Files and Streams

- System.IO namespace is used for performing file operations.
- To work with Directories, following classes are used:
  - DirectoryInfo class
  - Directory class

The DirectoryInfo class exposes instance methods for dealing with directories while the Directory class exposes static methods.

Both of these classes can not be inherited.

{Interview Question}

public sealed class DirectoryInfo : FileSystemInfo {…}

public static class Directory {……}

Working with Directories

DirectoryInfo class:

• The DirectoryInfo class provides various instance methods and properties for creating, deleting, and manipulating directories.

• The following table describes some of the common methods

Method	Description
Create	Creates a directory.
CreateSubdirectory	Creates a subdirectory.
Delete	Deletes a directory.
GetDirectories	Gets the subdirectories of the current directory.
GetFiles	Gets the file list from a directory.

- Some of the common properties of DirectoryInfo class

Properties	Description
Exists	Indicates if a directory exists.
Parent	Gets the parent of the current directory.
FullName	Gets the full path name of the directory.
CreationTime	Gets or sets the creation time of current directory.

To see how to use the DirectoryInfo class, consider the following example:

```csharp
static void Main(string[] args)
{
    string path = @"C:\My Folder";
    DirectoryInfo di = new DirectoryInfo(path);

    try
    {
        //---if directory does not exists---
        if (!di.Exists)
        {
            //---create the directory---
            di.Create();   //---c:\My Folder---

            //---creates subdirectories---
            di.CreateSubdirectory("Subdir1"); //---c:\My Folder\Subdir1---
            di.CreateSubdirectory("Subdir2"); //---c:\My Folder\Subdir2---
        }

        //---print out some info about the directory---
        Console.WriteLine(di.FullName);
        Console.WriteLine(di.CreationTime);

        //---get and print all the subdirectories---
        DirectoryInfo[] subDirs = di.GetDirectories();
        foreach (DirectoryInfo subDir in subDirs)
            Console.WriteLine(subDir.FullName);
```

```csharp
            //---get the parent of C:\My folder---
            DirectoryInfo parent = di.Parent;
            if (parent.Exists)
            {
                //---prints out C:\---
                Console.WriteLine(parent.FullName);
            }

            //---creates C:\My Folder\Subdir3---
            DirectoryInfo newlyCreatedFolder =
                di.CreateSubdirectory("Subdir3");

            //---deletes C:\My Folder\Subdir3---
            newlyCreatedFolder.Delete();
        }
        catch (IOException ex)
        {
            Console.WriteLine(ex.Message);
        }
        catch (Exception ex)
        {
            Console.WriteLine(ex.Message);
        }

        Console.ReadLine();
    }
}
```

Directory class:

• The Directory class is similar to DirectoryInfo class.

• The key difference between is that Directory exposes static members instead of instance members.

• **The Directory class also exposes only methods — no properties.**

Some of the commonly used methods are described in the following table.

Method	Description
CreateDirectory	Creates a subdirectory.
Delete	Deletes a specified directory.
Exists	Indicates if a specified path exists.
GetCurrentDirectory	Gets the current working directory.
GetDirectories	Gets the subdirectories of the specified path.
GetFiles	Gets the file list from a specified directory.
SetCurrentDirectory	Sets the current working directory.

Here's the previous program **rewritten to using the Directory class:**

```
static void Main(string[] args)
{
    string path = @"C:\My Folder";
    try
    {
        //---if directory does not exists---
        if (!Directory.Exists(path))
        {
            //---create the directory---
            Directory.CreateDirectory(path);

            //---set the current directory to C:\My Folder---
            Directory.SetCurrentDirectory(path);
            //---creates subdirectories---
            //---c:\My Folder\Subdir1---
            Directory.CreateDirectory("Subdir1");
            //---c:\My Folder\Subdir2---
            Directory.CreateDirectory("Subdir2");
        }
```

```csharp
        //---set the current directory to C:\My Folder---
        Directory.SetCurrentDirectory(path);

        //---print out some info about the directory---
        Console.WriteLine(Directory.GetCurrentDirectory());
        Console.WriteLine(Directory.GetCreationTime(path));

        //---get and print all the subdirectories---
        string[] subDirs = Directory.GetDirectories(path);
        foreach (string subDir in subDirs)
            Console.WriteLine(subDir);

        //---get the parent of C:\My folder---
        DirectoryInfo parent = Directory.GetParent(path);
        if (parent.Exists)
        {
            //---prints out C:\---
            Console.WriteLine(parent.FullName);
        }

        //---creates C:\My Folder\Subdir3---
        Directory.CreateDirectory("Subdir3");

        //---deletes C:\My Folder\Subdir3---
        Directory.Delete("Subdir3");
    }
    catch (IOException ex)
    {
        Console.WriteLine(ex.Message);
    }
    catch (Exception ex)
    {
        Console.WriteLine(ex.Message);
    }
    Console.ReadLine();
}
```

Working With Files

Working with Files Using the File and FileInfo Classes:

• The <u>File class provides static methods</u> for creating, deleting, and manipulating files

• <u>FileInfo class exposes instance members</u> for files manipulation.

Consider the following program, which creates, deletes, copies, renames, and sets attributes in files, **using the File class**:

```
static void Main(string[] args)
{
    string filePath = @"C:\temp\textfile.txt";
    string fileCopyPath = @"C:\temp\textfile_copy.txt";
    string newFileName = @"C:\temp\textfile_newcopy.txt";

    try
    {
        //---if file already existed---
        if (File.Exists(filePath))
        {
            //---delete the file---
            File.Delete(filePath);
        }

        //---create the file again---
        FileStream fs = File.Create(filePath);
        fs.Close();

        //---make a copy of the file---
        File.Copy(filePath, fileCopyPath);
```

```
        //--rename the file---
        File.Move(fileCopyPath, newFileName);

        //---display the creation time---
        Console.WriteLine(File.GetCreationTime(newFileName));

        //---make the file read-only and hidden---
        File.SetAttributes(newFileName, FileAttributes.ReadOnly);
        File.SetAttributes(newFileName, FileAttributes.Hidden);
    }
    catch (IOException ex)
    {
        Console.WriteLine(ex.Message);
    }
    catch (Exception ex)
    {
        Console.WriteLine(ex.Message);
    }
    Console.ReadLine();
}
```

Reading and Writing to Files

The File class contains four methods **to write content to a file**:

• WriteAllText() — Creates a file, writes a string to it, and closes the file

• AppendAllText() — Appends a string to an existing file

• WriteAllLines() — Creates a file, writes an array of string to it, and closes the file

• WriteAllBytes() — Creates a file, writes an array of byte to it, and closes the file

The following statements show how to use the various methods to write some content to a file:

```
string filePath = @"C:\temp\textfile.txt";
string strTextToWrite = "This is a string";
string[] strLinesToWrite = new string[] { "Line1", "Line2" };
byte[] bytesToWrite =
    ASCIIEncoding.ASCII.GetBytes("This is a string");

File.WriteAllText(filePath, strTextToWrite);
File.AppendAllText(filePath, strTextToWrite);
File.WriteAllLines(filePath, strLinesToWrite);
File.WriteAllBytes(filePath,bytesToWrite);
```

The File class also contains three methods **to read contents from a file:**

• ReadAllText() — Opens a file, reads all text in it into a string, and closes the file

• ReadAllLines() — Opens a file, reads all the text in it into a string array, and closes the file

• ReadAllBytes() — Opens a file, reads all the content in it into a byte array, and closes the file

The following statements show how to use the various methods to read contents from a file:

```
string filePath = @"C:\temp\textfile.txt";
string strTextToRead = (File.ReadAllText(filePath));
string[] strLinestoRead = File.ReadAllLines(filePath);
byte[] bytesToRead = File.ReadAllBytes(filePath);
```

NOTE:

The beauty of these methods is that you need not worry about opening and closing the file after reading or writing to it; they close the file automatically after they are done.

StreamReader and StreamWriter Classes:

• When dealing with text files, you may also want to use the StreamReader and StreamWriter classes.

• StreamReader is derived from the TextReader class, an abstract class that represents a reader that can read a sequential series of characters.

The following code snippet uses the StreamReader class to read lines from a text file:

```
try
{
    using (StreamReader sr = new StreamReader(filePath))
    {
        string line;
        while ((line = sr.ReadLine()) != null)
        {
            Console.WriteLine(line);
        }
    }
}
catch (Exception ex)
{
    Console.WriteLine(ex.ToString());
}
```

In addition to the ReadLine() method, the StreamReader class supports the following methods:

- Read() — Reads the next character from the input stream
- ReadBlock() — Reads a maximum of specified characters
- ReadToEnd() — Reads from the current position to the end of the stream

The StreamWriter class is derived from the abstract TextWriter class and is used for writing characters to a stream.

Example

```
try
{
    using (StreamWriter sw = new  StreamWriter(filePath))
    {
        sw.Write("Hello, ");
        sw.WriteLine("World!");
    }
}
catch (Exception ex)
{
    Console.WriteLine(ex.ToString());
}
```

BinaryReader and BinaryWriter Classes

If you are dealing with binary files, you can use the BinaryReader and BinaryWriter classes.

The following example reads binary data from one file and writes it into another, essentially making a copy of the file:

```
string filePath = @"C:\temp\VS2008Pro.png";
string filePathCopy = @"C:\temp\VS2008Pro_copy.png";

//---open files for reading and writing---
FileStream fs1 = File.OpenRead(filePath);
FileStream fs2 = File.OpenWrite(filePathCopy);

BinaryReader br = new BinaryReader(fs1);
BinaryWriter bw = new BinaryWriter(fs2);

//---read and write individual bytes---
for (int i = 0; i <= br.BaseStream.Length - 1; i++)
    bw.Write(br.ReadByte());

//---close the reader and writer---
br.Close();
bw.Close();
```

A File Explorer Program

Build a simple file explorer that displays all the subdirectories and files within a specified directory.

```csharp
class Program
{
    static string path = @"C:\Program Files\Microsoft Visual Studio 9.0\VC#";
    static void Main(string[] args)
    {
        DirectoryInfo di = new DirectoryInfo(path);
        Console.WriteLine(di.FullName);
        PrintFoldersinCurrentDirectory(di, -1);
        Console.ReadLine();
    }


    private static void PrintFoldersinCurrentDirectory(
        DirectoryInfo directory, int level)
    {
        level++;

        //---print all the subdirectories in the current directory---
        foreach (DirectoryInfo subDir in directory.GetDirectories())
        {
            for (int i = 0; i <= level * 3; i++)
                Console.Write(" ");
            Console.Write("|__");
```

```csharp
        //---display subdirectory name---
        Console.WriteLine(subDir.Name);

        //---display all the files in the subdirectory---
        FileInfo[] files = subDir.GetFiles();
        foreach (FileInfo file in files)
        {
            //---display the spaces---
            for (int i = 0; i <= (level+1) * 3; i++)
                Console.Write(" ");

            //---display filename---
            Console.WriteLine("* " + file.Name);
        }

        //---explore its subdirectories recursively---
        PrintFoldersinCurrentDirectory(subDir, level);
    }
  }
}
```

O/P

file:///C:/Documents and Settings/Wei-Meng Lee/Desktop/C# 2008 PR/FilesExplorer/FilesEx...

```
C:\Program Files\Microsoft Visual Studio 9.0\VC#
|__CSharpProjectItems
    |__Windows Forms
        *   CSControlInheritanceWizard.vsz
        *   CSFormInheritanceWizard.vsz
        *   CSharpItems.vsdir
|__CSharpProjects
|__DesignerTemplates
    |__1033
        *   InheritedTemplate.cs
        *   InheritedTemplate.Designer.cs
|__Snippets
    |__1033
        *   SnippetsIndex.xml
        |__NetFX30
            *   propa.snippet
            *   propdp.snippet
        |__OfficeDevelopment
            |__Excel
                |__Controls
                    *   AddSmartTagToExcelWorkbook.snippet
                    *   DataBindListObject.snippet
                    *   DataBindNamedRange.snippet
                    *   FilterListObject.snippet
                    *   HideOrShowWorksheet.snippet
```

The Stream Class

- A stream is an abstraction of a sequence of bytes.

- The bytes may come from a file, a TCP/IP socket, or memory.

- In .NET, a stream is represented by the **Stream class**.

- The Stream class forms the base class of all other streams, and it is also implemented by the following classes:
  - BufferedStream — Provides a buffering layer on another stream to improve performance
  - FileStream — Provides a way to read and write files
  - MemoryStream — Provides a stream using memory as the backing store
  - NetworkStream — Provides a way to access data on the network
  - CryptoStream — Provides a way to supply data for cryptographic transformation

- Streams fundamentally involve the following operations:
  - ❑ Reading
  - ❑ Writing
  - ❑ Seeking

The following code copies the content of one binary file and writes it into another using the Stream class:

```
try
{
    const int BUFFER_SIZE = 8192;
    byte[] buffer = new byte[BUFFER_SIZE];
    int bytesRead;

    string filePath = @"C:\temp\VS2008Pro.png";
    string filePath_backup = @"C:\temp\VS2008Pro_bak.png";

    Stream s_in = File.OpenRead(filePath);
    Stream s_out = File.OpenWrite(filePath_backup);

    while ((bytesRead = s_in.Read(buffer, 0, BUFFER_SIZE)) > 0)
    {
        s_out.Write(buffer, 0, bytesRead);
    }
    s_in.Close();
    s_out.Close();
}
catch (Exception ex)
{
    Console.WriteLine(ex.ToString());
}
```

In addition to the Read() and Write() methods, the Stream object supports the following methods:

• ReadByte() — Reads a byte from the stream and advances the position within the stream by one byte, or returns -1 if at the end of the stream

• WriteByte() — Writes a byte to the current position in the stream and advances the position within the stream by 1 byte

• Seek() — Sets the position within the current stream

The following example writes some text to a text file, closes the file, reopens the file, seeks to the fourth position in the file, and reads the next six bytes:

```
try
{
    const int BUFFER_SIZE = 8192;
    string text = "The Stream class is defined in the System.IO namespace.";
    byte[] data = ASCIIEncoding.ASCII.GetBytes(text);
    byte[] buffer = new byte[BUFFER_SIZE];
    string filePath = @"C:\temp\textfile.txt";

    //---writes some text to file---
    Stream s_out = File.OpenWrite(filePath);
    s_out.Write(data, 0, data.Length);
    s_out.Close();

    //---opens the file for reading---
    Stream s_in = File.OpenRead(filePath);

    //---seek to the fourth position---
    s_in.Seek(4, SeekOrigin.Begin);

    //---read the next 6 bytes---
    int bytesRead = s_in.Read(buffer, 0, 6);
    Console.WriteLine(ASCIIEncoding.ASCII.GetString(buffer, 0, bytesRead));

    s_in.Close();
    s_out.Close();
}
catch (Exception ex)
{
    Console.WriteLine(ex.ToString());
}
```

The FileStream Class

• The FileStream class is designed to work with files, and it supports both synchronous and asynchronous read and write operations.

• Consider an example for copying contents from one file to another using FileStream class.

```
try
{
    const int BUFFER_SIZE = 8192;
    byte[] buffer = new byte[BUFFER_SIZE];
    int bytesRead;

    string filePath = @"C:\temp\VS2008Pro.png";
    string filePath_backup = @"C:\temp\VS2008Pro_bak.png";

    FileStream fs_in = File.OpenRead(filePath);
    FileStream fs_out = File.OpenWrite(filePath_backup);

    while ((bytesRead = fs_in.Read(buffer, 0, BUFFER_SIZE)) > 0)
    {
        fs_out.Write(buffer, 0, bytesRead);
    }

    fs_in.Dispose();
    fs_out.Dispose();
    fs_in.Close();
    fs_out.Close();
}
catch (Exception ex)
{
    Console.WriteLine(ex.ToString());
}
```

This is a Synchronous File Read Write operation.