Chapter 5

Data Access – File System Data

Mrs. Swati Satpute
Fergusson College

Objective

- Program IO
- Stream
- System.IO Namespace
- Directory Class
- DirectoryInfo Class
- DriveInfo and Path Class
- File and FileInfo Class
- Stream class
- FileStream class
- BinaryReader & BinaryWriter Class
- Serialization
  - Types of Serialization

Examples

- Basic Demo - DirectoryInfo Class
- Stream Reader-Writer
- Binary Reader-Writer
- Binary Serialization

Objective

Understanding

- What is a Stream and how to use stream classes to access files.
- Using File object to manipulate files.
- Reading from and writing to files.
- Reading and writing compressed files.
- How to store and retrieve objects data using serialization.
- Monitoring file system by using FileWatcher class.

Input and Output Operations in C#

- I/O operations in C# is stream based.
- **Stream is flow of data from a source to a receiver** through a channel.
- Fundamental Operations of Streams
  - Stream Reading
  - Stream Writing
  - Stream Seeking
- Types of Streams
  - Byte Streams
  - Character Streams
- Some of the predefined streams are
  - Console.Out
  - Console.In
  - Console.Error

System.IO Namespace

- C# provides various stream classes to perform stream based I/O.
- These all classes are defined in System.IO namespace.

- There is difference between file and stream
  - A **file** is a collection of data or information that has a **name and persistent storage**.
  - Stream is a sequence of bytes travelling from a source to a destination over a communication path.

Why Files?

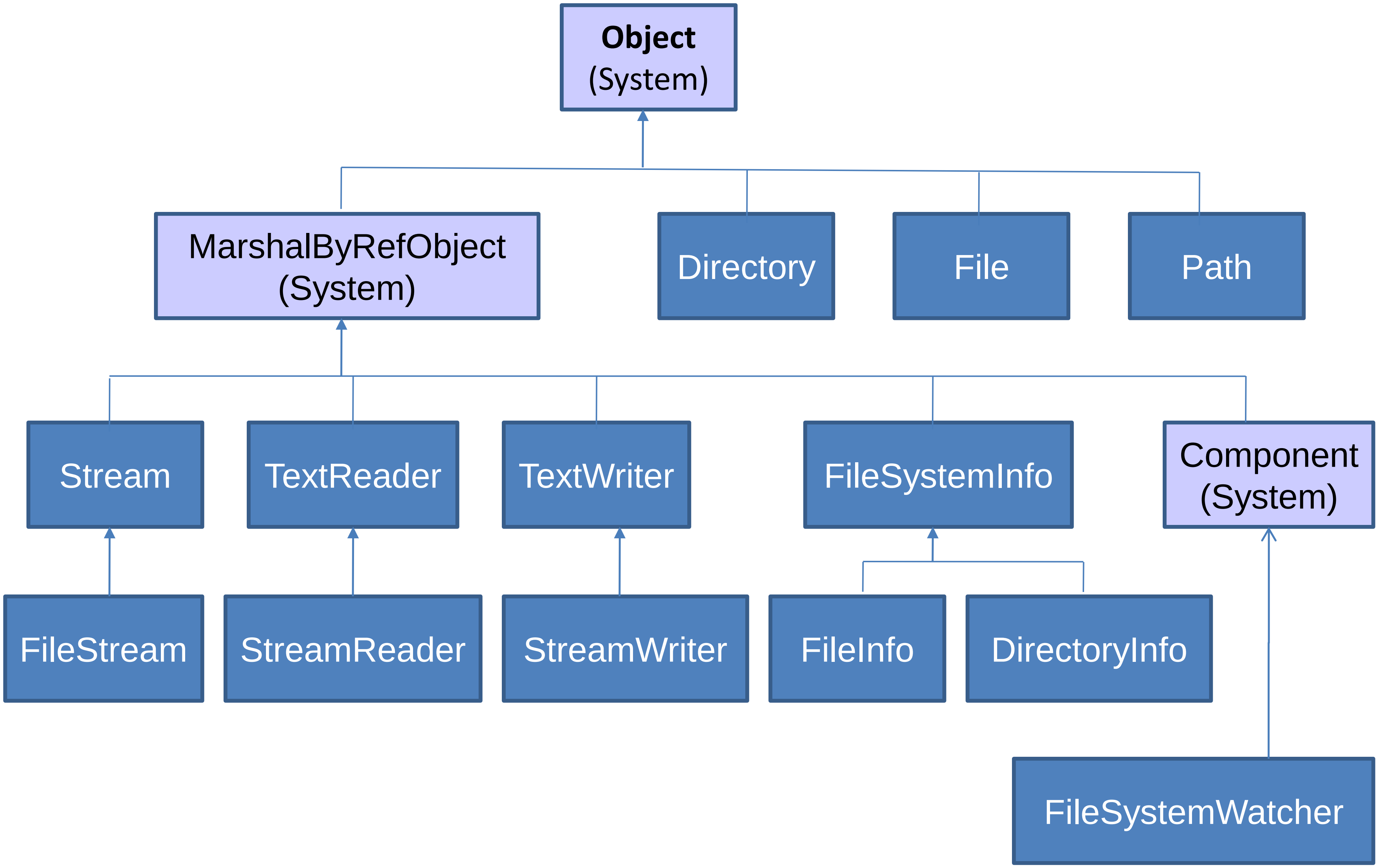
Using Files

- One can **store data** between instances of application.
- Provides a way to **share data across application**.

Logical types of files

- Configuration files
- Log files
- Comma separated files / data files

File System Classes



File System Classes

Name	Description
File	**Static class**, offers many static methods as **move**, **copy** and **delete** files.
Directory	**Static class**, offers many static methods as **move**, **copy** and **delete** directories.
Path	Utility class used to **manipulate path names**.
FileInfo	**Represents a physical file on disk**, has methods to manipulate this file.
DirectoryInfo	**Represents a physical directory on disk**, has methods to manipulate this directory .
FileSystemInfo	Base class for both **FileInfo** and **DirectoryInfo.** This allows class to deal with files and directories at the same time (using polymorphism).

File System Classes

Name	Description
FileStream	**Represents a file** that can be written to, or read from or both. This file can be read from or written to **asynchronously** or **synchronously**.
StreamReader	**Reads character data** from a stream. StreamReader can be created using FileStream as a base.
StreamWriter	**Writes character data** to a stream. StreamWriter can be created using FileStream as a base.
FileSystemWatcher	It is **used to monitor files and directories**. **Exposes event like Changed, Created, Deleted, Renamed.**

Directory Class

- Directory class provides **static methods** that perform **security checks** on all methods

- C# I/O system **provides full read/write access to new directories by default**

```
string DirectoryName = @“C:\MyDir”;
if(Directory.Exists(DirectoryName))
    Console.WriteLine(“Exists”);
else
{
    Directory.CreateDirectory (DirectoryName);
    Console.WriteLine(“Created”);
}
```

Directory class

Method	Description
CreateDirectory()	Creates a directory with the specified path.
Delete()	Deletes the directory and all the files into it.
GetDirectories()	Returns list of sub directories as a **string array.**
EnumerateDirectories()	Similar to GetDirectories() but returns an **IEnumerable <string> collection of directory names.**
GetFiles()	Returns **list of file** names present under current directory **as a string array.**
EnumerateFiles()	Like GetFiles(), but returns an **IEnumerable<string>** collection of filenames.

Directory class

Method	Description
GetFileSystemEntries()	Returns **list of file names and directory names** present under current directory as a string array.
EnumerateFileSystemEntries()	Like GetFileSystemEntries(), but returns an IEnumerable<string> collection of filenames and directory names.
Move()	Moves a specified directory to a new location. Can specify new name for the folder in the new location

Note: EnumerateXXX() methods introduced in .NET 4.0, **provide better performance** than GetXXX() when large amount of files and directories exist.

DirectoryInfo

- Class **represents single directory** on machine
- **Rules**
  - If application is making **single call**, then use **Directory** class.
  - In case of making **series of calls on one directory**, instantiate **DirectoryInfo class**.

Property	Description
Parent	(**Read – only property**) Represents parent directory of the current directory.
Root	(**Read – only property**) Represents Root directory of the current directory. e.g. C:\Net\Projects

DirectoryInfo Class

```
string DirectoryName = @“C:\MyDir”;

if(Directory.Exists(DirecoryName))
    Console.WriteLine(“Exists”);
else
{
    DirectoryInfo dir = new DirectoryInfo(DirectoryName);
    dir.Create ();
    Console.WriteLine(“Created”);
}
```

Path Name and Relative Path

- **Absolute Path**, explicitly specifies a file or directory location.

  e.g. C:\Net\Project\data.txt

- **Relative Path** are **relative to starting location**. No explicit drive is mentioned

  e.g. ..\data.txt

- Directory.GetCurrentDirectory() gives the current directory name where application is executing.

DriveInfo Class

- DriveInfo class is used to determine
  - Which drives are available?
  - What is the type of drives?
  - The capacity of drive
  - The available space on the drive

Demo

```
string strDrive = @“C:\”;
DriveInfo drv = new DriveInfo(strDrive);
Console.WriteLine(drv.AvailableFreeSpace.ToString(),
drv.Name);
```

Path Class

- A path is a **string that provides the location** of file or directory.
- The members of the Path class enables to
  - Determine whether a file name extension is part of path
  - Combine two strings into one path name

```
string strPath = @“C:\Net\Projects\test.txt”;
Console.WriteLine(Path.GetFileName(strPath));
Console.WriteLine(Path.GetTempPath());
```

File Class

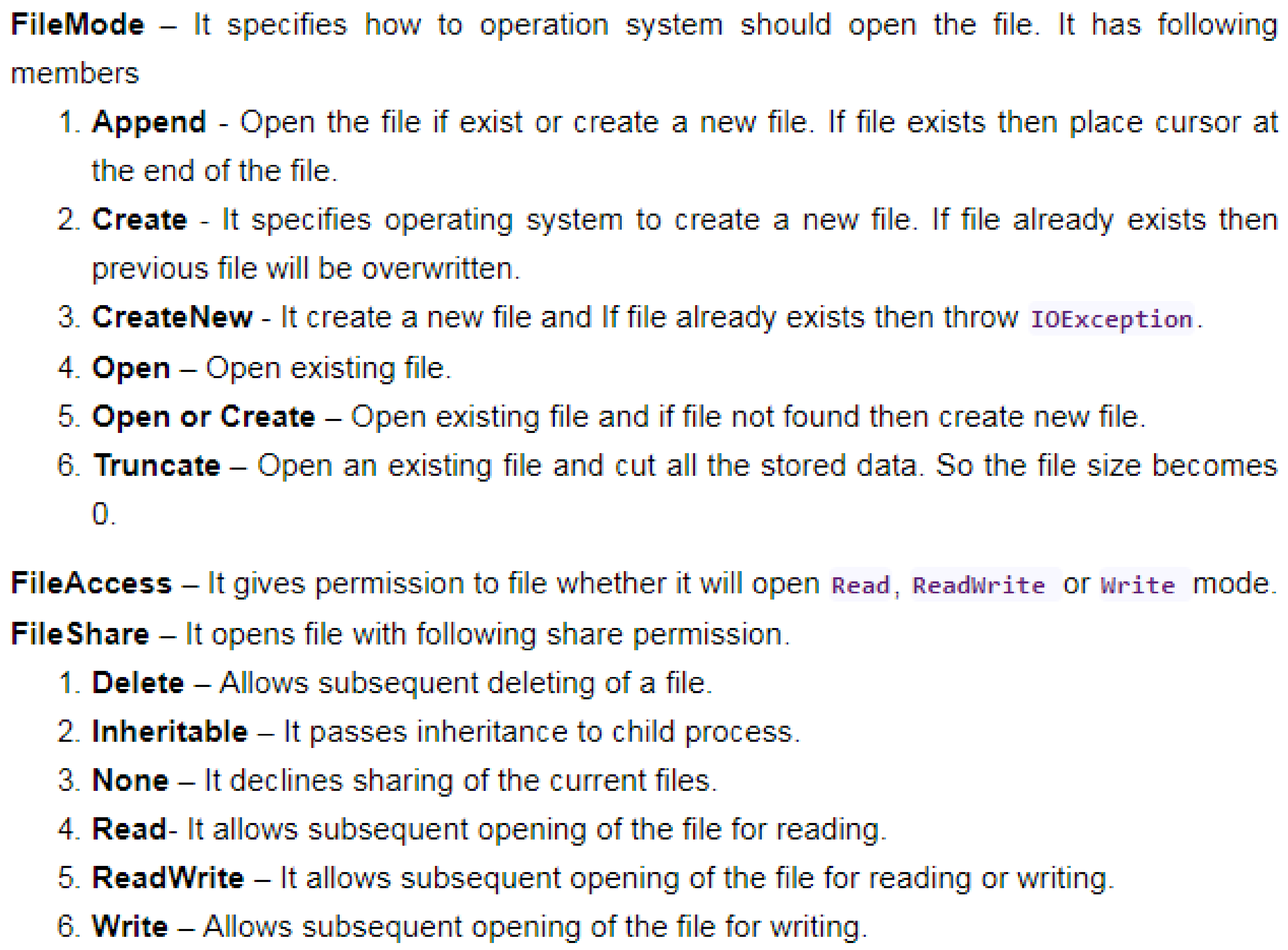
- The File class is **used to get and set file attributes**.

- It is a **Static class** and helps to manage a single class.

- The File class provides a static method that **performs security checks** on all methods.

- Three enumerations FileAccess, FileShare, and FileMode are provided to customize the behavior of various File Methods.

File Class



FileMode – It specifies how to operation system should open the file. It has following members

1. **Append** - Open the file if exist or create a new file. If file exists then place cursor at the end of the file.
2. **Create** - It specifies operating system to create a new file. If file already exists then previous file will be overwritten.
3. **CreateNew** - It create a new file and If file already exists then throw `IOException`.
4. **Open** – Open existing file.
5. **Open or Create** – Open existing file and if file not found then create new file.
6. **Truncate** – Open an existing file and cut all the stored data. So the file size becomes 0.

FileAccess – It gives permission to file whether it will open `Read`, `ReadWrite` or `Write` mode.

FileShare – It opens file with following share permission.

1. **Delete** – Allows subsequent deleting of a file.
2. **Inheritable** – It passes inheritance to child process.
3. **None** – It declines sharing of the current files.
4. **Read**- It allows subsequent opening of the file for reading.
5. **ReadWrite** – It allows subsequent opening of the file for reading or writing.
6. **Write** – Allows subsequent opening of the file for writing.

File class

Method	Description
Copy()	Copies file from source to destination.
Create()	Creates a file in the specified path.
Delete()	Deletes a file.
Open()	Returns a FileStream object at the specified path.
Move()	Moves file to specified location. Can specify new name to the file which is moving.

FileSystemInfo properties

- Methods of the FileSystemInfo class are used to perform **file and directory** manipulations.
- Base class for FileInfo and DirectoryInfo.

Property	Description
Attributes	Gets or sets the attributes of the current file or directory, using the FileAttributes enumeration .
CreationTime	Gets or sets the creation date and time of the current file .
Extension	(Read – only property) Retrieves the extension of the file.
Exists	**(Abstract property)** Implemented in FileInfo and DirectoryInfo. Determines if the file exists.

FileSystemInfo properties

Property	Description
LastAccessTime	Gets or sets the date and time that the current file was last accessed .
LastWriteTime	Gets or sets the date and time that the current file was last written to
FullName	(Read – only property) Retrieves the full path of the file

FileInfo Class

- Unlike File class, **FileInfo is not static**.
- FileInfo represents a file on disk, network location.

```
E.g. FileInfo objFile =  new FileInfo(“Test.txt”);
       if objFile.Exist()
        Console.WriteLine(“File Exists”);


       if File.Exist(Test.txt)
        Console.WriteLine(“File Exists”);
```

File Vs FileInfo

- FileInfo has most of the File class methods.
- **File class** should be **preferred** if **one operation to be performed.**
  - This will **result in faster operation as no object instantiation** is required.
- **FileInfo class is preferred** when **multiple operations** to be performed **on same file**.
  - This will result in faster operation as object is referring to correct (appropriate) file.
  - Static method has to find correct file for each reference.

FileInfo properties

Property	Description
Directory	(Read – only property) Retrieves a DirectoryInfo object representing the directory containing the current file.
DirectoryName	(Read – only property) Returns the path to the file’s directory.
IsReadOnly	Shortcut to the read - only attribute of the file. Accessible through Attributes property also
Length	(Read – only property) Return the **size of the file in bytes**, returned as long value

FileInfo Class

- FileInfo in itself **doesn’t represent Stream**.
- Stream object has to be **created to read or write to a file.**

```
FileInfo MyFileInfo = new FileInfo(“Data.txt”);
FileStream MyFileStream = MyFileInfo.OpenRead();
```

Streams

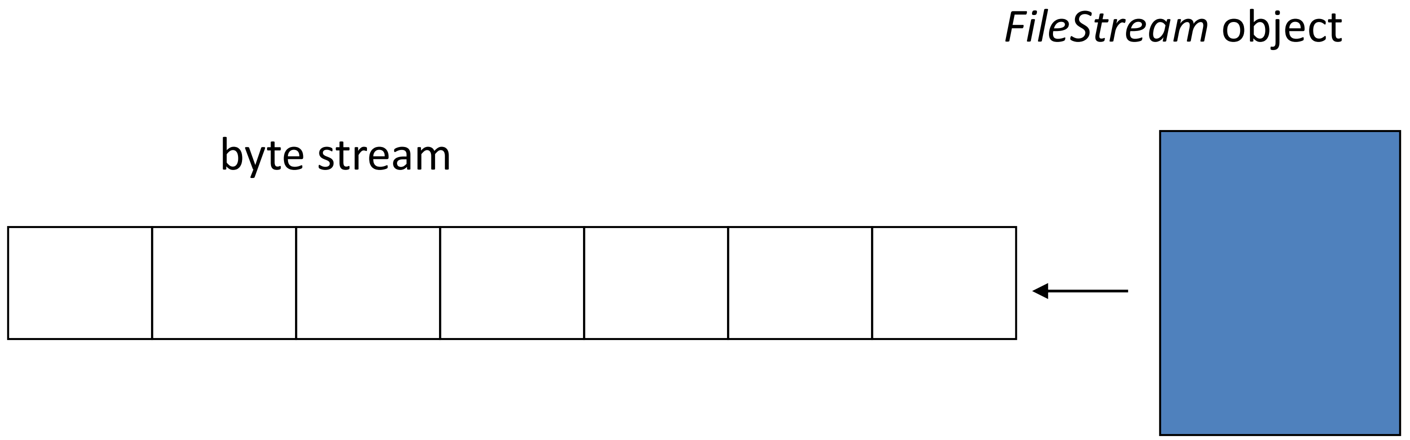
- Stream is an **abstract representation of a serial device.**
- **Serial device** is something that **stores data in linear manner and access the same way**.

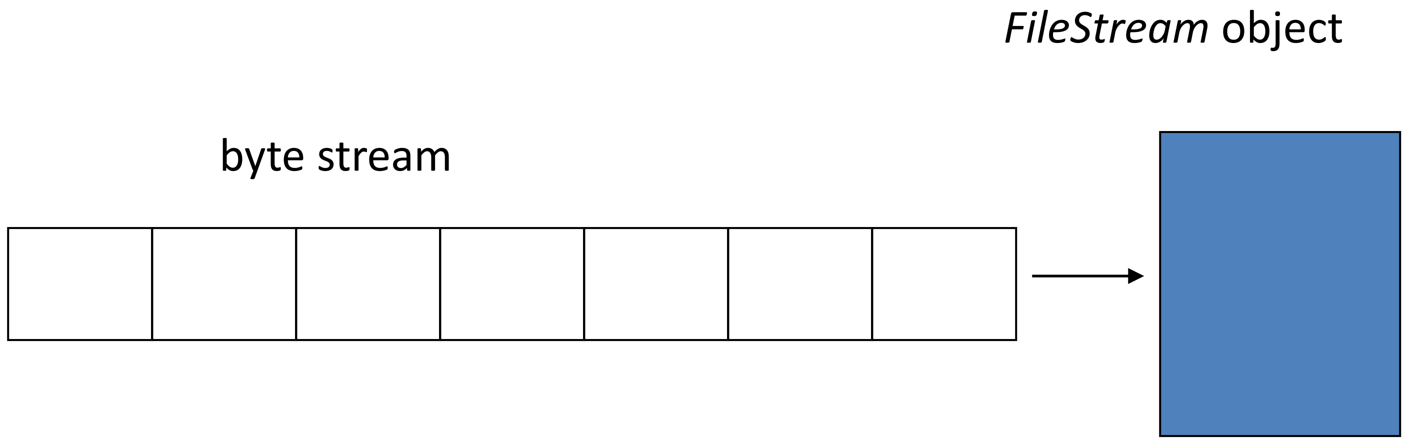
  i.e. byte by byte (1 at a time)
- Serial device examples are
  - Disk file
  - Network channel
  - Memory location
  - Printer
  - Any object that supports reading and writing in linear manner

Streams

- Since the data flow manner is fixed i.e. linear, **code** written intending one device **can be reused** for another device.
  - This **enables** writing **generic code routines**.

Byte Stream to File Stream





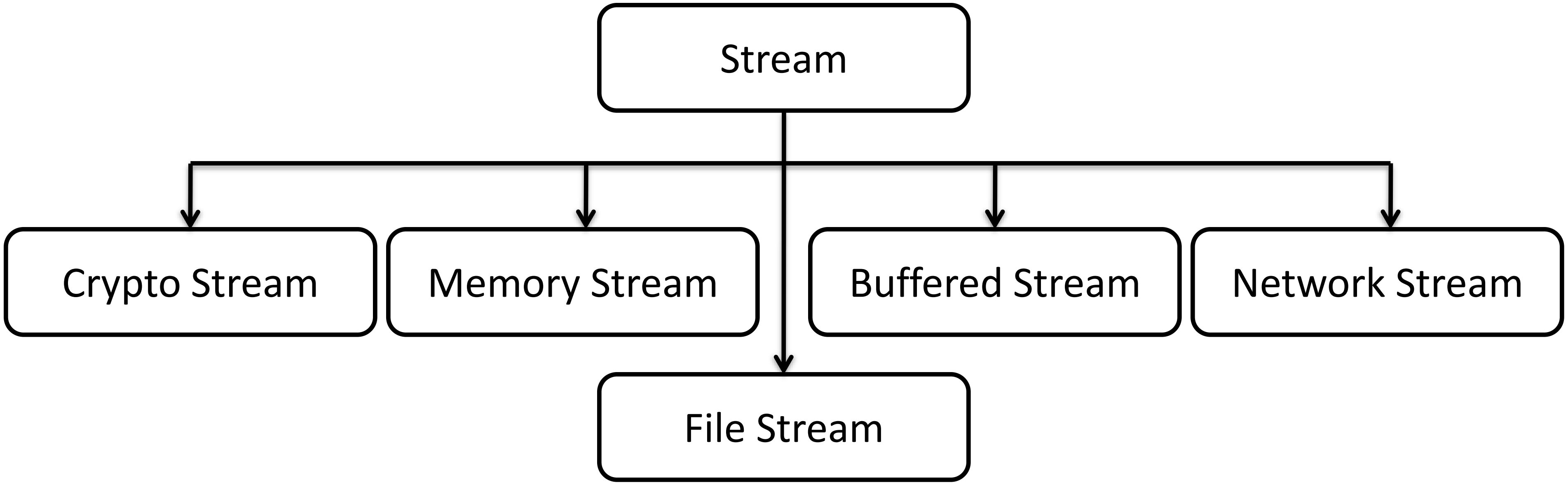
Types of Streams

- Input Stream
  - Used for reading data into memory.
  - E.g. Keyboard.

- Output Stream
  - Used to write data to external destination.
  - Destination can be
    - File
    - Printer
    - Network location

Stream Classes

- Stream is an abstract class for all other stream classes
- Common I/O Stream Classes



Stream Classes

- BufferedStream Class
  This stream **adds buffering to another** stream.

- CryptoStream Class
  This stream links data stream to cryptographic transformations. It is defined in the System.Security.Cryptography namespace

- MemoryStream Class
  Data encapsulated in a MemoryStream is directly accessible in memory

- NetworkStream Class
  A stream over a network connection is represented by NetworkStream. It is defined in System.Net.Sockets namespace.

FileStream Class

- This class **represents a stream pointing to a file** on disk or network path.
- FileStream class **operates on bytes and byte arrays.**
- FileStream object provides **random file access** facility. (i.e. accessing data at some point in the middle of file)

- Note:
  - **Stream classes operate on character data.**
  - Working with character data is easier too.

FileStream Class

```
FileStream myFileStream = new FileStream("Data.txt",
    FileMode.Append, FileAccess.ReadWrite);
```

Opening file for Reading

```
FileStream myFileStream = File.OpenRead("Data.txt");
```

Or

```
FileInfo myFileInfo = new FileInfo(“Data.txt”);
FileStream myFileStream = myFileInfo.OpenRead();
```

File Position

- **FileStream** class **maintains an internal pointer pointing to a location** from where the next read or write will be performed.
- This pointer **can** be utilized to **point to any location within file.**
- **Seek(offset, SeekOrigin)**
  - Offset is the number of position from SeekOrigin.
  - SeekOrigin can be **Begin**, **End**, **Current**.
- **Negative Seek is possible**.

```
myFileStream.Seek(-5, SeekOrigin.End);
```

Demo - Random FileAccess

- **Decoder** class is from System.Text namespace.
- It is used to **convert raw byte stream into** more useful items e.g. **characters**

Demo

```
Decoder d = Encoding.UTF8.GetDecoder();
d.GetChars(byteData, 0, byteData.Length, charData, 0);
```

- **Encoder** class is from System.Text namespace.
- It is used to **convert characters to raw byte stream.**

StreamWriter Object

- StreamWriter **enables to write characters and string to a file.**
- This class handles underlying conversions and writing to file.

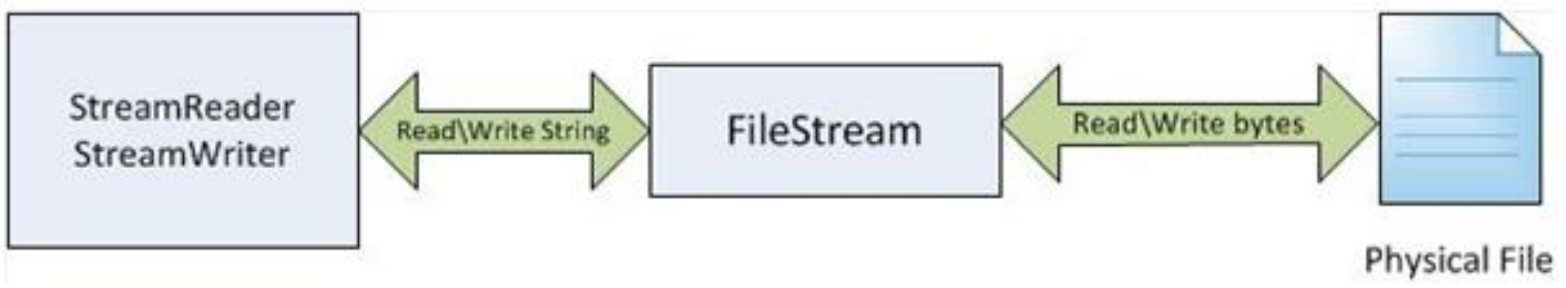
```
StreamWriter sw = new StreamWriter (@“c:\Net\Projects\data.txt”,
    true);
sw.Write (“We are in StreamWriter class.”);
sw.Close( );
```

true indicates append to existing file if present. If no file present then create new.

StreamWriter Object

- To specify FileMode and FileAccess attributes,
  - Create FileStream object
  - Create StreamWriter using FileStream object.

```
FileStream myFileStream = new FileStream(FilePath, FileMode.Append);
StreamWriter mysw = new StreamWriter(myFileStream);
```



StreamReader Object

- StreamReaders will be used to read data from files.

```
FileStream myFileStream = new FileStream(FilePath, FileMode.Open);
    StreamReader mysr = new StreamReader(myFileStream);
    strData = mysr.ReadLine();
```

Reading Data

- Data can be read using
  - ReadLine()
  - Read()

- In .Net 4.0, File.ReadLines() is introduced to read **large files**
  - Returns IEnumerable<string> collection

```
foreach (string strData in File.ReadLines("Data.txt"))
        Console.WriteLine(strData);
```

Delimited Files

- These are common form of Data storage, used to share data across application.
- Comma separated value (CSV) file is used for importing data from SQL server

BinaryWriter

Demo

- BinaryWriter class writes Primitive data type as int, uint or char **in binary to a stream**.
- As its name says BinaryWriter **writes binary files that uses a specific data layout for its bytes.**
- BinaryWriter **create binary file** that is not human understandable but the machine can understand it more smoothly.
- It **supports writing string in a specific encoding**.
- BinaryWriter class provides methods for writing primitive data types to a stream.

Compressed Files

- System.IO.Compression namspace enables Reading from and Writing to Compressed Files.
- Compression Classes
  - **DeflateStream**
  - **GZipStream**
- Both of the algorithms are **freely available.**
- **Compression takes place internally** while saving data to or reading data from the source.

GZipStream - DeCompression

```
FileStream myFileStream = new FileStream(strCompFileName,
FileMode.Open, FileAccess.Read);

GZipStream myCompressionStream = new
GZipStream(myFileStream, CompressionMode.Decompress);

 StreamReader mysw = new StreamReader(myCompressionStream);

      strData = mysw.ReadLine();
      Console.WriteLine(strData);
       mysw.Close();
```

Demo

Lab Assignment

1. Write a console Application which does the following
- Create **Demo** folder under “C:”
- Create a data.txt file containing
  - “Welcome to the File operation Demo”
  - Add current date and time

2. Write a console Application which uses BinaryWriter to log Exception details in a bin file

   - log details like Exception message, ID and date time.

   - Use BinaryReader to read the contents and display it on Screen

3. Write a console application for

   - reading directory contents

   - read files

   - print file details like file name, full name, size, creation time, Last accessTime

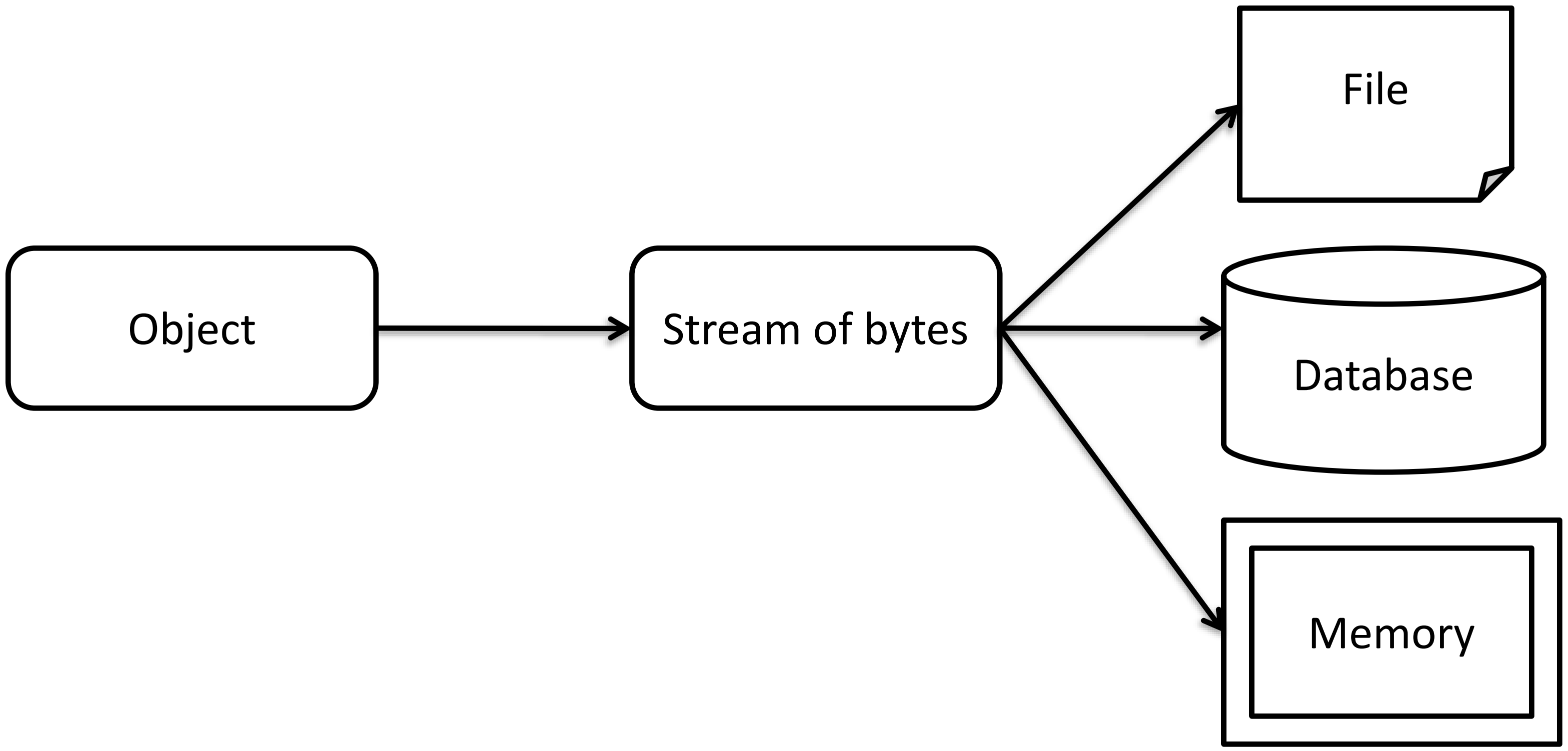
Serialization

- Serialization is a process of storing the state of an object to a storage medium.

- **Public** and **private** fields of the object and the name of the class, including the assembly containing the class, **are converted to a stream of bytes**, which is then **written to a data stream**.

- This **data** can be **retrieved back** and **object** can be **recreated** through a reverse process called **de-serialization**

Serialization



Serialization

- To perform serialization, the ***Serializable*** attribute is added to the class.

```
[Serializable]
Class Employee
{
    int EmpID;
    public string EmpName;

    public Employee (int id, string nm)
    {
      EmpID = id;
      EmpName = nm;
    }
}
```

- There are three types of serialization – **Binary**, **XML** and **SOAP**

Binary Serialization

- It writes **content** of object into **binary form** to a file.
- Class used for this is : BinaryFormatter
- Namespace : System.Runtime.Serialization.Formatters.Binary

```
FileStream fs = new FileStream (@"c:\myfile.txt", FileMode.Create, FileAccess.Write);
Employee emp = new Employee (10, "abc");
BinaryFormatter bf = new BinaryFormatter ( );

bf.Serialize (fs, emp);
fs.Close ( );
-
-
Employee emp1 = (Employee) bf.Deserialize (fs);
```

SOAP Serialization

- It can be used to **serialize objects into SOAP message**
- Class used for this is : SoapFormatter.
- SoapFormatter is a XML based formatter
- Namespace: System.Runtime.Serialization.Formatters.Soap

```
FileStream fs = new FileStream (@“c:\myfile.txt”, FileMode.Create, FileAccess.Write);
Employee emp = new Employee (10, “abc”);
SoapFormatter sf = new SoapFormatter ( );

sf.Serialize (fs, emp);
fs.Close ( );
-
-
Employee emp1 = (Employee) sf.Deserialize (fs);
```

XML Serialization

- It writes content of object into XML file
- It can **serialize only public members** of the class
- Class used for this is : XMLSerializer
- Namespace : System.Xml.Serialization

```
FileStream fs = new FileStream (@"c:\myfile.xml", FileMode.Create, FileAccess.Write);
Employee emp = new Employee (10, "abc");
XmlSerializer xs = new XmlSerializer (typeof (Employee));

xs.Serialize (fs, emp);
fs.Close ( );
-
-
Employee emp1 = (Employee) xs.Deserialize (fs);
```

[NonSerialized]

- If the object must be serialized, apply the NonSerialized attribute to specific fields that store sensitive data.
- Apply it to such field which should not be exposed to external system. Otherwise data will be exposed to others.
- E.g. password.

[NonSerialized]

```
[Serializable]
Class Employee
{
   int EmpID;
   public string EmpName;
    [NonSerialized] String Password

   public Employee (int id, string nm)
   {
     EmpID = id;
     EmpName = nm;
   }
}
```

Demo

- Binary Serialization – with [NonSerialized]

Lab Assignment

- You are assigned to develop a project in which project manager wants following functionality. Create Student Folder in D drive using **DirectoryInfo** class.
- Ask student's name and create a file with that name and store in Student folder.
- Ask student's details and save information in that file.
- Print following option on console screen.
  - View Saved File
  - View Directory Details

Expected Output

```
Student Folder Created Successfully at D:\Student

Please Enter your Name :
Steven Clark
Steven Clark file is created at D:\Student\Steven Clark.txt

Please Enter your Details. Your Name:
Steven Clark
Your Age :
22
Your Current City :
LA
Your Subject :
Computer Science
Information Saved on D:\Student\Steven Clark.txt

Select What you want Next.
Press 1 to view Saved File
Press 2 to view Directory Info
Press any key to Exit.
1
Student Name : Steven Clark
Age : 22
City : LA
Subject : Computer Science
```

References

- Book referred “Beginning Visual C# 2010” by Wrox publication.

Question Bank

- What is difference between Directory and DirectoryInfo class? When to use which class, explain.
- What is absolute path, relative path? Give examples.
- What is DriveInfo class? What it is sued for?
- Explain Streams.
- What are different types of streams?
- Explain Random file access facility of FileStream class.
- Explain Compression classes. Which algorithm they follow.
- What is serialization? What are different types of serialization?
- Which class is used for NTFS monitoring?