CMPE 1666- Intermediate Programming

Lecture 4–

Working With Data Files

Using Data Files

- Programs can write data into files stored on disks. This has the benefits that:
  - The data outlives the process that has written it.
  - The data can be used by many programs, not only the one that has written it.

Using the File class – For Writing to a file

- The File class provides for an easy way of handling text files.
- To use the File class you must add the line below to the program

  using System.IO;

- For writing into the file, we can use the following static methods:
  - **File.WriteAllLines(<filename>,<StringArray or string collection>)** – Creates a new file, writes the specified string array to the file, then closes it.
    - Eg. File.WriteAllLines("file1.txt",myStringArray);
    - File.WriteAllLines("file1.txt",myStringList);

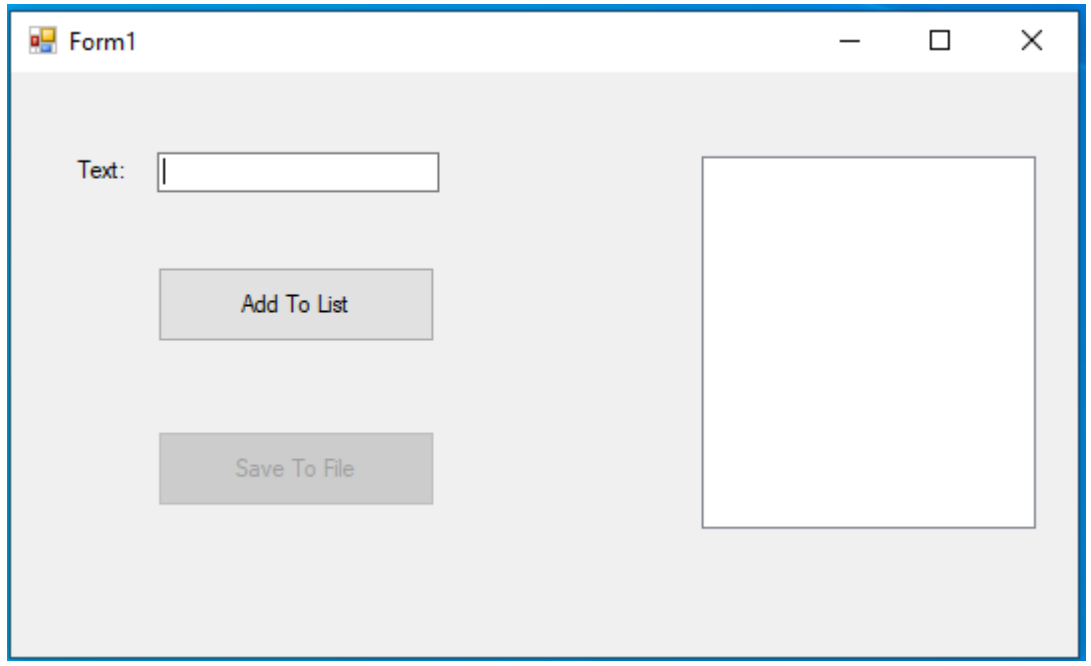
    Note that if no path name is provided, the file is created in the bin/Debug folder.
  - **File.WriteAllText(<filename>,<string>)-** Creates a new file, writes the specified string, then closes it.
    - Usage is similar to the above

Using the File class- For Writing to a file

- By default, new files will be created in the bin/Debug folder of our project.
- However, we can specify a path name together with the file name, to create the file in a folder of our choice.
- E.g. To create the file in our main project folder, we use the
  - File.WriteAllLines("../../file1.txt",myStringList);
- We can even go 2 levels higher to make the created file accessible to all our projects
  - File.WriteAllLines("../../../../file1.txt",myStringList);

Lecture4- Demo1

- Create an application with the UI shown below, then follow the instructions in the next slide.
- Use the following names for the controls:
  - Textbox: **UI_Input_Tbx**
  - Listbox: **UI_Display_Lbx**
  - Buttons: **UI_AddToList_Btn, UI_Save_Btn**



Lecture4- Demo1-contd

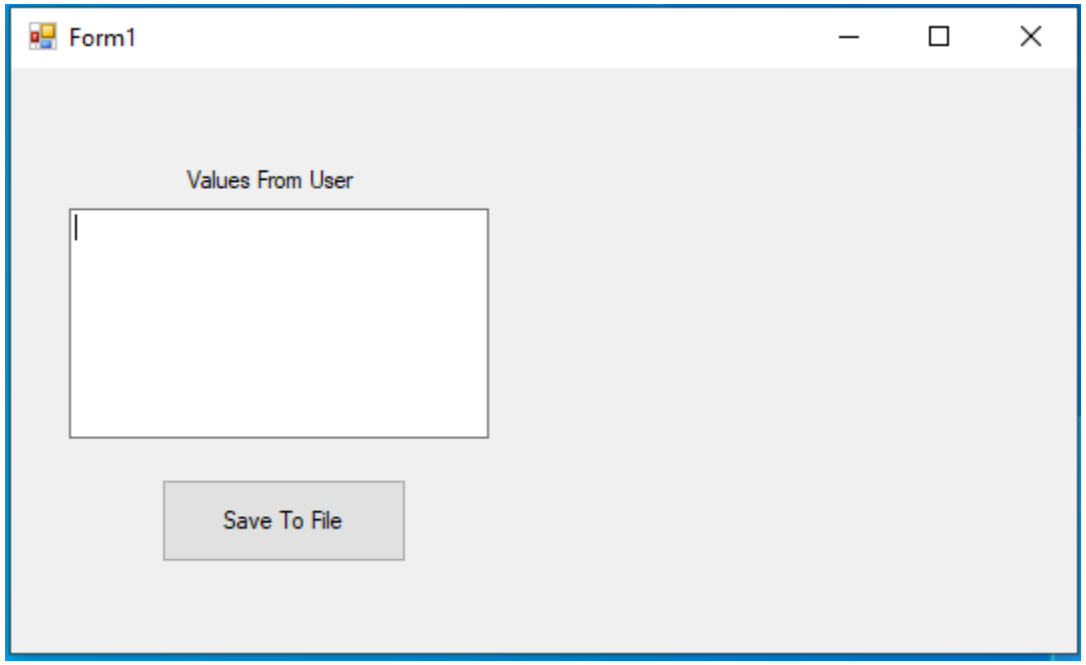
- In the application, create a string List variable called **textList** initialized to a new List, at the class level.
- Initially, the "Add **To List**" button must be enabled and the "**Save To File**" button disabled.
- Write an event listener for the "Add To List" button, such that, each time we click on it,
  - if the textbox is not empty, it takes the value from the textbox and adds it to textList
  - It also adds the value to the listbox
  - It clears the textbox
  - If we have already added 10 values to the list, the "**Add To List**" button is disabled and the "**Save To File**" Button is enabled.
- Write an event listener for the "**Save To File**" button, such that when we click on it, the data from the list is written to a file called "file1.txt" (in the default folder).
- In the event listener, include a Message Box to inform the user that the file has been saved.
- Close your program and open the file in notepad or any text editor to verify that the data has been written

Lecture4- Demo1b

- Modify the program from Lecture4-Demo1, such that when you click on the button "**Save To File**", the program also creates a file with path "../../../../file1.txt" and writes the list into it. (Note- you may want to change the text on the button to "**Save To Files**")
- This file will be created in the root folder of your projects.
- Open the file using a text editor and verify its contents.

Lecture4Demo2

- Create an application with the UI shown below.



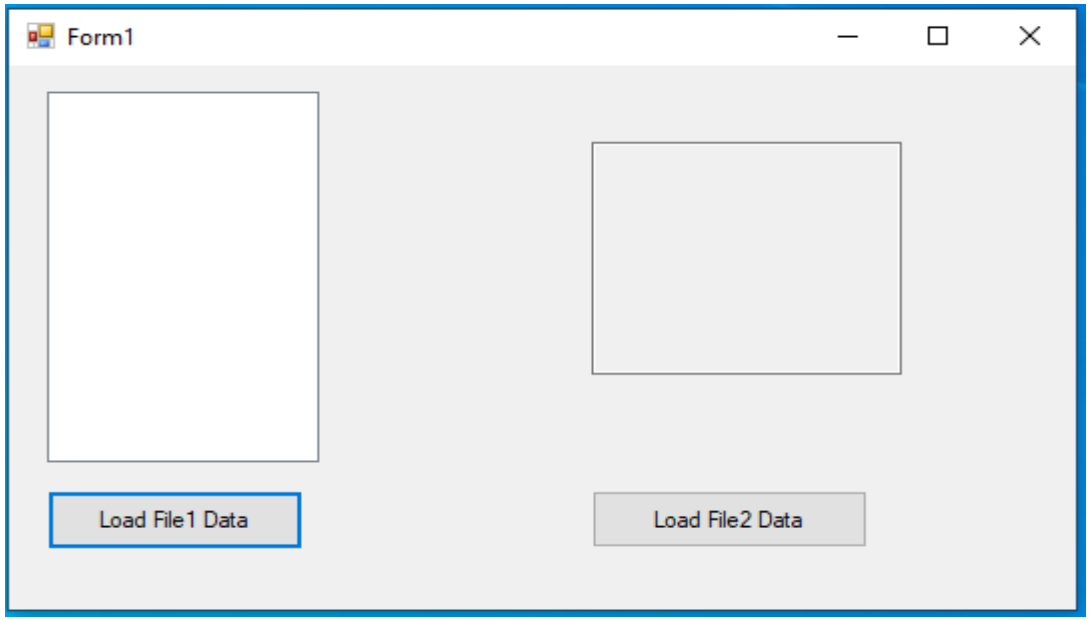
- The user can input a number of words in the multi-line textbox. When the user clicks on the button, the whole input from the textbox must be written to the file "../../../../file2.txt" using the **File.WriteAllText()** method.

Using the File class – For Reading from a File

- For reading from a file, we can use
  - <stringArray>=File.ReadAllLines(<filename>) –To read all the lines from a file and return them as an array of strings
    - E.g. string[] myArray=File.ReadAllLines(“file1.txt”)
  - The program can then iterate through the array of strings to work with the data.
  - <string>= File.ReadAllText(<filename>) – reads all the data from the file and returns it as one string.
    - string line=File.ReadAllText(“file1.txt”)

Lecture4- Demo 3

- In this application you will read the data from the files **file1** and **file2** that you created in the projects root folder in Demo1b and Demo2.
- Create an application with the UI shown below. Name the buttons UI_LoadFile1_Btn and UI_LoadFile2_Btn respectively. Name the listbox UI_DisplayFile1_Lbx and name the multi-line textbox UI_DisplayFile2_Tbx.



- Write the event handlers for the click events of the buttons, such that the one for the “Load File1 Data” button reads the data from file1 and displays it in the listbox, while the one for the “Load File2 Data” button reads the data from file2 and displays it in the multiline textbox.

Using Drag And Drop

- We can use the file explorer to drag and drop a file on any control.
- We'll work through Lecture4-Demo4 to study the Drag And Drop feature.
- For the control on which we are dropping the file, we'll have to set the "**AllowDrop**" property to true.
- As the items are dragged over the window, the **DragEnter** event handler will permit an icon change to show the user that the operation is permitted.
  - This means that the user will visually get a clue that the control will accept what is being dragged.
- We are interested only in files, we can implement the **DragEnter** handler as below:

```
private void [control]DragEnter(object sender, DragEventArgs e) {
        if (e.Data.GetDataPresent(DataFormats.FileDrop))
                e.Effect = DragDropEffects.Copy;
       else
                e.Effect = DragDropEffects.None;
        }
```

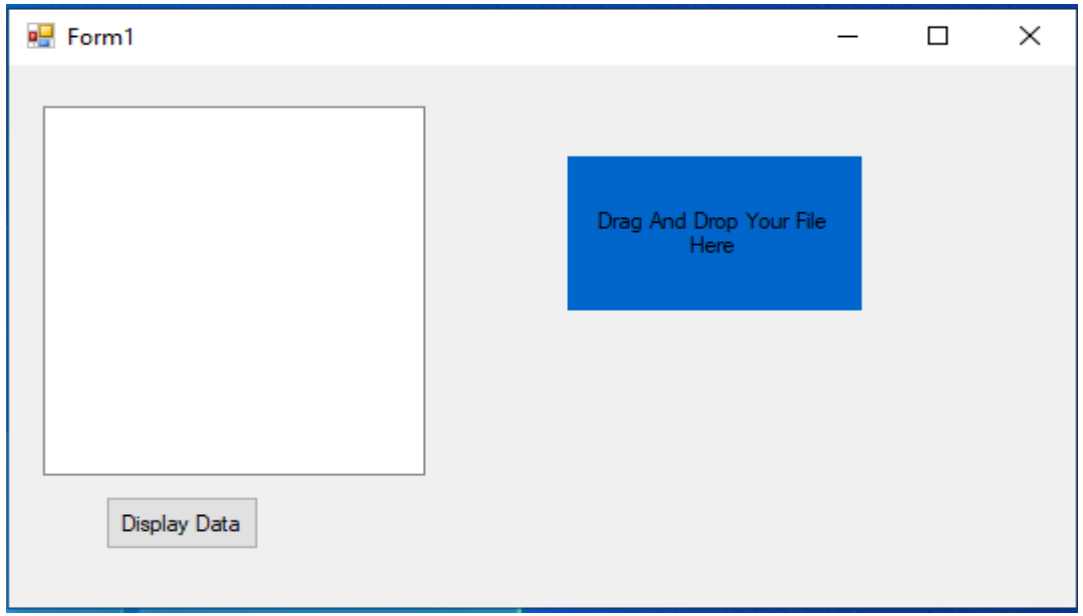
Using Drag And Drop

- Providing a **DragDrop** handler permits inspection of the drop data formats, and processing of the items dropped on the control.
- Types that are not expected should be ignored.
- Because the user may drop multiple items, you need to manage the collection of drop items, manage associated errors (not shown), and process the items as required:

```
private void [control]DragDrop(object sender, DragEventArgs e) {
    string fname =
((string[])e.Data.GetData(DataFormats.FileDrop)).First();
 …
```

Lecture4-Demo4

- Create an application with the UI shown below.
- Name the controls UI_DisplayData_Lbx, UI_DisplayData_Btn and UI_DragAndDrop_Lbl.
- Make the **BackColor** property of the label **HotTrack**, and set the **TextAlign** property to **MiddleCenter**
- Declare a class-level member variable to store the file name obtained from the drag and drop operation.
- Write the event handlers for **the DragEnter** and **DragDrop** events as indicated in the previous 2 slides.
- Add an event handler for the click event of the button such that the data from the dropped file is read and displayed in the listbox.
- Test the program will be with the file file1.txt created previously



File I/O Exceptions

- When working with files you need to perform exception handling.
- We can handle generic exceptions or we can choose to handle specific exceptions and decide what message we want to display on exception.
- The list of exceptions for file I/O is available at:

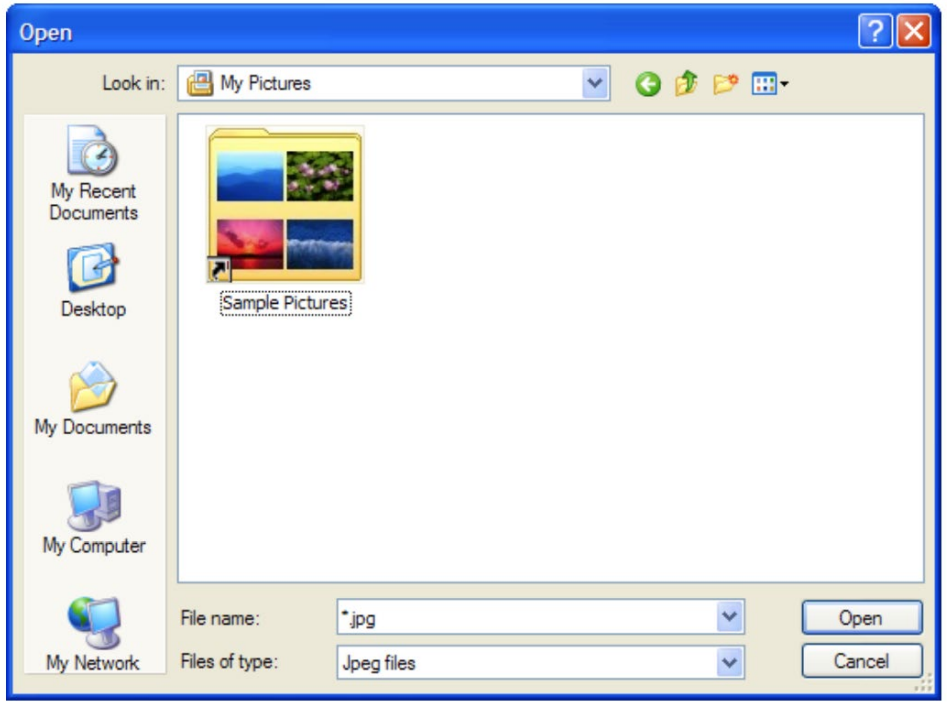
https://docs.microsoft.com/en-us/dotnet/standard/io/handling-io-errors

Stream Readers/Writers

- The File class is suitable for writing into files when all the data are already available in memory. It's also suitable for reading files where the whole data can fit in memory.
- However, for writing/reading bigger files, or for writing/reading onto/from network interfaces or different kinds of files, we make use of stream writer and stream reader objects
- We won't discuss Stream Readers and Writers in this course.

OpenFileDialog

- The OpenFileDialog is used to select a file to open.



OpenFileDialog Properties

Property	Description
Name	The dialog name. Usually the default name is best.
AddExtension	Automatically add the file extension.
CheckFileExists	Indicates whether a warning appears if the user specifies a file that does not exist.
FileName	The name of the selected file, including the path.
Filter	The file filters to display in the dialog box, for example "Jpeg files|*.jpg|Gif files|*.gif|All files|*.*"
Title	Title of the dialog box.
SafeFileName	The name of the file without the path.

OpenFileDialog.ShowDialog()

- The ShowDialog method displays the dialog and returns a result.
- The return type is DialogResult.
- If the return value is DialogResult.OK, then a file was selected.
- If the return value is DialogResult.Cancel, then the dialog was closed or Cancel was pressed without selecting a file.

OpenFileDialog.ShowDialog()

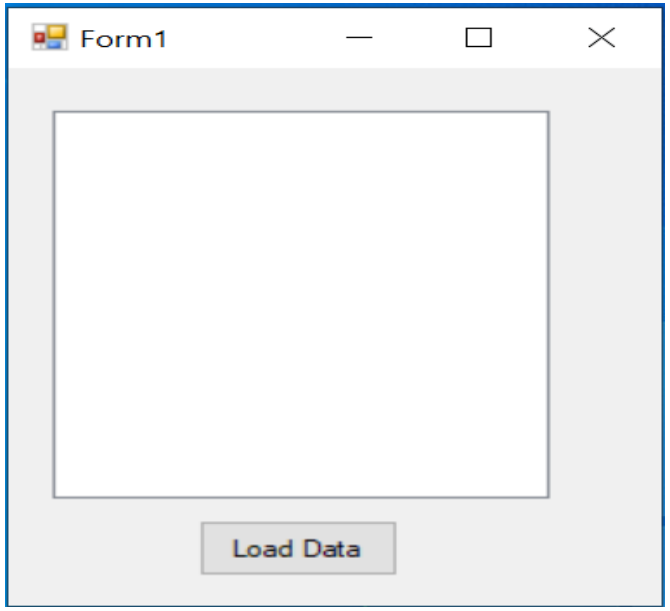
- Show the OpenFileDialog, then display the name of the selected file in the form caption.

```
if (openFileDialog1.ShowDialog() == DialogResult.OK)
{
this.Text = openFileDialog1.FileName;
//read in the file here
}
```

- Once the name of the file is obtained, the file can be processed as previously.

Lecture4-Demo5

- Develop an application with the UI shown below.



- Add an OpenFileDialog to the application
- Write the click event handler for the “Load Data” button, such that it opens the OpenFileDialog and allows the user to choose a file. The file must be read line by line and the data added to the Listbox.
- Use the file “file1.txt” created in earlier Demos to test your program.