**Урок 17-21: Файлове в C# (преговор с допълнение)**

Едно от важните приложения на компютрите в ежедневието е за бързо обработване на огромно количество информация. Но данните за такава обработка трябва да дойдат отнякъде и да бъдат въведени в компютъра. Ако те са голямо количество, не е подходящо да се използва графично приложение, тъй като трябва да се създадат твърде голям брой текстови полета. От друга страна, ако се използва интерфейса на конзолно приложение, то въвеждането на данните може да се наложи да се направи многократно, тъй като има вероятност да се допусне грешка, която не може да бъде поправена по никакъв начин, освен чрез започване на въвеждането отново. И двата варианта, разбира се, са много неприятни и неприемливи. Затова в езиците за програмиране е предвидена възможността програмата както да чете входните си данни от файл, така и да извежда изходните си данни във файл. Това не е винаги необходимо, но си представете колко много данни има за учениците от едно училище и колко по-лесно би било, ако те са записани във файлове, които компютрите да обработват.

От изучаваните до този момент приложни програми знаем, че те работят с файлове и имат менюта за отваряне, съхраняване и затваряне на файл. В програми на езика C# можем да направим същото, като за целта файловете се разглеждат като **потоци**.

**1. Потоци от данни**

**а) същност** – подредени последователности от байтове, които се изпращат от едно устройство и се получават в друго устройство. Потоците позволяват само последователен достъп до данните.

**б) дефиниране**

Всички потоци в езика C# се намират в пространството от имена **System.IO**, което означава, че ако искаме да използваме файлове, в началото на програмите трябва да добавяме директивата: **using System.IO;** Езикът C# предлага различни потоци, подходящи за различни дейности (BufferedStream, MemoryStream, Stream, FileStream, NetworkStream и др.), но в повечето програми се използват текстови потоци, които позволяват да се четат и извеждат данни в текстови файлове.

**в) физически и логически файл**

* **физически файл** – реалният файл, разположен на носител на данни
* **логически файл** – образ на файла в програмата

**2. Компонен RichTextBox**

Този компонент на графичния интерфейс се нарича **многоредова текстова кутия.** Тя се различава от компонента **TextBox** по това, че в нея може да се извежда текст на повече от един ред. Указание за преминаване на нов ред е низът **„\n**“, поставен в текста, който искаме да изведем.

**3. Четене от текстов файл**

* чрез променлива от клас **StreamReader** „запознаваме“ компютъра с файла, от който ще се чете

*StreamReader r = new StreamReader(име на файл, Encoding.GetEncoding("windows-1251"));*

Ако файлът не се намира в текущата папка на изпълнимия файл на програмата, тогава трябва да се зададе и пътят до него, като е препоръчително този път да започва от папката на изпълнимия файл – това прави програмата по-независима при евентуалния ѝ пренос на друг компютър.

* четене на данните от файла – чрез променливата r, която играе ролята на логически файл и може да стане по два начина.
* При ***първия начин*** програмата прочита целия текстов файл чрез метода **ReadToEnd()** на класа StreamReader наведнъж като един низ и след това може да бъде обработван или показан в интерфейсния елемент RichTextBox:

*string textFile = r.ReadToEnd(); richTextBox1.Text = textFile;*

Този вариант се предпочита, когато файлът е под 1 MB и не се налага обработка ред по ред

* При ***втория вариант*** файлът се чете ред по ред чрез метода **ReadLine()** на класа StreamReader, който работи по същия начин като метода ReadLine() на класа Console. За целта трябва да се използва оператор за цикъл. Най-подходящ е оператор while, тъй като обикновено не се знае от колко реда е файлът. Когато се достигне края на файла се прочита неговия знак за край **null** и това може да се използва за условие за край на цикъла. Променливата, която ще се използва в цикъла трябва да бъде от тип низ, тъй като текстовият файл е последователност от редове от тип низ:

*string line = "";*

*while(line!=null)*

*{*

*line = r.ReadLine();*

*richTextBox1.Text = richTextBox1.Text+line + "\n";*

*}*

Този вариант се използва, когато искаме да обработваме входните данни ред по ред.

* затваряне на файла - след като приключим работата с файла трябва да го затворим чрез метода **Close()** на класа StreamReader:

*r.Close();*

Ако искаме да започнем да четем файла отначало, трябва да го отворим отново.

**4. Записване в текстов файл**

Понякога се налага изходните данни от една програма да бъдат записани във файл – за да бъдат разпечатани, препратени до друг компютър, да бъдат използвани като входни данни в друга програма и т.н. Записването в текстов файл също е последователно.

* свързваме променлива от клас **StreamWriter** с физическия файла, в който се записваме:

*StreamWriter w = new StreamWriter(,, Encoding.GetEncoding("windows-1251"));*

* записване на поредния ред във файла – извикване се метода WriteLine, като за аргумент задаваме низа, който трябва да се запише във файла:

*w.WriteLine("Николай Петков");*

* затваряне на файла - след приключване на записването във файл, задължително трябва да затворим файла чрез метода **Close()** на класа StreamWriter:

*r.Close();*

По този начин във файла се добавя знак за край на файла и се извършва съхраняването му. Ако не затворим файла, тогава няма гаранция, че изпратеното действително е записано в него.

**5. Работа с компютър:**

**Задача 1:** Отворете графичното приложение Pizza (създадено в началото на учебната година) и изпълнете зад. 4.5

# Целия код, за който не може да се справи.

……….

private void button1\_Click(object sender, EventArgs e)

{

if ((radioButton1.Checked == false && radioButton2.Checked == false && radioButton3.Checked == false) || (radioButton4.Checked == false && radioButton5.Checked == false))

{

MessageBox.Show("Моля, изберете вариант на пицата и начин на плащане.", "Грешка");

}

if ((radioButton1.Checked == true || radioButton2.Checked == true || radioButton3.Checked == true) && (radioButton4.Checked == true || radioButton5.Checked == true))

{

MessageBox.Show("Вашата поръчка беше приета и записана в системата.", "Поръчка");

StreamWriter w = new StreamWriter("pizza.txt", true, Encoding.GetEncoding("UTF-8"));

w.WriteLine(comboBox1.Text);

w.WriteLine(label2.Text);

w.Close();

}

}

**Задача 2:** Отворете графичното приложение ***Bal***, създадено в предходните часове и добавете код:

* в процедурата за бутон „БАЛ1“ за записване на входящия номер, името и фамилията и пресметнатият бал 1 в текстов файл с ***bal1.txt***

StreamWriter w = **new** StreamWriter("bal1.txt", **true**, Encoding.**GetEncoding**("UTF-8"));

w.**WriteLine**("Входящ номер    " + "Собствено и фамилно име    " + "БАЛ 1");

w.**Write**(*textBox1*.Text + "            ");

w.**Write**(*textBox2*.Text + "              ");

w.**WriteLine**(*textBox8*.Text);

w.**WriteLine**();

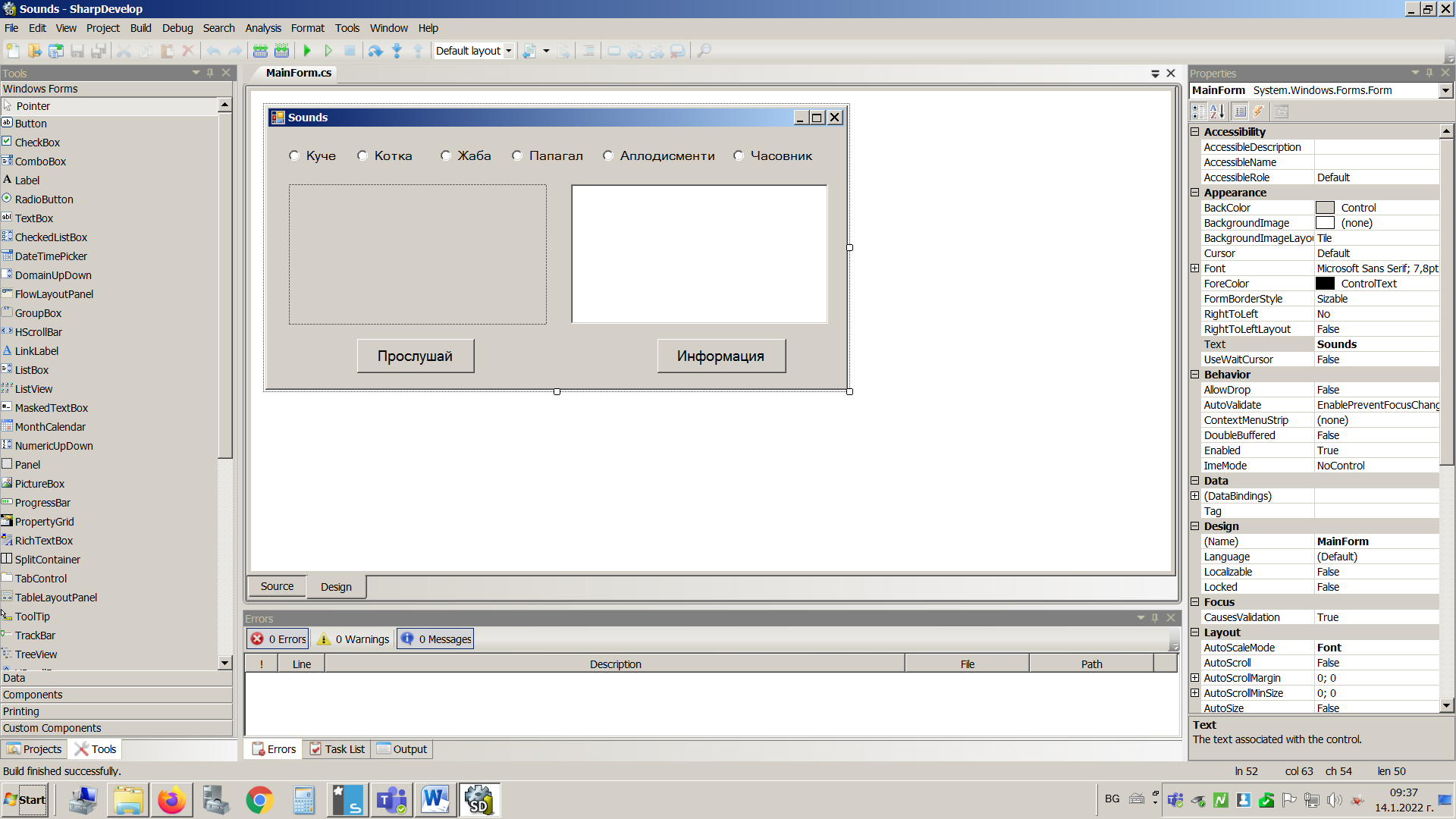
w.**Close**();

MessageBox.**Show**("Вашите данни бяха записани в системата.", "Данни за кандидат-гимназист");

* в процедурата за бутон „БАЛ2“ за записване на входящия номер, името и фамилията и пресметнатият бал 2 в текстов файл с ***bal2.txt***

**Задача 3:** Да се разработи програма „Звуци”, която да позволява на потребителя да избира измежду множество графични, звукови и текстови файлове (приложение Sounds).

Интерфейса трябва така да бъде организиран, че звуците да са достъпни чрез радио бутони. При избора на радио бутон да се визуализира графично представяне на звука, той да се прослуша чрез бутон *Прослушай*, а информация да се извежда чрез бутон *Информация*. Потребителят не трябва да има възможност да разтяга прозореца (*Упътване:* Забраняваме на формата да се максимизира и преоразмерява чрез свойствата ѝ MaximizeBox=False и FormBorderStyle=Fixed3D).



**Решение**

За да реализираме тази програма ще е необходимо да използваме следните компоненти: radioButton, pictureBox, richTextBox, button.

За да може да се прослушват отделните звуци е необходимо да се включи модула **using** System.Media;. В него са дефинирани операторите за просвирване на звукови некомпресиране файлове с разширение .wav:

SoundPlayer myPlayer = **new** System.Media.SoundPlayer();

myPlayer.SoundLocation = @"audio\dog.wav";

myPlayer.**Play**();

Първият оператор описва начина на изпълнение на звука. В нашия случай сме задали звуковия файл да се изпълни в зависимост от стандартните настройки на операционната система Windows, т.е. чрез Windows Media Player.

Вторият оператор описва пътя до съответния файл.

Третият оператор описва операцията за изпълнение на звука.

Изпълнението на всеки звуков файл трябва да е свързан с бутона *Прослушай*, като се прави проверка на кой от радио-бутоните свойството Checked е истина.

**if** (*radioButton1*.Checked==**true**)

{

SoundPlayer myPlayer = **new** System.Media.SoundPlayer();

myPlayer.SoundLocation = @"audio\dog.wav";

myPlayer.**Play**();

}

Визуализирането на съответнити изображение се изпълнява веднага след избора на радио-бутон, т.е. когато се генерира събитието OnClick. За зареждането му в елемента pictureBox се използва оператор, който съдържа един аргумент сочещ пътя за достъп до съответния файл:

void **radioButton1\_CheckedChanged**(object sender, EventArgs e)

{

*pictureBox1*.Image=Image.**FromFile**("image/dog.wmf");

}

Разработете останалата част от приложението.

Всичките необходими изображения се намират в папката image, звуковите файлове – в папка audio, а текстовите – в папка info.

|  |  |  |  |
| --- | --- | --- | --- |
| **Радио-бутон** | **Звук** | **Картинка** | **Информация** |
| Куче | dog.wav | dog.wmf | dog.txt |
| Котка | cat.wav | cat.wmf | cat.txt |
| Жаба | frog.wav | frog.wmf | frog.txt |
| Папагал | bird.wav | bird.wmf | bird.txt |
| Часовник | tick.wav | tick.wmf | tick.txt |
| Аплодисменти | applause.wav | applause.wmf | applause.txt |

Таблица 1. Представяне на радио-бутоните и съответстващите им звукови и графични файлове