Потоци и файлове

Лекция 6

Потоци

- **Потокът** (stream) представлява строго **подредена последователност** от цифрово кодирани сигнали, използвани за **предаване или приемане на данни** от и на различни устройства
- основно **средство за обмяна на информация** в компютърния свят
- **Потоците не** предоставят произволен достъп до данните си
- **Потоците** предоставят **само последователен достъп**, т.е може да манипулираме данните само в реда, в който те пристигат от потока

Потоци

- **Потоците (streams)** са важна част от всяка входно-изходна библиотека
- Те намират своето приложение, когато програмата трябва да "**прочете**" или "**запише**" данни от или във външен източник на данни – **файл**, други **компютри, сървъри** и т.н.
- терминът **вход (input) се асоциира с четенето** на информация, а терминът **изход (output)** – със **записването** на информация
- Потоците се използват за **четене и запис** на данни от и на **различни устройства**
- Те улесняват комуникацията между програма и файл, програма и отдалечен компютър и т.н.

ПОТОЦИ

- **Потокът** е **наредена последователност от байтове**, които се изпращат от едно приложение или входно устройство и се получават в друго приложение или изходно устройство
- Тези байтове се изпращат и получават един след друг и **винаги** пристигат в **същия ред, в който са били изпратени**
- **Потоците** са абстракция на комуникационен канал за данни, който свързва две устройства или програми
- **Потоците** позволяват **последователен** достъп до данните си

ВИДОВЕ ПОТОЦИ

В различните ситуации се използват **различни видове потоци**

- за работа с **текстови файлове**
- за работа с **бинарни (двоични) файлове**
- за работа със **символни низове**
- потоци, които се използват при **мрежова комуникация**

Следователно можем да разглеждаме потоците като **транспортен канал за данни**

Потоци

- Потоците са основното средство за обмяна на информация в компютърния свят
  - достъп до файлове на компютъра
  - мрежова комуникация между отдалечени компютри
- Например:
  - **печатането на принтер** е изпращане на поредица байтове към поток, свързан със съответния порт, към който е свързан принтера
  - **възпроизвеждане на звук от звуковата карта** - представлява поредица от байтове
  - **сканиране на документи** - на скенера се изпращат команди (чрез изходен поток) и след това се прочита сканираното изображение (като входен поток)

ТЕКСТОВ ПОТОК

- За да прочетем или запишем нещо от или във файл, трябва да **отворим поток** към дадения файл, да извършим **четене** или **запис** и да затворим потока
- Потоците могат да бъдат **текстови** или **бинарни**

Това разделение е свързано с **интерпретацията** на изпращаните и получаваните байтове

- **Текстов поток** - серия байтове се разглеждат като **текст** (в предварително зададено кодиране)

Модерните сайтове в Интернет не могат без потоци и така нареченият

- **streaming** (произлиза от stream) – представлява **поточно достъпване на обемни мултимедийни файлове**, идващи от Интернет

ОСНОВНИ ОПЕРАЦИИ С ПОТОЦИ

- **Създаване**

Свързваме потока с източник на данни, механизъм за пренос на данни или друг поток

Например, когато имаме **файлов поток**, тогава задаваме **името на файла** и **режима, в който го отваряме** (за четене, за писане или за четене и писане едновременно)

- **Четене**

Извличаме данни от потока. **Четенето** винаги се извършва **последователно** от текущата позиция на потока

Четенето е **блокираща операция** и ако отсрещната страна не е изпратила данни, докато се опитваме да четем или изпратените данни още не са пристигнали, може да се случи изключение

ОСНОВНИ ОПЕРАЦИИ С ПОТОЦИ

Запис

- **Изпращаме данни** в потока по специфичен начин
- Записът се извършва от **текущата позиция** на потока
- Записът потенциално може да е блокираща операция и да се забави, докато данните поемат по своя път

  Например ако изпращаме обемни данни по мрежов поток, операцията може да се забави, докато данните отпътуват по мрежата

ОСНОВНИ ОПЕРАЦИИ С ПОТОЦИ

Позициониране - преместване на текущата позиция на потока.

- Преместването се извършва спрямо **текуща позиция**, като можем да позиционираме **спрямо текуща позиция**, спрямо **началото на потока**, или спрямо **края на потока**
- Преместване можем да извършваме единствено в потоци, които поддържат позициониране
  - Например **файловите потоци** обикновено поддържат позициониране, докато **мрежовите не поддържат**

Затваряне

- **Приключване работата с потока** и **освобождаване на ресурсите** заети от него
- Ресурс, отворен от един потребител, обикновено не може да се ползва от останалите потребители

(в това число от други програми на същия компютър, които се и зпълняват паралелно на нашата програма)

Затваряне на потоци

- Затварянето трябва да се извършва **възможно най-рано** след **приключване на работата** с потока

```
using System.IO;

FileStream fileStream = new FileStream(@"c:\file.txt", FileMode.Open);
try
{
        // четене или писане от / във файл
}
finally
{       // затваряне на потока с метода Stream.Close()
        fileStream.Close();
}
```

Използването на **using** гарантира, че след излизането от тялото, автоматично ще се извика метода **Close().**

```
using (<stream object>) {...}
```

Потоци в .NET Framework
ОСНОВНИ КЛАСОВЕ

Намират се в пространството от имена **System.IO**

Основните класове са:

- **Stream** – **базов абстрактен клас** за всички потоци
- **BufferedStream FileStream**, **MemoryStream**, **GZipStream, NetworkStream**
- Всички потоци в C# задължително трябва да се затворят след като сме приключили работа с тях
- Оставянето на отворен поток или файл води до загуба на ресурси и може да блокира работата на други потребители или процеси във вашата система

ДВОИЧНИ ПОТОЦИ

- **Двоични потоци** - работят с двоични (бинарни) данни
- това ги прави универсални - може да се използват за четене на информация от всякакви файлове (картинки, музикални и мултимедийни файлове, текстови файлове и т.н.)
- **Основните класове** за четене и запис от и към двоични потоци са:
  - **FileStream**
  - **BinaryReader**
  - **BinaryWriter**

ДВОИЧНИ ПОТОЦИ

- **FileStream** - предлага **методи за четене и запис от бинарен файл**, пропускане на определен брой байтове, проверяване на броя достъпни байтове и метод за затваряне на потока
- **BinaryWriter** - позволява **записването в поток** от примитивни типове във вид на двоични стойности в специфично кодиране

  Той има един основен метод – **Write(…),** който позволява записване на всякакви примитивни типове данни – числа, символи, булеви стойности, масиви, стрингове и др.
- **BinaryReader** - позволява **четенето на данни** от примитивни типове записани с помощта на **BinaryWriter**
- Основните му методи позволяват да **четем символ, масив от символи, цели числа, числа с плаваща запетая** и др.

Класът FileStream Class

- Класът **FileStream -** пространството **System.IO** осъществява:
  - **четене от файл**
  - **запис във файл**
  - **затваряне на файлове**
- Този клас произлиза (наследява) от **абстрактния клас Stream**
- Първо трябва да се създаде обект – инстанция на класа **FileStream,** за да се **създаде нов файл или да бъде отворен съществуващ файл**

Класът FileStream Class

- Синтаксисът за **създаване на обект** от клас **FileStream** е следния:

```
FileStream <object_name> = new FileStream( <file_name>,
                <FileMode Enumerator>, <FileAccess Enumerator>,
                <FileShare Enumerator>);
```

```
FileStream  F = new FileStream("sample.txt", FileMode.Open, FileAccess.Read,
                                                FileShare.Read);
```

```
FileStream F = new FileStream("test.txt",   FileMode.OpenOrCreate,
                                                FileAccess.ReadWrite);
```

Класът FileStream - параметри

Параметър	Описание
FileMode	**Append:** It opens an existing file and puts cursor at the end of file, or creates the file, if the file does not exist. **Create:** It creates a new file. **CreateNew:** It specifies to the operating system, that it should create a new file. **Open: I**t opens an existing file. **OpenOrCreate:** It specifies to the operating system that it should open a file if it exists, otherwise it should create a new file. **Truncate**: It opens an existing file and truncates its size to zero bytes.
FileAccess	**Read** **ReadWrite** **Write**
FileShare	**Inheritable:** It allows a file handle to pass inheritance to the child processes **None:** It declines sharing of the current file **Read:** It allows opening the file for reading **ReadWrite:** It allows opening the file for reading and writing **Write:** It allows opening the file for writing

Пример: запис и четене в двоичен файл

```
namespace Lekcia6_FileStream_1
{
    class Program
    {
        static void Main(string[] args)
        {
            FileStream F = new FileStream("testbyte.txt",
                FileMode.OpenOrCreate, FileAccess.ReadWrite);
            for (int i = 1; i <= 20; i++)
            {
                F.WriteByte((byte)i);
            }
            F.Position = 0;
            for (int i = 0; i < 20; i++)
            {
                Console.Write(F.ReadByte() + " ");
            }
            F.Close();
            Console.ReadKey();

        }
    }
}
```

Отваряне на двоичен файл test.dat с изключения:

```
try
{
    FileStream fin = new FileStream("test", FileMode.Open);
}
catch(IOException exc)
{
    Console.WriteLine(exc.Message);
// Handle the error.
}
catch(Exception exc)
{ // catch any other exception.
        Console.WriteLine(exc.Message);
// Handle the error.
}
```

ТЕКСТОВИ ПОТОЦИ

- **Текстовите потоци** - работят само с текстови данни или с поредици от символи (**char**) и стрингове (**string**)
- Подходящи са за работа с текстови файлове
- Но това ги прави неизползваеми при работа с каквито и да е бинарни файлове
- Основните класове за работа с текстови потоци са **TextReader** и **TextWriter**
- Те са **абстрактни класове** и от тях не могат да бъдат създавани обекти
- Тези класове дефинират базова функционалност за четене и запис на класовете, които ги наследяват

ВАЖНИ МЕТОДИ НА ТЕКСТОВИТЕ ПОТОЦИ

- **ReadLine()** – чете един текстов ред и връща символен низ
- **ReadToEnd()** – чете всичко от потока до неговия край и връща символен низ
- **Write()** – записва символен низ в потока
- **WriteLine()** – записва един текстов ред в потока

Символите в .NET са **Unicode** символи, но **потоците** могат да работят освен с Unicode и с други кодирания, например стандартното за кирилицата **кодиране Windows-1251**

ВРЪЗКА МЕЖДУ ТЕКСТОВИ И БИНАРНИ ПОТОЦИ

- При писане на текст класът **StreamWriter** скрито от нас **превръща текста в байтове** преди да го запише на текущата позиция във файла
- **StreamWriter** използва кодирането, което му е зададено по време на създаването му
- По подобен начин работи и **StreamReader** класът
- Той вътрешно използва **StringBuilder** и когато чете бинарни данни от файла, ги конвертира към текст преди да ги върне като резултат от четенето

ВРЪЗКА МЕЖДУ ТЕКСТОВИ И БИНАРНИ ПОТОЦИ

- В операционната система няма понятие "**текстов файл**“
- **Файлът** винаги е поредица от байтове, а дали е текстов или бинарен зависи от интерпретацията на тези байтове
- Ако искаме да **разглеждаме** даден **файл** или **поток като текстов**, трябва да го четем и пишем с текстови потоци:

  StreamReader или **StreamWriter**
- Ако искаме да го разглеждаме като бинарен (двоичен), трябва да го четем и пишем с бинарен поток - **FileStream**

ЧЕТЕНЕ ОТ ТЕКСТОВ ФАЙЛ

- **Текстовите потоци** работят с **текстови редове**, т.е. интерпретират бинарните данни като поредица от редове, разделени един от друг със символ за нов ред
- **Символът за нов ред** не е един и същ за различните платформи и операционни системи
  - За UNIX и Linux той е **LF (0x0A)**
  - **за Windows** и DOS той е **CR + LF (0x0D + 0x0A)**
- Четенето на един текстов ред от даден файл или поток означава на практика **четене на поредица от байтове до прочитане на един от символите CR** или **LF** и преобразуване на тези байтове до текст, спрямо използваното в потока кодиране (encoding)

ЧЕТЕНЕ ОТ ТЕКСТОВ ФАЙЛ

Класът StreamReader - предоставя най-лесният начин за четене на текстов файл, наподобява четенето от конзолата

- **StreamReader** не е поток, но може да работи с потоци
- **Отваряне на текстов файл за четене**
  - Може да създадем **StreamReader** просто по име на файл (или пълен път до файла)
  - **За да четем от текстов файл**, трябва да създадем променлива от тип **StreamReader**, която да свържем с конкретен файл от файловата система на нашия компютър

```
// Create a StreamReader connected to a file
StreamReader reader = new StreamReader("test.txt");
```

  - ако файлът се намира в папката, където е компилиран проекта (поддиректория **bin\Debug**), то можем да подадем само **конкретното му име**
  - в противен случай може да подадем **пълния път до файла** или да използваме релативен път

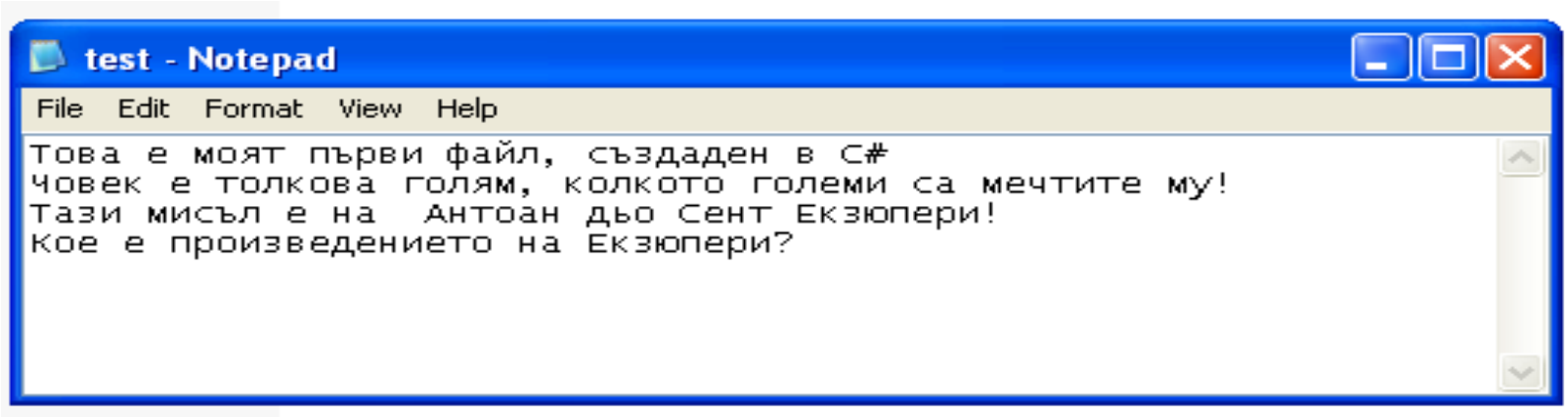
Пример – създаване на текстов файл и откриване на поток към него StreamWriter

```
using System.IO;

namespace Lectia6_TextFile_Zad1
{
    class Program
    {
        static void Main(string[] args)
        {
            //създаване на файл и откриване на поток към него
            StreamWriter sw = File.CreateText("test.txt");
            sw.WriteLine("Това е моят първи файл, създаден в C#");
            sw.WriteLine("Много по-трудно е да съдиш сам себе си, отколкото да
                  съдиш другите. Ако можеш да съдиш себе си правилно, значи ти си
                  истински мъдрец.");
            sw.Close();
           Console.ReadKey();
        }
    }
}
```

Добавяне на текст към вече създаден файл метод AppendText

```
//Дописване текст към вече създаден файл метод AppendText
      StreamWriter sw = File.AppendText("test.txt");
      sw.WriteLine("Тази мисъл е на  Антоан дьо Сент Екзюпери!");
      sw.WriteLine("Кое е произведението на Екзюпери?");
      sw.Close();
```

- Файлът се намира в следната папка:
- Lectia6_TextFile_Zad1\bin\Debug\test.txt



Четене от текстов файл StreamReader

- Първо трябва да се отвори **входен поток за четене от класа StreamReader**
- **Използва се метод**

```
File.OpenText("име на файла");
```

```
StreamReader sr = File.OpenText("test.txt");
```

- **Ако искаме да прочетем само един ред**

```
sr.ReadLine()
```

- **Ако искаме да изведем първият ред на конзолата**

```
System.Console.WriteLine(sr.ReadLine());
```

Четене от текстов файл
Lectia6_TextFile_Zad2

- Ако искаме да прочетем **всички редове от файла**

1) **Sr.ReadToEnd()**

```
System.Console.WriteLine("\n Извеждане на целия файл на екрана!\n");
System.Console.WriteLine(sr.ReadToEnd());
```

2)
```
//четене на всички редове последователно един след друг
bool flag = true;
while (flag)
{
    string strline = sr.ReadLine();
    if (strline == null)
        flag = false;
   System.Console.WriteLine(strline);
}
    sr.Close();
```

ПЪЛНИ И РЕЛАТИВНИ ПЪТИЩА

- При работата с файлове може да се използват **пълни пътища** (например **C:\Temp\example.txt**) или **релативни пътища**, спрямо директорията, от която е стартирано приложението (примерно **..\..\example.txt**)
- Ако се използват пълни пътища, при подаване на пълния път до даден файл не забравяйте да направите **escaping** на наклонените черти, които се използват за разделяне на папките
- В C# това можете да направите по **два начина:**
  - с **двойна наклонена черта**
  - с **цитирани низове**, започващи с @ преди стринговия литерал
- `string fileName = "C:\\Temp\\work\\test.txt";`
- `string theSamefileName = @"C:\Temp\work\test.txt";`

ПЪЛНИ И РЕЛАТИВНИ ПЪТИЩА

- Избягвайте пълни пътища и **работете с относителни**!
- Това прави приложението ви преносимо и по-лесно за инсталация и поддръжка
- Използването на пълен път до даден файл (примерно **C:\Temp\test.txt**) е лоша практика, защото прави програмата ви зависима от средата и непреносима
- Ако я прехвърлите на друг компютър, ще трябва да коригирате пътищата до файловете, които тя търси, за да работи коректно
- Ако използвате **относителен (релативен)** път спрямо текущата директория (например **..\..\example.txt**), вашата програма ще е **лесно преносима**

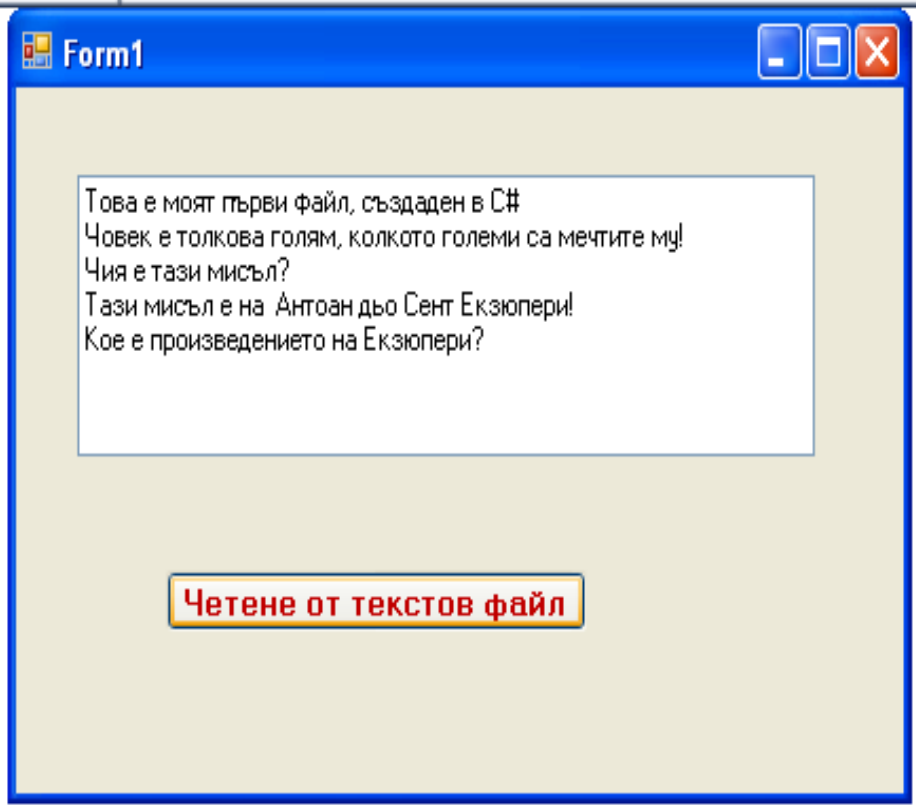
ПЪЛНИ И РЕЛАТИВНИ ПЪТИЩА

- При стартиране на C# програма **текущата** директория е **тази,** в която се намира **изпълнимият (.exe) файл**
- Най-често това е поддиректорията **bin\Debug** спрямо коренната директория на проекта
- Следователно, за да отворите файла **example.txt** от коренната директория на вашия Visual Studio проект, трябва да използвате **релативния път ..\..\example.txt**

Пример – Четене на файл и записване в текстова кутия

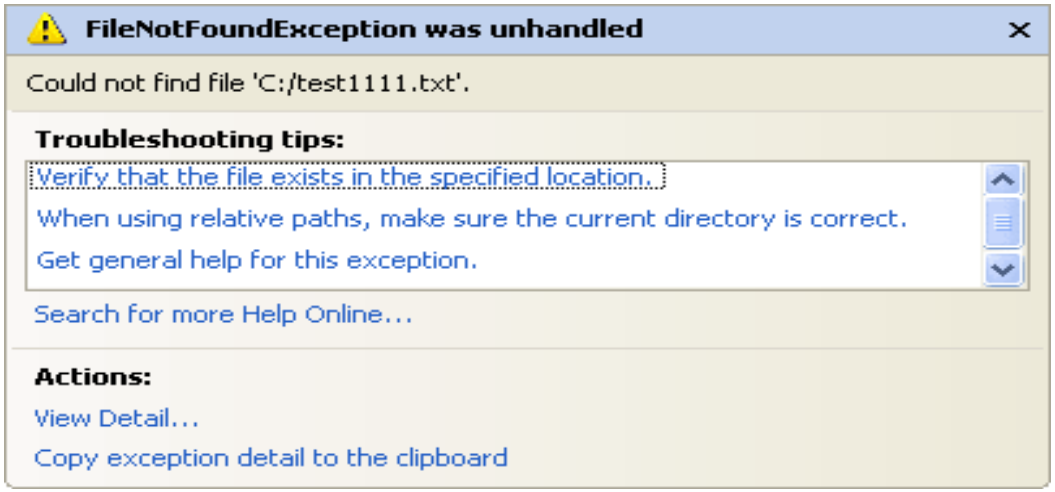
Lectia6_TextFile_Zad3.Form1 | button1_Click(object sender, EventArgs e)

```
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
using System.IO;

namespace Lectia6_TextFile_Zad3
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }

        private void button1_Click(object sender, EventArgs e)
        {
            string file_name = "test.txt";

          // file_name = Environment.GetFolderPath(Environment.SpecialFolder.MyDocuments) + file_name;
            StreamReader objReader = new StreamReader(file_name);
            textBoxOpen.Text = objReader.ReadToEnd();
            objReader.Close();
        }
    }
}
```



Обработка на изключения

- Ако не сте задали правилно името на файла или не е намерен, възниква изключение




```
if (File.Exists(file_name) == true)
        {
           StreamReader objReader = new StreamReader(file_name);
           textBoxOpen.Text = objReader.ReadToEnd();
           objReader.Close();
        }
        else
           MessageBox.Show("Няма такъв файл " + file_name);
```

Обработка на изключения

```
if ( System.IO.Directory.Exists( folder_location ) )
{
}
```

```
if (File.Exists("test.txt") == true)
{
      StreamReader textReader = new StreamReader("test.txt");
          ....
}
else
     MessageBox.Show("Няма такъв файл " + fileName, "Съобщение за
липсващ файл!",MessageBoxButtons.OK, MessageBoxIcon.Warning);
          }
```

Кодиране на файловете (encoding)

- В паметта на компютрите всичко се запазва в **двоичен вид**
- Това означава, че и **текстовите файлове** се представят **цифрово**, за да могат да бъдат съхранени в паметта, както и на твърдия диск
- Този процес се нарича **кодиране на файловете**
- **Кодирането** се състои в **заместването на текстовите символи** (цифри, букви, препинателни знаци и т.н.) **с точно определени поредици от числови стойности**

ASCII таблица - срещу **всеки символ** стои **определена стойност** (пореден номер)

Dec	Hx	Oct	Char	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr
0	0	000	NUL (null)	32	20	040		Space	64	40	100	@	@	96	60	140	`	`
1	1	001	SOH (start of heading)	33	21	041	!	!	65	41	101	A	A	97	61	141	a	a
2	2	002	STX (start of text)	34	22	042	"	"	66	42	102	B	B	98	62	142	b	b
3	3	003	ETX (end of text)	35	23	043	#	#	67	43	103	C	C	99	63	143	c	c
4	4	004	EOT (end of transmission)	36	24	044	$	$	68	44	104	D	D	100	64	144	d	d
5	5	005	ENQ (enquiry)	37	25	045	%	%	69	45	105	E	E	101	65	145	e	e
6	6	006	ACK (acknowledge)	38	26	046	&	&	70	46	106	F	F	102	66	146	f	f
7	7	007	BEL (bell)	39	27	047	'	'	71	47	107	G	G	103	67	147	g	g
8	8	010	BS (backspace)	40	28	050	(	(	72	48	110	H	H	104	68	150	h	h
9	9	011	TAB (horizontal tab)	41	29	051	)	)	73	49	111	I	I	105	69	151	i	i
10	A	012	LF (NL line feed, new line)	42	2A	052	*	*	74	4A	112	J	J	106	6A	152	j	j
11	B	013	VT (vertical tab)	43	2B	053	+	+	75	4B	113	K	K	107	6B	153	k	k
12	C	014	FF (NP form feed, new page)	44	2C	054	,	,	76	4C	114	L	L	108	6C	154	l	l
13	D	015	CR (carriage return)	45	2D	055	-	-	77	4D	115	M	M	109	6D	155	m	m
14	E	016	SO (shift out)	46	2E	056	.	.	78	4E	116	N	N	110	6E	156	n	n
15	F	017	SI (shift in)	47	2F	057	/	/	79	4F	117	O	O	111	6F	157	o	o
16	10	020	DLE (data link escape)	48	30	060	0	0	80	50	120	P	P	112	70	160	p	p
17	11	021	DC1 (device control 1)	49	31	061	1	1	81	51	121	Q	Q	113	71	161	q	q
18	12	022	DC2 (device control 2)	50	32	062	2	2	82	52	122	R	R	114	72	162	r	r
19	13	023	DC3 (device control 3)	51	33	063	3	3	83	53	123	S	S	115	73	163	s	s
20	14	024	DC4 (device control 4)	52	34	064	4	4	84	54	124	T	T	116	74	164	t	t
21	15	025	NAK (negative acknowledge)	53	35	065	5	5	85	55	125	U	U	117	75	165	u	u
22	16	026	SYN (synchronous idle)	54	36	066	6	6	86	56	126	V	V	118	76	166	v	v
23	17	027	ETB (end of trans. block)	55	37	067	7	7	87	57	127	W	W	119	77	167	w	w
24	18	030	CAN (cancel)	56	38	070	8	8	88	58	130	X	X	120	78	170	x	x
25	19	031	EM (end of medium)	57	39	071	9	9	89	59	131	Y	Y	121	79	171	y	y
26	1A	032	SUB (substitute)	58	3A	072	:	:	90	5A	132	Z	Z	122	7A	172	z	z
27	1B	033	ESC (escape)	59	3B	073	;	;	91	5B	133	[	[	123	7B	173	{	{
28	1C	034	FS (file separator)	60	3C	074	<	<	92	5C	134	\	\	124	7C	174			|
29	1D	035	GS (group separator)	61	3D	075	=	=	93	5D	135	]	]	125	7D	175	}	}
30	1E	036	RS (record separator)	62	3E	076	>	>	94	5E	136	^	^	126	7E	176	~	~
31	1F	037	US (unit separator)	63	3F	077	?	?	95	5F	137	_	_	127	7F	177	&#127;	DEL

Кодиране на файловете (encoding)

- **Кодиращите схеми** (character encodings) задават правила за преобразуване на текст в поредица от байтове и обратно
- Най-често при работа с **кирилица** се използват **UTF-8** и **Windows-1251**
- **UTF-8** е кодираща схема, при която най-често използваните символи (латинската азбука, цифри и някои специални знаци) се кодират в **един байт**
- по-рядко използваните **Unicode** символи (като кирилица, гръцки и арабски) се кодират в **два байта**,
- всички останали символи (китайски, японски и много други) се кодират в **3** или **4 байта**

Кодиране на файловете (encoding)

- Кодирането UTF-8 може да преобразува произволен Unicode текст в бинарен вид и обратно
- **Unicode** представлява индустриален стандарт, който позволява на компютри и други електронни устройства винаги да представят и манипулират по един и същи начин текст, написан на повечето от световните писмености
- Поддържа всички над **100 000 символа** от Unicode стандарта
- Кодирането **UTF-8 е универсално** и е подходящо **за всякакви езици**, азбуки и писмености

```
StreamReader reader = new StreamReader("test.txt",
      Encoding.GetEncoding("UTF-8"));
```

Кодиране на файловете (encoding)

Windows-1251 - обикновено се кодират текстове на кирилица (например съобщения изпратени по e-mail)

- съдържа **256 символа**, включващи **латинска азбука, кирилица** и някои **често използвани знаци**
- използва по **един байт за всеки символ**, но за сметка на това някои символи не могат да бъдат записани в него (например символите от китайската азбука) и се губят при опит да се направи това
- Това кодиране се използва **по подразбиране в Windows**, който е настроен за работа с български език

Кодиране на файловете (encoding)

- Други примери за кодиращи схеми (encodings или charsets) са **ISO 8859-1**, **Windows-1252**, **UTF-16**, **KOI8-R** и т.н.
- Те се ползват в специфични региони по света и дефинират свои набори от символи и правила за преминаване от текст към бинарни данни и на обратно
- За представяне на кодиращите схеми в .NET Framework се използва класът **System.Text.Encoding**, който се създава по следния начин:

```
Encoding win1251 = Encoding.GetEncoding("Windows-1251");
```

Отваряне на файл със задаване на кодиране

- Четенето и писането от и към текстови потоци изисква да се използва определено, **предварително зададено кодиране** на символите (**character encoding**)
- Кодирането може да се зададе при създаването на **StreamReader** обект като допълнителен втори параметър:

```
// Create a StreamReader connected to a file
StreamReader reader = new StreamReader("test.txt",
        Encoding.GetEncoding("Windows-1251"));
// Read file here...

// Close the reader resource after you've finished using it
reader.Close();
```

- Ако **не бъде** зададено специфично кодиране при отварянето на файла, се използва **стандартното кодиране UTF-8**
- В показания по-горе случай използваме кодиране **Windows-1251** - 8-битов (еднобайтов) набор символи, проектиран от Майкрософт за езиците, използващи кирилица като български, руски и други

Четене на кирилица

- Ако объркаме кодирането при четене или писане във файл са възможни няколко сценария:
  - Ако ползваме само **латиница,** всичко ще работи **нормално**
  - Ако използваме **кирилица** и четем с **грешен encoding**, ще прочетем **безсмислени символи**, които не могат да се прочетат
  - Ако записваме кирилица в **кодиране,** което **не поддържа кирилската азбука** (например **ASCII**), **буквите** от **кирилицата ще бъдат заменени** безвъзвратно със символа **"?"** (въпросителна)

Четене на текстов файл ред по ред – пример

- Препоръчително е да създавате текстовия файл в **Debug** папката на проекта (**.\bin\Debug**)
- Така той ще е в същата директория, в която е вашето компилирано приложение и няма да се налага да подавате пълния път до него при отварянето на файла
- Създаваме следният файл:

sample.txt

```
Ако на куп пред себе си заложиш
спечеленото, смело хвърлиш зар,
изгубиш, и започнеш пак, и можеш
да премълчиш за неуспеха стар;

Ако заставиш мозък, нерви, длани
и изхабени – да ти служат пак,
и крачиш, само с Волята останал,
която им повтаря: „Влезте в крак!
```

Lekcia6_TextFile_ReadLine.cs

```
class Lekcia6_TextFile_ReadLine

{
   static void Main()
  {
        // Create an instance of StreamReader to read from a file
        StreamReader reader = new StreamReader("Sample.txt");
        int lineNumber = 0;
        // Read first line from the text file
        string line = reader.ReadLine();
        // Read the other lines from the text file
        while (line != null)
        {     lineNumber++;
              Console.WriteLine("Line {0}: {1}", lineNumber, line);
              line = reader.ReadLine();
        }
              // Close the resource after you've finished using it
              reader.Close();
        }
}
```

Име на файл

Брои и показва текущият ред

Съхранява всеки прочетен ред

За да избегнем загубата на ресурси

Автоматично затваряне на потока след приключване на работа с него

- C# предлага конструкция за автоматично затваряне на потока или файла след приключване на работата с него
- Тази конструкция е **using**
- Синтаксисът й е следният:

```
using(<stream object>) { … }
```

- Използването на **using** гарантира, че след излизане от тялото й автоматично ще се извика метода **Close()**
- Това ще се случи дори, ако при четенето на файла възникне някакво изключение

Lekcia6_TextFile_ReadLine.cs

```
class Program
{
    static void Main()
    {
        // Create an instance of StreamReader to read from a file
        StreamReader reader = new StreamReader("Sample.txt");
        using (reader)
        {
             int lineNumber = 0;
             // Read first line from the text file
             string line = reader.ReadLine();
             // Read the other lines from the text file
            while (line != null)
            {
                lineNumber++;
                Console.WriteLine("Line {0}: {1}", lineNumber, line);
                line = reader.ReadLine();
          // Close the resource after you've finished using it
               Console.ReadKey();

            }
        }
    }
}
```

Винаги използвайте **using конструкцията** в C# за да затваряте коректно отворените потоци и файлове!

Пример – записване и четене на данни в/от текстов файл с using

Lect6_TextFile_Using_Console

```
string[] names = new string[] {"Anna", "Nina", "Petar", "Ivan"};
        using (StreamWriter sw = new StreamWriter("names.txt"))
        {
           foreach (string s in names)
           {
           sw.WriteLine(s);
           }
        }
        // Read and show each line from the file.
        string line = "";
        using (StreamReader sr = new StreamReader("names.txt"))
        {
           while ((line = sr.ReadLine()) != null)
              {
              Console.WriteLine(line);
              }
        }
        Console.ReadKey();
```

Писане в текстов файл

- Писането в текстови файлове е много удобен способ за съхранение на различни видове информация
  - Например - записване на резултатите от изпълнението на дадена програма
  - Записване на информацията от текстово поле във файл
  - ...
- Както при четенето на текстов файл, и при писането, се използва един подобен на конзолата клас, който се нарича **StreamWriter**

Класът StreamWriter

- Класът **StreamWriter** е част от пространството от имена **System.IO** и се използва изключително и само за работа с текстови данни
- Прилича на класа **StreamReader**, но предлага методи за **записване на текст във файл**
- За разлика от другите потоци, **преди да запише данните на желаното място**, той ги **превръща в байтове**
- **StreamWriter** ни дава възможност при създаването си да определим желания от нас encoding
- Можем да създадем инстанция на класа по следния начин:

```
StreamWriter writer = new StreamWriter("test.txt");
```

Класът StreamWriter

- В конструктора на класа можем да подадем като параметър, както път до файл, така и вече създаден поток, в който ще записваме, а също и кодираща схема

```
StreamWriter writer = new StreamWriter("test.txt",
false, Encoding.GetEncoding("Windows-1251"));
```

  - В този пример подаваме **път до файл** като първи параметър
  - Като втори подаваме **булева променлива**, която указва дали **ако файлът вече съществува**, данните да бъдат **залепени на края** на файла или файлът да бъде презаписан
  - Като трети параметър подаваме **кодираща схема** (encoding).

Отпечатване на числата от 1 до 20 в текстов файл – пример

```
class FileWriter
{
        static void Main()
        {
            // Create a StreamWriter instance
            StreamWriter writer = new StreamWriter("numbers.txt");
            // Ensure the writer will be closed when no longer used
            using(writer)
            {
           // Loop through the numbers from 1 to 20 and write them
                for (int i = 1; i <= 20; i++)
                {
                        writer.WriteLine(i);
                }
             }
        }
}
```

Прихващане на изключения при работа с файлове

- **FileNotFoundException** (желаният файл не е намерен) - най-често срещаната грешка при работа с файлове. Тя може да възникне при създаването на **StreamReader**
- **ArgumentException** - задаваме определен encoding при създаване на **StreamReader** или **StreamWriter**, който не се поддържа
- **IOException** - това е базов клас за всички входно-изходни грешки при работа с потоци

Прихващане на изключения при работа с файлове

Стандартният подход при обработване на изключения при работа с файлове е следният:

- декларираме **променливите** от клас **StreamReader** или **StreamWriter** в **try-catch** блок
- в блока ги инициализираме с нужните ни стойности и прихващаме и обработваме потенциалните грешки по подходящ начин.
- за затваряне на потоците използваме конструкция **using**.

Прихващане на изключения

```
static void Main(string[] args)
{
    try
    {
      // Create an instance of StreamReader to read from a file.
      // The using statement also closes the StreamReader.
    using (StreamReader sr = new StreamReader("c:/TextFileExc.txt"))
    {
        string line;
        // Read and display lines from the file until
        // the end of the file is reached.
        while ((line = sr.ReadLine()) != null)
        {
            Console.WriteLine(line);
        }
     }
    }
    catch (Exception e)
    {
        // Let the user know what went wrong.
        Console.WriteLine("The file could not be read:");
        Console.WriteLine(e.Message);
    }
       Console.ReadKey();
)
```

Прихващане на грешка при отваряне на файл – пример Lectia6_FileExceptions

```
class HandlingExceptions
{
    static void Main()
    {
        string fileName = @"somedir/somefile.txt";
          try
          {
                StreamReader reader = new StreamReader(fileName);
                Console.WriteLine("File {0} successfully opened.", fileName);
                Console.WriteLine("File contents:");
                using (reader)
                {
                        Console.WriteLine(reader.ReadToEnd());
                }
           }
           catch (FileNotFoundException)
           {
                Console.Error.WriteLine("Can not find file {0}.", fileName);
           }
           catch (DirectoryNotFoundException)
           {
                Console.Error.WriteLine("Invalid directory in the file path.");
           }
           catch (IOException)
          {
               Console.Error.WriteLine("Can not open the file {0}", fileName);
          }
        }
}
```

```
static void Main(string[] args)
{
   string str;
   FileStream fout;
   try
   {
      fout = new FileStream("test.txt", FileMode.Create);
   }
   catch (IOException exc)
   {
      Console.WriteLine(exc.Message);
        return;
   }
   using (StreamWriter fstr_out = new StreamWriter(fout))
   {     //  Create a  StreamWriter
   Console.WriteLine("Enter text ('stop' to quit).");
   do
   {
       Console.Write(": ");
       str = Console.ReadLine();
       if (str != "stop")
       {
         str = str + "\r\n"; // add newline
                          try
                          {
                              fstr_out.Write(str);
                          }
                          catch (IOException exc)
                          {
                              Console.WriteLine(exc.Message);
                              break;
```

Благодаря за вниманието!