Файловый ввод-вывод

Лекция 10

Обмен данными

- Работа с файлами является частным случаем процессов обмена данными между программой и её внешним окружением
- Частью внешнего окружения программы являются различные устройства ввода-вывода, внешние запоминающие устройства, другие программы, сетевые ресурсы

Потоки

- Обмен данными между программой и внешним окружением основан на концепции *потока*
- Поток – это абстракция, представляющая любую последовательность элементов данных, передаваемых или получаемых программой
- Программное представление потоков основывается на базовом абстрактном классе *Stream*

Байтовые потоки

- Минимальной единицей данных, передаваемых в операциях ввода-вывода является байт
- Потоки, оперирующие байтами, называются *байтовыми потоками*
- Базовый класс *Stream* является байтовым

Входные и выходные потоки

- Существуют два типа потоков: входные и выходные
- Входные потоки используются для чтения данных в оперативную память
- Выходные потоки используются для записи данных из оперативной памяти в некоторое внешнее место назначения (дисковый файл, местоположение в сети, принтер или другая программа)

Специализированные потоки

- Для различных видов источников данных, используемых программой, определены специализированные наследники класса *Stream*:
  - *FileStream* - обмен данными с файлами,
  - *NetworkStream* - передача данных по сети,
  - *PipeStream* - обмен данными между программами,
  - *MemoryStream* для массивов данных в оперативной памяти

Буферизация потоков

- Для повышения производительности компьютера при обменах данными программ с внешним окружением применяется буферизация потоков
- Буферизация позволяет производить вычисления одновременно с операциями ввода-вывода
- Для этого в оперативной памяти выделяются определённые участки, называемые *буферами ввода* и *буферами вывода*

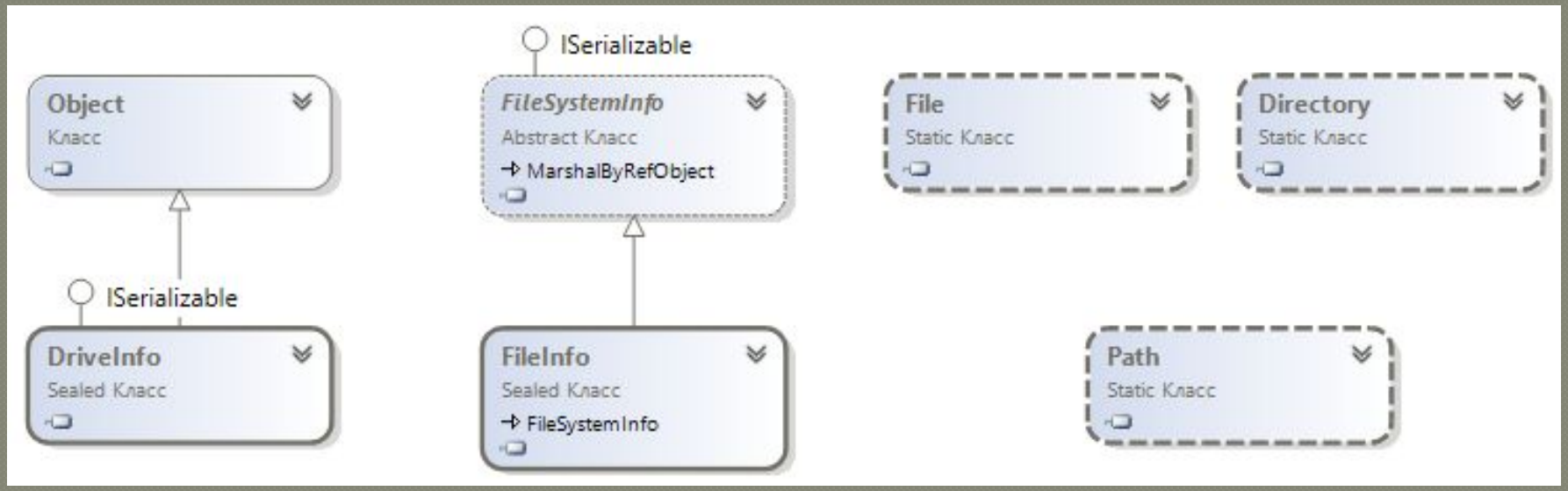
Буферизация потоков

- Программа записывает данные в буфер вывода и читает данные из буфера ввода
- В этом случае потоки вывода и ввода обеспечивают теперь обмен данными между буферами и внешним окружением программы
- Организация такого механизма обмена обеспечивается классом-оболочкой (*декоратором*) *BufferedStream*

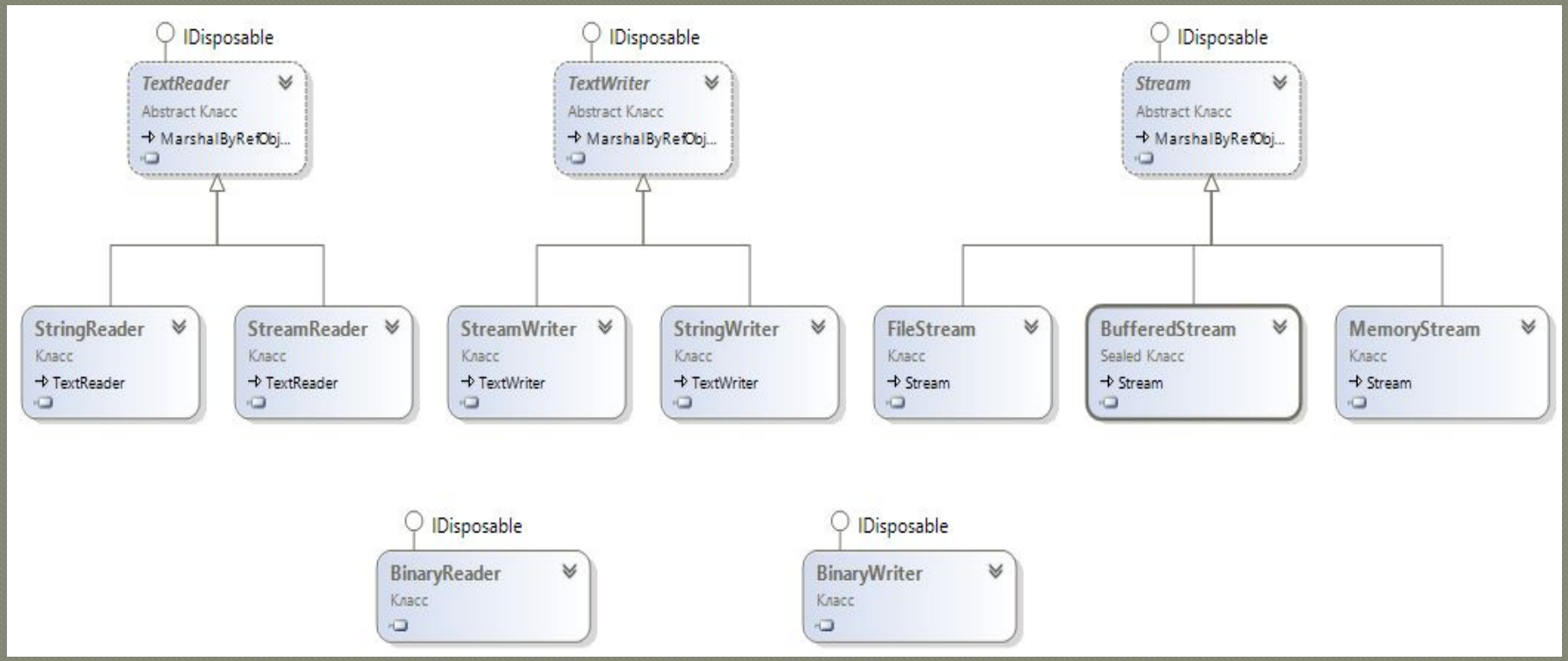
Работа с файлами

- Работа с файлами – наиболее традиционный способ использования постоянной памяти
- Для этого в C# имеется множество классов, содержащихся в пространстве имен *System.IO*
- Классы этого пространства имен можно разбить на две группы:
  - классы, использующие файловую систему;
  - классы, использующие потоки

Классы, использующие файловую систему



Некоторые потоковые классы



Классы файлового ввода-вывода

- В классе *FileStream* определено несколько конструкторов; чаще всего используется конструктор:

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

где

- *filename* – имя файла, с которым будет связан поток ввода-вывода (это либо полный путь к файлу, либо имя файла, находящегося в папке bin/debug проекта);
- *mode* – режим открытия файла

Режимы файлового потока

- Значения параметра mode определяются перечислением *FileMode*, определенным в классе *System.IO*:
  - *CreateNew = 1* - создание нового файла, при этом файл с таким же именем не должен существовать;
  - *Create = 2* - создание нового файла, при этом если существует файл с таким же именем, то он будет предварительно удален;
  - *Open = 3* - открытие существующего файла

Режимы файлового потока

- *OpenOrCreate = 4* - открытие, если файл существует, в противном случае создание нового файла;
- *Truncate = 5* - открытие существующего файла, с усечением его длины до нуля;
- *Append = 6* - добавление данных в конец файла

Исключения

- 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

Исключения

- *ArgumentException* - некорректен параметр mode;
- *SecurityException* - пользователь не обладает правами доступа;
- *DirectoryNotFoundException* - некорректно задан каталог

Другая версия конструктора

- Версия конструктора, позволяющая ограничить доступ к файлу только чтением или только записью:

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

- Параметр *how* определяет способ доступа к файлу и может принимать одно из значений, определенных перечислением *FileAccess*:
  - *Read = 1* - только чтение;
  - *Write = 2* - только запись;
  - *ReadWrite = 3* - и чтение, и запись

Операции чтения-записи

- После открытия файла внутренний указатель потока устанавливается на его начальный байт
- Для чтения очередного байта из потока используется метод *ReadByte()*, возвращающий значение типа int, представляемое этим байтом
- После прочтения очередного байта внутренний указатель перемещается на следующий байт

Операции чтения-записи

- Если достигнут 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 методом *Close()*
- При этом освобождаются системные ресурсы, связанные с файлом

Применение байтовых потоков

- Байтовые потоки удобно использовать при «внутренних» операциях с файлами, которые не связаны с передачей данных в поток пользовательского интерфейса или их преобразованием к определенному типу
- Примером такой операции может служить создание файла-копии

Классы-адаптеры

- Для преобразования потока байт в последовательность значений того или иного типа используются *классы-адаптеры*
- Различают два вида адаптеров:
  - текстовые,
  - двоичные
- Текстовые классы-адаптеры преобразуют байтовые потоки в потоки символов и наоборот

Текстовые адаптеры

- Такие преобразования особенно актуальны в случае, когда обмен данными происходит с использованием пользовательского интерфейса, поскольку текстовое представление информации наиболее привычно для человека
- Методы текстовых адаптеров объявлены в абстрактных классах *TextReader* и *TextWriter*

Символьные потоки

- Реализация методов этих абстрактных классов представлена в их классах-наследниках:
  - *StreamReader* и *StringReader*, наследующих *TextReader*;
  - *StreamWriter* и *StringWriter*, наследующих *TextWriter*
- Символьные потоки оперируют символами, которые могут быть представлены в различных *кодировках*

Создание символьных потоков

- Чтобы создать символьный поток, нужно поместить объект класса *Stream* (например, *FileStream*) "внутрь" объектов классов-оболочек *StreamWriter* или *StreamReader*
- В этом случае байтовый поток будет автоматически преобразовываться в символьный и наоборот

Поток 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

Типы исключений

- Этот конструктор может генерировать исключения следующих типов:
  - *ArgumentException*, если поток *stream* не открыт для вывода;
  - *ArgumentNullException*, если поток *stream* имеет null-значение

Второй вариант конструктора

- Позволяет открыть поток сразу через обращения к файлу:

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

- Параметр *name* определяет имя открываемого файла
- Например:

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

Третий вариант конструктора

- Определяет режим записи – дозапись или перезапись

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

- Параметр *appendFlag* принимает значение
  - true - данные нужно добавлять в конец файла;
  - false - файл необходимо перезаписать
- Например:

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

Поток StreamReader

- Предназначен для организации входного символьного потока
- В нем определено несколько конструкторов, один из них записывается следующим образом:

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

- Параметр *stream* определяет имя уже открытого байтового потока

Типы исключений

- Этот конструктор генерирует исключение типа *ArgumentException*, если поток *stream* не открыт для ввода
- Например, создать экземпляр класса *StreamReader* можно так:

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

Второй вариант конструктора

- Позволяет открыть поток сразу через обращения к файлу:

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

- Параметр name определяет имя открываемого файла
- Например:

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

Чтение данных

- Для построчного чтения данных из символьного потока предназначен метод *ReadLine()*
- Этот метод возвращает очередную строку текста, автоматически определяя положение завершающего её символа ‘\n’
- При этом, если будет достигнут конец файла, то метод *ReadLine()* вернет значение null

Чтение кириллицы

- **В C# символы реализуются кодировкой** Unicode
- **Для того, чтобы можно было обрабатывать текстовые файлы, содержащие русские символы рекомендуется вызывать следующий вид конструктора** *StreamReader:*

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

Чтение кириллицы

- Здесь в качестве второго параметра указан вызов статического метода *GetEncoding()* класса *Encoding,* который определен в пространстве имен *System.Text*

Предопределенные потоки

- **К символьным потокам относятся и так называемые *предопределенные потоки ввода-вывода*, используемые в консольных приложениях:**
  - *In* – предопределенный поток ввода,
  - *Out* – предопределенный поток вывода,
  - *Err* – предопределенный поток вывода сообщений об ошибках
- **Эти потоки реализованы в классе *Console* пространства имен *System;*** методы доступа к ним были рассмотрены ранее

Двоичные адаптеры

- Во многих приложениях требуется производить обмен числовыми данными, сохраняя их внутреннее представление
- В этом случае необходимо выполнять преобразования байтового потока в последовательность числовых значений, и наоборот
- Для этой цели используются классы-оболочки *BinaryReader* и *BinaryWriter*

Двоичные потоки

- Последовательности числовых данных в их внутреннем представлении называются *двоичными потоками*, а файлы, из которых они считываются или в которые они записываются – *двоичными файлами*
- Двоичные файлы хранят данные во внутреннем представлении и предназначены не для просмотра человеком, а только для программной обработки

Создание двоичного потока

- 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.
- Например:

```
BinaryWriter fOut=new BinaryWriter(new
FileStream("t.dat",FileMode.Create));
```

Произвольный доступ

- Двоичные файлы являются файлами с произвольным доступом; нумерация элементов в двоичном файле ведется с нуля
- Произвольный доступ обеспечивает метод *Seek*, имеющий синтаксис:

  Seek(long newPos, SeekOrigin pos)

- Здесь параметр *newPos* определяет новую позицию внутреннего указателя файла в байтах относительно исходной позиции pos

Значения параметра pos

Значение	Описание
`SeekOrigin.Begin`	Поиск от начала файла
`SeekOrigin.Current`	Поиск от текущей позиции указателя
`SeekOrigin.End`	Поиск от конца файла

Поток BinaryWriter

- Класс *BinaryWriter* поддерживает произвольный доступ к выходному двоичному потоку, обеспечивая, в частности, возможность выполнять запись в заданную позицию двоичного файла
- Метод *Write* этого класса имеет многочисленные перегрузки, предназначенные для записи данных разных типов

Методы потока BinaryWriter

Метод 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

Поток BinaryReader

- Класс *BinaryReader* поддерживает последовательный доступ к входному двоичному потоку, обеспечивая, в частности, возможность выполнять чтение данных различных типов из двоичного файла

Методы потока BinaryReader

Метод 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

Конец лекции