Урок 6

Делегаты. Файлы. Коллекции

Первое слово о делегатах. Учимся считывать и обрабатывать большие объёмы данных. Знакомимся с коллекциями.

Делегаты

Делегат представляет собой объект, который может ссылаться на метод. Следовательно, когда создается делегат, в итоге получается объект, содержащий ссылку на метод. Метод можно вызывать по этой ссылке. Иными словами, делегат позволяет вызывать метод, на который он ссылается.

По сути, делегат — это безопасный в отношении типов объект, указывающий на другой метод (или, возможно, список методов) приложения, который может быть вызван позднее.

В частности — объект делегата поддерживает три важных фрагмента информации:

- адрес метода, на котором он вызывается;
- аргументы (если есть) этого метода;
- возвращаемое значение (если есть) этого метода.

Как только делегат создан и снабжен необходимой информацией, он может во время выполнения динамически вызывать методы, на которые указывает.

Пример. Создадим метод, который будет выводить значения некоторых функций от a до b. Чтобы иметь возможность применять метод с различными функциями, используем механизм делегатов.

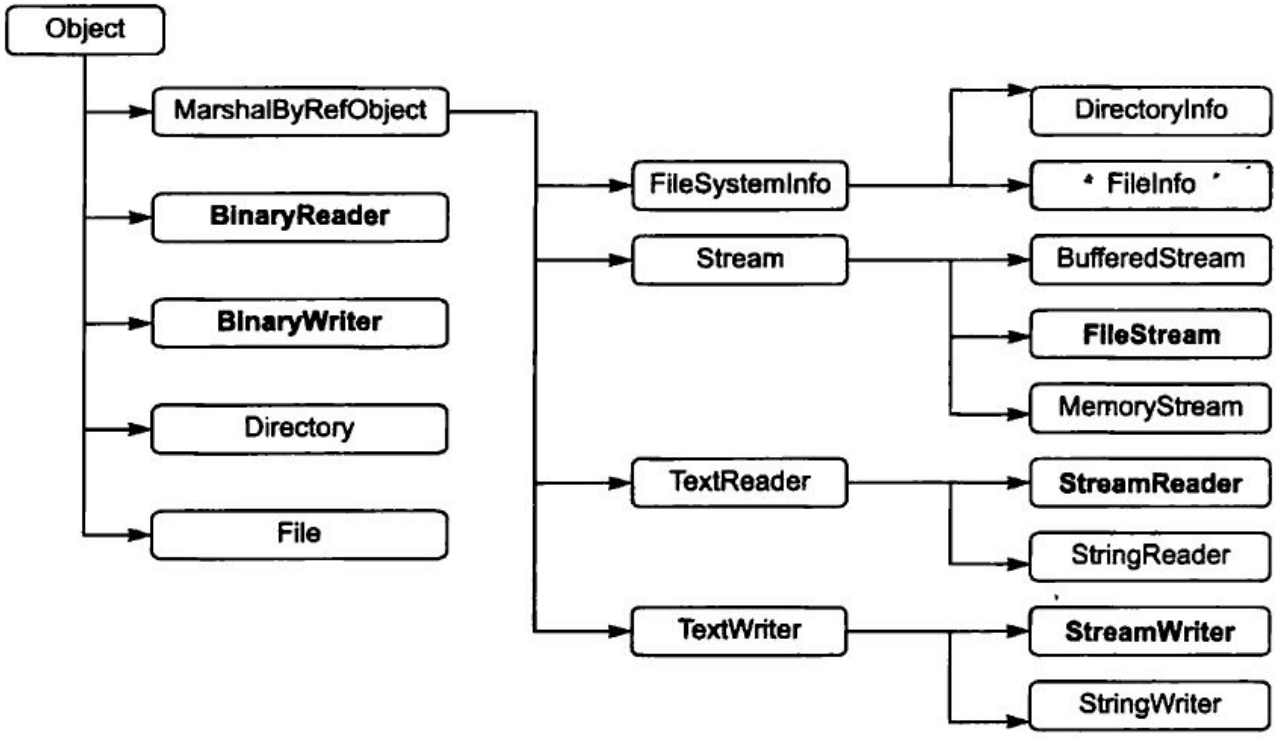
```
using System;
        // Описываем делегат. В делегате описывается сигнатура методов, на
        // которые он сможет ссылаться в дальнейшем (хранить в себе)
    public delegate double Fun(double x);

    class Program
    {
        // Создаем метод, который принимает делегат
        // На практике этот метод сможет принимать любой метод
        // с такой же сигнатурой, как у делегата
        public static void Table(Fun F, double x, double b)
        {
            Console.WriteLine("----- X ----- Y -----");
            while (x <= b)
            {
                Console.WriteLine("| {0,8:0.000} | {1,8:0.000} |", x, F(x));
               x += 1;
            }
            Console.WriteLine("----------------------");
        }
        // Создаем метод для передачи его в качестве параметра в Table
        public static double MyFunc(double x)
        {
            return x * x * x;
        }

        static void Main()
        {
// Создаем новый делегат и передаем ссылку на него в метод Table
            Console.WriteLine("Таблица функции MyFunc:");
// Параметры метода и тип возвращаемого значения, должны совпадать с делегатом
            Table(new Fun(MyFunc),-2,2);
              Console.WriteLine("Еще раз та же таблица, но вызов организован по
новому");
```

```
// Упрощение(с C# 2.0).Делегат создается автоматически.
            Table(MyFunc, -2, 2);
            Console.WriteLine("Таблица функции Sin:");
               Table(Math.Sin, -2, 2);        // Можно передавать уже созданные
методы
            Console.WriteLine("Таблица функции x^2:");
// Упрощение(с C# 2.0). Использование анонимного метода
            Table(delegate (double x) { return x * x; }, 0, 3);
        }
    }
```

Это небольшое знакомство с делегатами нам понадобится немного позже, а сейчас перейдём к изучению возможностей C# по работе с файловой системой.

Организация системы ввода-вывода

C#-программы выполняют операции ввода-вывода посредством потоков, которые построены на иерархии классов. Поток (stream) — это абстракция, которая генерирует и принимает данные. С помощью потока можно читать данные из различных источников (клавиатура, файл) и записывать в различные источники (принтер, экран, файл). Несмотря на то, что потоки связываются с различными физическими устройствами, характер поведения всех потоков одинаков. Поэтому классы и методы ввода-вывода можно применить ко многим типам устройств.



На самом низком уровне иерархии потоков ввода-вывода находятся потоки, оперирующие байтами. Это объясняется тем, что многие устройства при выполнении операций ввода-вывода ориентированы на байты. Однако для человека привычнее оперировать символами, поэтому разработаны символьные потоки, которые фактически представляют собой оболочки, выполняющие преобразование байтовых потоков в символьные и наоборот. Кроме этого, реализованы потоки для работы с int-, double-, short- значениями, которые также представляют оболочку для байтовых

потоков, но работают не с самими значениями, а с их внутренним представлением в виде двоичных кодов.

Центральную часть потоковой C#-системы занимает класс Stream пространства имен System.IO. Класс Stream представляет байтовый поток и является базовым для всех остальных потоковых классов.

Из класса Stream выведены такие байтовые классы потоков, как:

1. FileStream — байтовый поток, разработанный для файлового ввода-вывода.
2. BufferedStream заключает байтовый поток в оболочку и добавляет буферизацию, которая во многих случаях увеличивает производительность программы.
3. MemoryStream — байтовый поток, который использует память для хранения данных.

Программист может вывести собственные потоковые классы. Однако для подавляющего большинства приложений достаточно встроенных потоков.

Подробно мы рассмотрим класс FileStream, классы StreamWriter и StreamReader, представляющие собой оболочки для FileStream и позволяющие преобразовывать байтовые потоки в символьные. Также рассмотрим классы BinaryWriter и BinaryReader, представляющие собой оболочки для FileStream и позволяющие преобразовывать байтовые потоки в двоичные для работы с int-, double-, short- и т. д.

Байтовый поток

Чтобы создать байтовый поток, связанный с файлом, создается объект класса FileStream. При этом в классе определено несколько конструкторов. Чаще всего используется конструктор, который открывает поток для чтения и/или записи:

```
FileStream(string filename, FileMode mode)
```

где:

1) Параметр filename определяет имя файла, с которым будет связан поток ввода-вывода данных; при этом filename определяет либо полный путь к файлу, либо имя файла, который находится в папке bin/debug вашего проекта.

2) Параметр mode определяет режим открытия файла, который может принимать одно из возможных значений, определённых перечислением FileMode:

a) FileMode.Append предназначен для добавления данных в конец файла.
b) FileMode.Create предназначен для создания нового файла, при этом, если существует файл с таким же именем, он будет предварительно удален.
c) FileMode.CreateNew предназначен для создания нового файла, при этом файл с таким же именем не должен существовать.
d) FileMode.Open предназначен для открытия существующего файла.
e) FileMode.OpenOrCreate открывает файл, если тот существует, в противном случае создает новый.
f) FileMode.Truncate открывает существующий файл, но усекает его длину до нуля.

Другая версия конструктора позволяет ограничить доступ только чтением или только записью:

```
FileStream(string filename, FileMode mode, FileAccess how)
```

где:

1) Параметры filename и mode имеют то же назначение, что и в предыдущей версии конструктора.
2) Параметр how определяет способ доступа к файлу и может принимать одно из значений, определённых перечислением FileAccess:
   a) FileAccess.Read — только чтение;
   b) FileAccess.Write — только запись;
   c) FileAccess.ReadWrite — и чтение, и запись.

После установления связи байтового потока с физическим файлом внутренний указатель потока устанавливается на начальный байт файла.

Для чтения очередного байта из потока, связанного с физическим файлом, используется метод ReadByte(). После прочтения очередного байта внутренний указатель перемещается на следующий байт файла. Если достигнут конец файла, метод ReadByte() возвращает значение -1.

Для побайтовой записи данных в поток используется метод WriteByte().

После работы файл необходимо закрыть. Для этого достаточно вызвать метод Close(). При закрытии файла освобождаются системные ресурсы, ранее выделенные для него. Это даёт возможность использовать их для работы с другими файлами.

Рассмотрим пример использования класса FileStream для копирования одного файла в другой. Но вначале создадим текстовый файл data.txt в папке bin/debug текущего проекта и внесём в него произвольную информацию, например:

Привет!
1: Сообщение
1234567890
3,14

```
using System;
using System.IO;// для работы с потоками ввода-вывода
class Program
{
  static void Main()
  {
    try
    {
            FileStream fileIn = new FileStream("data.txt", FileMode.Open,
FileAccess.Read);
         FileStream fileOut = new FileStream("newData.txt", FileMode.Create,
FileAccess.Write);
    int i;
    while ((i = fileIn.ReadByte())!=-1)
    {
              //  запись очередного байта в поток, связанный с файлом fileOut
      fileOut.WriteByte((byte)i);
    }
    fileIn.Close();
    fileOut.Close();
  }
```

```
  catch (Exception exc)
  {
      Console.WriteLine(exc.Message);
  }
  Console.WriteLine("Файл успешно скопирован");
  }
}
```

Символьный поток

Чтобы создать символьный поток, нужно поместить объект класса Stream (например, FileStream) «внутрь» объекта класса StreamWriter или объекта класса StreamReader. В этом случае байтовый поток будет автоматически преобразовываться в символьный.

Класс StreamWriter предназначен для организации выходного символьного потока. В нем определено несколько конструкторов. Один из них записывается следующим образом:

```
StreamWriter(Stream stream);
```

Параметр stream определяет имя уже открытого байтового потока. Например, создать экземпляр класса StreamWriter можно следующим образом:

```
StreamWriter fileOut=new StreamWriter(new FileStream("text.txt",
FileMode.Create, FileAccess.Write));
```

Этот конструктор генерирует исключение типа ArgumentException, если поток stream не открыт для вывода, и исключение типа ArgumentNullException, если поток имеет null-значение.

Другой вид конструктора позволяет открыть поток сразу через обращение к файлу, где параметр name определяет имя открываемого файла:

```
StreamWriter(string name);
```

Например, обратиться к данному конструктору можно следующим образом:

```
StreamWriter fileOut=new StreamWriter("c:\\temp\\data.txt");
```

И еще один вариант конструктора StreamWriter:

```
StreamWriter(string name, bool appendFlag);
```

Здесь параметр name определяет имя открываемого файла. Параметр appendFlag может принимать значение true, если нужно добавлять данные в конец файла, или false, если файл необходимо перезаписать.

Например:

```
StreamWriter fileOut=new StreamWriter("t.txt", true);
```

Теперь для записи данных в поток fileOut можно обратиться к методу WriteLine. Это можно сделать следующим образом:

```
fileOut.WriteLine("test");
```

В данном случае в конец файла t.txt будет дописано слово test.

Класс StreamReader предназначен для организации входного символьного потока. Один из его конструкторов выглядит следующим образом:

```
StreamReader(Stream stream);
```

Параметр *stream* определяет имя уже открытого байтового потока.

Этот конструктор генерирует исключение типа ArgumentException, если поток *stream* не открыт для ввода.

Например, создать экземпляр класса StreamReader можно следующим образом:

```
StreamReader fileIn = new StreamReader(new FileStream("text.txt", FileMode.Open,
FileAccess.Read));
```

Как и в случае со StreamWriter, у класса StreamReader есть и другой вид конструктора, который позволяет открыть файл напрямую:

```
StreamReader (string name);
```

Параметр name определяет имя открываемого файла.

Обратиться к данному конструктору можно следующим образом:

```
StreamReader fileIn=new StreamReader ("c:\\temp\\data.txt");
```

В C# символы реализуются кодировкой Unicode. Чтобы было можно обрабатывать текстовые файлы, содержащие кириллические символы, созданные, например, в Блокноте, рекомендуется вызывать следующий вид конструктора StreamReader:

```
StreamReader fileIn=new StreamReader ("c:\temp\data.txt", Encoding.GetEncoding(1251));
```

Параметр Encoding.GetEncoding(1251) говорит о том, что будет выполняться преобразование из кода Windows-1251 (одна из модификаций кода ASCII, содержащая кириллические символы) в Unicode.Encoding.GetEncoding(1251), реализованном в пространстве имён System.Text.

Теперь для чтения данных из потока fileIn можно воспользоваться методом ReadLine. При этом, если будет достигнут конец файла, метод ReadLine вернёт значение null.

Рассмотрим пример, в котором данные из одного файла копируются в другой, но уже с использованием классов StreamWriter и StreamReader.

Данный способ копирования одного файла в другой даст нам тот же результат, что и при использовании байтовых потоков. Однако его работа менее эффективна, так как тратится дополнительное время на преобразование байтов в символы. Но у символьных потоков есть свои преимущества. Например, мы можем использовать регулярные выражения для поиска заданных фрагментов текста в файле.

```
using System;
using System.Text.RegularExpressions;
using System.IO;
// Поиск всех чисел в файле
    class Program
    {
        static void Main(string[] args)
        {
            StreamReader fileIn = new StreamReader("hobbit.txt");
            StreamWriter fileOut = new StreamWriter("numbers.txt", false);
            string text = fileIn.ReadToEnd();
            Regex r = new Regex(@"[-+]?\d+");
            Match integer = r.Match(text);
            while (integer.Success)
            {
                Console.WriteLine(integer);
                fileOut.WriteLine(integer);
                integer = integer.NextMatch();
        }
            fileIn.Close();
            fileOut.Close();
            Console.ReadKey();
        }
    }
```

Двоичные потоки

Двоичные файлы хранят данные в том же виде, в котором они представлены в оперативной памяти, то есть во внутреннем представлении. Двоичные файлы не применяются для просмотра человеком, они используются только для программной обработки.

Выходной поток BinaryWriter поддерживает произвольный доступ, то есть имеется возможность выполнять запись в произвольную позицию двоичного файла. Наиболее важные методы потока BinaryWriter:

Член класса	**Описание**
BaseStream	Определяет базовый поток, с которым работает объект BinaryWriter
Close	Закрывает поток
Flush	Очищает буфер
Seek	Устанавливает позицию в текущем потоке
Write	Записывает значение в текущий поток

Наиболее важные методы выходного потока BinaryReader:

Член класса	**Описание**
BaseStream	Определяет базовый поток, с которым работает объект BinaryReader
Close	Закрывает поток
PeekChar	Возвращает следующий символ потока без перемещения внутреннего указателя в потоке
Read	Считывает очередной поток байтов или символов и сохраняет в массиве, передаваемом во входном параметре
ReadBoolean, ReadByte, ReadInt32 и т.д	Считывает из потока данные определённого типа

Двоичный поток открывается на основе базового потока (например, FileStream). При этом двоичный поток будет преобразовывать байтовый поток в значения int-, double-, short- и т.д.

Рассмотрим пример формирования двоичного файла.

Попытка просмотреть двоичный файл через текстовый редактор неинформативная. Двоичный файл просматривается программным путём, например, следующим образом (должен быть создан файл data.bin).

```
using System;
using System.IO;
class Program
{
static void Main()
{
      FileStream f=new FileStream("data.bin",FileMode.Open);
      BinaryReader fIn=new BinaryReader(f);
      long n=f.Length/4; // определяем количество чисел в двоичном потоке

      int a;
      for (int i=0; i<n; i++)
      {
            a=fIn.ReadInt32();
            Console.Write(a+" ");
      }
      fIn.Close();
      f.Close();

  }

}

}
```

Двоичные файлы являются файлами с произвольным доступом, при этом нумерация элементов в двоичном файле ведется с нуля. Произвольный доступ обеспечивает метод Seek. Рассмотрим его синтаксис:

```
Seek(long newPos, SeekOrigin pos)
```

где параметр newPos определяет новую позицию внутреннего указателя файла в байтах относительно исходной позиции указателя, которая определяется параметром pos. В свою очередь, параметр pos должен быть задан одним из значений перечисления SeekOrigin:

Значение	**Описание**
SeekOrigin.Begin	Поиск от начала файла

SeekOrigin.Current	Поиск от текущей позиции указателя
SeekOrigin.End	Поиск от конца файла

После вызова метода Seek последующие операции чтения или записи будут выполняться с новой позиции внутреннего указателя файла.

Рассмотрим пример организации произвольного доступа к двоичному файлу:

```
using System;
using System.IO;
    class Program
    {
        static void Main()
        {
                                                                // изменение данных в
двоичном потоке
            FileStream f = new FileStream("data.dat", FileMode.Open);
            BinaryWriter fOut = new BinaryWriter(f);
               long n = f.Length;                // определяем количество байт в
байтовом потоке
            int a;
             for (int i = 0; i < n; i += 8) // сдвиг на две позиции, т.к. тип int
занимает 4 байта
            {
                fOut.Seek(i, SeekOrigin.Begin);
                fOut.Write(0);
            }
            fOut.Close();
                                                    // чтение данных из двоичного
потока
            f = new FileStream("data.dat", FileMode.Open);
            BinaryReader fIn = new BinaryReader(f);
               n = f.Length / 4;                // определяем количество чисел в
двоичном потоке
            for (int i = 0; i < n; i++)
            {
                a = fIn.ReadInt32();
                Console.Write(a + " ");
            }
            fIn.Close();
            f.Close();
        }
    }
```

Работа с файловой системой

В пространстве имен System.IO предусмотрено четыре класса, которые предназначены для работы с файловой системой компьютера, то есть для создания, удаления, переноса и т.д. файлов и каталогов.

Первые два типа — **Directory** и **File** — реализуют свои возможности с помощью *статических методов*, поэтому данные классы можно использовать без создания соответствующих объектов (экземпляров классов).

Классы **DirectoryInfo** и **FileInfo** обладают схожими функциональными возможностями с Directory и File, но требуют создания соответствующих экземпляров классов.

Класс FileInfo

Рассмотрим некоторые свойства класса:

Свойство	**Описание**
Attributes	Позволяет получить или установить атрибуты для данного объекта файловой системы. Для этого свойства используются значения и перечисления FileAttributes
CreationTime	Позволяет получить или установить время создания объекта файловой системы
Exists	Может быть использовано, чтобы определить, существует ли данный объект файловой системы
Extension	Позволяет получить расширение для файла
FullName	Возвращает имя файла или каталога с указанием пути к нему в файловой системе
LastAccessTime	Позволяет получить или установить время последнего обращения к объекту файловой системы
LastWriteTime	Позволяет получить или установить время последнего внесения изменений в объект файловой системы
Name	Возвращает имя указанного файла. Это свойство доступно только для чтения. Для каталогов возвращает имя последнего каталога в иерархии, если это возможно. Если нет, возвращает полностью определённое имя

В FileSystemInfo предусмотрено и несколько методов. Например, метод Delete() позволяет удалить объект файловой системы с жёсткого диска, а Refresh() — обновить информацию об объекте файловой системы.

Класс DirectoryInfo

Данный класс наследует члены класса FileSystemInfo и содержит дополнительный набор членов, которые предназначены для создания, перемещения, удаления, получения информации о каталогах и подкаталогах в файловой системе.

Наиболее важные члены класса содержатся в следующей таблице:

Член	**Описание**
Create() CreateSubDirectory()	Создают каталог (или подкаталог) по указанному пути в файловой системе
Delete()	Удаляет пустой каталог
GetDirectories()	Позволяет получить доступ к подкаталогам текущего каталога (в виде массива объектов DirectoryInfo)
GetFiles()	Позволяет получить доступ к файлам текущего каталога (в виде массива объектов FileInfo)
MoveTo()	Перемещает каталог и все его содержимое на новый адрес в файловой системе
Parent	Возвращает родительский каталог в иерархии файловой системы

Работа с типом DirectoryInfo начинается с создания экземпляра класса (объект), указывая при вызове конструктора в качестве параметра путь к нужному каталогу. Если мы хотим обратиться к текущему каталогу (то есть каталогу, в котором в настоящее время производится выполнение приложения), вместо параметра используется обозначение ".". Например:

```
// Создаем объект DirectoryInfo, который будет обращаться к текущему каталогу
DirectoryInfo dir1 = new DirectoryInfo(".");
// Создаем объект DirectoryInfo, который будет обращаться к каталогу C:\Temp
DirectoryInfo dir2 = new DirectoryInfo(@"C:\Temp");
```

Если мы попытаемся создать объект DirectoryInfo, связав его с несуществующим каталогом, будет сгенерировано исключение System.IO.DirectoryNotFoundException. Если же всё нормально, мы сможем получить доступ к данному каталогу.

В примере, который приведён ниже, мы создаём объект DirectoryInfo, который связан с каталогом C:\Temp, и выводим информацию о данном каталоге:

```
using System;
using System.IO;
 class Program
 {
     static void Main(string[] args)
     {
         DirectoryInfo dir = new DirectoryInfo(@"C:\Temp");
         Console.WriteLine("***** " + dir.Name + " *****");
         Console.WriteLine("FullName: {0}", dir.FullName);
         Console.WriteLine("Name: {0}", dir.Name);
         Console.WriteLine("Parent: {0}", dir.Parent);
         Console.WriteLine("Creation: {0}", dir.CreationTime);
         Console.WriteLine("Attributes: {0}", dir.Attributes.ToString());
         Console.WriteLine("Root: {0}", dir.Root);
     }
 }
```

Через DirectoryInfo можно не только получать доступ к информации о текущем каталоге, но и получить доступ к информации о его подкаталогах:

```
using System;
using System.IO;
    class Program
    {
        static void printDirect(DirectoryInfo dir)
        {
            Console.WriteLine("***** " + dir.Name + " *****");
            Console.WriteLine("FullName: {0}", dir.FullName);
            Console.WriteLine("Name: {0}", dir.Name);
            Console.WriteLine("Parent: {0}", dir.Parent);
            Console.WriteLine("Creation: {0}", dir.CreationTime);
            Console.WriteLine("Attributes: {0}", dir.Attributes.ToString());
            Console.WriteLine("Root: {0}", dir.Root);
        }

        static void Main(string[] args)
        {
            DirectoryInfo dir = new DirectoryInfo(@"C:\");
            printDirect(dir);
      DirectoryInfo[] subDirects = dir.GetDirectories();
            Console.WriteLine("Найдено {0} подкаталогов", subDirects.Length);
            foreach (DirectoryInfo d in subDirects)
            {
                printDirect(d);
            }
        }
    }
```

Работа с файлами

Класс FileInfo

Класс FileInfo предназначен для организации доступа к физическому файлу, который содержится на жестком диске компьютера. Он позволяет получать информацию об этом файле (например, о времени его создания, размере, атрибутах и т.п.), а также производить различные операции, например, по созданию файла или его удалению. Класс FileInfo наследует члены класса FileSystemInfo и содержит дополнительный набор членов, который приведён в следующей таблице:

Член	Описание
AppendText()	Создаёт объект StreamWriter для добавления текста к файлу
CopyTo()	Копирует уже существующий файл в новый
Create()	Создаёт новый файл и возвращает объект FileStream для взаимодействия с этим файлом
CreateText()	Создаёт объект StreamWriter для записи текстовых данных в новый файл
Delete()	Удаляет файл, которому соответствует объект FileInfo
Directory	Возвращает каталог, в котором расположен данный файл
DirectoryName	Возвращает полный путь к данному файлу в файловой системе
Length	Возвращает размер файла
MoveTo()	Перемещает файл в указанное пользователем место (этот метод позволяет одновременно переименовать данный файл)
Name	Позволяет получить имя файла
Open()	Открывает файл с указанными пользователем правами доступа на чтение, запись или совместное использование с другими пользователями
OpenRead()	Создаёт объект FileStream, доступный только для чтения
OpenText()	Создаёт объект StreamReader (о нём также будет рассказано ниже), который позволяет считывать информацию из существующего текстового файла
OpenWrite()	Создаёт объект FileStream, доступный для чтения и записи

Как мы видим, большинство методов FileInfo возвращает объекты (FIleStream, StreamWriter, StreamReader и т.п.), которые позволяют различным образом взаимодействовать с файлом, например, производить чтение или запись в него.

Коллекции

Коллекции в C# делятся на обобщённые и необобщённые. Начнём знакомство с необобщённых коллекций. Для работы с ними требуется подключить пространство имен System.Collections

Решим задачу. Есть файл в формате csv, представляющий собой информацию о студентах в следующем виде:

	firstName	secondName	univercity	faculty	department	age	course	group	city
Тип данных	Строка	Строка	Строка	Строка	Строка	Целое число	Целое число от 1 до 6	Целое число	Строка
Описание	Имя студента	Фамилия студента	Наименование университета	Наименование факультета	Наименование кафедры	Возраст студента	Номер курса; 1..4 бакалавр; 5..6 магистр	Номер группы	Название города студента

Формат .csv — это файл, в котором данные разделены между собой. В нашем файле как разделитель используется символ ‘;’. Считаем файл и решаем следующие задачи:

1. Сколько всего студентов?
2. Сколько всего бакалавров?
3. Сколько всего магистров?
4. Вывести всех студентов (по ФИО) в алфавитном порядке.

Пример использования коллекций

```
using System;
using System.Collections;
using System.IO;
    class Program
    {
        static void Main(string[] args)
        {
            int bakalavr = 0;
            int magistr = 0;
            // Создадим необобщенный список
            ArrayList list = new ArrayList();
            // Запомним время в начале обработки данных
            DateTime dt = DateTime.Now;
            StreamReader sr = new StreamReader("..\\..\\students_1.csv");
            while(!sr.EndOfStream)
            {
                try {
                    string[] s = sr.ReadLine().Split(';');
            // Console.WriteLine("{0}", s[0], s[1], s[2], s[3], s[4]);
                    list.Add(s[1]+" "+s[0]);// Добавляем склееные имя и фамилию
                    if (int.Parse(s[6]) < 5) bakalavr++; else magistr++;
                }
                catch
```

```
                {
                }
            }
            sr.Close();
            list.Sort();
            Console.WriteLine("Всего студентов:{0}", list.Count);
            Console.WriteLine("Магистров:{0}", magistr);
            Console.WriteLine("Бакалавров:{0}", bakalavr);
            foreach (var v in list) Console.WriteLine(v);
            // Вычислим время обработки данных
            Console.WriteLine(DateTime.Now - dt);
            Console.ReadKey();
        }
    }
```

Хотя необобщённые списки — это довольно мощный инструмент для работы, их использование чревато ошибками, связанными с типами данных, так как эти списки хранят в себе ссылки на объекты общего для всех типа object и за типом приходится следить программисту. Начиная с версии 2.0, в C# появились обобщённые списки.

Рассмотрим решение этой же задачи с использованием обобщённых списков. В данном случае требуется подключить пространство имен System.Collections.Generic.

```
using System;
using System.Collections.Generic;
using System.IO;
    class Student
    {
        public string lastName;
        public string firstName;
        public string university;
        public string faculty;
        public int course;
        public string department;
        public int group;
        public string city;
        int age;
        // Создаем конструктор
           public Student(string firstName, string lastName, string university,
string faculty, string department, int course,int age, int group, string city)
        {
            this.lastName = lastName;
            this.firstName = firstName;
            this.university = university;
       this.faculty = faculty;
            this.department = department;
            this.course = course;
            this.age = age;
            this.group = group;
            this.city = city;
        }
    }
    class Program
    {
         static int MyDelegat(Student st1, Student st2)          // Создаем метод
```

```
для сравнения для экземпляров
        {

                return String.Compare(st1.firstName, st2.firstName);            //
Сравниваем две строки
        }
    static void Main(string[] args)
        {
            int bakalavr = 0;
            int magistr = 0;
                                    List<Student>   list   =   new   List<Student>();
// Создаем список студентов
            DateTime dt = DateTime.Now;
            StreamReader sr = new StreamReader("students_6.csv");
            while (!sr.EndOfStream)
            {
                try
                {
                    string[] s = sr.ReadLine().Split(';');
                    // Добавляем в список новый экземпляр класса Student
                                                                        list.Add(new
Student(s[0],s[1],s[2],s[3],s[4],int.Parse(s[5]),int.Parse(s[6]),int.Parse(s[7])
,s[8]));
                             // Одновременно подсчитываем количество бакалавров и
магистров
                    if (int.Parse(s[5]) < 5) bakalavr++; else magistr++;
                }
                catch(Exception e)
                {
                    Console.WriteLine(e.Message);
                           Console.WriteLine("Ошибка!ESC - прекратить выполнение
программы");
                    // Выход из Main
                    if (Console.ReadKey().Key == ConsoleKey.Escape) return;
                }
            }
            sr.Close();
            list.Sort(new Comparison<Student>(MyDelegat));
            Console.WriteLine("Всего студентов:" + list.Count);
            Console.WriteLine("Магистров:{0}", magistr);
            Console.WriteLine("Бакалавров:{0}", bakalavr);
            foreach (var v in list) Console.WriteLine(v.firstName);
            Console.WriteLine(DateTime.Now - dt);
            Console.ReadKey();
        }
    }
```

Практическая часть урока

Задача 1. Последовательность Фибоначчи.

Создать файл и записать в него n первых членов последовательности Фибоначчи. Вывести на экран все компоненты файла с порядковым номером, кратным 3.

```
using System;
```

```
using System.IO;
namespace FibonachiSeries
{
    class Program
    {
        static void Save(string fileName, int n)
        {
                    FileStream fs = new FileStream(fileName, FileMode.Create,
FileAccess.Write);
            BinaryWriter bw = new BinaryWriter(fs);
             uint a0 = 0;                                        // uint - занимает
4 байта
            uint a1 = 1;
            uint a = a1;
            bw.Write(a0);
            bw.Write(a1);
            for (int i = 2; i < n; i++)
            {
                a = a0 + a1;
                bw.Write(a);
                a0 = a1;
              a1 = a;
            }
            fs.Close();
            bw.Close();
        }
        static void Load(string fileName)
        {
                     FileStream fs = new FileStream(fileName, FileMode.Open,
FileAccess.Read);
            BinaryReader br = new BinaryReader(fs);
            for (int i = 1; i <= fs.Length / 4; i++) // uint занимает 4 байта
            {
                uint a = br.ReadUInt32();
                if (i % 3 == 0) Console.WriteLine("{0,3} {1}",i, a);
            }
            br.Close();
            fs.Close();
        }
        static void Main(string[] args)
        {
            Save("data.bin", 33);
            Load("data.bin");
            Console.ReadKey();
        }
    }
}
```

Задача 2. Сложная задача ЕГЭ.

Для заданной последовательности неотрицательных целых чисел необходимо найти максимальное произведение двух её элементов, номера которых различаются не менее, чем на 8. Значение каждого элемента последовательности не превышает 100 000. Количество элементов последовательности равно 100 000. Сгенерировать файл из случайных чисел и решить эту задачу:

```
using System;
using System.IO;
```

```
namespace BigSeries
{
    class Program
    {
        static void Save(string fileName, int n)
        {
            Random rnd = new Random();
                    FileStream fs = new FileStream(fileName, FileMode.Create,
FileAccess.Write);
            BinaryWriter bw = new BinaryWriter(fs);
            for (int i = 1; i < n; i++)
            {
                bw.Write(rnd.Next(0,100000));      // int занимает 4 байта
            }
            fs.Close();
            bw.Close();
        }
        static void Load(string fileName)
        {
            DateTime d = DateTime.Now;
                     FileStream fs = new FileStream(fileName, FileMode.Open,
FileAccess.Read);
            BinaryReader br = new BinaryReader(fs);
            int[] a = new int[fs.Length / 4];
            for (int i = 0; i < fs.Length / 4; i++)    // int занимает 4 байта
            {
                a[i] = br.ReadInt32();

            }
            br.Close();
            fs.Close();
            int max = 0;
            for (int i = 0; i < a.Length; i++)
                for (int j = 0; j < a.Length; j++)
                     if (Math.Abs(i - j) >= 8 && (long)(a[i] * a[j] > max) max =
a[i] * a[j];
            Console.WriteLine(max);
            Console.WriteLine(DateTime.Now - d);
        }
        static void Main(string[] args)
        {
            Save("data.bin", 100000);
            Load("data.bin");
            Console.ReadKey();
        }
    }
}
```

Задача 3. Минимум функции.

Написать программу сохранения результатов вычисления заданной функции в файл для дальнейшей обработки файла. Разбить программу на две функции: одна записывает данные функции в файла на промежутке от a до b с шагом h, а другая считывает данные и находит минимум функции.

```
using System;
using System.IO;
namespace DoubleBinary
```

```
{
    class Program
    {
        public static double F(double x)
        {
            return x * x-50*x+10;
        }
        public static void SaveFunc(string fileName, double a,double b,double h)
        {
                        FileStream fs = new FileStream(fileName, FileMode.Create,
FileAccess.Write);
            BinaryWriter bw = new BinaryWriter(fs);
            double x = a;
            while (x<=b)
            {
                bw.Write(F(x));
                x += h;// x=x+h;
            }
            bw.Close();
            fs.Close();
     }
        public static double Load(string fileName)
        {
     FileStream fs = new FileStream(fileName, FileMode.Open, FileAccess.Read);
            BinaryReader bw = new BinaryReader(fs);
            double min = double.MaxValue;
            double d;
            for(int i=0;i<fs.Length/sizeof(double);i++)
            {
                        // Считываем значение и переходим к следующему
                d = bw.ReadDouble();
                if (d < min) min = d;
            }
            bw.Close();
            fs.Close();
            return min;
     }
        static void Main(string[] args)
        {
            SaveFunc("data.bin", -100, 100, 0.5);
            Console.WriteLine(Load("data.bin"));
            Console.ReadKey();
        }
    }
}
```

Задача 4. Сканер.

Написать программу нахождения всех адресов почты в заданной папке.

```
using System;
using System.Text.RegularExpressions;
using System.IO;
            // Пример сканирования папки D:\Temp на поиск всех адресов e-mail
```

```
namespace MailScan
{
    class Program
    {
        static void Main(string[] args)
        {
                // Получаем список файлов в папке. AllDirectories - сканировать
также и подпапки
                        string[]  fs  =  Directory.GetFiles("D:\\temp",  "*.*",
SearchOption.AllDirectories);
            // Просматриваем каждый файл в массиве
            foreach(var filename in fs)
        {
            // Создаем регулярное выражения дя поиска почтовых адресов
                Regex regex = new Regex(@"(\w+@[a-zA-Z_]+?\.[a-zA-Z]{2,6})");
            // Считываем файл
                string s = File.ReadAllText(filename);
                Console.WriteLine(filename);
            // Выводим найденные адреса на экран
                foreach (var c in regex.Matches(s))
                    Console.Write("{0} ",c);
            }
            Console.ReadKey();
        }
    }
}
```

Пример делегата

Предположим, нам нужно отсортировать массив целых чисел каким-то своеобразным способом. Для этого мы создаём собственную функцию, которая сравнивает два элемента, и передаём её посредством делегата в метод Sort класса Array.

```
using System;

namespace ConsoleApplication2
{
    class Program
    {
        static int Compare(int a,int b)
        {
            if (a%10 > b%10) return 1;
            if (a%10 < b%10) return -1;
            return 0;
        }

        static void Main(string[] args)
        {
            int[] a = new int[20];
            for (int i = 0; i < a.Length; i++)
                a[i] = i;
            Array.Sort(a,Compare);
            foreach (var el in a)
            {
                Console.Write("{0,4}",el);
```

```
            }

        }
    }
}
```

Пример записи файла различными способами

```
using System;
using System.IO;
using System.Diagnostics;
namespace CopySamples
{
    class Program
    {
        static void Main(string[] args)
        {
            long kbyte = 1024;
            long mbyte = 1024 * kbyte;
            long gbyte = 1024 * mbyte;
            long size = mbyte;
            //Write FileStream
            //Write BinaryStream
            //Write StreamReader/StreamWriter
            //Write BufferedStream

            Console.WriteLine("FileStream. Milliseconds:{0}",
FileStreamSample("D:\\temp\\bigdata0.bin",size));
            Console.WriteLine("BinaryStream. Milliseconds:{0}",
BinaryStreamSample("D:\\temp\\bigdata1.bin", size));
            Console.WriteLine("StreamWriter. Milliseconds:{0}",
StreamWriterSample("D:\\temp\\bigdata2.bin", size));
            Console.WriteLine("BufferedStream. Milliseconds:{0}",
BufferedStreamSample("D:\\temp\\bigdata3.bin", size));

            Console.ReadKey();
        }

        static long FileStreamSample(string filename, long size)
        {
            Stopwatch stopwatch = new Stopwatch();
            stopwatch.Start();
            FileStream fs = new FileStream(filename, FileMode.Create,
FileAccess.Write);
            //FileStream fs = new FileStream("D:\\temp\\bigdata.bin",
FileMode.CreateNew, FileAccess.Write);
            for (int i = 0; i < size; i++)
                fs.WriteByte(0);
            fs.Close();
            stopwatch.Stop();
```

```
            return stopwatch.ElapsedMilliseconds;
        }

        static long BinaryStreamSample(string filename, long size)
        {
            Stopwatch stopwatch = new Stopwatch();
            stopwatch.Start();
            FileStream fs = new FileStream(filename, FileMode.Create,
FileAccess.Write);
            BinaryWriter bw = new BinaryWriter(fs);
            for (int i = 0; i < size; i++)
                bw.Write((byte)0);
            fs.Close();
            stopwatch.Stop();
            return stopwatch.ElapsedMilliseconds;
        }

        static long StreamWriterSample(string filename, long size)
        {
            Stopwatch stopwatch = new Stopwatch();
            stopwatch.Start();
            FileStream fs = new FileStream(filename, FileMode.Create,
FileAccess.Write);
            StreamWriter sw = new StreamWriter(fs);
            for (int i = 0; i < size; i++)
                sw.Write(0);
            fs.Close();
            stopwatch.Stop();
            return stopwatch.ElapsedMilliseconds;
        }

        static long BufferedStreamSample(string filename, long size)
        {
            Stopwatch stopwatch = new Stopwatch();
            stopwatch.Start();
            FileStream fs = new FileStream(filename, FileMode.Create,
FileAccess.Write);
            int countPart = 4;//количество частей
            int bufsize = (int)(size / countPart);
            byte[] buffer = new byte[size];
            BufferedStream bs = new BufferedStream(fs,bufsize);
            //bs.Write(buffer, 0, (int)size);//Error!
            for (int i=0;i<countPart;i++)
                bs.Write(buffer, 0, (int)bufsize);
            fs.Close();
            stopwatch.Stop();
            return stopwatch.ElapsedMilliseconds;
        }
    }
}
```

Домашнее задание

1. Изменить программу вывода таблицы функции так, чтобы можно было передавать функции типа double (double, double). Продемонстрировать работу на функции с функцией a*x^2 и функцией a*sin(x).
2. Модифицировать программу нахождения минимума функции так, чтобы можно было передавать функцию в виде делегата.
   а) Сделать меню с различными функциями и представить пользователю выбор, для какой функции и на каком отрезке находить минимум. Использовать массив (или список) делегатов, в котором хранятся различные функции.
   б) *Переделать функцию Load, чтобы она возвращала массив считанных значений. Пусть она возвращает минимум через параметр (с использованием модификатора out).

3. Переделать программу *Пример использования коллекций* для решения следующих задач:
   а) Подсчитать количество студентов учащихся на 5 и 6 курсах;
   б) подсчитать сколько студентов в возрасте от 18 до 20 лет на каком курсе учатся (*частотный массив);
   в) отсортировать список по возрасту студента;
   г) *отсортировать список по курсу и возрасту студента;

4. **Считайте файл различными способами. Смотрите “Пример записи файла различными способами”. Создайте методы, которые возвращают массив byte (FileStream, BufferedStream), строку для StreamReader и массив int для BinaryReader.

Достаточно решить 2 задачи. Старайтесь разбивать программы на подпрограммы. Переписывайте в начало программы условие и свою фамилию. Все программы сделать в одном решении.

Используемая литература

Для подготовки данного методического пособия были использованы следующие ресурсы:

1. Гойвертс Я., Левитан С. Регулярные выражения. Сборник рецептов. — СПб: Символ-Плюс, 2009.
2. Дейтел П., Дейтел Х. Как программировать на Visual C# 2012. — СПб: Питер, 2014.
3. Климов А.С. — C#. Советы программистам. — СПб.: БХВ-Петербург, 2008.
4. Павловская Т.А. Программирование на языке высокого уровня. — СПб: Питер, 2009.
5. Петцольд Ч. Программирование на C#. Т.1. — М.: Русская редакция, 2001.
6. Шилдт Г. C# 4.0. Полное руководство. — М.: ООО «И.Д. Вильямс», 2011.
7. MSDN.