and FileNotFoundException to handle different error scenarios.

Console Output: The program provides informative messages via console output to indicate the status of file operations and serialization/deserialization processes. These messages help in understanding the flow of the program and any encountered issues.

Explanation

This program is a simple car information system implemented in C#. It allows users to interactively view a list of cars and search for specific cars based on their make and model. Here's how the program works:

The Car class defines the properties of a car, including its model, make, year, and color. The ToString() method is overridden to provide a formatted string representation of a car's information.

In the Main method, a list of Car objects is initialized with some sample car data. Then, an infinite while loop is used to continuously display a menu of options to the user and handle their input.

Within the loop, the user is prompted to enter their choice, and the program reads their input using Console.ReadLine(). The input is parsed into an integer using int.TryParse() to ensure it's a valid numeric choice.

A switch statement is used to handle different menu choices:

If the user selects option 1, all cars in the list are displayed by calling the ListAllCars() function.

If the user selects option 2, they are prompted to enter the make and model of the car they want to search for. The program then filters the list of cars using LINQ's Where method to find matching cars and displays them.

If the user selects option 3, the program exits gracefully with a goodbye message.

The ListAllCars() function iterates through the list of cars and prints each car's information to the console using Console.WriteLine().

The SearchCars() function prompts the user to enter the make and model of the car they want to search for. It then uses LINQ's Where method to filter the list of cars based on the user's input and displays the matching cars or a message if no matches are found.