Contents

TextFileHelper Class

Description

Purpose

This project is designed to teach you how to manipulate files and exceptions. It reinforces your understanding of class input/output operations on files and forces you to be creative with a limited toolbox.

Challenge

In short Develop a static class to perform simple operations on files: display its content, display its content without the comments, reverse its content, concatenate two files, and finally (for more experienced programmers), determine if the content of the file is balanced (as defined below).

In more details Your goal is to design and implement a static `TextFileHelper` class containing 5 methods (the fifth one being optional):

1. A `Display` method that takes a `string` as argument and displays the content of the file located at the corresponding path if it exists, an error message otherwise.
2. A `DisplayNoComment` method that takes a `string` and acts as the `Display` method, except that every line starting with the # character is ignored.
3. A `Concat` method that takes three `string` arguments, `arg1`, `arg2` and `arg3` and write at `arg3` a file containing the text stored in `arg1` followed by the text stored in `arg2`, if both files exist and if `arg3` does not exist, and that displays an error message otherwise.

4. A `Reverse` method that takes two `string` arguments, `arg1` and `arg2`, and write at `arg2` the content of the file located at `arg1`, reversed (that is: last line of `arg1` is first of `arg2`, etc., but the content of the lines themselves is identical). If `arg1` does not exists or if a file is already present at `arg2`, then the method should display an error message.
5. (optional) A `Balanced` method that takes a `string` argument and returns **`true`** if the file located at the corresponding path is balanced, **`false`** otherwise. A file is *balanced* if every `(` character has a matching `)` character coming after it. We give below some examples of balanced and un-balanced files.

⚠ Warning

You are allowed to use only `StreamReader`'s `ReadLine` and constructor, `StreamWriter`'s `WriteLine` and constructor, and the usual `Console`, `char`, `string` and `array` methods, lists, arrays, conditionals, etc. In particular, *do not* use the Linq library. `AppDomain`, `Path` and `File` methods can be used as in class or in the example below.

Submission

Feel free to download the project illustrated below to help you getting started. You will need to either write the required methods or to comment out the code between "// START commenting here" and "// END commenting here" before you can compile it.

Please, follow our guideline on project submission. In particular, make sure you write your name and the date in a delimited comment at the beginning of your file.

Example

Here is an example of a `Program.cs` `Main` method, that creates four test files and illustrates some of the desired functionalities:

```
using System;
using System.Collections.Generic;
using System.IO;

class Program
{
  static void Main()
  {
    /*
     * We first create three
```

```
 * string variables where
 * our three demo files
 * will be created.
 */
string directoryPath = AppDomain
  .CurrentDomain
  .BaseDirectory;
string ex0 = Path.Combine(directoryPath, "ex0.txt");
string ex1 = Path.Combine(directoryPath, "ex1.txt");
string ex2 = Path.Combine(directoryPath, "ex2.txt");
// We now call the "Init" method to create those
 ↪  files.
TextFileHelper.Init(ex0, ex1, ex2);

/*
 * The following is for demo purpose.
 * Feel free edit it as you see fit.
 */
// WARNING This code cannot be compiled unless the
 ↪  following is commented out.
// START commenting here
Console.WriteLine("*****\n* Testing Display\n*****");
TextFileHelper.Display(ex0);
Console.WriteLine(
  "*****\n* Testing DisplayNoComment\n*****"
);
TextFileHelper.DisplayNoComment(ex0);
Console.WriteLine("*****\n* Testing Reverse\n*****");
string ex3 = Path.Combine(directoryPath, "ex3.txt");
TextFileHelper.Reverse(ex0, ex3);
TextFileHelper.Display(ex3);
Console.WriteLine("*****\n* Testing Concat\n*****");
string ex4 = Path.Combine(directoryPath, "ex4.txt");
TextFileHelper.Concat(ex0, ex1, ex4);
TextFileHelper.Display(ex4);

Console.WriteLine("*****\n* Testing Balanced\n*****");
bool wb0 = TextFileHelper.Balanced(ex0);
bool wb1 = TextFileHelper.Balanced(ex1);
bool wb2 = TextFileHelper.Balanced(ex2);
bool wb3 = TextFileHelper.Balanced(ex3);

Console.WriteLine("ex0 is balanced: " + wb0);
Console.WriteLine("ex1 is balanced: " + wb1);
Console.WriteLine("ex2 is balanced: " + wb2);
Console.WriteLine("ex3 is balanced: " + wb3);
```

```
  }
}

static class TextFileHelper
{
  // This method creates three files at the paths
  // ex0P, ex1P and ex2P
  // that will serve for our tests.
  public static void Init(
    string ex0P,
    string ex1P,
    string ex2P,
    bool over = false
  )
  {
    if (
      !over
      && (
        File.Exists(ex0P)
        || File.Exists(ex1P)
        || File.Exists(ex0P)
      )
    )
    {
      Console.WriteLine(
        "One of the path already contains a file."
          + "\nCall the method with an additional
           ↪  parameter"
          + "\nset to \"true\" to override the existing
           ↪  file."
      );
    }
    else
    {
      try
      {
        StreamWriter ex0 = new StreamWriter(ex0P);
        ex0.Write(
          "This is the first demo file\n# This line starts
           ↪  with an \"#\").\n(An isolated parenthesis"
        );
        ex0.Close();
      }
      catch
      {
        Console.WriteLine(
          "There was an error creating ex0."
        );
      }
      try
      {
        StreamWriter ex1 = new StreamWriter(ex1P);
        ex1.Write(
          "(This is the second demo file"
           + "\n)There are parenthesis in it, but no pound
             ↪  (\"#\") character on the first line."
        );
        ex1.Close();
      }
      catch
      {
        Console.WriteLine(
          "There was an error creating ex1."
        );
      }
      try
      {
        StreamWriter ex2 = new StreamWriter(ex2P);
        ex2.Write(
          "# Just a comment )"
            + "\nand unbalanced parentheses ("
        );
        ex2.Close();
      }
      catch
      {
        Console.WriteLine(
          "There was an error creating ex2."
        );
      }
    }
  }
}
```

(Download this code)

Executing it with the properly implemented `TextFileHelper` missing methods should give something along the lines of:

```
*****
* Testing Display
*****
This is the first demo file
```

```
## This line starts with an "#").
(An isolated parenthesis
*****
* Testing DisplayNoComment
*****
This is the first demo file
(An isolated parenthesis
*****
* Testing Reverse
*****
(An isolated parenthesis
## This line starts with an "#").
This is the first demo file
*****
* Testing Concat
*****
This is the first demo file
## This line starts with an "#").
(An isolated parenthesis
(This is the second demo file
)There are parenthesis in it, but no pound ("#")
 ↪  character on the first line.
*****
* Testing Balanced
*****
ex0 is balanced: False
ex1 is balanced: True
ex2 is balanced: False
ex3 is balanced: True
```

Note that it is ok if you cannot reproduce this output *exactly*.

Bonuses

- Write the `Balanced` method.
- Handle gracefully exceptions if files do not exist,
- Give to `Concat` and `Reverse` an optional `bool` argument that allows overriding the file whose path is given as a last argument.

Submission

Please, follow our guideline on project submission. In particular, make sure you write your name and the date in a delimited comment at the beginning of your file.

Solution

A possible solution is shared in this archive