**TAMILARASI\_T\_P\_01\_ASSIGN\_04**

**Controller**

**using FileFragmentation.Models;**

**using FileFragmentation.Views;**

**using System;**

**using System.IO;**

**namespace FileFragmentation.Controllers**

**{**

**public class FileController**

**{**

**private readonly FileModel \_model;**

**private readonly FileView \_view;**

**public FileController(FileModel model, FileView view)**

**{**

**\_model = model;**

**\_view = view;**

**}**

**public void Start()**

**{**

**try**

**{**

**// Step 1: Create the input file with user-input paragraph**

**\_view.ShowMessage("Enter a paragraph to save in the file:");**

**string paragraph = \_view.GetInput();**

**\_model.WriteParagraphToFile(paragraph);**

**\_view.ShowMessage("File 'input.txt' created and paragraph written successfully.");**

**// Step 2: Fragment the input file**

**\_view.ShowMessage("Enter the number of characters per fragment:");**

**int fragmentSize = int.Parse(\_view.GetInput());**

**\_model.FragmentFile(fragmentSize);**

**int fileCount = Directory.GetFiles(\_model.FragmentPath).Length;**

**\_view.ShowMessage($"File fragmentation completed. {fileCount} fragment files created.");**

**// Step 3: Verify the existence and content of a specific fragment file**

**\_view.ShowMessage("Enter the fragment file name to verify (e.g., 001.txt):");**

**string fileName = \_view.GetInput();**

**if (\_model.FileExists(fileName))**

**{**

**string content = \_model.ReadFile(fileName);**

**\_view.ShowMessage($"File '{fileName}' exists. Content:\n{content}");**

**}**

**else**

**{**

**\_view.ShowMessage($"File '{fileName}' does not exist.");**

**}**

**// Step 4: Clean up by deleting fragment files**

**\_model.DeleteFragmentFiles();**

**\_view.ShowMessage("Cleanup completed. All fragment files have been deleted.");**

**}**

**catch (Exception ex)**

**{**

**\_view.ShowErrorMessage(ex.Message);**

**}**

**finally**

**{**

**// Pause to prevent console from closing**

**\_view.ShowMessage("Press any key to exit...");**

**Console.ReadKey(); // This line ensures the program waits for a key press before exiting**

**}**

**}**

**}**

**}**

**View**

**using System;**

**namespace FileFragmentation.Views**

**{**

**public class FileView**

**{**

**public void ShowMessage(string message)**

**{**

**Console.WriteLine(message);**

**}**

**public string GetInput()**

**{**

**return Console.ReadLine();**

**}**

**public void ShowErrorMessage(string errorMessage)**

**{**

**Console.WriteLine($"Error: {errorMessage}");**

**}**

**}**

**}**

**Model**

**using System;**

**using System.IO;**

**using System.Linq;**

**namespace FileFragmentation.Models**

**{**

**public class FileModel**

**{**

**public string InputFilePath { get; set; } = "input.txt";**

**public string FragmentPath { get; set; } = "fragments";**

**// Step 1: Write the user-input paragraph to the input.txt file**

**public void WriteParagraphToFile(string paragraph)**

**{**

**File.WriteAllText(InputFilePath, paragraph);**

**}**

**// Step 2: Fragment the input file based on user-defined size**

**public void FragmentFile(int fragmentSize)**

**{**

**Directory.CreateDirectory(FragmentPath);**

**string content = File.ReadAllText(InputFilePath);**

**int fragmentNumber = 1;**

**for (int i = 0; i < content.Length; i += fragmentSize)**

**{**

**string fragmentContent = new string(content.Skip(i).Take(fragmentSize).ToArray());**

**File.WriteAllText(Path.Combine(FragmentPath, $"{fragmentNumber:D3}.txt"), fragmentContent);**

**fragmentNumber++;**

**}**

**}**

**// Step 3: Check if the specific fragment file exists**

**public bool FileExists(string fileName)**

**{**

**return File.Exists(Path.Combine(FragmentPath, fileName));**

**}**

**// Step 4: Read the content of a specific fragment file**

**public string ReadFile(string fileName)**

**{**

**return File.ReadAllText(Path.Combine(FragmentPath, fileName));**

**}**

**// Step 5: Clean up by deleting fragment files**

**public void DeleteFragmentFiles()**

**{**

**DirectoryInfo directory = new DirectoryInfo(FragmentPath);**

**foreach (FileInfo file in directory.GetFiles())**

**{**

**file.Delete();**

**}**

**}**

**}**

**}**

**Program.cs**

**using FileFragmentation.Controllers;**

**using FileFragmentation.Models;**

**using FileFragmentation.Views;**

**namespace FileFragmentation**

**{**

**class Program**

**{**

**static void Main(string[] args)**

**{**

**var model = new FileModel();**

**var view = new FileView();**

**var controller = new FileController(model, view);**

**controller.Start();**

**}**

**}**

**}**

#### **1. Model: FileModel.cs**

csharp

Copy code

using System;  
using System.IO;  
using System.Linq;  
  
namespace FileFragmentation.Models  
{  
 public class FileModel  
 {  
 public string InputFilePath { get; set; } = "input.txt";  
 public string OutputFilePath { get; set; } = "output.txt";  
 public string FragmentPath { get; set; } = "fragments";  
  
 public void WriteParagraphToFile(string paragraph)  
 {  
 File.WriteAllText(InputFilePath, paragraph);  
 }  
  
 public void FragmentFile(int fragmentSize)  
 {  
 Directory.CreateDirectory(FragmentPath);  
  
 string content = File.ReadAllText(InputFilePath);  
 int fragmentNumber = 1;  
  
 for (int i = 0; i < content.Length; i += fragmentSize)  
 {  
 string fragmentContent = new string(content.Skip(i).Take(fragmentSize).ToArray());  
 string fileName = $"{fragmentNumber:D3}.txt";  
 File.WriteAllText(Path.Combine(FragmentPath, fileName), fragmentContent);  
 Console.WriteLine($"Created {fileName} with content: {fragmentContent}"); // Display the fragment file name and content  
 fragmentNumber++;  
 }  
 }  
  
 public bool FileExists(string fileName)  
 {  
 return File.Exists(Path.Combine(FragmentPath, fileName));  
 }  
  
 public string ReadFile(string fileName)  
 {  
 return File.ReadAllText(Path.Combine(FragmentPath, fileName));  
 }  
  
 public void DefragmentFiles()  
 {  
 var files = Directory.GetFiles(FragmentPath).OrderBy(f => f);  
 string content = string.Join("", files.Select(f => File.ReadAllText(f)));  
 File.WriteAllText(OutputFilePath, content);  
 }  
  
 public bool CompareInputAndOutput()  
 {  
 string inputContent = File.ReadAllText(InputFilePath);  
 string outputContent = File.ReadAllText(OutputFilePath);  
 return inputContent == outputContent;  
 }  
  
 public void DeleteFragmentFiles()  
 {  
 DirectoryInfo directory = new DirectoryInfo(FragmentPath);  
 foreach (FileInfo file in directory.GetFiles())  
 {  
 file.Delete();  
 }  
 }  
 }  
}

#### **2. Controller: FileController.cs**

csharp

Copy code

using FileFragmentation.Models;  
using FileFragmentation.Views;  
using System;  
using System.IO;  
using System.Linq;  
  
namespace FileFragmentation.Controllers  
{  
 public class FileController  
 {  
 private readonly FileModel \_model;  
 private readonly FileView \_view;  
  
 public FileController(FileModel model, FileView view)  
 {  
 \_model = model;  
 \_view = view;  
 }  
  
 public void Start()  
 {  
 try  
 {  
 \_view.ShowMessage("Enter the paragraph to save in the file:");  
 string paragraph = \_view.GetInput();  
 \_model.WriteParagraphToFile(paragraph);  
 \_view.ShowMessage($"File '{\_model.InputFilePath}' created and paragraph written successfully.");  
  
 \_view.ShowMessage("Enter the number of characters per fragment:");  
 int fragmentSize = int.Parse(\_view.GetInput());  
  
 \_model.FragmentFile(fragmentSize);  
 \_view.ShowFragmentationCompleted();  
  
 \_view.ShowMessage("Enter the fragment file name to verify (e.g., 001.txt):");  
 string fileName = \_view.GetInput();  
 if (\_model.FileExists(fileName))  
 {  
 \_view.ShowMessage($"File '{fileName}' exists. Content:\n{\_model.ReadFile(fileName)}");  
 }  
 else  
 {  
 \_view.ShowMessage($"File '{fileName}' does not exist.");  
 }  
  
 \_model.DefragmentFiles();  
 \_view.ShowMessage("Files successfully defragmented into 'output.txt'.");  
 \_view.ShowMessage($"Content of '{\_model.OutputFilePath}':\n{File.ReadAllText(\_model.OutputFilePath)}");  
  
 if (\_model.CompareInputAndOutput())  
 {  
 \_view.ShowMessage("Comparison successful: The files match.");  
 }  
 else  
 {  
 \_view.ShowMessage("Error: The input and output files do not match.");  
 }  
  
 \_model.DeleteFragmentFiles();  
 \_view.ShowMessage("Cleanup completed. All fragmented files have been deleted.");  
  
 \_view.ShowMessage("Press any key to exit...");  
 Console.ReadKey(); // Prevents the program from exiting until a key is pressed  
 }  
 catch (Exception ex)  
 {  
 \_view.ShowErrorMessage(ex.Message);  
 }  
 }  
 }  
}

#### **3. View: FileView.cs**

csharp

Copy code

using System;  
  
namespace FileFragmentation.Views  
{  
 public class FileView  
 {  
 public void ShowMessage(string message)  
 {  
 Console.WriteLine(message);  
 }  
  
 public string GetInput()  
 {  
 return Console.ReadLine();  
 }  
  
 public void ShowFragmentationCompleted()  
 {  
 Console.WriteLine("\nFile fragmentation completed.");  
 Console.WriteLine("Fragmented files:");  
 foreach (string file in System.IO.Directory.GetFiles("fragments"))  
 {  
 Console.WriteLine(System.IO.Path.GetFileName(file));  
 }  
 }  
  
 public void ShowErrorMessage(string errorMessage)  
 {  
 Console.WriteLine($"Error: {errorMessage}");  
 }  
 }  
}

#### **4. Program: Program.cs**

csharp

Copy code

using FileFragmentation.Controllers;  
using FileFragmentation.Models;  
using FileFragmentation.Views;  
  
namespace FileFragmentation  
{  
 class Program  
 {  
 static void Main(string[] args)  
 {  
 var model = new FileModel();  
 var view = new FileView();  
 var controller = new FileController(model, view);  
  
 controller.Start();  
 }  
 }  
}

### **Expected Console Session**

plaintext

Copy code

Enter the paragraph to save in the file:  
Hello, this is a test paragraph. It will be fragmented into smaller pieces.  
  
File 'input.txt' created and paragraph written successfully.  
  
Enter the number of characters per fragment:  
10  
  
Created 001.txt with content: Hello, thi  
Created 002.txt with content: s is a tes  
Created 003.txt with content: t paragraph  
Created 004.txt with content: . It will b  
Created 005.txt with content: e fragmented  
Created 006.txt with content: into smaller  
Created 007.txt with content: pieces.  
  
File fragmentation completed.  
Fragmented files:  
001.txt  
002.txt  
003.txt  
004.txt  
005.txt  
006.txt  
007.txt  
  
Enter the fragment file name to verify (e.g., 001.txt):  
001.txt  
  
File '001.txt' exists. Content:  
Hello, thi  
  
Files successfully defragmented into 'output.txt'.  
Content of 'output.txt':  
Hello, this is a test paragraph. It will be fragmented into smaller pieces.  
  
Comparison successful: The files match.  
Cleanup completed. All fragmented files have been deleted.  
  
Press any key to exit...

### **Explanation of Changes:**

* **File Fragmentation**: Each fragment's file name and content is printed to the console as it's created (matching the expected session).
* **File Verification**: The user is prompted to check a specific file, and its content is displayed if it exists.
* **Exit Control**: The program waits for the user to press any key before exiting, ensuring the console doesn't close immediately.

This code should now fully match your requirements!

**EXPLANATION**

Here’s a line-by-line explanation of the code:

### **1. FileModel.cs: Handles the core file operations (creating files, fragmenting, defragmenting, etc.)**

csharp

Copy code

using System;  
using System.IO;  
using System.Linq;

* Imports necessary namespaces: System for basic operations, System.IO for file handling, and System.Linq for querying arrays or collections.

csharp

Copy code

namespace FileFragmentation.Models  
{  
 public class FileModel  
 {  
 public string InputFilePath { get; set; } = "input.txt";  
 public string OutputFilePath { get; set; } = "output.txt";  
 public string FragmentPath { get; set; } = "fragments";

* Defines the FileModel class inside the FileFragmentation.Models namespace.
* InputFilePath, OutputFilePath, and FragmentPath are properties that specify file paths for input, output, and fragmented files respectively, with default values.

csharp

Copy code

public void WriteParagraphToFile(string paragraph)  
{  
 File.WriteAllText(InputFilePath, paragraph);  
}

* **WriteParagraphToFile**: Takes a string (paragraph) and writes it to the input.txt file using File.WriteAllText, which overwrites any existing content.

csharp

Copy code

public void FragmentFile(int fragmentSize)  
{  
 Directory.CreateDirectory(FragmentPath);

* **FragmentFile**: Takes an integer fragmentSize and splits the input.txt file into smaller pieces of that size.
* Directory.CreateDirectory(FragmentPath) ensures the directory for fragments exists (creates it if it doesn’t).

csharp

Copy code

string content = File.ReadAllText(InputFilePath);  
int fragmentNumber = 1;

* Reads the entire content of input.txt into a string content.
* Initializes fragmentNumber to 1, which is used to name each fragment file.

csharp

Copy code

for (int i = 0; i < content.Length; i += fragmentSize)  
{  
 string fragmentContent = new string(content.Skip(i).Take(fragmentSize).ToArray());  
 string fileName = $"{fragmentNumber:D3}.txt";  
 File.WriteAllText(Path.Combine(FragmentPath, fileName), fragmentContent);  
 Console.WriteLine($"Created {fileName} with content: {fragmentContent}");  
 fragmentNumber++;  
}

* Loops through the content string, breaking it into fragments of the specified fragmentSize.
* **content.Skip(i).Take(fragmentSize)** takes a substring starting at position i with length fragmentSize.
* The fragment is saved to a file named 001.txt, 002.txt, etc., using zero-padded filenames.
* Each file is written using File.WriteAllText, and the fragment’s content is printed to the console.
* fragmentNumber++ increments the file number for the next fragment.

csharp

Copy code

public bool FileExists(string fileName)  
{  
 return File.Exists(Path.Combine(FragmentPath, fileName));  
}

* **FileExists**: Checks whether a specific fragment file (e.g., 001.txt) exists by combining the FragmentPath and fileName.

csharp

Copy code

public string ReadFile(string fileName)  
{  
 return File.ReadAllText(Path.Combine(FragmentPath, fileName));  
}

* **ReadFile**: Reads the content of a specified fragment file and returns it as a string.

csharp

Copy code

public void DefragmentFiles()  
{  
 var files = Directory.GetFiles(FragmentPath).OrderBy(f => f);  
 string content = string.Join("", files.Select(f => File.ReadAllText(f)));  
 File.WriteAllText(OutputFilePath, content);  
}

* **DefragmentFiles**: Reads all the fragment files, orders them by filename (to ensure correct sequence), and combines their content into one string.
* File.WriteAllText writes the combined content to the output file (output.txt).

csharp

Copy code

public bool CompareInputAndOutput()  
{  
 string inputContent = File.ReadAllText(InputFilePath);  
 string outputContent = File.ReadAllText(OutputFilePath);  
 return inputContent == outputContent;  
}

* **CompareInputAndOutput**: Compares the original input file (input.txt) with the defragmented output file (output.txt), returning true if they match.

csharp

Copy code

public void DeleteFragmentFiles()  
{  
 DirectoryInfo directory = new DirectoryInfo(FragmentPath);  
 foreach (FileInfo file in directory.GetFiles())  
 {  
 file.Delete();  
 }  
}

* **DeleteFragmentFiles**: Deletes all fragment files in the FragmentPath directory by iterating over them and calling file.Delete().

### **2. FileController.cs: Handles program flow and interaction between the model and view.**

csharp

Copy code

using FileFragmentation.Models;  
using FileFragmentation.Views;  
using System;  
using System.IO;  
using System.Linq;

* Imports necessary namespaces and references FileModel and FileView from their respective namespaces.

csharp

Copy code

namespace FileFragmentation.Controllers  
{  
 public class FileController  
 {  
 private readonly FileModel \_model;  
 private readonly FileView \_view;

* The FileController class is part of the FileFragmentation.Controllers namespace.
* \_model and \_view are private fields used to interact with the FileModel (data logic) and FileView (user interface).

csharp

Copy code

public FileController(FileModel model, FileView view)  
{  
 \_model = model;  
 \_view = view;  
}

* The constructor initializes the \_model and \_view fields with the provided FileModel and FileView instances.

csharp

Copy code

public void Start()  
{  
 try  
 {  
 \_view.ShowMessage("Enter the paragraph to save in the file:");  
 string paragraph = \_view.GetInput();  
 \_model.WriteParagraphToFile(paragraph);  
 \_view.ShowMessage($"File '{\_model.InputFilePath}' created and paragraph written successfully.");

* **Start Method**: The main method that runs the program logic.
* Prompts the user for a paragraph, writes it to input.txt using the model's WriteParagraphToFile method, and informs the user that the file was created.

csharp

Copy code

\_view.ShowMessage("Enter the number of characters per fragment:");  
int fragmentSize = int.Parse(\_view.GetInput());  
\_model.FragmentFile(fragmentSize);  
\_view.ShowFragmentationCompleted();

* Asks the user for the number of characters per fragment, then calls FragmentFile to split the input file accordingly.
* The method ShowFragmentationCompleted is used to display the list of generated fragment files.

csharp

Copy code

\_view.ShowMessage("Enter the fragment file name to verify (e.g., 001.txt):");  
string fileName = \_view.GetInput();  
if (\_model.FileExists(fileName))  
{  
 \_view.ShowMessage($"File '{fileName}' exists. Content:\n{\_model.ReadFile(fileName)}");  
}  
else  
{  
 \_view.ShowMessage($"File '{fileName}' does not exist.");  
}

* Prompts the user to enter a specific fragment file name to verify. If the file exists, its content is displayed; otherwise, a message informs the user that the file doesn't exist.

csharp

Copy code

\_model.DefragmentFiles();  
\_view.ShowMessage("Files successfully defragmented into 'output.txt'.");  
\_view.ShowMessage($"Content of '{\_model.OutputFilePath}':\n{File.ReadAllText(\_model.OutputFilePath)}");

* Calls DefragmentFiles to combine all the fragments back into the output.txt file.
* Displays the content of the defragmented file (output.txt).

csharp

Copy code

if (\_model.CompareInputAndOutput())  
{  
 \_view.ShowMessage("Comparison successful: The files match.");  
}  
else  
{  
 \_view.ShowMessage("Error: The input and output files do not match.");  
}

* Compares the input and output files using CompareInputAndOutput. If they match, the user is informed; otherwise, an error message is shown.

csharp

Copy code

\_model.DeleteFragmentFiles();  
\_view.ShowMessage("Cleanup completed. All fragmented files have been deleted.");

* Deletes all fragment files and displays a cleanup completion message.

csharp

Copy code

\_view.ShowMessage("Press any key to exit...");  
Console.ReadKey();

* Waits for the user to press any key before exiting the program.

csharp

Copy code

catch (Exception ex)  
{  
 \_view.ShowErrorMessage(ex.Message);  
}  
}  
}

* Catches any exceptions during program execution and displays the error message using ShowErrorMessage.

### **3. FileView.cs: Handles user interaction and output.**

csharp

Copy code

using System;

* Imports the System namespace for basic functionality like input/output.

csharp

Copy code

namespace FileFragmentation.Views  
{  
 public class FileView  
 {  
 public void ShowMessage(string message)  
 {  
 Console.WriteLine(message);  
 }

* **ShowMessage**: A simple method to display a message to the user in the console using Console.WriteLine.

csharp

Copy code

public string GetInput()  
{  
 return Console.ReadLine();  
}

* **GetInput**: Reads input from the user and returns it as a string.

csharp

Copy code

public void ShowFragmentationCompleted()  
{  
 Console.WriteLine("\nFile fragmentation completed.");  
 Console.WriteLine("Fragmented files:");  
 foreach (string file in System.IO.Directory.GetFiles("fragments"))  
 {  
 Console.WriteLine(System.IO.Path.GetFileName(file));  
 }  
}

* **ShowFragmentationCompleted**: Displays a message indicating that fragmentation is complete, followed by the names of the fragmented files.

csharp

Copy code

public void ShowErrorMessage(string errorMessage)  
{  
 Console.WriteLine($"Error: {errorMessage}");  
}

* **ShowErrorMessage**: Displays any error messages encountered during program execution.

### **4. Program.cs: Entry point of the program.**

csharp

Copy code

using FileFragmentation.Controllers;  
using FileFragmentation.Models;  
using FileFragmentation.Views;  
  
namespace FileFragmentation  
{  
 class Program  
 {  
 static void Main(string[] args)  
 {  
 var model = new FileModel();  
 var view = new FileView();  
 var controller = new FileController(model, view);

* Imports necessary namespaces for controllers, models, and views.
* In the Main method, creates instances of FileModel, FileView, and FileController to initiate the program.

csharp

Copy code

controller.Start();

* Calls the Start method in the FileController to run the entire program flow.