**1. FILE HANDLING IN C#**

**1.1 Introduction**

File handling is a mechanism in C# that allows developers to perform various operations on files such as creating, reading, writing, appending, copying, and deleting files. The System.IO namespace provides several classes that enable easy and efficient file manipulation.

C# supports both **binary** and **text-based** file operations. File handling is important when data must be stored persistently on disk instead of being held temporarily in memory.

**1.2 Important Classes in System.IO Namespace**

|  |  |
| --- | --- |
| **Class Name** | **Description** |
| **File** | Provides static methods to create, copy, delete, move, and open files. |
| **FileInfo** | Provides instance methods for file operations. Similar to File but allows working with object-oriented features. |
| **StreamReader** | Reads characters from a text file sequentially. |
| **StreamWriter** | Writes text data to a file. |
| **BinaryReader** | Reads primitive data types as binary values. |
| **BinaryWriter** | Writes primitive data types in binary format. |
| **Directory** | Provides static methods for creating, moving, and deleting directories. |
| **DirectoryInfo** | Similar to Directory, but provides object-based access. |
| **Path** | Provides methods to manipulate file or directory path information. |

**1.3 Basic File Operations**

1. **Creating a File**

To create a new file, use the File.Create() or File.WriteAllText() method.

File.WriteAllText("sample.txt", "Welcome to Dhruv Training");

1. **Reading from a File**

The File.ReadAllText() or File.ReadAllLines() methods are used to read data from a file.

string content = File.ReadAllText("sample.txt");

Console.WriteLine(content);

1. **Appending to a File**

To add content to an existing file without overwriting it, use the File.AppendAllText() method.

File.AppendAllText("sample.txt", "\nThis is an additional line.");

1. **Deleting a File**

To delete a file, use the File.Delete() method.

File.Delete("sample.txt");

**1.4 Using StreamReader and StreamWriter**

For reading or writing text files line by line, the StreamReader and StreamWriter classes are used.

Example:

using (StreamWriter sw = new StreamWriter("data.txt"))

{

sw.WriteLine("Hello Udaya");

sw.WriteLine("Welcome to .NET Training");

}

using (StreamReader sr = new StreamReader("data.txt"))

{

string line;

while ((line = sr.ReadLine()) != null)

{

Console.WriteLine(line);

}

}

The using block ensures that resources are automatically released after the operation.

**1.5 Exception Handling in File Operations**

File operations often generate exceptions such as file not found, access denied, or path not valid. Common exceptions include:

* FileNotFoundException
* IOException
* UnauthorizedAccessException
* DirectoryNotFoundException

It is good practice to use **try-catch-finally** blocks to handle such exceptions.

**2. LINQ (LANGUAGE INTEGRATED QUERY)**

**2.1 Introduction**

LINQ (Language Integrated Query) is a feature introduced in C# 3.0 that provides a uniform syntax for querying data from different sources like arrays, collections, XML, or databases. It enables developers to write SQL-like queries directly within C# code.

Namespace:

using System.Linq;

**2.2 Advantages of LINQ**

1. Unified approach to querying different data sources.
2. Strongly typed – errors are caught at compile time.
3. Readable and concise code.
4. Supports deferred execution.
5. Eliminates the need for complex loops and conditions.

**2.3 LINQ Syntax Types**

LINQ provides two syntaxes:

1. **Query Syntax**  
   Similar to SQL:
2. var result = from n in numbers
3. where n > 50
4. orderby n descending
5. select n;
6. **Method Syntax**  
   Uses lambda expressions and extension methods:
7. var result = numbers.Where(n => n > 50).OrderByDescending(n => n);

**2.4 Common LINQ Methods**

|  |  |
| --- | --- |
| **Method** | **Description** |
| **Where()** | Filters a sequence based on a condition. |
| **Select()** | Projects elements into a new form. |
| **OrderBy()** | Sorts elements in ascending order. |
| **OrderByDescending()** | Sorts elements in descending order. |
| **GroupBy()** | Groups elements that share a common attribute. |
| **Sum()**, **Count()**, **Average()**, **Max()**, **Min()** | Performs aggregate calculations. |
| **First()**, **Last()**, **Single()** | Retrieves specific elements. |

**2.5 LINQ with Collections Example**

Example using a list of integers:

List<int> nums = new List<int>() {10, 20, 30, 40, 50};

var result = from n in nums

where n > 30

select n;

foreach (var n in result)

Console.WriteLine(n);

**2.6 LINQ with Objects Example**

List<Student> students = new List<Student>()

{

new Student(){Name="Udaya", Marks=85},

new Student(){Name="Arun", Marks=45},

new Student(){Name="Shetty", Marks=75}

};

var passed = students.Where(s => s.Marks >= 50)

.OrderByDescending(s => s.Marks);

foreach (var s in passed)

Console.WriteLine($"{s.Name} - {s.Marks}");

**2.7 LINQ Grouping and Aggregation Example**

var group = students.GroupBy(s => s.Marks >= 50 ? "Pass" : "Fail");

foreach (var g in group)

{

Console.WriteLine(g.Key + ":");

foreach (var student in g)

Console.WriteLine(" " + student.Name);

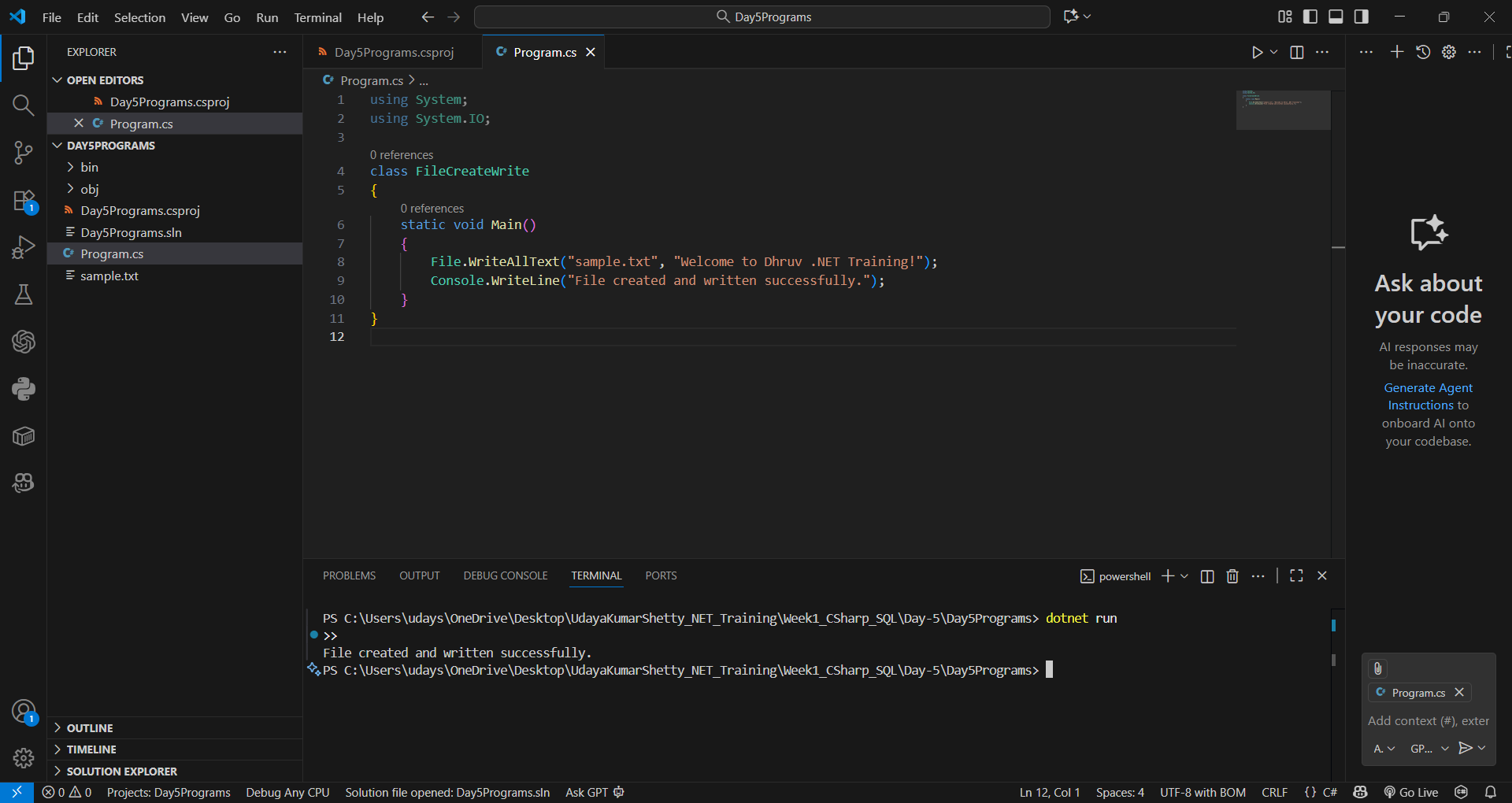
}

**SUMMARY**

|  |  |
| --- | --- |
| **Concept** | **Description** |
| **File Handling** | Reading, writing, and managing files. |
| **StreamReader/Writer** | Sequential text file reading/writing. |
| **LINQ** | Query syntax integrated with C# for data filtering and sorting. |

**Snapshots:**

1. **Basic File Handling**
   * **Create and Write**



* + **Read**

A screenshot of a computer

AI-generated content may be incorrect.

* + **Append**

A screenshot of a computer

AI-generated content may be incorrect.

* + **Delete**

A screenshot of a computer program

AI-generated content may be incorrect.

1. **StreamReader and StreamWriter**

A screenshot of a computer program

AI-generated content may be incorrect.

1. **LINQ Basics – List of Numbers**

A screenshot of a computer program

AI-generated content may be incorrect.

1. **LINQ with Objects**

A screenshot of a computer program

AI-generated content may be incorrect.

1. **LINQ GroupBy and Aggregation**

A screen shot of a computer

AI-generated content may be incorrect.