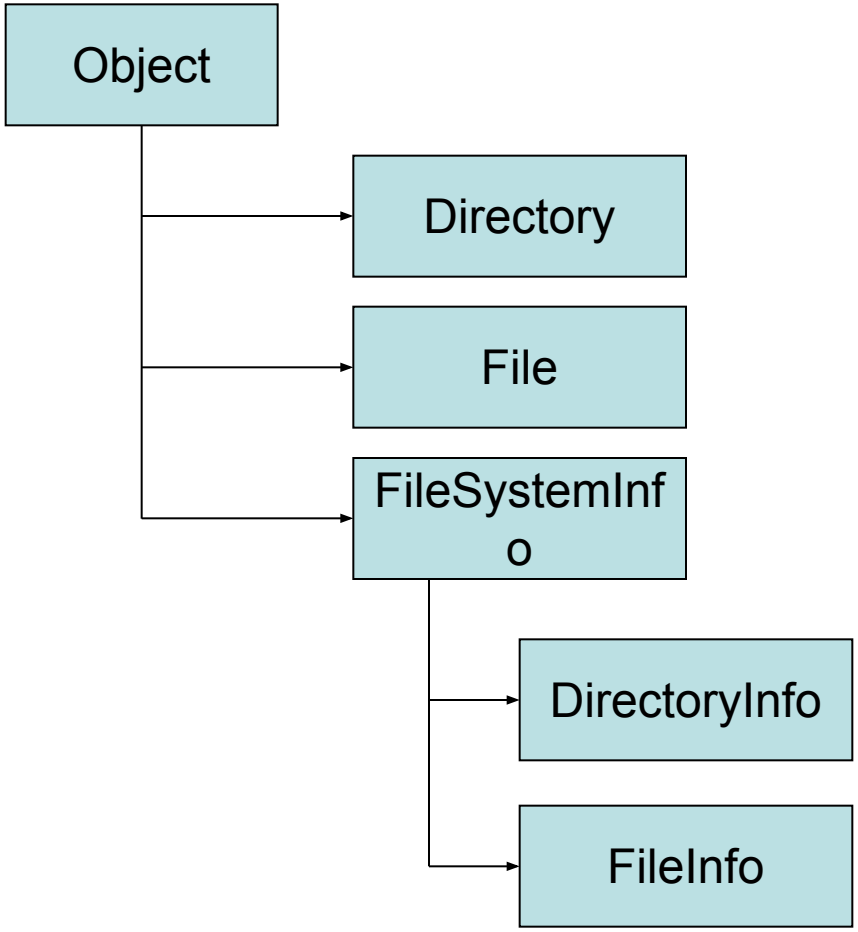
Ввод-вывод

Сериализация объектов

Наиболее важные классы пространства имен System.IO

BinaryReader BinaryWriter	Работа со стандартными типами данных в двоичном виде
BufferedStream	Временное хранилище для потока байтов
Directory, DirectoryInfo File, FileInfo	Работа с каталогами и файлами. File и Directory – наборы стат. мет.
FileStream	Обеспечивает произвольный доступ к файлу как потоку байтов
MemoryStream	То же, но для потока байтов в оперативной памяти
StreamWriter, StreamReader	Последовательные считывание и запись в файл текстовой информации
StringReader, StringWriter	То же, но для оперативной памяти

Иерархия типов



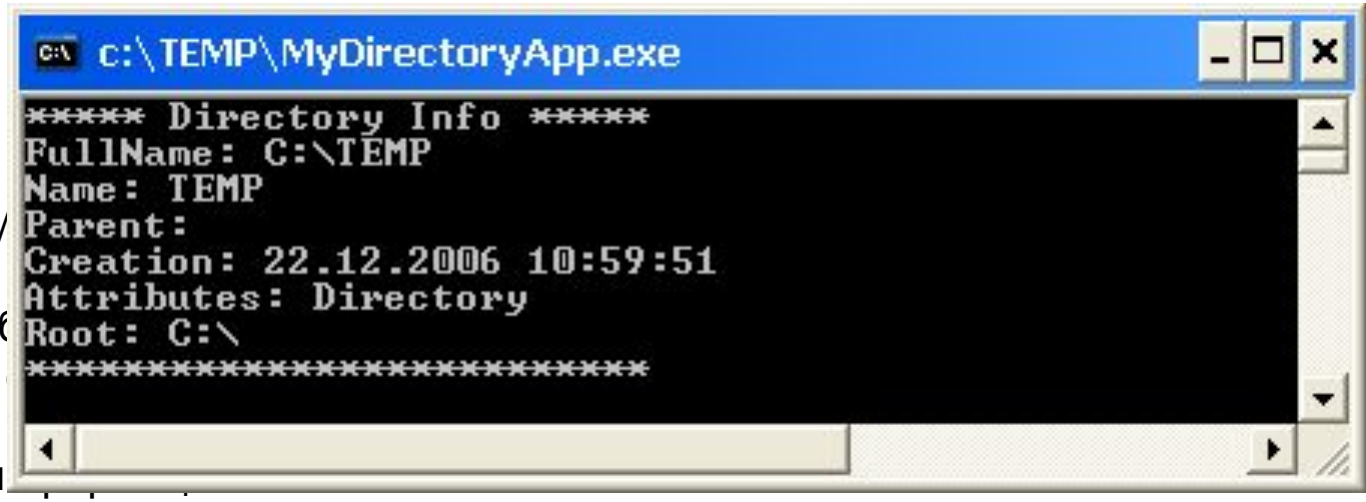
Свойства и методы класса FileSystemInfo

Attributes	Получить, установить атрибуты
CreationTime	Время создания
Exists	Существует ли данный объект ф.с.
Extension	Расширение файла
FullName	Полное (с путем) имя файла
LastAccessTime	Время последнего доступа к файлу
LastWriteTime	Время последней записи в файл
Name	Имя файла
Delete()	Удалить объект ф.с.
Refresh()	Обновить информацию о ф.с.

Тип DirectoryInfo

Create() CreateSubDirectory()	Создают каталог или подкаталог
Delete()	Удаляет каталог со всем содержимым
GetDirectories()	Возвращает массив строковых значений, представляющих все подкаталоги
GetFiles()	Файлы текущего каталога (массив объектов FileInfo)
MoveTo()	Перемещает каталог и все его содержимое на новое место
Parent	Возвращает родительский каталог

Работа с DirectoryInfo



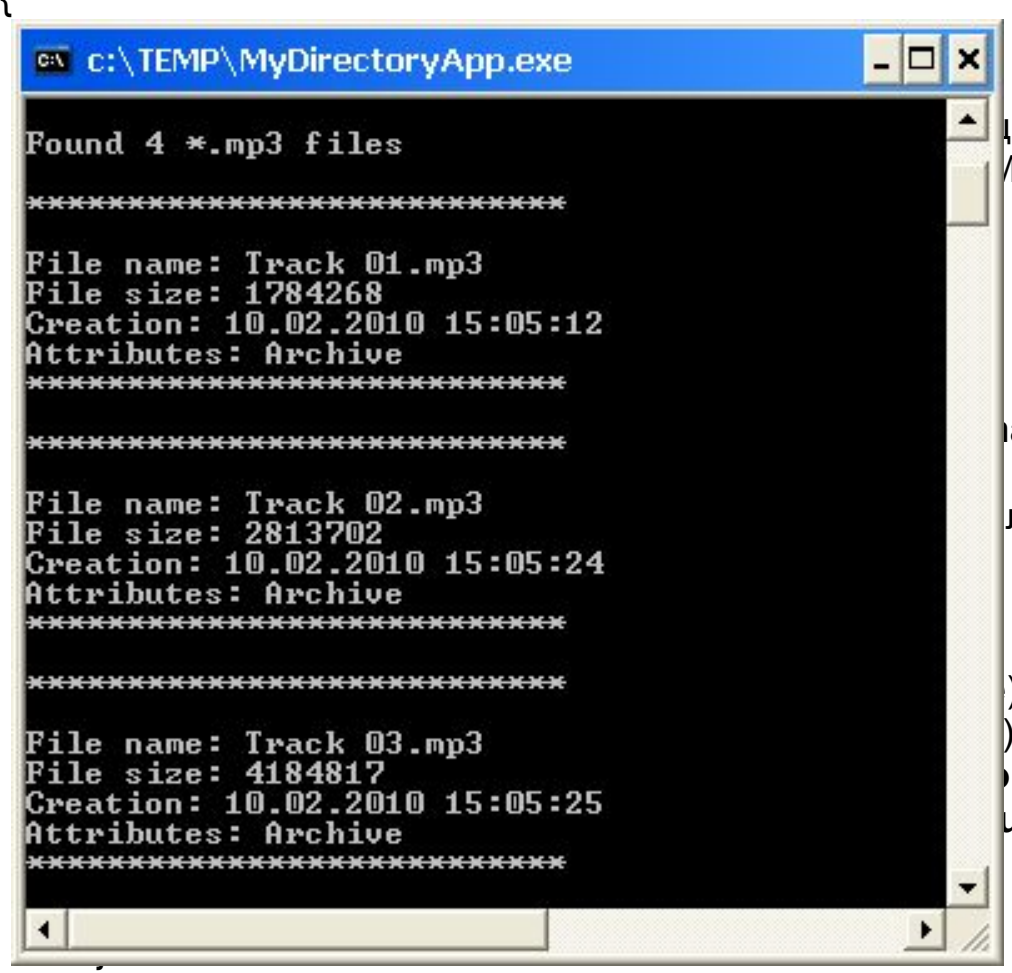

```
class MyDirectory
{
        public static void M
        {
                // Создаем об
                DirectoryInfo

                // Выводим и
                Console.WriteLine("***** Directory Info *****");
                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);
                Console.WriteLine("**********\n");
        }
}
```

Перечисление FileAttributes

Archive	Файл для резервного копирования
Compressed	Сжатый файл
Directory	Файл является каталогом
Encrypted	Шифрованный файл
Hidden	Скрытый файл
Normal	Обычный
Offline	На сервере, но кэширован
ReadOnly	Только для чтения
System	Системный

Доступ к файлам через Directory и FileInfo

```
class MyDirectory
{
```



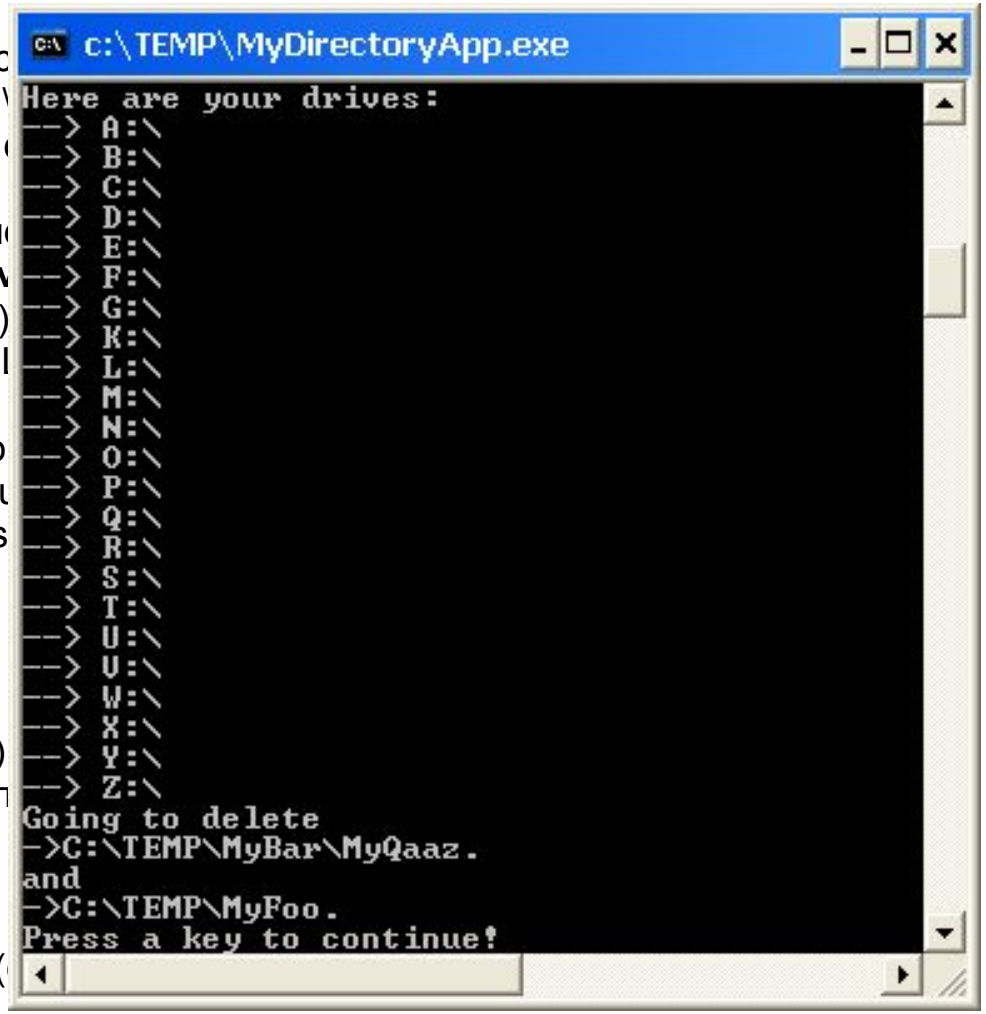

```
ий C:\TEMP
MP");

apFiles.Length);

ле

);
);
nTime);
utes.ToString());
}
```

Создание подкаталогов при помощи класса DirectoryInfo

```
class MyDirectory
{
	public static void Main(String[] args)
	{
		DirectoryInfo dir = new DirectoryInfo(@"C:\TEMP");
		...

		// Создаем в C:\TEMP новые подкаталоги
		try
		{
			// Создаем C:\TEMP\MyFoo
			dir.CreateSubdirectory("MyFoo");
			Console.WriteLine("Created: {0}", d.FullName);

			// Создаем C:\TEMP\MyBar\MyQaaz
			dir.CreateSubdirectory(@"MyBar\MyQaaz");
	Console.WriteLine("Created: {0}", d.FullName);
		}
		catch(IOException e) { Console.Write(e.Message);}
	}
}
```

Статические члены класса Directory

```
public static void Main(String[] args)
{
  // Создаем объект DirectoryInfo, соответс
  DirectoryInfo dir = new DirectoryInfo(@"C:\
  // А теперь воспользуемся несколькими

  // Выводим информацию обо всех логич
  string[ ] drives = Directory.GetLogicalDriv
  Console.WriteLine("Here are your drives:")
  foreach(string s in drives)  { Console.Writel

  // Удаляем только что созданные катало
  Console.Write(Going to delete\n->" + dir.Fu
          dir.FullName + " \\MyFoo.\n" + "Pres
  Console.Read();

  try
  {
      Directory.Delete(@"C:\TEMP\MyFoo")
    // Необязательный второй параметр оп
      Directory.Delete(@"C:\TEMP\MyBar",
  }

  catch(IOException e) { Console.WriteLine(
}
```



Класс FileInfo (1)

AppendText()	Создает объект StreamReader для добавления текста к файлу
CopyTo()	Копирует существующий файл в новый
Create()	Создает файл и возвращает объект FileStream
CreateText()	Создает объект StreamWriter для записи текстовых данных в новый файл
Delete()	Удаляет файл, которому соответствует объект FileInfo
Directory	Возвращает каталог, в котором расположен файл
DirectoryName	Возвращает полный путь к файлу

Класс FileInfo (2)

Length	Возвращает размер файла
MoveTo()	Перемещает файл (и/или переименовывает)
Name	Возвращает имя файла
Open()	Открывает файл с указанными правами доступа или для совместного использования
OpenRead()	Создает объект FileStream, доступный только для чтения
OpenText()	Создает объект StreamReader, который позволяет считывать текстовую информацию
OpenWrite()	Создает объект FileStream, доступный для чтения и записи

Манипуляция файлом

```
public class FileManipulator
{
	public static int Main(string[] args)
	{
		// Создаем новый файл в корневом каталоге диска C:
		FileInfo f = new FileInfo(@"C:\Test.txt");
		FileStream fs = f.Create();

		// Выводим основную информацию о созданном нами файле
		Console.WriteLine("Creation: {0}", f.CreationTime);
		Console.WriteLine("Full Name: {0}", f.FullName);
		Console.WriteLine("Full atts: {0}", f.Attributes.ToString());
		Console.Write("Press a key to delete file");
		Console.Read();

		// Закрываем FileStream и удаляем файл
		fs.Close();
		f.Delete();

		return 0;
	}
}
```

Использование метода FileInfo.Open()

```
// Get a new FileStream object.
FileInfo f2 = new FileInfo(@"C:\HelloThere.ini");
FileStream  s = f2.Open (FileMode.OpenOrCreate,
                FileAccess.ReadWrite,
                FileShare.None);

        // Write 20 bytes to the dat file...
        for(int i = 0; i < 256; i++)
        {
              s.WriteByte((byte)i);
        }
```

Значение перечисления FileMode

Append	Открывает файл, если существует, и ищет конец этого файла или создает новый файл.
Create	Создает новый, перезаписывая старый, если есть.
CreateNew	Создает новый. Если такой уже есть – исключение IOException
Open	Открыть существующий файл.
OpenOrCreate	Открыть существующий или создать.
Truncate	Открыть и обнулить.

Значение перечисления FileAccess

Read	Файл будет открыт только для чтения
ReadWrite	Файл будет открыт только для чтения и записи
Write	Только для добавления данных без считывания

Значение перечисления FileShare

Read	Позволяет открывать другим пользователям на чтение
ReadWrite	На чтение и запись
Write	Только на запись
None	Запрещает совместное использование файла

Методы FileInfo.OpenRead и FileInfo.OpenWritle

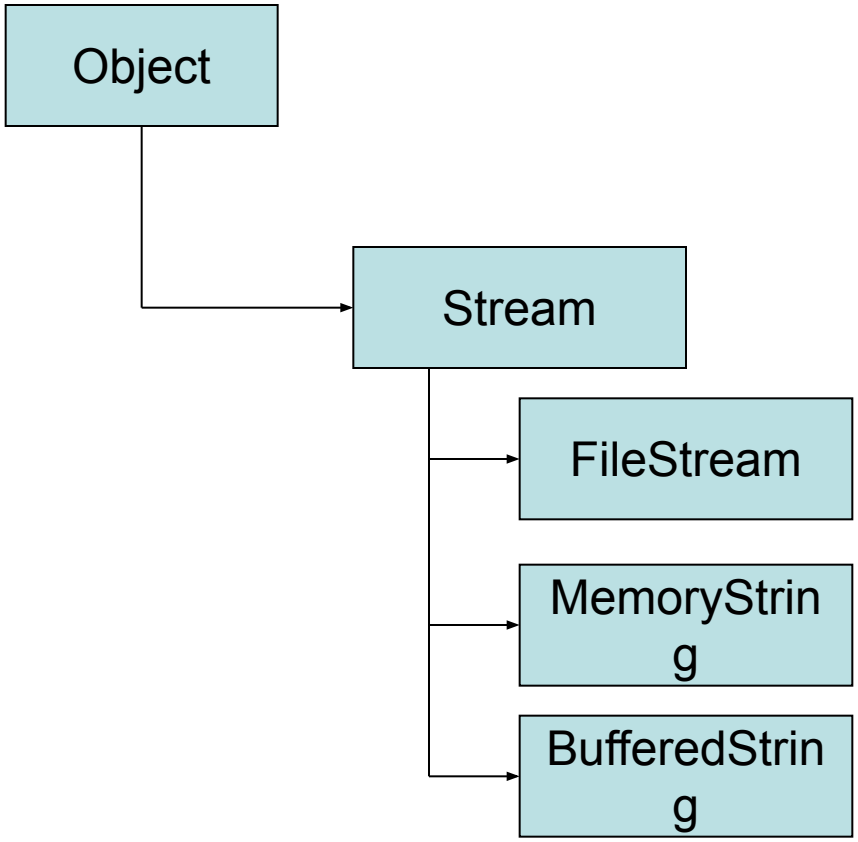
```
FileInfo fr = new FileInfo(@"C:\boot.ini");
FileStream readOnlyStream = fr.OpenRead();
readOnlyStream.Close();
```

```
FileInfo fw = new FileInfo(@"C:\boot.ini");
FileStream writeOnlyStream = fw.OpenWrite();
writeOnlyStream.Close();
```

Методы FileInfo.OpenText, FileInfo.CreateText, FileInfo.AppendText

```
FileInfo fs = new FileInfo(@"C:\boot.ini");
StreamReader sReader = fs.OpenText();
sReader.Close();

FileInfo fs = new FileInfo(@"C:\test.txt");
fs.Open(FileMode.Create,FileAccess.ReadWrite)
StreamWriter sWriter = fs.CreateText();
sWriter.Close();
```

Абстрактный класс Stream



Члены класса Stream

CanRead CanSeek CanWrite	Определяет возможности, которые может поддерживать данный поток
Close()	Закрывает текущий поток
Flush()	Сбрасывает незавершенные операции на диск
Length	Возвращает длину потока в байтах
Position	Определяет позицию в текущем потоке
Read() ReadByte()	Читают последовательность байтов или байт
Seek()	Устанавливает позицию в текущем потоке
SetLength()	Устанавливает длину текущего потока
Write() WriteByte()	Записывают последовательность байтов или байт

Работа с объектом FileStream

```
FileStream  s = f2.Open(FileMode.OpenOrCreate, FileAccess.ReadWrite,
                                FileShare.None);

// Write 20 bytes to the dat file...
for(int i = 0; i < 256; i++)
{
  s.WriteByte((byte)i);
}

// Reset internal position.
s.Position = 0;

// Read 20 bytes from the dat file...
for(int i = 0; i < 256; i++)
{
  Console.Write(s.ReadByte());
}

s.Close();
```

Работа с объектом MemoryStream

```
// Создаем объект MemoryStream точно определенного объема
MemoryStream myMemStream = new MemoryStream();
myMemStream.Capacity = 256;

// Записываем байты в myMemStream
for(int i = 0; i < 256; i++)
{
      myMemStream.WriteByte((byte)i);
}

// Переставляем внутренний указатель на начало
myMemStream.Position = 0;

// Считываем байты из потока
for(int i = 0; i < 256; i++)
{
      Console.Write(myMemStream.ReadByte());
}
myMemStream.Close();
```

Наиболее важные члены MemoryStream

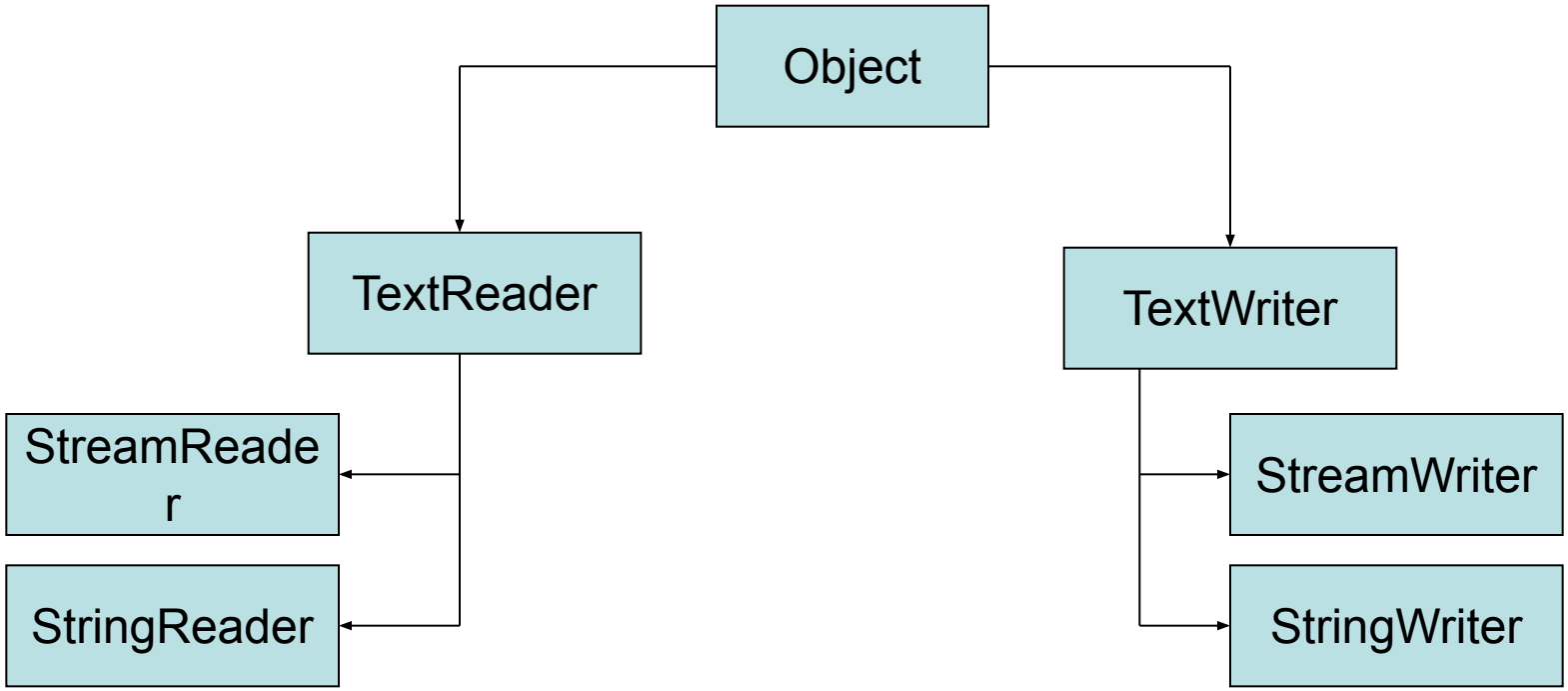
Capacity	Получить/установить количество байтов, выделенных под этот поток
GetBuffer()	Массив байтов, при помощи которых поток был создан
ToArray()	Записывает все содержимое потока в массив байтов (вне зависимости от Position)
WriteTo()	Записывает все содержимое в другой объект, производный от Stream (например, FileStream)

```
FileStream dump= new FileStream("dump.dat",FileMode.Create,FileAccess.ReadWrite);
myMemStream.WriteTo(dump);

byte[ ] bytesinMemory = myMemStream.ToArray();
myMemStrea.Close();
```

Класс BufferedStream

```
// Создаем объект BufferedStream
    BufferedStream myFileBuffer = new BufferedStream(dumpFile);

    // Добавляем несколько байт
    byte [ ] str = {127, 0x77, 0x4, 0x0, 0x0, 0x16};
    myFileBuffer.Write(str, 0, str.Length);


    myFileBuffer.Close();   // Сброс на диск только сейчас
```

Классы StreamReader и StreamWriter



Наиболее важные члены TextWriter

Close()	Закрывает соответствующий объект
Flush()	Очищает все буфуры и сбрасывает их содержимое на диск
NewLine	Используется для определения последовательности символов, означающих начало новой строки
Write()	Записывает новый отрезок текста в поток без применения последовательности символов новой строки
WriteLine()	Записывает новую строку, добавляя символы NewLine

Запись в текстовый файл

```
// Создаем файл
FileInfo f = new FileInfo("Thoughts.txt");

// Получаем объект StreamWriter и с его помощью записываем в файл
// несколько строк текста
StreamWriter writer = f.CreateText();
writer.WriteLine("Don't forget Mother's Day this year...");
writer.WriteLine("Don't forget Father's Day this year...");
writer.WriteLine("Don't forget these numbers:");

for(int i = 0; i < 10; i++);
{
  writer.Write(i + " ");
}
// Вставляем символ начала новой строки
writer.Write(writer.NewLine);

// Метод Close() автоматически очищает все буферы!
writer.Close();
Console.WriteLine("Created file and wrote some thoughts...");
```

Наиболее важные члены TextReader

Peek()	Возвращает следующий символ, не изменяя текущую позицию
Read()	Считывает данные из потока на входе
ReadBlock()	Считывает указанное количество символов с определенной позиции
ReadLine()	Считывает строку
ReadToEnd()	Считывает данные до конца потока (возвращает как единое значение типа string)

Чтение текстового файла

```
// А теперь выводим информацию из файла на консоль при помощи
// StreamReader

Console.WriteLine("Here are your thoughts:\n");
StreamReader sr = File.OpenText("Thoughts.txt");

string input = null;
while ((input = sr.ReadLine()) !=null)
    {
      Console.WriteLine(input);
    }
sr.Close();
```

```
string alldata = st.ReadToEnd();
sr.Close();
```

Класс StringWriter

```
// Получаем объект StringWriter и с его помощью записываем
// в файл несколько строк текста
StringWriter writer = new StringWriter();
writer.WriteLine("Don't forget Mother's Day this year...");
writer.WriteLine("Don't forget Father's Day this year...");
writer.WriteLine("Don't forget these numbers:");

for(int i = 0; i < 10; i++);
{
     writer.Write(i + " ");
}

// Вставляем символ начала новой строки
writer.Write(writer.NewLine);

// Метод Close() автоматически очищает все буферы!
writer.Close();
Console.WriteLine("Stored thoughts in a StringWriter...");

// Получаем копию содержимого StringBuffer (в виде значения типа string)
// и выводим ее на консоль
Console.WriteLine("Contents: {0}", writer.ToString());
```

Доступ через StringWriter.GetStringBuilder()

```
// Получаем объект StringBuilder и выводим его содержимое
StringBuilder str = writer.GetStringBuilder();
string allOfTheData = str.ToString();
Console.WriteLine("StringBuilder says:\n{0} ", allOfTheData);

// Вставляем в буфер новый элемент, позиция вставки 20
str.Insert(20, "INSERTED STUFF");
allOfTheData = str.ToString();
Console.WriteLine("New StringBuilder says:\n{0}", allOfTheData);

// Удаляем вставленный элемент
str.Remove(20, "INSERTED STUFF".Length);
allOfTheData = str.ToString();
Console.WriteLine("Original says:\n{0}", allOfTheData);
```

Доступ через StringWriter.GetStringBuilder()

```
// Получаем объект StringBuilder и выводим его содержимое
StringBuilder str = writer.GetStringBuilder();
string allOfTheData = str.ToString();
Console.WriteLine("StringBuilder says:\n{0} ", allOfTheData);

// Вставляем в буфер новый элемент, позиция вставки 20
str.Insert(20, "INSERTED STUFF");
allOfTheData = str.ToString();
Console.WriteLine("New StringBuilder says:\n{0}", allOfTheData);

// Удаляем вставленный элемент
str.Remove(20, "INSERTED STUFF".Length);
allOfTheData = str.ToString();
Console.WriteLine("Original says:\n{0}", allOfTheData);
```

Доступ через StringWriter.GetStringBuilder()

```
StringReader sr = new StringReader(writer.ToString());

string input = null;
while ((input = sr.ReadLine()) != null)
{
  Console.WriteLine (input);
}

sr.Close();
```

Наиболее важные члены BinaryWriter

BaseStrem	Возвращает поток, с которым работает BinaryWriter
Close()	Закрывает поток
Flush()	Очищает буфер
Seek()	Устанавливает позицию в текущем потоке
Write()	Записывает значение

Наиболее важные члены BinaryReader

BaseStrem	Возвращает поток, с которым работает BinaryWriter
Close()	Закрывает поток
PeekChar()	Возвращает текущий символ
Read()	Считывает поток байтов и сохраняет в массиве
ReadXXXX()	Считывает данные определенного типа (Int32, Byte)

Запись двоичных файлов

```
FileStream myFStream = new FileStream("temp.dat",
                        FileMode.OpenOrCreate, FileAccess.ReadWrite);

// Записываем двоичные данные
BinaryWriter binWrit  = new BinaryWriter(myFStream);
binWrit.WriteString("Hello as binary info...");
int myInt = 99;
float myFloat = 9984.82343F;
bool myBool = false;
char[] myCharArray = {'H', 'e', 'l', 'l', 'o'};
binWrit.Write(myInt);
binWrit.Write(myFloat);
binWrit.Write(myBool);
binWrit.Write(myCharArray);
```

Чтение двоичных файлов

```
// Устанавливаем внутренний указатель на начало
binWrit.BaseStream.Position = 0;

// Считываем двоичную информацию как поток байтов
Console.WriteLine("Reading binary data...");
BinaryReader binRead = new BinaryReader(myFStream);
int temp = 0;
while(binRead.PeekChar() != -1)
{    Console.Write(binRead.ReadByte());
     temp = temp + 1;
     if(temp = = 5)
     { // Добавляем пустую строку через каждые 5 байтов
       temp = 0;
       Console.WriteLine();
     }
}
// Все закрываем
binWrit.Close();    binRead.Close();  myFStream.Close();
```

Вывод стандартных объектов

```
// Открываем файл изображения в каталоге приложения
Console.WriteLine("Modifying a bitmap in memory");
myFStream = new FileStream("Paint Splatter.bmp", FileMode.Open,
    FileAccess.ReadWrite);

// Создаем объект Bitmap на основе открытого потока
Bitmap rawBitmap = new Bitmap(myFStream);

// Рисуем белый крест поперек изображения (наш код применим лишь в том
// случае,  если высота и ширина изображения одинаковы)
for(int i = 0; i < rawBitmap.Width; i++)
{
      rawBitmap.SetPixel(i, i, Color.White);
      rawBitmap.SetPixel(rawBitmap.Width - i) - 1, i - 1, Color.White);
}

// А теперь сохраняем измененное изображение в файл
rawBitmap.Save("newImage.bmp");
myFStream.Close();
```

Сохранение объектов .NET

- Сериализация – процесс преобразования объекта в линеную последовательность байт
- Сериализация подразумевает сохранение информации и родительских объектах

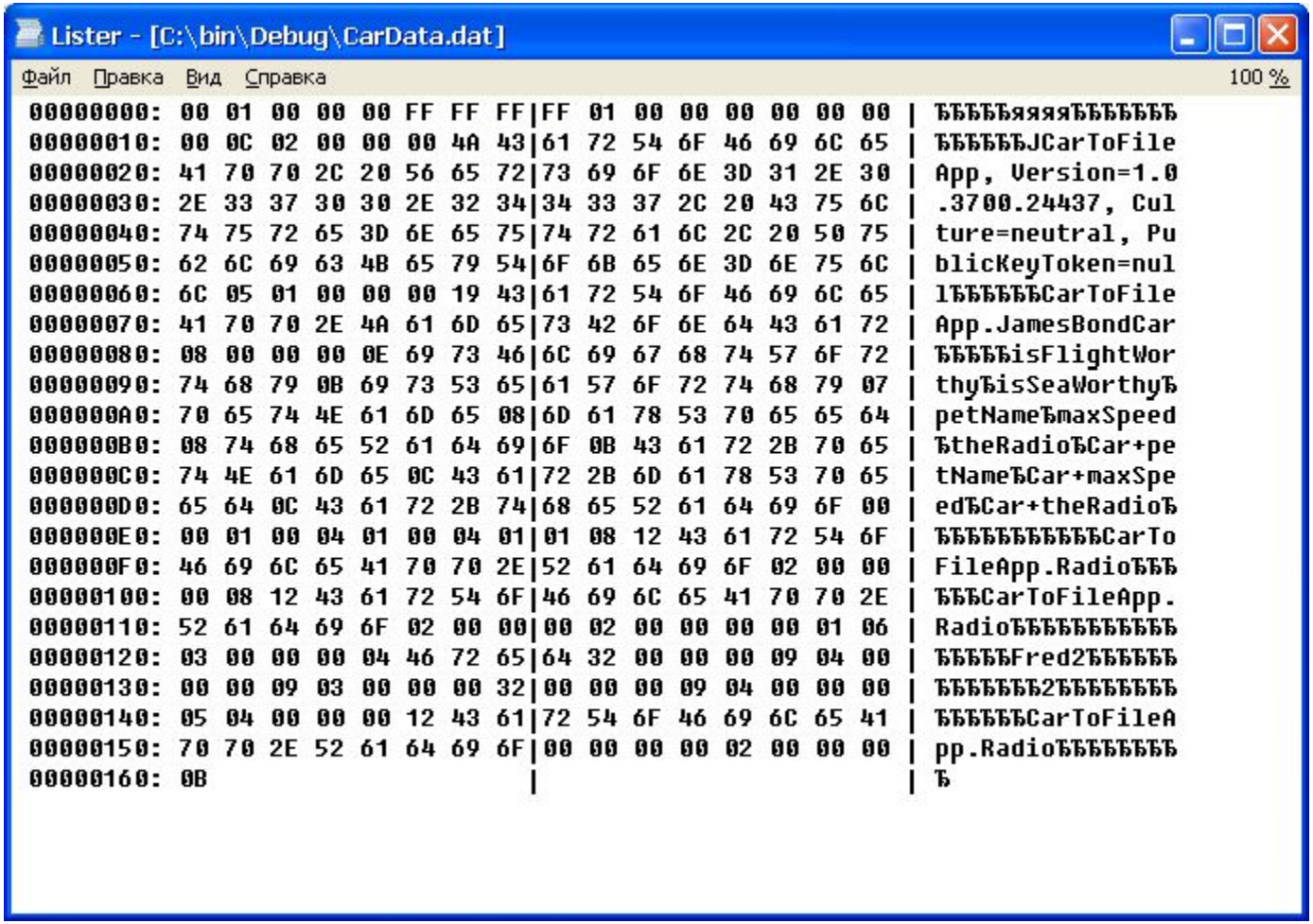
Настройка объектов для сериализации

```
// Класс Radio может быть сериализован
[Serializable]
public class Radio
{
	// Однако нам нет необходимости сохранять это число
	[NonSerialized]
	private int objectIDNumber = 9;

	public Radio(){}
	public void On(bool state)
	{
		if(state = = true)
			MessageBox.Show("Music is on...");
		else
			MessageBox.Show("No tunes...");
	}
}
```

Сериализация в двоичном формате

```
using System.Runtime.Serialization.Formatters.Binary;

public static void Main()
{
    // Создаем объект JamesBondCar и выполняем с ним всякие действия
    JamesBondCar myAuto = new JamesBondCar("Fred", 50, false, true);
    myAuto.TurnOnRadio(true);
    myAuto.GoUnderWater();

    // Создаем поток для записи в файл
    FileStream myStream = File.Create("CarData.dat");

    // Помещаем объектный граф в поток в двоичном формате
    BinaryFormatter myBinaryFormat = new BinaryFormatter();
    myBinaryFormat.Serialize(myStream, myAuto);
    myStream.Close();
    ...
}
```

Полученный двоичный файл



Десериализация в двоичном формате

```
using System.Runtime.Serialization.Formatters.Binary;

public static void Main()
{

    myStream = FileOpenRead("CarData.dat");

    JamesBondCar carFromDisk =
        (JamesBondCar) myBinaryFormat.Deserialize(myStream);

    Console.WriteLine(CarFromDisk.PetName);
    CarFromDisk.TurnOnRadio(true);
    myStream.Close();

}
```

Сериализация в формате SOAP

```
using System.Runtime.Serialization.Formatters.Soap;

// Сохраняем тот же самый объект в формате SOAP
FileStream myStream = File.Create("CarData.xml");
SoapFormatter myXMLFormat = new SoapFormatter();
myXMLFormat.Serialize(myStream, myAuto);
myStream.Close();

// Восстанавливаем объект из файла SOAP
myStream = File.OpenRead("CarData.xml");
JamesBondCar carFromXML =
    (JamesBondCar)myXMLFormat.Deserialize(myStream);

Console.WriteLine(carFromXML.PetName + " is alive!");
myStream.Close();
```

Результирующий XML-файл

```
<SOAP-ENV:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"
    xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
    xmlns:clr="http://schemas.microsoft.com/soap/encoding/clr/1.0"
    SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
<SOAP-ENV:Body>
<a1:JamesBondCar id="ref-1"
    xmlns:a1="http://schemas.microsoft.com/clr/nsassem/CarToFileApp/CarToFileApp%2C%20Version
    %3D1.0.3700.24437%2C%20Culture%3Dneutral%2C%20PublicKeyToken%3Dnull">
<isFlightWorthy>false</isFlightWorthy>
<isSeaWorthy>true</isSeaWorthy>
<petName id="ref-3">Fred</petName>
<maxSpeed>50</maxSpeed>
<theRadio href="#ref-4"/>
<Car_x002B_petName href="#ref-3"/>
<Car_x002B_maxSpeed>50</Car_x002B_maxSpeed>
<Car_x002B_theRadio href="#ref-4"/>
</a1:JamesBondCar>
<a1:Radio id="ref-4"
    xmlns:a1="http://schemas.microsoft.com/clr/nsassem/CarToFileApp/CarToFileApp%2C%20Version
    %3D1.0.3700.24437%2C%20Culture%3Dneutral%2C%20PublicKeyToken%3Dnull">
</a1:Radio>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Пользовательская сериализация

```
// Интерфейс ISerializable

public interface ISerializable
{
 public virtual void GetObjectData
     (Serialization info, SreamingContext context);
}

// Специальный конструктор для десериализации
Class SoeClass
{
 private SomeClass (Serialization si, SreamingContext ctx) { … }
}
```

Класс SerializationInfo

- AddValue() – многократно перегружен
- GetXXXX() – GetString, GetInt32 …

Простой пример пользовательской сериализации

```
public class CustomCarType : ISerializable
{       public string petName;
        public int maxSpeed;
        public CustomCarType(string s, int i) { petName = s; maxSpeed = i; }

        // Передаем информацию о состоянии объекта объекту Formatter
        public void GetObjectData(SerializationInfo si, StreamingContext ctx)
        {
        // Каков тип нашего потока?
        Console.WriteLine("[GetObjectData] Context State: {0}", ctx.State.Format());

                si.AddValue("CapPetName", petName);
                si.AddValue("MaxSpeed", maxSpeed);
        }
        // А теперь позаботимся о специальном варианте конструктора
        private CustomCarType(SerializationInfo si, StreamingContext ctx)
        {
                // Каков тип нашего потока?
                Console.WriteLine("[ctor] Context State: {0}", ctx.State.Format());

                petName = si.GetString("CapPetName");
                maxSpeed = si.GetInt32("maxSpeed");
        }
}
```

Простой пример пользовательской сериализации

```
public static int Main(string[] args)
{
    CustomCarType myAuto = new CustomCarType("Siddhartha", 50);
    Stream myStream = File.Create("CarData.dat");

    // Задействуем интерфейс ISerializable
    BinaryFormatter myBinaryFormat = new BinaryFormatter();
    myBinaryFormat.Serialize(myStream, myAuto);
    myStream.Close();

    myStream = File.OpenRead("CarData.dat");

    // Вызываем спецконструктор
    CustomCarType carFromDisk =
        (CustomCarType)myBinaryFormat.Deserialize(myStream);

    Console.WriteLine(carFromDisk.petName + " is alive!");
    return 0;
}
```

Использование диалога Open

```
protected void menuItemOpen_Click (object sender, System.EventArgs e)
{
    // Настраиваем свойства диалогового окна для открытия файлов
    OpenFileDialog myOpenFileDialog = new OpenFileDialog();
    myOpenFileDialog.InitialDirectory = ".";
    myOpenFileDialog.Filter = "car files (*.car)|*.car|All files(*.*)|*.*";
    myOpenFileDialog.FilterIndex = 1;
    myOpenFileDialog.RestoreDirectory = true;

    // Восстанавливаем объекты автомобилей
    if(myOpenFileDialog.ShowDialog() = = DialogResult.OK)
    { // Очищаем текущий массив
      arTheCars.Clear();

      Stream myStram = null;
      if((myStream = myOpenFileDiaog.OpenFile()) != null)
      {
            BinaryFormatter myBinaryFormat = new BinaryFormatter();
            arTheCars = (ArrayList)myBinaryFormat.Deserilize(myStream);
            myStream.Close();
            UpdateGrid();
      }
    }
}
```

Использование диалога Save As

```
protected void menuItemSave_Click (object sender, System.EventArgs e)
{
    // Настраиваем свойства диалогового окна для сохранения файлов
    SaveFileDialog mySaveFileDialog = new SaveFileDialog();
    mySaveFileDialog.InitialDirectory = ".";
    mySaveFileDialog.Filter = "car files (*.car)|*.car|All files(*.*)|*.*";
    mySaveFileDialog.FilterIndex = 1;
    mySaveFileDialog.RestoreDirectory = true;
    mySaveFileDialog..FileName = "carDoc";

    // Сохраняем объекты автомобилей
    if(mySaveFileDialog.ShowDialog() = = DialogResult.OK)
    {
      Stream myStram = null;
      if((myStream = mySaveFileDiaog.OpenFile()) != null)
      {
            BinaryFormatter myBinaryFormat = new BinaryFormatter();
            myBinaryFormat.Serialize(myStream, arTheCars);
            myStream.Close();
      }
    }
}
```