Review Material 07

File Access Methods in C#

📌 Introduction

File handling is one of the most crucial concepts in any programming language. In C#, the `System.IO` namespace provides various classes for working with files and directories. File access methods are techniques used to perform operations such as creating, reading, writing, and updating files.

📂 Types of File Access Methods

C# supports several file access techniques:

1. **Sequential Access** – Reading and writing data from start to end.
2. **Random Access** – Jumping to specific positions in a file.
3. **Binary File Access** – Reading and writing binary data.
4. **Text File Access** – Reading and writing textual data line by line.

🛠️ Commonly Used Classes in `System.IO`

Class	Description
`File`	Static methods for file operations
`FileInfo`	Provides instance methods for file operations
`StreamReader` / `StreamWriter`	For reading/writing text
`BinaryReader` / `BinaryWriter`	For reading/writing binary
`FileStream`	Base class for file I/O

✏️ 1. Writing to a File (Text)

```
using System;
using System.IO;

class WriteTextFile {
    static void Main() {
        string filePath = "sample.txt";
        using (StreamWriter writer = new StreamWriter(filePath)) {
            writer.WriteLine("Hello, world!");
            writer.WriteLine("This is a C# file access example.");
        }
        Console.WriteLine("Text written to file.");
    }
}
```

🧠 Explanation:

- `StreamWriter` writes text line-by-line.
- `using` ensures the file is closed properly.

📖 2. Reading from a File (Text)

```
using System;
using System.IO;

class ReadTextFile {
    static void Main() {
        string filePath = "sample.txt";
        if (File.Exists(filePath)) {
            using (StreamReader reader = new StreamReader(filePath)) {
                string line;
                while ((line = reader.ReadLine()) != null) {
                    Console.WriteLine(line);
                }
            }
        } else {
            Console.WriteLine("File not found.");
        }
    }
}
```

🧠 Explanation:

- This C# console program reads and displays the contents of sample.txt line by line using a StreamReader. It first checks if the file exists and then safely reads it using a using block to ensure proper resource cleanup.

📋 3. Appending to a File

```
using System;
using System.IO;

class AppendToFile {
    static void Main() {
        string filePath = "sample.txt";
        using (StreamWriter writer = new StreamWriter(filePath, append: true)) {
            writer.WriteLine("This line is appended.");
        }
        Console.WriteLine("Line appended.");
    }
}
```

🧠 Explanation:

- This C# program appends a new line of text to `sample.txt` using a `StreamWriter` with the `append: true` parameter. The `using` block ensures the file is properly closed after writing.

💾 4. Binary File Access

Writing Binary:

```
using System;
using System.IO;

class WriteBinaryFile {
    static void Main() {
        string filePath = "data.bin";
```

```
        using (BinaryWriter writer = new BinaryWriter(File.Open(filePath,
FileMode.Create))) {
            writer.Write(123); // int
            writer.Write("C# Binary File"); // string
        }
        Console.WriteLine("Binary data written.");
    }
}
```

🧠 Explanation:

- This C# program writes binary data to a file named `data.bin` using a `BinaryWriter`. It writes an integer and a string in binary format, and the `using` block ensures the file stream is properly closed.

Reading Binary:

```
using System;
using System.IO;

class ReadBinaryFile {
    static void Main() {
        string filePath = "data.bin";
        using (BinaryReader reader = new BinaryReader(File.Open(filePath,
FileMode.Open))) {
            int num = reader.ReadInt32();
            string text = reader.ReadString();
            Console.WriteLine($"Number: {num}, Text: {text}");
        }
    }
}
```

🧠 Explanation:

- This C# program reads binary data from `data.bin` using a `BinaryReader`. It retrieves an integer and a string in the same order they were written and displays them, with the `using` block ensuring proper file handling.

🌀 5. Random Access (Seek)

```
using System;
using System.IO;

class RandomAccess {
    static void Main() {
        string filePath = "random.bin";
        using (FileStream fs = new FileStream(filePath, FileMode.OpenOrCreate)) {
            fs.Seek(10, SeekOrigin.Begin);
            byte[] data = System.Text.Encoding.UTF8.GetBytes("C#");
            fs.Write(data, 0, data.Length);
        }
        Console.WriteLine("Wrote at offset 10.");
    }
}
```

🧠 Explanation:

- This C# program demonstrates random file access by using `FileStream` to write data starting at byte offset 10 in `random.bin`. It uses `Seek` to move the file pointer and writes the string "C#" as bytes.

6. File and Directory Management

```
using System;
using System.IO;

class FileDirectoryDemo {
    static void Main() {
        string path = "exampleDir";
        if (!Directory.Exists(path)) {
            Directory.CreateDirectory(path);
            Console.WriteLine("Directory created.");
        }

        string filePath = Path.Combine(path, "info.txt");
        File.WriteAllText(filePath, "Sample directory file");
        Console.WriteLine("File created in directory.");
    }
}
```

Explanation:

- This C# program creates a directory named `exampleDir` if it doesn't exist, then creates a file `info.txt` inside it with sample text using `File.WriteAllText`. It demonstrates both directory and file operations.

---

7. Deleting and Checking Files

```
using System;
using System.IO;

class FileCheckDelete {
    static void Main() {
        string filePath = "sample.txt";
        if (File.Exists(filePath)) {
            File.Delete(filePath);
            Console.WriteLine("File deleted.");
        } else {
            Console.WriteLine("File does not exist.");
        }
    }
}
```

Explanation:

- This C# program checks if the file `sample.txt` exists. If it does, the file is deleted using `File.Delete()`. Otherwise, it prints a message indicating that the file doesn't exist.

Summary Table of File Access Methods

Operation	Method/Class Used	Description
Create file	`File.Create`, `FileStream`	Create a new file
Write text	`StreamWriter`	Write strings to a file
Read text	`StreamReader`	Read lines from a file
Append text	`StreamWriter (append: true)`	Add data to existing file

Operation	Method/Class Used	Description
Binary write	`BinaryWriter`	Write binary data
Binary read	`BinaryReader`	Read binary data
Seek/Random access	`FileStream.Seek()`	Access specific position
Check file	`File.Exists()`	Check if file exists
Delete file	`File.Delete()`	Remove a file
Directory ops	`Directory` class	Create, delete, list dirs

This table summarizes common file access methods in C#, listing key classes and methods used for creating, reading, writing, and managing text and binary files, as well as performing directory operations. It highlights both sequential and random access capabilities.

Best Practices in File Access

- Always close streams or use `using`.
- Handle exceptions (e.g., `FileNotFoundException`).
- Use `Path.Combine()` for portable paths.
- Validate file paths and sanitize user input.
- Backup important files before writing.

Common Assessment Questions

1. **Explain the difference between `StreamWriter` and `BinaryWriter`.**
2. **Write a C# program that reads a file line-by-line and counts the number of words.**
3. **Create a program that logs user input to a file and reads it back.**
4. **Demonstrate how to access a specific byte in a binary file.**
5. **What are the advantages of using `using` statements with file operations?**

Sample Coding Challenge

Problem:
Write a C# console application that:

- Prompts the user to input name and age.
- Saves the data to a binary file.
- Later, reads and displays the stored data.

Solution:

```
using System;
using System.IO;

class UserDataBinary {
    static void Main() {
        string path = "userdata.bin";

        // Writing
        using (BinaryWriter writer = new BinaryWriter(File.Open(path,
FileMode.Create))) {
            Console.Write("Enter Name: ");
            string name = Console.ReadLine();
            Console.Write("Enter Age: ");
            int age = Convert.ToInt32(Console.ReadLine());

            writer.Write(name);
            writer.Write(age);
        }

        // Reading
        using (BinaryReader reader = new BinaryReader(File.Open(path, FileMode.Open)))
{
            string name = reader.ReadString();
            int age = reader.ReadInt32();

            Console.WriteLine($"Name: {name}, Age: {age}");
        }
    }
}
```

🧠 Explanation:

- This C# program collects a user's name and age from console input, then saves the data in binary format to `userdata.bin` using a `BinaryWriter`. It then reads the same data back using a `BinaryReader` and displays it, demonstrating both writing and reading structured binary data.

🧠 Final Tips for Assessment

- Understand the difference between text and binary files.
- Practice both reading and writing operations.
- Know how to use `FileStream` for advanced control.
- Use `Path`, `File`, and `Directory` classes for general file management.
- Practice try-catch blocks for robust applications.

📚 References

- Microsoft Docs - System.IO
- C# Programming Yellow Book by Rob Miles
- Exam and Lab guides for C# Fundamentals

Review Questions

Q1. Which namespace is primarily used for file handling in C#?
A.) System.Text
B.) System.IO
C.) System.Data
D.) System.Net

Q2. What class would you typically use to read text files line by line?
A.) BinaryReader
B.) FileStream
C.) StreamReader
D.) FileInfo

Q3. Which method is used to check if a file exists?
A.) File.Delete()
B.) File.Check()
C.) File.Exists()
D.) File.Verify()

Q4. What does `StreamWriter` do when used with `append: true`?
A.) Deletes the file before writing
B.) Appends new text to the end of the file
C.) Overwrites the file
D.) Creates a binary file

Q5. Which class is used for writing binary data to a file?
A.) FileWriter
B.) BinaryWriter
C.) DataStreamWriter
D.) StreamWriter

Q6. Which `FileMode` value is used to open a file or create it if it doesn't exist?
A.) FileMode.Create
B.) FileMode.Append
C.) FileMode.Open
D.) FileMode.OpenOrCreate

Q7. Which method allows you to jump to a specific position in a file stream?
A.) Seek()
B.) Move()
C.) Jump()
D.) Navigate()

Q8. What will `StreamReader.ReadLine()` return at the end of a file?
A.) null
B.) EOF
C.) 0
D.) ""

Q9. Which method is used to delete a file in C#?
A.) File.Remove()
B.) File.Destroy()
C.) File.Erase()
D.) File.Delete()

Q10. Which statement ensures that file resources are automatically closed and disposed?
A.) lock
B.) using
C.) try-catch
D.) static

Answers: https://github.com/clydeatmcm/IT104/blob/main/ICT-Proficiency-Test/CSharp/07-AnswerKeys.txt