File handling and Multithreading (C#)

File refers to a collection of records.

1. Writing Data to a File

```
using System;
using System.IO;

class Program
{
    static void Main()
    {
        string filePath = "example.txt"; // File path where data will be written

        // Data to write into the file
        string data = "Hello, this is a simple file handling example using File.Creat

        try
        {
            // Creating a new file or overwriting the existing one
            using (FileStream fs = File.Create(filePath))
            {
                // Create a StreamWriter object to write to the file
                using (StreamWriter writer = new StreamWriter(fs))
                {
                    writer.WriteLine(data); // Write the data to the file
                }
            }

            Console.WriteLine("Data successfully written to the file.");
        }
        catch (Exception ex)
```

```
        {
            Console.WriteLine("An error occurred: " + ex.Message);
        }
    }
}
```

2. Reading data

```
using System;
using System.IO;

class Program
{
    static void Main()
    {
        string filePath = "example.txt"; // File path to read data from

        try
        {
            // Check if the file exists
            if (File.Exists(filePath))
            {
                // Using StreamReader to read the file content
                using (StreamReader reader = new StreamReader(filePath))
                {
                    string fileContent = reader.ReadToEnd(); // Reads all content from
                    Console.WriteLine("File Content:");
                    Console.WriteLine(fileContent);
                }
            }
            else
            {
                Console.WriteLine("The file does not exist.");
            }
        }
        catch (Exception ex)
```

```
        {
            Console.WriteLine("An error occurred: " + ex.Message);
        }
    }
}
```

#Multithreading

A T**hread** is a basic unit of execution in a program. It's like a "task" or "sequence of instructions" that the computer can run independently.

Multithreading is the ability of a program to run multiple threads simultaneously, allowing different tasks to be executed at the same time. This helps improve performance, especially on multi-core processors.

→ Creating a Thread Program in C#.

(i) Thread class :- Create a new thread by instantiating the Thread class.

(ii) ThreadStart Delegate :- Specify the method to be executed by the thread using the ThreadStart delegate.

```
using System;
using System.Threading;

class Program
{
    // Method to be executed by the first thread
    static void Task1()
    {
        Console.WriteLine("Task 1 is starting.");
        Thread.Sleep(2000); // Simulate work by sleeping for 2 seconds
        Console.WriteLine("Task 1 is completed.");
    }
```

```
    // Method to be executed by the second thread
    static void Task2()
    {
        Console.WriteLine("Task 2 is starting.");
        Thread.Sleep(1000); // Simulate work by sleeping for 1 second
        Console.WriteLine("Task 2 is completed.");
    }

    static void Main()
    {
        // Creating the first thread to execute Task1
        Thread thread1 = new Thread(Task1);

        // Creating the second thread to execute Task2
        Thread thread2 = new Thread(Task2);

        // Starting both threads
        thread1.Start();
        thread2.Start();

        // Wait for thread1 to complete before proceeding with the main thread
        thread1.Join();
        Console.WriteLine("Main thread is waiting for thread1 to complete.");

        // Wait for thread2 to complete before proceeding with the main thread
        thread2.Join();
        Console.WriteLine("Main thread is waiting for thread2 to complete.");

        // Main thread message after both threads are finished
        Console.WriteLine("Both threads have completed. Main thread exiting.");
    }
}
```

→Thread Life-Cycle

1. New :- The thread is created and in a “new” state.

2. Runnable/Running :- The thread is ready to be executed or is currently executing.
3. Blocked :- The thread is temporarily paused and waiting for a resource or event.
4. Terminated