Занятие 5, ч2. Строки, регулярные выражения, ввод-вывод и работа с файловой системой

Тренер: Алексей Дышлевой

Основные вопросы

- Робота з рядками. Клас String. Рядкові літерали. Специфікатори формату й глобалізація. Object.ToString, IFormattable, ICustomFormatter та CultureInfo. Порівняння рядків.
- Регулярні вирази. Пошук та групування. Заміна тексту за допомогою Regex.
- Клас Console. Метод ReadLine. Метод ReadKey. Метод Read. Метод WriteLine. Метод Write
- Файловий ввід-вивід. Класи StreamReader та StreamWriter. Класи FileAccess та FileAttribute. Класи Directory та DirectoryInfo, Path, FileInfo и DriveInfo. Класи BinaryReader та BinaryWriter. Класи StringReader та StringWriter. Класи TextReader та TextWriter. Класи XmlReader та XmlWriter. Клас ToBase64Transform. Класи FileStream, BufferedStream та MemoryStream
- Серіалізація та десеріалізація

Работа с файлами

- Простейший механизм работы с файлами и каталогами реализуется с помощью классов File, FileInfo, Directory, DirectoryInfo, Path

Класс File

- Класс File это служебный класс, предоставляющий различные связанные с файлами функции, доступные через статические методы

Класс File

Метод	Описание	Пример кода
AppendAllText	Позволяет открыть существующий файл, добавить текст в этот файл, а затем закрыть файл, выполнив все действия в одну операцию.	string filePath = "..."; string fileContents = "..."; File.AppendAllText(filePath, fileContents);
Copy	Позволяет скопировать существующий файл в новое место.	string sourceFile = "..."; string destFile = "..."; bool overwrite = false; File.Copy(sourceFile,destFile, overwrite);
Create	Позволяет создать новый файл файловой системы Windows. Метод возвращает объект FileStream, который позволяет взаимодействовать с файлом с помощью потоковой модели.	string filePath = "..."; int bufferSize = 128; FileStream file = File.Create(filePath,bufferSize, FileOptions.None);
Delete	Позволяет удалить файл из файловой системы Windows.	string filePath = "..."; File.Delete(filePath);
Exists	Позволяет определить, существует ли указанный файл.	string filePath = "..."; bool exists = File.Exists(filePath);
GetCreationTime	Позволяет получить время создания файла.	string filePath = "..."; DateTime time = File.GetCreationTime(filePath);
GetLastAccessTime	Позволяет получить время последнего доступа к файлу.	string filePath = "..."; DateTime time = File.GetLastAccessTime(filePath);
Move	Позволяет переместить файл в новое место. Этот метод можно также использовать для переименования файлов.	string sourceFile = "..."; string destFile = "..."; File.Move(sourceFile,destFile);
ReadAllText	Позволяет читать весь текст из файла в строковую переменную.	string filePath = "..."; string fileContents = File.ReadAllText(filePath);
SetCreationTime	Позволяет установить время создания файла.	string filePath = "..."; File.SetCreationTime(filePath, DateTime.Now);
SetLastAccessTime	Позволяет установить время последнего доступа к файлу	string filePath = "..."; File.SetLastAccessTime(filePath, DateTime.Now);
WriteAllText	Позволяет создать новый файл, записать текст в этот файл, а затем закрыть файл, выполнив все действия в одну операцию.	string filePath = "..."; string fileContents = "..."; File.WriteAllText(filePath, fileContents);

Класс FileInfo

- Класс FileInfo предоставляет несколько свойств и экземплярных методов, которые позволяют создавать, копировать, перемещать файлы и обрабатывать их содержимое. При создании экземпляра класса FileInfo, необходимо указать путь к файлу в файловой системе.
- string filePath = @"C:\Temp\SomeFile.txt";
- FileInfo file = new FileInfo(filePath);

- Объект FileInfo можно использовать в качестве оболочки для файла, который предоставляет различные данные и функции через свойства и методы. Класс FileInfo можно также использовать для создания новых файлов.

Класс FileInfo

Член	Описание	Пример кода
CreationTime (свойство)	Позволяет получить или установить время создания для конкретного файла.	string filePath = "..."; FileInfo file = new FileInfo(filePath); file.CreationTime = DateTime.Now; ... DateTime time = file.CreationTime;
CopyTo (метод)	Позволяет копировать файл в новое место в файловой системе.	string filePath = "..."; FileInfo file = new FileInfo(filePath); string destPath = "..."; file.CopyTo(destPath);
Delete (свойство)	Позволяет удалить файл.	string filePath = "..."; FileInfo file = new FileInfo(filePath); file.Delete();
DirectoryName (свойство)	Позволяет получить путь к каталогу файла.	string filePath = "..."; FileInfo file = new FileInfo(filePath); string dirPath = file.DirectoryName;
Exists (свойство)	Позволяет определить, существует ли заданный файл.	string filePath = "..."; FileInfo file = new FileInfo(filePath); bool exists = file.Exists;
Extension (свойство)	Позволяет получить расширение файла.	string filePath = "..."; FileInfo file = new FileInfo(filePath); string ext = file.Extension;
Length (свойство)	Позволяет получить длину файла в байтах.	string filePath = "..."; FileInfo file = new FileInfo(filePath); long length = file.Length;
Name (свойство)	Позволяет получить имя файла.	string filePath = "..."; FileInfo file = new FileInfo(filePath); string name = file.Name;
Open (метод)	Позволяет открыть файл файловой системы Windows. Метод возвращает объект FileStream, который позволяет взаимодействовать с файлом с помощью потоковой модели.	string filePath = "..."; FileInfo file = new FileInfo(filePath); FileStream stream = file.Open(FileMode.OpenOrCreate);

Чтение и запись в файлы

- Класс File содержит статические методы, которые можно использовать для выполнения атомарных операций прямого чтения и записи в файлы. Эти методы атомарны, поскольку оборачивают несколько основных функций в один вызов метода.
- При использовании класса File для чтения данных из файла, существует много альтернативных методов, которые можно использовать, каждый из которых предлагает различное поведение.
- Класс FileInfo содержит экземплярные методы, которые при чтении и записи в файлы полагаются на классы FileStream и StreamReader.

Чтение из файлов

- ReadAllBytes позволяет прочитать содержимое файла в виде данных типа byte, которые сохраняются в массив
  - string filePath = “someFile.txt";
  - byte[] data = File.ReadAllBytes(filePath);
- ReadAllLines позволяет прочитать текстовый файл от начала до конца, строка за строкой, и сохранить каждую строку в массив строк
  - string filePath = “someFile.txt";
  - string[] lines = File.ReadAllLines(filePath);
- ReadAllText позволяет читать файл от начала до конца и сохранять данные из файла в строковую переменную
  - string filePath = “someFile.txt";
  - string data = File.ReadAllText(filePath);

Запись в файлы

- AppendAllLines используется для записи содержимого массива строк в текстовый файл. Если указанный путь не существует, будет создан новый файл
  - string filePath = «someFile.txt";
  - string[] fileLines = { "Line 1", "Line 2", "Line 3" };
  - File.AppendAllLines(filePath, fileLines);
- AppendAllText используется для записи содержимого строковой переменной в текстовый файл
  - string fileContents = “Writing to a file called someFile.txt";
  - File.AppendAllText(filePath, fileContents);
- WriteAllBytes используется для записи содержимого массива в двоичный файл. Если файл уже существует, содержимое файла будет перезаписано
  - byte[] fileBytes = { 12, 134, 12, 8, 32 };
  - File.WriteAllBytes(filePath, fileBytes);

Запись в файлы

- WriteAllLines ведет себя аналогично методу AppendAllLines, т.е. позволяет записывать содержимое массива строк в текстовый файл. Основным отличием является то, что, если файл существует, его содержимое будет записано заново. Если файл не существует, будет создан новый файл
  - string[] fileLines = { "Line 1", "Line 2", "Line 3" };
  - File.WriteAllLines(filePath, fileLines);
- WriteAllText ведет себя аналогично методу AppendAllText, т.е. записывает содержимое строки переменной в текстовый файл. Основным отличием является то, что, если файл существует, файл будет перезаписан. Если файл не существует, будет создан новый файл
  - string fileContents = "Writing to a file called someFile.txt"";
  - File.WriteAllText(filePath, fileContents);

Задача

- Записать в файл текст, числа целого и действительного типов. Прочитать эти данные из файла.

Класс Directory

- Класс Directory, как и класс File, является служебным классом, который обеспечивает различные операции, позволяющие управлять папками и каталогами. Класс Directory предоставляет свои функции в рамках статических методов

Класс Directory

Метод	Описание	Пример кода
CreateDirectory	Позволяет создавать все еще не существующие каталоги, указанные в пути.	string dirPath = @"C:\NewFolder\SubFolder"; Directory.CreateDirectory(dirPath);
DeleteDirectory	Позволяет удалить один или несколько каталогов файловой системы.	string dirPath = @"C:\Users\Student\" + "MyDirectory"; bool deleteSubFolders = true; Directory.Delete(dirPath,deleteSubFolders);;
GetDirectories	Позволяет получить все подкаталоги по указанному пути.	string dirPath = "..."; string[] dirs = Directory.GetDirectories(dirPath);
GetFiles	Позволяет получить все файлы по указанному пути.	string dirPath = "..."; string[] files = Directory.GetFiles(dirPath);
Exists	Позволяет определить, существует ли каталог по указанному пути.	string dirPath = "..."; bool dirExists = Directory.Exists(dirPath);
Move	Позволяет переместить каталог. Нельзя использовать метод для перемещения каталогов с разных носителей.	string sourcePath = "..."; string destPath = "..."; Directory.Move(sourcePath, destPath);

Класс DirectoryInfo

- Класс DirectoryInfo предоставляет несколько свойств и экземплярных методов, которые позволяют работать с каталогами. Как и в классе FileInfo, при создании экземпляра класса DirectoryInfo, обычно указывается путь к каталогу файловой системы
  - string dirPath = @"C:\Users\User\Documents\";
  - DirectoryInfo dir = new DirectoryInfo (dirPath);
- Объект DirectoryInfo можно использовать в качестве обертки для каталога, который предоставляет различные данные и функции через свойства и методы. Класс DirectoryInfo можно также использовать для создания нового каталога.
  - string dirPath = @"C:\Users\User\Documents\";
  - DirectoryInfo dir = new DirectoryInfo(dirPath);
  - if (!dir.Exists)
  - {
  - dir.Create();
  - }

Класс DirectoryInfo

Член	Описание	Пример кода
Create (метод)	Позволяет создавать каталоги по указанному пути. Если каталог уже существует, метод игнорируется.	string dirPath = "..."; DirectoryInfo dir = new DirectoryInfo(dirPath); dir.Create();
Delete (метод)	Позволяет удалить несколько каталогов. Если каталог не может быть найден, генерируется исключение DirectoryNotFoundException.	string dirPath = "..."; DirectoryInfo dir = new DirectoryInfo(dirPath); dir.Delete();
Exists (свойство)	Позволяет определить, существует ли каталог по указанному пути.	string dirPath = "..."; DirectoryInfo dir = new DirectoryInfo(dirPath); bool exists = dir.Exists;
FullName (свойство)	Позволяет получить полный путь к каталогу.	string dirPath = "..."; DirectoryInfo dir = new DirectoryInfo(dirPath); string fullName = dir.FullName;
GetDirectories (метод)	Позволяет получить все подкаталоги по указанному пути. Этот метод возвращает массив DirectoryInfo, который позволяет использовать каждый из членов DirectoryInfo во всех подкаталогах.	string dirPath = "..."; DirectoryInfo dir = new DirectoryInfo (dirPath); DirectoryInfo[] dirs = dir.GetDirectories();
GetFiles (метод)	Позволяет получить все файлы по указанному пути. Этот метод возвращает массив FileInfo, который позволяет использовать каждый из членов FileInfo во всех файлах каталога.	string dirPath = "..."; DirectoryInfo dir = new DirectoryInfo(dirPath); FileInfo[] files = dir.GetFiles();
MoveTo (метод)	Позволяет переместить каталог. Нельзя использовать метод для перемещения каталогов с разных носителей.	string dirPath = "..."; DirectoryInfo dir = new DirectoryInfo(dirPath); string destPath = "..."; dir.MoveTo(destPath);
Name (свойство)	Позволяет получить имя каталога.	string dirPath = "..."; DirectoryInfo dir = new DirectoryInfo(dirPath); string dirName = dir.Name;
Parent (свойство)	Позволяет получить родительский каталог.	string dirPath = @"C:\Users\Student\Music\"; DirectoryInfo dir = new DirectoryInfo(dirPath); DirectoryInfo parentDir = dir.Parent;

Пример

- Перечисление и демонстрация подробной информации о всех подкаталогах и файлах, которые они содержат.
- string dirPath = @"C:\Users\User\Documents\";
- // Get all sub directories in the Documents directory.
- string[] subDirs = Directory.GetDirectories(dirPath);
- foreach (string dir in subDirs)
- {
- // Display the directory name.
- Console.WriteLine("{0} contains the following files:", dir);
- // Get all the files in each directory.
- string[] files = Directory.GetFiles(dir);
- foreach (string file in files)
- {
- // Display the file name.
- Console.WriteLine(file);
- }
- }

Задача

- Создать 2 директории в некоторой корневой папке. Переместить одну директорию в другую. Переместить в эту директорию существующий файл.

Управление путями

- Файлы хранятся в папках. Все файлы и папки имеют названия. Сочетание имени файла и папки, в которой он находится, представляют собой путь к этому файлу. Различные файловые системы могут иметь различные соглашения и правила, согласно которым установлены легальные файл и путь к файлу. Класс Path содержит методы, которые можно использовать для анализа и построения файлов и имен папок для указанной файловой системы.

Класс Path

Метод	Описание	Пример кода
GetDirectoryName	Позволяет получить все каталоги по указанному пути.	string path = @"C:\Temp\SubFolder\MyFile.txt"; string dirs = Path.GetDirectoryName(path);
GetExtension	Позволяет получить расширение указанного файла.	string path = @"C:\Temp\SubFolder\MyFile.txt"; string ext = Path.GetExtension(path);
GetFileName	Позволяет получить имя файла, включая расширение по указанному пути.	string path = @"C:\Temp\SubFolder\MyFile.txt"; string fileName = Path.GetFileName(path);
GetFileNameWithoutExtension	Позволяет получить имя файла без расширения по указанному пути.	string path = @"C:\Temp\SubFolder\MyFile.txt"; string fileName = Path.GetFileNameWithoutExtension(path);
GetRandomFileName	Позволяет генерировать случайные папки или файлы.	string fileName = Path.GetRandomFileName();
GetTempFileName	Позволяет создать новый временный файл в локальной временной папке Windows. Метод возвращает абсолютный путь к файлу.	string tempFilePath = Path.GetTempFileName();
GetTempPath	Позволяет получить путь к локальной временной папке Windows.	string tempPath = Path.GetTempPath();

Задача

- Создать дерево директорий от одной корневой. У каждой директории должно быть 3 вложенных, глубина вложения – 3 уровня. В каждую директорию скопировать некоторый файл.

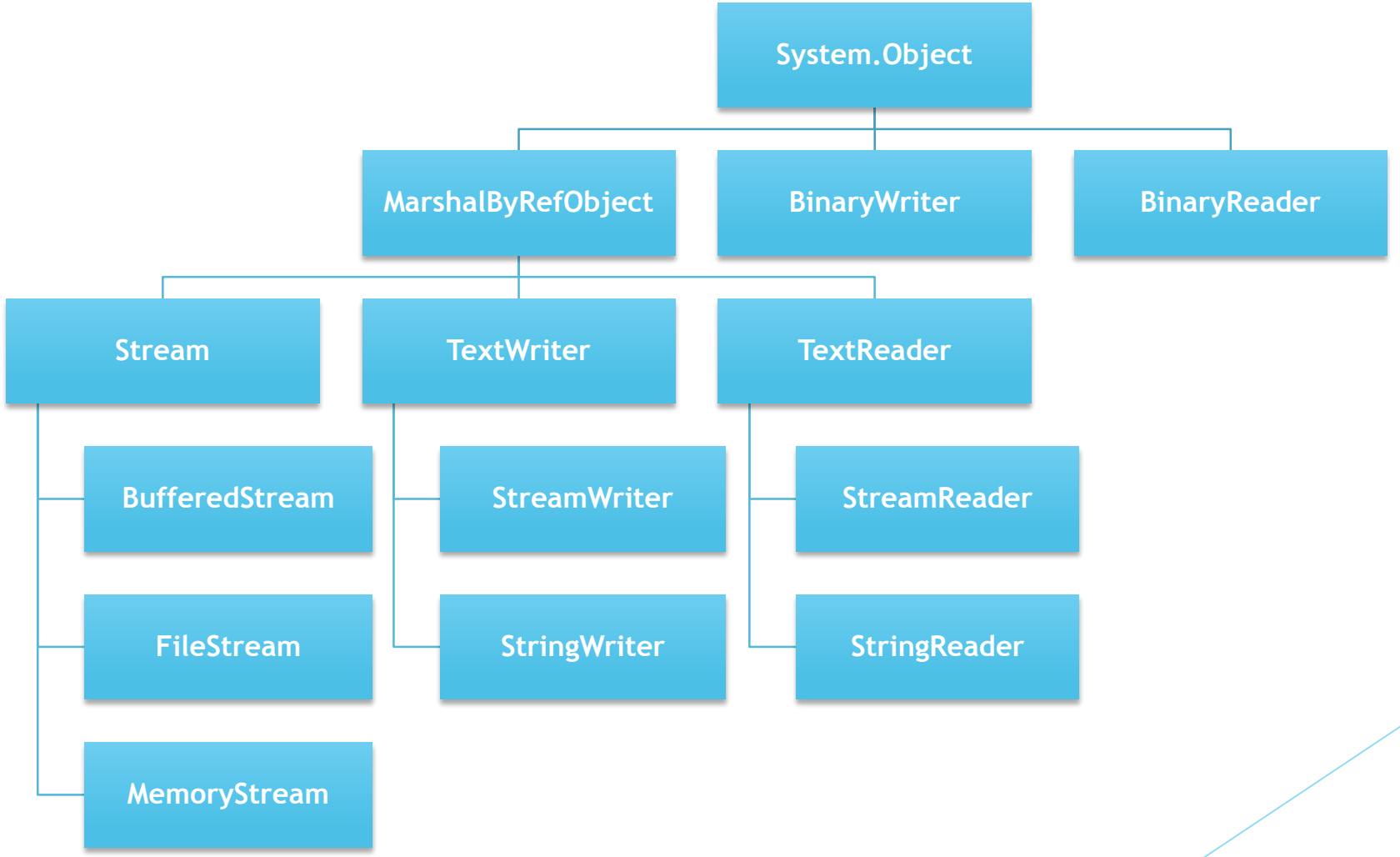
Файловый ввод-вывод

- При работе с данными, данные иногда становится слишком большими, чтобы загружать их в память и передавать одной атомарной операцией
- Для решения этой проблемы .NET Framework позволяет использовать потоки.
- **Поток** представляет собой последовательность байтов, которые могут поступать из файла файловой системы, сети связи или памяти. Потоки позволяют считывать или записывать данные в источник данных посредством небольших управляемых пакетов данных. Потоки обеспечивают следующие операции:
  - Чтение из потока – это перенос информации из потока в структуру данных, такую как массив байтов.
  - Запись в поток – это передача данных из структуры данных в поток.
  - Поиск в потоке – это выяснение и изменение текущей позиции внутри потока. Возможность поиска зависит от вида резервного хранилища потока. Например, в сетевых потоках отсутствует унифицированное представление текущего положения, поэтому обычно они не поддерживают поиск.

Классы для работы с потоками

- .NET Framework предоставляет несколько потоковых классов, позволяющих работать с различными данными и источниками данных. Для выбора какого-либо из них, необходимо учитывать следующее:
  - Какой тип данных читается или записывается, например, двоичный или буквенно-цифровой.
  - Где хранятся данные, например, в локальной файловой системе, в памяти или на веб-сервере в сети.
- Внутренне, объект Stream поддерживает указатель, который ссылается на текущее местоположение в источнике данных. Когда впервые строится объект Stream для источника данных, этот указатель позиционируется перед первым байтом. Когда происходит чтение или запись данных, класс Stream продвигает этот указатель к концу данных, которые читаются или записываются. Класс Stream нельзя использовать напрямую. Вместо этого, нужно реализовать специализации этого класса, которые оптимизированы для выполнения потокового ввода/вывода для конкретных типов источников данных. Таким образом, в каждом резервном хранилище (резервное хранилище – устройство хранения информации, например диск или память) используется собственный поток как реализация класса Stream.

Классы для работы с потоками



Чтение и запись двоичных данных

- Поток, который устанавливается с помощью объекта FileStream всего лишь сырая последовательность байтов. Если файл содержит структурированные данные, необходимо преобразовать последовательность байтов в соответствующие типы. Это может быть трудоемкой, подверженной ошибкам задачей.
- Библиотека классов .NET Framework содержит классы, используемые для чтения и записи текстовых данных и примитивных типов в поток, который открыт с помощью объекта FileStream.
- Эти классы включают StreamReader, StreamWriter, BinaryReader и BinaryWriter.

Класс FileStream

- Это элементарный поток, который может записывать или читать только один байт или массив байтов.
- Класс имеет функционал для работы с файлами, но этот процесс часто бывает трудоемким.
- Недостаток FileStream – необходимость оперировать низкоуровневыми байтами.
- Для упрощения работы с файлами существуют другие классы, которые реализуют файловое взаимодействие с текстовыми данными и другими типами .Net.

Классы BinaryReader и BinaryWriter

- Многие приложения хранят данные в сырой бинарной форме, поскольку двоичная запись быстрая и занимает мало места на диске
- BinaryReader и BinaryWriter наследуются непосредственно от System.Object. Позволяют читать и записывать дискретные типы данных в потоки в двоичном формате.

- string filePath = "...";
- FileStream file = new FileStream(filePath);
- ...
- BinaryReader reader = new BinaryReader(file);
- ...
- BinaryWriter writer = new BinaryWriter(file);

Классы BinaryReader и BinaryWriter

	BinaryReader
Член	Описание
BaseStream (свойство)	Получает доступ к базовому потоку, используемому объектом BinaryReader.
Close (метод)	Закрывает объект BinaryReader и базовый поток.
Read (метод)	Выполняет чтение символов из базового потока и перемещает текущую позицию в потоке вперед в соответствии с используемой кодировкой и конкретным символом в потоке, чтение которого выполняется в настоящий момент.
ReadByte (метод)	Считывает из текущего потока следующий байт и перемещает текущую позицию в потоке на один байт вперед.
ReadBytes (метод)	Считывает указанное количество байтов из текущего потока в массив байтов и перемещает текущую позицию на это количество байтов.

	BinaryWriter
Член	Описание
BaseStream (свойство)	Получает доступ к базовому потоку, который используется объектом BinaryWriter.
Close (метод)	Закрывает объект BinaryWriter и базовый поток. Любые данные из буфера сбрасываются в базовый поток.
Flush (метод)	Явно флеширует какие-либо данные из текущего буфера в основной поток.
Seek (метод)	Устанавливает позицию в текущем потоке, таким образом, чтобы писать в конкретные байты.
Write (метод)	Записывает данные в поток, продвигаясь в нем. Метод предоставляет несколько перегрузок, позволяющих записывать все примитивные типы данных в поток.

Пример

```
using System;
using System.IO;
 class ConsoleApplication
{
    const string fileName = "AppSettings.dat";
    static void Main()     {
       WriteDefaultValues();
       DisplayValues();
    }
    public static void WriteDefaultValues()     {
        using (BinaryWriter writer = new BinaryWriter(File.Open(fileName, FileMode.Create)))
       {
           writer.Write(1.250F);
           writer.Write(@"c:\Temp");
           writer.Write(10);
           writer.Write(true);
       }
    }
    public static void DisplayValues()     {
       float aspectRatio;
       string tempDirectory;
       int autoSaveTime;
       bool showStatusBar;
       if (File.Exists(fileName))
       {
           using (BinaryReader reader = new BinaryReader(File.Open(fileName, FileMode.Open)))
           {
               aspectRatio = reader.ReadSingle();
               tempDirectory = reader.ReadString();
               autoSaveTime = reader.ReadInt32();
               showStatusBar = reader.ReadBoolean();
           }
           Console.WriteLine("Aspect ratio set to: " + aspectRatio);
           Console.WriteLine("Temp directory is: " + tempDirectory);
           Console.WriteLine("Auto save time set to: " + autoSaveTime);
           Console.WriteLine("Show status bar: " + showStatusBar);
      }
   }
 }
```

Классы StreamWriter и StreamReader

- Аналогично использованию классов BinaryReader и BinaryWriter, при инициализации классов StreamReader или StreamWriter следует предоставить объект потока для обработки взаимодействия с источником данных, как показано в следующем примере. Наследуються от абстрактных классов TextWriter и TextReader.
- string destinationFilePath = "...";
- FileStream file = new FileStream(destinationFilePath);
- ...
- StreamReader reader = new StreamReader(file);
- ...
- StreamWriter writer = new StreamWriter(file);

StreamReader

Член	Описание
Close (метод)	Закрывает объект StreamReader и базовый поток.
EndOfStream (свойство)	Определяет, достигнут ли конец потока.
Peek (метод)	Получает следующий доступный символ в потоке, но не использует его.
Read (метод)	Выполняет чтение следующего символа из входного потока и перемещает положение символа на одну позицию вперед.
ReadBlock (метод)	Выполняет чтение из текущего потока блок символов начиная с указанного индекса.
ReadLine (метод)	Выполняет чтение строки символов из текущего потока и возвращает данные в виде строки.
ReadToEnd (метод)	Считывает поток от текущего положения до конца.

Пример

```
try
{
 // Create an instance of StreamReader to read from a file.
// The using statement also closes the StreamReader.
   using (StreamReader sr = new StreamReader("TestFile.txt"))
   {
      string line;
     // Read and display lines from the file until the end of
     // the file is reached.
     while ((line = sr.ReadLine()) != null)
     {
        Console.WriteLine(line);
     }
   }
}
catch (Exception e)
 {
    // Let the user know what went wrong.
    Console.WriteLine("The file could not be read:");
    Console.WriteLine(e.Message);
 }
```

StreamWriter

Член	Описание
AutoFlush (свойство)	Получает или задает значение, определяющее, будет ли StreamWriter флешировать буфер в основной поток после каждого вызова StreamWriter.Write.
Close (метод)	Закрывает текущий объект StreamWriter и базовый поток.
Flush (метод)	Очищает все буферы для текущего средства записи и вызывает запись всех данных буфера в основной поток.
NewLine (свойство)	Получает или задает признак конца строки, используемой текущим TextWriter.
Write (метод)	Записывает данные в поток и продвигается в нем.
WriteLine (метод)	Позволяет записывать данные в поток, за которыми следует признак конца строки, и продвинуться в потоке.

Пример

```
DirectoryInfo[] cDirs = new DirectoryInfo(@"c:\").GetDirectories();
 // Write each directory name to a file.
 using (StreamWriter sw = new StreamWriter("CDriveDirs.txt"))
 {
    foreach (DirectoryInfo dir in cDirs)
    {
       sw.WriteLine(dir.Name);
    }
 }
 // Read and show each line from the file.
 string line = "";
 using (StreamReader sr = new StreamReader("CDriveDirs.txt"))
 {
    while ((line = sr.ReadLine()) != null)
    {
      Console.WriteLine(line);
    }
 }
```

Другие классы для работы с файлами

- StringWriter/ StringReader – позволят обращаться с текстовой информацией, как с потоком символов в памяти (позволяет работать с данными в буфере, не создавая файл)
- TextWriter/ TextReader – данные рассматриваются как последовательный поток символов (т.е., как текст)
- XmlWriter/ XmlReader – прямой доступ к данным XML
- BufferedStream – буферизация в операциях чтения и записи в потоки
- MemoryStream – создается поток, резервным хранилищем которого является память

- FileSystemWatcher – используется для программного отслеживания состояния файлов в системе

Задача

- Создать класс студент с полями: имя, фамилия, дата рождения, курс, средний балл.
- Записать данные о нескольких студентах в файл, и прочитать из него.
- Реализовать работу с файлами в бинарной и текстовой форме.

Сериализация

- Сериализация – процесс сохранения объектов в долговременной памяти (файлах) в период выполнения системы
- Десериализация – обратный процесс, восстановление состояния объектов, хранимых в долговременной памяти
- Поддерживаются 2 формата сохранения данных: бинарный и XML-файл.
- Бинарная сериализация:
  - using System.Runtime.Serialization;
  - using System.Runtime.Serialization.Formatters.Binary;
  - BinaryFormatter bf = new BinaryFormatter();
- XML сериализация:
  - using System.Xml.Serialization;
  - XmlSerializer bf = new XmlSerializer(this.GetType());

Пример 1 (сериализация)

Задача

- Записать данные о студентах в файл и прочитать с помощью сериализации

Граф объектов сериализации

- При сериализации объектов производных классов, данные из базовых классов тоже автоматически сериализуются.
- Для этого используется граф объектов, куда записываются все связи.
- Каждый объект в графе получает уникальное числовое значение.

- Например, если существует базовый класс Person, от которого наследуются Student и Worker. Классу Person буде присвоено значение 1, Student – 2, Worker – 3. Граф будет иметь следующий вид:
- [Person 1, ref 2, ref 3], [Student 2], [Worker 3]

Конфигурирование объектов для сериализации

- Используются атрибуты [Serialazible] и [NonSerialized]

Особенности XML-сериализации

- При использовании XMLSerializer не все данные класса будут сериализироваться.
- Это закрытые (private) поля, для которых не реализованы свойства.

SOAP сериализация

- SoapFormatter сохраняет состояние объекта в виде сообщения SOAP (стандартный XML-формат для передачи и приема сообщений об веб-служб).
- using System.Runtime.Serialization.Formatters.Soap;
- Находится в отдельной сборке. По этому сначала надо установить ссылку на System.Runtime.Serialization.Formatters.Soap.dll

Интерфейс IFormatter

- IFormatter определяет основные методы Serialaie(), Deserialazible()

Интерфейс ISerialazible

- Позволяет объекту управлять его собственной сериализацией и десериализацией.
- GetObjectData - Заполняет объект SerializationInfo данными, необходимыми для сериализации целевого объекта
- Метод вызывается форматером автоматически во время процесса сериализации
- Кроме того, должен определятся конструктор со специальной сигнатурой
- protected SomeClass (SerializationInfo si, StreamingContext ctx) {…}

Пример 2 – реализация ISerialazible

Задача

- Реализовать интерфейс ISerialazible для класса студент

Диалоговые окна файловой системы

- Для создание диалогового окна открытия или сохранения файла .NET Framework предоставляет классы OpenFileDialog и SaveFileDialog
- Классы OpenFileDialog и SaveFileDialog обеспечивают функциональность, позволяющую пользователю просматривать файл или указывать имя файла, а также создавать любые требуемые папки. Функциональные возможности этих классов доступны через различные свойства и методы, которые можно использовать для настройки поведения диалоговых окон в соответствии с требованиями.
- Однако, не диалоговое окно фактически открывает или сохраняет указанный файл; все это делается построением пути и имени файла, которые приложение может использовать, чтобы открыть или сохранить файл.

Свойства OpenFileDialog и SaveFileDialog

Свойства	Описание
CheckFileExists	Позволяет поручить диалоговое окно, отображающее предупреждение, если пользователь указывает не существующий файл.
FileName	Позволяет получить или задать путь к файлу, который выбран в диалоговом окне.
Filter	Позволяет ограничить типы файлов, которые пользователь может выбрать в диалоговом окне.
InitialDirectory	Позволяет получить или установить каталог по умолчанию, который отображается, когда диалоговое окно показывается впервые.
Title	Позволяет указать заголовок для диалогового окна.

Пример

- // Создание
- OpenFileDialog openDlg = new OpenFileDialog();
- ...
- SaveFileDialog saveDlg = new SaveFileDialog();
- ...
- openDlg.Title = "Browse for a file to open";
- openDlg.Multiselect = false;
- openDlg.InitialDirectory = @"C:\Users\User\Documents";
- openDlg.Filter = "Word (*.doc) | *.doc;";
- ...
- saveDlg.Title = "Browse for a save location";
- saveDlg.DefaultExt = "doc";
- saveDlg.AddExtension = true;
- saveDlg.InitialDirectory = @"C:\Users\User\Documents";
- saveDlg.OverwritePrompt = true;
- //Для появления диалогового окна во время работы приложения необходимо вызвать метод ShowDialog:
- ...
- openDlg.ShowDialog();
- ...
- saveDlg.ShowDialog();
- //чтобы получить путь, выбранный пользователем, запросить свойство FileName:
- ...
- string selectedFileName = openDlg.FileName;
- ...
- string selectedFileName = saveDlg.FileName;

Домашнее задание

- Создать класс для бинарной и XML сериализации и десериализации данных из класса клиентов банка.
- Реализовать собственную сериализацию и десериализацию с помощью ISertialazible
- Создать класс для поиска текстовых файлов по ключевым словам