Chapter97:FileandStreamI/O

Parameter	Details
path	The location of the file.
append	If the file exist, true will add data to the end of the file (append), false will overwrite the file.
text	Text to be written or stored.
contents	A collection of strings to be written.
source	The location of the file you want to use.
dest	The location you want a file to go to.

Manages files.

Section97.1:ReadingfromafileusingtheSystem.IO.Fileclass

You can use the **System.IO.File.ReadAllText** function to read the entire contents of a file into a string.

```
string text = System.IO.File.ReadAllText(@"C:¥MyFolder¥MyTextFile.txt");
```

You can also read a file as an array of lines using the **System.IO.File.ReadAllLines** function:

```
string[] lines = System.IO.File.ReadAllLines(@"C:¥MyFolder¥MyTextFile.txt");
```

Section97.2:Lazilyreadingafileline-by-lineviaan IEnumerable

When working with large files, you can use the System.IO.File.ReadLines method to read all lines from a file into an IEnumerable<string>. This is similar to System.IO.File.ReadAllLines, except that it doesn't load the whole file into memory at once, making it more efficient when working with large files.

```
IEnumerable<string> AllLines = File.ReadLines("file_name.txt", Encoding.Default);
```

The second parameter of File.ReadLines is optional. You may use it when it is required to specify encoding.

It is important to note that calling ToArray, ToList or another similar function will force all of the lines to be loaded at once, meaning that the benefit of using ReadLines is nullified. It is best to enumerate over the IEnumerable using a foreach loop or LINQ if using this method.

Section97.3:AsyncwritetexttoafileusingStreamWriter

```
// filename is a string with the full path
// true is to append
using (System.IO.StreamWriter file = new System.IO.StreamWriter(filename, true))
{
    // Can write either a string or char array
    await file.WriteAsync(text);
}
```

Section 97.4: CopyFile

File static class

Filestatic class can be easily used for this purpose.

```
File.Copy(@"sourcePath¥abc.txt", @"destinationPath¥abc.txt");
File.Copy(@"sourcePath¥abc.txt", @"destinationPath¥xyz.txt");
```

Remark: By this method, file is copied, meaning that it will be read from the source and then written to the destination path. This is a resource consuming process, it would take relative time to the file size, and can cause your program to freeze if you don't utilize threads.

Section 97.5: Writing lines to a file using the System.IO.StreamWriter class

The **System.IO.StreamWriter** class:

> Implements a TextWriter for writing characters to a stream in a particular encoding.

Using the `WriteLine` method, you can write content line-by-line to a file.

Notice the use of the `using` keyword which makes sure the StreamWriter object is disposed as soon as it goes out of scope and thus the file is closed.

```
string[] lines = { "My first string", "My second string", "and even a third string" };
using (System.IO.StreamWriter sw = new System.IO.StreamWriter(@"C:¥MyFolder¥OutputText.txt"))
{
    foreach (string line in lines)
    {
        sw.WriteLine(line);
    }
}
```

Note that the StreamWriter can receive a second `bool` parameter in it's constructor, allowing to `Append` to a file instead of overwriting the file:

```
bool appendExistingFile = true;
using (System.IO.StreamWriter sw = new System.IO.StreamWriter(@"C:¥MyFolder¥OutputText.txt",
appendExistingFile ))
{
    sw.WriteLine("This line will be appended to the existing file");
}
```

Section 97.6: Writing to a file using the System.IO.File class

You can use the **System.IO.File.WriteAllText** function to write a string to a file.

```
string text = "String that will be stored in the file";
System.IO.File.WriteAllText(@"C:¥MyFolder¥OutputFile.txt", text);
```

You can also use the **System.IO.File.WriteAllLines** function which receives an `IEnumerable<String>` as the

second parameter (as opposed to a single string in the previous example). This lets you write content from an array of lines.

```
string[] lines = { "My first string", "My second string", "and even a third string" };
System.IO.File.WriteAllLines(@"C:¥MyFolder¥OutputFile.txt", lines);
```

Section 97.7: CreateFile

File static class

By using `Create` method of the `File` static class we can create files. Method creates the file at the given path, at the same time it opens the file and gives us the `FileStream` of the file. Make sure you close the file after you are done with it.

ex1:

```
var fileStream1 = File.Create("samplePath");
/// you can write to the fileStream1
fileStream1.Close();
```

ex2:

```
using(var fileStream1 = File.Create("samplePath"))
{
    /// you can write to the fileStream1
}
```

ex3:

```
File.Create("samplePath").Close();
```

FileStream class

There are many overloads of this classes constructor which is actually well documented here. Below example is for the one that covers most used functionalities of this class.

```
var fileStream2 = new FileStream("samplePath", FileMode.OpenOrCreate, FileAccess.ReadWrite, FileShare.None);
```

You can check the enums for FileMode, FileAccess, and FileShare from those links. What they basically means are as follows:

FileMode: Answers "Should file be created? opened? create if not exist then open?" kinda questions.

FileAccess: Answers "Should I be able to read the file, write to the file or both?" kinda questions.

FileShare: Answers "Should other users be able to read, write etc. to the file while I am using it simultaneously?" kinda questions.

Section 97.8: MoveFile

File static class

File static class can easily be used for this purpose.

```
File.Move(@"sourcePath¥abc.txt", @"destinationPath¥xyz.txt");
```

Remark1: Only changes the index of the file (if the file is moved in the same volume). This operation does not take relative time to the file size.

Remark2: Cannot override an existing file on destination path.

Section 97.9: DeleteFile

```
string path = @"c:¥path¥to¥file.txt";
File.Delete(path);
```

While `Delete` does not throw exception if file doesn't exist, it will throw exception e.g. if specified path is invalid or caller does not have the required permissions. You should always wrap calls to `Delete` inside try-catch block and handle all expected exceptions. In case of possible race conditions, wrap logic inside lock statement.

Section 97.10: Files and Directories

Get all files in Directory

```
var FileSearchRes = Directory.GetFiles(@Path, "*.*", SearchOption.AllDirectories);
```

Returns an array of `FileInfo`, representing all the files in the specified directory.

Get Files with specific extension

```
var FileSearchRes = Directory.GetFiles(@Path, "*.pdf", SearchOption.AllDirectories);
```

Returns an array of `FileInfo`, representing all the files in the specified directory with the specified extension.