عنوان مضمون

Visual Programming-II

توسط : صفری

بهار1398

Visual Programming-II

- **Course Code:** 78031
- **Lecturer:** Baba Ali Safari
- **Course Credit:** 3
- **Class Hours:** 48
- **Theory:** 2
- **Practice:** 1

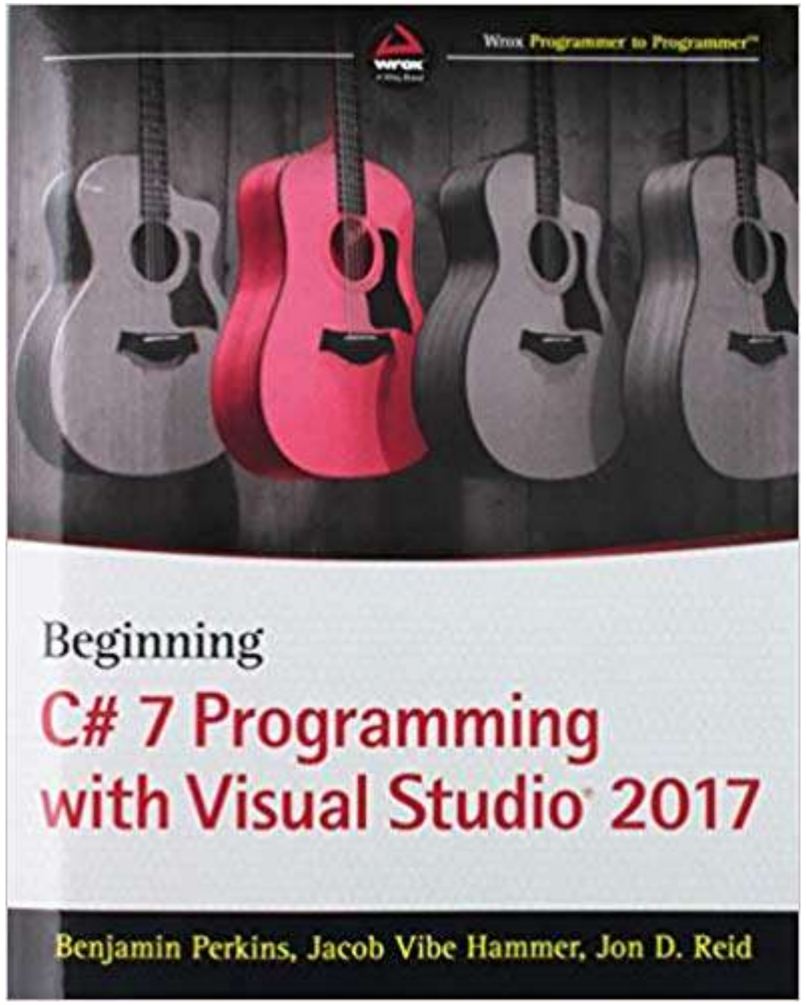
Textbooks:

BEGINNING C# Programming with Visual Studio 2017

Benjamin Perkins

Jacob Vibe Hammer

Jon D. Reid


Wrox Programmer to Programmer™
Beginning
C# 7 Programming with Visual Studio® 2017
Benjamin Perkins, Jacob Vibe Hammer, Jon D. Reid

Topic to be Covered:

- C# File I/O (Week 1)
- Class library (Week 2)
- windows services (Week 3)
- Events (Week 4)
- Collections (Week 5)
- Generics (Week 6)
- WCF services (Week 7)
- Mid Term (Week 8)
- Multithreading (Week 9)
- Connecting to database using ADO.net (Week 10)
- Basic Desktop Programming (Week 11)
- Basic Desktop Programming (Week 12)
- Basic Desktop Programming (Week 13)
- C# Advanced Desktop Programming (CREATING AND STYLING CONTROLS, THE MAIN WINDOW) (Week 14)
- C# Advanced Desktop Programming (Animations, WPF USER CONTROLS) (Week 15)
- Overview (Week 16)

Grading Policy:

- Lab Assignments and Quizzes: 20%
- Midterm Exam: 20%
- Project: 20%
- Final Exam: 40%

File I/O

File I/O

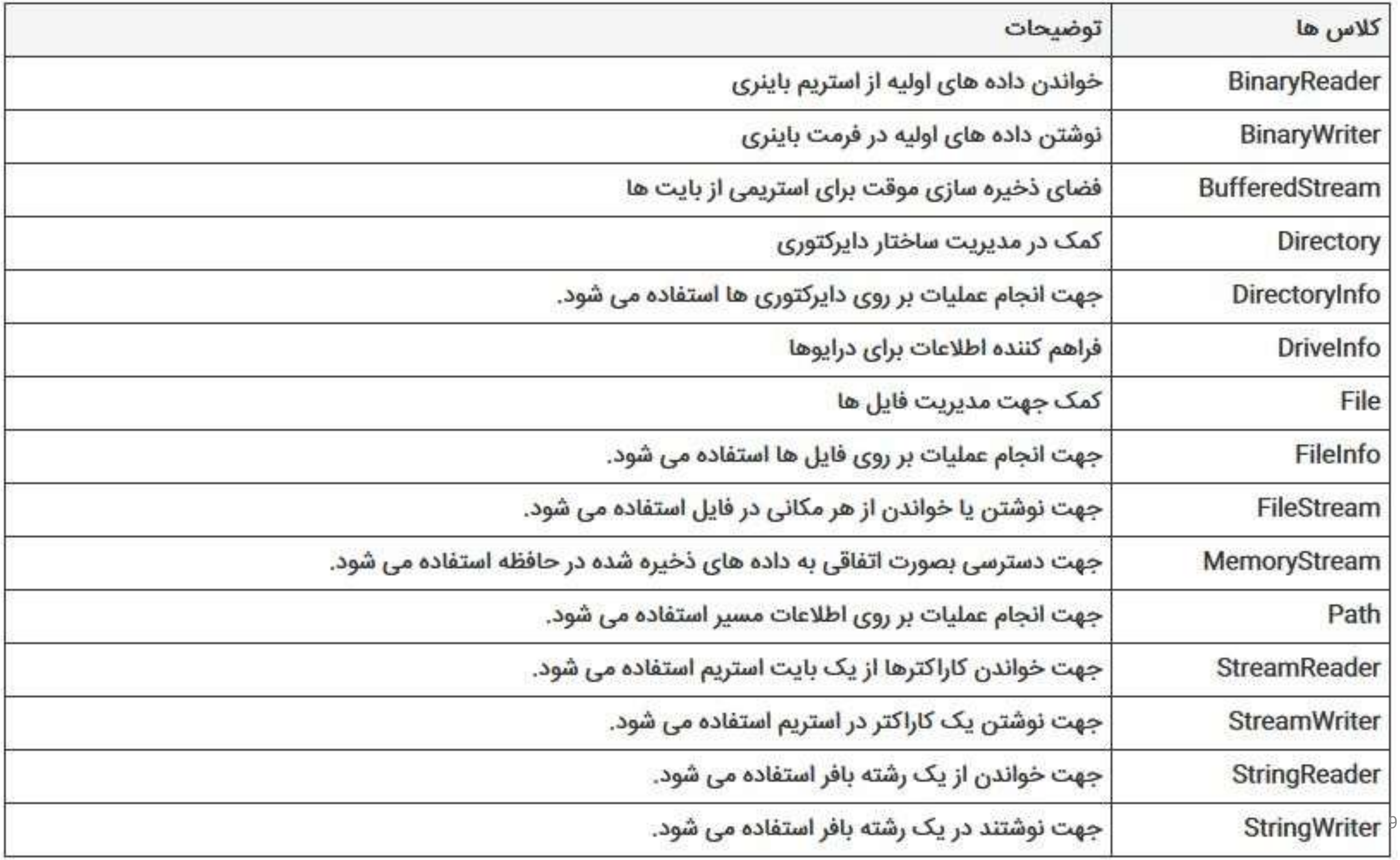
A **file** is a collection of data stored in a disk with a specific name and a directory path. When a file is opened for reading or writing, it becomes a **stream**.

The stream is basically the sequence of bytes passing through the communication path. There are two main streams: the **input stream** and the **output stream**. The **input stream** is used for reading data from file (read operation) and the **output stream** is used for writing into the file (write operation).

C# I/O Classes

- The System.IO namespace has various classes that are used for performing numerous operations with files, such as creating and deleting files, reading from or writing to a file, closing a file etc.

- The following table shows some commonly used non-abstract classes in the System.IO namespace −



کلاس ها	توضیحات
BinaryReader	خواندن داده های اولیه از استریم باینری
BinaryWriter	نوشتن داده های اولیه در فرمت باینری
BufferedStream	فضای ذخیره سازی موقت برای استریمی از بایت ها
Directory	کمک در مدیریت ساختار دایرکتوری
DirectoryInfo	جهت انجام عملیات بر روی دایرکتوری ها استفاده می شود.
DriveInfo	فراهم کننده اطلاعات برای درایوها
File	کمک جهت مدیریت فایل ها
FileInfo	جهت انجام عملیات بر روی فایل ها استفاده می شود.
FileStream	جهت نوشتن یا خواندن از هر مکانی در فایل استفاده می شود.
MemoryStream	جهت دسترسی بصورت اتفاقی به داده های ذخیره شده در حافظه استفاده می شود.
Path	جهت انجام عملیات بر روی اطلاعات مسیر استفاده می شود.
StreamReader	جهت خواندن کاراکترها از یک بایت استریم استفاده می شود.
StreamWriter	جهت نوشتن یک کاراکتر در استریم استفاده می شود.
StringReader	جهت خواندن از یک رشته بافر استفاده می شود.
StringWriter	جهت نوشتند در یک رشته بافر استفاده می شود.

Example

```
System.IO.File.WriteAllText("D:\\Test.txt", " The following table shows
some commonly ");
```

```
List<string> lines = new List<string>();
lines.Add("This is line1");
lines.Add("This is line2");
System.IO.File.WriteAllLines("d:\\file.txt", lines);
```

Example

```
var text = "Hello ITPro.ir";
var textBytes = System.Text.Encoding.UTF8.GetBytes(text);
System.IO.File.WriteAllBytes("D:\\file.txt", textBytes);
```

Example

- var text = System.IO.File.ReadAllText("D:\\file.txt");

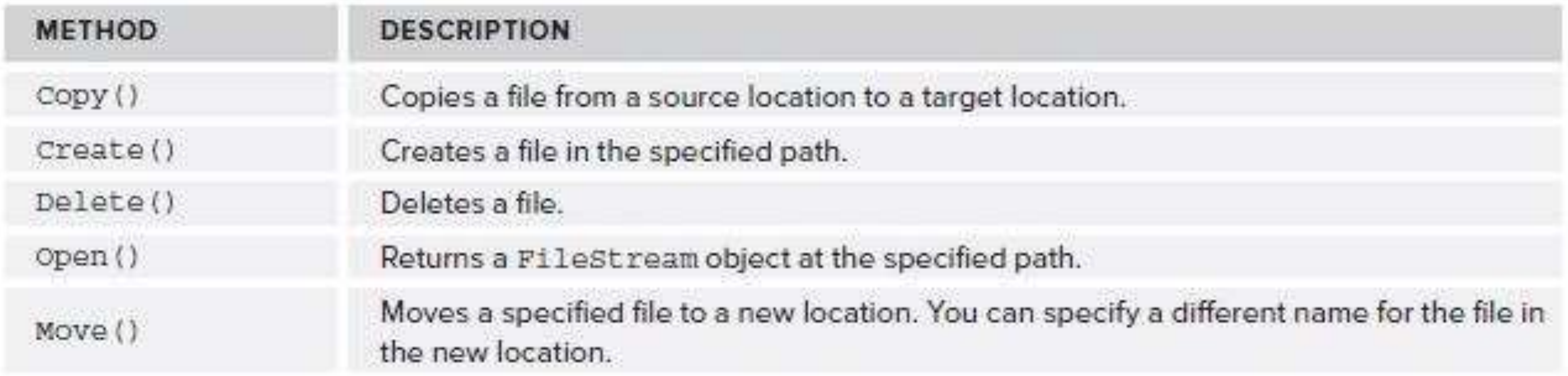
- var lines = System.IO.File.ReadAllLines("D:\\file.txt");

```
foreach (var line in lines)
{
    Console.WriteLine(line)
}
```

Save Files and open Files Using the SaveFileDialog and the openFileDialog Components

- Display the **Save File** dialog box and call a method to save the file selected by the user
- Display the **Open File** dialog box and call a method to Open the file selected by the user

Methods of the File Class



METHOD	DESCRIPTION
`Copy()`	Copies a file from a source location to a target location.
`Create()`	Creates a file in the specified path.
`Delete()`	Deletes a file.
`Open()`	Returns a `FileStream` object at the specified path.
`Move()`	Moves a specified file to a new location. You can specify a different name for the file in the new location.

Example

- `System.IO.File.Copy("d:\\file.txt", "e:\\copy.txt");`
- `System.IO.File.Move("d:\\file.txt", "e:\\copy.txt");`
-
```
if (System.IO.File.Exists("d:\\data.dat"))
{
        // your code
}
```
- `System.IO.File.Delete("d:\\data.dat");`

The FileStream Class

The **FileStream** class in the System.IO namespace helps in reading from, writing to and closing files. This class derives from the abstract class Stream.

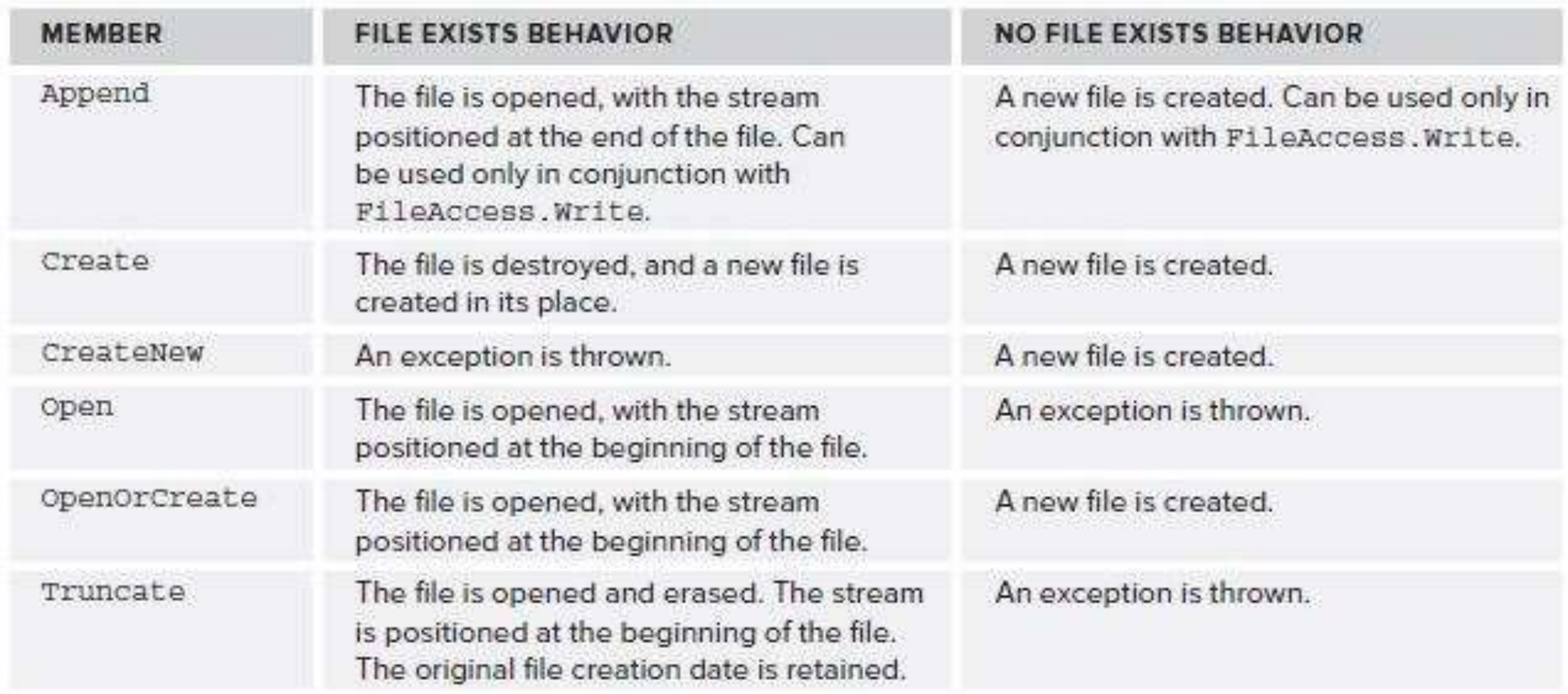
You need to create a **FileStream** object to create a new file or open an existing file. The syntax for creating a **FileStream** object is as follows −

```
FileStream aFile = new FileStream(filename, FileMode.<Member>,
FileAccess.<Member>);
```

For example, we create a FileStream object **F** for reading a file named **sample.txt as shown** −

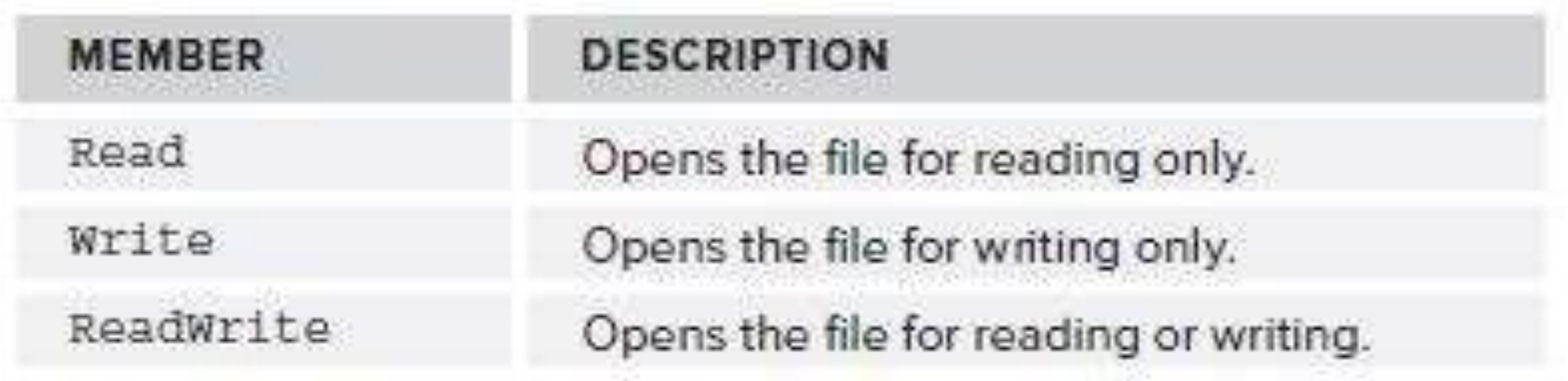
```
FileStream F = new FileStream("sample.txt", FileMode.Open,
FileAccess.Read);
```

FileMode Enumeration Members



MEMBER	FILE EXISTS BEHAVIOR	NO FILE EXISTS BEHAVIOR
`Append`	The file is opened, with the stream positioned at the end of the file. Can be used only in conjunction with `FileAccess.Write`.	A new file is created. Can be used only in conjunction with `FileAccess.Write`.
`Create`	The file is destroyed, and a new file is created in its place.	A new file is created.
`CreateNew`	An exception is thrown.	A new file is created.
`Open`	The file is opened, with the stream positioned at the beginning of the file.	An exception is thrown.
`OpenOrCreate`	The file is opened, with the stream positioned at the beginning of the file.	A new file is created.
`Truncate`	The file is opened and erased. The stream is positioned at the beginning of the file. The original file creation date is retained.	An exception is thrown.

FileAccess Enumeration Members



MEMBER	DESCRIPTION
`Read`	Opens the file for reading only.
`Write`	Opens the file for writing only.
`ReadWrite`	Opens the file for reading or writing.

Example

The following program demonstrates use of the FileStream class −

```
using System;
using System.IO;

namespace FileIOApplication {
  class Program {
    static void Main(string[] args) {
      FileStream F = new FileStream("test.txt", FileMode.OpenOrCreate,
        FileAccess.ReadWrite);

      for (int i = 1; i <= 20; i++) {
        F.WriteByte((byte)i);
      }
      F.Position = 0;
      for (int i = 0; i <= 20; i++) {
        Console.Write(F.ReadByte() + " ");
      }
      F.Close();
      Console.ReadKey();
    }
  }
}
```

When the above code is compiled and executed, it produces the following result –

```
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 -1
```

File Position

- The FileStream class maintains an internal file pointer that points to the location within the file where the next read or write operation will occur.
- In most cases, when a file is opened, it points to the beginning of the file, but this pointer can be modified. This enables an application to read or write anywhere within the file, which in turn enables random access to a file and the capability to jump directly to a specific location in the file.
- This can save a lot of time when dealing with very large files because you can instantly move to the location you want.

File Position

- The method that implements this functionality is the **Seek()** method, which takes two parameters.
- The first parameter specifies how far to move the file pointer, in bytes.
- The second parameter specifies where to start counting from, in the form of a value from the **SeekOrigin** enumeration.
- The **SeekOrigin** enumeration contains three values: Begin, Current, and End.

File Position

- For example, the following line would move the fi le pointer to the eighth byte in the file, starting from the very first byte in the file:
- aFile.Seek(8, SeekOrigin.Begin);

- The following line would move the file pointer two bytes forward, starting from the current position. If this were executed directly after the previous line, then the file pointer would now point to the tenth byte in the file:
- aFile.Seek(2, SeekOrigin.Current);

Example

```
private void save_Click(object sender, EventArgs e)
      {
         saveFileDialog1.ShowDialog();
          String s = saveFileDialog1.FileName;
          FileStream f = new FileStream(s,FileMode.Append,FileAccess.Write);
          f.Seek(10,System.IO.SeekOrigin.End);
          byte[] info = new UTF8Encoding(true).GetBytes(textbox.Text);
          f.Write(info,0,info.Length);
          f.Close();
      }
```