Организация C#-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).

- Центральную часть потоковой C#-системы - класс ***Stream*** пространства имен ***System.IO***.

- Класс ***Stream*** представляет байтовый поток и является базовым для всех остальных потоковых классов.

- Из класса ***Stream*** выведены байтовые классы потоков :

1) **FileStream** - байтовый поток, разработанный для файлового ввода-вывода

2) **BufferedStream** - заключает в оболочку байтовый поток и добавляет буферизацию (часто увеличивает производительность программы);

3) **MemoryStream** - байтовый поток, который использует память для хранения данных.

- Программист может вывести собственные потоковые классы.

Байтовый поток

Чтобы создать байтовый поток, связанный с файлом, создается объект класса **FileStream**

FileStream(string filename, FileMode mode) // конструктор, который открывает поток для
// чтения и/или записи

где:

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

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

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

Если попытка открыть файл оказалась неуспешной, то генерируется одно из исключений:

- FileNotFoundException - файл невозможно открыть по причине его отсутствия,
- IOException - файл невозможно открыть из-за ошибки ввода-вывода,
-
- ArgumentNullException - имя файла представляет собой null-значение,
- ArgumentException - некорректен параметр mode,
- SecurityException - пользователь не обладает правами доступа,
- DirectoryNotFoundException - некорректно задан каталог.

Байтовый поток

Другая версия конструктора ограничивает 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, определенных 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.

Метод ReadByte() и WriteByte()

- 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 достинут конец файла, то метод ReadByte() возвращает значение -1.
- Для побайтовой записи данных в поток используется метод WriteByte().
- По завершении работы с файлом его необходимо закрыть (метод Close ()).
- При закрытии файла освобождаются системные ресурсы, ранее выделенные для этого файла, что дает возможность использовать их для работы с другими файлами.

```
/*использование класса FileStream, для копирования одного файла в другой, текстовый
    файл text.txt находиться в папке bin/debug текущего проекта*/
using System;
using System.Text;
using System.IO;                    //для работы с потоками
  namespace ConsoleApplication1{
    class Program       {
      static void Main()          {
        try             {
          FileStream fileIn = new FileStream("text.txt", FileMode.Open, FileAccess.Read);
          FileStream fileOut = new FileStream("newText.txt", FileMode.Create, FileAccess.Write);
          int i;
          while ((i = fileIn.ReadByte()) != -1)                 {
            //запись очередного файла в поток, связанный с файлом fIleOut
            fileOut.WriteByte((byte)i);
          }
          fileIn.Close();
          fileOut.Close();
        }
        catch (Exception EX)
        {               Console.WriteLine(EX.Message);              }
      }     }       }
```

Символьный поток

Создание символьного потока:

помещаем объект класса Stream (например, FileStream) «внутрь» объекта класса StreamWriter или объекта класса StreamReader (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);

Где stream определяет имя уже открытого байтового потока.

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

Этот конструктор генерирует исключение :

- ***ArgumentException***, если поток не открыт для вывода,
- ***ArgumentNullException***, если поток имеет null-значение.

Символьный поток

Другой вид конструктора позволяет открыть поток сразу через обращения к файлу:

StreamWriter(string name);

где параметр **name** определяет имя открываемого файла.

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

И еще один вариант конструктора StreamWriter:

StreamWriter(string name, bool appendFlag);

где 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.

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

Для записи данных в поток fileOut можно обратиться к методу WriteLine:

fileOut.WriteLine("test");

//В конец файла t.txt будет дописано слово test

Символьный поток

Класс StreamReader предназначен для организации входного символьного потока

Один из конструкторов:

StreamReader(Stream stream);

где параметр stream определяет имя уже открытого байтового потока.

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

Например:

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

У класса StreamReader есть другой вид конструктора (позволяет открыть файл напрямую):

StreamReader (string name);

где параметр name определяет имя открываемого файла.

Обратиться к данному конструктору можно:

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

Символьный поток

В C# символы реализуются кодировкой Unicode, для того, чтобы можно было обрабатывать текстовые файлы, содержащие русский символы, рекомендуется вызывать следующий вид конструктора StreamReader:

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

- Параметр Encoding.GetEncoding(1251) - будет выполняться преобразование из кода Windows-1251 (одна из модификаций кода ASCII, содержащая русские символы) в Unicode.

- Encoding.GetEncoding(1251) реализован в пространстве имен System.Text.

Для чтения данных из потока fileIn можно воспользоваться методом **ReadLine** (если будет достигнут конец файла, то метод ReadLine вернет значение null).

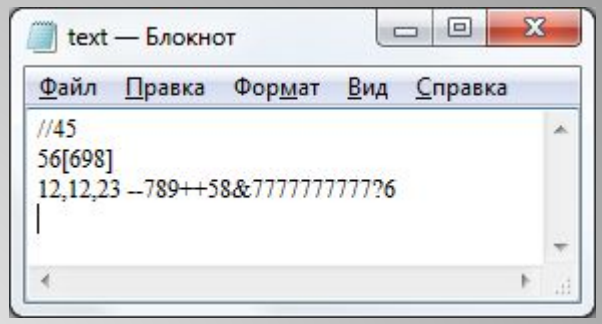
/*данные из одного файла копируются в другой, с использованием классов StreamWriter и StreamReader*/

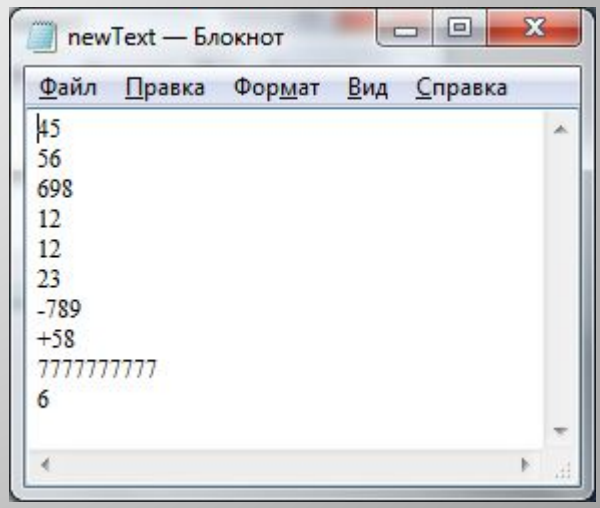
```
static void Main()
{
  StreamReader fileIn = new StreamReader("text.txt",
Encoding.GetEncoding(1251));
  StreamWriter fileOut=new StreamWriter("newText.txt", false);
  string line;
  while ((line=fileIn.ReadLine())!=null)    //пока поток не пуст
  {
        fileOut.WriteLine(line);
  }
  fileIn.Close();
  fileOut.Close();
}
```

Работа данного способа копирования будет менее эффективной, т.к. будет тратиться дополнительное время на преобразование байтов в символы.

```
/*использование регулярных выражений для поиска заданных фрагментов текста в файле*/
using System;
using System.Text;
using System.IO;
using System.Text.RegularExpressions;

namespace ConsoleApplication1        {
   class Program       {
      static void Main()          {
         StreamReader fileIn = new StreamReader("text.txt");
         StreamWriter fileOut = new StreamWriter("newText.txt", false);
         string text = fileIn.ReadToEnd();
         Regex r = new Regex(@"[-+]?\d+");
         Match integer = r.Match(text);
         while (integer.Success)
         {
            fileOut.WriteLine(integer);
            integer = integer.NextMatch();
         }
         fileIn.Close();
         fileOut.Close();
      }      }      }
```





Двоичные потоки

- Двоичные файлы хранят данные в том же виде, в котором они представлены в оперативной памяти (во внутреннем представлении).

- Двоичные файлы не применяются для просмотра человеком, они используются только для программной обработки.

- Выходной поток **BinaryWriter** поддерживает произвольный доступ, т.е. имеется возможность выполнять запись в произвольную позицию двоичного файла.

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

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 определенного типа

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

//Формирования двоичного файла:

```
static void Main()
{
        //открываем двоичный поток
        BinaryWriter fOut=new BinaryWriter(new
FileStream("t.dat",FileMode.Create));

        //записываем данные в двоичный поток
        for (int i=0; i<=100; i+=2)
        {
            fOut.Write(i);
        }
        fOut.Close();       //закрываем двоичный поток
}
```

//Просмотр двоичного файла :

```
static void Main()
{
        FileStream f=new FileStream("t.dat",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.

Seek(long newPos, SeekOrigin pos)

- где параметр newPos определяет новую позицию внутреннего указателя файла в байтах относительно исходной позиции указателя, которая определяется параметром pos.

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 следующие операции чтения или записи будут выполняться с новой позиции внутреннего указателя файла.

```
//организация произвольного доступа к двоичному файлу (на примере файла t.dat):
  static void Main()          {
        //изменение данных в двоичном потоке
        FileStream f=new FileStream("t.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("t.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();      }
```

```
        /*Поток BinaryReader не имеет метода Seek, однако используя возможности потока
    FileStream можно организовать произвольный доступ  при чтении  двоичных файлов:*/

static void Main()
{
    //Записываем в файл t.dat целые числа от 0 до 100
    FileStream f=new FileStream("t.dat",FileMode.Open);
    BinaryWriter fOut=new BinaryWriter(f);
    for (int i=0; i<100; ++i)
    {
      fOut.Write(i);;
    }
    fOut.Close();
    //Объекты f и fIn связаны с одним и тем же файлом
    f=new FileStream("t.dat",FileMode.Open);
    BinaryReader fIn=new BinaryReader(f);
    long n=f.Length;     //определяем количество байт потоке
//Читаем данные из файла t.dat, перемещая внутренний указатель на 8 байт, т.е. на два целых
    числа
    for (int i=0; i<n; i+=8)
    {
      f.Seek(i,SeekOrigin.Begin);
      int a=fIn.ReadInt32();
      Console.Write(a+" ");
    }
    fIn.Close();
    f.Close();
}
```

Перенаправление стандартных потоков

Тремя стандартными потоками, доступ к которым осуществляется через свойства

- **Console.Out,**
- **Console.In,**
- **Console.Error,**

могут пользоваться все программы, работающие в пространстве имен System.

- Свойство Console.Out относится к стандартному выходному потоку(по умолчанию это консоль).
- Например, при вызове метода Console.WriteLine() информация автоматически передается в поток Console.Out.

- Свойство Console.In относится к стандартному входному потоку(по умолчанию – клавиатура).
- Например, при вводе данных с клавиатуры информация автоматически передается потоку Console.In, к которому можно обратиться с помощью метода Console.ReadLine().

- Свойство Console.Error относится к ошибкам в стандартном потоке(источник которого по умолчанию – консоль).

- Но эти потоки могут быть перенаправлены на любое совместимое устройство ввода-вывода (на работу с физическими файлами)

Перенаправление стандартных потоков

Перенаправить стандартный поток можно с помощью методов:

- SetIn(),
- SetOut()
- SetError(),

которые являются членами класса Console:

- ***static void SetIn(TextReader input)***
- ***static void SetOut(TextWriter output)***
- ***static void SetError(TextWriter output)***

```
class Program       {
      //перенаправления потоков :двумерный массив вводится из файла input.txt, а выводится в файл output.txt
        static void Main()          {
          try             {
            int[,] MyArray;                StreamReader file = new StreamReader("input.txt");
            Console.SetIn(file);   // перенаправляем стандартный входной поток на file
            string line = Console.ReadLine();  string[] mas = line.Split(' ');
            int n = int.Parse(mas[0]);             int m = int.Parse(mas[1]);
            MyArray = new int[n, m];
            for (int i = 0; i < n; i++)                {
              line = Console.ReadLine();
              mas = line.Split(' ');
              for (int j = 0; j < m; j++)                  {                    MyArray[i, j] = int.Parse(mas[j]);                    }
            }
            PrintArray("исходный массив:", MyArray, n, m);
            file.Close();              }
          catch             { Console.WriteLine("Возникла ошибка"); }
        }
 static void PrintArray(string a, int[,] mas, int n, int m)                {
StreamWriter file=new StreamWriter("output.txt"); // перенаправляем стандартный входной поток на file
Console.SetOut(file);       Console.WriteLine(a);
for (int i = 0; i < n; i++){
for (int j=0; j<m; j++) Console.Write("{0} ", mas[i,j]);         Console.WriteLine();}
file.Close();   }
      }
```

Восстановить исходное состояние потока ***Console.In*** можно:

```
TextWriter str = Console.In;  // первоначально сохраняем исходное состояние входного потока
…
Console.SetIn(str);    // при необходимости восстанавливаем исходное состояние входного потока
```

Так же можно восстановить исходное состояние потока ***Console.Out***:

```
TextWriter str = Console.Out;  // первоначально сохраняем исходное состояние выходного потока
…
// при необходимости восстанавливаем исходное состояние выходного потока
Console.SetOut(str);
```