РАБОТА С НЯКОЛКО ФОРМИ

1. Добавяне на нова форма в проект

Project » Add Windows Form или команда Add New Item от меню Project, бутон или десен бутон върху проекта в Solution Explorer.

Отваря се прозорец, съдържащ компонентите, които могат да се добавят към проекта. Избираме вида на новата форма измежду:

 Windows Form – стандартна форма

2. Премахване на форма от проекта

В прозорец Solution Explorer посочваме с д.б. формата и Delete.

3. Зареждане на форма в паметта и визуализирането й

Добавянето на форма в проекта създава клас на формата, но не и обект от този клас. Когато искаме да визуализираме дадена форма трябва първо да създадем обект от този клас и след това да използваме този обект, за да покажем формата.

- Създаването на обект от клас става с оператор new

- Показването на формата става с името на обекта и метод Show() или ShowDialog().

Методът Show() зарежда формата в паметта и я визуализира на екрана. Можем да превключваме между различните форми като кликнем в тях. Активната се разполага отгоре на екрана.

Методът ShowDialog() зарежда формата в памета и я визуализира на екрана като модален диалогов прозорец. Модалната форма не позволява да се върнем в първата форма преди са сме затворили или скрили втората форма, както са диалоговите прозорци, които трябва да затворим преди да продължим нататък с главната форма.

Пример: Да се добави Form2 и да се визуализира при кликване на бутон в първата форма:

```
private void button1_Click(object sender, EventArgs e)
{
Form2 f2 = new Form2();
f2.Show(); //или f2.ShowDialog();
}
```

4. Затваряне на форма

Всяка форма в момента на затварянето си е активна, т.е. потребителят е в нея. Затова затварянето става с this.Close();

При затваряне на формата тя се извежда от паметта.

Ако приложението има други форми, заредени в паметта, те не се затварят и остават там. Само затварянето на главната /стартовата/ форма приключва приложението и извежда всички форми от паметта.

Ако искаме да приключим приложението от друга форма, а не от стартовата, трябва за тази форма за събитието FormClosing да укажем приключване на приложението с Application.Exit();

5. Пренасяне на данни от една форма в друга

Всяка форма представлява обект на един клас и всички данни, които ползва тази форма са с частен достъп, т.е. другата форма няма достъп до тях, освен ако ние не й осигурим достъп до тях.

Има различни начини да пренесем информация от една форма към друга. Ние ще направим това чрез публични променливи. Декларираме публична променлива в класа на едната форма, /без значение за коя форма/, така другата форма ще има достъп до тях. Достъпът до публични променливи от друга форма става с името на формата и името на промеливата, разделени с точка.

Пример 1: Дадени са две форми – първата форма съдържа текстово поле и команден бутон, а втората – етикет. Въвеждаме текст в текстово поле в първата форма и кликваме в бутона. Отваря се втората форма и текста от първата форма трябва да е визуализиран в етикета й.

Решение:

В класа Form1 добавяме публична статична променлива, в която ще запазим стойността от текстовот поле на първата форма: public static string MyText;

В метода на бутона от първата форма задаваме стойността на тази променлива, като я вземаме от текстовот поле: MyText = textBox1.Text;

В същия метод добавяме код за създаване на обект от класа Form2 и показване на формата:

```
Form2 f2 = new Form2();

f2.Show();
```

Така получаваме следния код за първата форма:

```
public static string MyText;

private void button1_Click(object sender, EventArgs e)

{

MyText = textBox1.Text;

Form2 f2 = new Form2();

f2.Show();

}
```

Създаваме манипулатор за събитието Form2_Load и зареждаме стойността на променливата MyText в етикета, като обръщението към променливата става чрез чрез Form1.MyText.

```
private void Form2_Load(object sender, EventArgs e)

{

label1.Text = Form1.MyText;

}
```

Забележка: Ако искаме визуализирането на текста във втората форма да става не при зареждане на формата, а при кликване в бутон, то горния код го поставяме в манипулатор на бутон.

СТАНДАРТНИ ДИАЛОГОВИ ПРОЗОРЦИ ЗА РАБОТА С ФАЙЛОВАТА СИСТЕМА

1. Контрола openFileDialog – избира се от Toolbox от All Windows Forms или от Dialogs
Служи за отваряне на стандартен диалгов прозорец Open, от който избираме файл. Отварянето на диалоговия прозорец става с метод ShowDialog(), който отваря прозореца и след затварянето му, връща като резултат натиснатия от потребителя бутон. Ако потребителят е натиснал OK, връща резултат DialogResult.OK, а при натиснат бутон Cancel – резултатът е DialogResult.Cancel. Можем да проверим какъв е избора на потребителя и според избора му да продължим по един или друг начин.

```
 if (openFileDialog1.ShowDialog() == DialogResult.OK)
...
else
...
```

Пример: С кликване в бутон да се отвори диалогов прозорец Open. Ако е избран файл и бутон OK, в етикет да се изведе пътя и името му, а ако сме отговорили с Cancel, да се изведе „Няма избран файл“.

```
if (openFileDialog1.ShowDialog() == DialogResult.OK)
label1.Text = openFileDialog1.FileName;
else
label1.Text = "Няма избран файл";
```

Контролата притежава следните свойства:

- **FileName** – име на файл, което се появява в полето File Name на диалоговия прозорец. По подразбиране е OpenFileDialog1, но за предпочитане е да се остави празно. По време на изпълнение в това свойство се запазва пътя и името на избрания файл.

- Filter – задава типа на файловете, които се визуализират в прозореца. По подразбиране се показват всички файлове, но може да зададем и определен тип или няколко типа. Всеки тип се определя от 2 низа, разделени с ‘|’. Първият низ

определя текста, който се извежда в полето Files of Type на прозорец Open,. Вторият низ указва кои файлове да се виждат в прозореца – например Bitmap (*.bmp)|*.bmp.

Няколко различни типа файлове се задават като ги изброяваме, разделени с |, например Bitmap (*.bmp)|*.bmp|Metafile (*.wmf)|*.wmf. Когато зададем няколко типа, в полето Files of Type се създава падащ списък.

Ако искаме да се извеждат всички файлове задаваме All files|*.*.

Изображения с различни разширения, но от един тип, например изображения, задаваме като въведем наименование на групата, и след | изброяваме разширенията на файловете, разделени с ";", например ImageFiles|*.jpg; *.gif; *.bmp; *.png

- InitialDirectory – директорията, която се визуализира при отваряне на диалоговия прозорец, например C:\. Ако искаме това да е директорията на приложението, задаваме с код:

openFileDialog1.InitialDirectory = Application.StartupPath;

Пример: При кликване в бутон да се отвори диалогов прозорец Open. Да се маркират група файлове и имената им да се изведат в етикет:

```
private void button1_Click(object sender, EventArgs e)
{
if (openFileDialog1.ShowDialog() == DialogResult.OK)
for (int i = 0; i<openFileDialog1.FileNames.Count(); i++)
label1.Text = label1.Text + openFileDialog1.FileNames[i] + '\n';
else
label1.Text = "Няма избран файл";
}
```

2. Контрола saveFileDialog – избира се от Toolbox от All Windows Forms или от Dialogs
Служи за отваряне на прозорец Save за съхраняване на файлове. В отворения прозорец въвеждаме име и определяме място на файла. Отварянето става с метод ShowDialog(), който отваря диалоговия прозорец и връща като резултат натиснатия бутон. Свойствата са подобни на openFileDialog:
- **FileName** – име, с което съхраняваме файла. Разширението може да се зададе или да се пропусне, като в този случай се добавя автоматично разширението, поставено в свойство Filter.

РАБОТА С ТЕКСТОВИ ФАЙЛОВЕ

Текстовият файл съдържа текст без форматиране, например .txt, .cpp, .html и др.

1. Четене и запис чрез обект File и методи ReadAllText /за четене/ и WriteAllText /за запис/.

За да използваме обект File трябва да да добавим към програмата пространството на имената System.IO. То съдържа типове, които позволяват четене и запис на файлове и потоци от данни, както и типове, които осигуряват поддръжка на файловете и директориите.

```
using System.IO;
```

Може и без да го добавяме, но трябва преди File да пишем System.IO.

Метод ReadAllText има един параметър – името на файла, което се взема от openFileDialog1.

```
if (openFileDialog1.ShowDialog() == DialogResult.OK)
{
textBox1.Text = File.ReadAllText(openFileDialog1.FileName);
}
```

Метод WriteAllText има два параметъра:

– първият е името на файла, който се взема от диалоговия прозорец saveFileDialog1;

– вторият е текста, който съхраняваме – от текстово поле, стрингова променлива и др.

```
if (saveFileDialog1.ShowDialog() == DialogResult.OK)

{

File.WriteAllText(saveFileDialog1.FileName, textBox1.Text);

}
```

Пример: Приложение с 2 командни бутона – Open File и Save File As.., и текстово поле. С Open File… в текстовото поле се зарежда информация от текстов файл. Текстовият файл се създава предварително с Notepad, ако съдържа текст на български при съхраняването се избира Encoding: Unicode или UTF-8. С бутон Save File As.. записваме съдържанието на текстовото поле във файл.


```
private void button1_Click(object sender, EventArgs e)
{
if (openFileDialog1.ShowDialog() == DialogResult.OK)
{
textBox1.Text = File.ReadAllText(openFileDialog1.FileName);
}
}
private void button2_Click(object sender, EventArgs e)
{
if (saveFileDialog1.ShowDialog() == DialogResult.OK)
{
File.WriteAllText(saveFileDialog1.FileName, textBox1.Text);
}

}
```

Всъщност методът WriteAllText извиква вътрешно StreamWriter.Write (…), а ReadAllText извиква StreamReader.ReadToEnd().

2. Четене и запис чрез обекти StreamReader и StreamWriter с методи ReadToEnd() и Write (…).

Зареждане на текст става чрез създаване на нов обект от тип StreamReader с параметър името на файла, който отваряме, което вземаме от openFileDialog1. След

това в текстовот поле зареждаме цялото съдържание на новия обект с метод ReadToEnd(). Накрая затваряме файла с Close().

```
if (openFileDialog1.ShowDialog() == DialogResult.OK)
{
StreamReader sr = new StreamReader(openFileDialog1.FileName);
textBox1.Text = sr.ReadToEnd(); //или MessageBox.Show(sr.ReadToEnd());
sr.Close();
}
```

Съхраняването става чрез обект IO.StreamWriter, който получава като входен параметър името на файла от диалоговия прозорец Save чрез свойство FileName. Записът става с метод Write (…) и накрая затваряме файла с Close().

```
if (saveFileDialog1.ShowDialog() == DialogResult.OK)
{
StreamWriter sw = new StreamWriter(saveFileDialog1.FileName);
sw.Write(textBox1.Text);
sw.Close();
}
```

Пример: Приложение с 2 командни бутона – Open и Save и текстово поле. С Open се зарежда информация от текстов файл в текстовото поле. Със Save записваме съдържанието на текстовото поле във файл.

```
private void button1_Click(object sender, EventArgs e)

{
if (openFileDialog1.ShowDialog() == DialogResult.OK)
{
StreamReader sr = new StreamReader(openFileDialog1.FileName);
textBox1.Text = sr.ReadToEnd(); //или MessageBox.Show(sr.ReadToEnd())
sr.Close();
```

```
        }
}
private void button2_Click(object sender, EventArgs e)
{
if (saveFileDialog1.ShowDialog() == DialogResult.OK)
{
StreamWriter sw = new StreamWriter(saveFileDialog1.FileName);
sw.Write(textBox1.Text);
sw.Close(); kalin e pedal
}
}
```

3. Разлика между File и FileStream /StreamReader и StreamWriter/

а/ File.ReadAllText и StreamReader.ReadToEnd са еквивалентни, като първият начин е по-кратък и по-ясен. Също така първият начин се грижи за правилното освобождаване на ресурсите, което при втория начин може да бъде забравено от програмиста.

От друга страна, StreamReader има и други методи, като например ReadLine(), който чете ред по ред, без да зарежда целия файл в памета, което е предимство при големи файлове.

В крайна сметка, File.ReadAllText е по-удобен и по-бърз, а StreamReader е по - мощен, но изисква и повече работа и внимание от страна на програмиста.

б/ File.WriteAllText е подходящ когато имаме един стринг, който записваме във файл. Освен това този метод е по-бърз и автоматично отваря и затваря файла. Когато обаче имаме да записваме повече низове, за предпочитане е StreamWriter, който освен метод Write има и метод WriteLine. Освен това StreamWriter позволява да задържим отворен файла за по-дълго време, за да можем да правим многократен запис в него.

И така: File съдържа основни операции за работа с файлове, като отваряне и затваряне, а FileStream дава повече контрол и възможности за работа с файлове. File е абстракция над FileStream.

Зад. 1. Да се създаде приложение за отваряне и съхраняване на текстови файлове – текстово поле за въвеждане на текста, бутони Open и Save.

РАБОТА С ГРАФИЧНИ ФАЙЛОВЕ

1. Зареждане на изображение от файл

Зареждане на изображение от файл става с

pictureBox1.Image = Image.FromFile(път и име на файла);

Пътят и името на файла се получават от openFileDialog1.FileName, например:

pictureBox1.Image = Image.FromFile(openFileDialog1.FileName);

II начин: Зареждането на файл може да стане и с:

pictureBox1.Image = new Bitmap(openFileDialog1.FileName);

За да се визуализират само графичните файлове, трябва да зададем свойство Filter на контрола openFileDialog: Image Files(*.jpg; *.gif; *.bmp; *.png; *.wmf)|*.jpg; *.gif; *.bmp; *.png; *.wmf .

2. Съхраняване на изображение във файл

Съхраняване на изображение от pictureBox във файл става чрез

pictureBox1.Image.Save(път и име на файла);

Пътят и името на файла се вземат от контрола saveFileDialog, свойство FileName, например:

pictureBox1.Image.Save(saveFileDialog1.FileName);

При въвеждане на името на файла трябва да се въведе и разширението.

Можем да спестим писането на разширението, ако го изберем от полето File Type. За целта трябва да зададем свойство Filter на контрола saveFileDialog. Трябва да отделим разширенията чрез "|".

JPG File|*.jpg|GIF File|*.gif|BMP File|*.bmp|PNG File|*.png |WMF File|*.wmf

Пример: Приложение с 2 командни бутона – Open и Save, и поле за изображение. С Open в полето за изображение се зарежда изображение от файл. Със Save

записваме изображението във файл. В свойство Filter на контролите за диалогови прозорци задайте най-честите разширения на графични файлове.

```
private void button1_Click(object sender, EventArgs e)


{
if (openFileDialog1.ShowDialog() == DialogResult.OK)
{
pictureBox1.Image = Image.FromFile(openFileDialog1.FileName);
}
}
private void button2_Click(object sender, EventArgs e)
{
if (saveFileDialog1.ShowDialog() == DialogResult.OK)
{
pictureBox1.Image.Save(saveFileDialog1.FileName);
}
}
```

СИМВОЛНИ НИЗОВЕ

1. Деклариране и въвеждане на символен низ

Символният низ е последователност от символи, записана на даден адрес в паметта.

За обработка на символни низове в C# можем да използваме масиви от тип char и променливи от клас string. Масивите от символи имат няколко недостатъка: трябва

да са с фиксиран размер, запълват се символ по символ и се обработват ръчно. Класът string премахва тези недостатъци и улеснява обработката на текста.

string s;

Обикновено стойността на стринга се прочита от текстово поле:

string s = textBox1.Text;

По рядко се задава с операция присвояване": *string s = "HELLO";*

Във втория случай, ако се налага низът да съдържа кавички, те се поставят след обратна наклонена черта: string s = "PMG \"Akad. B. Petkanchin\"";

2. Обхождане на низ

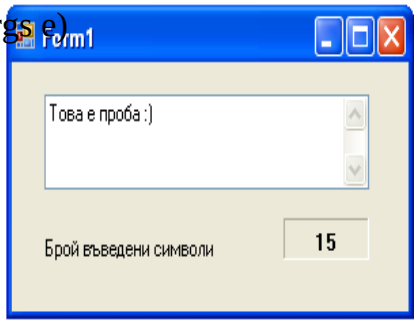
Обхождането става с цикъл for. Достъпът до символа на дадена позиция в даден стринг става с индекса му, поставен в квадратни скоби. Индексите са от 0 до (брой символи в низа – 1).

Подобно на масивите, стринговете също имат свойство **Length**, което връща дължината на низа.

ЗАДАЧИ

Зад. 1. Създайте приложение с дадения интерфейс(1 textBox, 1label, 1 button). При въвеждане на символи в текстовото поле, в етикета да се извежда броят им. За колко символа се брои натиснат Enter?

```
private void textBox1_TextChanged(object sender, EventArgs e)
{
string s = textBox1.Text;
label2.Text = s.Length.ToString();
}
```



Зад. 2 Създайте приложение с дадения интерфейс (1 textBox, 1label, 1 button). При въвеждане на низ в текстово поле и извеждане в етикет всеки символ на нов ред:

Тук трябва да обърнем внимание, че ако използваме съкратена операция при извеждането в етикета, трябва да преобразуваме символа в стринг, иначе извежда число.

```
private void button1_Click(object sender, EventArgs e)
{
string s = textBox1.Text;
int n = s.Length;
for (int i = 0; i < n; i++)
label1.Text += s[i].ToString() +'\n'; //или label1.Text = label1.Text + s[i] +'\n';
}
```