УО «ПОЛОЦКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

ЛАБОРАТОРНАЯ РАБОТА №3

по дисциплине «Базы данных»
для специальности 1-40 01 01
«Программное обеспечение информационных технологий»

ТЕМА: Особенности работы с файлами на языке C# в среде разработки Microsoft Visual Studio 2008**.**

НОВОПОЛОЦК 2011

МАТЕРИАЛЫ ПОДГОТОВИЛА:

старший преподаватель кафедры технологий программирования

Бураченок Ирина Брониславовна

ТЕМА: Особенности работы с файлами на языке C# в среде разработки Microsoft Visual Studio 2008**.**

ЦЕЛЬ: Научить пользователя основным принципам и приемам работы с с файлами прямого и произвольного доступа на языке C# в среде разработки Microsoft Visual Studio 2008**.**

Результат обучения:

После успешного завершения занятия пользователь должен:

- получить представление о работе с текстовыми файлами на C# (открывать и закрывать файл, записывать в файл, читать из файла);
- получить представление о работе с бинарными файлами (произвольного доступа) на C# (открывать и закрывать файл, записывать в файл, читать из файла);
- уметь использовать дополнительные операторы для работы с файлами (переименование, копирование, уничтожение).

План занятия:

1. Общие сведения о файлах и классах для работы с ними.

2. Работа с файлами с текстовыми файлами.

3. Работа с файлами с бинарными файлами.

4. Использование дополнительных классов и методов для работы с файлами.

1. ОБЩИЕ СВЕДЕНИЯ О ФАЙЛАХ И КЛАССОВ ДЛЯ РАБОТЫ С НИМИ

Файл — именованная информация на внешнем носителе, например на жестком или гибком магнитном диске. Логически файл можно представить как конечное количество последовательных байтов. Передача данных с файла в оперативную память называется чтением, или вводом, обратный процесс — записью, или выводом.

Ввод-вывод в C# выполняется с помощью подсистемы ввода-вывода и классов библиотеки **.NET**. Обмен данными реализуется с помощью потоков.

Поток (stream) — это абстрактное понятие, относящееся к любому переносу данных от источника к приемнику. Потоки обеспечивают надежную работу как со стандартными, так и с определенными пользователем типами данных. Поток определяется как последовательность байтов и не зависит от конкретного устройства, с которым производится обмен (оперативная память, файл на диске, клавиатура или принтер). Обмен с потоком для повышения скорости передачи данных производится, как правило, через специальную область оперативной памяти — буфер. Буфер выделяется для каждого открытого файла. При записи в файл вся информация сначала направляется в буфер и там накапливается до тех пор, пока весь буфер не заполнится. Только после этого или после специальной команды сброса происходит передача данных на внешнее устройство. При чтении из файла данные вначале считываются в буфер, причем не столько, сколько запрашивается, а сколько помещается в буфер. Механизм буферизации позволяет более быстро и эффективно обмениваться информацией с внешними устройствами.

Для поддержки потоков для работы с файлами библиотека **.NET** содержит иерархию классов. Основная часть классов, используемых в лабораторной работе, представлена на рис. 1. Эти классы определены в пространстве имен **System.I0**. Помимо классов там описано большое количество перечислений для задания различных свойств и режимов.

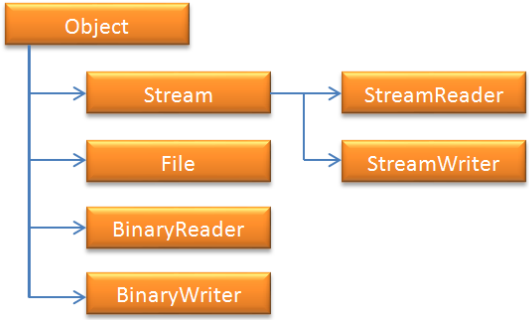

Рис. 1. *Классы для работы с файлами библиотеки* ***.NET***

В **.NET** используется кодировка **Unicode**, в которой каждый символ кодируется двумя байтами. Классы, работающие с текстом, являются оболочками классов, использующих байты. Они автоматически выполняют перекодирование из байтов в символы и обратно.

Двоичные и байтовые потоки хранят данные в том же виде, в котором они представлены в оперативной памяти, то есть при обмене с файлом происходит побитовое копирование информации. Двоичные файлы применяются не для просмотра их человеком, а для использования в программах.

Доступ к файлам может быть последовательным и произвольным.

Последовательный доступ — доступ к файлу, когда очередной элемент можно прочитать (записать) только после аналогичной операции с предыдущим элементом.

Произвольным или прямой доступ — доступ к файлу, при котором выполняется чтение (запись) произвольного элемента по заданному адресу.

Текстовые файлы позволяют выполнять только последовательный доступ, а в бинарных потоках можно использовать оба метода. Прямой доступ в сочетании с отсутствием преобразований обеспечивает высокую скорость получения нужной информации. Поэтому он чаще используется для построения и работы с файлами базы данных и с файлами, где время выборки информации имеет значение.

2. РАБОТА С ТЕКСТОВЫМИ ФАЙЛАМИ

Текстовый файл это последовательность символов, в том числе и управляющих.

Для работы с текстовым файлом используются следующие классы:

Таблица 1. *Классы, используемые для работы с текстовыми файлами*

Класс	Описание
StreamWriter	Класс создает файл для записи или открывает существующий файл для добавления. Конструктор класса содержит следующие параметры: **StreamWriter(Stringpath,boolapend)** **path**-абсолютный или относительный путь к файлу; **apend**-режим создания файла(**false**-файл создается заново, **true**-открывается для добавления);
StreamReader	Класс открывает файл для чтения. Конструктор класса содержит следующие параметры: **StreamWriter(Stringpath)** **path**-абсолютный или относительный путь к файлу. При отсутствии файла возникает исключение **FileNotFoundException**;

Таблица 2. *Методы класса **StreamWriter***

Метод	Описание
Write(String str)	Вывод строки в файл
WriteLine(String format, Object params)	Форматный вывод строки в файл
Close	Закрытие файлового потока

Таблица 3. *Методы класса **StreamReader***

Метод	Описание
Read()	Выполняет чтение следующего символа из файла и перемещает положение на одну позицию вперед.
ReadLine()	Выполняет чтение следующего строки из файла и перемещает положение на одну строку вперед.
Peek()	Возвращает следующий доступный символ, но не использует его. Если достигает конца файла то возвращает 0.
Close	Закрытие файлового потока.

2.1. ЗАПИСЬ В ФАЙЛ

Пример 1.

Напишем программу записи в текстовый файл следующей информации содержащей Имя и год рождения человека. В качестве метода записи в текстовый файл будем использовать **Write()**. Листинг программы 1:

```
//Если 2 Edit-а непустые
if (textBox1.Text != ""&& textBox2.Text != "")
{
//Создаем файл,если файл существует,
//то открывается для добавления
StreamWriter f = newStreamWriter("ГодР.txt",true);
f.Write(textBox1.Text);
f.Write(' ');
f.Write(textBox2.Text);
f.WriteLine();
f.Close();
}
```

Результат работы программы представлен на рисунке 2.

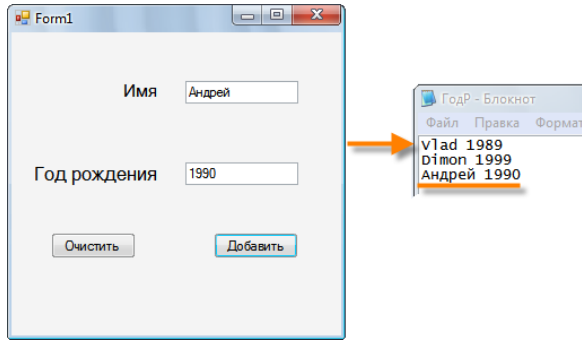

Рис. 2. *Результат работы программы*

Для наглядного представления отличий метода **Write** от **WriteLine** этот же пример напишем используя метод **WriteLine**. Листинг программы 1.2:

```
//Если 2 Edit-а не пустые
if (textBox1.Text != ""&& textBox2.Text != "")
{
StreamWriter f = newStreamWriter("ГодР.txt", true);
f.WriteLine("{0} {1}", textBox1.Text, textBox2.Text);
f.Close();
}
```

2.2. ЧТЕНИЕ ИЗ ФАЙЛА

Пример 2

Напишем программу которая считывает **ГодР.txt**, записанный примером 1, и выводит его содержимое в **ListBox**. Листинг программы 1.2:

```
try
{
StreamReader f = newStreamReader("ГодР.txt");
String s;
//Пока не конец файла
              while (f.Peek() >= 0)
              {
            s = f.ReadLine();
            listBox1.Items.Add(s);
              }
f.Close();
}
catch (FileNotFoundException)
{
       listBox1.Items.Add("Ошибка открытия файла!");
}
```

Результат работы программы представлен на рисунке 3.

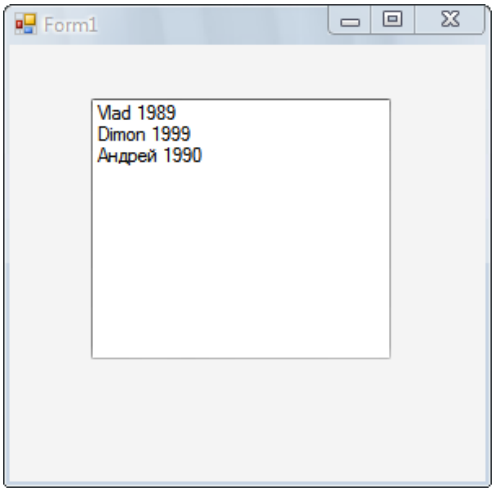

Рис. 3. *Результат работы программы*

3. РАБОТА С БИНАРНЫМИ ФАЙЛАМИ

Бинарный файл – файл содержащий информацию, представленную в двоичном (бинарном) виде. Причем эта информация может быть структурированной. Для работы с текстовым файлом используются следующие классы:

Таблица 4. *Классы для работы с текстовым файлом*

Класс	Описание
BinaryWriter	Класс создает файл для записи или открывает существующий файл для добавления. Конструктор класса содержит следующие параметры: **BinaryWriter(FileStream fs)** **fs**-файловый поток вывода в файл;
BinaryReader	Класс открывает файл для чтения. Конструктор класса содержит следующие параметры: **StreamReader (FileStream fs)** **fs**-файловый поток считывания из файла. При отсутствии файла возникает исключение **FileNotFoundException**;
FileStream	Конструктор класса содержит: **FileStream(String put,FileAccess acc)** **put**-путь к файлу; **acc**-права доступа к файлу(принимается с перечислением **FileMode**);

Таблица 5. *Значения перечисления **FileMode***

Значение	Описание
Append	Открыть файл, если он существует, и установить текущий указатель в конец файла. Если файл не существует, создать новый файл
Create	Создать новый файл. Если в каталоге уже существует файл с таким же именем, он будет стерт
CreateNew	Создать новый файл. Если в каталоге уже существует файл с таким же именем, возникает исключение **IOException**
Open	Открыть существующий файл
OpenOrCreate	Открыть файл, если он существует. Если нет, создать файл с таким именем
Truncate	Открыть существующий файл. После открытия он должен быть обрезан до нулевой длины

Методы классов **StreamWriter** и **BinaryReader** аналогичны методам классов **StreamWriter** и **StreamReader**.

Пример 3

База данных содержит одну таблицу – сведения о студентах группы. Каждая строка этой таблицы представляет собой упорядоченный набор значений: ФИО, Дата рождения, Группа, Адрес, Телефон. Напишем программу для работы с БД, выполняющую следующие функции: добавление в файл записи, считывание из файла. Листинг программы 3:

```
public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
            textBox1.Visible = false;
            textBox2.Visible = false;
            textBox3.Visible = false;
            textBox4.Visible = false;
            textBox5.Visible = false;
```

```
        }
        private void button1_Click(object sender, EventArgs e)
        {
  BinaryReader f = new BinaryReader(new
            FileStream("Студент", FileMode.Open));
            String s;
            int i = 1;
            while (f.PeekChar() >= 0)
            {
                if (i == 1) {
                    s = f.ReadString();
                    listBox1.Items.Add(s);
                    ++i;
                }
                if (i == 2)
                {
                    s = f.ReadString();
                    listBox2.Items.Add(s);
                    ++i;
                }
                if (i == 3)
                {
                    s = f.ReadString();
                    listBox3.Items.Add(s);
                    ++i;
                }
                if (i == 4)
                {
                    s = f.ReadString();
                    listBox4.Items.Add(s);
                    ++i;
                }
                if (i == 5)
                {
                    s = f.ReadString();
                    listBox5.Items.Add(s);
                    i = 1;
                }


            }
            f.Close();
        }
        private void button4_Click(object sender, EventArgs e)
        {
            listBox1.Items.Add(textBox1.Text);
            listBox2.Items.Add(textBox2.Text);
            listBox3.Items.Add(textBox3.Text);
            listBox4.Items.Add(textBox4.Text);
            listBox5.Items.Add(textBox5.Text);
            textBox1.Visible = false;
            textBox2.Visible = false;
            textBox3.Visible = false;
            textBox4.Visible = false;
            textBox5.Visible = false;
        }
        private void button2_Click(object sender, EventArgs e)
```

```
        {
            textBox1.Visible = true;
            textBox2.Visible = true;
            textBox3.Visible = true;
            textBox4.Visible = true;
            textBox5.Visible = true;
        }
        private void button3_Click(object sender, EventArgs e)
        {
    BinaryWriter f = new BinaryWriter(new
            FileStream("Студент", FileMode.Create));
            int k = listBox1.Items.Count;
            for (int i = 0; i < k; i++)
            {
                f.Write(listBox1.Items[i].ToString());
                f.Write(listBox2.Items[i].ToString());
                f.Write(listBox3.Items[i].ToString());
                f.Write(listBox4.Items[i].ToString());
                f.Write(listBox5.Items[i].ToString());
            }
            f.Close();
        }
}
```

Результат работы программы представлен на рисунке 4.

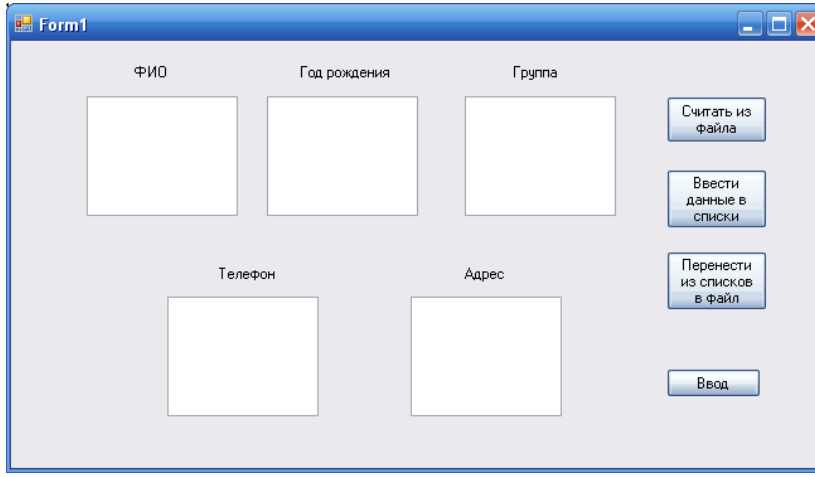

Рис. 4. *Результат работы программы*

4. ИСПОЛЬЗОВАНИЕ ДОПОЛНИТЕЛЬНЫХ КЛАССОВ И МЕТОДОВ ДЛЯ РАБОТЫ С ФАЙЛАМИ

Дополнительные методы работы с файлами содержит класс **File**.

Таблица 6. *Методы класса **File***

Метод	Описание
Create(String put)	Создание файла, **put**-путь к файлу;

Exists(String put)	Проверка существование файла;
Delete(String put)	Удаление файла;
Move(String put1, String put2)	Перемещение файла, находящегося в **put1** в **put2**
Copy(String put1, String put2)	Копирование файла, находящегося в **put1** в **put2**

Пример, демонстрирующий применение этих методов:

```
using System;
using System.IO;
class Test
{
    public static void Main()
    {
        // Создаем файл.
        File.Create("C:\\a.txt");
        // Проверка существования файла.
           if(File.Exists("C:\b.txt"))
           {
                   // Удаление файла.
                   File.Delete("C:\\1.txt");
           }
        // Переименование файла a.txt в b.txt.
    File.Move("C:\\a.txt", "C:\\b.txt");
        // Перемещениефайла.
        File.Move("C:\\c.txt", "D:\\c.txt");
        // Копированиефайла.
        File.Copy("D:\\z.txt", "D:\\x.txt");
    }
}
```

5. ЗАДАНИЕ

Создайте приложение для работы с различными видами текстовых файлов.