# Работа с файлами

Чтение из текстового файла

Запись в текстовый файл

Чтение произвольных файлов в шестнадцатеричном режиме

Чтение и запись двоичных данных

Поиск текста в файлах, заданных маской

- Под файлом понимают поименованные данные, хранящиеся на диске (кроме имени указывается иногда каталог).
- При открытии файла для чтения и/или записи, он автоматически становится потоком.
- Поток - это то, над чем можно производить операции чтения и записи. Данные, получаемые из сети, клавиатурный ввод и вывод на экран в консольных приложениях - являются примерами потоков.
- Для работы с файлами используется пространством имен System.IO , которое аккумулирует все необходимые классы.

# Фрагмент пространства System.IO:

- BinaryReader
- BinaryWriter
- BufferedStream
- Directory
- DirectoryInfo
- DirectoryNotFoundException
- EndOfStreamException
- ErrorEventArgs File
- FileInfo
- FileLoadException
- FileNotFoundException
- FileStream
- FileSystemEventArgs
- FileSystemInfo
- FileSystemWatcher
- InternalBufferOverflowException
- IODescriptionAttribute

1. StreamReader - Предназначен для чтения текстовых файлов или потоков.
2. StreamWriter - Данный класс используется для записи текста в новые или уже существующие файлы.
3. FileStream - Класс FileStream является базовым классом для открытия, чтения, записи и закрытия файлов. Данный класс наследуется от абстрактного класса Stream, следовательно большинство его свойств и методов являются производными из этого класса.
4. BinaryWriter - Класс, производный от Object. Предназначен для записи любой информации, не являющейся текстовой в файл.
5. BinaryReader - Класс, производный от Object. Предназначен для чтения любой информации, не являющейся текстовой в файл.
6. FileInfo, File - Эти классы предоставляют информацию о файлах. Оба класса предлагают одинаковые методы за исключением того, что методы File статические и требуют указания имени файла в качестве аргумента. Свойства и методы FileInfo нестатические, и имя файла, к которому будут применяться эти методы и свойства, указывается в аргументе конструктора.
7. DirectoryInfo,Directory - Эти классы предоставляют информацию о каталогах. Оба класса предлагают одинаковые методы за исключением того, что методы Directory статические и требуют указания имени каталога в качестве аргумента. Свойства и методы DirectoryInfo нестатические, и имя каталога, к которому будут применяться эти методы и свойства, указывает аргумент конструктора.

# Чтение из текстового файла

```
//считывает указанный пользователем файл построчно и выводит его на экран
// Класс для чтения текстовых файлов
   class ReadSomeFile    {
      static void Main(string [] args /* Параметры командной строки */  )
      {          string FileName;
         // Если в командной строке параметров нет
         if(args.Length == 0)          {
            Console.Write("Введите путь к файлу: ");
            FileName = Console.ReadLine();          }
         else          {
            FileName = args[0];          }
               try          {
            // Открываем поток для чтения файла  с кодировкой  по умолчанию
            StreamReader sr = new StreamReader(FileName, Encoding.Default);
            string line;
            // Считываем построчно до конца файла
            while((line = sr.ReadLine()) != null)              {
               // Вывод на экран
               Console.WriteLine(line);              }
            // Закрываем поток
            sr.Close();          }
         catch(Exception exc)          {
            // Сообщение об ошибке
            Console.WriteLine(exc.Message);          }        }   }
```

```
Введите путь к файлу:
Пустое имя пути не допускается.
Для продолжения нажмите любую клавишу . . .
```

```
Введите путь к файлу: g:\2.txt
Файл "g:\2.txt" не найден.
Для продолжения нажмите любую клавишу . . .
```

```
C:\Windows\system32\cmd.exe
Введите путь к файлу: g:\1.txt
Для ознакомления с методами этих классов
вы можете воспользоваться MSDN
Для продолжения нажмите любую клавишу . . .
```

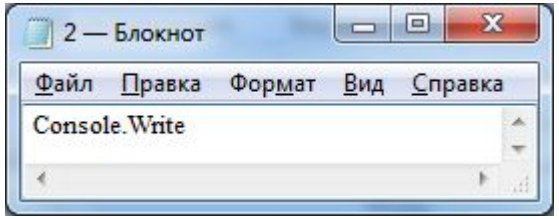
# Запись в текстовый файл

```
// программа записывает введенные пользователем с клавиатуры строки в файл, дописывая
      время начала и окончания работы пользователя
// Класс для записи текстовых файлов
   class WriteSomeFile    {
      static void Main()       {
         Console.WriteLine("Введите любой текст.");
         Console.WriteLine("Ввод пустой строки - признак окончания ввода.");
         // Открываем поток дял записи в файл с кодировкой по умолчанию
         StreamWriter sw = new StreamWriter("User.log", true,   Encoding.Default);
        string line;
         sw.WriteLine("----- Начало сеанса -----");
         // Запись текущего времени
         sw.WriteLine(DateTime.Now);
         sw.WriteLine("------------------------");
         sw.WriteLine();
         do           {
            // Считываем строку с клавиатуры
            line = Console.ReadLine();
            // Записываем строку в файл
            sw.WriteLine(line);
         } while(line != "");
         sw.WriteLine("----- Окончание сеанса -----");
         sw.WriteLine(DateTime.Now);
         sw.WriteLine("---------------------------");
         // Закрываем поток
         sw.Close();       }    }
```

```
мама мыла раму
рома чмтал книгу

----- Окончание сеанса -----
14.03.2011 10:35:00
------------------------
----- Начало сеанса -----
14.03.2011 10:37:14
------------------------

оллддд

----- Окончание сеанса -----
14.03.2011 10:37:56
------------------------
----- Начало сеанса -----
14.03.2011 10:45:48
------------------------

Поступай с людьми так как хотел
чтобы поступали с тобой....

----- Окончание сеанса -----
14.03.2011 10:47:01
```

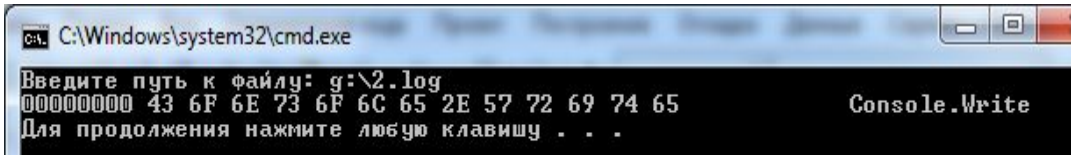
# Чтение произвольных файлов в шестнадцатеричном режиме

```
// Класс для считывания и отображения файла в 16-чном режиме
   class ReadHexFile    {
      static void Main(string [] args /* Параметры командной строки */)       {
         string FileName;
         // Если параметров командной строки нет
         if(args.Length == 0)          {
            Console.Write("Введите путь к файлу: ");
            FileName = Console.ReadLine();          }
         else          {
            FileName = args[0];          }
                  FileStream fs = null;
         try          {
            // Открываем поток на чтение
            fs = new FileStream(FileName, FileMode.Open, FileAccess.Read);          }
         catch(Exception exc)
         {
            // Вывод сообщения об ошибке
            Console.WriteLine(exc.Message);
            return;       }
```



# Чтение произвольных файлов в шестнадцатеричном режиме

```
// Выделяем массив под размер файла
        byte [] arr = new byte[fs.Length];
        // Считываем весь файл в массив
        int N = fs.Read(arr, 0, (int)fs.Length);
        // Закрываем поток
        fs.Close();
        int i = 0, j = 0, k = 0;
        // Отображаем информацию
        for(i = 0; i < N; i += 16)          {
           Console.Write("{0:X8}", i);
           for(j = 0; j < 16 && j + i < N; j++)            {
              Console.Write("{0,3:X2}", arr[j + i]);            }
               for(k = 0; k < 17 - j; k++)
              Console.Write("{0,3}", ' ');
           for(j = 0; j < 16 && j + i < N; j++)            {
              // Если управляющий символ
              if(Char.IsControl((char)arr[j + i]))
                 Console.Write('.');
              else
                 Console.Write((char)arr[j + i]);            }
           Console.WriteLine();          }       }   }
```



# Чтение и запись двоичных данных

```
// производится запись в файл и чтение из файла двоичных данных  показан механизм сохранения и загрузки данных класса в (из) файл (-а)
// Класс студент
class Student    {
    string firstname;    // Имя
    string lastname; // Фамилия
    string address; // Адрес
    string phone; // Телефон
        DateTime birthday; // Дата рождения
    int number; // Номер зачетки
    // Свойства
    public string FirstName       {
      get          {             return firstname;          }
      set          {             firstname = value;          }       }
    public string LastName       {
      get          {             return lastname;          }
      set          {             lastname = value;          }       }
    public string Address       {
      get          {             return address;          }
      set          {             address = value;          }       }
    public string Phone       {
      get          {             return phone;          }
      set          {             phone = value;          }       }
    public DateTime BirthDay       {
      get          {             return birthday;          }
      set          {             birthday = value;          }       }
    public int Number       {
      get          {             return number;          }
      set          {             number = value;          }       }
```

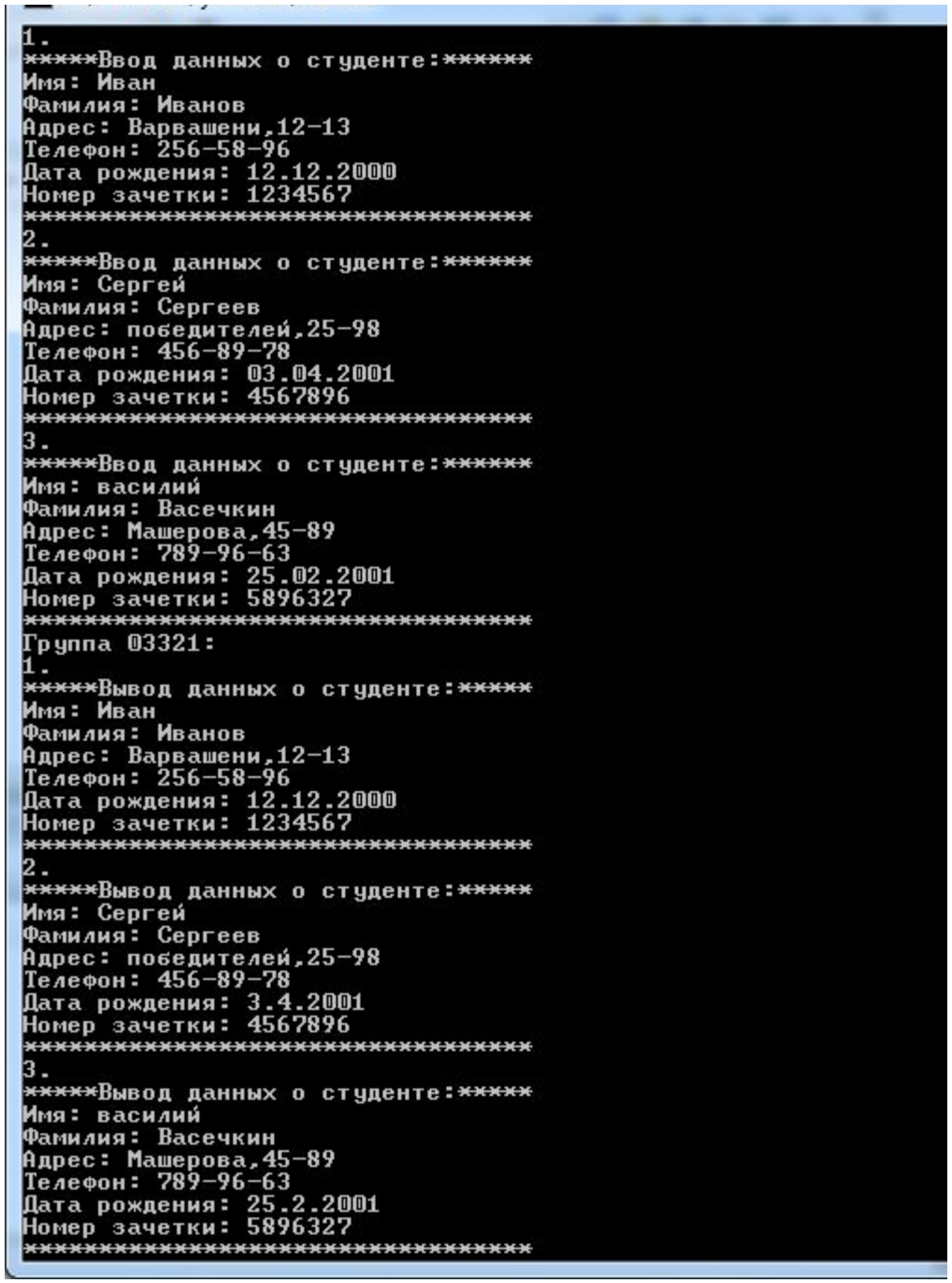
```
// Поверхностное копирование объекта
    public Student Clone()      {
      // Вызываем функцию базового класса (Object)  для поверхностного копирования
    объекта
      return (Student)MemberwiseClone();      }
// Ввод данных
    public void Input()      {
      Console.WriteLine("*****Ввод данных о студенте:******");
      Console.Write("Имя: ");                 firstname = Console.ReadLine();
      Console.Write("Фамилия: ");         lastname = Console.ReadLine();
      Console.Write("Адрес: ");              address = Console.ReadLine();
      Console.Write("Телефон: ");           phone = Console.ReadLine();
      Console.Write("Дата рождения: ");
      try          {
        // Считывание даты
        birthday = Convert.ToDateTime(Console.ReadLine());          }
      catch          {
        Console.WriteLine("Ошибка ввода, используется текущая дата");
      birthday = DateTime.Now;          }
      Console.Write("Номер зачетки: ");
       try          {
        number = Convert.ToInt32(Console.ReadLine());          }
      catch          {
        Console.WriteLine("Ошибка ввода, используется номер 0");
        number = 0;          }
      Console.WriteLine("************************************");      }
```

```
// Вывод данных
    public void Print()      {
       Console.WriteLine("*****Вывод данных о студенте:*****");
      Console.WriteLine("Имя: {0}", firstname);          Console.WriteLine("Фамилия: {0}", lastname);
       Console.WriteLine("Адрес: {0}", address);          Console.WriteLine("Телефон: {0}", phone);
       Console.WriteLine("Дата рождения: {0}.{1}.{2}",  birthday.Day, birthday.Month, birthday.Year);
       Console.WriteLine("Номер зачетки: {0}", number);
       Console.WriteLine("**********************************");      }
    // Запись в файл
    public void Write(BinaryWriter bw)      {
       // Все данные записываются по отдельности
       bw.Write(firstname);
       bw.Write(lastname);
       bw.Write(address);
       bw.Write(phone);
       bw.Write(birthday.Year);
       bw.Write(birthday.Month);
       bw.Write(birthday.Day);
       bw.Write(number);      }
    // Статический метод для чтения из файла информации и создания нового объекта на ее основе
    public static Student Read(BinaryReader br)      {
```

```
        // Считывание производится в порядке,  соответствующем записи
        Student st = new Student();
        st.firstname = br.ReadString();
        st.lastname = br.ReadString();
        st.address = br.ReadString();
        st.phone = br.ReadString();
        int year = br.ReadInt32();
        int month = br.ReadInt32();
        int day = br.ReadInt32();
        st.birthday = new DateTime(year, month, day);
        st.number = br.ReadInt32();
        return st;       }    }
// Класс Group
class Group : ICloneable    {
      string groupname;         // Название группы
          Student [] st;           // Массив студентов
    // Свойства
    public string GroupName       {
       get          {             return groupname;          }
       set          {             groupname = value;          }       }
    public Student [] Students       {
       get          {             return st;          }
       set          {             st = value;          }       }
```

```
// Конструктор, получающий название группы и количество студентов
      public Group(string gn, int n)       {
         groupname = gn;
         // По умолчанию в группе 10 студентов
         if(n < 0 || n > 10)
            n = 10;
         st = new Student[n];
         // Создаем студентов
         for(int i = 0; i < n; i++)
            st[i] = new Student();       }
      // Аналог конструктора копирования
      public Group(Group gr)       {
         // Создаем массив студентов
         st = new Student[gr.st.Length];
         // Копируем название группы
         groupname = gr.groupname;
         // Копируем каждого индивидуума
         for(int i = 0; i < gr.st.Length; i++)
            st[i] = gr.st[i].Clone();       }
      // Заполняем группу
      public void Input()       {
         for(int i = 0; i < st.Length; i++)          {
            Console.WriteLine("{0}.", i + 1);
            st[i].Input();          }       }
      // Изменение данных конкретного студента
      public void InputAt(int n)       {
         if(st == null || n >= st.Length || n < 0)
            return;
         st[n].Input();       }
      // Вывод списка группы
```

```
// Возврат независимой копии группы
        return gr;      }
    // Запись в файл
    public void Write(BinaryWriter bw)      {
        // Сохраняем название группы
        bw.Write(groupname);
        // Сохраняем количество студентов
        bw.Write(st.Length);
        for(int i = 0; i < st.Length; i++)
            // Для сохранения студента вызывается
            // соответствующий метод из класса Student
            st[i].Write(bw);      }
    // Статический метод для чтения из файла информации и создания нового объекта на ее
    основе
    public static Group Read(BinaryReader br)      {
        string gn = br.ReadString();
        int n = br.ReadInt32();
        Student [] st = new Student[n];
        for(int i = 0; i < n; i++)
            // Для считывания студента вызывается    соответствующий метод из класса Student
            st[i] = Student.Read(br);
        // Создаем пустую группу
        Group gr = new Group(gn, 0);
        // Записываем в нее студентов
        gr.st = st;          return gr;      }   }
// Тестирование
class Test   {
    static void Main()      {
        // Группа
        Group gr = new Group("03321", 3);
```


1.
*****Ввод данных о студенте:******
Имя: Иван
Фамилия: Иванов
Адрес: Варвашени,12-13
Телефон: 256-58-96
Дата рождения: 12.12.2000
Номер зачетки: 1234567
**********
2.
*****Ввод данных о студенте:******
Имя: Сергей
Фамилия: Сергеев
Адрес: победителей,25-98
Телефон: 456-89-78
Дата рождения: 03.04.2001
Номер зачетки: 4567896
**********
3.
*****Ввод данных о студенте:******
Имя: василий
Фамилия: Васечкин
Адрес: Машерова,45-89
Телефон: 789-96-63
Дата рождения: 25.02.2001
Номер зачетки: 5896327
**********
Группа 03321:
1.
*****Вывод данных о студенте:*****
Имя: Иван
Фамилия: Иванов
Адрес: Варвашени,12-13
Телефон: 256-58-96
Дата рождения: 12.12.2000
Номер зачетки: 1234567
**********
2.
*****Вывод данных о студенте:*****
Имя: Сергей
Фамилия: Сергеев
Адрес: победителей,25-98
Телефон: 456-89-78
Дата рождения: 3.4.2001
Номер зачетки: 4567896
**********
3.
*****Вывод данных о студенте:*****
Имя: василий
Фамилия: Васечкин
Адрес: Машерова,45-89
Телефон: 789-96-63
Дата рождения: 25.2.2001
Номер зачетки: 5896327
**********

# Поиск текста в файлах, заданных маской

```
//производится поиск указанного текста в файлах, соответствующих введенной маске
//Маска задается в формате MSDOS: * - любой символ в любом количестве, ? - один любой символ
//Для поиска файлов и текста в них используются регулярные выражения
// Класс для поиска текста в файлах, заданных DOS'вской маской
class Search    {
    static void Main()       {
        Console.Write("Введите путь к каталогу: ");
        string Path = Console.ReadLine();
        Console.Write("Введите маску для файлов: ");
        string Mask = Console.ReadLine();
        Console.Write("Введите текст для поиска в файлах: ");
        string Text = Console.ReadLine();
        // Дописываем слэш (в случае его отсутствия)
        if(Path[Path.Length - 1] != '\\')
            Path += '\\‘;
        // Создание объекта на основе введенного пути
        DirectoryInfo di = new DirectoryInfo(Path);
        // Если путь не существует
        if(!di.Exists)          {
            Console.WriteLine("Некорректный путь!!!");
            return;          }
        // Преобразуем введенную маску для файлов  в регулярное выражение
        // Заменяем . на \.
```

```
    Mask = Mask.Replace(".", @"\." /* (".", "\\.") */);
    // Заменяем ? на .
    Mask = Mask.Replace("?", ".");
    // Заменяем * на .*
    Mask = Mask.Replace("*", ".*");
    // Указываем, что требуется найти точное соответствие маске
    Mask = "^" + Mask + "$";
     // Создание объекта регулярного выражения     на основе маски
    Regex regMask = new Regex(Mask, RegexOptions.IgnoreCase);
    // Экранируем спецсимволы во введенном тексте
    Text = Regex.Escape(Text);
    // Создание объекта регулярного выражения     на основе текста
    Regex regText = new Regex(Text, RegexOptions.IgnoreCase);
    // Вызываем функцию поиска
    ulong Count = FindTextInFiles(di, regText, regMask);
    Console.WriteLine("Всего обработано файлов: {0}.", Count);       }
// Функция поиска
static ulong FindTextInFiles(DirectoryInfo di, Regex regText, Regex regMask)       {
    // Поток для чтения из файла
    StreamReader sr = null;
    // Список найденных совпадений
    MatchCollection mc = null;
// Количество обработанных файлов
    ulong CountOfMatchFiles = 0;
    FileInfo [] fi = null;
    try          {
       // Получаем список файлов
       fi = di.GetFiles();          }
    catch          {             return CountOfMatchFiles;          }
```

```
    // Перебираем список файлов
        foreach(FileInfo f in fi)        {
          // Если файл соответствует маске
          if(regMask.IsMatch(f.Name))          {
       ++CountOfMatchFiles;  // Увеличиваем счетчик
            Console.WriteLine("File " + f.Name + ":");
sr = new StreamReader(di.FullName + @"\" + f.Name,   Encoding.Default); // Открываем файл
       string Content = sr.ReadToEnd();     // Считываем целиком
       sr.Close(); // Закрываем файл
       mc = regText.Matches(Content);     // Ищем заданный текст
            // Перебираем список вхождений
            foreach(Match m in mc)          {
              Console.WriteLine("Текст найден в позиции {0}.", m.Index);          }
/* Отладочная информация*/
       if(mc.Count == 0)          {
            Console.WriteLine("В данном файле запрошенный текст не найден.");          }
     }       }
/* Отладочная информация*/
     Console.WriteLine("Количество обработанных файлов в каталоге {0} {1}",
            di.FullName, CountOfMatchFiles);
              // Получаем список подкаталогов
       DirectoryInfo [] diSub  = di.GetDirectories();
       // Для каждого из них вызываем (рекурсивно)    эту же функцию поиска
       foreach(DirectoryInfo diSubDir in diSub)
          CountOfMatchFiles += FindTextInFiles(diSubDir, regText, regMask);
            return CountOfMatchFiles;      }  } // Возврат количества обработанных файлов
```