



CHAPTER 6

C#

FILES AND STREAMS

Contents

2

1. Introduction
2. Data Hierarchy
3. Files and Streams
4. Working with Directory and File classes
5. Working with Text files(Sequential access file)
6. Working with Binary Files (Random access files)
7. Serialization

Introduction

3

- Variables and arrays offer only temporary storage of data: the data is lost when a local variable "goes out of scope" or when the program terminates.
 - By contrast, files (and databases) are used for long-term retention of large amounts of data, even after the program that created the data terminates.
 - Data maintained in files often is called **persistent data**.
- Computers store files on secondary storage devices, such as magnetic disks, optical disks, flash memory and magnetic tapes.

- Most applications require data to be stored in files
 - Spreadsheets
 - Word processing
 - etc
- C# has an extensive set of classes for handling files of various types
 - These are to be found in the **System.IO namespace**
- The general concept of streams will be applied to both files and networks

- In C#, streams are simply sequences of bytes
 - No formatting structure is placed on this byte stream, this is imposed by the application
 - We can read a byte stream from an *input stream* object
 - We can write a byte stream to an *output stream* object
 - The input and output stream objects are created from class ***FileStream***

Data Hierarchy

6

- Ultimately, all data items that computers process are reduced to combinations of 0s and 1s.
 - This occurs because it's simple and economical to build electronic devices that can assume two stable states one state represents 0 and the other represents 1.
- It's remarkable that the impressive functions performed by computers involve only the most fundamental manipulations of 0s and 1s.

10/22/2024

□ Bits

- The smallest data item that computers support is called a bit (short for “binary digit”—a digit that can assume one of two values).

□ Characters

- It's preferable to program with data in forms such as decimals digits, letters and special symbols and many others. These referred to as **characters**.

□ Fields

- It is a group of characters that conveys meaning.

□ **Records**

- Record is composed of several related fields.

□ **Files**

- File is a group of related records
- **Record key**
 - To facilitate the retrieval of specific records from a file, at least one field in each record is chosen as a record key, which identifies a record as belonging to a particular person or entity and distinguishes that record from all others.
- **Sequential Files**
 - A common organization of records in a file, a sequential file, in which records typically are stored in order by a record-key field

- **Database**

- Related data are often stored in a **database**. A collection of programs designed to create and manage databases is called a **database management system (DBMS)**.

Files and Streams

10

- C# views each file as a **sequential stream of bytes**.
 - Each file ends either with an **end-of-file marker** or at a specific byte number that's recorded in a system maintained administrative data structure.
 - When a file is opened, an object is created and a stream is associated with the object.
 - When a console app executes, the runtime environment creates three stream objects that are accessible via properties **Console.Out**, **Console.In** and **Console.Error**, respectively.

What is Stream?

11

- A stream is an **ordered sequence of bytes**, which is send from one application or input device to another application or output device.
 - These bytes are written and read one after the other and always arrive in the same order as they were sent.
 - Streams are an **abstraction of a data communication channel that connects two devices or applications**.
- Every time when you read or write from or to a file, you have to **open a stream** to the corresponding file, do the reading or writing, and then **close the stream**.

- There are two types of streams – **text streams** and **binary streams** but this separation has to do with the interpretation of the sent and received bytes.
 - Sometimes, for convenience, a sequence of bytes can be treated as text (in a predefined encoding) and is referred to as a text stream.
- Streams allow **sequential data access** (manipulate the data only in the order in which it arrives from the stream).

□ Basic operations with Streams

- **Creation**
 - Connect the stream to a data source, a mechanism for data transfer or another stream.
- **Reading**
 - Extracting data from the stream.
- **Writing**
 - Sending data to the stream in a specific way
- **Positioning**
 - Move the current position of the stream.
- **Closing**
 - Complete the work with the stream and releases the occupied resources
 - Always **close the streams and files you work with!**

Streams in .NET – Basic Classes

14

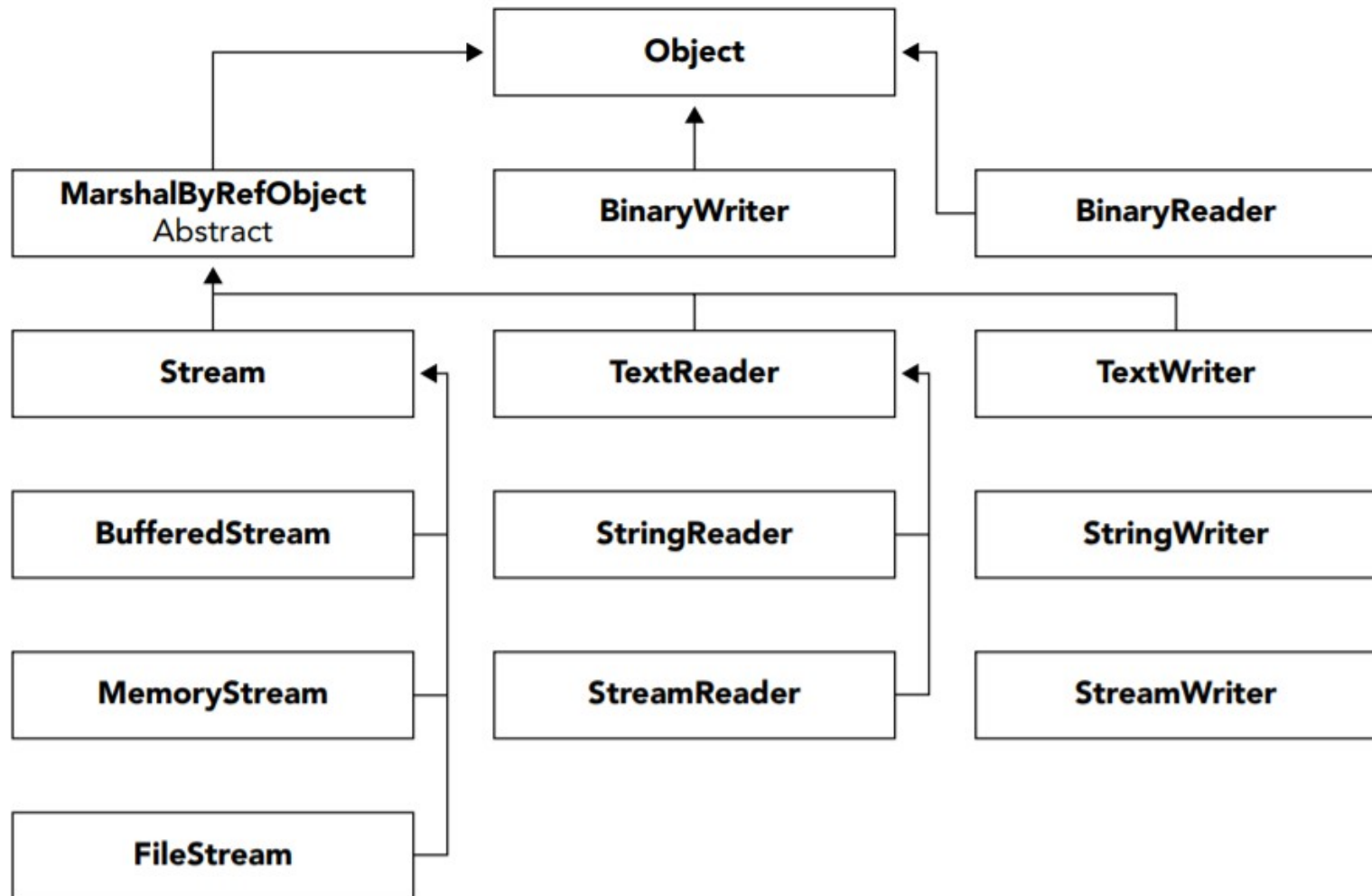
- In .NET Framework classes for working with streams are located in the namespace **System.IO**.
 - We can distinguish two main types of streams – those who work with **binary data** and those who work with **text data**.
 - **Binary Stream**
 - **FileStream**, **BinaryReader** and **BinaryWriter**.
 - **Text Stream**
 - **TextReader** and **TextWriter**.
 - **StreamReader** and **StreamWriter**

□ **Binary Stream**

- The main classes that we use to read and write from and to binary streams are:
 - **FileStream:** provides us with various methods for reading and writing from a binary file.
 - **BinaryReader:** enables you to write primitive types and binary values in a specific encoding to a stream.
 - **BinaryWriter:** allows you to read primitive data types and binary values recorded using a BinaryWriter.

□ Text Streams

- The main classes for working with text streams in .NET are **TextReader** and **TextWriter**.
- The classes **StreamReader** and **StreamWriter** directly inherit the **TextReader** and **TextWriter** classes and implement functionality for reading and writing textual information to and from a file.
- To create an object of type **StreamReader** or **StreamWriter**, we need a file or a string, containing the file path.
 - Working with these classes, we can use all of the methods that we are already familiar with, to work with the console.



Classes File and Directory

18

- Information is stored in **files**, which are organized in **directories** (also called folders).
 - Classes File and Directory enable programs to **manipulate files and directories on disk**.
 - **Class File** can **determine information about files** and can be used to **open files** for **reading** or **writing**.
 - **Class Directory** provides capabilities for **manipulating directories**.
 - Each class contains a large set of **static methods** for both manipulation and

Static methods of File

Static Method	Description
AppendText	Returns a StreamWriter that appends text to an existing file or creates a file if one does not exist.
Copy	Copies a file to a new file.
Create	Creates a file and returns its associated FileStream.
CreateText	Creates a text file and returns its associated StreamWriter.
Delete	Deletes the specified file.
Exists	Returns true if the specified file exists and false otherwise.
GetCreationTime	Returns a DateTime object representing when the file was created.
GetLastAccessTime	Returns a DateTime object representing when the file was last accessed.
GetLastWriteTime	Returns a DateTime object representing when the file was last modified.
Move	Moves the specified file to a specified location.
Open	Returns a FileStream associated with the specified file and equipped with the specified read/write permissions.
OpenRead	Returns a read-only FileStream associated with the specified file.
OpenText	Returns a StreamReader associated with the specified file.
OpenWrite	Returns a read/write FileStream associated with the specified file.

Static methods of Directory

Static method	Description
CreateDirectory	Creates a directory and returns its associated DirectoryInfo object.
Delete	Deletes the specified directory.
Exists	Returns true if the specified directory exists and false otherwise.
GetDirectories	Returns a string array containing the names of the subdirectories in the specified directory.
GetFiles	Returns a string array containing the names of the files in the specified directory.
GetCreationTime	Returns a DateTime object representing when the directory was created.
GetLastAccessTime	Returns a DateTime object representing when the directory was last accessed.
GetLastWriteTime	Returns a DateTime object representing when items were last written to the directory.
Move	Moves the specified directory to a specified location.

Reading/Writing a Sequential Access File (Text File)

21

□ Reading to a Text File

□ Class for reading a Text file:

StreamReader

- The System.IO. StreamReader class provides the easiest way to read a text file, as it resembles reading from the console.

```
1 // Create a StreamReader connected to a file
  StreamReader reader = new StreamReader("test.txt");
  // Read the file here ...
  // Close the reader resource after you've finished using it
  reader.Close();
```

□ Full and Relative Paths

- When working with files we can use full paths (e.g. C:\Temp\example.txt) or relative paths, to the directory from which the application was started (e.g. ..\..\example.txt).
- Note that if the file is located in the folder where the compiled project (subdirectory **bin\Debug**) is, we can only provide its filename. Otherwise, we have to provide the full file path or relative path.

2. Reading a Text File Line by Line

- We use **ReadLine()** to read the line in the file.
- Other methods: **Read** and **ReadToEnd**

3. Closing the stream

- Once we have read everything we need from the file, we should not forget to close the object `StreamReader`, as to avoid loss of resources. For this, we use the method **Close()**.

□ Automatic Closing of the Stream after Working with It

- The correct way to handle the file closing is through the `using` keyword:
using () { ... }

- The C# construct `using(...)` ensures that after leaving its body, the method `Close()` will automatically be called. This will happen even if an exception occurs when

```
StreamReader reader = new StreamReader("Sample.txt");
using (reader)
{
    int lineNumber = 0;
    // Read first line from the text file
    string line = reader.ReadLine();
    // Read the other lines from the text file
    while (line != null)
    {
        lineNumber++;
        Console.WriteLine("Line {0}: {1}", lineNumber, line);
        line = reader.ReadLine();
    }
}
```

□ Writing to a Text File

□ Class for writing a Text file: **StreamWriter**

- The class StreamWriter is part of the System.IO namespace and is used exclusively for working with text data

1. **Open a Text File for writing**

- We can create an instance of the class the
`StreamWriter writer = new StreamWriter("test.txt");`

2. Writing a Text File Line by Line

- We use **WriteLine()** to write a line in the file.
- Other methods: **Write**
- Be sure to close the stream after you finish using it! The best way to dispose any unused resources is with the **using** construct in C#.

Example: File Writing

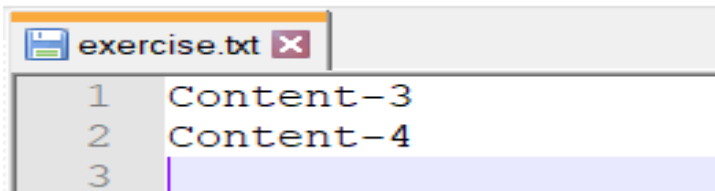
27

```
/*
 *   Write the following text inside first.txt
 *
 *       My favorite programming languages are:
 *       1. C#
 *       2. Java
 *       3. C++
 */
string firstFileName = "first.txt";
using (StreamWriter output = new StreamWriter(firstFileName))
{
    // new text for the file
    output.WriteLine("My favorite programming languages are:");
    // append the following text
    output.WriteLine("1. C#");
    // append the following text
    output.WriteLine("2. Java");
    // append the following text
    output.WriteLine("3. C++");
}
```

Exercise: What is the content of the exercise.txt file?

28

```
string exercise = "exercise.txt";
using (StreamWriter writer1 = new StreamWriter(exercise))
{
    // new text for the file
    writer1.WriteLine("Content-1");
    // append the following text
    writer1.WriteLine("Content-2");
}
using (StreamWriter writer2 = new StreamWriter(exercise, true))
{
    // new text for the file
    writer2.WriteLine("Content-3");
    // append the following text
    writer2.WriteLine("Content-4");
}
```



The screenshot shows a text editor window with the title 'exercise.txt'. The content of the file is displayed as follows:

1	Content-3
2	Content-4
3	

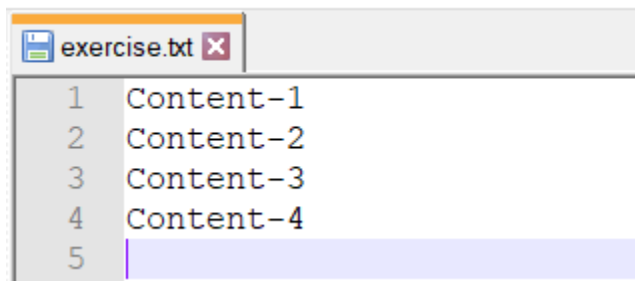
10/22/2024

StreamWriter: Appending Text with new writer object

29

- To append the text for the next stream writer we will use the second parameter of the StreamWriter constructor

```
using (StreamWriter writer2 = new StreamWriter(exercise, true))
{
    // new text for the file
    writer2.WriteLine("Content-3");
    // append the following text
    writer2.WriteLine("Content-4");
}
```



```
exercise.txt
1 Content-1
2 Content-2
3 Content-3
4 Content-4
5
```

□ Input / Output Exception Handling

- The most common exception when working with files is the **FileNotFoundException** (its name infers that the desired file was not found). It can occur when creating StreamReader.
- When setting a specified encoding by the creation of a StreamReader or a StreamWriter object, an **ArgumentException** can be thrown. This means, that the encoding we have chosen is not supported.
- Another common mistake is **IOException**. This is the base class for all IO errors when working with streams.

Exercise 1:

31

- Catch exceptions that occur when working with files.
- 1. A text file words.txt is given, enroll the numbers from 1 to 20, each number on a separate line in the given file.
- 2. Write a program that reads a text file and prints its odd lines on the console.
- 3. Write a program that joins two text files and records the results in a third file.

Random Access Files

32

- Sequential-access files are inappropriate for so-called “**instant-access**” applications, in which a particular record of information must be located immediately.
 - Popular instant-access applications include airline-reservation systems, banking systems, point-of-sale systems, automated-teller machines and other kinds of transaction-processing systems requiring **rapid access to specific data**.
- This type of instant access is made possible by **random-access files**.

Reading/Writing a Binary File

33

- **System.IO.BinaryWriter** and **System.IO.BinaryReader** can be used for writing and reading primitive data types.
 - byte, short, int, long, float, double, bool, char and also the data type string and StringBuilder to a stream (binary file manipulation).
- Class **FileStream** provides methods for writing streams to files and inherits from class Stream, so we can **pass the FileStream object as an argument to the BinaryWriter** constructor.
 - We can use the BinaryWriter to write bytes directly to the file

□ Writing Binary File

```
FileStream fs = new FileStream("info.txt",  
                               FileMode.OpenCreate, FileAccess.Write);  
BinaryWriter bw = new BinaryWriter(fs);  
bw.Write(123);  
bw.Write(56.78);  
bw.Write('A');  
bw.Write(true);  
bw.Close();  
fs.Close();
```

□ Reading Binary File

```
FileStream fs = new FileStream("info.txt",  
                               FileMode.OpenCreate, FileAccess.Read);  
BinaryReader br = new BinaryReader(fs);  
int i = br.ReadInt32();  
double d = br.ReadDouble();  
char c = br.ReadChar();  
bool b = br.ReadBoolean();  
br.Close();  
fs.Close();
```

- FileStream objects support random access to files using the **FileStream.Seek** method.
 - The Seek method allows the read/write position to be moved to any position within the file.
 - This is done with byte offset reference point parameters.
 - The byte offset is relative to the seek reference point, which can be the **beginning**, the **current** position, or the **end** of the underlying file, as represented by the three properties of the **SeekOrigin** class.

Seek Method

37

- The basic syntax for this method is:
Seek (value , SeekOrigin.Origin)
 - Where
 - **value** – is the point relative to Origin from which to begin seeking
 - **Origin** – specifies the position of the file pointer as a reference point for value, it could be three values: Begin, End and Current

Example

38

```
//Open the file for reading purpose:
using (FileStream read = new FileStream(filename, FileMode.Open))
{
    Console.WriteLine("In Word " + name);
    int first = read.ReadByte();
    Console.WriteLine("The First Character is: \"{0}\" ", (char)first);

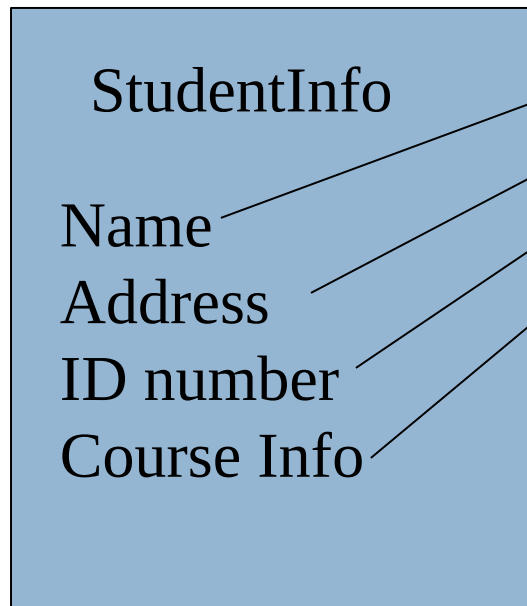
    long length = read.Length;
    //Move the File Ponter in the middle of the name:
    read.Seek(length / 2, SeekOrigin.Begin);
    int middle = read.ReadByte();
    Console.WriteLine("The Middle Character is: \"{0}\" ", (char)middle);

    //Move the file pointer before one character from the end:
    read.Seek(-1, SeekOrigin.End);
    int last = read.ReadByte();
    Console.WriteLine("The Last Character is: \"{0}\" ", (char)last);
}
```

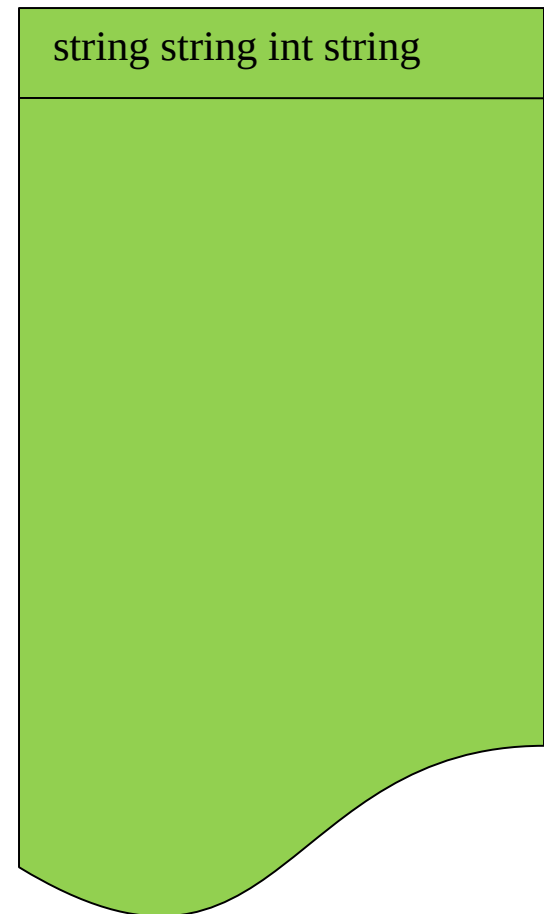
Serialization

- It is easy to write the individual fields of a *record* object to a file
- For example, we can create a record that stores information about a student
 - Name
 - Address
 - ID number
 - Course enrolled for
 - etc
- We can output each field of this record to a file (either text or binary)

- Often we want to store a complete object in a stream.
 - This is achieved by a process called object **serialization**.
 - By serialization we mean converting an object, or a connected graph of objects (a set of objects with some set of references to each other), stored within memory into a linear sequence of bytes.
 - This string of bytes contains all of the information that was held in the object started with.



Record file



- To serialize an object, the object class needs to be marked with the **[Serializable]** attribute or needs to implement the *ISerializable* interface
 - Requires the **System.Runtime.Serialization** namespace
- Also we require a **BinaryFormatter** object to serialize/de-serialize the object before writing to or reading from file
 - using
System.Runtime.Serialization.Formatters.Binary;
 - using System.Runtime.Serialization;

Writing and Reading an Object

43

□ Writing Objects

- `BinaryFormatter formatter= new BinaryFormatter();`
- `FileStream output =new FileStream(fileName, FileMode.Open, FileAccess.Write );`
- ...
- `formatter.Serialize( output, record);`

□ Reading an object

- `BinaryFormatter reader = new BinaryFormatter();`
- `FileStream input=new FileStream(fileName, FileMode.Open, FileAccess.Read );`
- `Object obj= reader.Deserialize( input );`

Example

44

```
[Serializable]
class Course
{
    string courseTitle, courseCode;
    public Course()
    {
    }
    public Course(string title, string code)
    {
        courseTitle = title;
        courseCode = code;
    }
    static void Main(string[] args)
    {
        Course c1 = new Course("Introduction to Programming", "CS200");
        Course c2 = new Course("Introduction to Database", "CS201");
        ... ←
        Console.ReadKey();
    }
}
```

10/22/2024

... writing

45



```
...  
BinaryFormatter formatter = new BinaryFormatter();  
FileStream output = new FileStream(fileName, FileMode.OpenOrCreate,  
                                   FileAccess.ReadWrite);  
try  
{  
    formatter.Serialize(output, c1);  
    formatter.Serialize(output, c2);  
}  
catch (SerializationException e)  
{  
    Console.WriteLine(e.Message.ToString());  
}  
finally  
{  
    output.Close();  
}
```

... reading

46

...
↓

```
BinaryFormatter bfl = new BinaryFormatter();  
FileStream fs1 = new FileStream(fileName, FileMode.  
OpenOrCreate, FileAccess.ReadWrite);  
try  
{  
    while (fs1.Position!=fs1.Length)  
    {  
        Student s = (Student)bfl.Deserialize(fs1);  
        Console.WriteLine("First Name:" + s.Name);  
        Console.WriteLine("First Name:" + s.Id);  
    }  
}  
catch (SerializationException)  
{  
    Console.WriteLine("Error");  
}  
finally  
{  
    fs1.Close();  
}
```

Exercise 2:

47

2. (File of Student Grades) Create a program that stores student grades in a text file. The file should contain the name, ID number, class taken and grade of every student.
 - The entries should be displayed in the following format:
LastName, FirstName: ID# Class Grade

Some sample data below:

Jones, Bob: 1 "Introduction to Computer Science" "A-"

Johnson, Sarah: 2 "Data Structures" "B+"

Smith, Sam: 3 "Data Structures" "C"

Exercise 3:

48

3. (File of Student Grades: **Serialization**)
Create a program that stores student objects in a text file. The file should contain the name, ID number, class taken and grade of every student.
 - The entries should be displayed in the following format:
LastName, FirstName: ID# Class Grade

Some sample data below:

Jones, Bob: 1 "Introduction to Computer Science" "A-"

Johnson, Sarah: 2 "Data Structures" "B+"

Smith, Sam: 3 "Data Structures" "C"

Questions?

Thank You