

Съдържание

- Какво е Windows Forms?
- Програмни компоненти. Компонентен модел на .NET
- Програмен модел на Windows Forms
- Основни класове. Йерархия на класовете
- Класът `Control`. Други базови контроли
- Модел на пречертаване на контролите
- Форми, прозорци и диалози – класът `Form`
- Основни контроли – `TextBox`, `Label`, `Button`
- Поставяне на контроли във формата
- Управление на събитията
- Windows Forms редакторът на VS.NET

Съдържание (2)

- Стандартни диалогови кутии
- Извикване на диалогови кутии
- Други Windows Forms контроли. Менюта. Ленти с инструменти. Статус ленти
- Диалог за избор на файл
- MDI приложения
- Валидация на данни
- Свързване на данни (Data Binding). Навигация с `CurrencyManager`
- Контролата `DataGrid`
- Master-Details навигация
- Наследяване на форми

Съдържание (3)

- Пакетът `System.Drawing` и GDI+
 - Печатане на принтер
 - Потребителски контроли
 - Хостинг на контроли в Internet Explorer
 - Нишки и Windows Forms
 - Влачене (Drag and Drop)
 - Конфигурационен файл на приложението
- 

Какво е Windows Forms?

- Windows Forms

- Библиотека за изграждане на прозоречно-ориентиран графичен потребителски интерфейс (GUI)
- Поддържа концепцията за Rapid Application Development (RAD)
 - компонентно-ориентирана архитектура
 - управление, базирано на събития
- Прилича на GUI средствата на Delphi и Visual Basic 6, с мощта на MFC
- Съдържа богат набор от контроли

Какво е Windows Forms?

- Windows Forms

- Позволява наследяване и разширяване на форми и контроли
- Поддържа ActiveX контроли
- Поддържа печатане на принтер
- Контролите могат да се изпълняват в Internet Explorer
 - подобно на Java аpletите
 - без предварителна регистрация (за разлика от ActiveX контролите)
- Силна поддръжка на графика (GDI+)

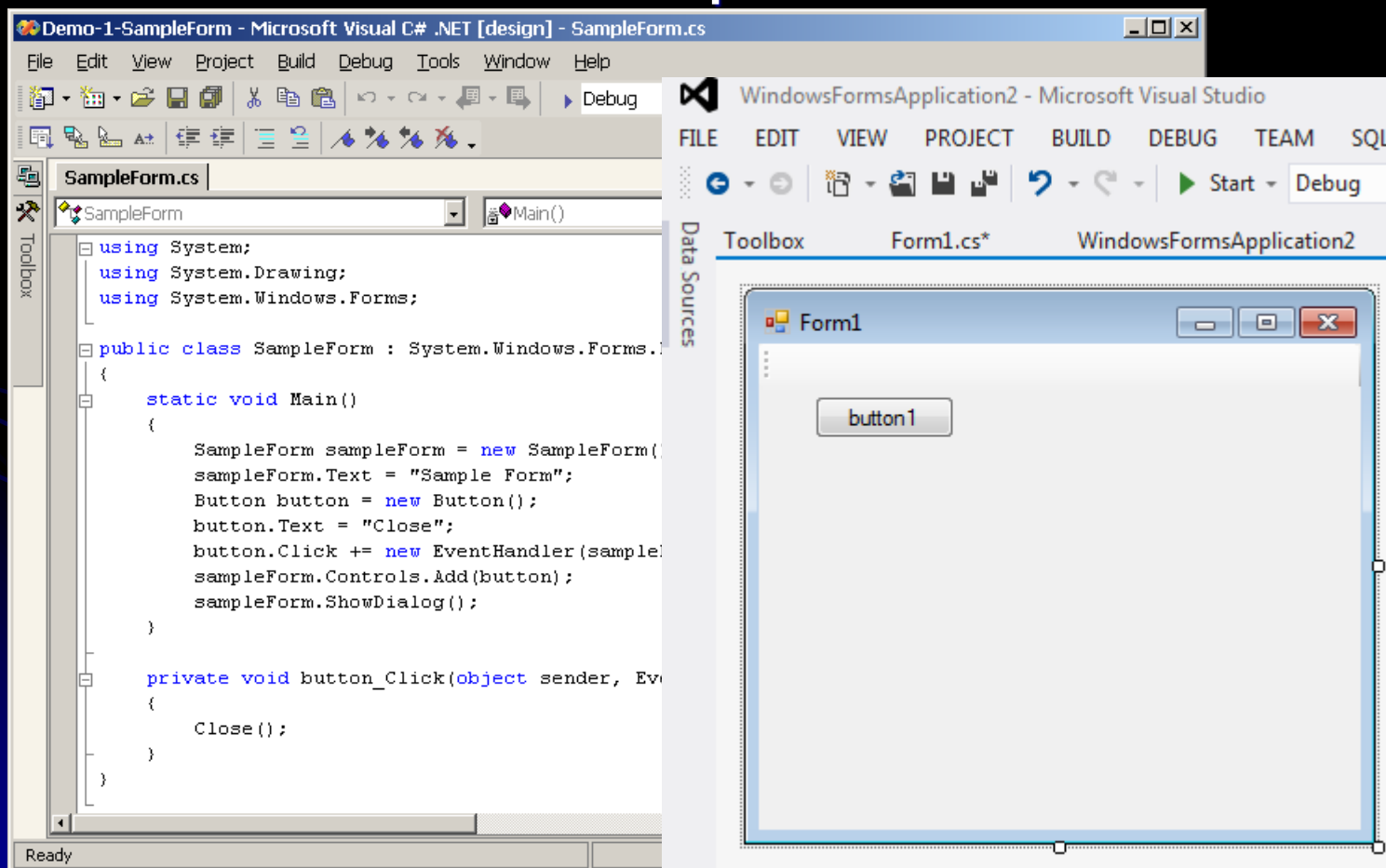
Windows Forms – пример

```
public class SampleForm : System.Windows.Forms.Form
{
    static void Main()
    {
        SampleForm sampleForm = new SampleForm();
        sampleForm.Text = "Sample Form";
        Button button = new Button();
        button.Text = "Close";
        button.Click +=
            new EventHandler(sampleForm.button_Click);
        sampleForm.Controls.Add(button);
        sampleForm.ShowDialog();
    }

    private void button_Click(object sender, EventArgs e)
    {
        Close();
    }
}
```

Демо 1

- Windows Forms приложение



Пакетите от Windows Forms

System.Windows.Forms

Design

Осигурява средства за работа с прозорци, диалози, контроли за въвеждане на текст, за избор, менюта, ленти с инструменти, таблици, дървета и др.

ComponentModel

System.Drawing

Drawing2D

Imaging

Осигурява достъп до GDI+ функциите на Windows – работа с точки, линии, геометрични фигури, изобразяване на картинки, текст, шрифтове, печатане на принтер и др.

Програмни компоненти

- Компоненти

- Програмни единици (класове), които решават специфична задача
- Преизползваеми (reusable)
- Имат ясно дефиниран интерфейс, който описва техните:
 - свойства
 - методи
 - събития
- Използват се като част от други компоненти или програми

Компонентен модел на .NET

- Компонентният модел дефинира:
 - стандарти за разработката и използване на програмни компоненти
 - жизнен цикъл на компонентите
- Компонентният модел на .NET Framework
 - налага правила за създаване и използване на .NET компоненти (програмен модел)
 - дефинира класове и интерфейси, които поддържат описанието на компоненти

Компонентен модел на .NET

- Компонентният модел на .NET
 - позволява дефиниране на поведението на компонентите
 - по време на дизайн (design-time behavior)
 - по време на работа (runtime behavior)
 - осигурява лесна преизползваемост (reusability)
 - дефинира компоненти и контейнери
- CLR осигурява междуетикова съвместимост на компонентите
- Основната функционалност на компонентния модел се намира в `System.ComponentModel`

Windows Forms и комп. модел

- Windows Forms е базиран на компонентния модел на .NET
- Компонентният модел дефинира компоненти и контейнери
- Windows Forms дефинира контроли и контейнер-контроли
- Контролите са видими за потребителя компоненти:
 - контейнер-контроли (форми, диалози, панели, ...) – съдържат други контроли
 - контроли (бутони, текстови полета, ...) – съдържат се в контейнер контролите

Програмен модел на WinForms

- Програмният модел на Windows Forms дефинира:
 - форми (прозорци и диалози)
 - контроли (текстови полета, бутони, менюта, ленти с инструменти, ...)
 - събития, които ги управляват
 - жизнен цикъл на приложенията
 - модел на пречертаване на контролите
 - управление на фокуса и навигация
- Жизненият цикъл на приложенията е базиран на съобщения
 - Контролите получават съобщения за потребителските действия и реагират по специфичен начин

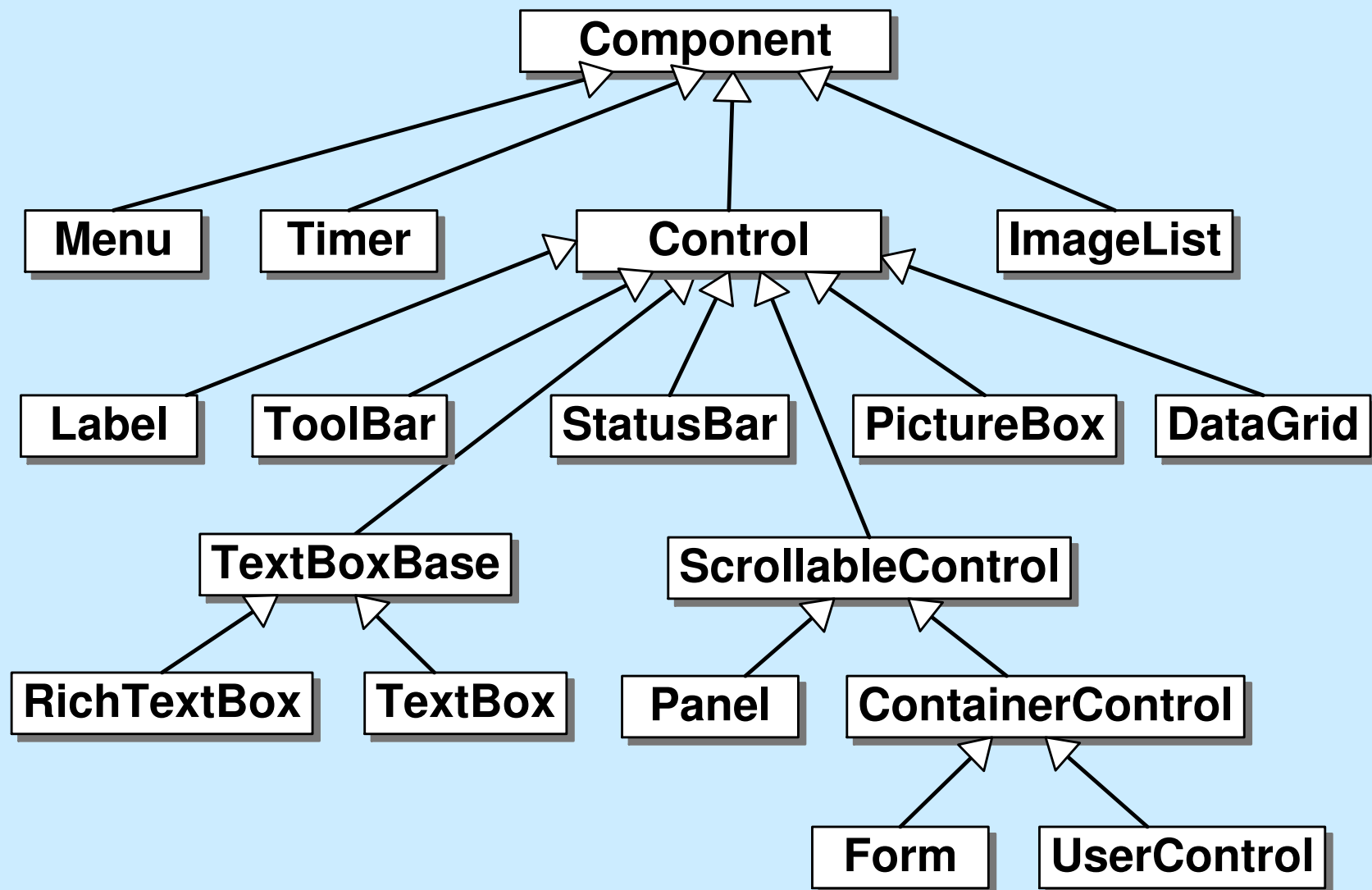
Програмен модел на WinForms

- Главната нишка на всяко Windows Forms приложение работи така:
 - Постоянно слуша за съобщения
 - При получаване на съобщение (напр. преместване на мишката, натискане на клавиш или др.) го обработва така:
 - намира контролата, за която се отнася съобщението
 - предава ѝ съобщението
 - ако контролата е контейнер-контрола, тя търси в себе си за коя от нейните контроли е съобщението и ѝ го предава
 - При затваряне на главната форма на приложението, то спира изпълнението си

ОСНОВНИ КЛАСОВЕ

- Библиотеката Windows Forms дефинира:
 - съвкупност от базови класове за контролите и контейнер-контролите
 - множество графични контроли
- Основни базови класове:
 - `Component` – .NET компонент
 - `Control` – графична контрола (компонента с графичен образ)
 - `ScrollableControl` – контрола, която поддържа скролиране на съдържанието си
 - `ContainerControl` – контрола, която съдържа други контроли и управлява поведението на фокуса

Йерархия на класовете



Класът Control

- Класът `System.Windows.Forms.Control` е основа на всички графични Windows Forms контроли
- Неговите свойства са типични за всички Windows Forms контроли
- По-важните свойства на класа `Control`:
 - `Anchor`, `Dock` – задават по какъв начин контролата се "закотвя" за контейнера си
 - `Bounds` – задава размера и позицията на контролата в нейния контейнер
 - `BackColor` – задава цвета на фона
 - `ContextMenu` – задава контекстно меню (popup menu) за контролата

Класът Control

- По-важните свойства на класа `Control`:
 - `Controls` – съдържа колекция от вложените контроли (ако има)
 - `CanFocus` – връща дали контролата може да получава фокуса
 - `Enabled` – позволява забраняване на контролата (тя става видима, но неактивна)
 - `Font` – задава шрифта (име, стил, размер)
 - `ForeColor` – задава цвета на контролата
 - `Location` – съдържа позицията на контрола в нейния контейнер
 - `Parent` – задава контейнер-контролата, съдържаща текущата контрола

Класът Control

- По-важните свойства на класа Control:
 - Size – съдържа размерите на контролата
 - TabIndex – определя реда при навигация с клавиша [TAB]
 - TabStop – задава дали контролата може да се фокусира при навигация с [TAB]
 - Text – задава текст, свързан с контролата
 - Visible – задава видимост на контролата
- По-важни методи на класа Control:
 - Focus() – фокусира контролата
 - Hide(), Show() – скрива/показва контролата

Класът Control

- По-важните събития на класа Control:
 - Click – настъпва при щракване с мишката върху контролата
 - Enter, Leave – настъпват при активиране и деактивиране на контролата
 - KeyDown, KeyUp – настъпват при натискане и отпускане на клавиш (или комбинация)
 - KeyPress – при натискане на нефункционален клавиш
 - MouseDown, MouseUp, MouseHover, MouseEnter, MouseLeave, MouseMove, MouseWheel – настъпват при събития от мишката, настъпили върху контролата

Класът Control

- По-важните събития на класа Control:
 - Move – настъпва при преместване на контролата
 - Paint – настъпва при пречертаване на контролата
 - Resize – настъпва при промяна на размера на контролата
 - TextChanged – настъпва при промяна на СВОЙСТВОТО Text на контролата
 - Validating – използва се за валидация на данните, въведени в контролата

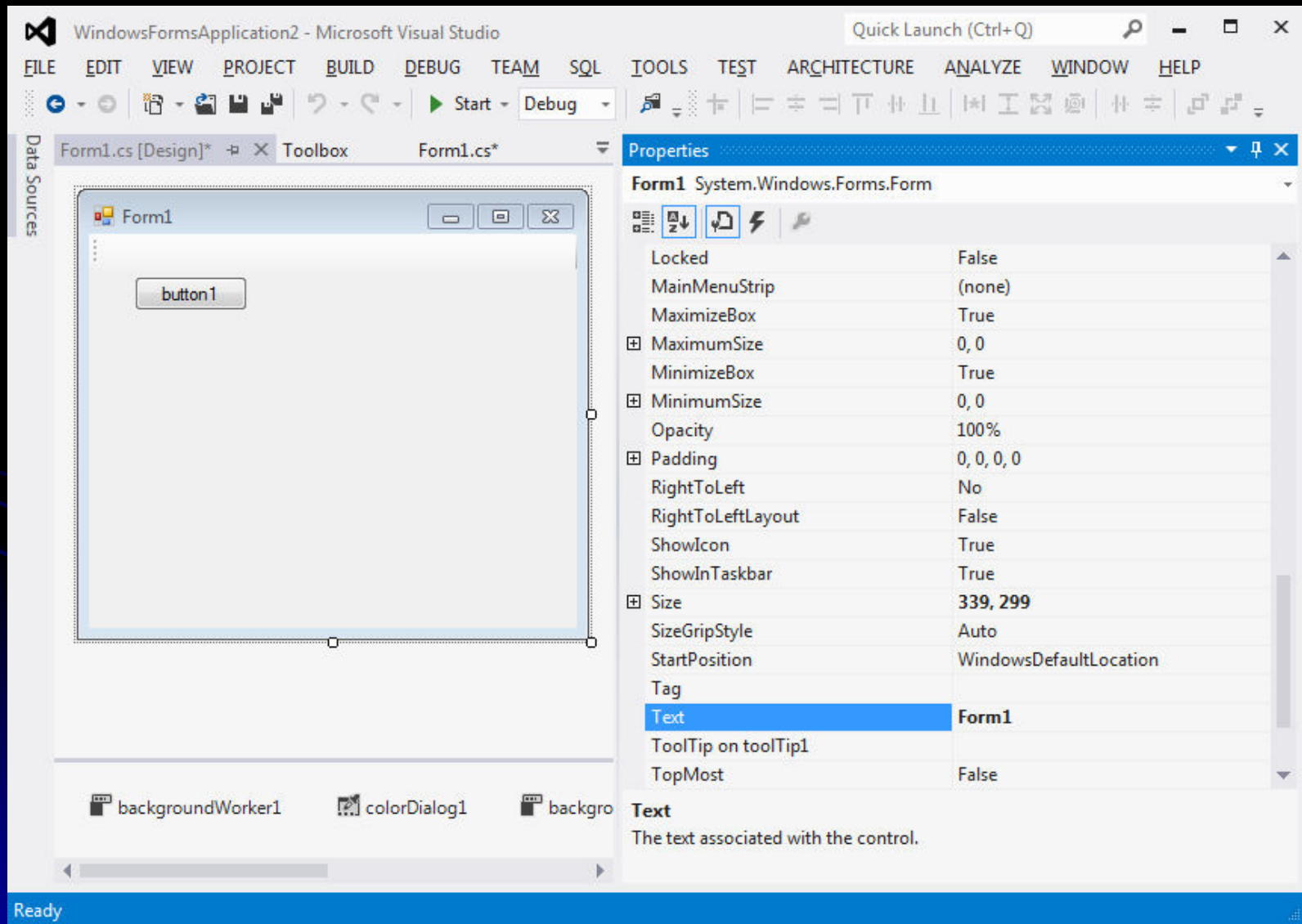
Пречертаване на контролите

- В Windows Forms контролите често се пречертават
 - при преместване на прозорец
 - при смяна на активния прозорец
 - при промяна на размера или позицията на някоя контрола
- Пречертаването става на два етапа:
 - `Invalidate()` – подготвя за пречертаване дадената контрола или неин участък и изпраща съобщение "пречертай"
 - `Paint()` – обработва съобщението "пречертай", изпратено от `Invalidate()` и обновява графично указания участък

Други базови контроли

- Класът `ScrollableControl`
 - добавя функционалност за скролиране
 - `AutoScroll` – задава дали при нужда контролата ще получи автоматично скролиращи ленти
 - `HScroll`, `VScroll` – задават дали контролата да има хоризонтална и вертикална скролираща лента
- Класът `ContainerControl`
 - осигурява функционалност за управление на фокуса
 - `ActiveControl` – съдържа контролата, която е на фокус

Form



Форми, прозорци и диалози

- Класът `System.Windows.Forms.Form`
 - Представява форма (прозорец или диалогова кутия) в GUI приложенията
 - В него могат да се добавят контроли
- Основни свойства на класа `Form`:
 - `FormBorderStyle` – указва типа на рамката на формата
 - `Sizable` – стандартна разширяема рамка
 - `FixedDialog` – диалог с фиксирани размери
 - `None` – без рамка
 - `FixedToolWindow` – кутия с инструменти с фиксиран размер

Форми, прозорци и диалози

- Основни свойства на класа `Form`:
 - `Controls` – списък с контролите разположени във формата
 - `Text` – заглавие на прозореца
 - `Size` – размери на прозореца
 - `ClientSize` – размер на вътрешността на формата (без рамката ѝ)
 - `AcceptButton` – бутон по подразбиране
 - `ActiveControl` – контролата, която държи фокуса
 - `ControlBox` – задава дали формата да има контроли за затваряне, минимизация и т. н.

Форми, прозорци и диалози

- Основни свойства на класа `Form`:
 - `Icon` – задава икона на прозореца
 - `KeyPreview` – ако се зададе `true`, позволява формата да обработва събитията от клавиатурата преди да ги предаде на фокусираната контрола
 - `MinimumSize`, `MaximumSize` – задава ограничения за размера на формата
 - `Modal` – задава дали формата е модална
 - `Opacity` – задава прозрачност (0.00 – 1.00)
 - `MdiChildren`, `MdiParent` – извлича/задава подчинените форми/собственика на текущата форма в MDI режим

Форми, прозорци и диалози

- Основни свойства на класа `Form`:
 - `TopMost` – задава дали формата стои над всички други прозорци (`always on top`)
 - `WindowState` – извлича състоянието на формата (нормална, минимизирана или максимизирана)
- Основни методи на класа `Form`:
 - `Close()` – затваря формата (скрива я и освобождава използваните ресурси)
 - `Show()` – показва формата и я активира
 - `ShowDialog()` – показва формата в модален режим и връща като резултат `DialogResult` свойството

Form

WindowsFormsApplication2 - Microsoft Visual Studio

Quick Launch (Ctrl+Q)

FILE EDIT VIEW PROJECT BUILD DEBUG TEAM SQL FORMAT TOOLS TEST ARCHITECTURE ANALYZE WINDOW HELP

Start Debug

Object Browser Form1.cs [Design]*

Data Sources

Form1

button1

backgroundWorker1 colorDialog1 backgro

Form1 System.Windows.Forms.Form

Enter	
FontChanged	
ForeColorChanged	
FormClosed	
FormClosing	
GiveFeedback	
HelpButtonClicked	
HelpRequested	
ImeModeChanged	
InputLanguageChanged	
InputLanguageChanging	
KeyDown	
KeyPress	
KeyUp	
Layout	
Leave	
Load	
LocationChanged	

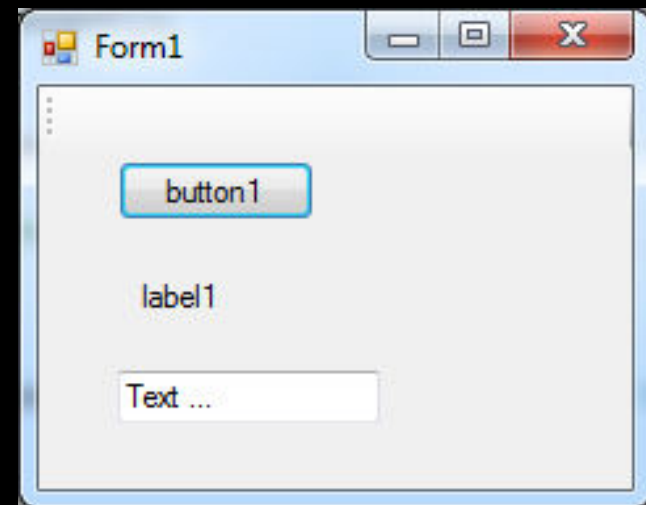
Load
Occurs whenever the user loads the form.

Форми, прозорци и диалози

- Основни методи на класа `Form`:
 - `LayoutMdi (...)` – пренарежда дъщерните (MDI) форми (каскадно, хоризонтално, вертикално)
- Основни събития на класа `Form`:
 - `Activated / Deactivate` – извиква при активиране / деактивиране на формата (получаване / загуба на фокуса)
 - `Closing` – извиква се при затваряне на формата. Реализацията може да предизвика отказване на затварянето
 - `Load` – извиква се еднократно при първото показване на формата. Често се ползва за инициализиране на състоянието

ОСНОВНИ КОНТРОЛИ

- TextBox –  – поле за въвеждане на текст. По-важни свойства:
 - Multiline – задава дали да се допуска въвеждане на няколко реда
 - Text (Lines) – съдържа въведения текст
- Label –  – изобразява текст във формата. Важни свойства:
 - Text – текстът, който се изобразява
- Button –  – бутон за натискане По-важни свойства и събития:
 - Click – активира се при натискане
 - Text – задава текста върху бутона



Поставяне на контроли

- Поставянето на контроли във формата става чрез `Controls.Add`:

```
Form form = new Form();  
  
Button button = new Button();  
button.Text = "Close";  
  
form.Controls.Add(button);
```

- Редът на контролите (т. нар. z-order) се определя от реда на поставянето им (последната контрола е най-отгоре)
- Windows Forms дизайнерът на Visual Studio .NET се грижи за правилното поставяне на контролите

Управление на събитията

- Прихващането на събития става така:

```
Form form = new Form();  
Button button = new Button();  
button.Click += new EventHandler(  
    this.button_Click);  
  
...  
  
private void button_Click(  
    object sender, EventArgs e)  
{  
    // Handle the "click" event  
}
```

- Windows Forms дизайнерът на Visual Studio .NET генерира автоматично обработчиците на събития

Управление на събитията

- Типове събития в Windows Forms:
 - EventHandler – проста нотификация
 - без допълнителни данни
 - KeyEventHandler – събития от клавиатурата
 - подава се кой е натиснатият клавиш и състоянието на [Ctrl], [Shift] и [Alt]
 - MouseEventArgs – събития от мишката
 - подава се позицията на мишката и състоянието на бутоните ѝ
 - CancelEventHandler – събития, които могат да откажат започнало действие

Прост калкулатор – пример

```
using System;
using System.Drawing;
using System.Windows.Forms;

public class CalculatorForm : Form
{
    private TextBox TextBoxNumber1;
    private TextBox TextBoxNumber2;
    private TextBox TextBoxSum;
    private Button ButtonCalc;
    private Label LabelPlus;
    private Label LabelEquals;

    public CalculatorForm()
    {
        TextBoxNumber1 = new TextBox();
        TextBoxNumber1.Bounds = new Rectangle(
            new Point(16, 16), new Size(72, 20));
        TextBoxNumber1.MaxLength = 10;
    }
}
```

(примерът продължава)

Прост калкулатор – пример

```
LabelPlus = new Label();  
LabelPlus.AutoSize = true;  
LabelPlus.Location = new Point(94, 19);  
LabelPlus.Text = "+";
```

```
TextBoxNumber2 = new TextBox();  
TextBoxNumber2.Bounds = new Rectangle(  
    new Point(112, 16), new Size(72, 20));  
TextBoxNumber2.MaxLength = 10;
```

```
LabelEquals = new Label();  
LabelEquals.AutoSize = true;  
LabelEquals.Location = new Point(191, 18);  
LabelEquals.Text = "=";
```

```
TextBoxSum = new TextBox();  
TextBoxSum.Bounds = new Rectangle(  
    new Point(208, 16), new Size(72, 20));  
TextBoxSum.ReadOnly = true;
```

(примерът продължава)

Прост калкулатор – пример

```
ButtonCalc = new Button();  
ButtonCalc.Bounds = new Rectangle(  
    new Point(16, 48), new Size(264, 23));  
ButtonCalc.Text = "Calculate sum";  
ButtonCalc.Click += new EventHandler(  
    this.ButtonCalc_Click);
```

```
this.AcceptButton = ButtonCalc;  
this.ClientSize = new Size(298, 87);  
this.Controls.Add(TextBoxNumber1);  
this.Controls.Add(LabelPlus);  
this.Controls.Add(TextBoxNumber2);  
this.Controls.Add(LabelEquals);  
this.Controls.Add(TextBoxSum);  
this.Controls.Add(ButtonCalc);  
this.FormBorderStyle =  
    FormBorderStyle.FixedDialog;  
this.MaximizeBox = false;  
this.MinimizeBox = false;  
this.Text = "Calculator";
```

```
}
```

(примерът продължава)

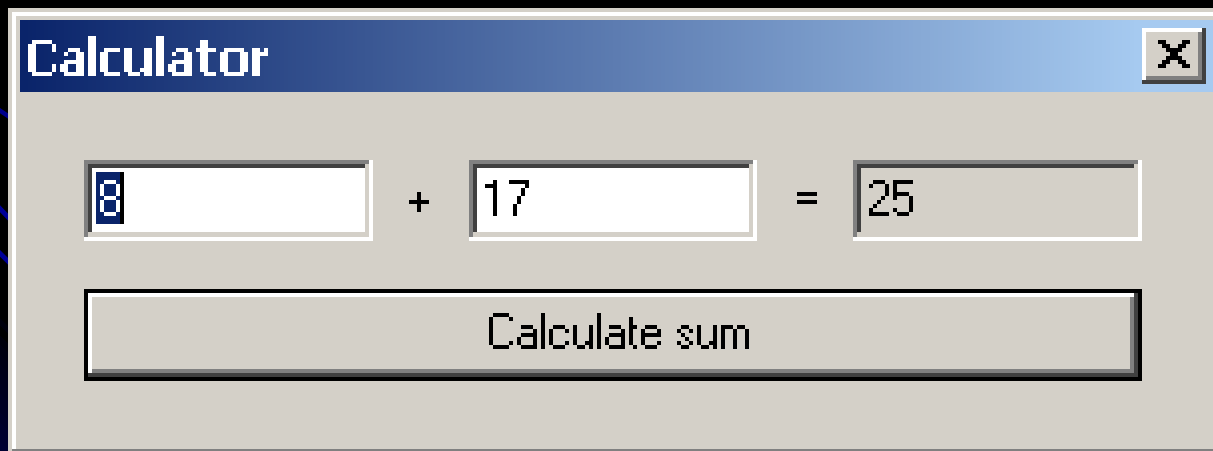
Прост калкулатор – пример

```
private void ButtonCalc_Click(  
    object aSender, EventArgs aArgs)  
{  
    try  
    {  
        int value1 =  
            Int32.Parse(TextBoxNumber1.Text);  
        int value2 =  
            Int32.Parse(TextBoxNumber2.Text);  
        int sum = value1 + value2;  
        TextBoxSum.Text = sum.ToString();  
    }  
    catch (FormatException)  
    {  
        TextBoxSum.Text = "Invalid!";  
    }  
  
    TextBoxNumber1.SelectAll();  
    TextBoxNumber2.SelectAll();  
}
```

(примерът продължава)

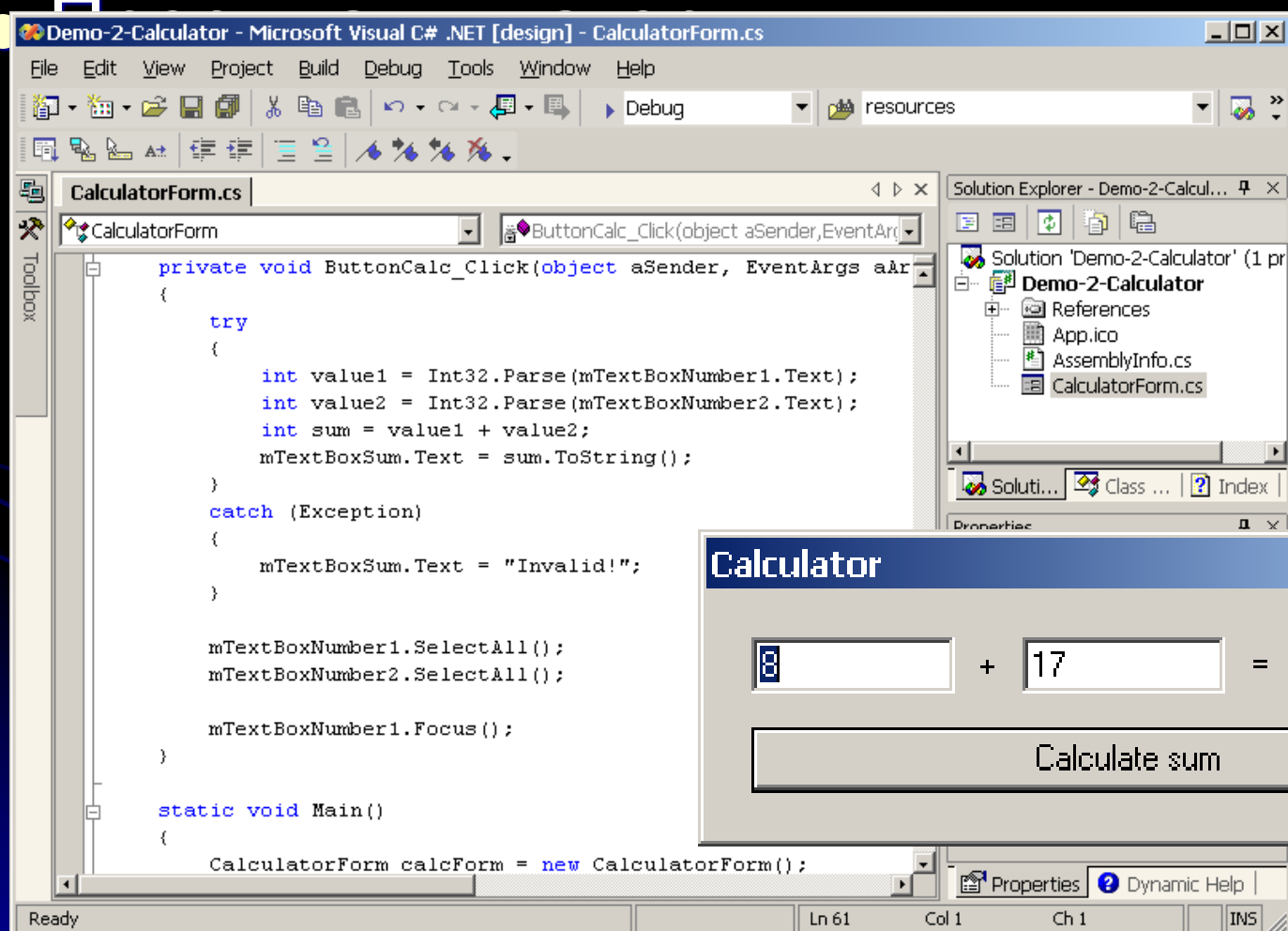
Прост калкулатор – пример

```
        TextBoxNumber1.Focus();  
    }  
  
    static void Main()  
    {  
        CalculatorForm CalcForm = new CalculatorForm();  
        Application.Run(CalcForm);  
    }  
}
```



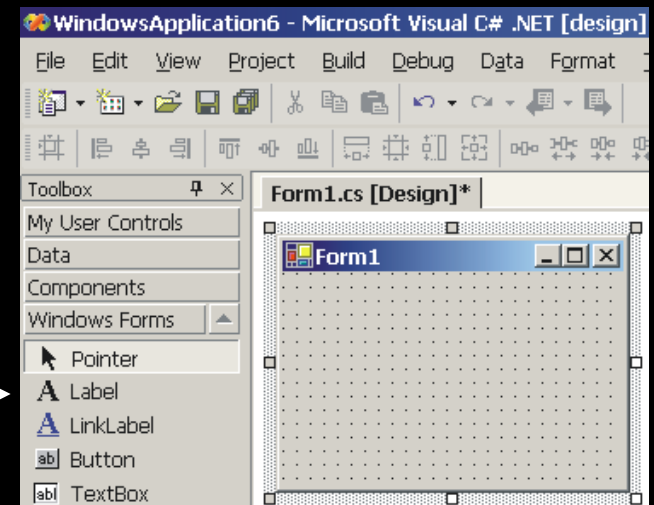
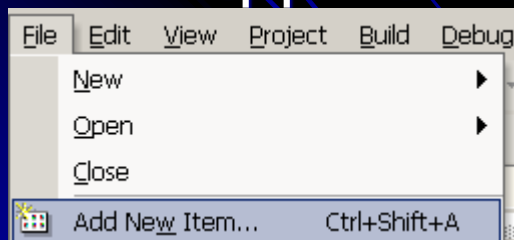
The image shows a screenshot of a simple calculator application window titled "Calculator". The window has a standard Windows-style title bar with a close button (X) in the top right corner. The main area of the window contains three text input fields arranged horizontally, separated by a plus sign (+) and an equals sign (=). The first field contains the number "8", the second field contains "17", and the third field contains "25". Below these fields is a single button labeled "Calculate sum".

Демонстрация

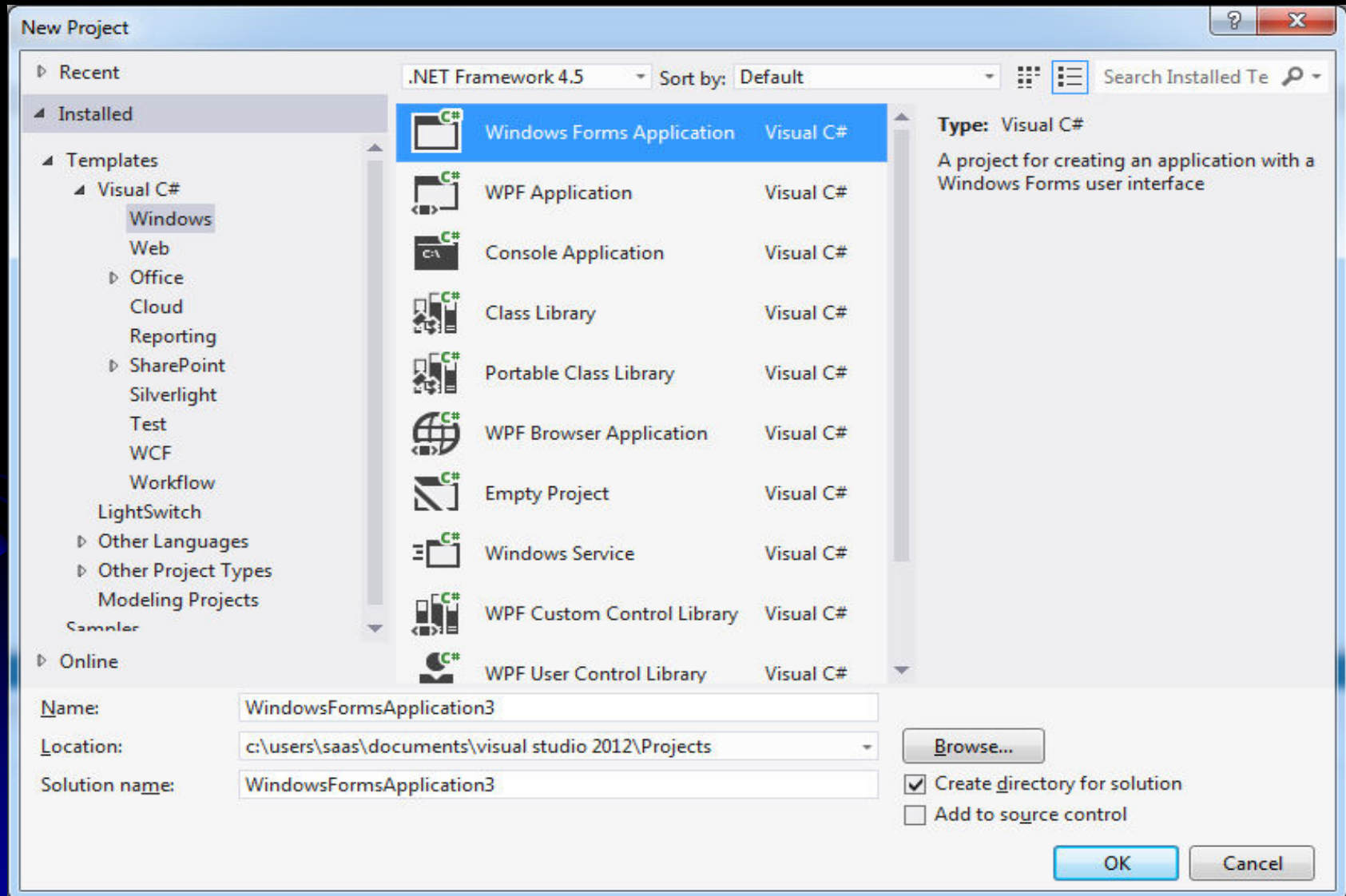


Windows Forms и VS.NET

- Windows Forms редакторът на VS.NET позволява:
 - създаване на форми
 - добавяне на контроли
 - добавяне на неграфични компоненти
 - настройка на свойствата
 - добавяне на събития
- Създаване на форма:

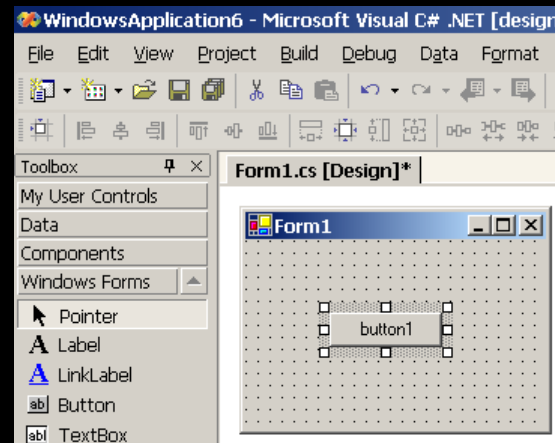
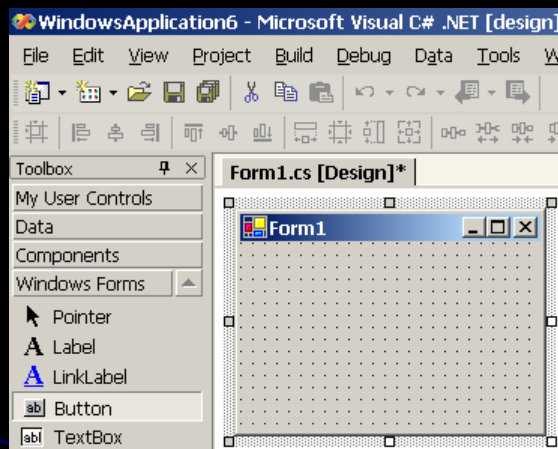


Visual Studio 2012

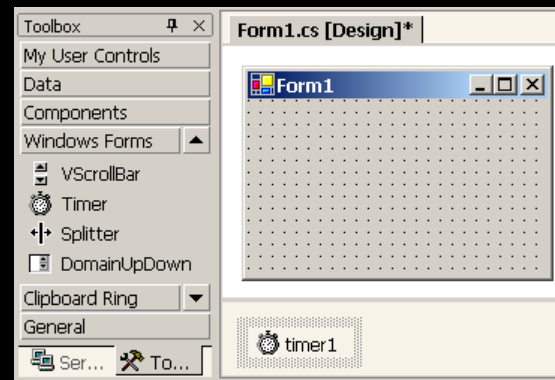
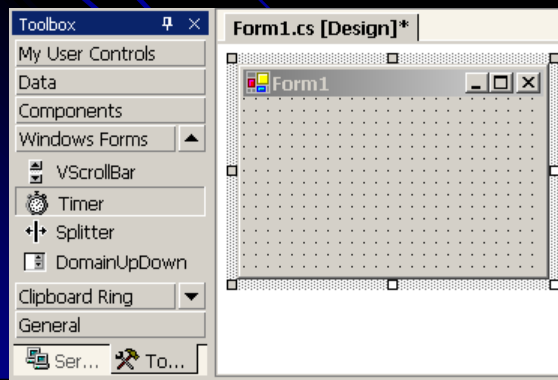


Windows Forms и VS.NET

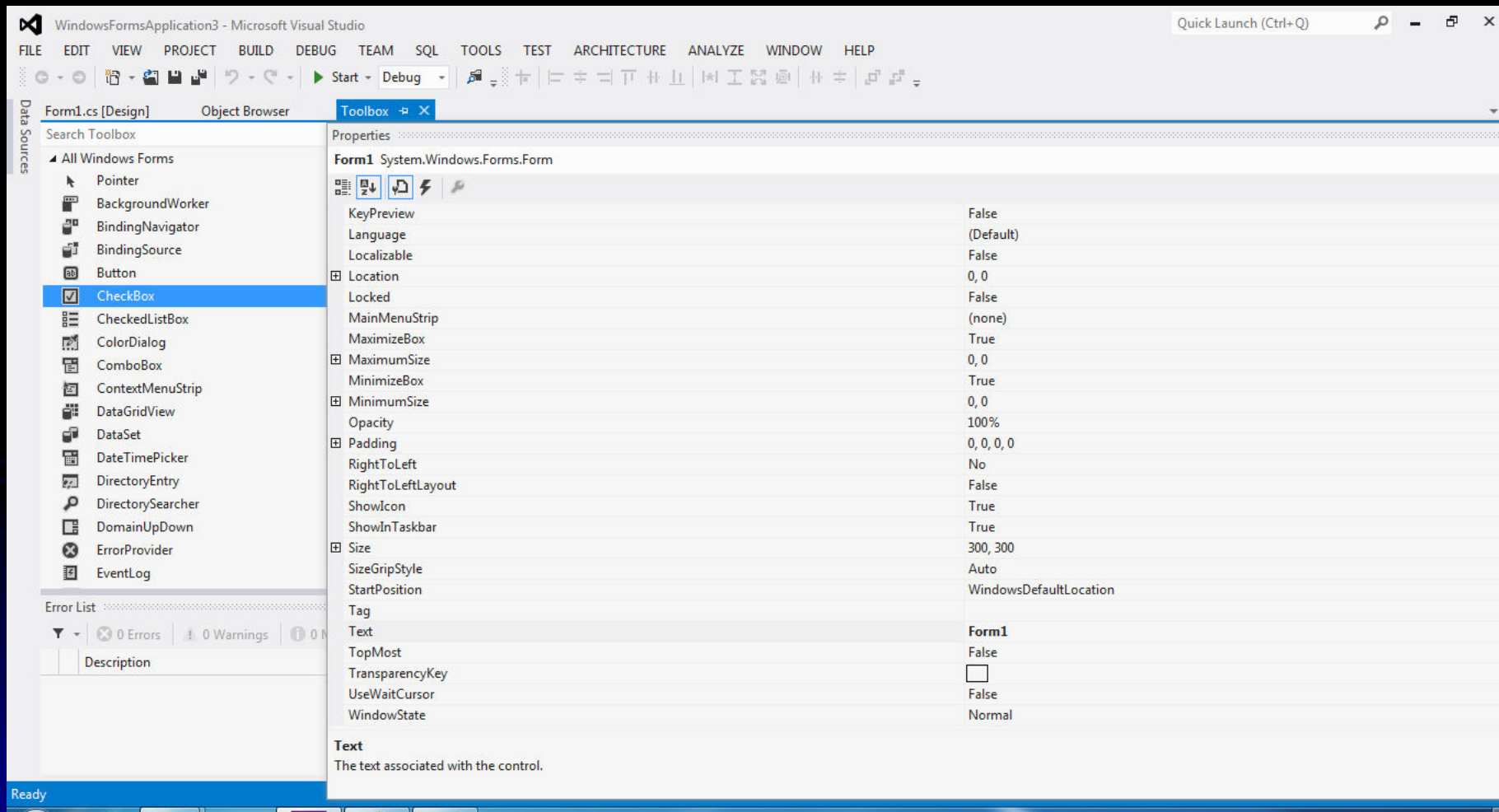
- Добавяне на контрола:



- Добавяне на неграфични компоненти:

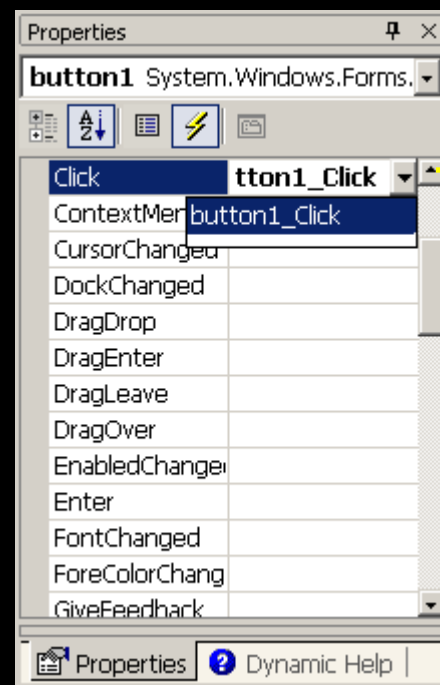
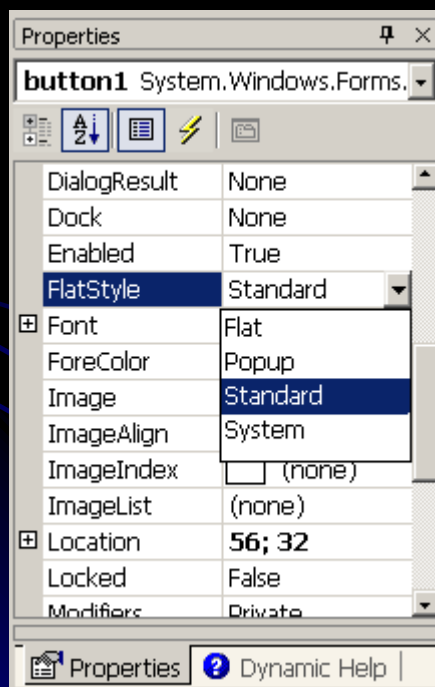


Visual Studio 2012



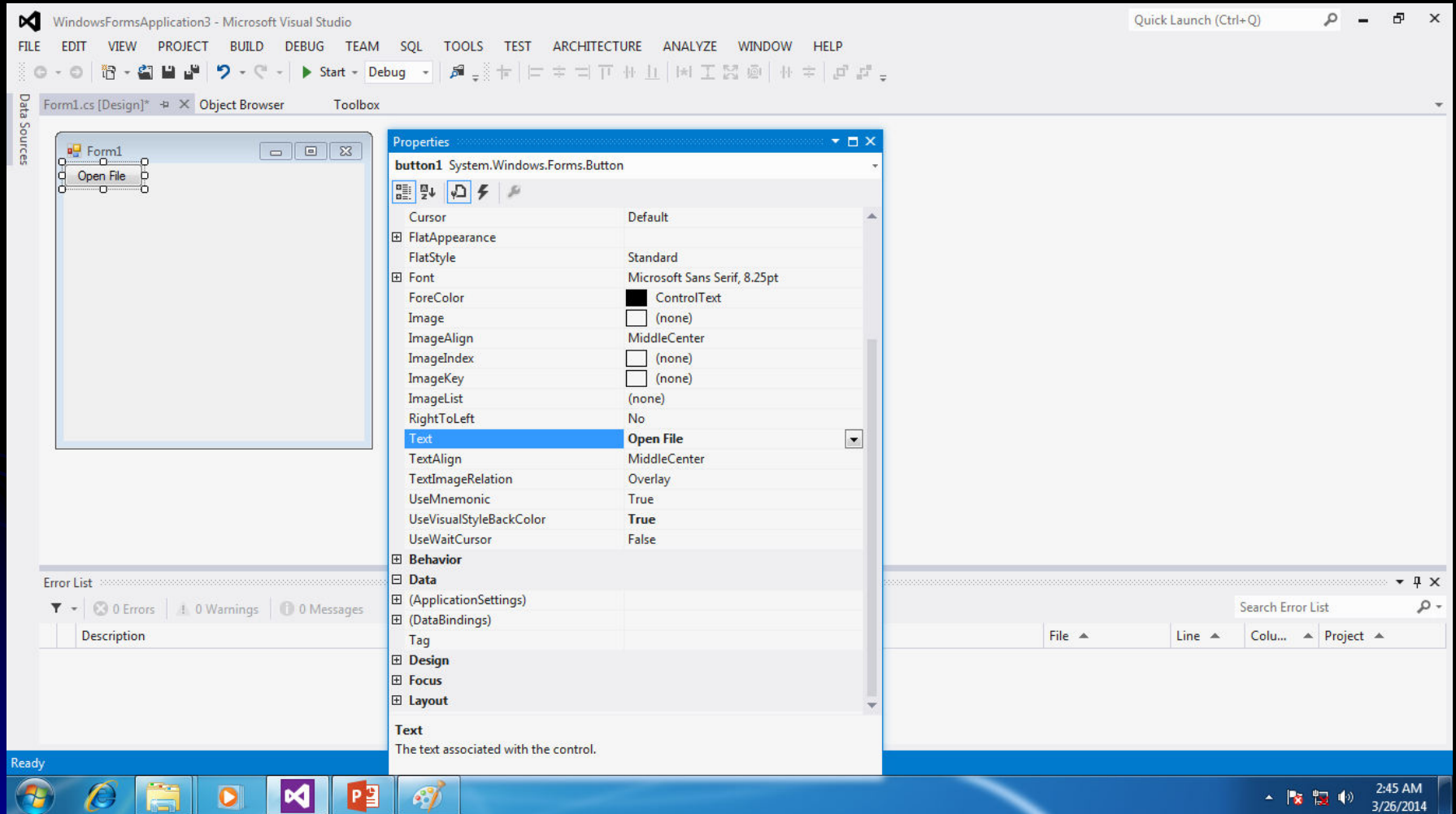
Windows Forms и VS.NET

- Настройка на свойствата:
- Добавяне на обработчици на събития:

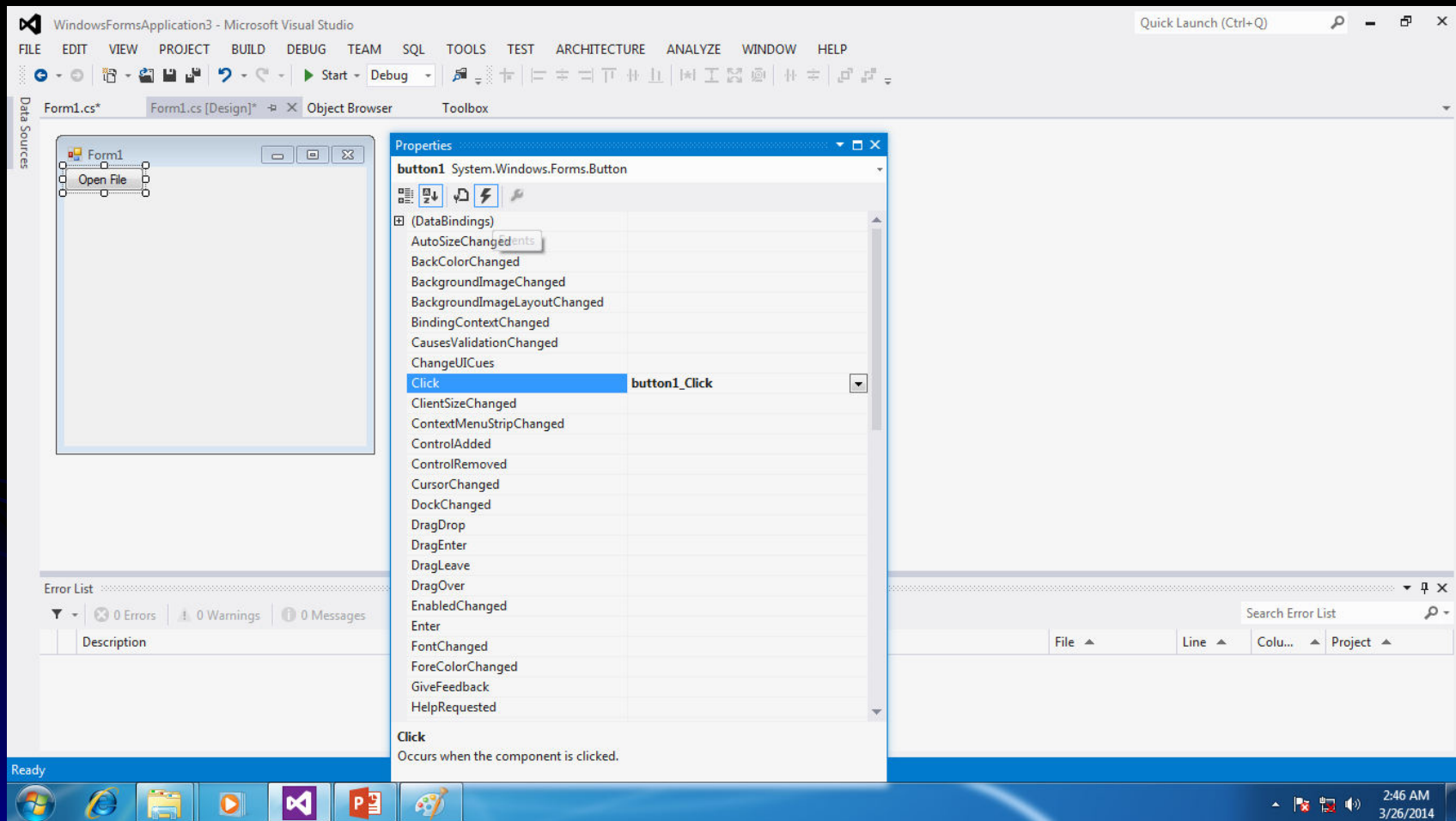


```
private void button1_Click(object sender, System.EventArgs e)
{
    |
}
```

VS 2012

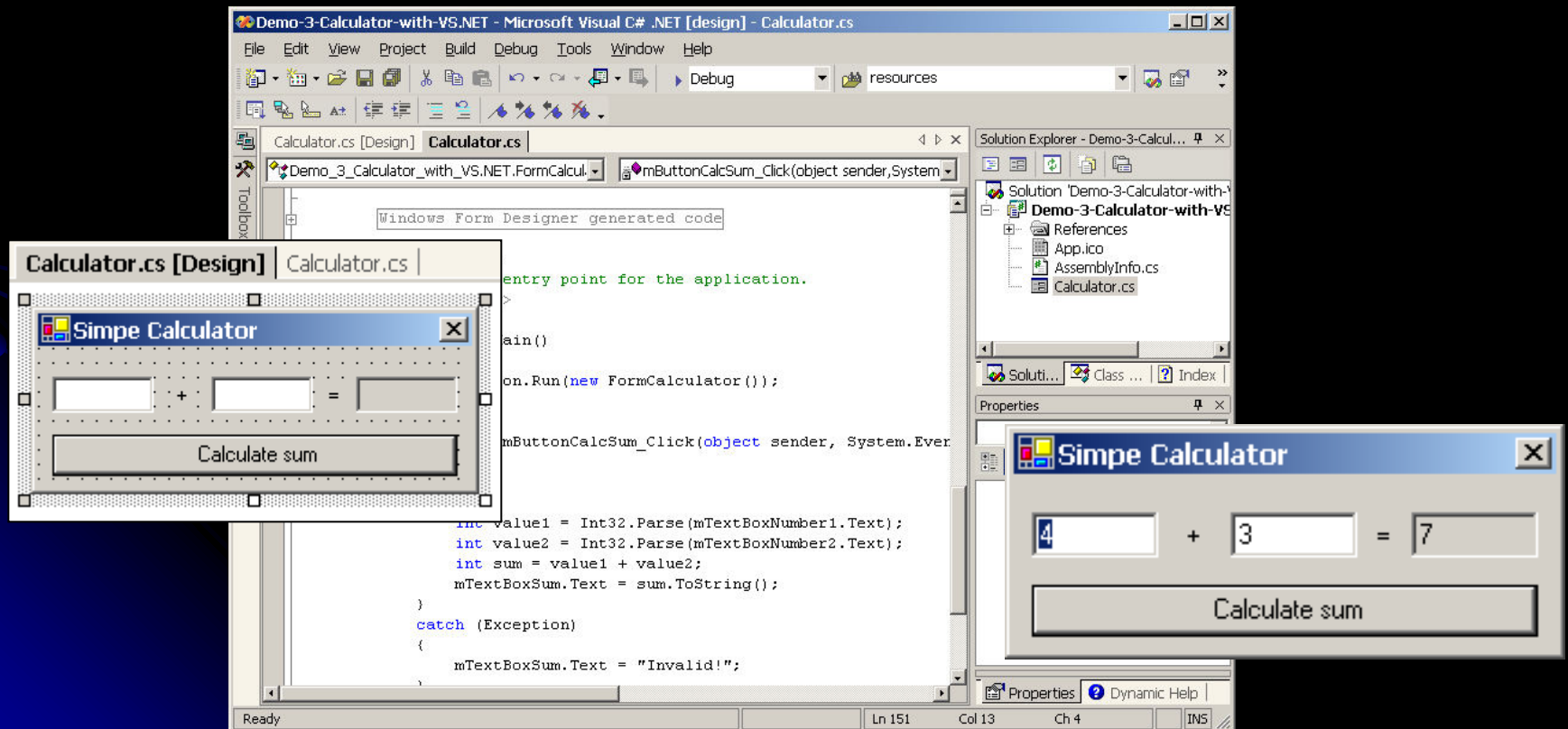


VS 2012



Демонстрация

- Създаване на прост калкулатор с Windows Forms редактора на VS.NET

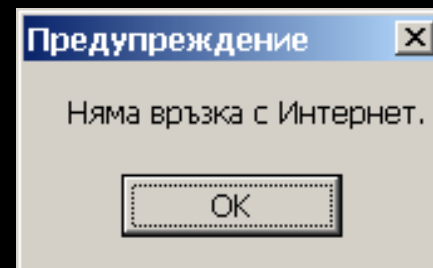


Стандартни диалогови кутии

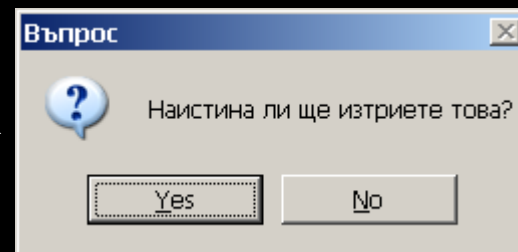
- Класът `MessageBox` позволява извеждане на стандартни диалогови кутии:
 - съобщения към потребителя
 - въпросителни диалози

Примери:

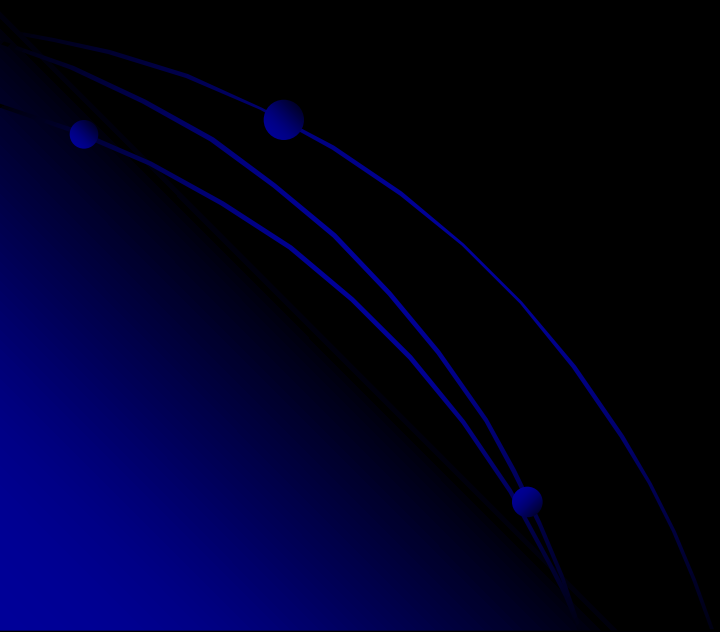
```
MessageBox.Show("Няма връзка с  
Интернет.", "Предупреждение");
```



```
bool confirmed =  
    MessageBox.Show("Наистина ли " +  
        "ще изтриете това?", "Въпрос",  
        MessageBoxButtons.YesNo,  
        MessageBoxIcon.Question) ==  
        DialogResult.Yes;
```



Windows Forms - GUI

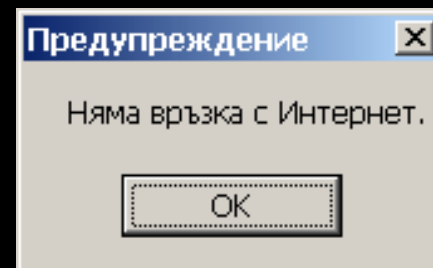


Стандартни диалогови кутии

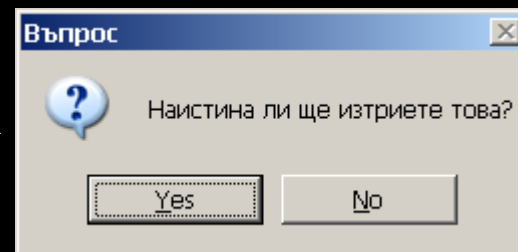
- Класът `MessageBox` позволява извеждане на стандартни диалогови кутии:
 - съобщения към потребителя
 - въпросителни диалози

Примери:

```
MessageBox.Show("Няма връзка с  
Интернет.", "Предупреждение");
```



```
bool confirmed =  
    MessageBox.Show("Наистина ли " +  
        "ще изтриете това?", "Въпрос",  
        MessageBoxButtons.YesNo,  
        MessageBoxIcon.Question) ==  
        DialogResult.Yes;
```



Извикване на диалогови кутии

- Потребителските диалогови кутии се извикват така:

```
DialogResult result = dialog.ShowDialog();
```

- ShowDialog() показва модално диалога
- Типът DialogResult съдържа резултата (OK, Yes, No, Cancel и др.)
- Задаване на DialogResult:
 - Автоматично – чрез свойството DialogResult на бутоните
 - Ръчно – преди затваряне на диалога чрез свойството му DialogResult

Диалогови кутии – пример

В MainForm класа:

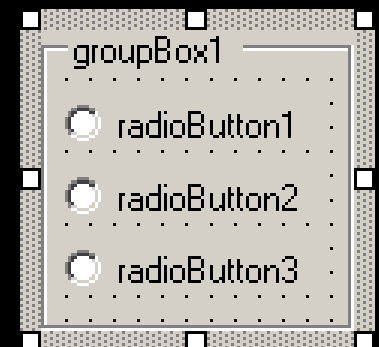
```
private void ButtonCallDialog_Click(  
    object sender, System.EventArgs e)  
{  
    DialogForm dialog = new DialogForm();  
    if (dialog.ShowDialog() == DialogResult.OK)  
    {  
        string userName = dialog.UserName;  
        MessageBox.Show("You entered: " + userName);  
    }  
    else  
    {  
        MessageBox.Show("You canceled the dialog.");  
    }  
}
```

В DialogForm класа:

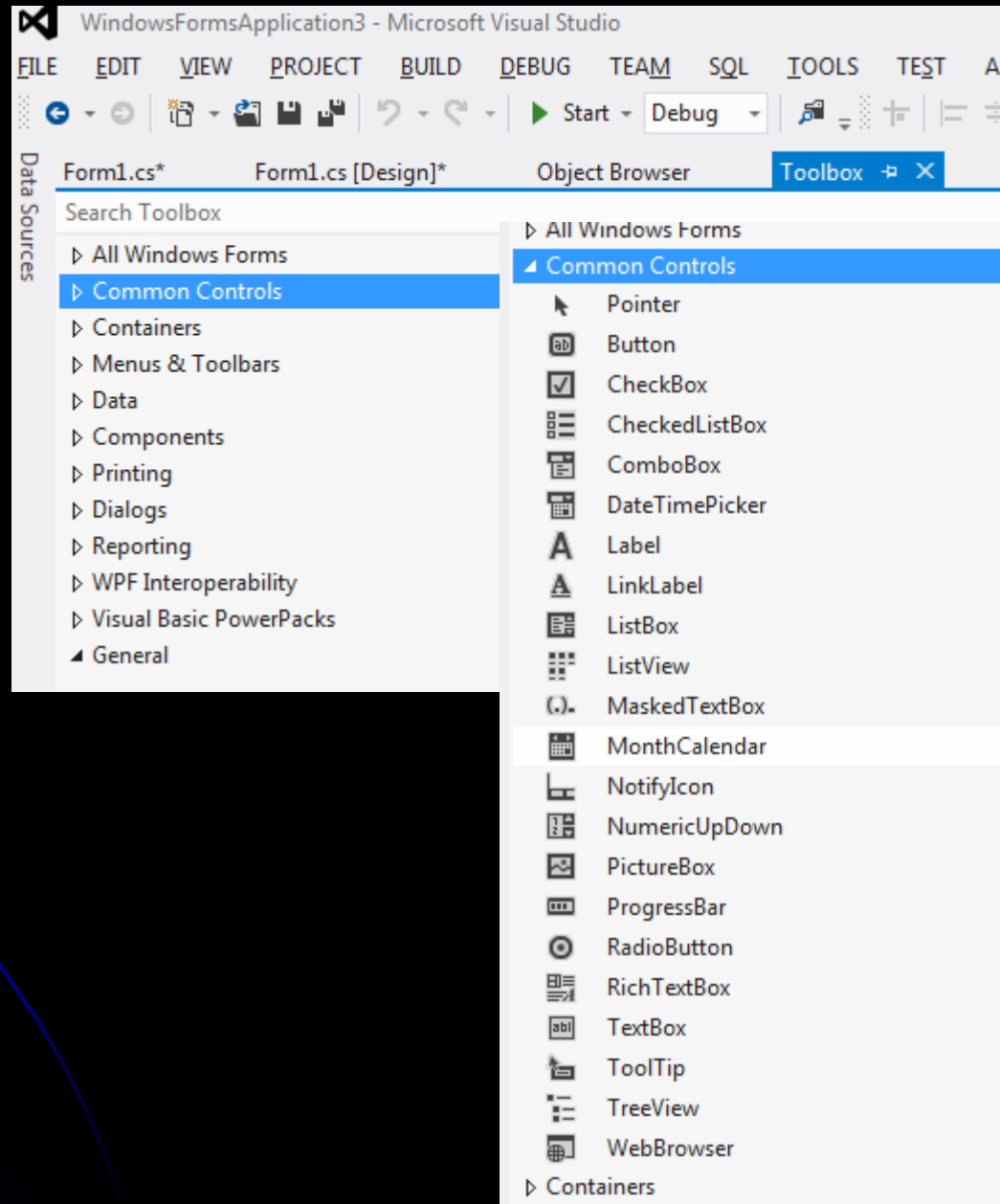
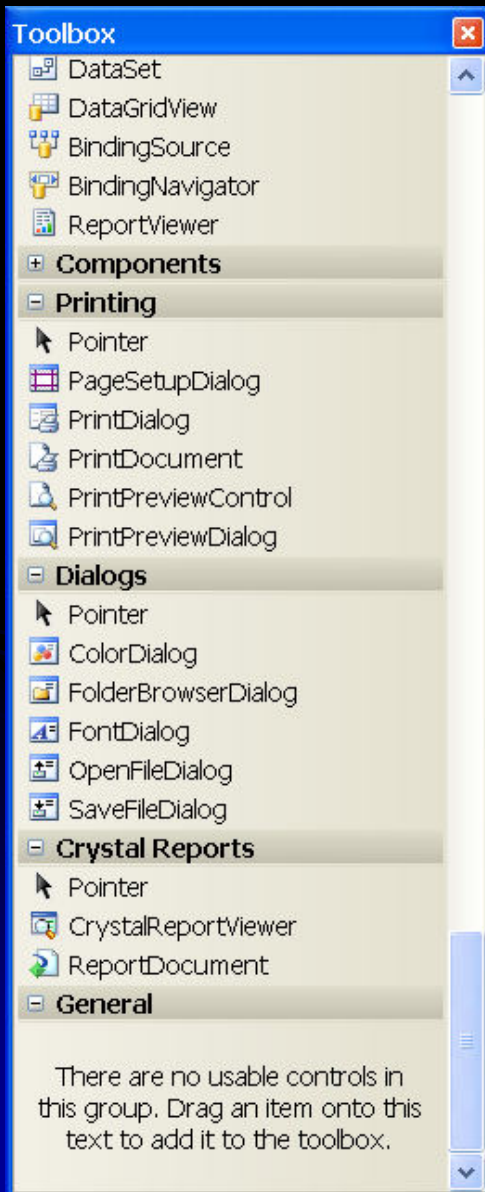
```
public string UserName {  
    get { return TextBoxName.Text; }  
}
```

Други Windows Forms контроли

- CheckBox –  – кутия за избор в стил "да/не". По-важни свойства:
 - Checked – задава дали е избрана
- RadioButton –  – контрола за алтернативен избор
 - Checked – задава дали е избрана
 - CheckedChanged – активира се при промяна на Checked свойството
- Използва се в групи:
 - Само един RadioButton е избран в даден момент
 - Поставят се в GroupBox, Panel или TabPage



Toolbox and Controls



Design Time

Form1

Open Dialog -img

Font Dialog

Color Dialog

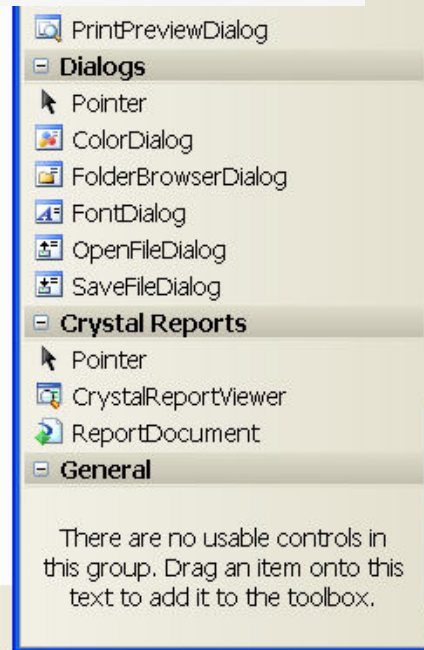
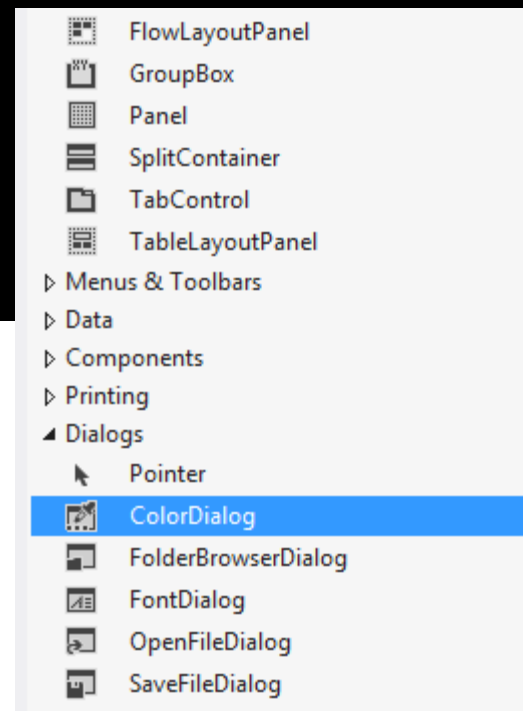
Save Dialog

Open Form

Open Dialog - txt

text from form2...

Close



colorDialog1

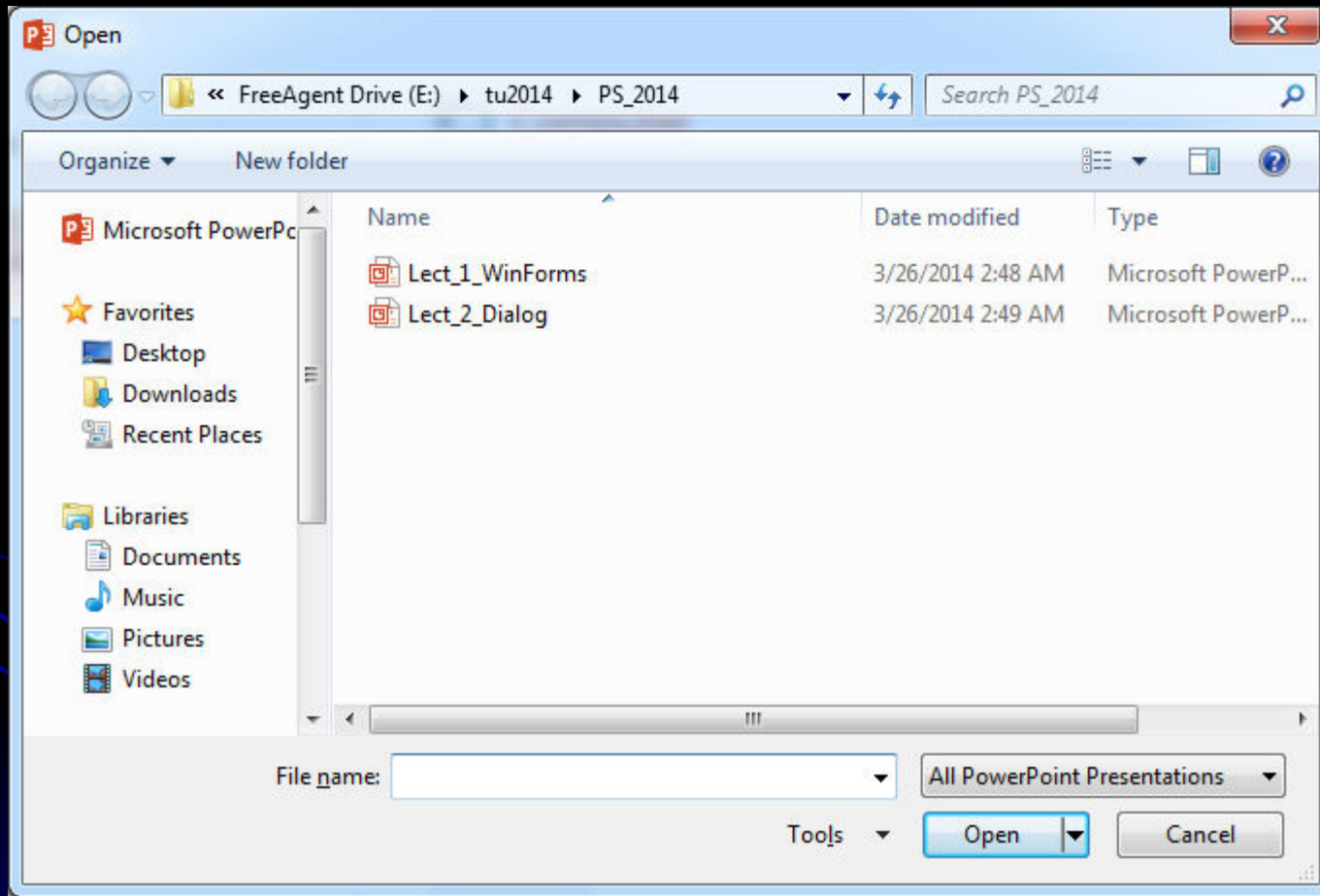
folderBrowserDialog1

dlgFont

dlgOpenFile

dlgSaveFile

Open Dialog



Code – Open Dialog

```
dlgResult = dlgOpenFile.ShowDialog();
```

```
if (dlgResult == DialogResult.Cancel)  
    return;
```

```
try  
{
```

```
    m_strFileName = dlgOpenFile.FileName;
```

```
    StreamReader sr = new StreamReader(m_strFileName);
```

```
    txtBody.Text = sr.ReadToEnd();
```

```
    sr.Close();
```

```
    m_bModified = false;
```

```
}
```

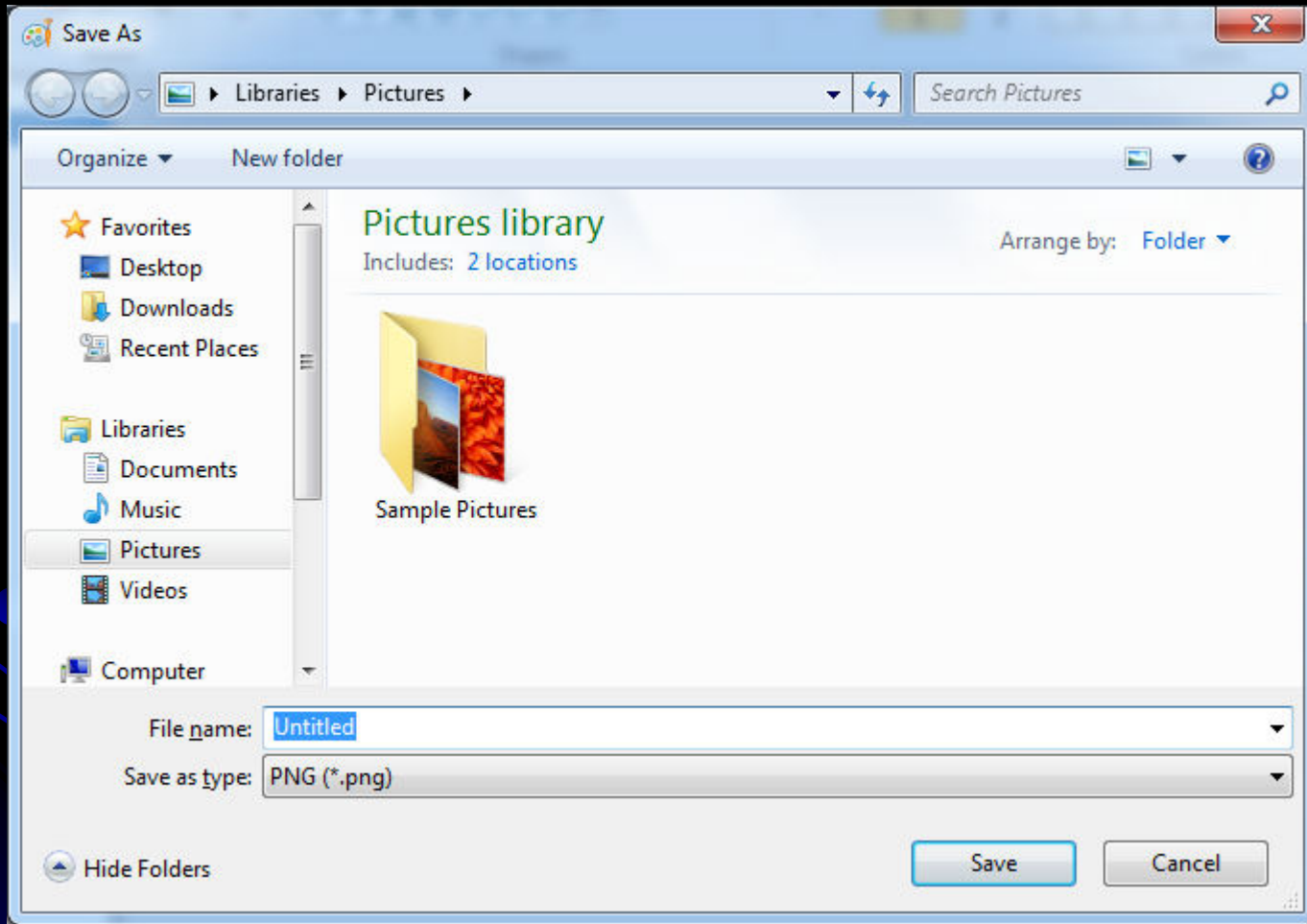
```
catch (Exception err)
```

```
{
```

```
    MessageBox.Show(err.Message, "Error", MessageBoxButtons.OK,  
    MessageBoxIcon.Error);
```

```
}
```


Save Dialog



Code – Save Dialog

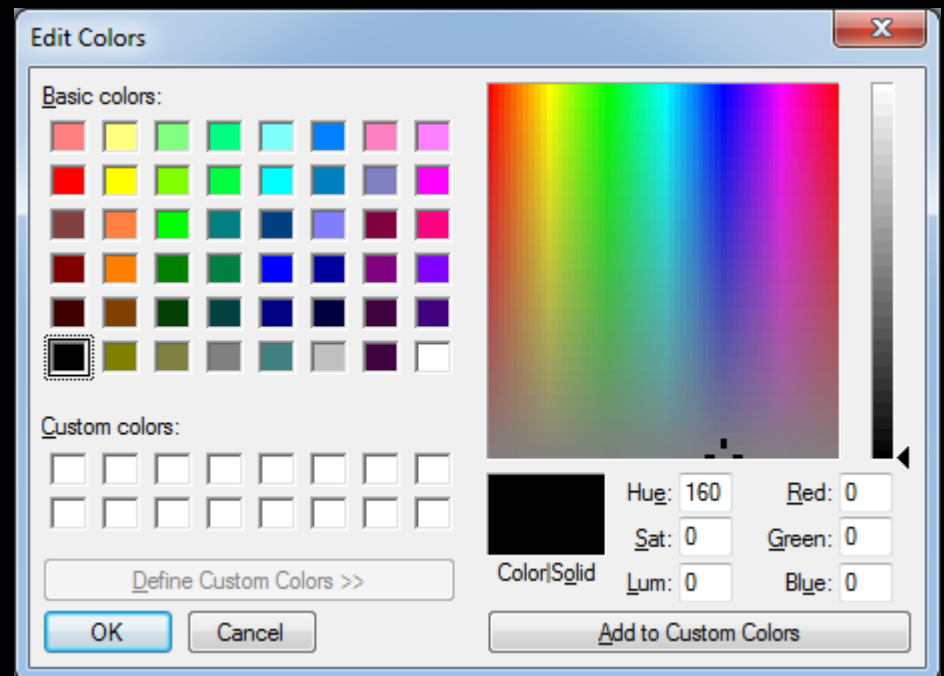
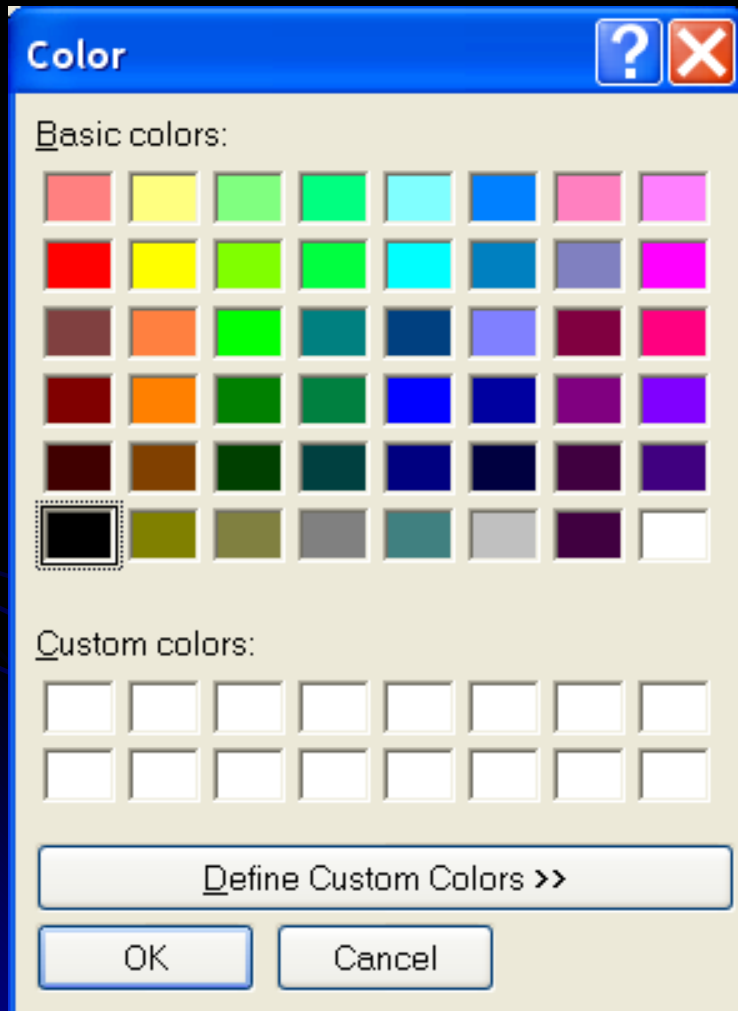
```
dlgSaveFile.FileName = "mydoc" + m_intDocNumber.ToString() + ".txt";
dlgResult = dlgSaveFile.ShowDialog();

if (dlgResult == DialogResult.Cancel)
    return;

try
{
    m_strFileName = dlgSaveFile.FileName;
    m_sw = new StreamWriter(m_strFileName);
    m_sw.Write(txtBody.Text);
    m_sw.Close();

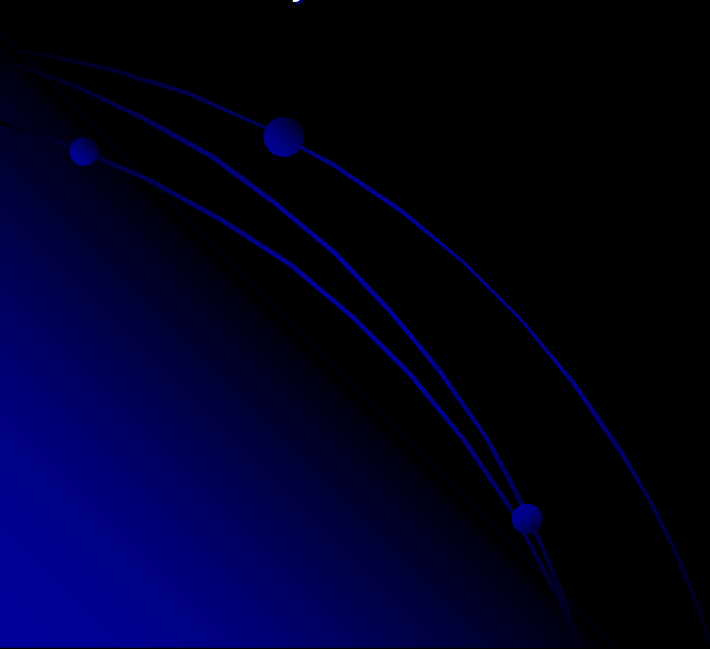
    m_intDocNumber++;
    m_bModified = false;
    this.Text = "C# Simple Notepad: " + m_strFileName;
}
catch (Exception err)
{
    MessageBox.Show(err.Message, "Error", MessageBoxButtons.OK,
        MessageBoxIcon.Error);
}
```

Color Dialog

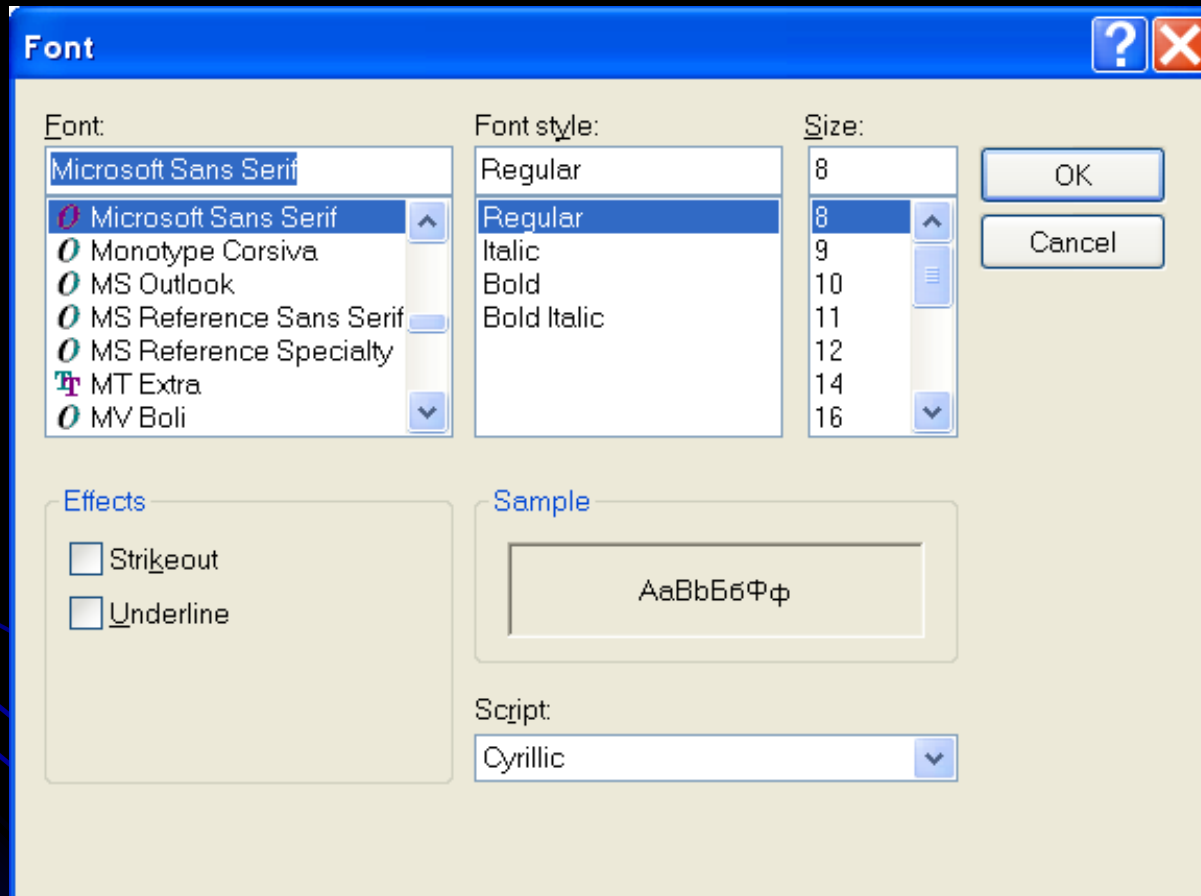


Code – Color Dialog

```
if (colorDialog1.ShowDialog() == DialogResult.OK)
{
    txtBody.BackColor = colorDialog1.Color;
}
```



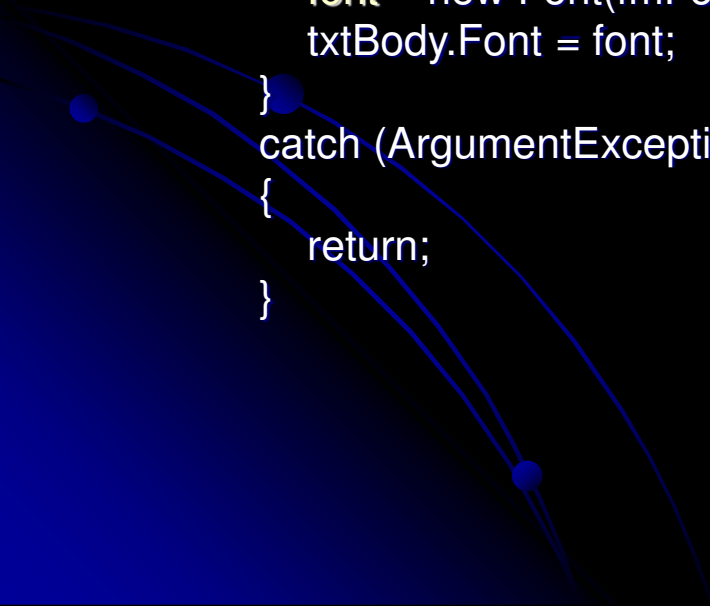
Font Dialog



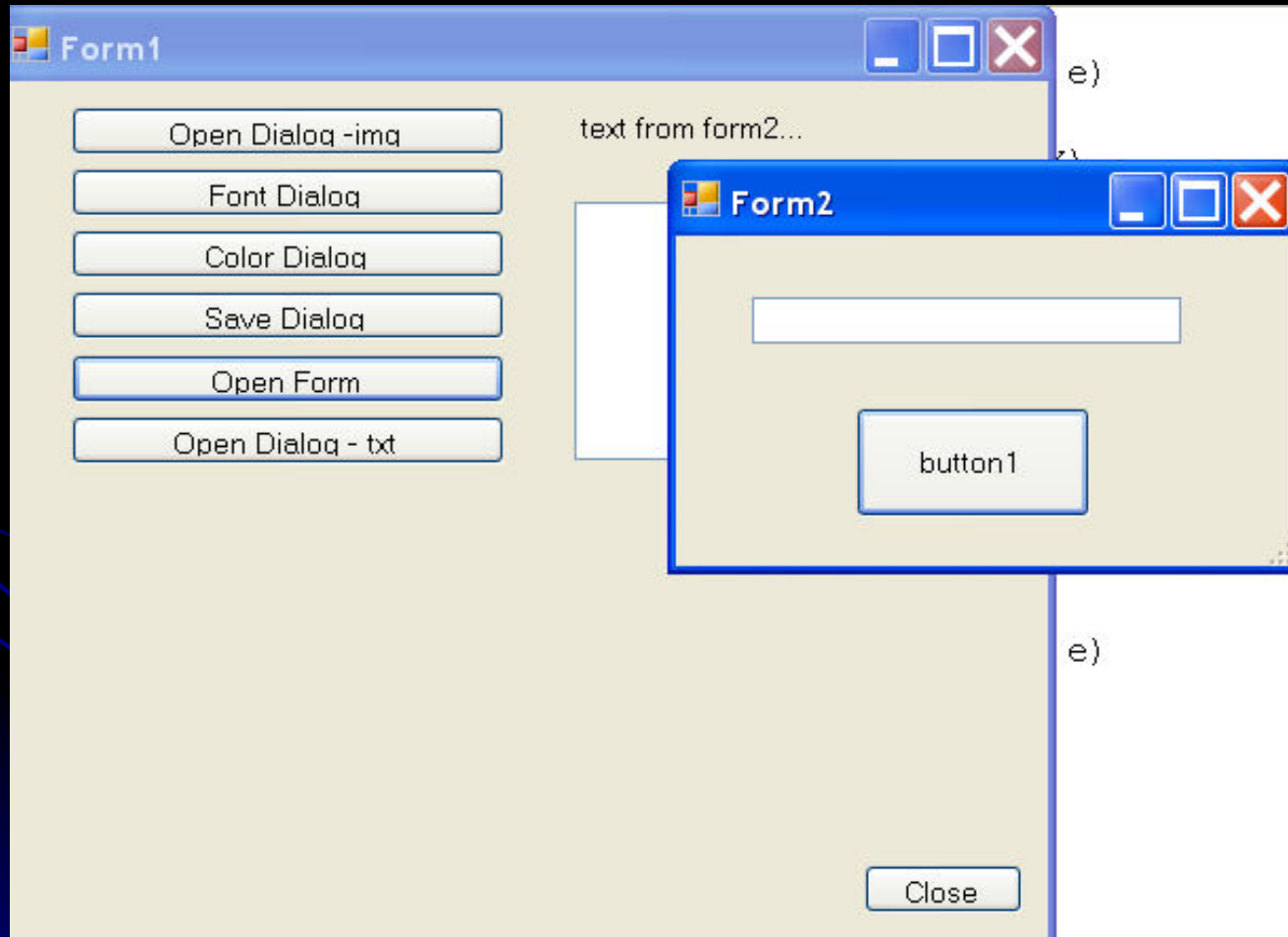
Code – Font Dialog

```
if (dlgFont.ShowDialog() == DialogResult.Cancel)
    return;
```

```
    FontFamily fmFontName = dlgFont.Font.FontFamily;
    float fFontSize = dlgFont.Font.Size;
    FontStyle fsStyle = dlgFont.Font.Style;
    Font font;
    try
    {
        font = new Font(fmFontName, fFontSize, txtBody.Font.Style ^ fsStyle);
        txtBody.Font = font;
    }
    catch (ArgumentException)
    {
        return;
    }
```



Open New Form

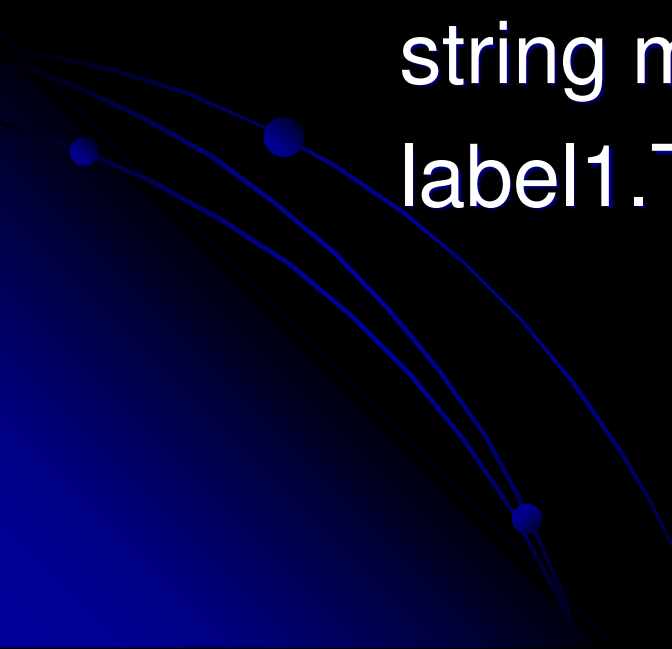


Code – Show Form

```
Form2 frmNew = new Form2();  
frmNew.ShowDialog();
```

(parameter from Form2 to Form1)

```
string myString = frmNew.MyString;  
label1.Text = myString;
```



Form 2

// parameter

- public partial class Form2 : Form
{

```
private string _MyString;  
public string MyString  
{  
    get { return _MyString; }  
}
```

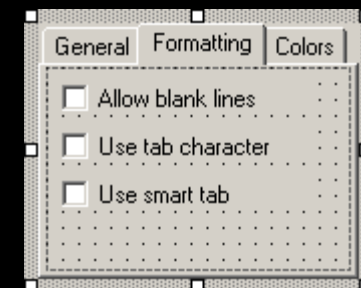
.....

// close form

```
_MyString = textBox1.Text;  
Close ();  
// Application.Exit(); //- Close Application
```

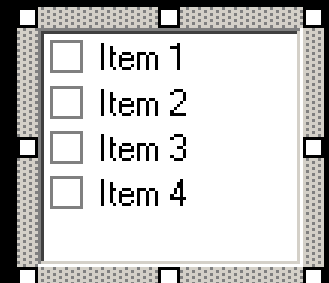
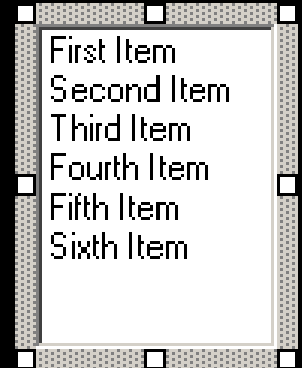
Други Windows Forms контроли

- PictureBox – изобразява картинки. По-важни свойства:
 - Image – задава картинката
 - SizeMode – задава дали картинката да се разшири / намали или центрира
- Panel – представлява контейнер, който съдържа група други контроли
- TabControl, TabPage – осигуряват ползване на табове



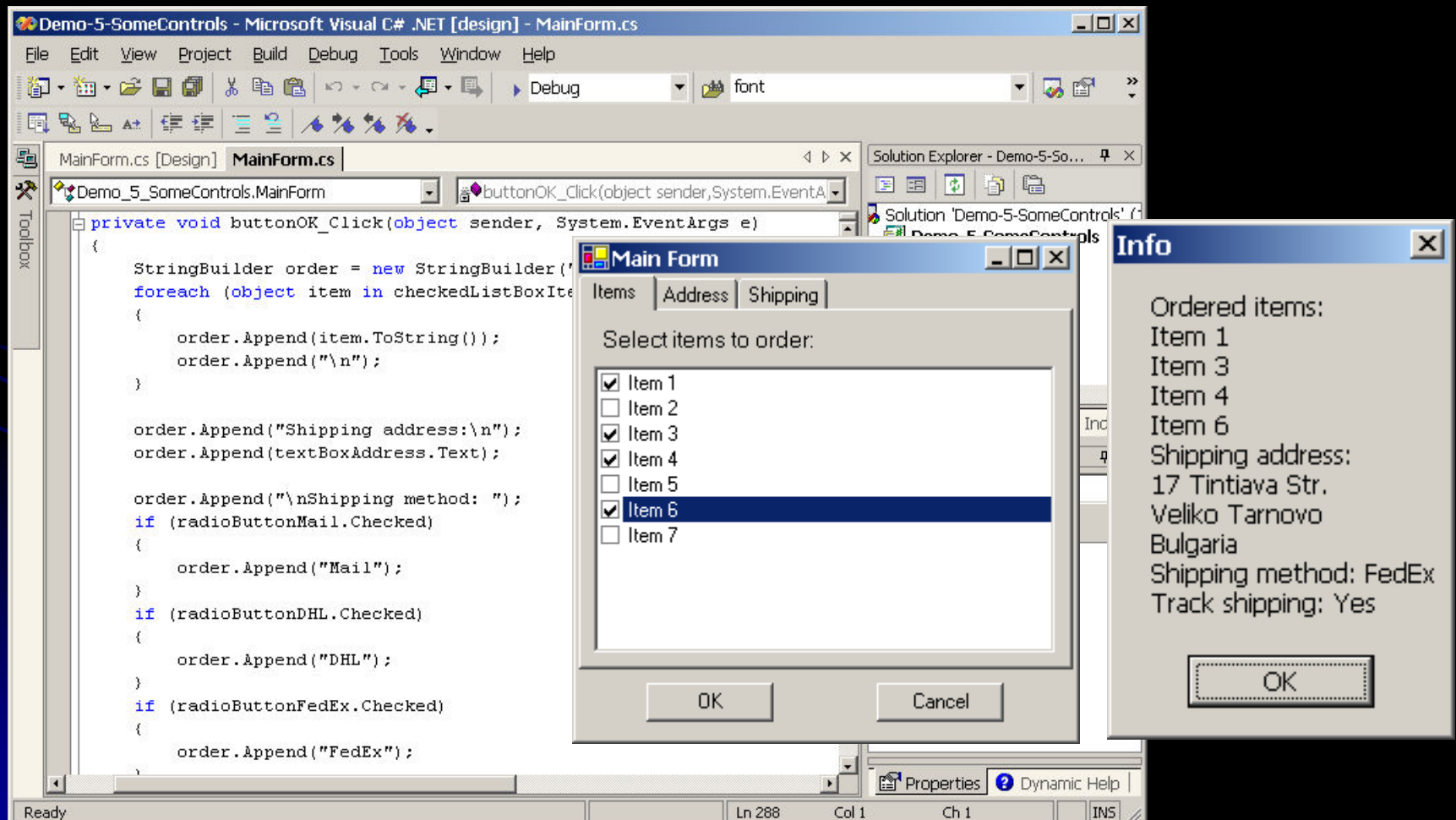
Други Windows Forms контроли

- **ListBox** – изобразява списък с низове. По-важни свойства:
 - **Items** – задава списъка
 - **SelectionMode** – разрешава / забранява селектиране на няколко елемента едновременно
 - **SelectedIndex**, **SelectedIndices**, **SelectedItem** – връща избрания елемент (или избраните елементи)
- **CheckedListBox** – списък от възможности за избор "да/не":
 - **Items** – задава възможностите
 - **CheckedItems** – връща избраните



Демонстрация #5

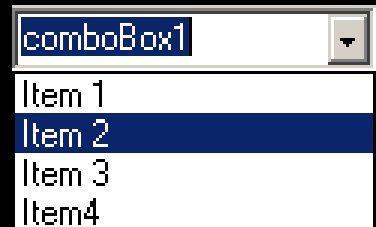
● Работа с някои контроли



Други Windows Forms контроли

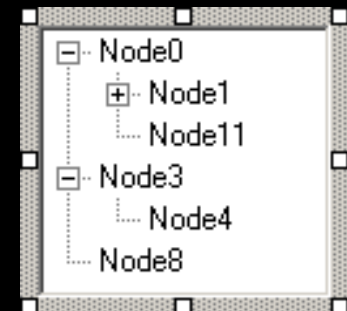
- ComboBox – кутия за редакция на текст с възможност за drop-down алтернативен избор. По-важни свойства:

- Text – въведения текст
- Items – възможни стойности
- DropDownStyle – задава стил – дали само се избира стойност от списъка или може да се въвежда друга стойност



- TreeView – изобразява дървовидни данни. Основни свойства:

- Nodes – съдържа дървото (списък от TreeNode обекти)
- SelectedNode – избрания възел



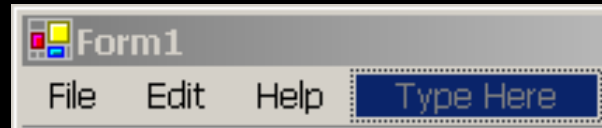
Други Windows Forms контроли

- LinkLabel –  – препратка (hyperlink):
 - Text – съдържание на връзката
 - LinkClicked – активира се при щракване върху препратката
- RichTextBox – кутия за редакция на текст с форматиране (Rich Text Format)
 - LoadFile, SaveFile – зарежда/записва RTF
 - SelectionFont, SelectionColor, SelectionAlignment – задават шрифт, цвят и подравняване на избрания текст



Менюта

- MainMenu – падащо меню



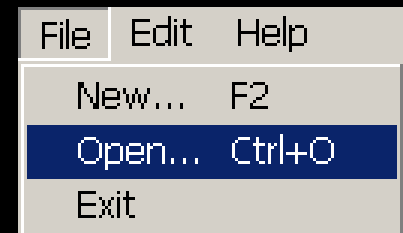
- Съдържа списък от MenuItem елементи
- MenuItem – елемент от меню

- Text – заглавие на елемента, например “&New” или “Op&en...” или “–”

- ShortCut – кратък клавиш

- Click – събитие “избиране”

- ContextMenu – контекстно меню (появяващо се меню)



- Съдържа списък от MenuItem елементи

Ленти с инструменти

- **ToolBar** – лента с инструменти (с бутони) – 
 - **Buttons** – съдържа списък от **ToolBarButton** елементи
 - **ImageList** – задава картинките за бутоните
 - **ButtonClick** – активира се при натискане на бутон
 - подава се **ToolBarButtonClickEventArgs** с информация кой бутон е бил натиснат
- **ToolBarButton** – бутон в лентата
- **ImageList** – списък с картинки

Статус ленти

- `StatusBar` –  – лента за състоянието
 - `Panels` – съдържа секциите на лентата
 - `ShowPanels` – включва / изключва показване на панелите
- `StatusBarPanel` – секция в лентата
 - `Text` – съдържание на панела
 - `Icon` – икона на панела

Диалог за избор на файл

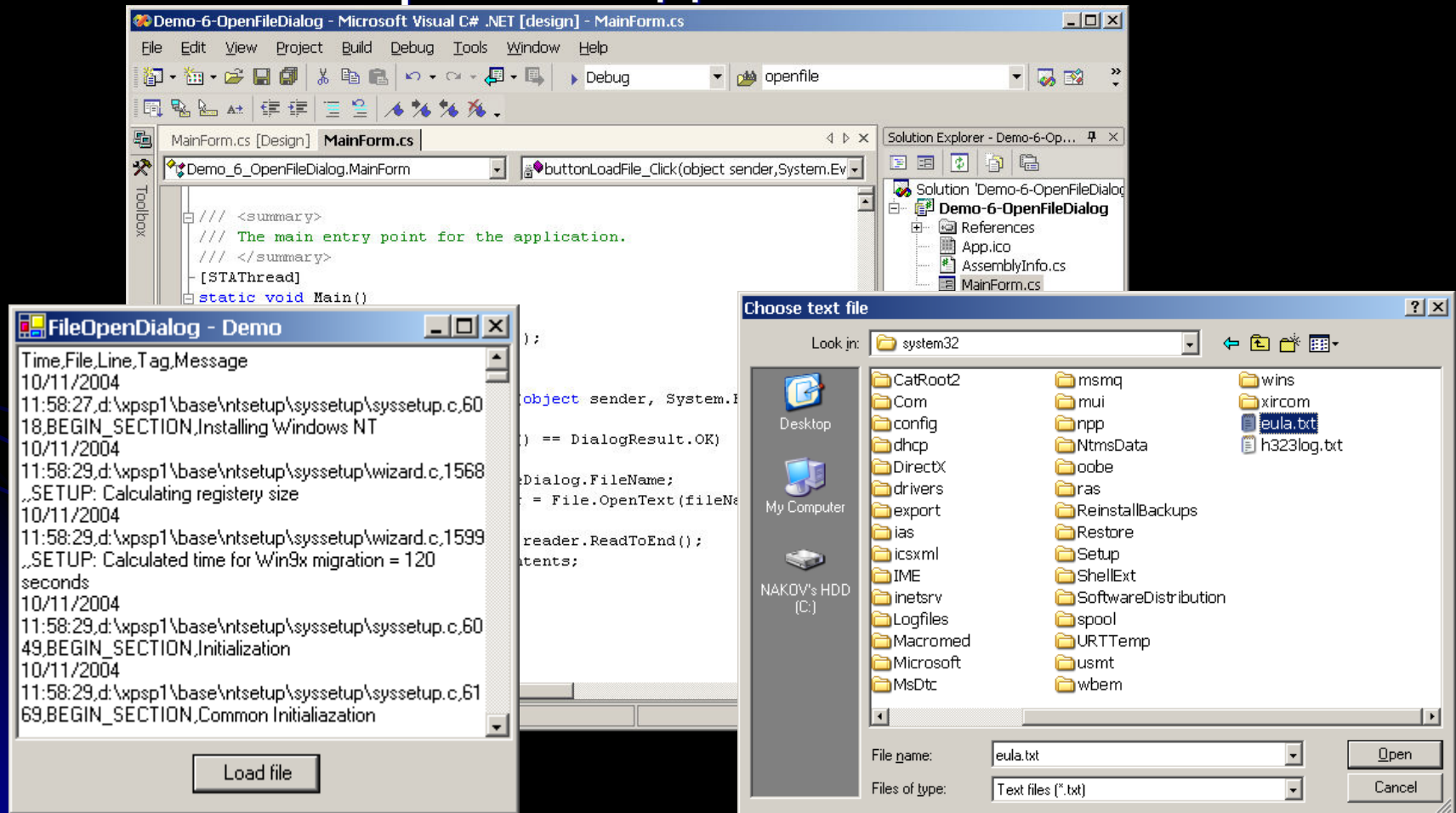
- OpenFileDialog – диалог за избор на файл (при отваряне)
 - Title – заглавие на диалога
 - InitialDirectory – начална директория
 - Filter – задава възможните файлови разширения
 - FilterIndex – задава активния филтър
 - Multiselect – избор на много файлове
 - FileName, FileNames – избрания файл
- SaveFileDialog – диалог за избор на файл (при записване)

Файлов диалог – пример

```
private void ButtonLoadFile_Click(  
    object sender, System.EventArgs e)  
{  
    OpenFileDialog openFileDialog = new OpenFileDialog();  
    openFileDialog.Filter = "Text files (*.txt)|*.txt|" +  
        "Log files (*.log)|*.log";  
    openFileDialog.Title = "Choose text file";  
  
    if (openFileDialog.ShowDialog() == DialogResult.OK)  
    {  
        string fileName = openFileDialog.FileName;  
        using (StreamReader reader =  
            File.OpenText(fileName))  
        {  
            string fileContents = reader.ReadToEnd();  
            textBox.Text = fileContents;  
        }  
    }  
}
```

Демонстрация #6

- Работа с файлов диалог

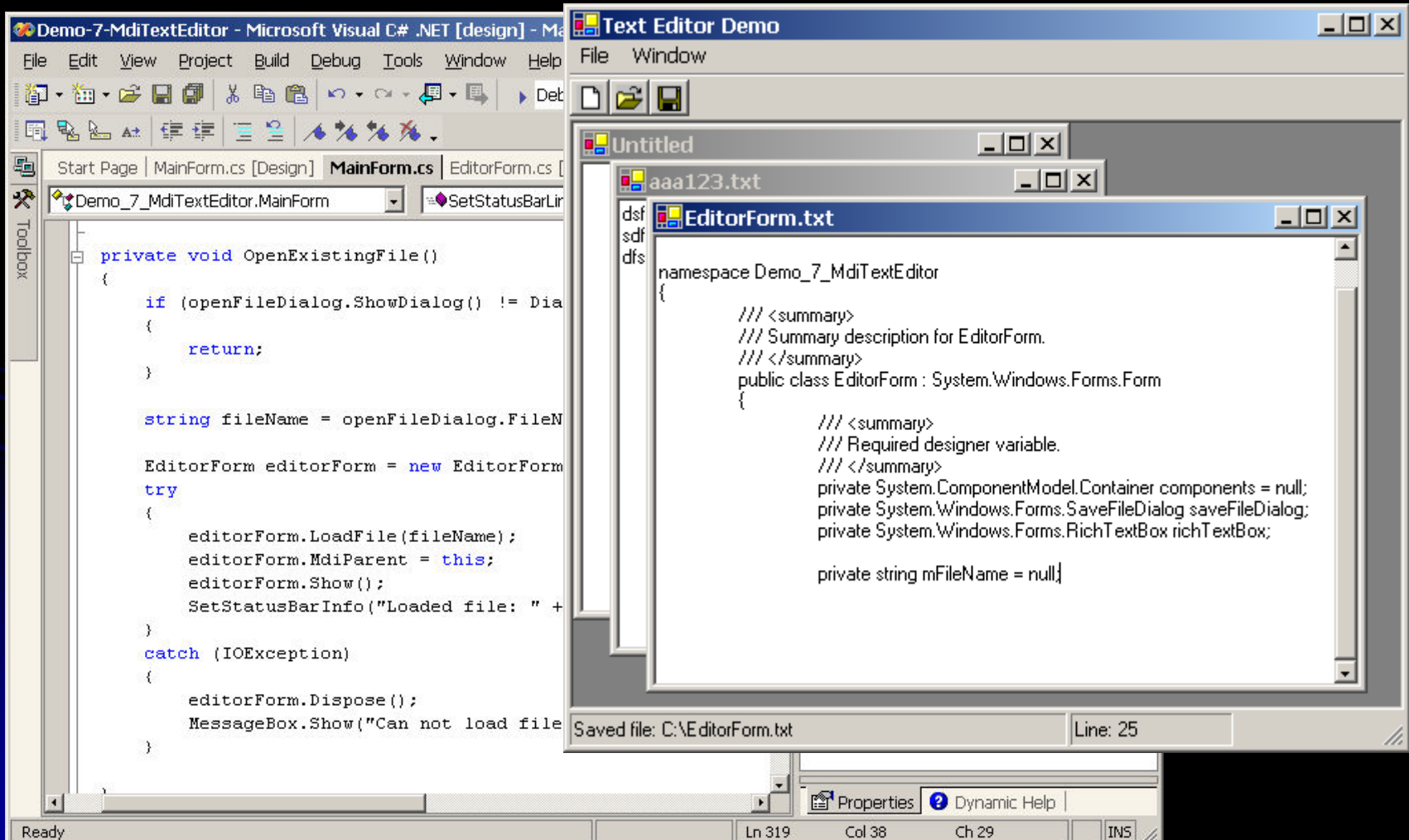


MDI приложения

- MDI (Multiple Document Interface)
приложенията поддържат работа с
няколко документа едновременно
- MDI контейнери (MDI parents)
 - Съдържат други форми
 - Задава им се `IsMdiContainer = true`
 - Обикновено имат меню Window за смяна на
активната форма (`MdiList = true`)
- MDI формите (MDI children)
 - Съдържат се в контейнер-формата
 - Задава им се `MdiParent = <контейнер>`

Демонстрация #7

- Мултидокументов текстов редактор



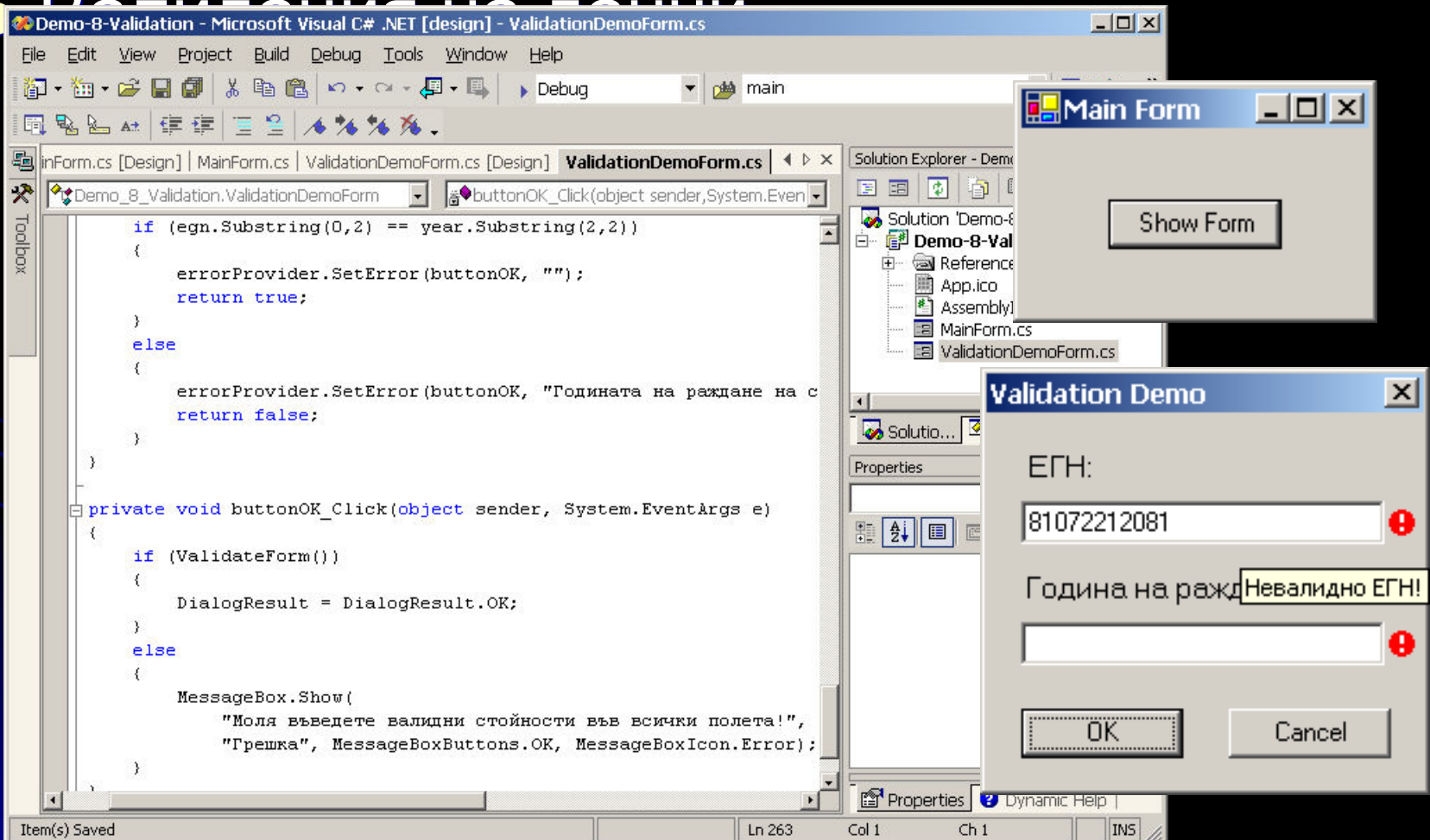
Валидация на данни

- Валидацията е необходима, когато в дадена контрола трябва да се допуска въвеждане само на коректни данни
- В Windows Forms има стандартни средства за валидация:
 - `Validating` – събитие за валидация на данните в класа `Control`
 - Подава се параметър от тип `CancelEventArgs`
 - Ако се зададе `Cancel=true` – не пуска потребителя да напусне контролата
 - `ErrorProvider` – отбелязва графично

Валидация на данни – пример

```
private TextBox TextBox1;  
private ErrorProvider errorProvider;  
...  
  
private void TextBox1_Validating(object sender,  
    System.ComponentModel.CancelEventArgs e)  
{  
    try  
    {  
        Int32.Parse(TextBox1.Text);  
        errorProvider.SetError(TextBox1, "");  
    }  
    catch (FormatException)  
    {  
        errorProvider.SetError(  
            TextBox1, "Integer number expected!");  
        e.Cancel = true;  
    }  
}
```


Демонстрация #8



Свързване на данни

- **Свързването на данните (Data Binding)**
 - Осигурява автоматично прехвърляне на данни между контроли и източници на данни
 - Например: свързване на масив съдържащ имена на градове с ComboBox контрола
- **Източници на данни**
 - IList – масиви и колекции
 - IBindingList – поддържа се от DataView
 - поддържа нотификация за промяна
- **Контроли, поддържащи data binding**
 - Всички Windows Forms контроли
 - TextBox, ComboBox, ListBox, DataGrid

Свързване на данни

- Видове свързване

- Просто свързване (simple binding)

- Свързване на контрола с единичен обект
 - Свързване на контрола с единичен (текущ) елемент от списък
 - Например: `TextBox` и `CheckBox`

- Сложно свързване (complex binding)

- Свързване на списъчна контрола със списък
 - Например: `ListBox`, `ComboBox`, `DataGrid`
 - Поддържа се текущо избран елемент (активен) от списъка

Свързването е едноточностно!

- Промяна на дадено свързано свойство от дадена контрола променя данните в източника, към който то е свързано
- Обратното не е вярно!
 - При промяна на източника на данни свързаните към него контроли не си променят свойствата
- След промяна на данните в източника на данни за отразяване на промените в свързаните с него контроли се прави:
 1. Премахване (изтриване) на свързването
 2. Добавяне на свързването отново

Просто свързване

- Свързване на контрола към обект
- При промяна на `TextBox1.Text` се променя

Свързаният обект

```
class Customer
{
    private string mName;
    public string Name
    {
        get { return mName; }
        set { mName = value; }
    }
}

Customer cust = new Customer();
cust.Name = "Бай Иван";

TextBox1.DataBindings.Add(
    new Binding("Text", cust, "Name"));
```

Демонстрация #9

- Свързване на контрола към обект

The screenshot displays the Microsoft Visual C# .NET [design] - MainForm.cs window. The code in the MainForm.cs file is as follows:

```
private void Form1_Load(object sender, System.EventArgs e)
{
    mCustomer = new Customer();
    mCustomer.Name = "Бай Иван";

    textBoxCustomerName.DataBindings.Add(
        new Binding("Text", mCustomer, "Name"));
}

private void buttonShowCustomer_Click(object sender, EventArgs e)
{
    string customerName = mCustomer.Name;
    MessageBox.Show(customerName);
}

class Customer
{
    private string mName;
    public string Name
    {
        get { return mName; }
        set { mName = value; }
    }
}
```

The Solution Explorer on the right shows the project structure:

- Solution 'Demo-9-Binding-Control-To-Object'
- References
- App.ico
- AssemblyInfo.cs
- MainForm.cs

Overlaid on the bottom right are two dialog boxes:

- Binding Control To Object**: A dialog with a text box containing "Бай Иван", and three buttons: "Show Customer", "Change Customer", and "Rebind".
- Dialog Box**: A small dialog with the text "Бай Иван" and an "OK" button.

Просто свързване

- Свързване на контрола към списък

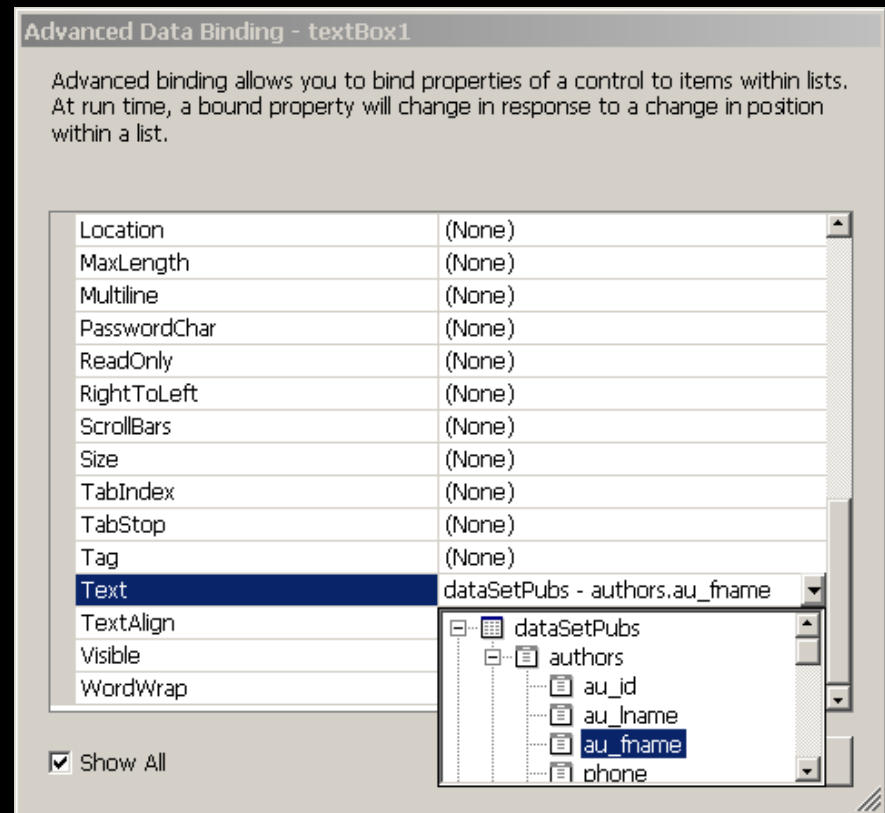
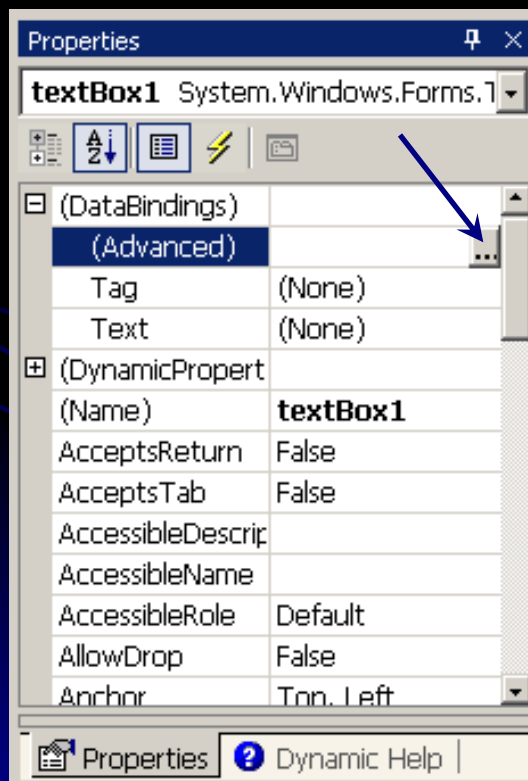
```
string[] towns = {"София", "Пловдив", "Варна"};  
TextBoxTowns.DataBindings.Add(  
    new Binding("Text", towns, ""));
```

- Свързване на контрола към таблица

```
// Имаме DataSet ds с таблица Towns с колони id и name:  
DataTable towns = new DataTable("Towns");  
towns.Columns.Add(new DataColumn("id", typeof(int)));  
towns.Columns.Add(new DataColumn("name", typeof(string)));  
DataSet ds = new DataSet();  
ds.Tables.Add(towns);  
  
TextBoxTowns.DataBindings.Add(  
    new Binding("Text", ds, "Towns.name"));
```

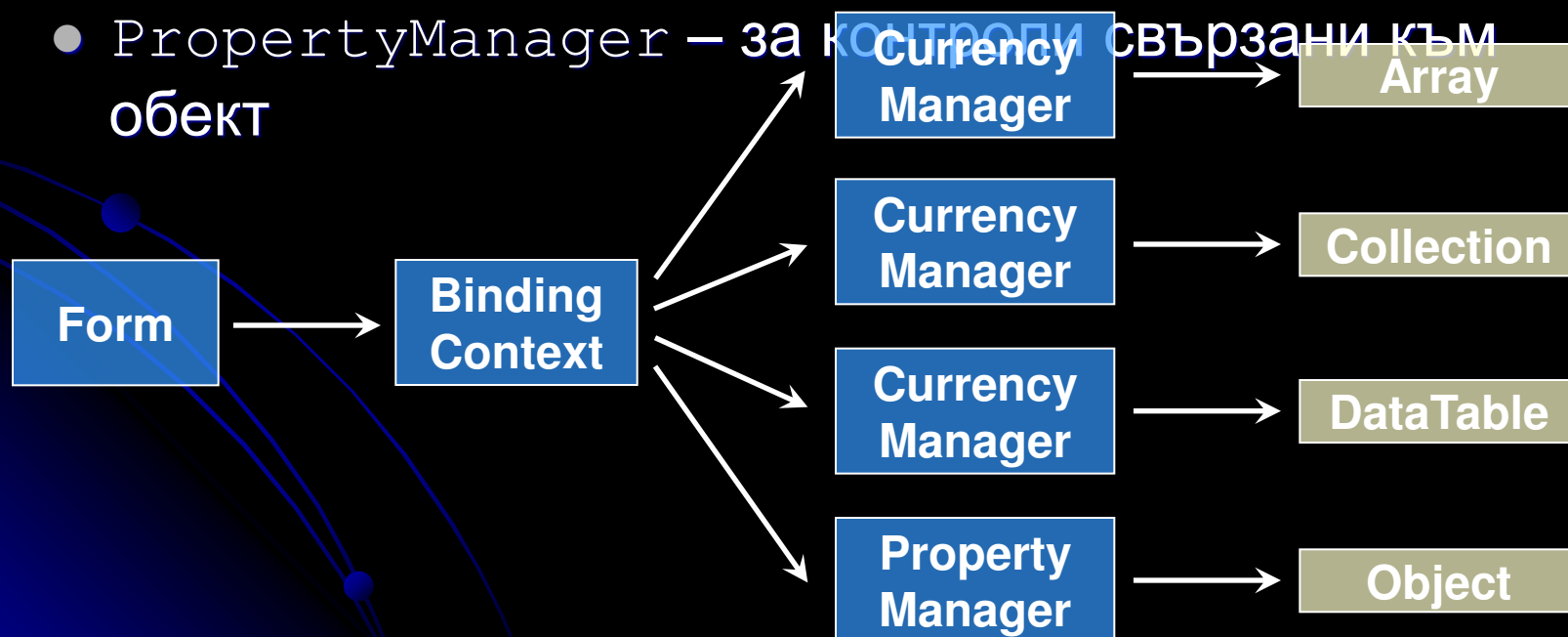
Просто свързване с VS.NET

- Свързването може да става и по време на дизайн с VS.NET (ако използваме за източник DataSet)



BindingContext

- Формата пази информация за свързаните контроли в своя BindingContext обект
 - CurrencyManager – за контроли свързани към списък. Съдържа позицията в списъка
 - PropertyManager – за контроли свързани към обект



Навигация с CurrencyManager

- CurrencyManager класът пази текущата позиция в списъка – източник на данни
 - Position – съдържа позицията
 - Count – съдържа размера на списъка
- Навигация по източника на данни
 - Извличане на CurrencyManager обекта:

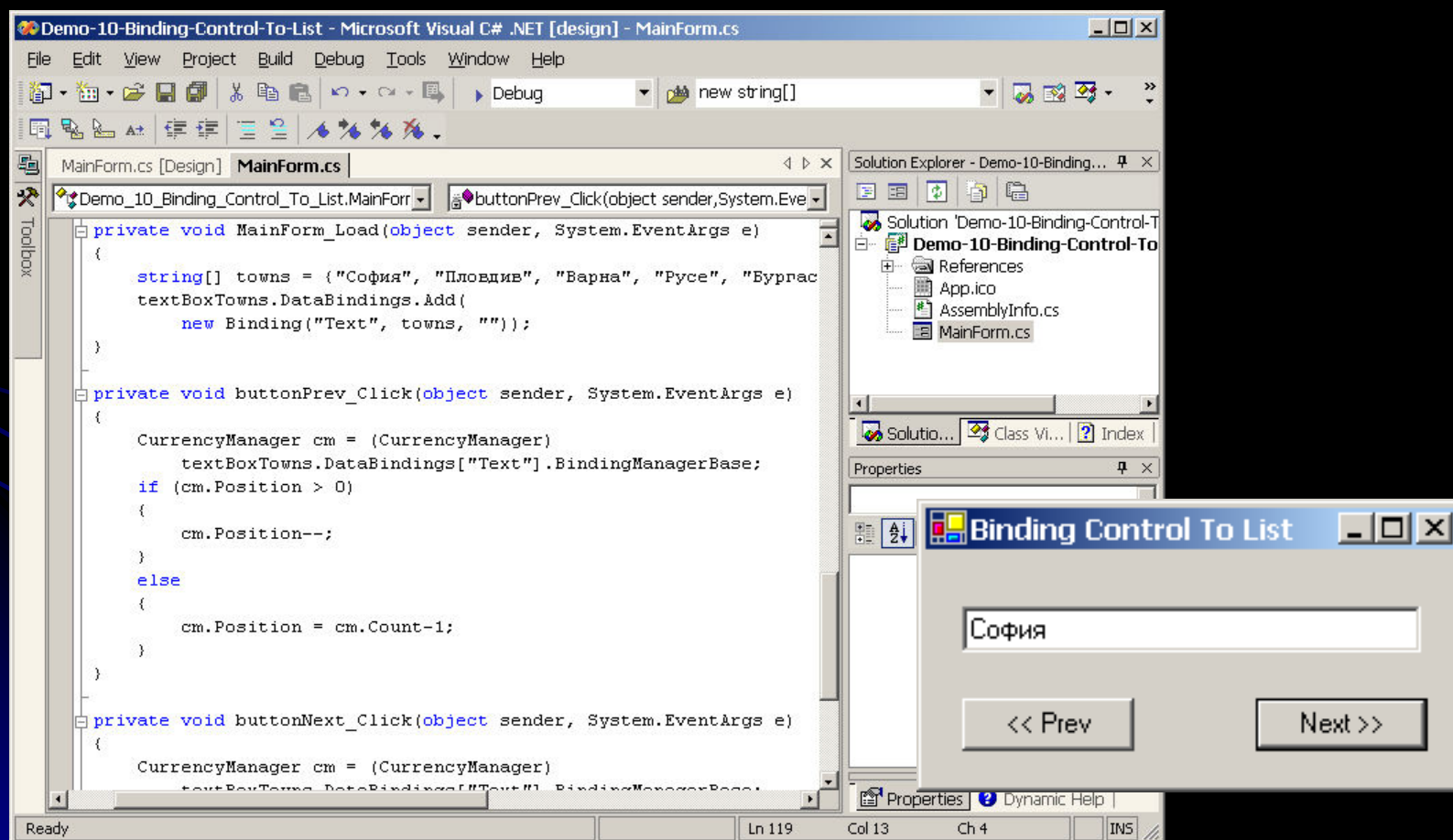
```
CurrencyManager cm = (CurrencyManager)
    textBox1.DataBindings["Text"].BindingManagerBase;
// Може и така:
CurrencyManager cm = (CurrencyManager)
    form1.BindingContext[dataTableCustomers];
```

- Навигация по списъка:

```
cm.Position++;
```

Демонстрация #10

- Свързване на контрола към списък и



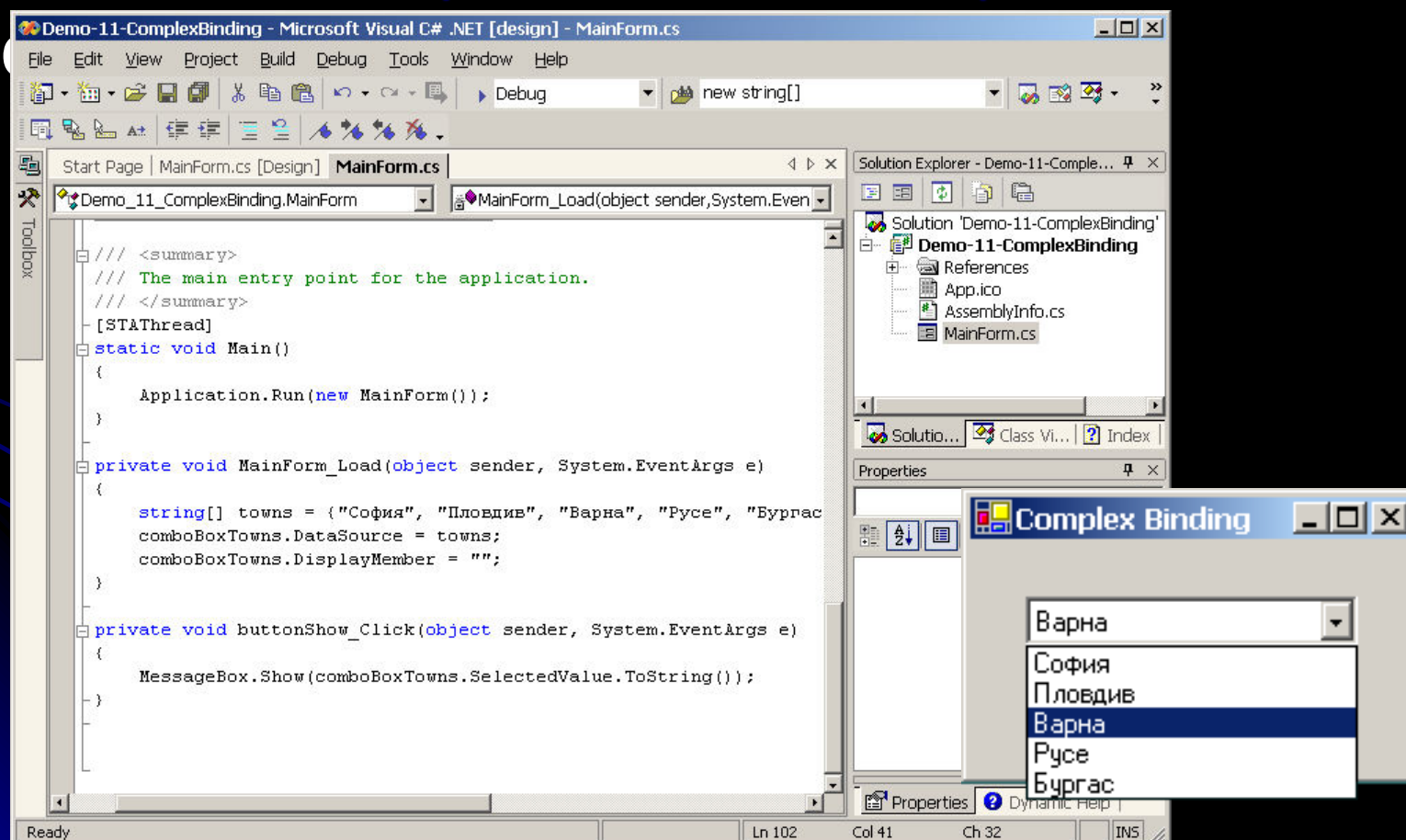
Сложно свързване

- Свързване на контрола към списък
 - Използва се при списъчни контроли:
 - `ListBox`, `ComboBox` и др.
 - Задават се свойствата:
 - `DataSource` – списък с данните
 - `DisplayMember` – път до полето, което да се визуализира
 - `ValueMember` – път до полето, от което се получава резултата
 - Пример:

```
comboBox1.DataSource = dataSetCountries;  
comboBox1.DisplayMember = "Countries.CountryCode";  
comboBox1.ValueMember = "Countries.Name";
```

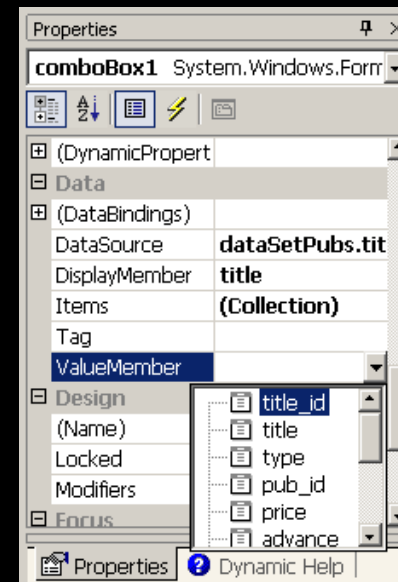
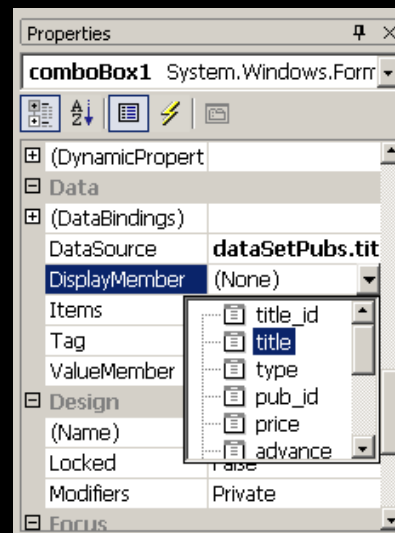
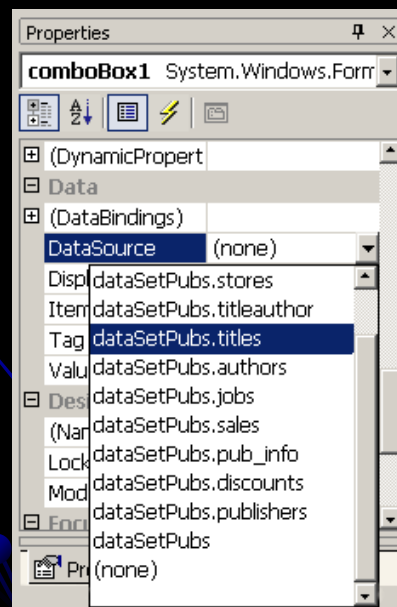
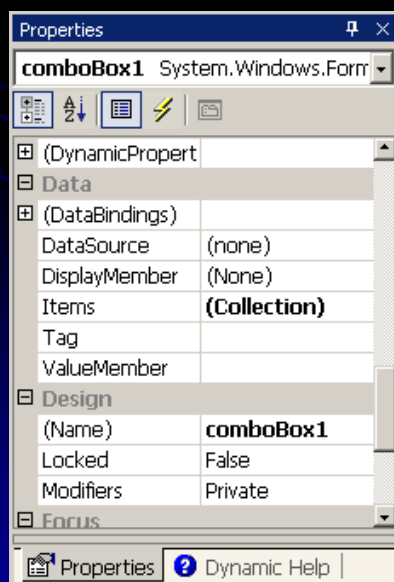
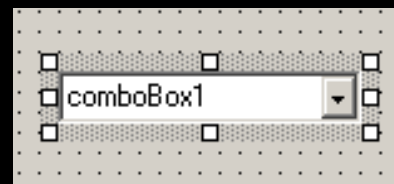
Демонстрация #11

- Сложно свързване на контрола към



Сложно свързване с VS.NET

- Свързването може да става и по време на дизайн с VS.NET (ако ползваме за източник DataSet)



Контролата DataGrid

- DataGrid контролата визуализира таблични данни
 - Осигурява навигация по редове и колони
 - Позволява редактиране на данните
 - Слаба функционалност
 - Ще бъде заменена в бъдещи версии на .NET Framework
 - Използва се най-често с ADO.NET DataSet и DataTable
 - DataSource – задава източника на данни
 - DataMember – задава пътя до данните в рамките на източника

DataGrid – пример

```
private void MainForm_Load(
    object sender, System.EventArgs e)
{
    // Create table "Towns"
    DataTable towns = new DataTable("Towns");
    towns.Columns.Add(
        new DataColumn("id", typeof(int)));
    towns.Columns.Add(
        new DataColumn("name", typeof(string)));

    // Add some rows in the table
    DataRow row = towns.NewRow();
    row["id"] = 1;
    row["name"] = "София";
    towns.Rows.Add(row);

    row = towns.NewRow();
    row["id"] = 2;
    row["name"] = "Пловдив";
    towns.Rows.Add(row);
}
```


DataGrid – пример

```
row = towns.NewRow();  
row["id"] = 3;  
row["name"] = "Барна";  
towns.Rows.Add(row);
```

```
row = towns.NewRow();  
row["id"] = 4;  
row["name"] = "Pyce";  
towns.Rows.Add(row);
```

```
// Add table "Towns" to the DataSet  
DataSet ds = new DataSet();  
ds.Tables.Add(towns);
```

```
// Bind the DataGrid to the DataSet  
DataGridTowns.DataSource = ds;  
DataGridTowns.DataMember = "Towns";
```

```
}
```

Демонстрация #12

● Работа с DataGrid контрола

The screenshot shows the Microsoft Visual C# .NET IDE in design mode for a project named "Demo-12-DataGrid". The main window displays the code for `MainForm.cs`, which is in the `MainForm_Load` event handler. The code creates a `DataSet` named `ds` and adds a table named `Towns` with four rows of data. It then binds a `DataGrid` control to this data set.

```
row = towns.NewRow();
row["id"] = 2;
row["name"] = "Пловдив";
towns.Rows.Add(row);

row = towns.NewRow();
row["id"] = 3;
row["name"] = "Варна";
towns.Rows.Add(row);

row = towns.NewRow();
row["id"] = 4;
row["name"] = "Русе";
towns.Rows.Add(row);

// Add table "Towns" to the DataSet
DataSet ds = new DataSet();
ds.Tables.Add(towns);

// Bind the DataGrid to the DataSet
dataGridTowns.DataSource = ds;
dataGridTowns.DataMember = "Towns";
```

The `Solution Explorer` on the right shows the project structure, including `References`, `App.ico`, `AssemblyInfo.cs`, and `MainForm.cs`.

In the foreground, a separate window titled "DataGrid Demo" displays the data from the `Towns` table in a grid format:

	id	name
▶	1	София
	2	Пловдив
	3	Варна
	4	Русе
*		

The status bar at the bottom indicates the application is "Ready", with the cursor at line 121, column 36, on page 27.

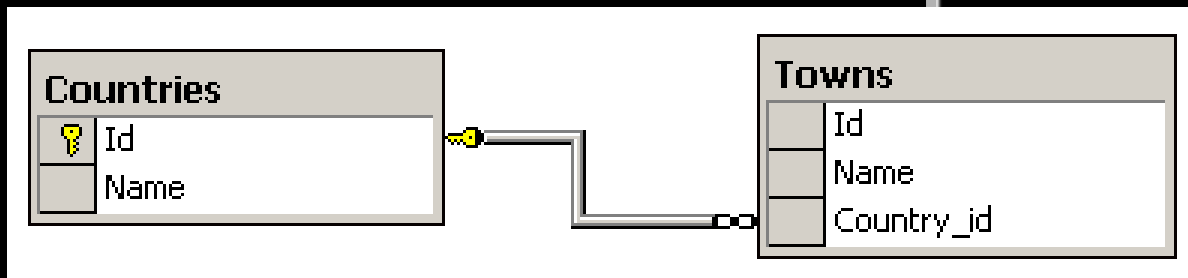
Контролата DataGridView

- По-важни свойства на DataGridView
 - ReadOnly – разрешава / забранява редакция
 - CaptionVisible – показва / скрива заглавието
 - ColumnHeadersVisible – показва / скрива заглавията на колоните
 - RowHeadersVisible – показва / скрива колоната вляво от редовете
 - TableStyles – задава стилове за таблицата
 - Активен е само първият стил
 - MappingName – задава таблицата, за която се отнася дефинираният стил
 - GridColumnStyles – задава форматирането на отделните колони – заглавие, ширина и др.

Master-Details навигация

- Master-Details навигацията отразява взаимоотношения тип "1 към много"
 - Например: 1 регион има много области
- В ADO.NET DataSet обектите се поддържат релации тип "главен/подчинен"
 - Използват се DataRelation обектите в DataSet
- Windows Forms поддържа Master-Details навигация
 - На главната контрола се задава за източник на данните главната таблица
 - На подчинената контрола се задава за източник на данните релацията на таблицата
 - Контролите се свързват с един и същ DataSet

Master-Details – пример



```
// Bind the master grid to the master table  
DataGridCountries.DataSource =  
    datasetCountriesAndTowns;  
DataGridCountries.DataMember =  
    "Countries";
```

```
// Bind the detail grid to the relationship  
DataGridTowns.DataSource =  
    datasetCountriesAndTowns;  
DataGridTowns.DataMember =  
    "Countries.CountriesTowns";
```

Демонстрация #14

● Master-Details навигация

The screenshot shows the Microsoft Visual C# .NET [design] environment for MainForm.cs. The Windows Form Designer generated code is visible, showing the Main method and the MainForm_Load event handler. The MainForm_Load method calls sqlDataAdapterRegion.Fill(dataSetNorthwind.Region) and sqlDataAdapterTerritories.Fill(dataSetNorthwind.Territories) to populate the data. The Master-Detail Demo window is open, showing a list of regions (Eastern, Western, Northern, Southern) and a corresponding list of territories for the selected region (Eastern). The territories list includes columns for code (код) and name (област).

Windows Form Designer generated code

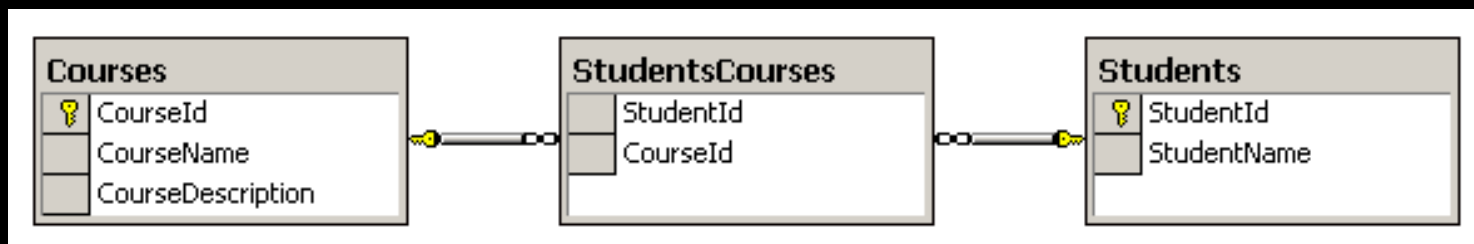
```
/// <summary>  
/// The main entry point for the application.  
/// </summary>  
[STAThread]  
static void Main()  
{  
    Application.Run(new MainForm());  
}  
  
private void MainForm_Load(object sender, EventArgs e)  
{  
    sqlDataAdapterRegion.Fill(dataSetNorthwind.Region);  
    sqlDataAdapterTerritories.Fill(dataSetNorthwind.Territories);  
}
```

Master-Detail Demo

Регион:	Област:
Eastern	код област
Western	01581 Westboro
Northern	01730 Bedford
Southern	01833 Georgetow
	02116 Boston
	02139 Cambridge
	02184 Braintree

Релации "много към много"

- Релации тип "много към много" не се поддържат от DataSet и DataGrid
- Могат да бъдат сведени до Master-Details чрез добавяне на изглед в базата от данни



```
CREATE VIEW View_StudentsCourses AS
SELECT StudentId, StudentName, CourseId, CourseName
FROM Students, Courses, StudentsCourses
WHERE Students.StudentId = StudentsCourses.StudentId
AND Courses.CourseId = StudentsCourses.CourseId
```

Релации "много към много"

◆ E	Courses	(Courses)
⚙ E	CourseId	int
E	CourseName	string
E	CourseDescri	string



◆ E	VIEW_StudentsCourses	(VIEW_StudentsCourses)
⚙ E	StudentId	int
E	StudentName	string
⚙ E	CourseId	int
E	CourseName	string

Many-to-many relationships

Курсове			
	CourseId	CourseName	CourseDescription
+	1	.NET	.NET Framework
▶ +	2	Java	Java Fundamentals
+	3	Algorithms	Basic algorithms

Студенти				
	StudentId	StudentName	CourseId	CourseName
▶	1	Иван	2	Java
	2	Петър	2	Java
	5	Мария	2	Java
	8	Милена	2	Java
	7	Силвия	2	Java
*				

Наследяване на форми

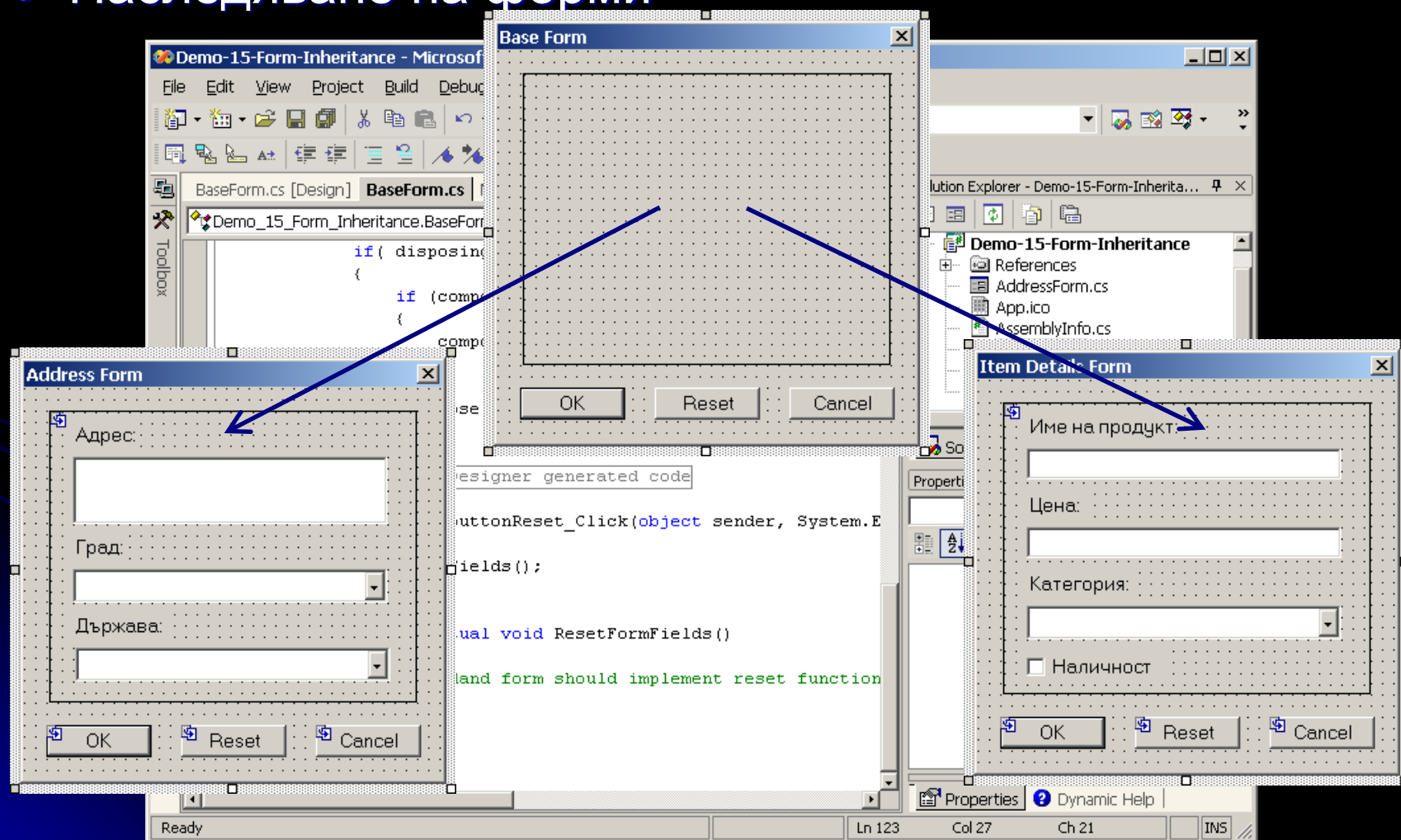
- Наследяването на форми
 - Позволява повторно използване на части от потребителския интерфейс
 - Възможно е да се променят наведнъж общите части на много форми
 - Чрез подмяна на единичен DLL файл може да се променят всички форми
 - Формите-наследници могат да добавят функционалност
 - Виртуални методи могат да реализират специфичната за наследниците функционалност, достъпна от базовата форма

Наследяване на форми

- Базовата форма е най-обикновена форма
 - Контролите, които могат да се променят, се обявяват като `protected`
 - Останалите контроли не могат да се променят от формите-наследници
- За наследяване на форма се наследява класът на базовата форма
 - Формите-наследници се създават се от Visual Studio .NET (чрез File | Add New Item ... | Inherited Form)
- При промяна на базовата форма приложението трябва да се прекомпилира

Демонстрация #15

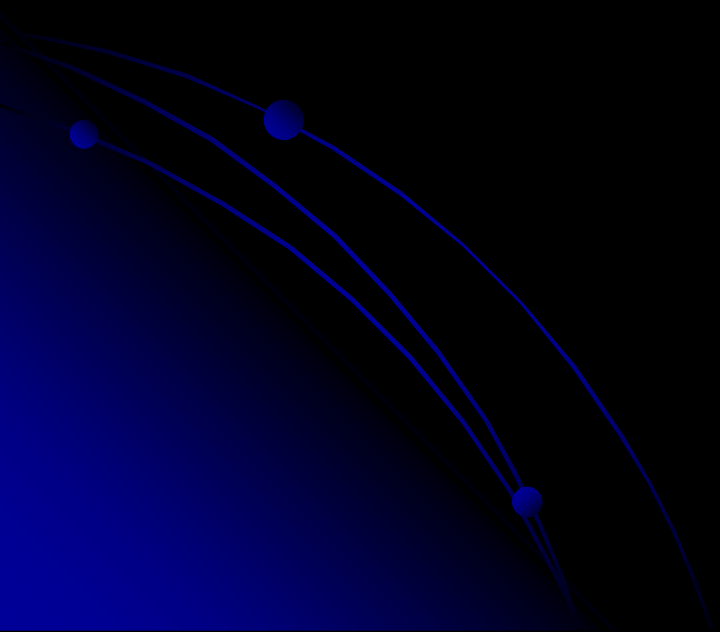
- Наследяване на форми



Печатане на принтер

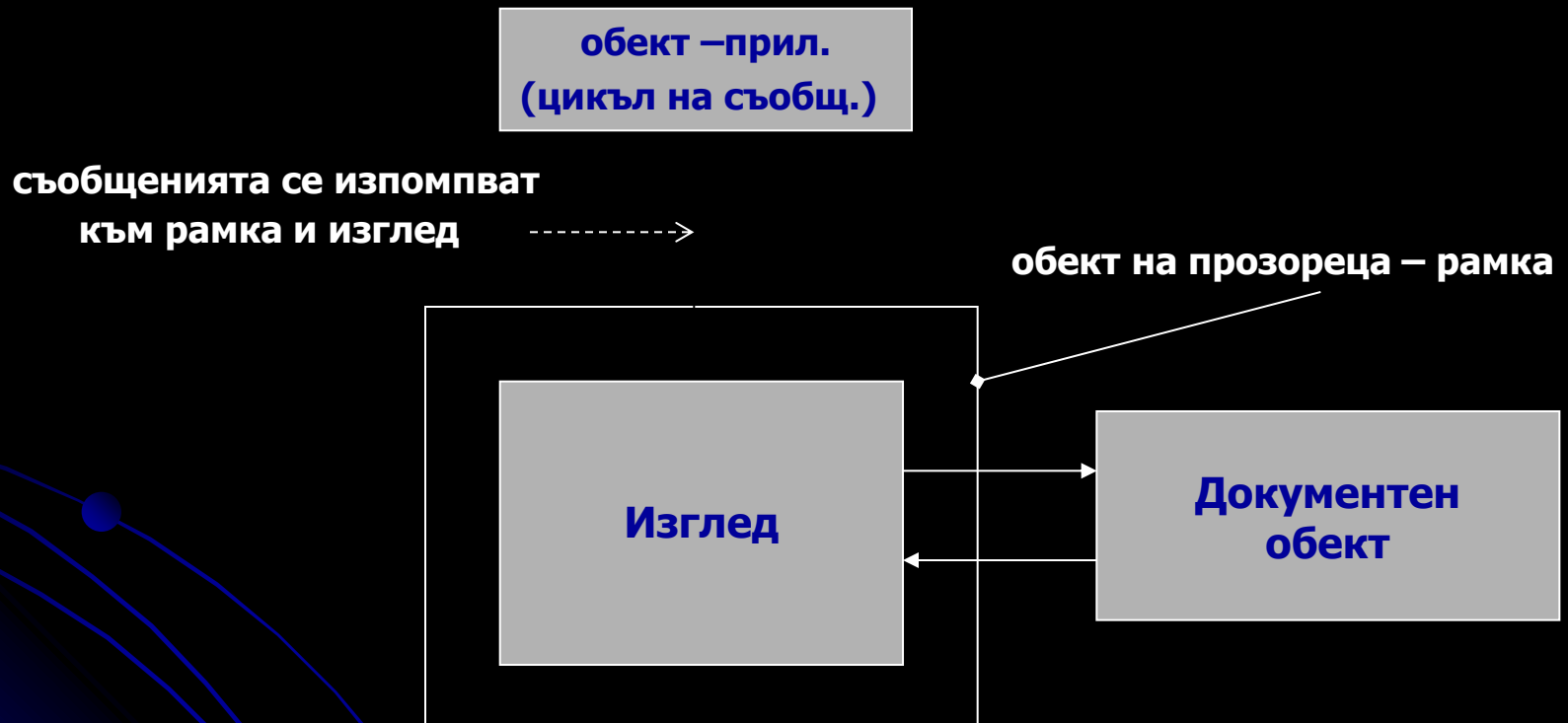
- Използват се 3 ключови класа:
 - `PrintDialog`
 - Стандартен диалог за печатане на принтер
 - `PrintController`
 - Управлява процеса на печатане и активира събития, свързани с него
 - Предоставя `Graphics` повърхността
 - `PrintDocument`
 - Описва характеристиките на отпечатвания документ
 - Съдържа `PrinterSettings` върнати от `PrintDialog`

Document/View – MFC



Програмна архитектура

Document/View



Document class:

GetFirstViewPosition(); GetNextView(); OnNewDocument(); OnOpenDocument();
Serialize()

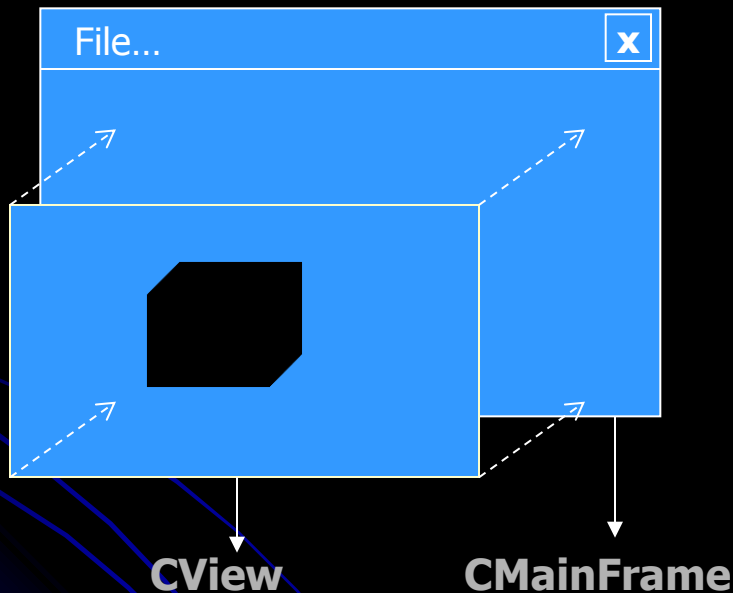
View class

```
CViewCountDoc* pDoc =  
    dynamic_cast<CViewCountDoc*>(GetDocument());  
Добре е в прил. да има собствена реал. на GetDocument():  
ViewCountDoc* CViewCountView::GetDocumet()  
{  
    CViewCountDoc* pDoc =  
        dynamic_cast<CViewCountDoc*>(GetDocument());  
    ASSERT_VALID( pDoc);  
    Return pDoc;  
}  
void CMyView::OnDraw(CDC* pDC)  
{  
    CMyDoc* pDoc = GetDocument();  
    CString string = pDoc->GetString();  
    CRect rect;  
    GetClientRect(&rect);  
    pDC->DrawText( string, rect,.....);  
}  
....OnDraw(); OnInitialUpdate(); OnUpdate()..
```

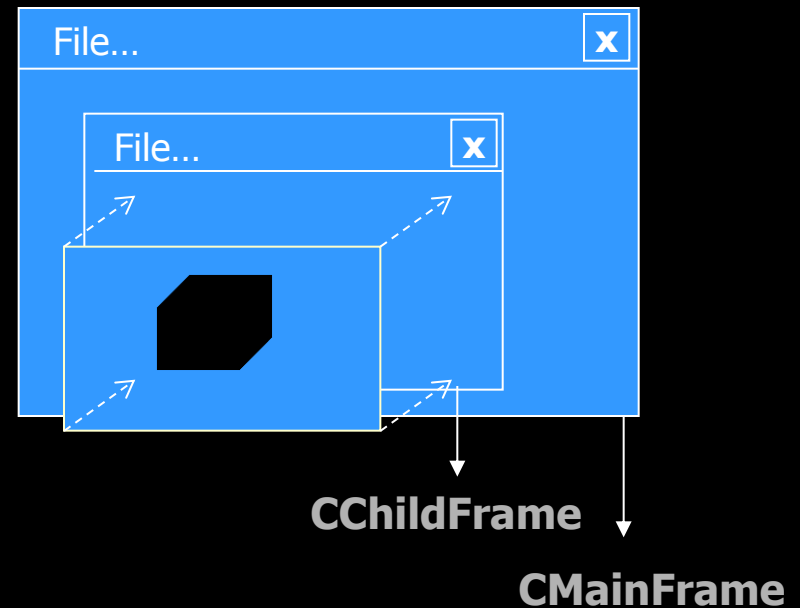
Main Frame & Child Frame Windows

OnClose(); преоразмеряване; работа с toolbar..
Произхождат от CFrameWnd и CMDIFrameWnd

SDI



MDI



можем да модифицираме поведението на MDI child frame (CChildFrame):
`BOOL CChildFrame::PreCreateWindow( CREATESTRUCT& cs) {...}`

Application class: наследява CWinApp.
Инициализации; стартиране; качване на doc. Template...

Document Template

Прави връзка м/ду (при това само по 1) document class , view, frame. Има CSingleDocTemplate и CMultipleDocTemplate. Съдържа:

- документен низ (7 полета),
\\nMy_app\\My_app\\n\\n\\nMy_app.Document\\nMy_app Document
- Иконата на изображението
- Менюто при активно изображение.

SDI документния шаблон се създава още в InitInstanse():

```
CSingleDocTemplate * pDocTemplate;
```

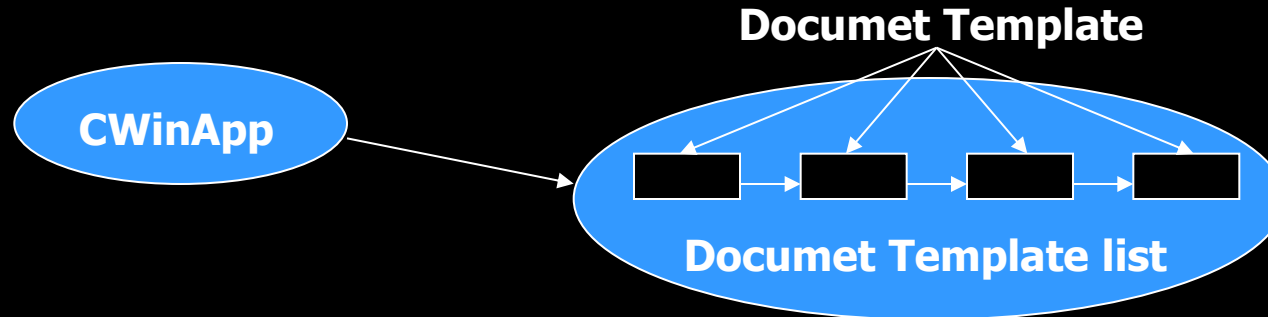
```
pDocTemplate = new CSingleDocTemplate( IDR_MAINFRAME, RUNTIME_CLASS(  
CMyDoc), RUNTIME_CLASS( CMainFrame), RUNTIME_CLASS( CMyView));
```

```
AddDocTemplate(pDocTemplate);
```

```
// следват редове зарег. на док. типове в REG, създаване обектите от  
шаблона, показване рамката и изгледа.
```

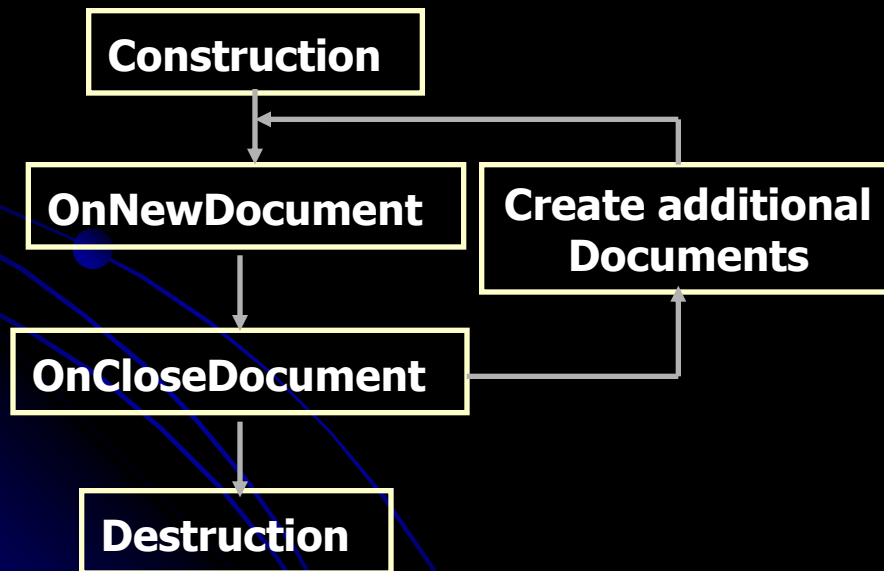
Включва се цикълът за обр. съобщения, като командните съобщения (от меню, акселер. , toolbar) се насочват за обработка към обектите - изглед→ документ→шаблон→рамка→приложение→::DefWindowProc() (command routing).

Ако MDI има повече документи и изображения (CDocManager):

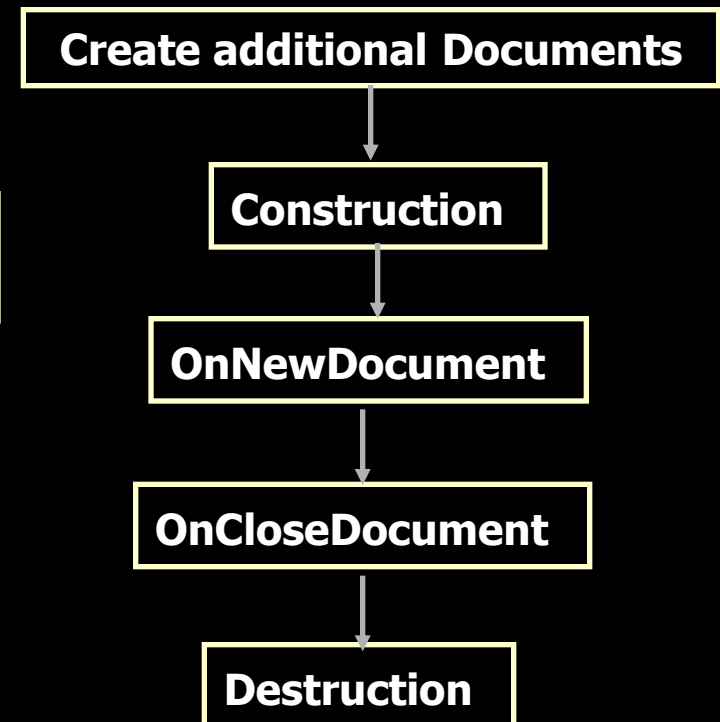


Време на живот на документа и изображението

SDI

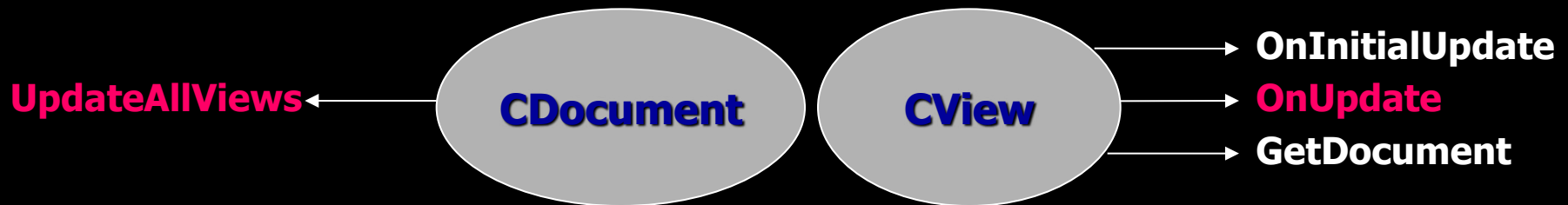


MDI



CWinApp::OnFileNew() — създава док от template; **OnNewDocument()**→**DeleteContents()**
CWinApp::OnFileOpen()- диалог;**CDocument::OnOpenDocument()**→**DeleteContents()**,сери.
Updating Document & Views

класовете документи и изображения си взаимодействат:



Основни View класове към MFC

CView — Стандартен дъщерен прозорец за вашето изображение.

CScrollView — Добавя scroll възможности към CView .

CFormView — Предоставя диалог за вграждане на контроли в изображението

CRecordView — Произведен на горния, използва се при работа с ODBC БД — изобразява запис от ODBC база данни.

CDaoRecordView — Подобен на горния, но за работа с DAO .

CEditView — Съдържа edit control, който попълва клиентската област .

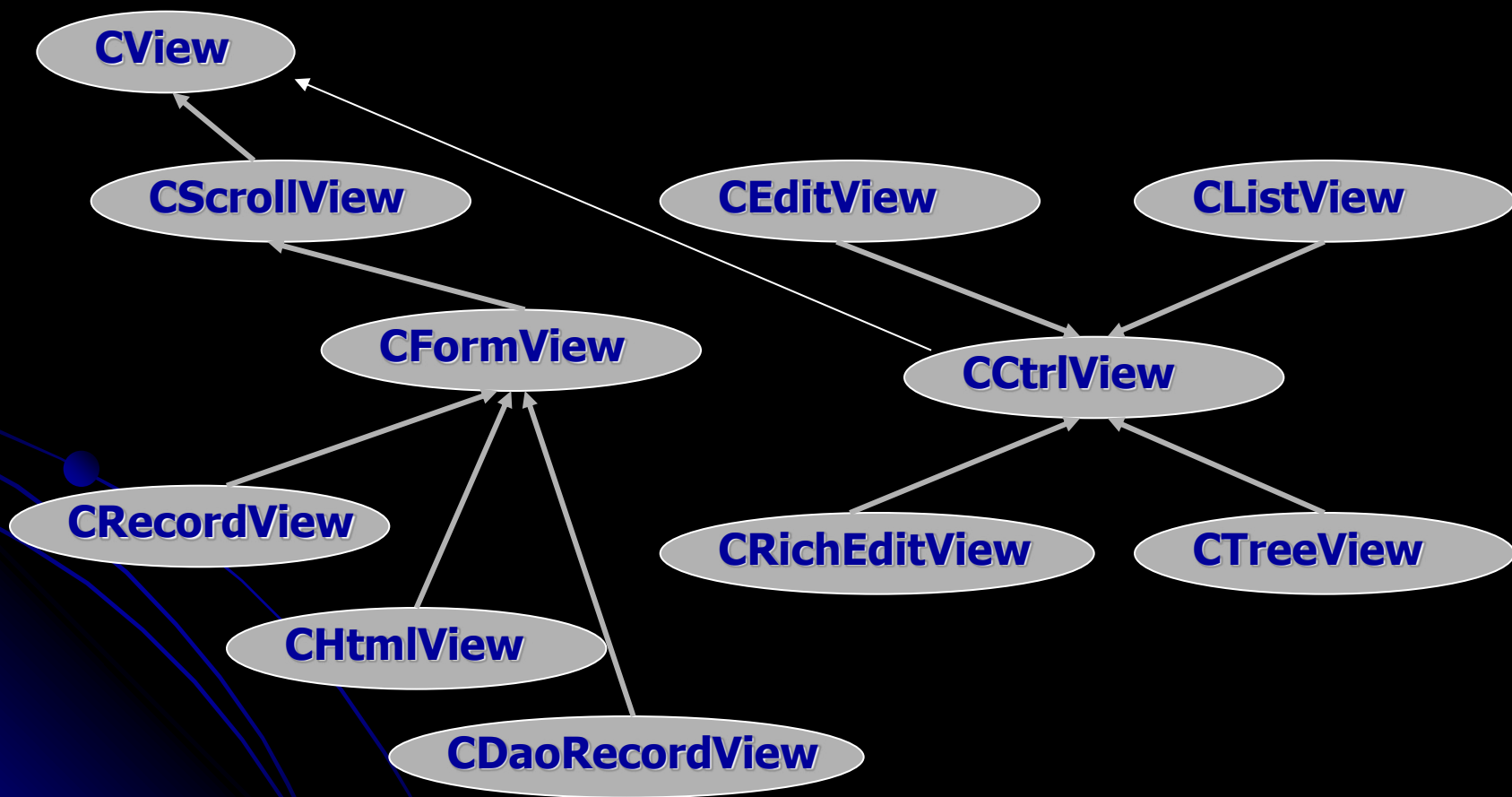
CTreeView — Съдържа tree view control, който попълва клиентската област на изображението.

CRichEditView — Съдържа rich edit control, който попълва клиентската област на изображението.

CListView — Съдържа list control.

CHtmlView — Създава изгледи от html формати и други формати, поддържани от контрола WebBrowser на Microsoft Internet Explorer.

Наследствени връзки между тези класове



CScrollView

```
void CMyView::OnInitialUpdate()  
{ CScrollView::OnInitialUpdate();  
  SetScrollSizes(MM_TEXT, CSize(1280,1024)); // +стъпка скролиране  
}
```

Основни ф-ии на класа:

CPoint pos = GetScrollPosition(); ScrollToPosition(CPoint(100,100));

CSize size = GetTotalSize(); // връща лог. размер на изгледа

int nWidth = size.cx;

int nHeight = size.cy;

SetScaleToFitSize(GetTotalSize()); // мащабира лог. изглед

SetScrollSize(....) // за динамична корекция на лог изглед

OnDraw(&dc)

Преди OnDraw(), във ф-ия OnPaint(): CPaintDC dc(this); OnPrepareDC(&dc);
OnDraw(&dc);

Ако искате OnDraw() да прерисува само инвалидизирания правоъгълник:.

CRect rect;

pDC->GetClipBox(&rect); // попълва размера на инвалид. правоъгълник.

HTML изгледи

CHtmlView: обвивка на IWebBrowser2

Невиртуални операции (готови методи) на класа:

GetBusy	показва дали в момента е download
GetLocatinURL	извлича URL адресана текущата страница
GoBack	GoForward Refresh
Stop	преустановява зареждането
Navigate	по зададен URL адрес изобразява текуща страница

ето пример:

ON_COMMAND(ID_BACK, OnBack) в message map на CMyView
ON_COMMAND(ID_REFRESH, OnRefresh)

...

```
void CMyView::OnBack()
```

```
{ GoBack() }
```

...

Преход към друга страница можете да направите:

```
Navigate( _T(http://www.microsoft.com));
```

“тънки клиенти”:

```
void CHtmlClockView::OnInitialUpdate()
```

```
{ ....Navigate(string);...
```

Дървовидни и списъчни изгледи

CTreeView

CTreeCtrl& CTreeView::GetTreeCtrl();

Интересни методи на класа:

::SetImageList() // свързва текста с картинка

::InsertItem() // вмъкване, корен, подредба, картинка и т.н.

::DeleteItem()

::SetItemText() **::SortChildren()** **::GetSelectedItem()**

CListView: получава функционалност от съответния контрол

Създаване на изгледи, базирани на контроли (наследници на CCtrlView:

- * да се декларира клас, наследник на напр. CCtrlView, като в конструктора на базовия се подава името на създавания.

- * да се опише ф-ия (GetTabCtrl() или подобна) връщаща this към обхващания контрол:

CTabCtrl& GetTabCtrl() const {return (CTabCtrl*) this;}

- * да се предефинират поне 2 метода:

CMyView::PreCreateWindow(..) // на CWnd; указва стил; инициализира библиотеката за общи контроли (::InitCommonControls())

::OnInitialUpdate() // начално изобразяване

- После **GetTabCtrl().метод_на_tab_контрола()**

Създаване на много документи и много изгледи в приложението

MDI - разлики с SDI:

- 1. Много отворени документи**
2. Различни типове документи в MDI
3. MDI поддържа меню-опция Windows...
4. Много менюта в MDI
5. Много прозорци-рамки в MDI

MDI:

- **наследява главната рамка не от CFrameWnd, а от CMDIFrameWnd.**
- * **наследява дъщерните рамки от CMDIChildWnd**
- **за шаблони на документа се използва не CSingleDocTemplate а CMultiDocTemplate с указан параметър в конструктора му (където се указваше прозорецът рамка) не главния, а класът на дъщерния прозорец – рамка.**
- **MDI имат поне два ресурса за меню – когато няма отворен документ и когато има поне 1 отворен.**

Превключване на менюта; изгледи, отворени документи... се поемат от генериран код

Различни типове документи в MDI приложение

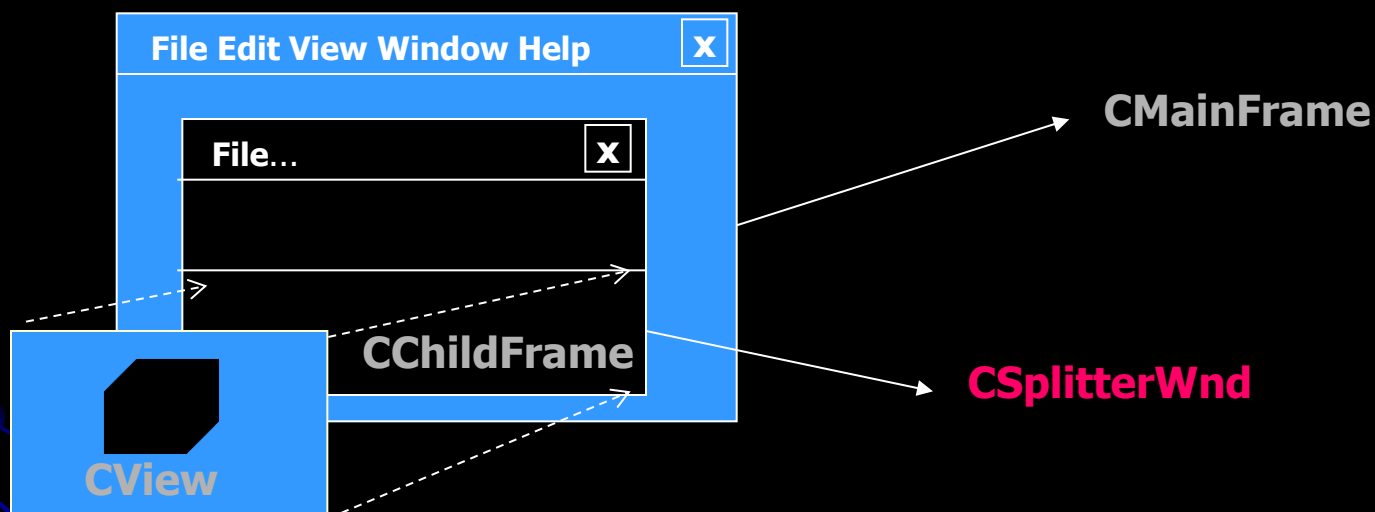
За да създадем прил. Обработващо 2 типа документи(при File→New):

- Създаваме нов клас за документ и нов клас за изглед – например CPiramDoc и CPiramView. Класовете да са динамично създаваеми.
 - За новия тип създайте 4 нови ресурса – икона, меню, таблица с акселератори (ако трябва) и документен низ. Всички те имат общ ID – например IDR_PIRAMTYPE.
 - Ако е нужно се създава и собствена дъщерна рамка за новия тип док.
 - Модифицираме InitInstance() за добавяне новия документен шаблон:
CMultiDocTemplate* pDocTemplate;
pDocTemplate = new CMultiDocTemplate(описаните по-долу 4 парам.);
 - *ресурсен ID*
 - *Указател към CRuntime class (RUNTIME_CLASS()) за документа.*
 - *Указател към CRuntime class за рамката.*
 - *Указател към CRuntime class за изгледа (CPiramView).*
- регистрирате новия шаблон:
AddDocTemplate(pDocTemplate);
Ако имате повече документни типове – за всеки в InitInstance() подобен код. CMultiDocTemplate има грижа да организира списъка.

Това може да се направи и в SDI, ако искате да работите с повете документни типове (но едва ли е нужно)

Разцепени изображения

Ето връзката между елементите на split изображението



Създаване на приложение, поддържащо динамични

сплитери (max 2 части, указани при създаване; имат обща лента за скролиране; по подразбиране се базират на общ view, но ако създадете наследник на CSplitterWnd и предефинирате ::CreateView() – то може и на различен)

1. деклариране на нова пром. към CChildFrame (или CMainFrame):

// Implementation

private:

CSplitterWnd m_wndSplitter;

2. предефиниране на виртуалната OnCreateClient() на рамката :

```
BOOL CChildFrame:: OnCreateClient( LPCREATESTRUCT lpcs,  
CCreateContext* pContext)  
{ CSize size (10, 10);  
Return m_wndSplitter.Create(this, 2, 1, size, pContext);}
```

Параметър1: родителя на прозореца със сплитери – прозорец- рамка

Параметър2: максималния брой панели на които може да се раздели

Параметър4: макс. ширина и височина при които панелите се създават

Параметър5: стандартна структура, едно от полетата ѝ е име на класа на изгледа, използван за панелите, *doc; *doc_template.

3. CSplitView::OnDraw()

CSplitterWnd има методи за придвижване на разделителя

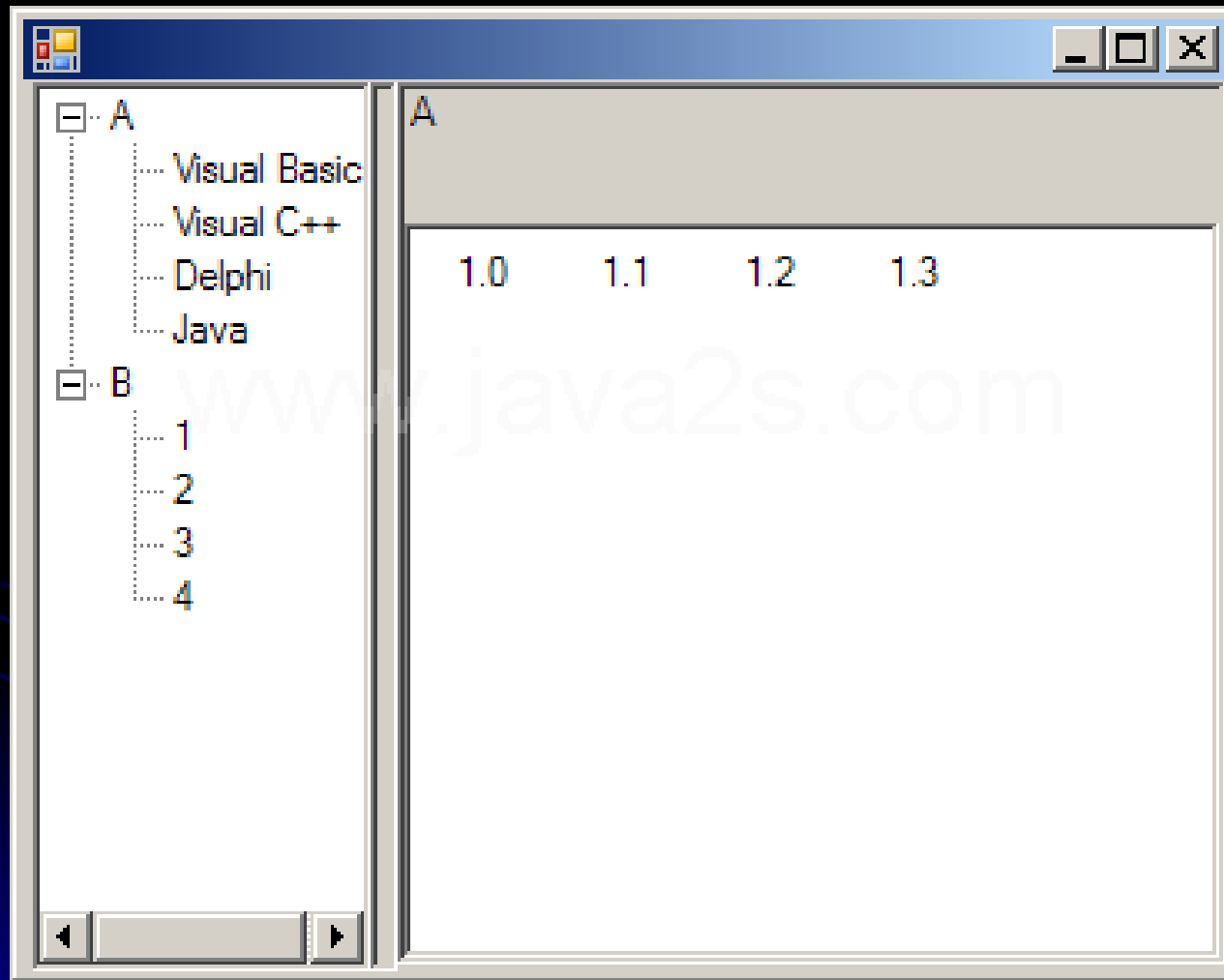
Може създаване вложени прозорци с дин. Сплитери , в прозорец-статичен

Синхронизиране с промените в документа става както беше описано

Създаване на приложение, поддържащо статично split изображение (max 16 X 16, създадени при начално изобразяване; променят се само размерите; базират се на разл. Изображения: пр. Windows Explorer)

- Ако в AppWizard укажете създаване не с MFC-базиран стил на прил., а Windows Explorer базирано, то се структурират 2 статични сплитера
- в класа на прозореца рамка (CMainFrame или CChildFrame):
CSplitterWnd m_wndSplitter;
- предефиниране :
BOOL CChildFrame::OnCreateClient(LPCREATESTRUCT lpc, CreateContext* pContext)
{
if (m_wndSplitter.CreateStatic(this, 2, 1) == FALSE) {..
// **явно, динамично създаване на изглед** за всеки статичен панел от прозореца:
BOOL bCreated = m_wndSplitter.CreateView(ред_0, кол_0, нач.координати, RUNTIME_CLASS(клас изглед) за панеларазмери на панела);
bCreated = m_wndsplitter.CreateView(1, 0,...);
// координатите са долу ляво, указва се и изглед за втория split view
SetActiveView((CView*) m_wndsplitter.GetPanel(0,0); //активен панел
Return TRUE;
}

Splitter Control



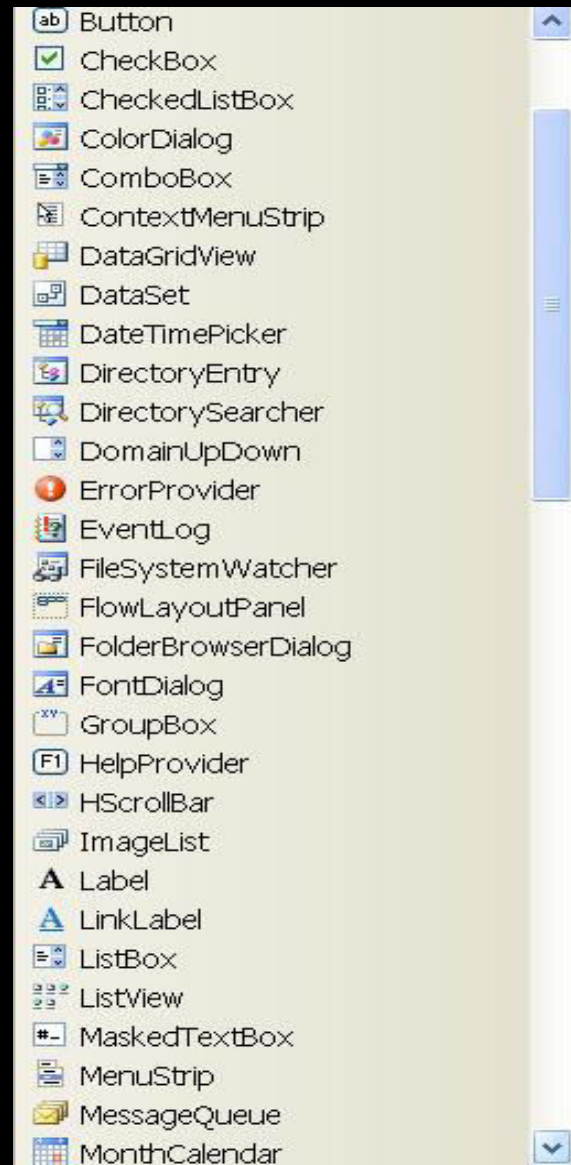
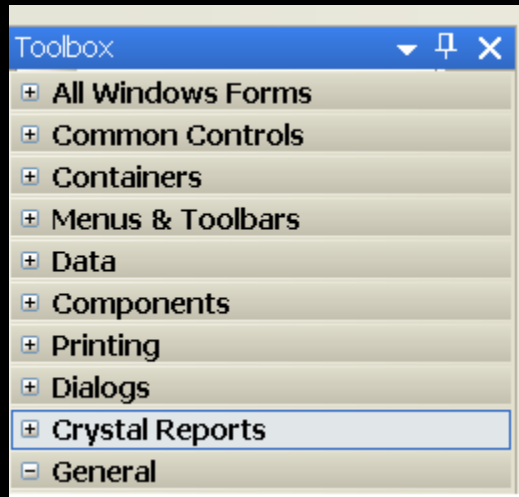
'Splitter1

```
,  
Me.Splitter1.Location = New System.Drawing.Point(0, 0)  
Me.Splitter1.Name = "Splitter1"  
Me.Splitter1.Size = New System.Drawing.Size(3, 266)  
Me.Splitter1.TabIndex = 0  
Me.Splitter1.TabStop = False  
,
```

'TreeView1

```
,  
Me.TreeView1.Dock = System.Windows.Forms.DockStyle.Left  
Me.TreeView1.Location = New System.Drawing.Point(3, 0)  
Me.TreeView1.Name = "TreeView1"  
Me.TreeView1.Nodes.AddRange(New System.Windows.Forms.TreeNode()  
{New System.Windows.Forms.TreeNode("A", New System.Windows.Forms.TreeNode()  
{New System.Windows.Forms.TreeNode("Visual Basic"), New System.Windows.Forms.TreeNode("Visua  
l C++"), New System.Windows.Forms.TreeNode("Delphi"), New System.Windows.Forms.TreeNode("Jav  
a")}}, New System.Windows.Forms.TreeNode("B", New System.Windows.Forms.TreeNode() {New Syste  
m.Windows.Forms.TreeNode("1"), New System.Windows.Forms.TreeNode("2"), New System.Windows.F  
orms.TreeNode("3"), New System.Windows.Forms.TreeNode("4")}}}))  
Me.TreeView1.Size = New System.Drawing.Size(101, 266)  
Me.TreeView1.TabIndex = 1
```

Toolbox



Containers

- Pointer
- FlowLayoutPanel
- GroupBox
- Panel
- SplitContainer
- TabControl
- TableLayoutPanel

- VScrollBar
- WebBrowser
- ReportViewer

Common Controls

- Pointer
- Button
- CheckBox
- CheckedListBox
- ComboBox
- DateTimePicker
- Label
- LinkLabel
- ListBox
- ListView
- MaskedTextBox
- MonthCalendar
- NotifyIcon
- NumericUpDown
- PictureBox
- ProgressBar
- RadioButton
- RichTextBox
- TextBox
- ToolTip
- TreeView
- WebBrowser

- StatusStrip
- TabControl
- TableLayoutPanel
- Timer
- TextBox
- ToolStrip
- ToolStripContainer
- ToolTip
- TrackBar
- TreeView
- VScrollBar
- WebBrowser
- ReportViewer

Menus & Toolbars

- Pointer
- ContextMenuStrip
- MenuStrip
- StatusStrip
- ToolStrip
- ToolStripContainer

Data

- Pointer
- DataSet
- DataGridView
- BindingSource
- BindingNavigator
- ReportViewer

Components

Printing

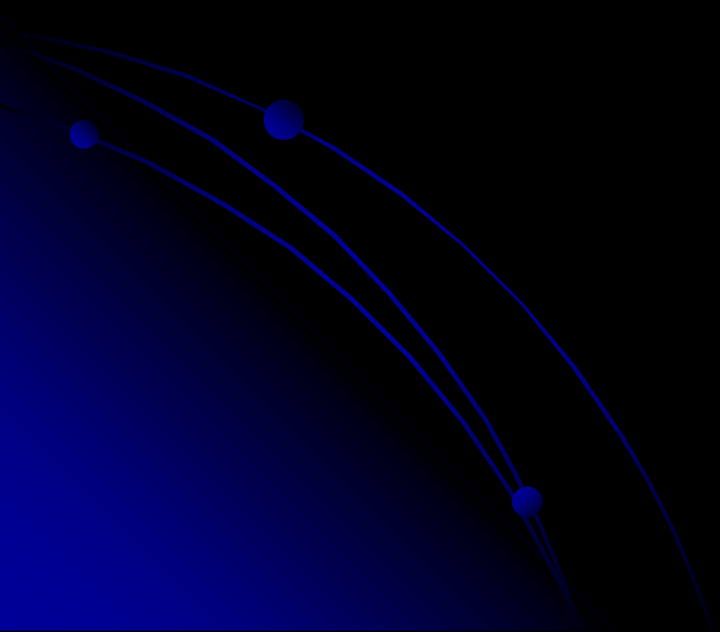
- Pointer
- PageSetupDialog
- PrintDialog
- PrintDocument
- PrintPreviewControl
- PrintPreviewDialog

Dialogs

- Pointer
- ColorDialog
- FolderBrowserDialog
- FontDialog
- OpenFileDialog
- SaveFileDialog

Crystal Reports

GDI +



Graphics class's methods

System.Drawing

DrawArc	Draws an arc from the specified ellipse.
DrawBezier	Draws a cubic bezier curve.
DrawBeziers	Draws a series of cubic Bezier curves.
DrawClosedCurve	Draws a closed curve defined by an array of points.
DrawCurve	Draws a curve defined by an array of points.
DrawEllipse	Draws an ellipse.
DrawImage	Draws an image.
DrawLine	Draws a line.
DrawPath	Draws the lines and curves defined by a GraphicsPath.
DrawPie	Draws the outline of a pie section.
DrawPolygon	Draws the outline of a polygon.
DrawRectangle	Draws the outline of a rectangle.
DrawString	Draws a string.
FillEllipse	Fills the interior of an ellipse defined by a bounding rectangle.
FillPath	Fills the interior of a path.
FillPie	Fills the interior of a pie section.
FillPolygon	Fills the interior of a polygon defined by an array of points.
FillRectangle	Fills the interior of a rectangle with a Brush.
FillRectangles	Fills the interiors of a series of rectangles with a Brush.
FillRegion	Fills the interior of a Region.



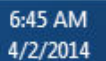
WindowsFormsApplication4.Form1 OnPaint(PaintEventArgs pe)

100 %

9

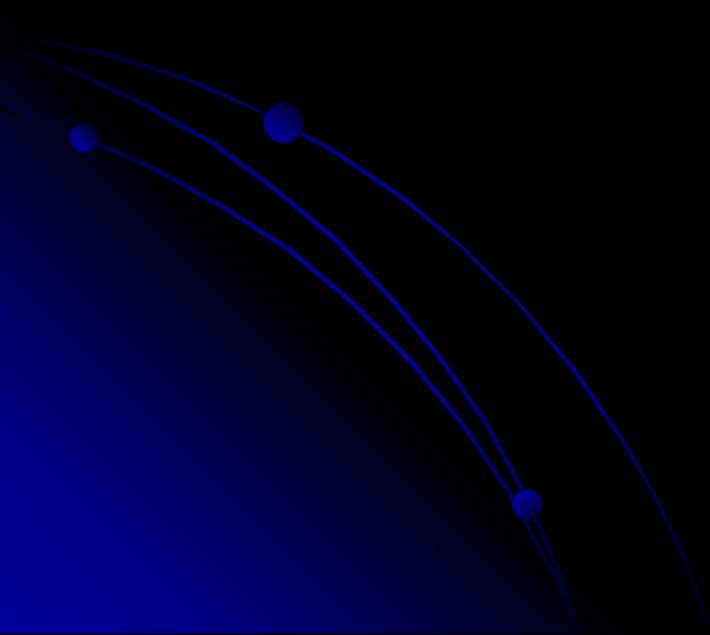
Project ▲

INS



Graphics Objects

Brush	Used to fill enclosed surfaces with patterns, colors, or bitmaps.
Pen	Used to draw lines and polygons, including rectangles, arcs, and pies
Font	Used to describe the font to be used to render text
Color	Used to describe the color used to render a particular object. In GDI+ color can be alpha blended



The Pen Class

A pen draws a line of specified width and style. You always use Pen constructor to create a pen. The constructor initializes a new instance of the **Pen class**. You can initialize it with a color or brush.

Initializes a new instance of the **Pen** class with the specified color.

public Pen(Color);

Initializes a new instance of the Pen class with the specified Brush.

public Pen(Brush);

Initializes a new instance of the Pen class with the specified Brush and width.

public Pen(Brush, float);

Initializes a new instance of the Pen class with the specified Color and Width.

public Pen(Color, float);

Here is one example:

Pen pn = **new** Pen(Color.Blue); or Pen pn = **new** Pen(Color.Blue, 100);

Some of its most commonly used properties are:

Align ment	Gets or sets the alignment for objects drawn with this Pen.
Brush	Gets or sets the Brush that determines attributes of this Pen.
Color	Gets or sets the color of this Pen.
Width	Gets or sets the width of this Pen.

Graphics Class

```
private void form1_Paint(object  
    sender, PaintEventArgs e)  
{  
    Graphics g = e.Graphics;  
}
```

----- OR

```
protected override void  
    OnPaint(PaintEventArgs e)  
{  
    Graphics g = e.Graphics;  
}
```

The Color Structure

A Color structure represents an ARGB color. Here are ARGB properties of it:

A	Gets the alpha component value for this Color .
B	Gets the blue component value for this Color .
G	Gets the green component value for this Color .
R	Gets the red component value for this Color .

You can call the Color members. Each color name (say Blue) is a member of the Color structure.

Here is how to use a Color structure:

```
Pen pn = new Pen( Color.Blue );
```


The Font Class

The Font class defines a particular format for text such as font type, size, and style attributes. You use font constructor to create a font. Initializes a new instance of the Font class with the specified attributes.

`public Font(string, float);`

Initializes a new instance of the Font class from the specified existing

Font and FontStyle.

`public Font(Font, FontStyle);`

Where FontStyle is an enumeration and here are its members:

Member Name	Description
Bold	Bold text.
Italic	Italic text.
Regular	Normal text.
Strikeout	Text with a line through the middle.
Underline	Underlined text.

Here is one example:

Graphics g ;

Font font = new Font("Times New Roman", 26);

Some of its most commonly used properties are:

Bold	Gets a value indicating whether this Font is bold.
FontFamily	Gets the FontFamily of this Font .
Height	Gets the height of this Font .
Italic	Gets a value indicating whether this Font is Italic.
Name	Gets the face name of this Font .
Size	Gets the size of this Font .
SizeInPoints	Gets the size, in points, of this Font .
Strikeout	Gets a value indicating whether this Font is strikeout (has a line through it).
Style	Gets style information for this Font .
Underline	Gets a value indicating whether this Font is underlined.
Unit	Gets the unit of measure for this Font .

The Brush Class

The **Brush** class is an abstract base class and cannot be instantiated. We always use its derived classes to instantiate a brush object, such as

SolidBrush, TextureBrush, RectangleGradientBrush, and LinearGradientBrush.

Here is one example:

```
LinearGradientBrush lBrush = new LinearGradientBrush(rect, Color.Red,  
Color.Yellow, LinearGradientMode.BackwardDiagonal);
```

OR

```
Brush brsh = new SolidBrush(Color.Red), 40, 40, 140, 140);
```

The SolidBrush class defines a brush made up of a single color.

Brushes are used to fill graphics shapes such as rectangles, ellipses, pies, polygons, and paths.

The **TextureBrush** encapsulates a Brush that uses an fills the interior of a shape with an image.

The **LinearGradientBrush** encapsulates both two-color gradients and custom multi-color gradients.

The Rectangle Structure

`public Rectangle(Point, Size);` or `public Rectangle(int, int, int, int);`

The Rectangle structure is used to draw a rectangle on WinForms. Besides its constructor, the Rectangle structure has following members:

Bottom	Gets the y-coordinate of the lower-right corner of the rectangular region defined by this Rectangle .
Height	Gets or sets the width of the rectangular region defined by this Rectangle .
IsEmpty	Tests whether this Rectangle has a Width or a Height of 0.
Left	Gets the x-coordinate of the upper-left corner of the rectangular region defined by this Rectangle .
Location	Gets or sets the coordinates of the upper-left corner of the rectangular region represented by this Rectangle .
Right	Gets the x-coordinate of the lower-right corner of the rectangular region defined by this Rectangle .
Size	Gets or sets the size of this Rectangle .
Top	Gets the y-coordinate of the upper-left corner of the rectangular region defined by this Rectangle .
Width	Gets or sets the width of the rectangular region defined by this Rectangle .
X	Gets or sets the x-coordinate of the upper-left corner of the rectangular region defined by this Rectangle .
Y	Gets or sets the y-coordinate of the upper-left corner of the rectangular region defined by this Rectangle .

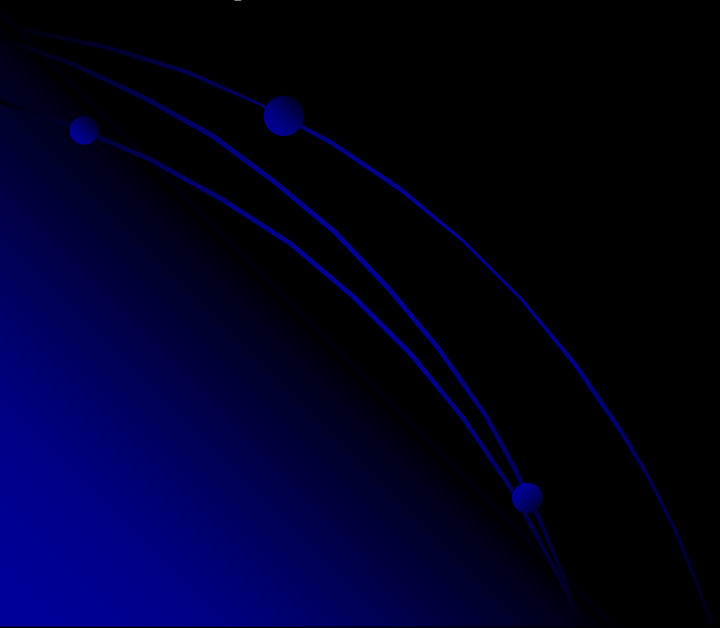
The Point Structure

This structure is similar to the POINT structure in C++. t represents an ordered pair of x and y coordinates that define a point in a two-dimensional plane.

The member x represents the x coordinates and y represents the y coordinates of the plane.

Here is how to instantiate a point structure:

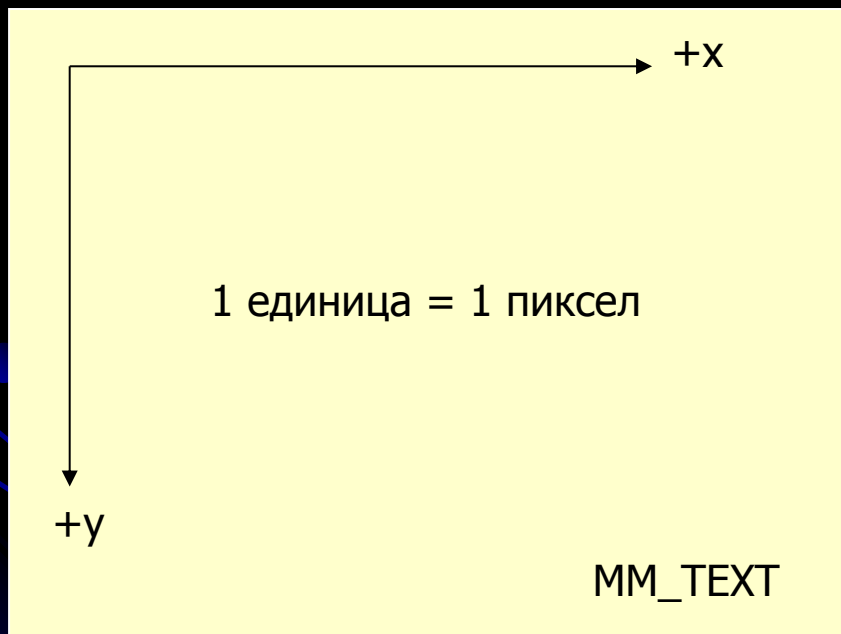
```
Point pt1 = new Point( 30, 30);  
Point pt2 = new Point( 110, 100);
```



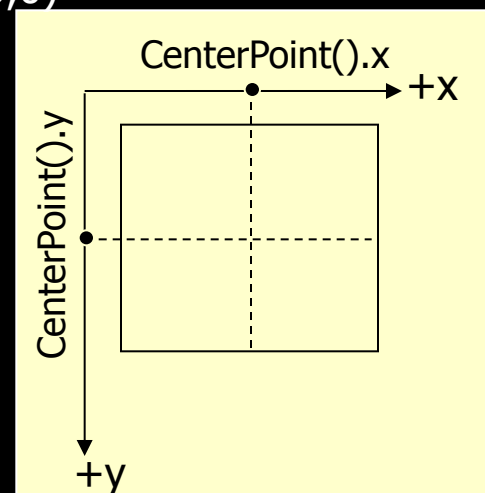
координати

(0,0)

Дисплейна повърхност

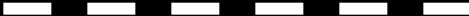


(0,0)



Pen

PS_SOLID 

PS_DASH 

PS_DOT 

PS_DASHDOT 

PS_DASHDOTDOT 

PS_NULL

PS_INSIDEFRAME 

Brush

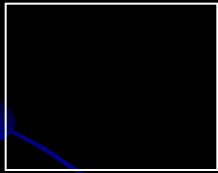
Стилове щриховки



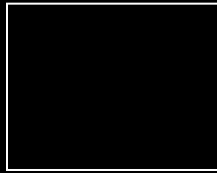
HS_BDIAGONAL



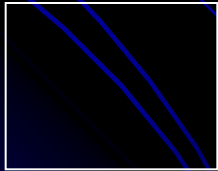
HS_FDIAGONAL



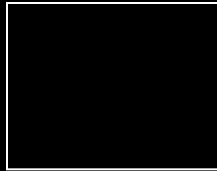
HS_CROSS



HS_HORIZONTAL



HS_DIAGCROSS

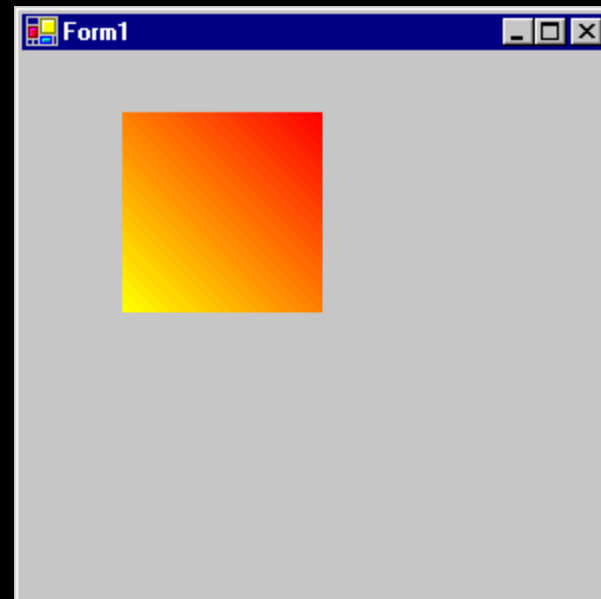


HS_VERTICAL

Drawing a rectangle

You can override OnPaint event of your form to draw an rectangle. The LinearGradientBrush encapsulates a brush and linear gradient.

```
protected override void OnPaint(PaintEventArgs pe)
{
    Graphics g = pe.Graphics ;
    Rectangle rect = new Rectangle(50, 30, 100, 100);
    LinearGradientBrush IBrush = new LinearGradientBrush(rect, Color.Red, Color.Yellow,
    LinearGradientMode.BackwardDiagonal);
    g.FillRectangle(IBrush, rect);
}
```



Drawing an Arc

DrawArc function draws an arc. This function takes four arguments.

First is the Pen. You create a pen by using the **Pen** class.

The Pen constructor takes at least one argument, the color or the brush of the pen.

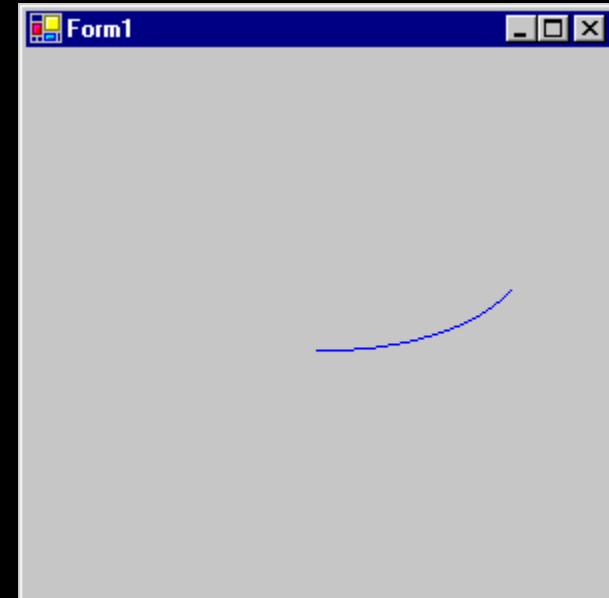
Second argument width of the pen or brush is optional.

Pen pn = new Pen(Color.Blue); or Pen pn = new Pen(Color.Blue, 100);

The second argument is a rectangle. You can create a rectangle by using Rectangle structure. The Rectangle constructor takes four int type arguments and they are left and right corners of the rectangle.

Rectangle rect = new Rectangle(50, 50, 200, 100);

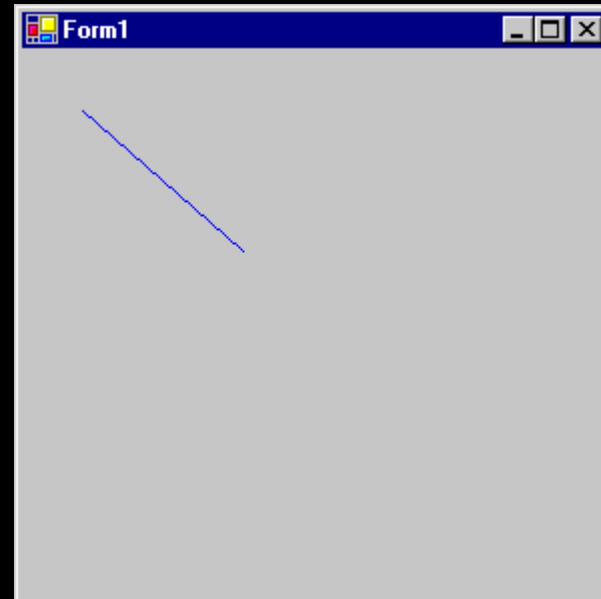
```
protected override void OnPaint(PaintEventArgs pe)
{
    Graphics g = pe.Graphics ;
    Pen pn = new Pen( Color.Blue );
    Rectangle rect = new Rectangle(50, 50, 200, 100);
    g.DrawArc( pn, rect, 12, 84 );
}
```



Drawing a Line

DrawLine function of the Graphics class draws a line. It takes three parameters, a pen, and two Point class parameters, starting and ending points. Point class constructor takes x, y arguments.

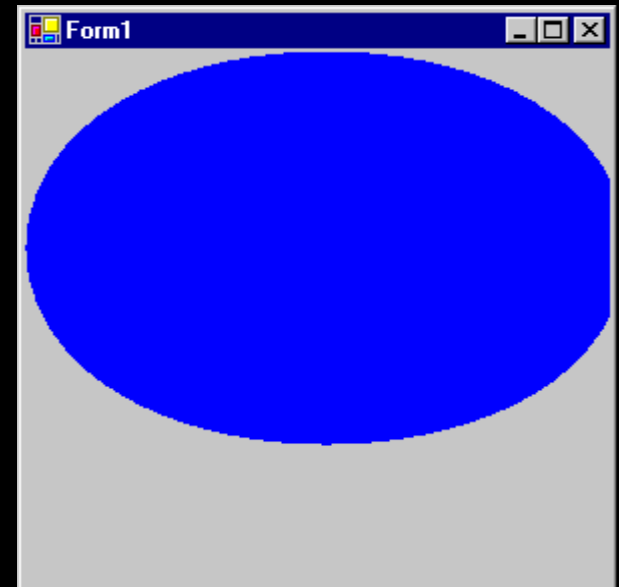
```
protected override void OnPaint(PaintEventArgs pe)
{
    Graphics g = pe.Graphics ;
    Pen pn = new Pen( Color.Blue );
    // Rectangle rect = new Rectangle(50, 50, 200, 100);
    Point pt1 = new Point( 30, 30);
    Point pt2 = new Point( 110, 100);
    g.DrawLine( pn, pt1, pt2 );
}
```



Drawing an Ellipse

An ellipse(or a circle) can be drawn by using DrawEllipse method. This method takes only two parameters, Pen and rectangle.

```
protected override void OnPaint(PaintEventArgs pe)
{
    Graphics g = pe.Graphics ;
    Pen pn = new Pen( Color.Blue, 100 );
    Rectangle rect = new Rectangle(50, 50, 200, 100);
    g.DrawEllipse( pn, rect );
}
```



The FillPath

Drawing bazier curves is little more complex than other objects.

protected override void OnPaint(PaintEventArgs pe)

{

Graphics g = pe.Graphics;

g.FillRectangle(new SolidBrush(Color.White), ClientRectangle);

GraphicsPath path = new GraphicsPath(new Point[] {

new Point(40, 140), new Point(275, 200),

new Point(105, 225), new Point(190, 300),

new Point(50, 350), new Point(20, 180), },

new byte[] {

(byte)PathPointType.Start,

(byte)PathPointType.Bezier,

(byte)PathPointType.Bezier,

(byte)PathPointType.Bezier,

(byte)PathPointType.Line,

(byte)PathPointType.Line,

});

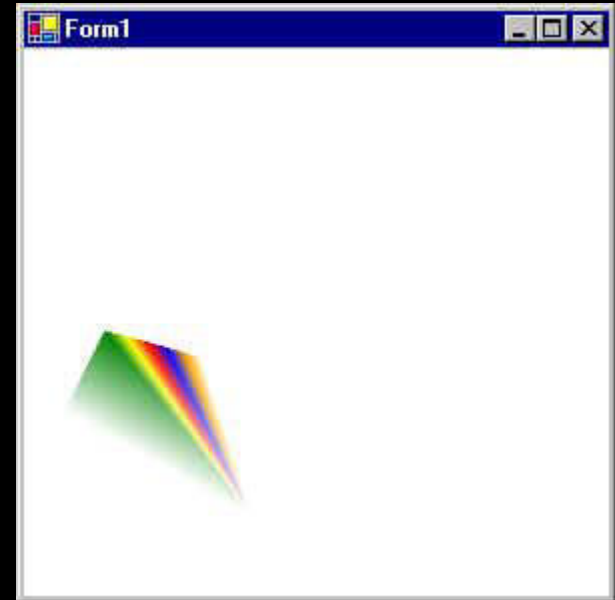
PathGradientBrush pgb = new PathGradientBrush(path);

pgb.SurroundColors = new Color[] { Color.Green, Color.Yellow, Color.Red, Color.Blue,

Color.Orange, Color.White, };

g.FillPath(pgb, path);

}



Drawing Text and Strings

You can override OnPaint event of your form to draw an rectangle.
The LinearGradientBrush encapsulates a brush and linear gradient.

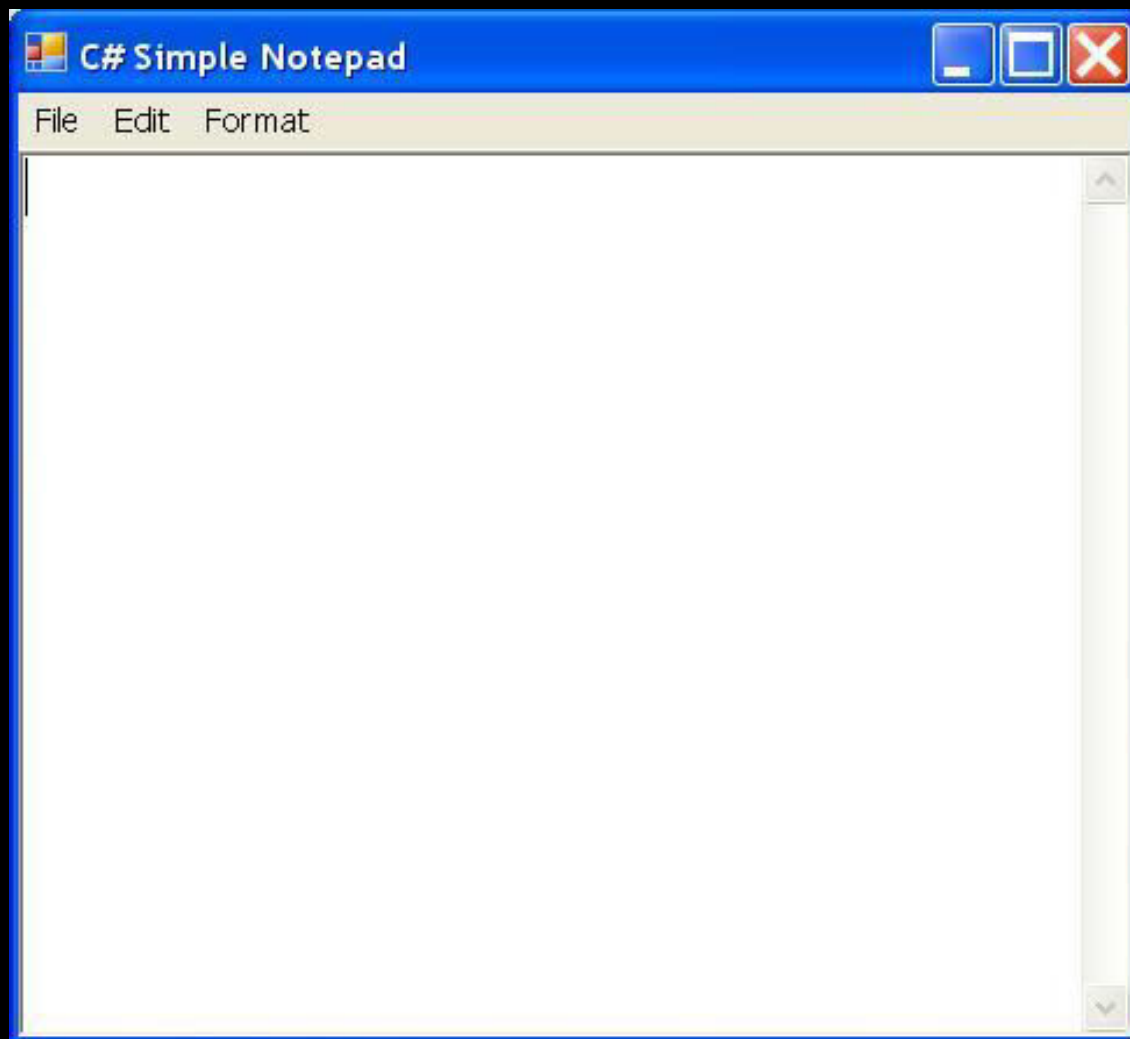
```
protected override void OnPaint(PaintEventArgs pe)
{
    Font fnt = new Font("Verdana", 16);
    Graphics g = pe.Graphics;
    g.DrawString("GDI+ World", fnt, new SolidBrush(Color.Red), 14,10);
}
```



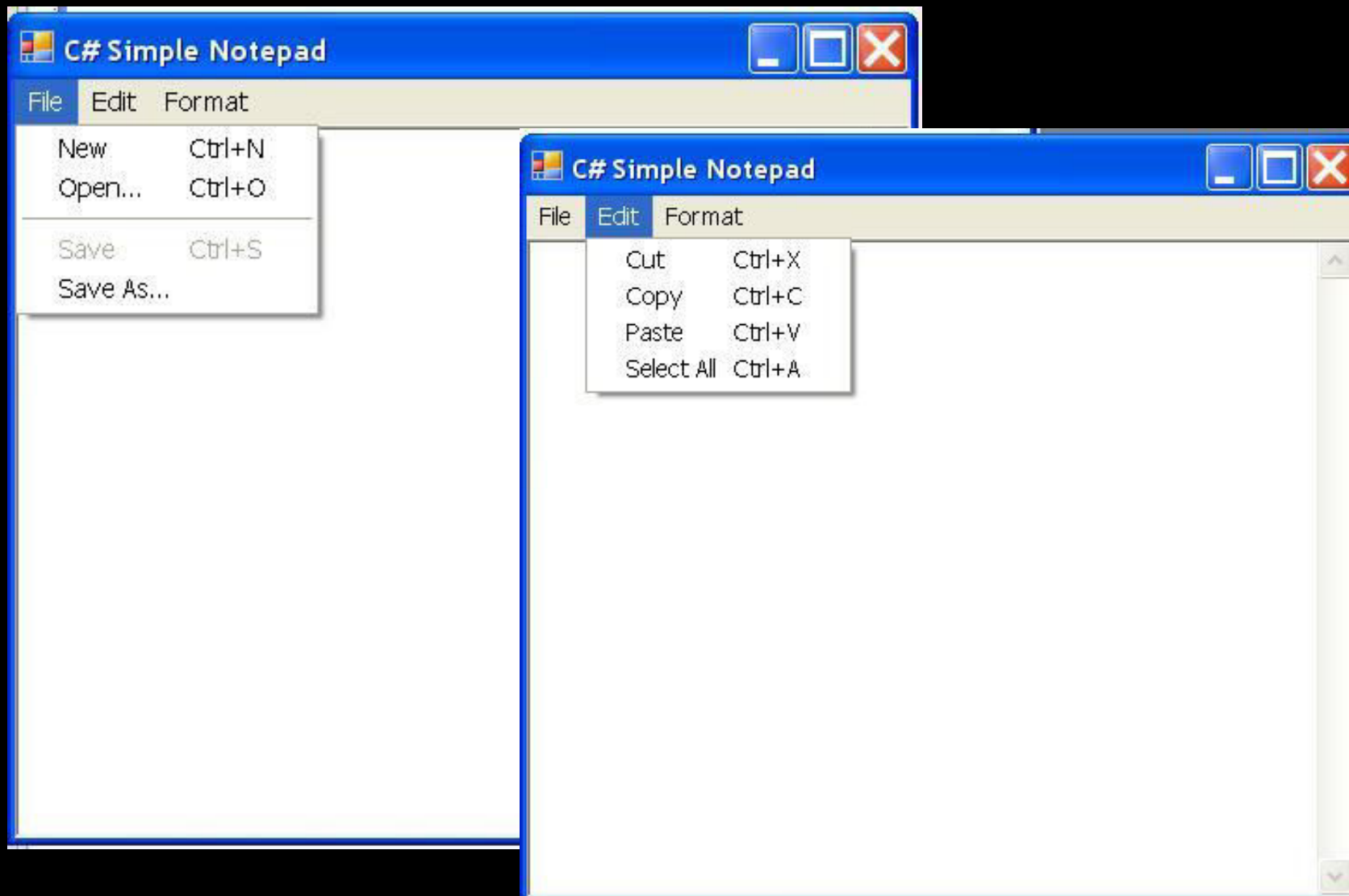
Таймер. Реализация на текстов редактор (Timer, txt Editor)

Лекция 3

Пример – txt редактор



Меню



Реализация - open

```
private void menuItemOpen_Click(object sender, System.EventArgs e)
{
    CheckChanged(sender,e);
    dlgResult = dlgOpenFile.ShowDialog();

    if(dlgResult == DialogResult.Cancel)
        return;

    try
    {
        m_strFileName = dlgOpenFile.FileName;
        StreamReader sr = new StreamReader(m_strFileName);
        txtBody.Text = sr.ReadToEnd();
        sr.Close();
        m_bModified = false;
    }
    catch(Exception err)
    {
        MessageBox.Show(err.Message, "Error", MessageBoxButtons.OK, MessageBoxIcon.Error)
    }
}
```

Save as

```
private void menuItemSaveAs_Click(object sender, System.EventArgs e)
{
    dlgSaveFile.FileName = "mydoc" + m_intDocNumber.ToString() + ".txt";
    dlgResult = dlgSaveFile.ShowDialog();

    if(dlgResult == DialogResult.Cancel)
        return;

    try
    {
        m_strFileName = dlgSaveFile.FileName;
        m_sw = new StreamWriter(m_strFileName);
        m_sw.Write (txtBody.Text);
        m_sw.Close();
        menuItemSave.Enabled = true;
        m_intDocNumber++;
        m_bModified = false;
        this.Text = "C# Simple Notepad: " + m_strFileName;
    }
    catch(Exception err)
    {
        MessageBox.Show(err.Message, "Error", MessageBoxButtons.OK, MessageBoxIcon.Error);
    }
}
```

Edit

```
#region Implementation of the Edit Menu Items
private void menuItemCut_Click(object sender, System.EventArgs e)
{
    //put the selected Text in the Clipboard
    Clipboard.SetDataObject(txtBody.SelectedText);
    txtBody.SelectedText = ""; //remove the selected text.. So you think it has been
}

private void menuItemCopy_Click(object sender, System.EventArgs e)
{
    txtBody.Copy();
}

private void menuItemPast_Click(object sender, System.EventArgs e)
{
    txtBody.Paste();
}

private void menuItemSelectAll_Click(object sender, System.EventArgs e)
{
    txtBody.SelectAll();
}
```

class System.EventArgs

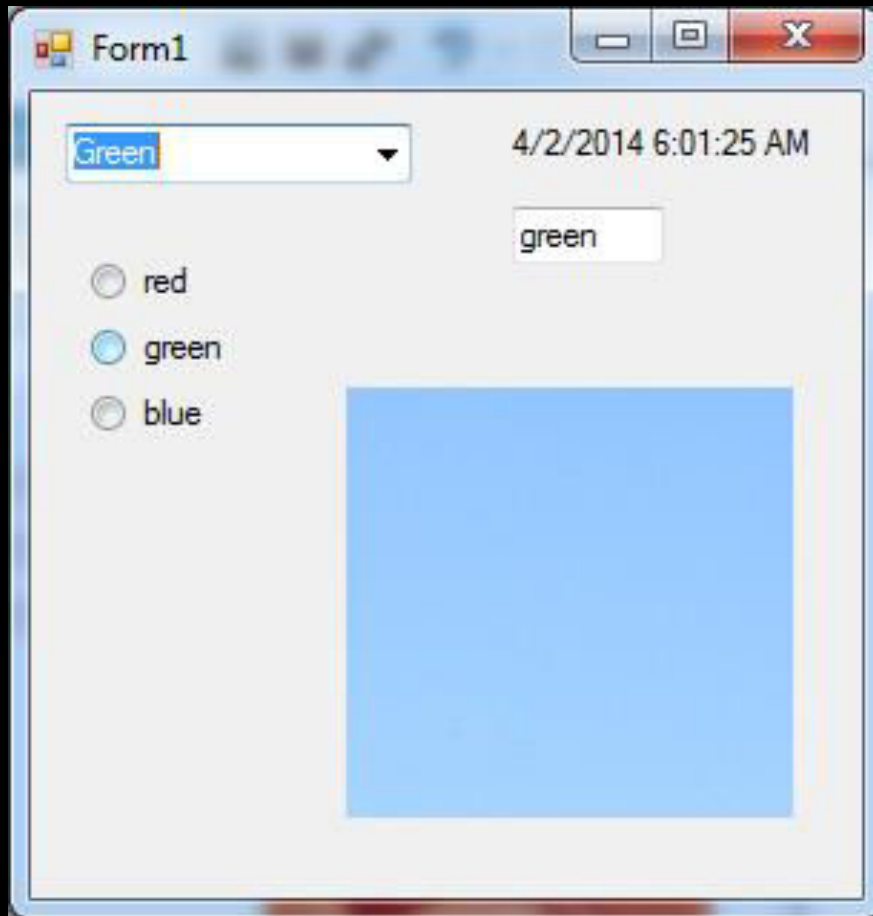
System.EventArgs is the base class for classes containing event

Font Change

```
private void menuItemFont_Click(object sender, System.EventArgs e)
{
    dlgResult = dlgFont.ShowDialog();
    if(dlgResult == DialogResult.Cancel)
        return;

    FontFamily fmFontName = dlgFont.Font.FontFamily;
    float fFontSize = dlgFont.Font.Size;
    FontStyle fsStyle = dlgFont.Font.Style;
    Font font;
    try
    {
        font = new Font(fmFontName, fFontSize, txtBody.Font.Style ^ fsStyle);
        txtBody.Font = font;
    }
    catch (ArgumentException)
    {
        return;
    }
}
```

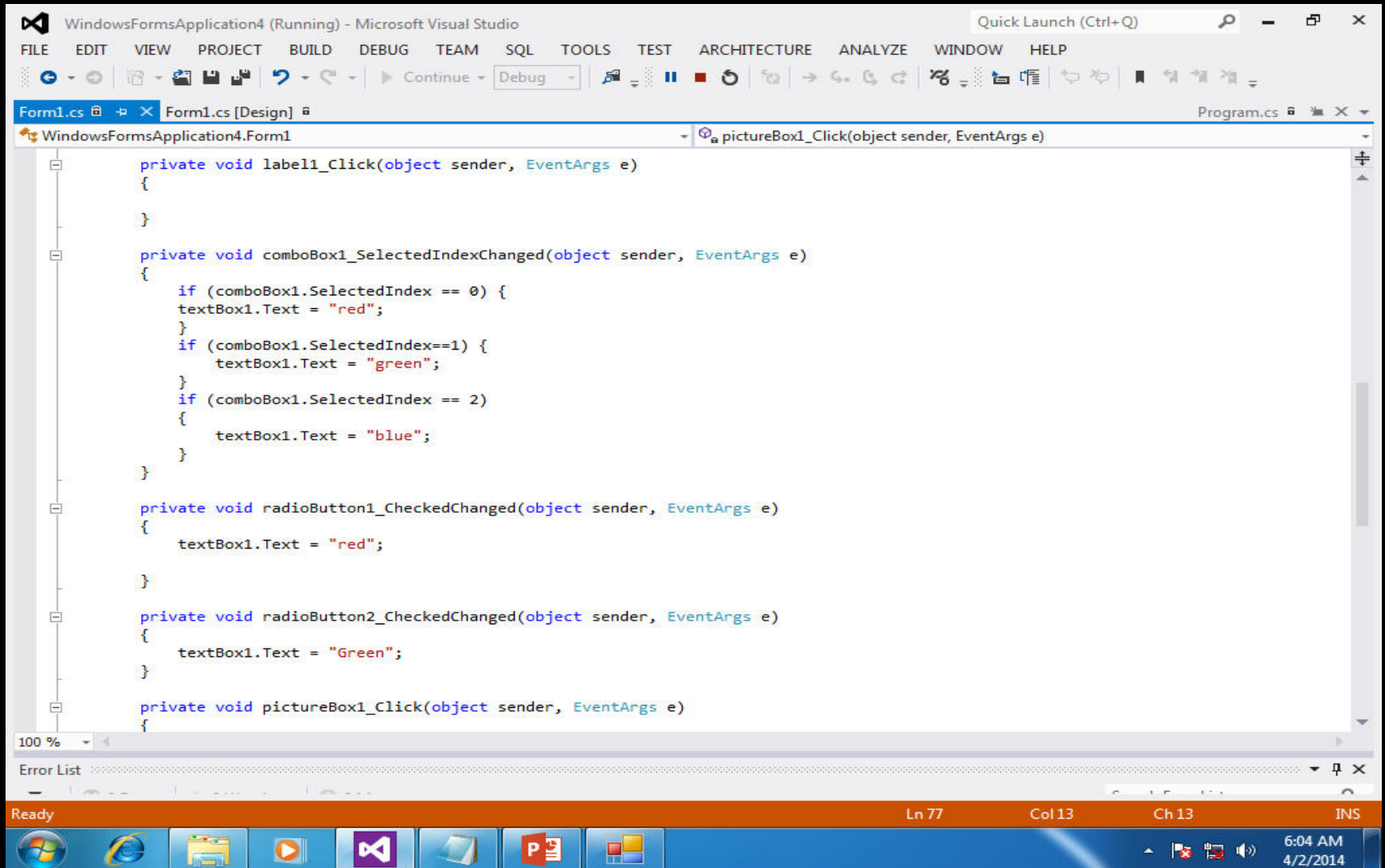
Example - 2



The screenshot shows a Windows application window titled "Form1". Inside the window, there is a ComboBox at the top left displaying "Green". To its right is a label showing the date and time "4/2/2014 6:01:25 AM". Below the ComboBox are three radio buttons labeled "red", "green", and "blue". The "green" radio button is selected. To the right of the radio buttons is a TextBox containing the text "green". Below the TextBox is a large PictureBox displaying a solid blue square.

1. ComboBox
2. RadioButton
3. Timer
4. PictureBox
5. TextBox
6. Label

Events



Използване на Таймер

- Създаване на приложение
- Добавяне на контрола Timer
- Основни събития (Events)
- Основни свойства (Properties)
- Примери използващи Timer контрол

Timer

The screenshot displays the Microsoft Visual Studio IDE for a project named "WindowsFormsApplication4". The main design view shows a form titled "Form1" with a dropdown menu, a "Time" label, a text box, three radio buttons labeled "red", "green", and "blue", and a large image of orange petals. A "timer1" control is visible in the component tray. The Properties window on the right shows the "timer1" control of type "System.Windows.Forms.Timer". The "Interval" property is set to 1000 milliseconds. The "Behavior" section shows "Enabled" is True. The "Design" section shows the name is "timer1", "GenerateMember" is True, and "Modifiers" are Private. A tooltip for the "Interval" property is visible, stating: "Interval: The frequency of Elapsed events in milliseconds." The Error List at the bottom shows 0 Errors, 0 Warnings, and 0 Messages. The Windows taskbar at the bottom shows the time as 5:58 AM on 4/2/2014.

WindowsFormsApplication4 - Microsoft Visual Studio

Quick Launch (Ctrl+Q)

FILE EDIT VIEW PROJECT BUILD DEBUG TEAM SQL TOOLS TEST ARCHITECTURE ANALYZE WINDOW HELP

Object Browser Toolbox Form1.cs [Design]* Program.cs

Form1

Time

red green blue

timer1

Properties

timer1 System.Windows.Forms.Timer

Behavior

Enabled True

Interval 1000

Data

(ApplicationSettings)

Tag

Design

(Name) timer1

GenerateMember True

Modifiers Private

Interval

The frequency of Elapsed events in milliseconds.

Error List

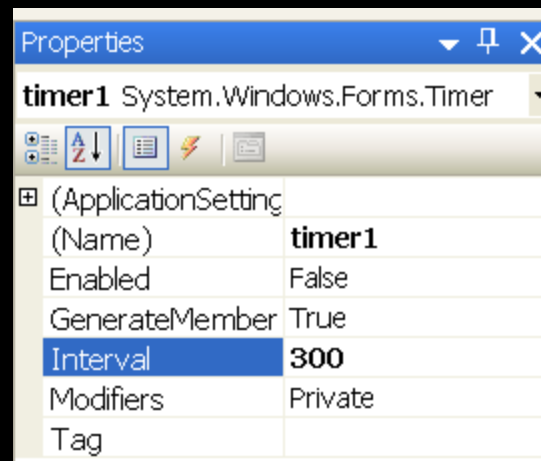
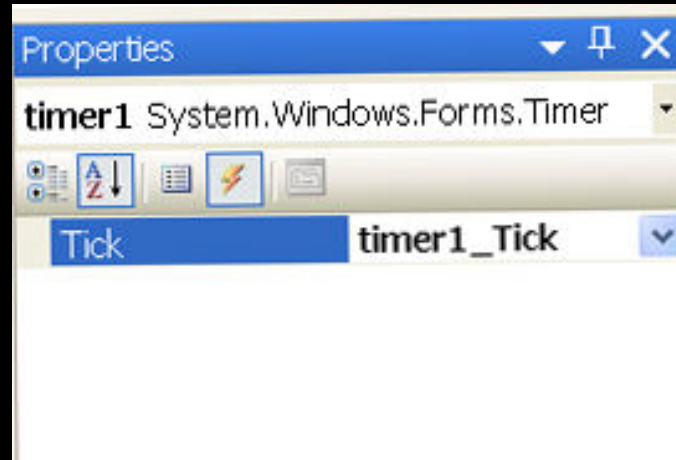
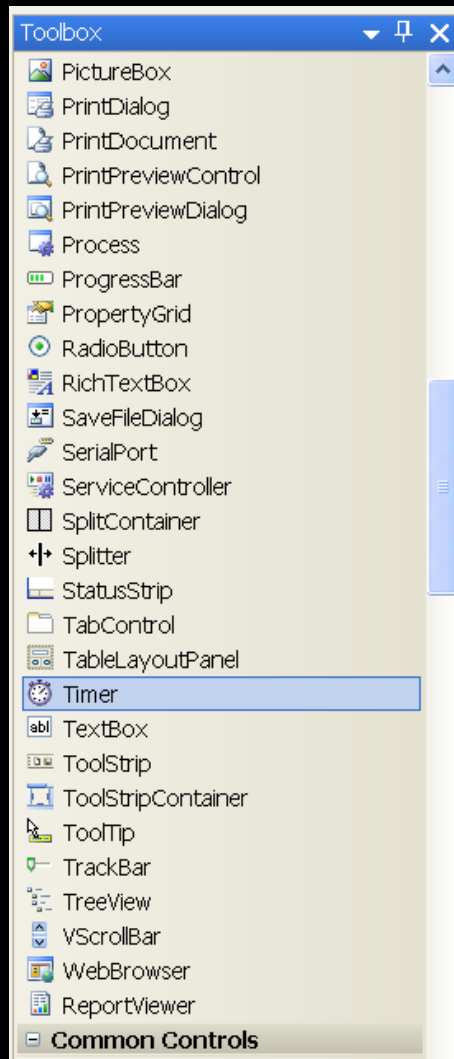
0 Errors 0 Warnings 0 Messages

Description File Line Colu... Project

Ready

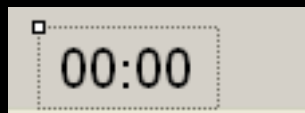
5:58 AM 4/2/2014

Примери с таймер

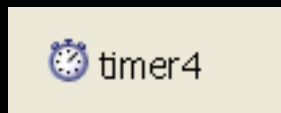


Използване на таймер за визуализация на дигитален часовник

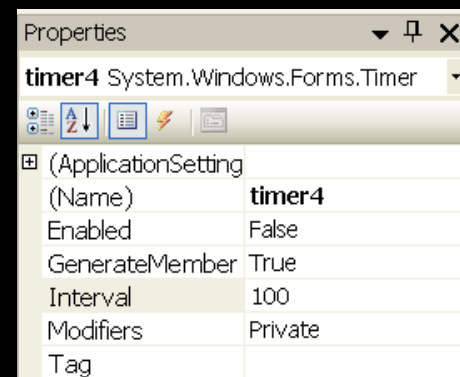
1. Добавяме контрол Label -



2. Добавяме контрол Timer



3. Задаваме интервал на Timer – 1000ms=1s
4. Определяме функционалността на таймера, която ще се изпълнява на всяка 1 секунда



```
private void timer4_Tick(object sender, EventArgs e)
{
    label1.Text = DateTime.Now.ToLongTimeString();
}
```

5. Стартираме Timer

```
timer4.Enabled = true;
```

Използване на таймер за анимация и SlideShow

```
private void timer2_Tick(object sender, EventArgs e)
{
    i++;
    if (i > 6)
    {
        i = 0;
    }
    pictureBox3.Image = Image.FromFile("d:\\TU_2010\\pic" + i.ToString() + ".bmp")
}
```

WindowsFormsApplication4 - Microsoft Visual Studio

Quick Launch (Ctrl+Q)

FILE EDIT VIEW PROJECT BUILD DEBUG TEAM SQL TOOLS TEST ARCHITECTURE ANALYZE WINDOW HELP

Form1.cs Object Browser Toolbox Form1.cs [Design] Program.cs

WindowsFormsApplication4.Form1

comboBox1_SelectedIndexChanged(object sender, EventArgs e)

```
public int i;
public Form1()
{
    InitializeComponent();
    timer1.Interval = 1000;
    timer1.Enabled = true;
}

private void timer1_Tick(object sender, EventArgs e)
{
    label1.Text = DateTime.Now.ToString();
    i++;
    if (i > 8) {
        i=1;
    }
    pictureBox1.Image = Image.FromFile (i + ".jpg" );
}

private void textBox1_TextChanged(object sender, EventArgs e)
{
}
```

100 %

Error List

0 Errors 0 Warnings 0 Messages

Search Error List

Description	File	Line	Column	Project
-------------	------	------	--------	---------

Ready Ln 54 Col 14 Ch 14 INS

6:11 AM 4/2/2014

Task ?

- Preview
- Next

Клавиатура -прихващане

```
private void txtInput_KeyDown(object sender, System.Windows.Forms.KeyEventArgs e)
{
    Console.WriteLine("KeyCode name: " + e.KeyCode);
    Console.WriteLine("KeyCode key code: " + ((int)e.KeyCode));
    Console.WriteLine("KeyData name: " + e.KeyData);
    Console.WriteLine("KeyData key code: " + ((int)e.KeyData));
    Console.WriteLine("KeyValue: " + e.KeyValue);
    Console.WriteLine("Handled: " + e.Handled);
}
```

```
private void txtInput_KeyPress(object sender, System.Windows.Forms.KeyPressE
ventArgs e)
{
    char keyChar;
    keyChar = e.KeyChar;

    Console.WriteLine("KeyPress event.");
    Console.WriteLine("KeyChar: " + keyChar);
    Console.WriteLine("KeyChar Code: " + (int)keyChar);
    Console.WriteLine("Handled: " + e.Handled);

    lblUpper.Text = keyChar.ToString().ToUpper();
    lblLower.Text = keyChar.ToString().ToLower();

    if (keyChar.ToString() == "$")
    {
        txtInput.AppendText("#");
        e.Handled = true;
    }
}
```



```
private void txtInput_KeyUp(object sender, System.Windows.Forms.  
KeyEventArgs e)  
{  
    Console.WriteLine("KeyCode name: " + e.KeyCode);  
    Console.WriteLine("KeyCode key code: " + ((int)e.KeyCode));  
    Console.WriteLine("KeyData name: " + e.KeyData);  
    Console.WriteLine("KeyData key code: " + ((int)e.KeyData));  
    Console.WriteLine("KeyValue: " + e.KeyValue);  
    Console.WriteLine("Handled: " + e.Handled);  
}
```

Таблица на Key Code

Symbolic constant name	Value (hexadecimal)	Keyboard (or mouse) equivalent
VK_LBUTTON	01	Left mouse button
VK_RBUTTON	02	Right mouse button
VK_CANCEL	03	Control-break processing
VK_MBUTTON	04	Middle mouse button (three-button mouse)
VK_BACK	08	BACKSPACE key
VK_TAB	09	TAB key
VK_CLEAR	0C	CLEAR key
VK_RETURN	0D	ENTER key
VK_SHIFT	10	SHIFT key
VK_CONTROL	11	CTRL key
VK_MENU	12	ALT key
VK_PAUSE	13	PAUSE key
VK_CAPITAL	14	CAPS LOCK key
VK_ESCAPE	1B	ESC key
VK_SPACE	20	SPACEBAR
VK_PRIOR	21	PAGE UP key
VK_NEXT	22	PAGE DOWN key
VK_END	23	END key
VK_HOME	24	HOME key
VK_LEFT	25	LEFT ARROW key
VK_UP	26	UP ARROW key
VK_RIGHT	27	RIGHT ARROW key
VK_DOWN	28	DOWN ARROW key
VK_SELECT	29	SELECT key
VK_PRINT	2A	PRINT key
VK_EXECUTE	2B	EXECUTE key
VK_SNAPSHOT	2C	PRINT SCREEN key
VK_INSERT	2D	INS key

WPF (Avalon),

- Windows App GUI
- Web App GUI
- HTML
- XML
- WPF
- LINQ

HTML- HyperText Markup Language

<form>

**First name: <input type="text" name="firstname" />
**

Last name: <input type="text" name="lastname" />

</form>

```
<form>
First name: <input type="text" name="firstname" /><br />
Last name: <input type="text" name="lastname" />
</form>
```

How the HTML code above looks in a browser:

First name:

Last name:

Document Structure

<html>

<head><title>My First Web Page</title>

</head>

<body bgcolor="white">

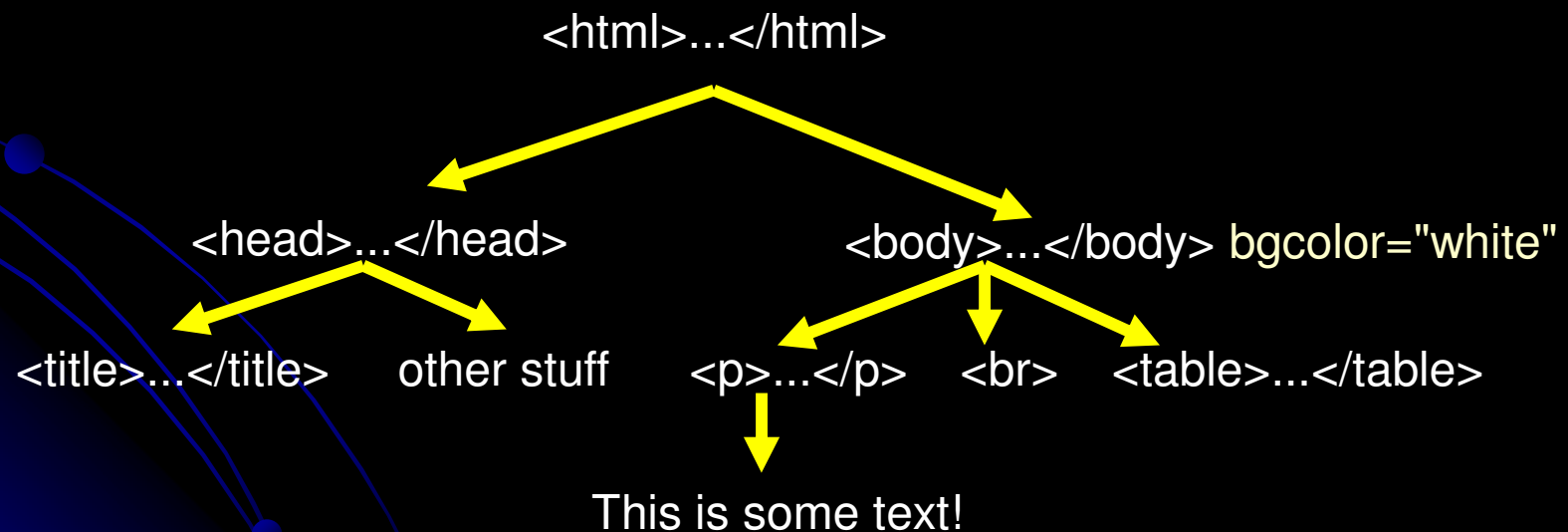
<p>A Paragraph of Text.</p>

</body>

</html>

Nested Tags

- Like a tree, each element is contained inside a parent element
- Each element may have any number of attributes



Basic Tags

<html>

<head><title>My First Web Page</title>

</head>

<body bgcolor="white">

<p>A Paragraph of Text.</p>

</body>

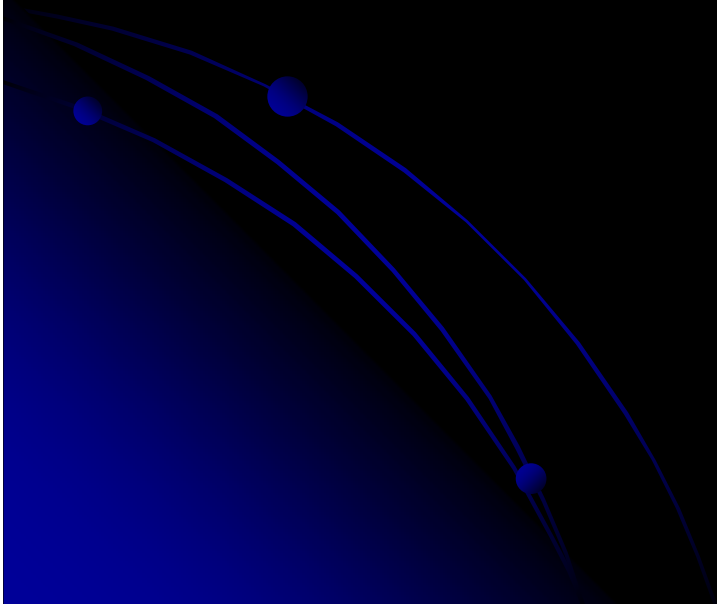
</html>

Basic Tags

- `<!doctype html public "-//w3c//dtd html 4.0 transitional//en">`
 - Preamble which identifies content as HTML
- `<h1>...</h1>`
 - H1-6 where larger number means smaller heading
- `<p>`
 - Includes vertical whitespace unlike `<br>`

Basic Tags

- `<hr>` horizontal rule
- `<br>` new line
- `<b>...</b>` bold
- `<i>...</i>` italicize text in between



Lists

- Unordered Lists

 Apples

 Oranges

- Ordered Lists

 One

 Two

- Can be nested

- 0 Apples

1. Fuji

2. Granny Smith

- 0 Oranges

Image Files

- ``
- JPEG
 - Best for photos
 - Public standard
- GIF
 - Best for simple images
 - Older standard
- PNG – Portable Network Graphics
 - Public standard replacement for GIF
- SVG – Scalable Vector Graphics
 - Series of drawing commands
 - Uses XML

Tables

- `<table>...</table>`
- `<tr>...</tr>` for each row
- `<td>...</td>` for each element in a row
- `<th>...</th>` for header row

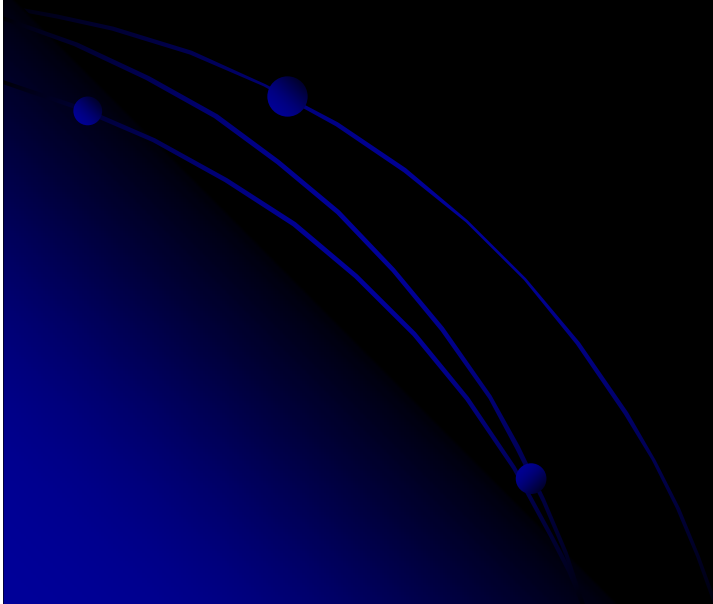


Table Example

```
<table border="1">
```

```
<tr>
```

```
<td>row 1, cell 1</td>
```

```
<td>row 1, cell 2</td>
```

```
</tr>
```

```
<tr>
```

```
<td>row 2, cell 1</td>
```

```
<td>row 2, cell 2</td>
```

```
</tr>
```

```
</table>
```

row 1, cell 1	row 1, cell 2
row 2, cell 1	row 2, cell 2

Comments

- <!-- This is a comment -->
- <!--

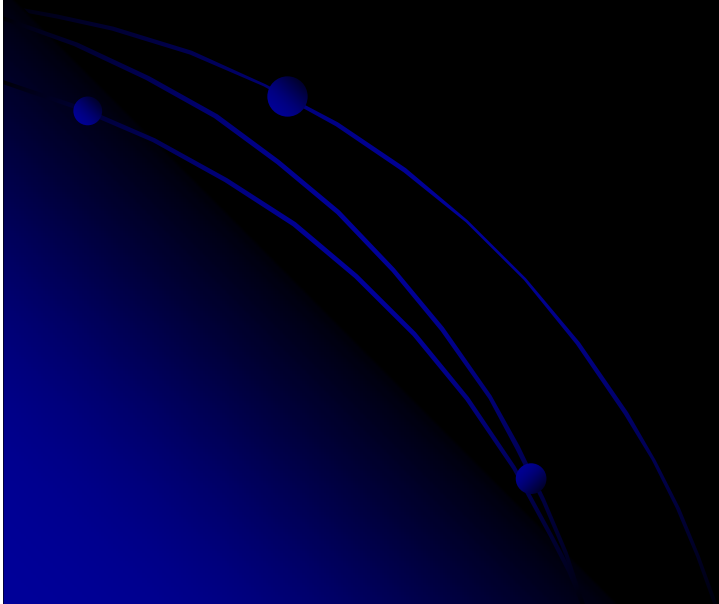
This paragraph,
is also a
comment...

-->



Special HTML

- `<` → `<`
- `>` → `>`
- `&` → `&`
- ` ` → space



Anchor Tag (Links)

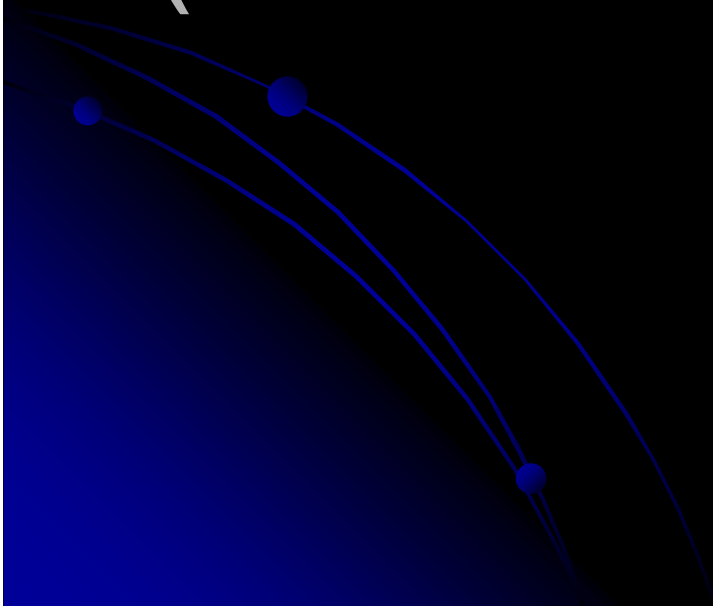
Absolute HREFs specify fully qualified URLs.

- `<a href="http://www.yahoo.com/">Yahoo!</a>`
- `<a href="reldoc.html">In this directory!</a>`
- `<a href="a/doc.html">In sub-directory a!</a>`

Relative HREFs are relative to the directory containing the current HTML file.

XML

(eXtensible Markup Language)



XML – пример

```
<?xml version="1.0"?>
<library name=".NET Developer's Library">
  <book>
    <title>Programming Microsoft .NET</title>
    <author>Jeff Prosise</author>
    <isbn>0-7356-1376-1</isbn>
  </book>
  <book>
    <title>Microsoft .NET for Programmers</title>
    <author>Fergal Grimes</author>
    <isbn>1-930110-19-7</isbn>
  </book>
</library>
```

XML – D

заглавна
част
(пролог)

атрибут

```
<?xml version="1.0"?>
```

```
<library name=".NET Developer's Library">
```

```
<book>
```

отварящ
таг

```
<title>Programming Microsoft .NET</title>
```

```
<author>Jeff Prosise</author>
```

```
<isbn>0-7356-1376-1</isbn>
```

```
</book>
```

елемент

```
<book>
```

```
<title>Microsoft .NET for Programmers</title>
```

затварящ
таг

```
<author>Fergal Grimes</author>
```

```
<isbn>1-930110-19-7</isbn>
```

```
</book>
```

стойност на елемент

```
</library>
```

XSD схеми

```
<xs:element name="book">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="title"/>
      <xs:element ref="author"/>
      <xs:element ref="isbn"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
<xs:element name="title" type="xs:string"/>
<xs:element name="author" type="xs:string"/>
<xs:element name="isbn" type="xs:string"/>
</xs:schema>
```

XAML - Extensible Application Markup Language

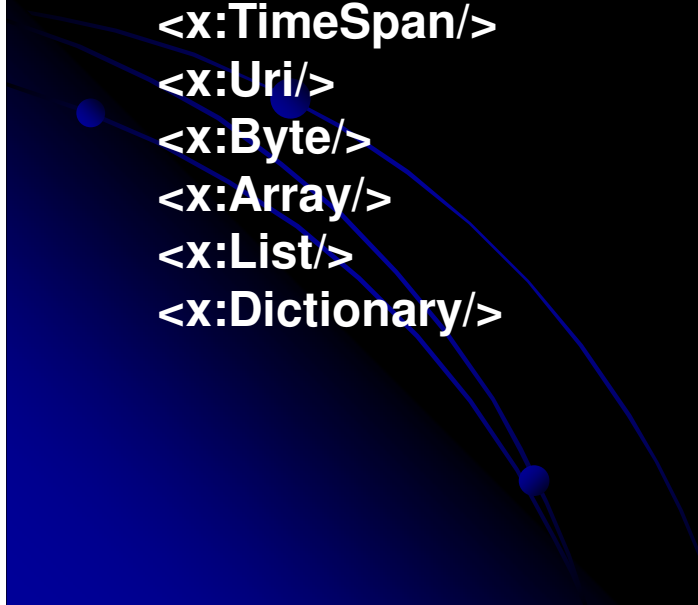
Introduction to XAML – XML

Advantages of XAML :

- XAML code is short and clear to read
- Separation of designer code and logic
- Graphical design tools like Expression Blend require XAML as source.

The following types are included into the XAML language:

**<x:Object/>
<x:Boolean/>
<x:Char/>
<x:String/>
<x:Decimal/>
<x:Single/>
<x:Double/>
<x:Int16/>
<x:Int32/>
<x:Int64/>
<x:TimeSpan/>
<x:Uri/>
<x:Byte/>
<x:Array/>
<x>List/>
<x:Dictionary/>**



Hello World

- C# example

```
using System;
using System.Windows;

namespace AvalonExample {
    class MyApp {
        [STAThread]
        static void Main() {
            MessageBox.Show("Hello World!");
        }
    }
}
```

Hello World

- XAML example

```
<Page  
xmlns="http://schemas.microsoft.com/winfx/avalon/2005"  
xmlns:x="http://schemas.microsoft.com/winfx/xaml/2005">  
    <TextBlock>Hello World!</TextBlock>  
</Page>
```


Application Object

- Application object acts as container for more complex applications
 - MainWindow
 - Application events like
 - Startup & Shutdown

```
public class MyApp : Application
{
    [STAThread]
    static void Main(string[] args)
    {
        MyApp app = new MyApp();
        app.Startup +=
app.OnApplicationStartup;
        app.Run(args);
    }
}
```

XAML vs. Code

As an example we build a simple StackPanel with a textblock and a button in XAML and compare it to the same code in C#.

```
<StackPanel>
    <TextBlock Margin="20">Welcome to the World of XAML</TextBlock>
    <Button Margin="10" HorizontalAlignment="Right">OK</Button>
</StackPanel>
```

The same expressed in C# will look like this:

```
// Create the StackPanel
StackPanel stackPanel = new StackPanel();
this.Content = stackPanel;

// Create the TextBlock
TextBlock textBlock = new TextBlock();
textBlock.Margin = new Thickness(10);
textBlock.Text = "Welcome to the World of XAML";
stackPanel.Children.Add(textBlock);

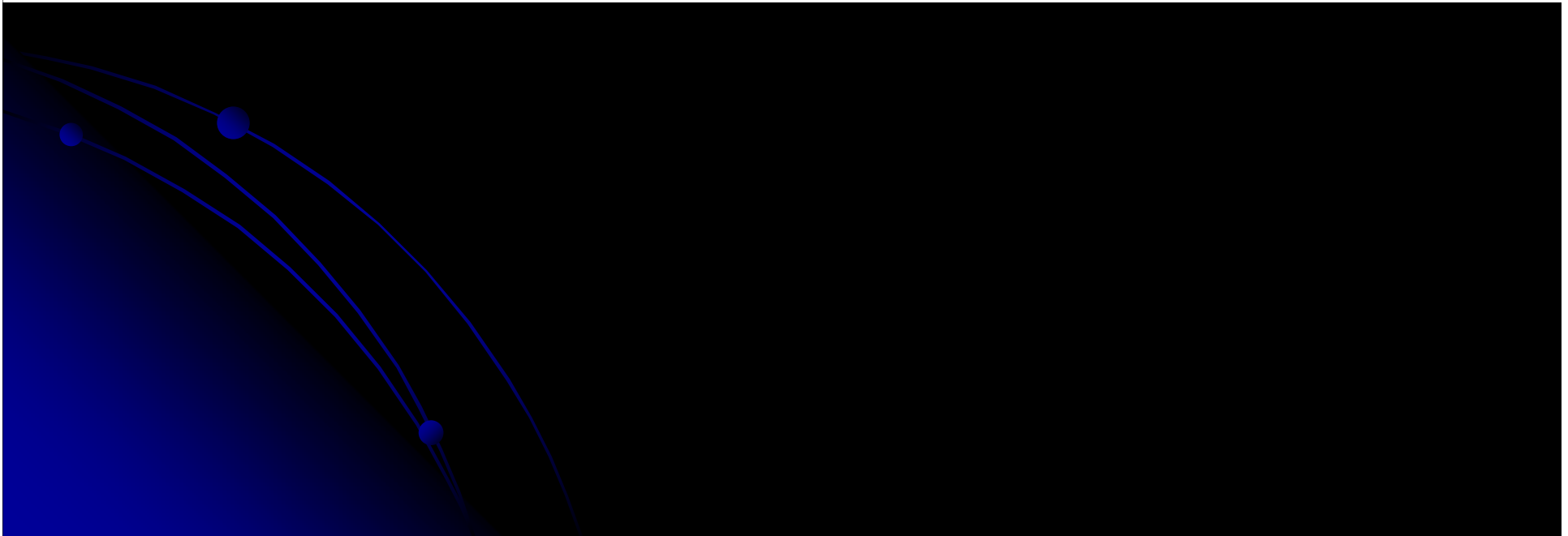
// Create the Button
Button button = new Button();
button.Margin = new Thickness(20);
button.Content = "OK";
stackPanel.Children.Add(button);
```

Properties as Elements

```
<Button>  
  <Button.Content>  
    <Image Source="Images/OK.png" Width="50" Height="50" />  
  </Button.Content>  
</Button>
```

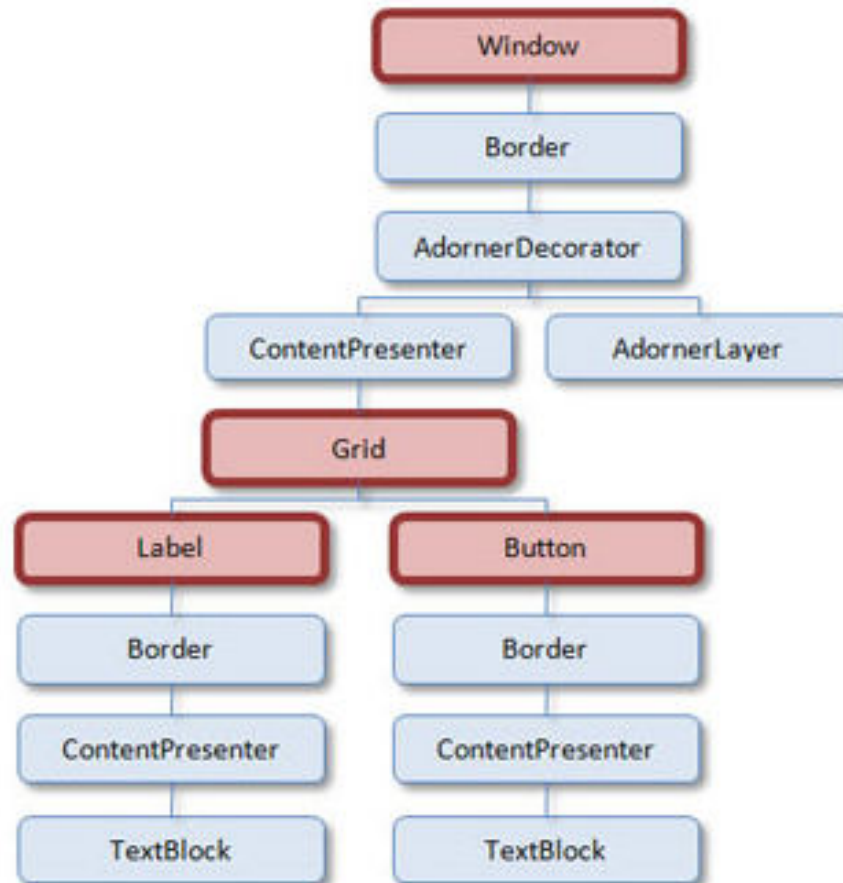
Implicit Type conversion

```
<Border BorderBrush="Blue" BorderThickness="0,10">  
</Border>
```



Logical- and Visual Tree

```
<Window>  
  <Grid>  
    <Label Content="Label" />  
    <Button Content="Button" />  
  </Grid>  
</Window>
```



The Logical Tree

The **logical tree** describes the relations between elements of the user interface. The logical tree is responsible for:

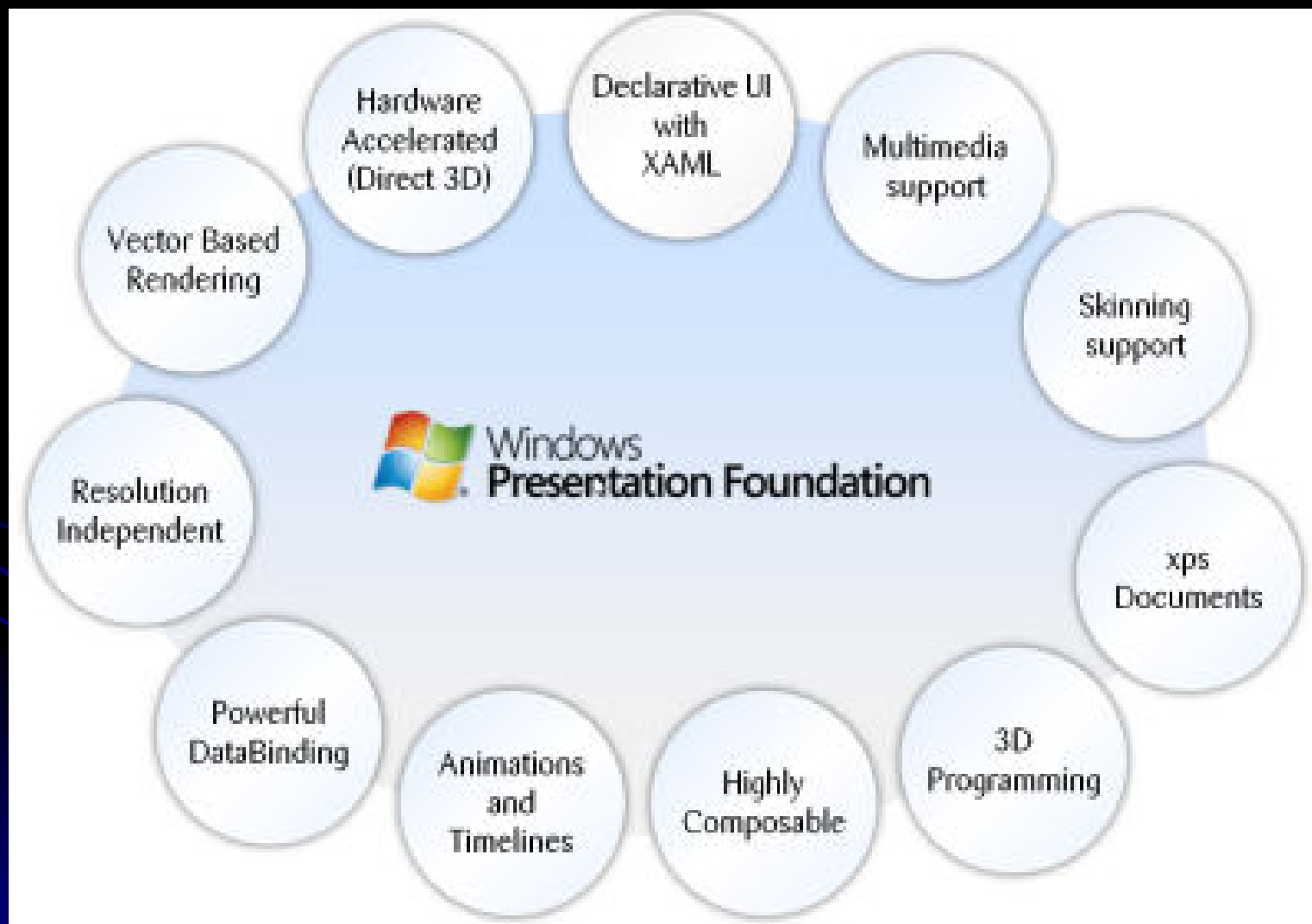
- Inherit `DependencyProperty` values
- Resolving `DynamicResources` references
- Looking up element names for bindings
- Forwarding `RoutedEvents`

The Visual Tree

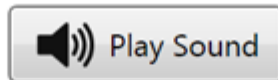
The **visual tree** contains all logical elements including all visual elements of the template of each element. The visual tree is responsible for:

- Rendering visual elements
- Propagate element opacity
- Propagate Layout- and `RenderTransforms`
- Propagate the `IsEnabled` property.
- Do Hit-Testing
- `RelativeSource (FindAncestor)`

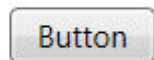
WPF



Rich composition

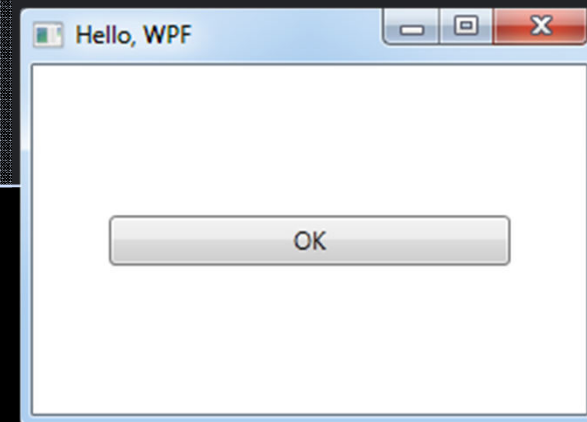


```
<Button>
  <StackPanel Orientation="Horizontal">
    <Image Source="speaker.png" Stretch="Uniform"/>
    <TextBlock Text="Play Sound" />
  </StackPanel>
</Button>
```

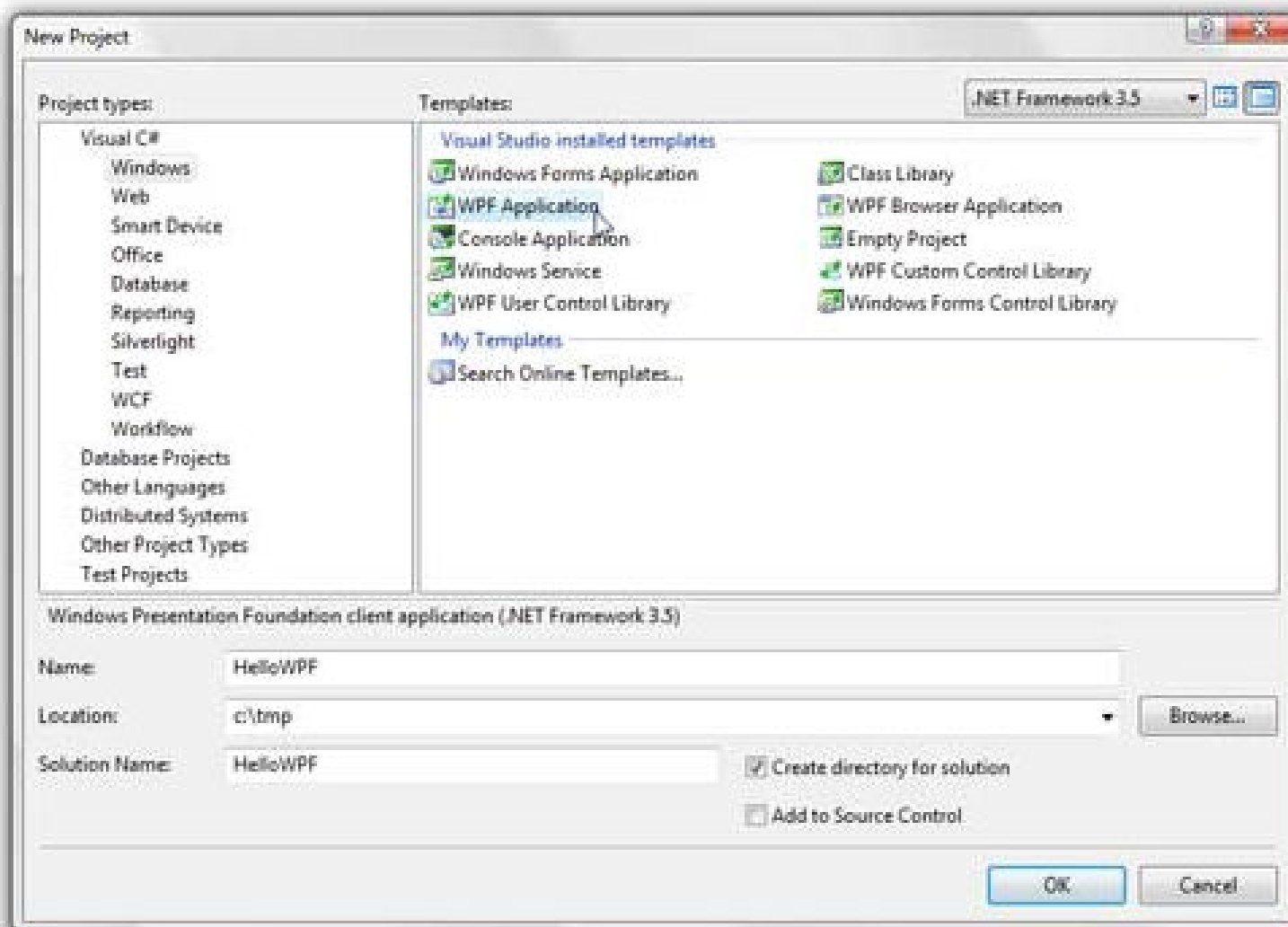


- ◆Constructs button with caption "OK"

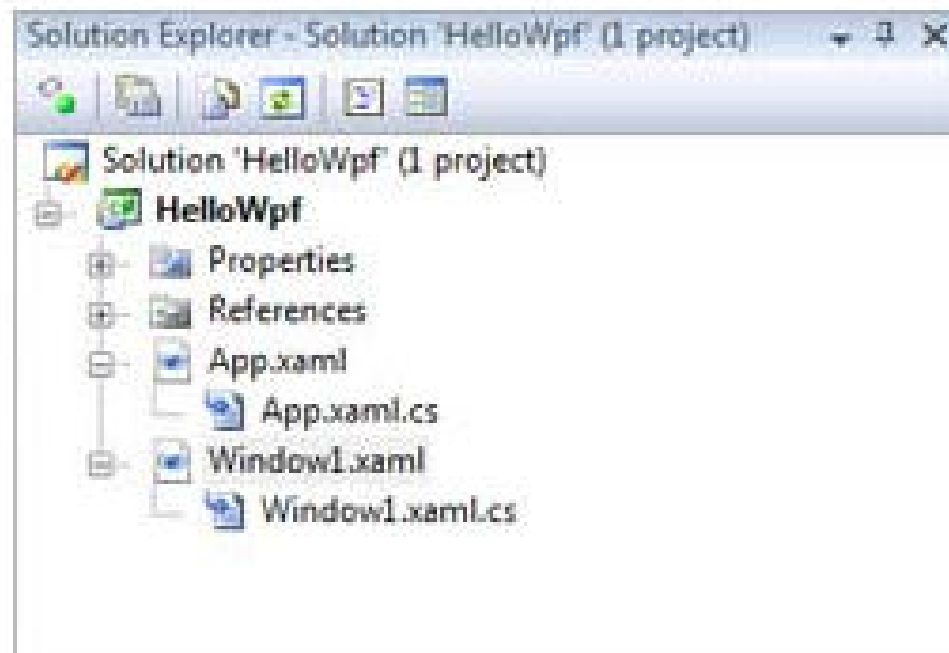
```
<Window x:Class="MyFirstWpfApp.MainWindow" xmlns="
  "http://schemas.microsoft.com/winfx/2006/xaml/presentation"
  xmlns:x=http://schemas.microsoft.com/winfx/2006/xaml
  Title="Hello, WPF" Height="300" Width="300">
  <Button x:Name="button"
    Width="200"
    Height="25">OK</Button>
</Window>
```



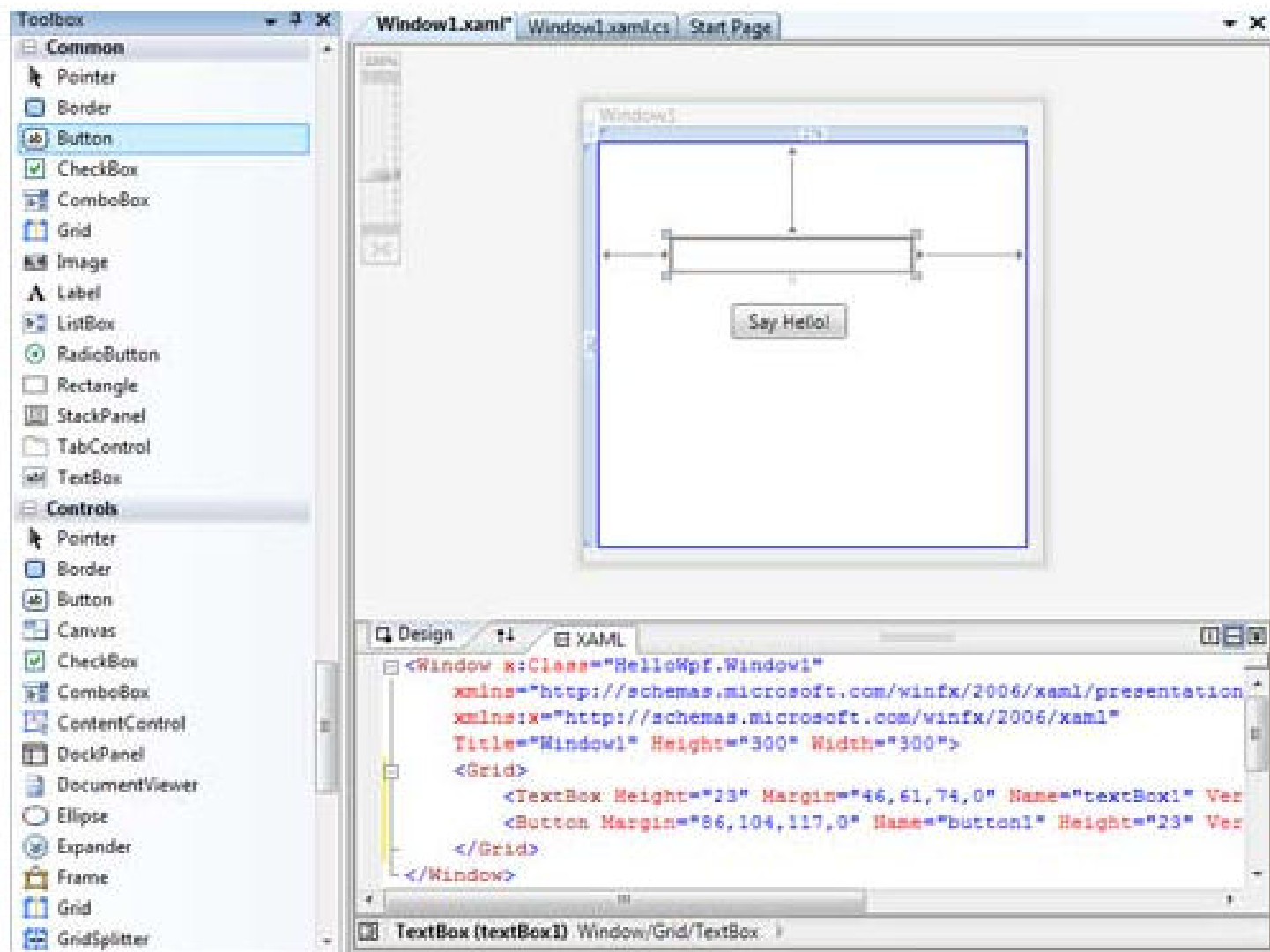
Example - VS



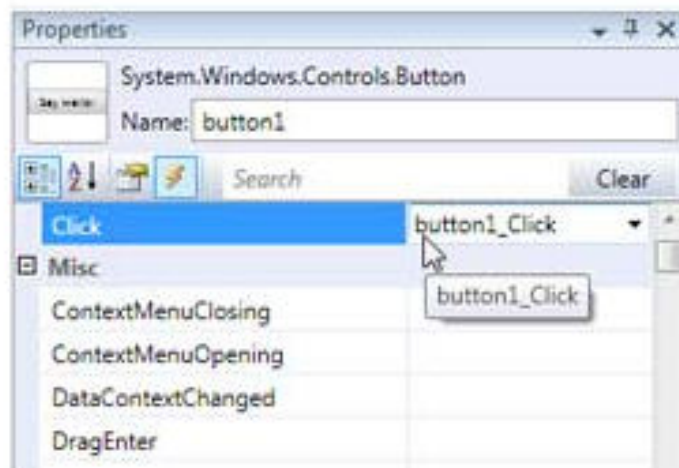
Step 1



Step 2



Step 3



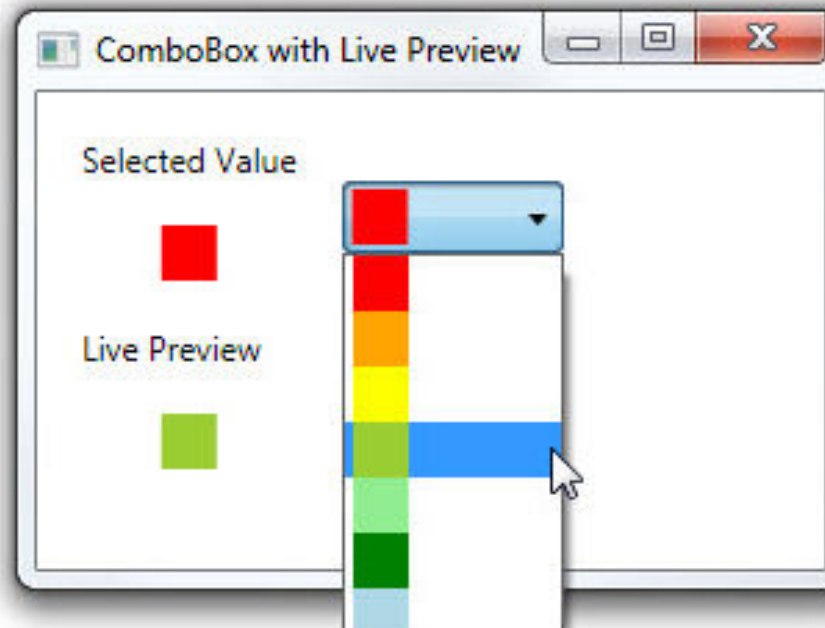
Visual Studio automatically creates a method in the code-behind file that gets called when the button is clicked.

```
private void button1_Click(object sender, RoutedEventArgs e)
{
    textBox1.Text = "Hello WPF!";
}
```

Step 4



ComboBox with Live Preview



```
<Window xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        xmlns:l="clr-namespace:LivePreviewComboBox">

    <StackPanel>
        <TextBlock Text="Preview Value:" />
        <TextBlock Text="{Binding LivePreviewItem, ElementName=liveBox}" />
        <l:LivePreviewComboBox x:Name="liveBox"/>
    </StackPanel>

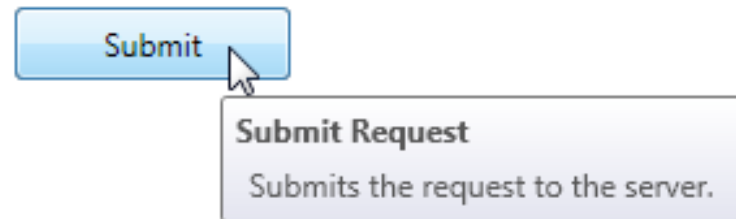
</Window>
```

Style



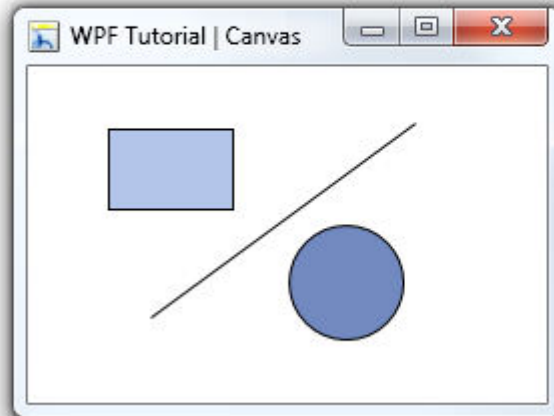
```
<StackPanel Orientation="Horizontal" VerticalAlignment="Top">  
  <Button Background="Orange" FontStyle="Italic"  
    Padding="8,4" Margin="4">Styles</Button>  
  <Button Background="Orange" FontStyle="Italic"  
    Padding="8,4" Margin="4">are</Button>  
  <Button Background="Orange" FontStyle="Italic"  
    Padding="8,4" Margin="4">cool</Button>  
</StackPanel>
```


ToolTips in WPF



```
<Button Content="Submit">
  <Button.ToolTip>
    <ToolTip>
      <StackPanel>
        <TextBlock FontWeight="Bold">Submit Request</TextBlock>
        <TextBlock>Submits the request to the server.</TextBlock>
      </StackPanel>
    </ToolTip>
  </Button.ToolTip>
</Button>
```

Canvas Panel



```
<Canvas>
  <Rectangle Canvas.Left="40" Canvas.Top="31" Width="63" Height="41" Fill="Blue" />
  <Ellipse Canvas.Left="130" Canvas.Top="79" Width="58" Height="58" Fill="Blue" />
  <Path Canvas.Left="61" Canvas.Top="28" Width="133" Height="98" Fill="Blue"
        Stretch="Fill" Data="M61,125 L193,28"/>
</Canvas>
```

Override the Z-Order of Elements

Normally the Z-Order of elements inside a canvas is specified by the order in XAML. But you can override the natural Z-Order by explicitly defining a `Canvas.ZIndex` on the element.



```
<Canvas>
  <Ellipse Fill="Green" Width="60" Height="60" Canvas.Left="30" Canvas.Top="20"
    Canvas.ZIndex="1"/>
  <Ellipse Fill="Blue" Width="60" Height="60" Canvas.Left="60" Canvas.Top="40"/>
</Canvas>
```

Selection Modes

The calendar control provides multiple modes for selection. You can set the `SelectionMode` property to `SingleDate`, `SingleRange`, `MultipleRanges` or `None`.

Selection Mode: MultipleRange ▾

März 2010						
Mo	Di	Mi	Do	Fr	Sa	So
22	23	24	25	26	27	28
1	2	3	4	5	6	7
8	9	10	11	12	13	14
15	16	17	18	19	20	21
22	23	24	25	26	27	28
29	30	31	1	2	3	4

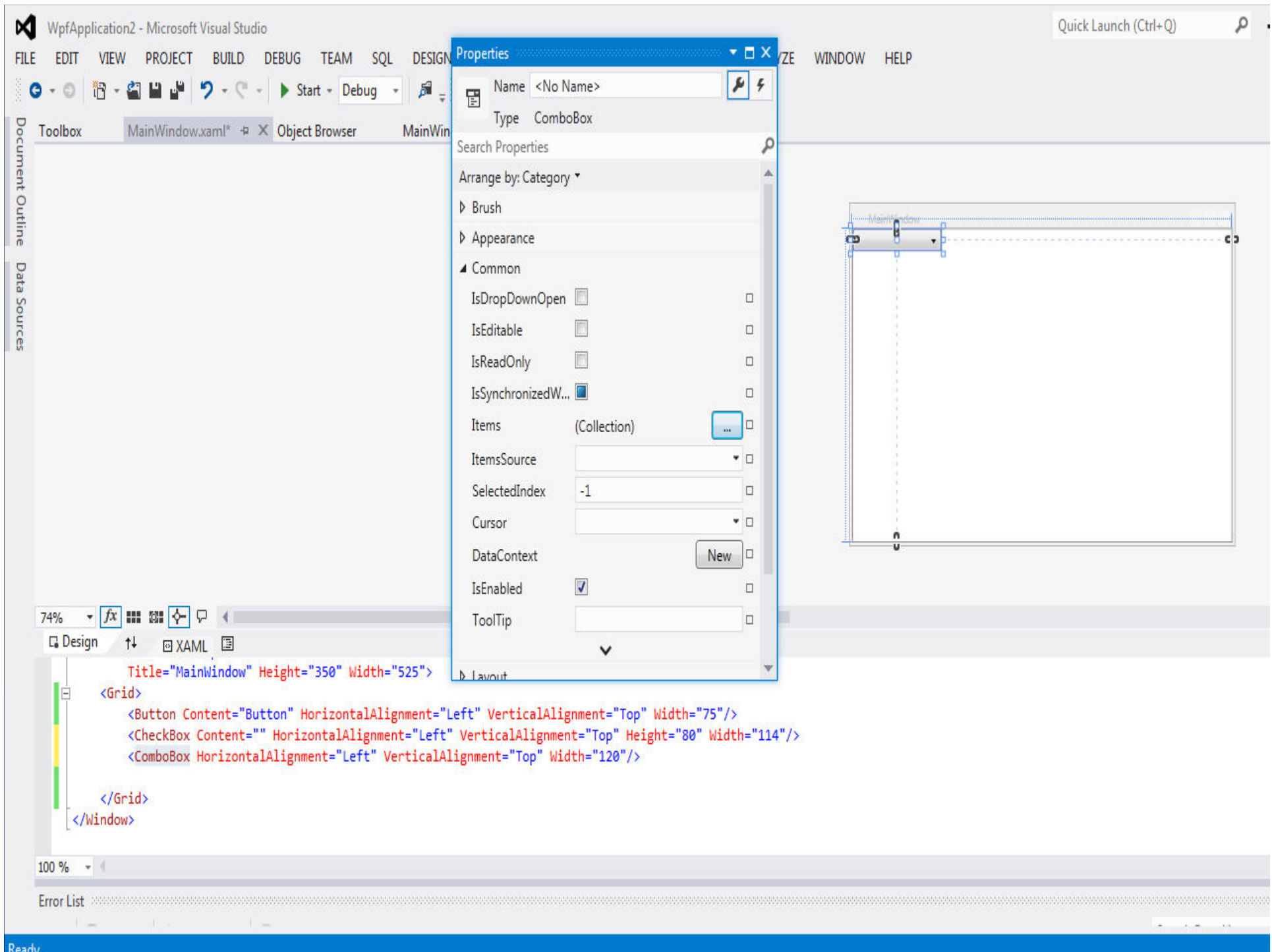
Blackout dates

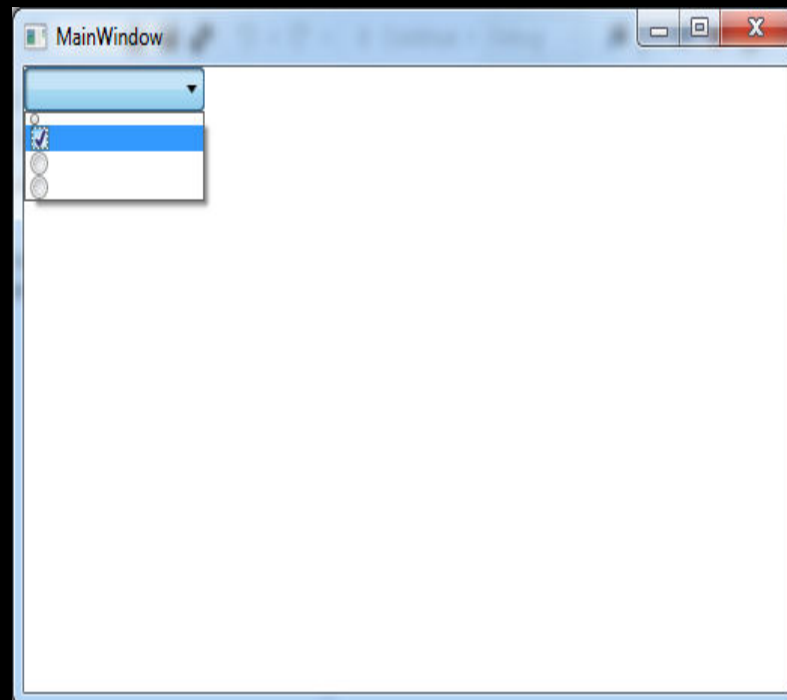
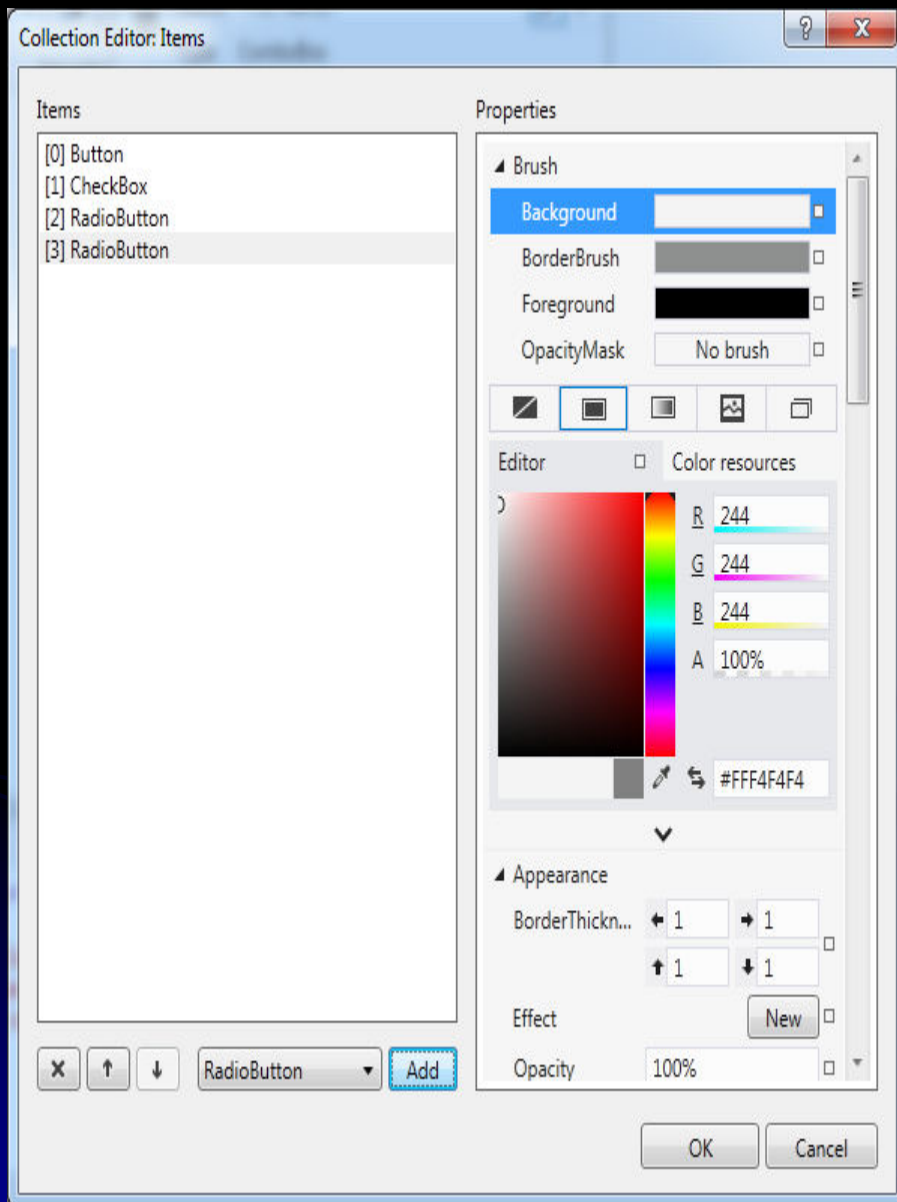
The calendar control provides a feature to black out setting the `BlackoutDates` property to one or m

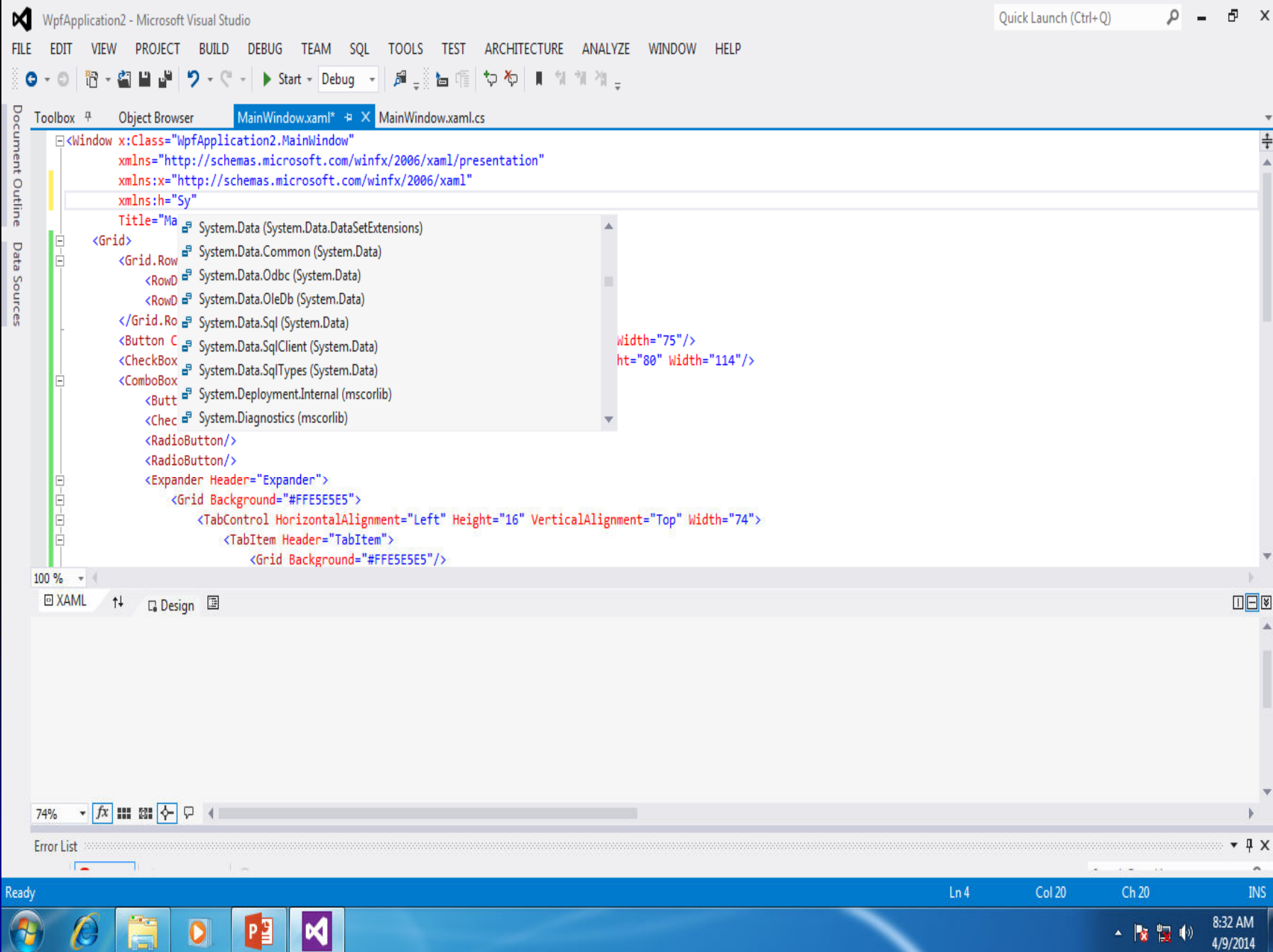
Januar 2010						
Mo	Di	Mi	Do	Fr	Sa	So
28	29	30	31	1	2	3
4	5	6	7	8	9	10
11	12	13	14	15	16	17
18	19	20	21	22	23	24
25	26	27	28	29	30	31
1	2	3	4	5	6	7

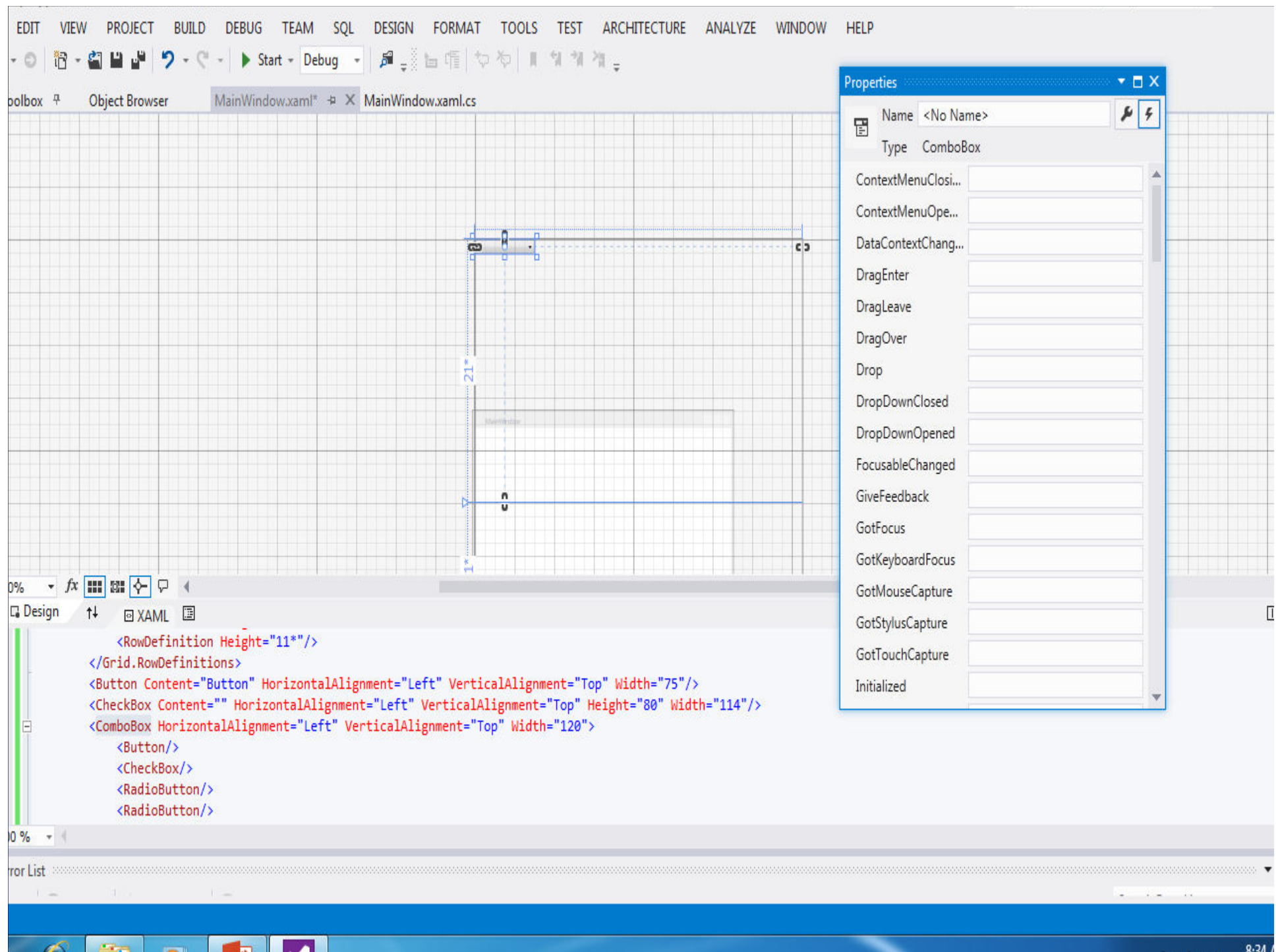
```
<Calendar SelectionMode="MultipleRange" />
```

```
<Calendar SelectionMode="{(Binding SelectedItem, ElementName=selectionmode)}" >
  <Calendar.BlackoutDates>
    <CalendarDateRange Start="01/01/2010" End="01/06/2010" />
    <CalendarDateRange Start="05/01/2010" End="05/03/2010" />
  </Calendar.BlackoutDates>
</Calendar>
```









http://www.wpftutorial.net



Christian Mosers

WPF Tutorial.net



Home



Share...



Feed

About

Home

WPF Inspector

Getting started ▶

Fundamentals ▶

Patterns ▶

User Experience ▶

Controls ▶

Custom Controls ▶

Layout ▶

Input ▶

Data Binding ▶

Templates and Styles ▶

Welcome to the WPF Tutorial



Tip of the Day

Setting a local value on a dependency property clears an eventually set binding. Use `SetCurrentValue()` in .NET 4.0 to work around this issue.



Popular Articles

[Learn WPF in two Weeks](#)

[Introduction to WPF](#)

[Create a simple WPF application](#)

LINQ

LINQ – Language INtegrated Query
(заявки вградени в езика)

Пример LINQ

```
using System;
using System.Query;
using System.Collections.Generic;

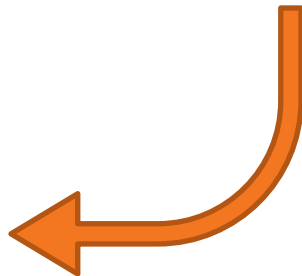
class app {
    static void Main() {

        string[] names = { "Burke", "Connor",
                           "Frank", "Everett",
                           "Albert", "George",
                           "Harris", "David" };

        var expr = from s in names
                    where s.Length == 5
                    orderby s
                    select s.ToUpper();

        foreach (string item in expr)
            Console.WriteLine(item);
    }
}
```

BURKE
DAVID
FRANK



Основни фундаменти на LINQ

- LINQ - използвани езици

C# 3.0

VB 9

- Особености

- Lambda Expressions
- Query Expressions

- Delegate functions
- Type inference
- Anonymous types
- Extension methods
- Expression trees

- private void FilterWithCSharp2()
- }
- string[] contacts = { "Gogo", "George", "Trifon", "Kiril", "Shumi" };
- //display result
- foreach (string name in FilterResults2(contacts))
- {
- Console.WriteLine(name);
- }
- {
- {
- private IEnumerable FilterResults2(string[] Data)
- {
- //filter array
- foreach (string name in Data)
- {
- if (name.StartsWith("G"))
- return name;
- {
- {

```
private static void FilterWithCSharp3() {  
    string[] contacts = { "Galcho", "George", "Trifon", "Kiril",  
        "Shumi" };  
    var result = from s in contacts  
        where s.StartsWith("G")  
select s;  
    //display result  
    foreach (string name in result) {  
        Console.WriteLine(name);  
    }  
}
```

```
private static void FilterWithCSharp3_2() {  
    string[] contacts = { " Gogo ", "George", "Trifon", "Kiril", "Shumi" };  
    var result = contacts.Where<string>(x => x.StartsWith("G"));  
    //display result  
    foreach (string name in result) {  
        Console.WriteLine(name);  
    }  
}
```

Подразбиращи се типове

- Declare Variable with “var” keyword
- Compiler infers correct type
 - Based on initialization

```
Customer c = new Customer(“Bob”, “Smith”, 1234);  
var c = new Customer(“Bob”, “Smith”, 1234);
```

- Only for local, (non-null) initialized variables

```
var c;           // No  
var c = null;    // No  
var c = default(string); // Yes  
public var DoThis(int x){} // No  
public void DoThis(var x){} // No
```

Инициализация на обекти

```
Invoice i = new Invoice { CustomerId = 123, Name = "Test" };
```

Is equivalent to:

```
Invoice I = new Invoice();  
i.CustomerId = 123;  
i.Name = "Test";
```


Anonymous

- Object Initializers Can be Used w/o a Type
- Result is an *Anonymous Type*

```
Invoice i = new Invoice { CustomerId = 123, Name = "SmithCo" };  
var i2 = new Invoice { CustomerId = 123, Name = "SmithCo" };  
var i3 = new {CustomerId = 123, Name = "SmithCo" };  
var i4 = new {123, "SmithCo"};
```

- Class Generated Under the Covers
- Frequently Utilized by LINQ

Lambda Expressions

Predicates

- Predicate
 - $(p) \Rightarrow p.\text{Gender} == \text{"F"}$
- Projection
 - $(p) \Rightarrow p.\text{Gender} ? \text{"F"} : \text{"Female"}$
 - “Each person p becomes string “Female” if Gender is “F””

Query Expressions

- Introduce SQL-Like Syntax to Language
- Compiled to Traditional C# (via Extension Methods)

```
from itemName in srcExpr
join itemName in srcExpr on keyExpr equals keyExpr
    (into itemName)?
let itemName = selExpr
where predExpr
orderby (keyExpr (ascending | descending)?) *
select selExpr
group selExpr by keyExpr
into itemName query-body
```

The LINQ Project

C# 3.0

Visual Basic 9.0

Others

.NET Language Integrated Query

LINQ to
Objects

LINQ to
DataSets

LINQ to
SQL

LINQ to
Entities

LINQ to
XML



Objects



Relational

```
<book>  
  <title/>  
  
  <author/>  
    <year/>  
    <price/>  
</book>
```

XML

LINQ to Objects

C# 3.0

Visual Basic 9.0

Others

.NET Language Integrated Query

LINQ to
Objects

LINQ to
DataSets

LINQ to
SQL

LINQ to
Entities

LINQ to
XML



Objects



Relational

```
<book>  
  <title/>  
  
  <author/>  
  <year/>  
  <price/>  
</book>
```

XML

LINQ to Objects

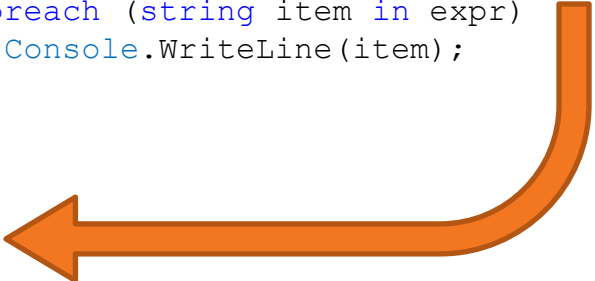
- Native query syntax in C# and VB
- IntelliSense
- Autocompletion
- Query Operators can be used against any .NET collection (IEnumerable<T>)
- Select, Where, GroupBy, Join, etc.
- Deferred Query Evaluation
- Lambda Expressions

```
using System;
using System.Query;
using System.Collections.Generic;

class app {
    static void Main() {
        string[] names = { "Burke", "Connor",
                           "Frank", "Everett",
                           "Albert", "George",
                           "Harris", "David" };
        Func<string, bool> filter = s => s.Length == 5;
        Func<string, string> extract = s => s;
        Func<string, string> project = s => s.ToUpper();

        IEnumerable<string> expr = names
            .Where(filter)
            .OrderBy(extract)
            .Select(project);

        foreach (string item in expr)
            Console.WriteLine(item);
    }
}
```



BURKE
DAVID
FRANK

LINQ to SQL

C# 3.0

Visual Basic 9.0

Others

.NET Language Integrated Query

LINQ to
Objects

LINQ to
DataSets

LINQ to
SQL

LINQ to
Entities

LINQ to
XML



Objects



Relational

```
<book>
  <title/>

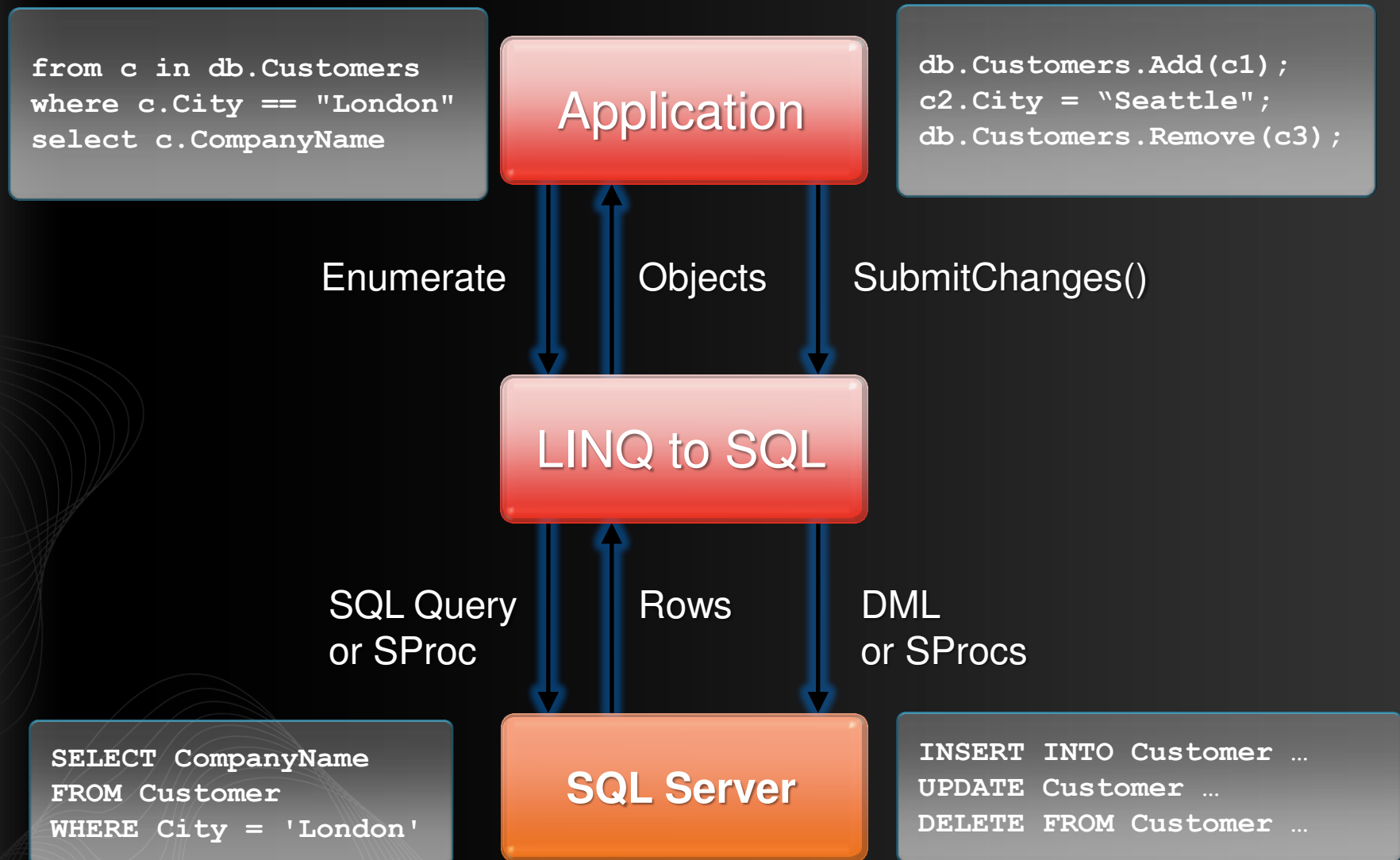
  <author/>
  <year/>
  <price/>
</book>
```

XML

LINQ to SQL Overview

- ORM Designer
 - Maps Relational Structures to Classes
- Delay Load for Expensive Values
 - Can disable universally
 - Enable for specific items
(*DataShape* applied to *DataContext*)
- Tables Have CRUD Definitions (Default = use runtime)
 - Can instead point to stored procedures

LINQ to SQL Architecture



The *DataContext* Class

- At First, *Seems* Similar to Connection object from ADO.NET
 - Queries
 - Commands
 - Transactions
- But Does Much More:
 - Accessing objects in the object-relational framework
 - Base for derived database specialization (automated by gen tools)
 - Adds Caching and Tracking
 - And More

LINQ to XML

C# 3.0

Visual Basic 9.0

Others

.NET Language Integrated Query

LINQ to
Objects

LINQ to
DataSets

LINQ to
SQL

LINQ to
Entities

LINQ to
XML



Objects



Relational

```
<book>
  <title/>
  <author/>
  <year/>
  <price/>
</book>
```

XML

LINQ to XML

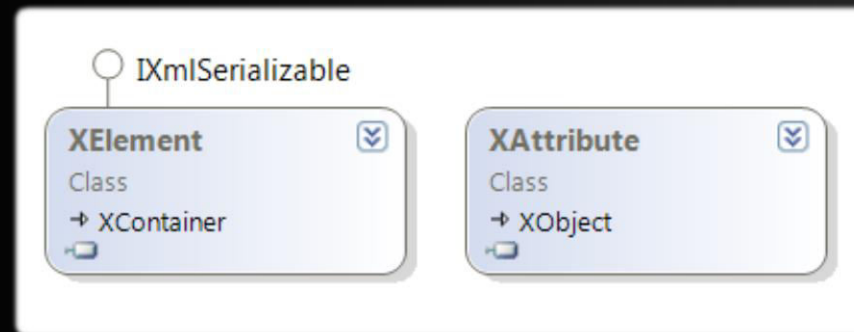
- Large Improvement Over Existing Model
- Supports:
 - Creating XML
 - Loading & querying XML
 - Modifying & saving XML
 - Streaming, Schema, Annotations, Events

LINQ to XML

- New XML API implemented in v3.5 assembly
 - `System.Xml.Linq.dll`
- Namespaces
 - `System.Xml.Linq`
 - `System.Xml.Schema`
 - `System.Xml.XPath`
- Can be used independently of LINQ

Key Classes in System.Xml.Linq

- System.Xml.Linq is a “DOM like” API
 - Manipulates an XML tree in memory
- Naturally work with both XML **documents** and **fragments**
- The two key classes in System.Xml.Linq



Loading Xml Content

- Loading Xml is performed with;
 - **XElement.Load**
 - **XDocument.Load**
- Both support loading from
 - **URI, XmlReader, TextReader**

```
XmlReader reader = XmlReader.Create("myData.xml");  
XElement element = XElement.Load(reader);
```

Modifying XML

- XML tree exposed by **XElement** et al. is modifiable
- Modifications through methods such as:
 - **XElement.Add()**
 - **XElement.Remove()**
 - **XElement.ReplaceWith()**
- Modified tree can be persisted via
 - **XElement.Save(), XDocument.Save()**
 - Both supporting filename, TextWriter, XmlWriter.

```
XElement element = new XElement("foo");  
element.Save(@"c:\temp\foo.xml");|
```


Creating an XML Document

```
XNamespace ns = "http://example.books.com";
XDocument books = new XDocument(
    new XElement(ns + "bookstore",
        new XElement(ns + "book",
            new XAttribute("ISBN", isbn),
            new XElement(ns + "title", "ASP.NET Book"),
            new XElement(ns + "author",
                new XElement(ns + "first-name", a.FirstName),
                new XElement(ns + "last-name", a.LastName)
            )
        )
    );

books.Save(@"C:\Books.xml");
```

VB 9: XML Literals

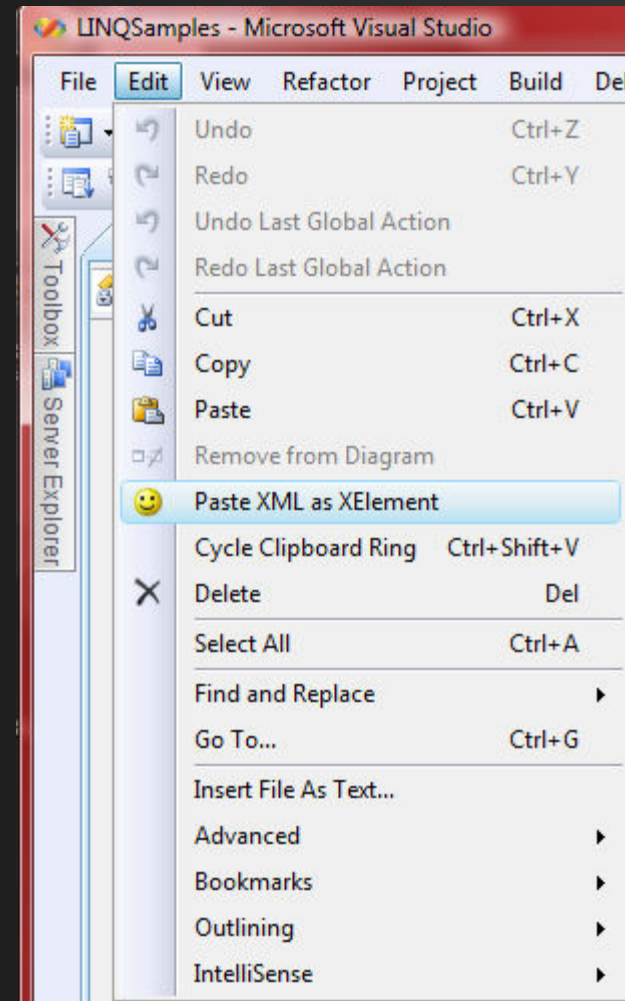
- XML Can Be Used Inline
- Compiled to *XElement* Expression
- Can Be Code-Driven and Dynamic
 - Including LINQ Expressions

```
Dim books = <bookstore xmlns="http://examples.books.com">
    <book ISBN=<%= isbn %>>
        <title>ASP.NET Book</title>
        <author>
            <first-name><%= a.FirstName %></first-name>
            <last-name><%= a.LastName %></last-name>
        </author>
    </book>
</bookstore>

books.Save("C:\Books.xml");
```

...Meanwhile in C#

- No XML Literals, But There's Something to Close the Gap
- “Paste XML as XElement” Add-in
 - Add XML to Clipboard
 - Edit -> Past XML as XElement
- Included in VS2008 Samples
 - Help -> Samples



LINQ to Entities

C# 3.0

Visual Basic 9.0

Others

.NET Language Integrated Query

LINQ to
Objects

LINQ to
DataSets

LINQ to
SQL

LINQ to
Entities

LINQ to
XML



Objects



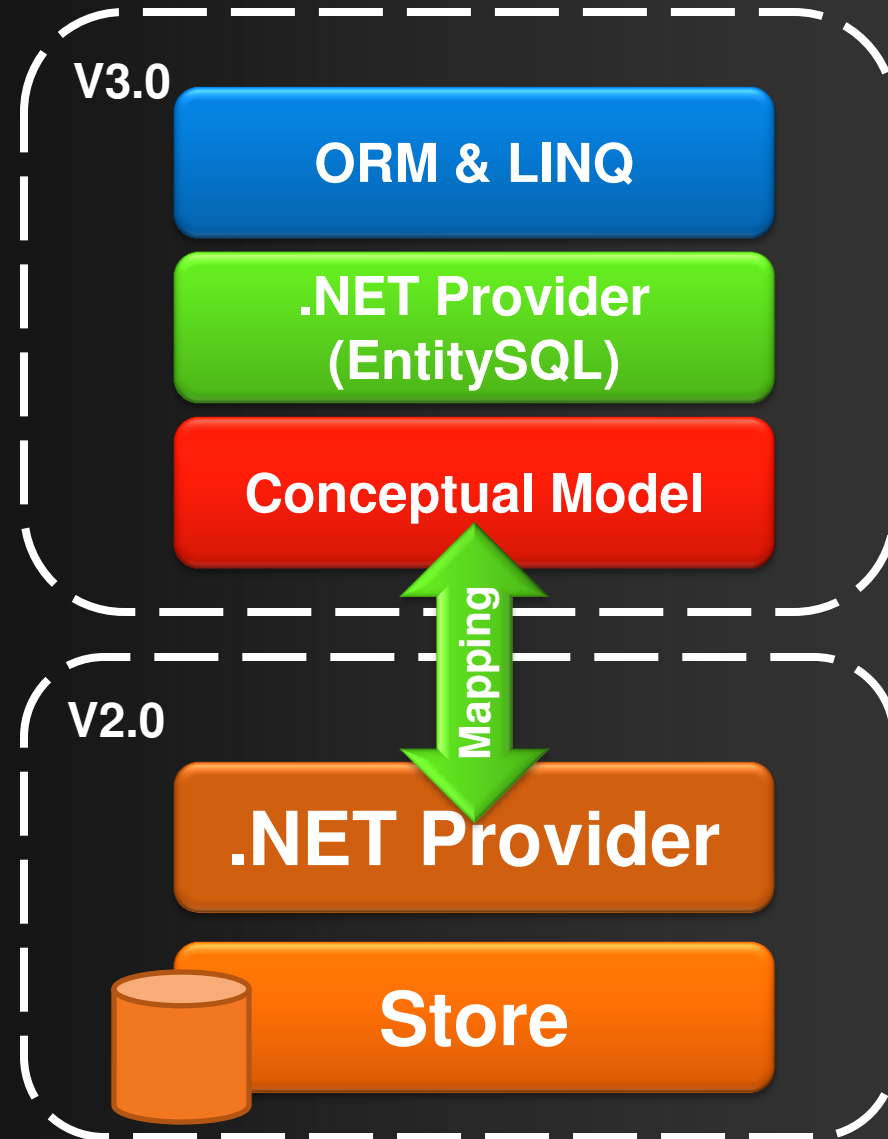
Relational

```
<book>  
  <title/>  
  
  <author/>  
    <year/>  
    <price/>  
</book>
```

XML

ADO.NET Entity Framework

- Schema and Store independence
- Higher-Level Constructs
 - Relationships
 - Inheritance
- Functionality
 - Flexibility of Entity Framework model & mapping
 - Productivity of LINQ
- Ships post-VS 2008
 - At Beta 2 Today
 - Tooling at CTP 1



LINQ to Entities

- New Data Access API implemented in assembly
 - `System.Data.Entity.dll`
 - `System.Data.Entity.Design.dll`
- Namespaces
 - `System.Data.Entity`
 - `System.Data.Objects`
 - and more...

LINQ to Entities Example

```
using(OrderTracking orderTracking = new OrderTracking()) {  
    var orders = from order in orderTracking.SalesOrders  
                  where order.Status == "Pending Stock Verification" &&  
                        order.SalesPerson.State == "WA"  
                  select order;  
  
    foreach(SalesOrder order in orders) {  
        List<StockAppProduct> products = new List<StockAppProduct>  
        ( from orderLine in order.Lines  
          select new StockAppProduct {  
              ProductID = orderLine.Product.ID,  
              LocatorCode = ComputeLocatorCode(orderLine.Product) } );  
  
        if(StockApp.CheckAvailability(products)) {  
            order.Status = "Shippable";  
        }  
    }  
    orderTracking.SaveChanges();  
}
```

LINQ

- A C# language extension:
 - Use SQL-like syntax in C#.
- Outline:
 - Examples
 - Understanding the witchcraft
 - Delegate functions
 - Lambda expressions
 - Type inference
 - Anonymous types
 - Extension methods
 - Expression trees

Searching in Collections

- Begin with a simple array of, say, Customers.

```
Customer[] customers = new Customer[30];  
customers[0] = new Customer(...);  
...  
customers[29] = new Customer(...);
```

Searching in Collections: The Old Way

- Find the names of all London customers:

```
List<string> londoners = new List<string>();
```

```
foreach (Customer c in customers) {  
    if (c.City == "London") {  
        londoners.add(c.Name);  
    }  
}
```

Searching in Collections: The LINQ Way

```
string[] londoners =  
    from c in customers  
    where c.City == "London"  
    select c.Name;
```

No loops!

SQL-like!

Declarative!

LINQ: How Does It Work?

- LINQ syntax = shorthand for method invocation.
- “Translation maps”

Syntax Translation Example

```
string[] londoners =  
    from c in customers  
    where c.City == "London"  
    select c.Name;
```

```
string[] londoners =  
    customers.  
    Where(expression) .  
    Select(expression) ;
```



Expressions == Methods?

- `Where()` wants a Boolean *method*.
- The method acts as a *filter*.
- Likewise for `Select()`: a *translation* method.

Lambda Expressions

- Lambda expression syntax:

(argumentList) => expression
oneArgument => expression



Shades of ML...

- Arguments *optionally* typed.
 - Type inference mechanism.
 - More on that later...

Where's Where () ?

- We invoked `Where ()` on `Customers []`.
- On the resulting `Customers []`, we invoked `Select ()`.
- New methods for arrays!?

Extension Methods

```
class Utils {  
    public static firstChar(this string s)  
    {  
        return s.charAt(0);  
    }  
}
```

- So far, just a simple static method.
- Can be used like any other.

Extension Methods

- But now...

```
Using Utils;
```

```
class Demo {  
    void Foo() {  
        string s = "Hello";  
        Console.WriteLine(s.firstChar());  
    }  
}
```

Extension Methods

- Static methods that *seem to* extend existing types.
- `Where()`, `Select()`, etc. `extend IEnumerable<T>`.
 - Defined in `System.Query`.
- Applicable to one-dimensional array types.

Query Your Own Types!

- LINQ can be applied to *any* type.
- Just implement `Where()`, `Select()`, etc.

LINQ and Relational Data

- Let's obtain a DB-table type, and query it.

```
DbCustomers c = new DbCustomers("my.mdb");  
string[] londoners =  
    from c in customers  
    where c.City == "London"  
    select c.Name;
```

This Makes No Sense!

- But... `Where()` applies the filter to every record.
- ... on the client!
- `SELECT * FROM CUSTOMERS`, and filter with a simple loop!?

Back To Lambda Expressions

- Lambda expressions can be converted to anonymous methods.
- Can *also* be converted to ***expression trees***.
 - A run-time representation of the syntax tree.

Example...

- Our code yields:

```
string[] londoners = customers.  
    Where(c => c.City == "London").  
    Select(c => c.Name);
```

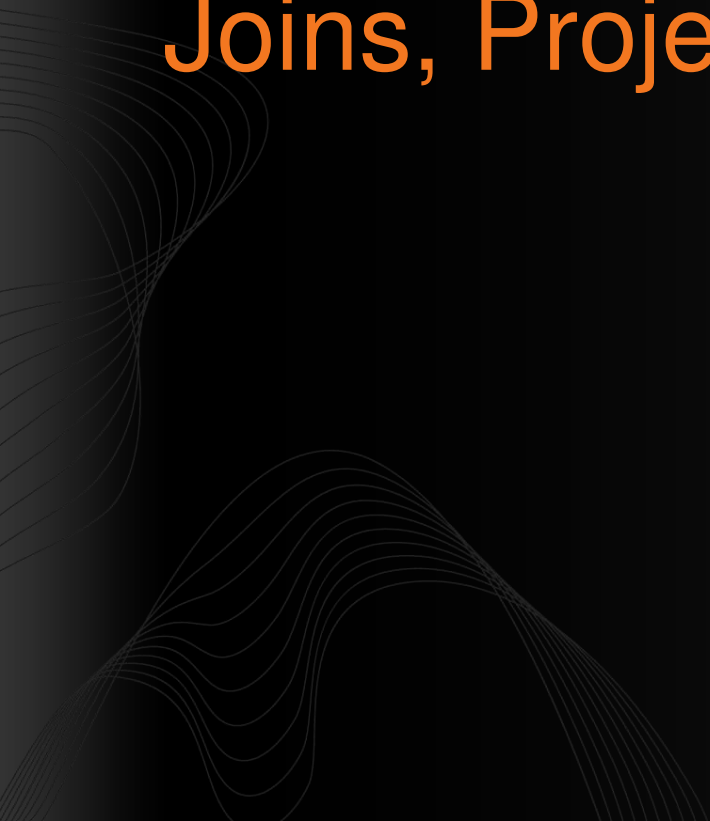
where “customers” is of type DbCustomers.

- No
DbCustomers.Where(delegate(Customer c)) method exists.
- However:
DbCustomers.Where(
 Expression<Func<Customer,bool>> xt)

Expression Trees

- *A data type.*
- Represents lambda expressions at runtime.
- Used it to generate SQL at runtime.
 - Guaranteed to be valid.

Relational Algebra: Joins, Projections



Multiple Generators (Cartesian Product / Join)

```
OrderData[] od = from c in customers
    where c.City == "London"
    from o in c.Orders
    where o.OrderDate.Year == 2005
    select new OrderData(c.Name, o.OrderId, o.Total);
```

```
OrderData[] od = customers.
    Where(c => c.City == "London").
    SelectMany(c =>
        c.Orders.
        Where(o => o.OrderDate.Year == 2005).
        Select(o => new OrderData(c.Name,
                                o.OrderId, o.Total))
    );
```



Projections

- Using LINQ's **select**:

```
from c in customers
where c.City == "London"
select
    new AddressBookEntry(c.Name, c.Phone);
```

Ad-Hoc Types are Nameless

- How do we store the result?

```
??? q = from ... select new {...};
```

- The ad-hoc type is *nameless*!
- Can't use `Object`
 - Can't downcast to access the properties.

Auto-Typed Variables

- `var x = 7;` *// x will be of type int*
- `var q = from ... select new {...};`
// q will be an array of the anonymous type
`Console.WriteLine(q[0].Name);`
- Local variables only.

Relational Data in the OO World?

- “Regular” LINQ queries yields “rectangular” data.

- e.g.,

```
var od = from c in customers
         where c.City == "London"
         from o in c.Orders
         where o.OrderDate.Year == 2005
         select new { c.Name, o.OrderId, o.Total };
```

= multiple lines per customer.

Relational Data in the OO World?

Joe Average	122324	100.23
Joe Average	315523	90.00
Joe Average	989722	49.95
John Smith	874366	49.90
John Smith	324777	95.00
Miss Piggy	435882	234.65
Kermit	345529	345.21
Kermit	423340	95.95

OO Data in the OO World!

- Nested queries:

```
var od =  
  from c in customers  
  where c.City == "London"  
  select new {  
    c.Name,  
    Orders = from o in c.Orders  
              where o.OrderDate.Year == 2005  
              select { o.OrderId, o.Total }  
  };
```

OO Data in the OO World!

Joe Average

122324

100.23

315523

90.00

989722

49.95

John Smith

874366

49.90

324777

95.00

Miss Piggy

435882

234.65

Kermit

345529

345.21

423340

95.95

Ad-Hoc Type Equality

- Two ad-hoc type expressions that have the same ordered set of property names and types share the same ad-hoc type.

```
var p1 = new { Name = "Moo", Age = 12 };  
var p2 = new { Name = "Elk", Age = 17 };  
p1 = p2;
```

Expression Tree Generation

```
Expression<Func<int, bool>> exprLambda = x => (x & 1) == 0;
```

```
ParameterExpression xParam = Expression.Parameter(typeof(int), "x");  
Expression<Func<int, bool>> exprLambda = Expression.Lambda<Func<int,  
    bool>>(    Expression.EQ(  
                Expression.BitAnd(  
                    xParam, Expression.Constant(1)),  
                Expression.Constant(0)),  
    xParam);
```

Some LINQ Examples

```
from m in typeof(string).getMethods  
select m.Name;
```

String[]

Clone
Compare
Format
CopyTo
Copy
IndexOf
IndexOf
IndexOf
Insert
Substring
Substring
...

Some LINQ Examples

```
from m in typeof(string).getMethods
where !m.IsStatic
select m.Name;
```

String[]

Clone
Compare
CopyTo
Copy
IndexOf
IndexOf
IndexOf
Insert
Substring
Substring
...

Some LINQ Examples

```
from m in typeof(string).getMethods
where !m.IsStatic
orderby m.name
select m.Name;
```

String[]

Clone
Compare
Copy
CopyTo
IndexOf
IndexOf
IndexOf
Insert
Substring
Substring
...

Some LINQ Examples

```
from m in typeof(string).getMethods
where !m.IsStatic
orderby m.name
select m.Name
groupby m.name;
```

KeyGroupPairs[]

Key = Clone	Group = ...
Key = Compare	Group = ...
Key = Copy	Group = ...
Key = CopyTo	Group = ...
Key = IndexOf	Group = ...
Key = Insert	Group = ...
Key = Substring	Group = ...
...	...

Some LINQ Examples

```
from m in typeof(string).getMethods
where !m.IsStatic
orderby m.name
select m.Name
groupby m.name
into g
select new {
    Method = g.Key,
    Overloads = g.Group.Count
};
```

var []

Name = Clone	Overloads = 1
Name = Compare	Overloads = 2
Name = Copy	Overloads = 1
Name = CopyTo	Overloads = 1
Name = IndexOf	Overloads = 5
Name = Insert	Overloads = 3
Name = Substring	Overloads = 6
...	...

Updates with LINQ?

- Future feature: updating capabilities.

```
from c in customers
where c.City == "Peking"
set c.City = "Beijing";    ??
```

DLINQ

- ADO.NET's next-generation.
- Among other features: a database-to-classes utility.
 - Class per DB table.
 - Property per DB column.
 - Foreign keys, etc. reflected in the classes.
- Result: valid table/field names in the SQL.

Extension Methods + Boxing...

```
delegate void Proc();  
static class TestExtensions  
{  
    static public void Times(this Int32 n, Proc proc)  
    {  
        for (int i = 1; i <= n; i++)  
            proc();  
    }  
}
```

Which allows you to write:

```
13.Times(() => Console.Write("X"));
```

Ламбда изрази

Ламбда изразите са естественото продължение на анонимните методи, които бяха въведени в C# 2.0. За да проследим развитието ще започнем от самото начало като обясним накратко делегатите. Още с първата си версия C# предостави специален вид променлива, която сочи към методи – делегатите. Това позволява делегати да се предават като параметри, да се променят и да се извиква сочената функция. Използването на делегати позволява елегантно промяна на програмния поток в зависимост от функциите, към които сочат.

Делегати

В следващият пример са извършени следните операции:

- Дефиниране на делегат Operation, който приема два параметъра.
- Реализиран метод Multiply със същите параметри като делегата Operation.
- В метода се създава променлива deleg, сочеща към инстанция на делегата Operation, която сочи към метода Multiply. Променливата deleg се извиква като нормален метод.

```
public delegate long Operation(long first, int second);
static long Multiply(long i, int j){
    return i * j;
}
{
static void Main(string[] args){
    Operation deleg = new Operation(Multiply);
    long res = deleg(12, 13);
}
```

- **От анонимни методи към ламбда изрази**

Ламбда изразите правят синтаксиса още по сбит. Прототипът е следният:

(Списък с параметри) => тяло на метода

Типът на параметрите не е задължителен и може да се пропусне:

(x, string s) => s.Length > x

Ако тялото на метода съдържа повече от един ред тогава цялото тяло се огражда с {}.

Използвайки ламбда изразите горният пример изглежда така:

```
static void Main(string[] args){  
    Operation deleg3 = (long i, int j) => {return i * j; };  
    long res = deleg2(14, 13);  
}
```

Правила за използване на ламбда изрази:

- Скобите на списъка с параметри може да се пропусне, ако параметърът е един - n => n<10, освен ако не се указва типа на параметрите – (string n) => n<10.
 - Скобите на списъка с параметри е задължителен ако параметрите са повече от един или няма входни параметри.
 - Може да се достъпват променливи от външният метод също като при анонимните методи.
- Ламбда изразите са много често използвани в LINQ, както ще видим по късно в това ръководство. За да покажем силата на тази конструкция ще разгледаме следният код

Resources

- LINQ Home
<http://msdn.com/linq>
- 101 LINQ Examples
<http://tinyurl.com/2rsmpd> [VB]
- <http://tinyurl.com/ymmyr7> [C#]
- LINQ Videos
<http://www.asp.net/learn/linq-videos>
- Scott Guthrie's Blog
weblogs.asp.net/scottgu/
- LINQPad
<http://www.linqpad.net>

Примери

<http://msdn.microsoft.com/en-us/vcsharp/aa336746>

<http://www.galcho.com/articles/LINQTutorial/TutorialLINQ.pdf>



Примери :

```
public void Linq1()
{
    int[] numbers = { 5, 4, 1, 3, 9, 8, 6, 7, 2, 0 };

    var lowNums =
        from n in numbers
        where n < 5
        select n;

    Console.WriteLine("Numbers < 5:");
    foreach (var x in lowNums)
    {
        Console.WriteLine(x);
    }
}
```

Numbers + 1:

6
5
2
4
10
9
7
8
3
1

```
public void Linq24()
{
    int[] numbers = { 5, 4, 1, 3, 9, 8, 6, 7, 2, 0 };

    var firstNumbersLessThan6 = numbers.TakeWhile(n => n < 6);

    Console.WriteLine("First numbers less than 6:");
    foreach (var n in firstNumbersLessThan6)
    {
        Console.WriteLine(n);
    }
}
```

Result

First numbers less than 6:

5
4
1
3

OrderBy - Simple

```
public void Linq28()
{
    string[] words = { "cherry", "apple", "blueberry" };

    var sortedWords =
        from w in words
        orderby w
        select w;

    Console.WriteLine("The sorted list of words:");
    foreach (var w in sortedWords)
    {
        Console.WriteLine(w);
    }
}
```

Result

The sorted list of words:

apple
blueberry
cherry

Distinct

```
public void Linq46()
{
    int[] factorsOf300 = { 2, 2, 3, 5, 5 };

    var uniqueFactors = factorsOf300.Distinct();

    Console.WriteLine("Prime factors of 300:");
    foreach (var f in uniqueFactors)
    {
        Console.WriteLine(f);
    }
}
```

Result

Prime factors of 300:

2
3
5

This sample uses Count to get the number of unique factors of 300.

```
public void Linq73()
{
    int[] factorsOf300 = { 2, 2, 3, 5, 5 };

    int uniqueFactors = factorsOf300.Distinct().Count();

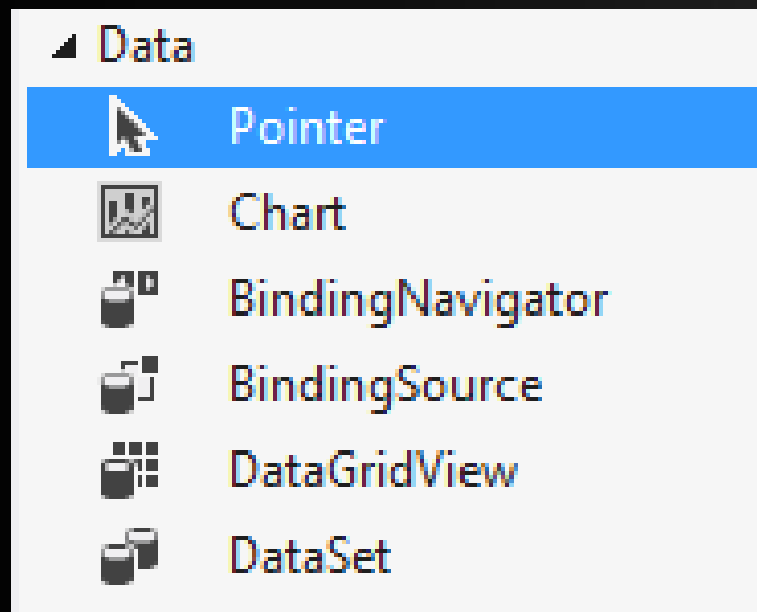
    Console.WriteLine("There are {0} unique factors of 300.", uniqueFactors);
}
```

Result

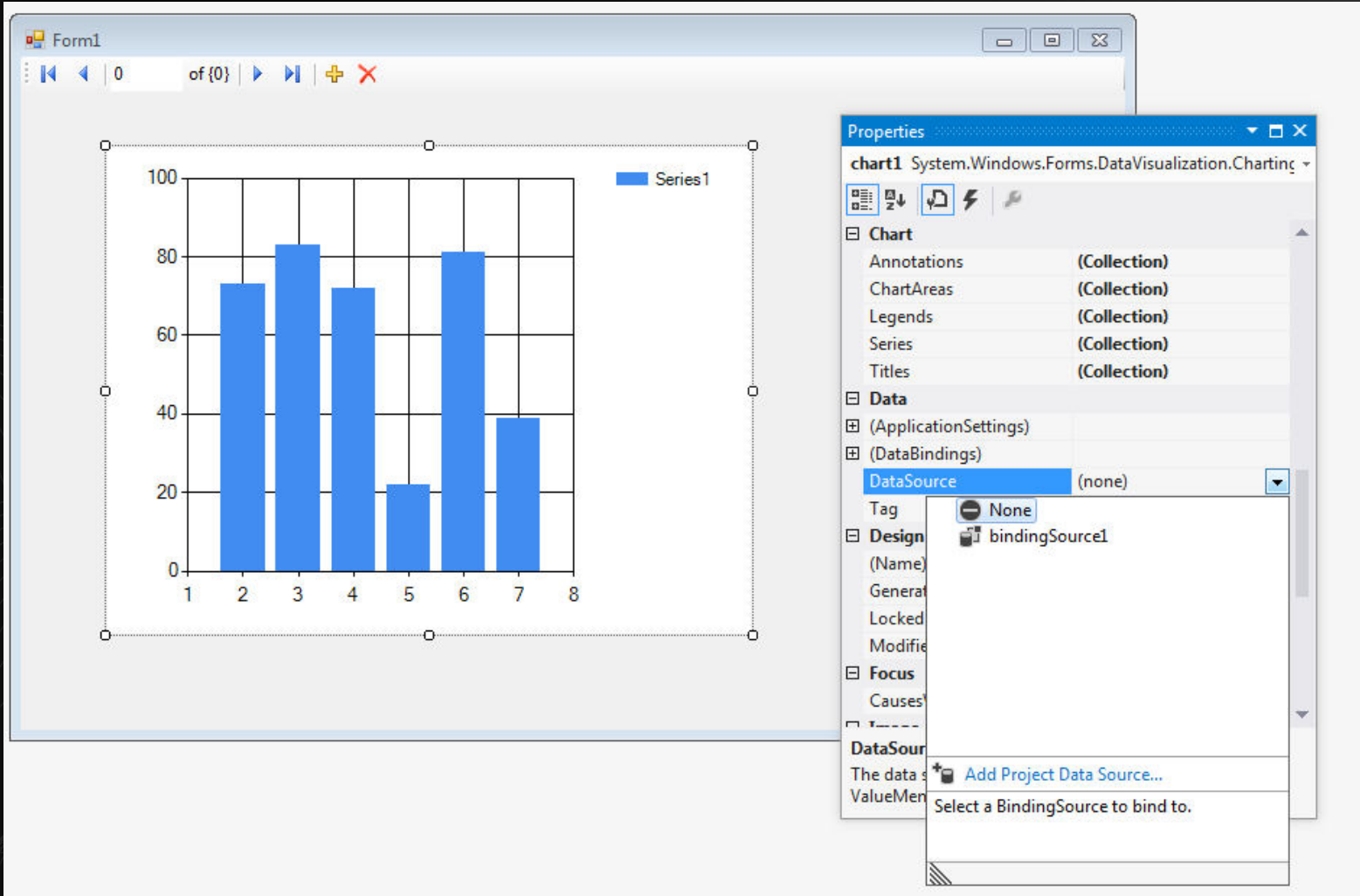
There are 3 unique factors of 300.

Компоненти за работа с DB в .NET

Toolbox



Chart



Chart

//populate dataset with some demo data..

```
DataSet dataSet = new DataSet();
DataTable dt = new DataTable();
dt.Columns.Add("Name", typeof(string));
dt.Columns.Add("Counter", typeof(int));
DataRow r1 = dt.NewRow();
r1[0] = "Demo";
r1[1] = 8;
dt.Rows.Add(r1);
DataRow r2 = dt.NewRow();
r2[0] = "Second";
r2[1] = 15;
dt.Rows.Add(r2);
dataSet.Tables.Add(dt);
```

//prepare chart control...

```
Chart chart = new Chart();
chart.DataSource = dataSet.Tables[0];
chart.Width = 600;
chart.Height = 350;
```

//create serie...

```
Series serie1 = new Series();
serie1.Name = "Serie1";
serie1.Color = Color.FromArgb(112, 255, 200);
serie1.BorderColor = Color.FromArgb(164, 164, 164);
serie1.ChartType = SeriesChartType.Column;
serie1.BorderDashStyle = ChartDashStyle.Solid;
serie1.BorderWidth = 1;
serie1.ShadowColor = Color.FromArgb(128, 128, 128);
serie1.ShadowOffset = 1;
serie1.IsValueShownAsLabel = true;
serie1.XValueMember = "Name";
serie1.YValueMembers = "Counter";
serie1.Font = new Font("Tahoma", 8.0f);
serie1.BackSecondaryColor = Color.FromArgb(0, 102, 153);
serie1.LabelForeColor = Color.FromArgb(100, 100, 100);
chart.Series.Add(serie1);
```

//create chartareas...

```
ChartArea ca = new ChartArea();
ca.Name = "ChartArea1";
ca.BackColor = Color.White;
ca.BorderColor = Color.FromArgb(26, 59, 105);
ca.BorderWidth = 0;
ca.BorderDashStyle = ChartDashStyle.Solid;
ca.AxisX = new Axis();
ca.AxisY = new Axis();
chart.ChartAreas.Add(ca);
```

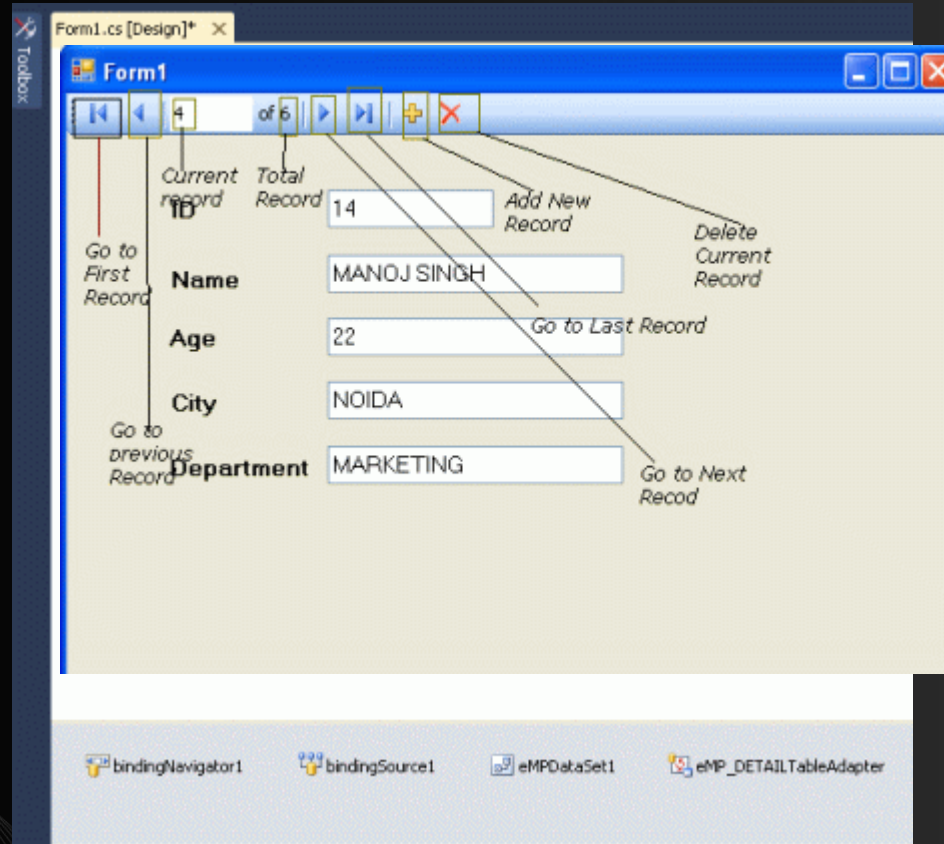
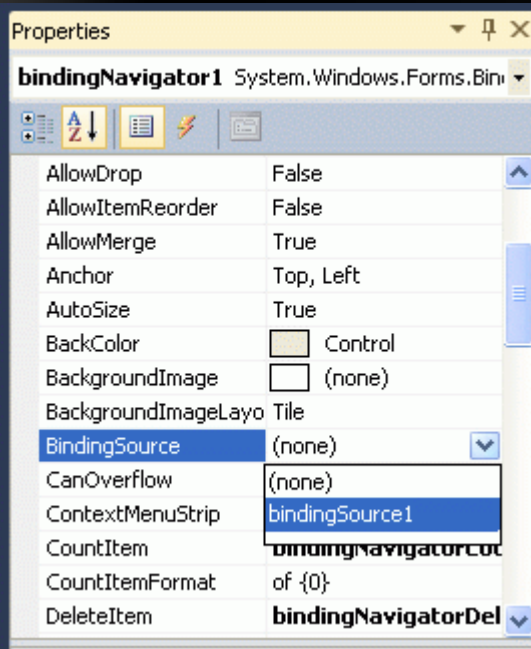
//databind...

```
chart.DataBind();
```

//save result...

```
chart.SaveImage(@"c:\myChart.png", ChartImageFormat.Png);
```


Binding Navigator

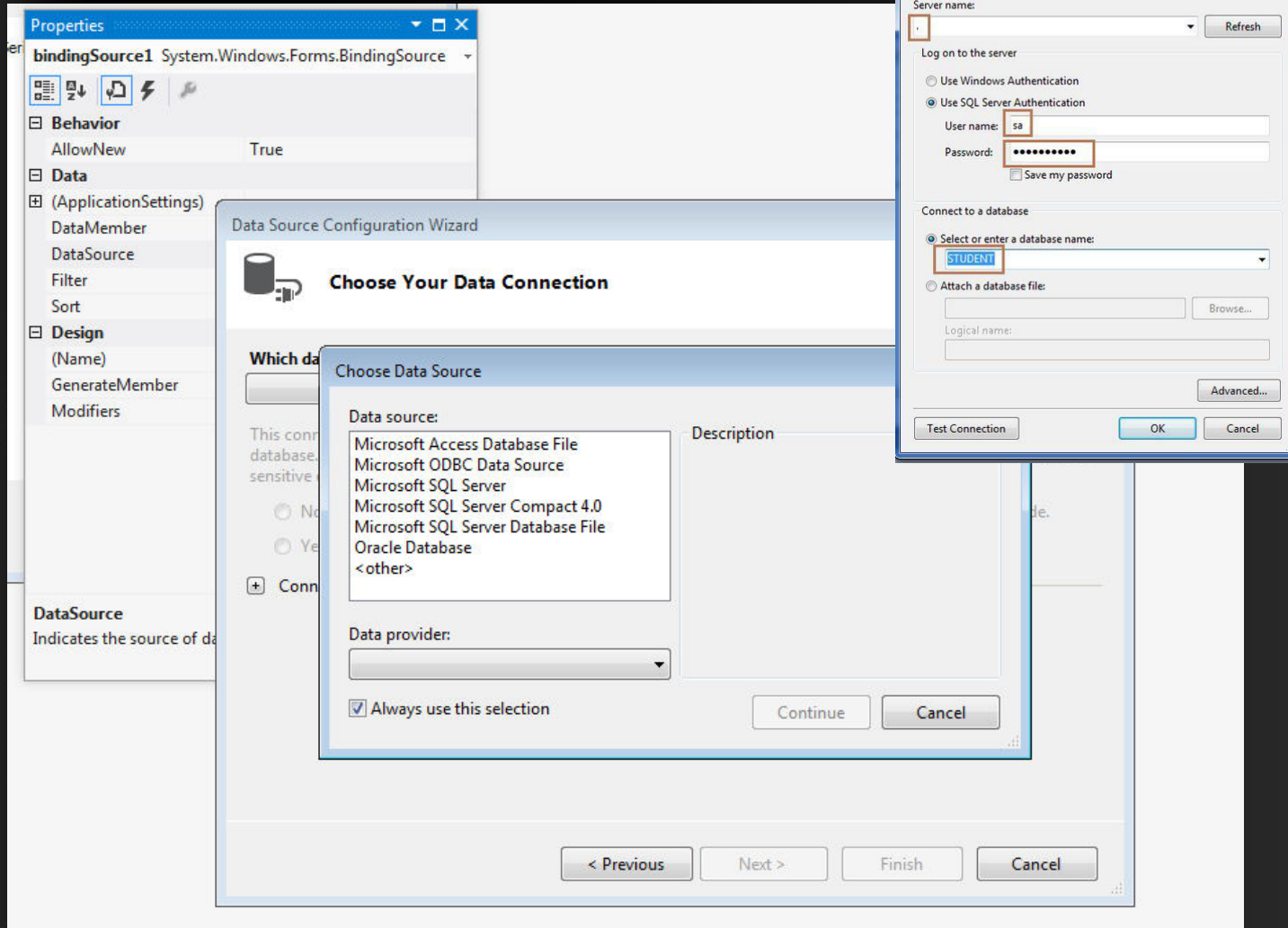


BN - Code

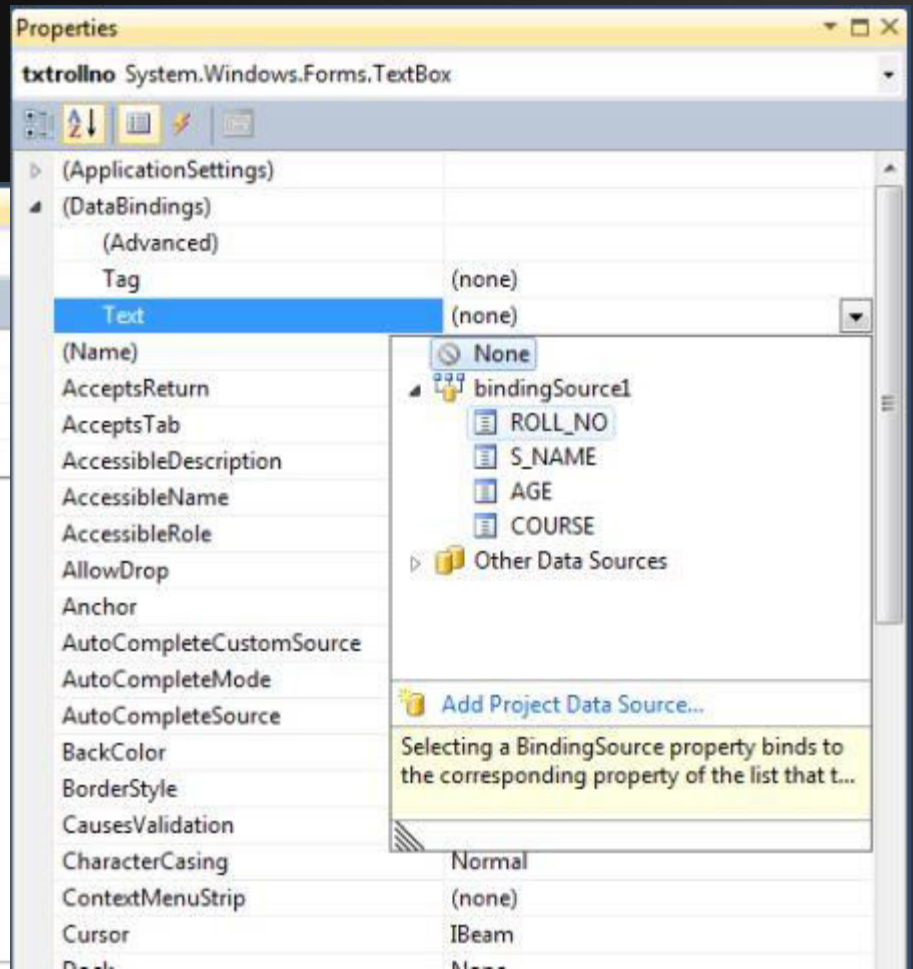
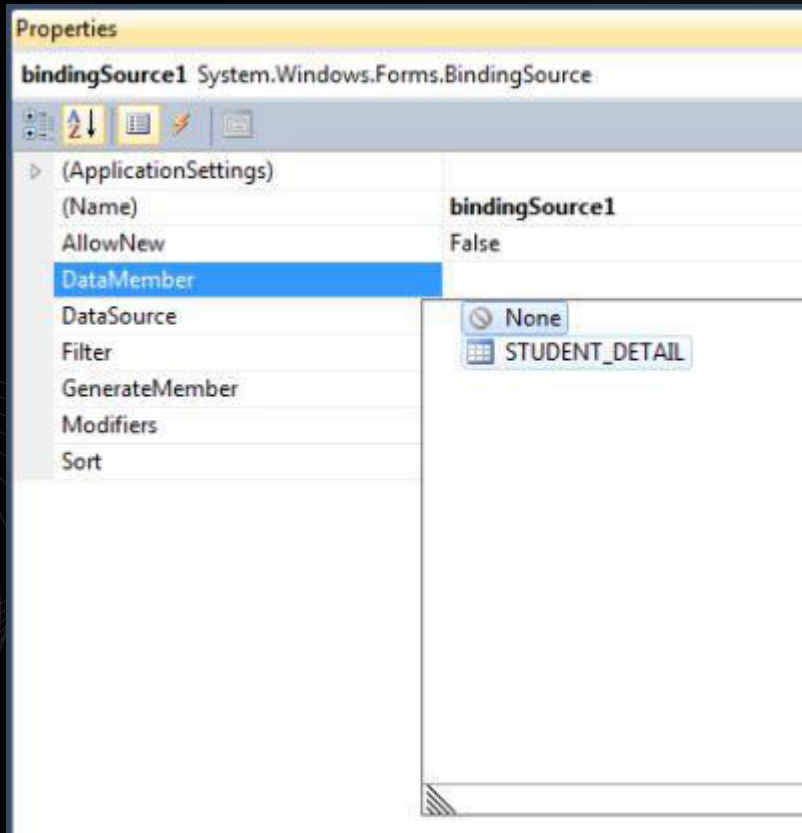
```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
using System.Data.SqlClient;

namespace WindowsFormsApplication1
{
    public partial class Form2 : Form
    {
        public Form2()
        {
            InitializeComponent();
        }
        SqlDataAdapter da;
        DataSet ds;
        private void Form2_Load(object sender, EventArgs e)
        {
            da = new SqlDataAdapter("select * from emp_detail","database=emp;server=.;user=sa;password=wintellect");
            ds = new DataSet();
            da.Fill(ds);
            bindingSource1.DataSource = ds.Tables[0];
            bindingNavigator1.BindingSource = this.bindingSource1;
            txtid.DataBindings.Add(new Binding("Text", bindingSource1, "E_ID", true));
            txtname.DataBindings.Add(new Binding("Text", bindingSource1, "E_NAME", true));
            txtage.DataBindings.Add(new Binding("Text", bindingSource1, "E_AGE", true));
            txtcity.DataBindings.Add(new Binding("Text", bindingSource1, "E_CITY", true));
            txtdepartment.DataBindings.Add(new Binding("Text", bindingSource1, "E_DEPARTMENT", true));
        }
    }
}
```

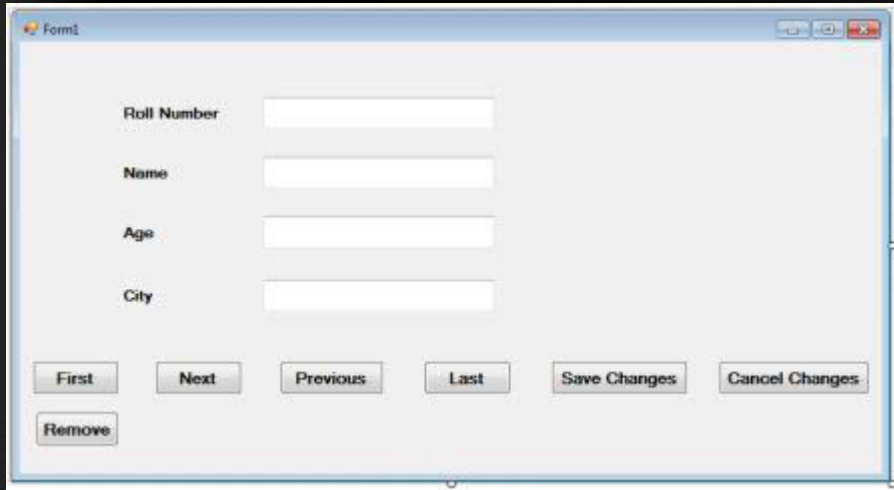
BindingSource 1



BindingSource 2



BindingSource 3



The screenshot shows a Windows Form titled 'Form1'. It contains four text boxes for data entry, labeled 'Roll Number', 'Name', 'Age', and 'City'. Below these text boxes, there are seven buttons: 'First', 'Next', 'Previous', 'Last', 'Save Changes', 'Cancel Changes', and 'Remove'. The form is designed for navigating through and editing data from a data source.

```
namespace bindingsourceapps
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }
        private void Form1_Load(object sender, EventArgs e)
        {
            // TODO: This line of code loads data into the
            'sSTUDENTDataSet.STUDENT_DETAIL' table. You can move, or remove it, as
            needed.
            this.sSTUDENT_DETAILTableAdapter.Fill(this.sSTUDENTDataSet.STUDE
            NT_DETAIL);
        }

        private void btnfirst_Click(object sender, EventArgs e)
        {
            bindingSource1.MoveFirst();
        }

        private void btnnext_Click(object sender, EventArgs e)
        {
            bindingSource1.MoveNext();
        }

        private void btnprevious_Click(object sender, EventArgs e)
        {
            bindingSource1.MovePrevious();
        }

        private void btnlast_Click(object sender, EventArgs e)
        {
            bindingSource1.MoveLast();
        }

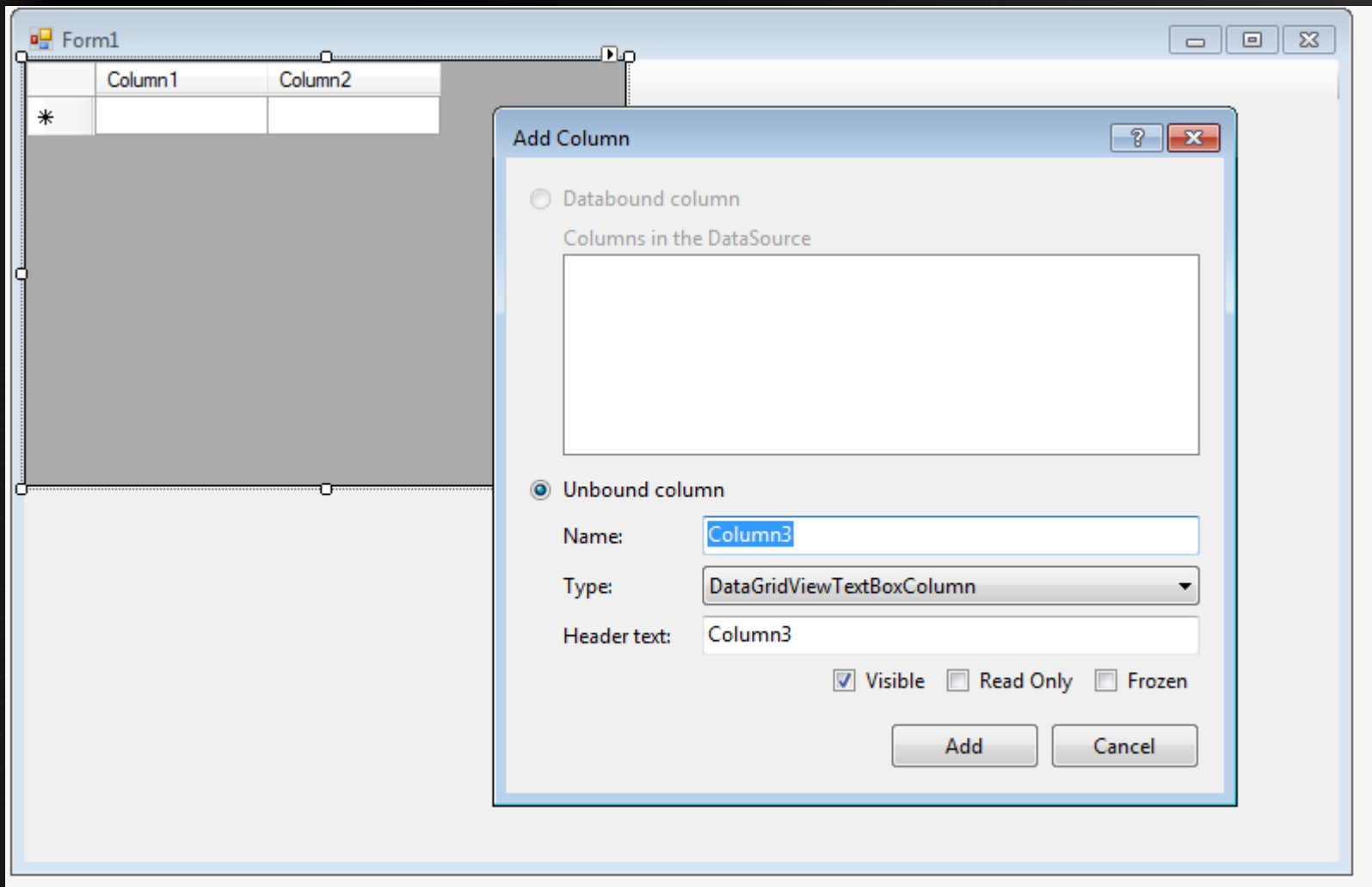
        private void btnsavechanges_Click(object sender, EventArgs e)
        {
            bindingSource1.EndEdit();
        }

        private void btncancelchanges_Click(object sender, EventArgs e)
        {
            bindingSource1.CancelEdit();
        }

        private void btnremove_Click(object sender, EventArgs e)
        {

```

DataGridView



DataGridView

```
using System;
using System.Data;
using System.Windows.Forms;
using System.Data.SqlClient;

namespace WindowsApplication1
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }

        private void button1_Click(object sender, EventArgs e)
        {
            string connectionString = "Data Source=.;Initial Catalog=pubs;Integrated Security=True";
            string sql = "SELECT * FROM Authors";
            SqlConnection connection = new SqlConnection(connectionString);
            SqlDataAdapter dataadapter = new SqlDataAdapter(sql, connection);
            DataSet ds = new DataSet();
            connection.Open();
            dataadapter.Fill(ds, "Authors_table");
            connection.Close();
            dataGridView1.DataSource = ds;
            dataGridView1.DataMember = "Authors_table";
        }
    }
}
```

DataGridView

```
private void GetData(string selectCommand)
{
    try
    {
        String connectionString =
            "Integrated Security=SSPI;Persist Security Info=False;" +
            "Initial Catalog=Northwind;Data Source=localhost";

        // Create a new data adapter based on the specified query.
        dataAdapter = new SqlDataAdapter(selectCommand, connectionString);

        // Create a command builder to generate SQL update, insert, and
        // delete commands based on selectCommand. These are used to
        // update the database.
        SqlCommandBuilder commandBuilder = new SqlCommandBuilder(dataAdapter);

        // Populate a new data table and bind it to the BindingSource.
        DataTable table = new DataTable();
        table.Locale = System.Globalization.CultureInfo.InvariantCulture;
        dataAdapter.Fill(table);
        bindingSource1.DataSource = table;

        // Resize the DataGridView columns to fit the newly loaded content.
        dataGridView1.AutoSizeColumns(
            DataGridViewAutoSizeColumnsMode.AllCellsExceptHeader);
    }
    catch (SqlException)
    {
        MessageBox.Show("To run this example, replace the value of the " +
            "connectionString variable with a connection string that is " +
            "valid for your system.");
    }
}
```


DataGridView

[C# DataGridView Binding - SQL Server dataset](#)

[C# DataGridView Binding - OLEDB dataset](#)

[C# DataGridView Sorting and Filtering](#)

[C# DataGridView Add Columns and Rows](#)

[C# DataGridView Hide Columns and Rows](#)

[C# DataGridView Read Only Columns and Rows](#)

[Add Button to C# DataGridView](#)

[Add CheckBox to C# DataGridView](#)

[Add ComboBox to C# DataGridView](#)

[Add Image to C# DataGridView](#)

[Add ViewLink to C# DataGridView](#)

[C# DataGridView Paging](#)

[C# DataGridView Formatting](#)

[C# DataGridView Template](#)

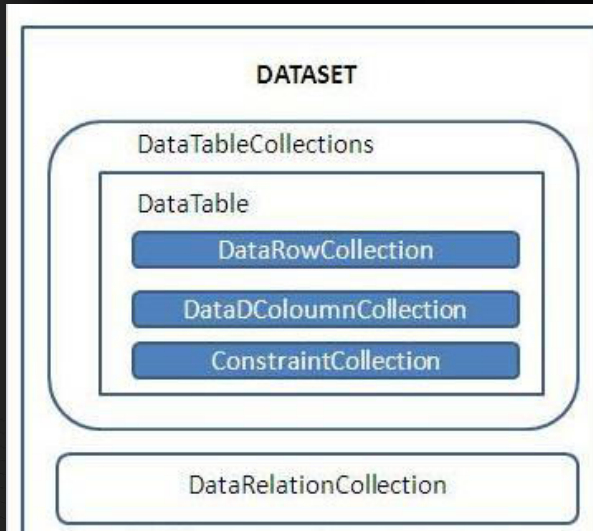
[C# DataGridView Printing](#)

[C# DataGridView Export to Excel](#)

[C# DataGridView Loading data from Excel](#)

[C# DataGridView Database Operations](#)

DataSet



```
private void button1_Click(object sender, EventArgs e)
{
    string connetionString = null;
    SqlConnection connection ;
    SqlCommand command ;
    SqlDataAdapter adapter = new SqlDataAdapter();
    DataSet ds = new DataSet();
    int i = 0;
    string sql = null;

    connetionString = "Data Source=ServerName;Initial
Catalog=DatabaseName;User ID=UserName;Password=Password";
    sql = "Your SQL Statement Here";

    connection = new SqlConnection(connetionString);

    try
    {
        connection.Open();
        command = new SqlCommand(sql, connection);
        adapter.SelectCommand = command;
        adapter.Fill(ds);
    }
}
```

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

```
{
```

```
    string connetionString = null;
```

```
    SqlConnection connection ;
```

```
    SqlCommand command ;
```

```
    SqlDataAdapter adapter = new SqlDataAdapter();
```

```
    DataSet ds = new DataSet();
```

```
    string firstSql = null;
```

```
    string secondSql = null;
```

```
    connetionString = "Data Source=ServerName;Initial Catalog=DatabaseName;User ID=UserName;Password=Password";
```

```
    firstSql = "Your First SQL Statement Here";
```

```
    secondSql = "Your Second SQL Statement Here";
```

```
    connection = new SqlConnection(connetionString);
```

```
    try
```

```
    {
```

```
        connection.Open();
```

```
        command = new SqlCommand(firstSql, connection);
```

```
        adapter.SelectCommand = command;
```

```
        adapter.Fill(ds, "Table1");
```

```
        adapter.SelectCommand.CommandText = secondSql;
```

```
        adapter.Fill(ds, "Table2");
```

```
        //creating data relations
```

```
        DataRelation relation ;
```

```
        DataColumn table1Column ;
```

```
        DataColumn table2Column ;
```

```
        //retrieve column
```

```
        table1Column = ds.Tables["Table1"].Columns[0];
```

```
        table2Column = ds.Tables["table2"].Columns[0];
```

```
        //relating tables
```

```
        relation = new DataRelation("relation", table1Column, table2Column);
```

```
        //assign relation to dataset
```

```
        ds.Relations.Add(relation);
```

```
}
```

DataSet - Relation

DataSet

```
private void button1_Click(object sender, EventArgs e)
{
    string connetionString = null;
    OleDbConnection connection ;
    OleDbDataAdapter oledbAdapter ;
    DataSet ds = new DataSet();
    string sql = null;
    int i = 0;

    connetionString = "Provider=Microsoft.Jet.OLEDB.4.0;Data Source=Your mdb filename;";
    sql = "Your SQL Statement Here";

    connection = new OleDbConnection(connetionString);
    try
    {
        connection.Open();
        oledbAdapter = new OleDbDataAdapter(sql, connection);
        oledbAdapter.Fill(ds);

        for (i = 0; i <= ds.Tables[0].Rows.Count - 1; i++)
        {
            MessageBox.Show (ds.Tables[0].Rows[i].ItemArray[0] + " -- " + ds.Tables[0].Rows[i].ItemArray[1]);
        }
    }
    catch (Exception ex)
    {
        MessageBox.Show("Can not open connection ! ");
    }
}
```

DataSet — Binding ComboBox

```
private void button1_Click(object sender, EventArgs e)
{
    string connetionString = null;
    SqlConnection connection;
    SqlCommand command;
    SqlDataAdapter adapter = new SqlDataAdapter();
    DataSet ds = new DataSet();
    int i = 0;
    string sql = null;

    connetionString = "Data Source=ServerName;Initial Catalog=databasename;User ID=userid;Password=yourpassword";
    sql = "select au_id,au_lname from authors";

    connection = new SqlConnection(connetionString);

    try
    {
        connection.Open();
        command = new SqlCommand(sql, connection);
        adapter.SelectCommand = command;
        adapter.Fill(ds);
        .

        comboBox1.DataSource = ds.Tables[0];
        comboBox1.ValueMember = "au_id";
        comboBox1.DisplayMember = "au_lname";

    }
    catch (Exception ex)
```

LINQ

LINQ – Language INtegrated Query
(заявки вградени в езика)

Пример LINQ

```
using System;
using System.Query;
using System.Collections.Generic;

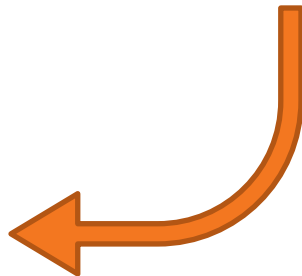
class app {
    static void Main() {

        string[] names = { "Burke", "Connor",
                           "Frank", "Everett",
                           "Albert", "George",
                           "Harris", "David" };

        var expr = from s in names
                    where s.Length == 5
                    orderby s
                    select s.ToUpper();

        foreach (string item in expr)
            Console.WriteLine(item);
    }
}
```

BURKE
DAVID
FRANK



- private void FilterWithCSharp2()
- }
- string[] contacts = { "Gogo", "George", "Trifon", "Kiril", "Shumi" };
- //display result
- foreach (string name in FilterResults2(contacts))
- {
- Console.WriteLine(name);
- }
- {
- {
- private IEnumerable FilterResults2(string[] Data)
- }
- //filter array
- foreach (string name in Data)
- {
- if (name.StartsWith("G"))
- return name;
- {
- {


```
private static void FilterWithCSharp3() {  
    string[] contacts = { "Galcho", "George", "Trifon", "Kiril",  
        "Shumi" };  
    var result = from s in contacts  
        where s.StartsWith("G")  
select s;  
    //display result  
    foreach (string name in result) {  
        Console.WriteLine(name);  
    }  
}
```

```
private static void FilterWithCSharp3_2() {  
    string[] contacts = { " Gogo ", "George", "Trifon", "Kiril", "Shumi" };  
    var result = contacts.Where<string>(x => x.StartsWith("G"));  
    //display result  
    foreach (string name in result) {  
        Console.WriteLine(name);  
    }  
}
```

Инициализация на обекти

```
Invoice i = new Invoice { CustomerId = 123, Name = "Test" };
```

Is equivalent to:

```
Invoice I = new Invoice();  
i.CustomerId = 123;  
i.Name = "Test";
```

Lambda Expressions

Predicates

- Predicate
 - $(p) \Rightarrow p.\text{Gender} == \text{"F"}$
- Projection
 - $(p) \Rightarrow p.\text{Gender} ? \text{"F"} : \text{"Female"}$
 - “Each person p becomes string “Female” if Gender is “F””

Query Expressions

- Introduce SQL-Like Syntax to Language
- Compiled to Traditional C# (via Extension Methods)

```
from itemName in srcExpr
join itemName in srcExpr on keyExpr equals keyExpr
    (into itemName)?
let itemName = selExpr
where predExpr
orderby (keyExpr (ascending | descending)?) *
select selExpr
group selExpr by keyExpr
into itemName query-body
```

The LINQ Project

C# 3.0

Visual Basic 9.0

Others

.NET Language Integrated Query

LINQ to
Objects

LINQ to
DataSets

LINQ to
SQL

LINQ to
Entities

LINQ to
XML



Objects



Relational

```
<book>  
  <title/>  
  
  <author/>  
    <year/>  
    <price/>  
</book>
```

XML

LINQ to Objects

C# 3.0

Visual Basic 9.0

Others

.NET Language Integrated Query

LINQ to
Objects

LINQ to
DataSets

LINQ to
SQL

LINQ to
Entities

LINQ to
XML



Objects



Relational

```
<book>
  <title/>

  <author/>
  <year/>
  <price/>
</book>
```

XML

LINQ to Objects

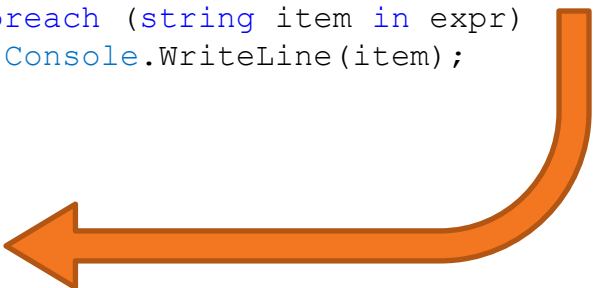
- Native query syntax in C# and VB
- IntelliSense
- Autocompletion
- Query Operators can be used against any .NET collection (IEnumerable<T>)
- Select, Where, GroupBy, Join, etc.
- Deferred Query Evaluation
- Lambda Expressions

```
using System;
using System.Query;
using System.Collections.Generic;

class app {
    static void Main() {
        string[] names = { "Burke", "Connor",
                           "Frank", "Everett",
                           "Albert", "George",
                           "Harris", "David" };
        Func<string, bool> filter = s => s.Length == 5;
        Func<string, string> extract = s => s;
        Func<string, string> project = s => s.ToUpper();

        IEnumerable<string> expr = names
            .Where(filter)
            .OrderBy(extract)
            .Select(project);

        foreach (string item in expr)
            Console.WriteLine(item);
    }
}
```



BURKE
DAVID
FRANK

LINQ to SQL

C# 3.0

Visual Basic 9.0

Others

.NET Language Integrated Query

LINQ to
Objects

LINQ to
DataSets

LINQ to
SQL

LINQ to
Entities

LINQ to
XML



Objects



Relational

```
<book>
  <title/>

  <author/>
  <year/>
  <price/>
</book>
```

XML

LINQ to XML

C# 3.0

Visual Basic 9.0

Others

.NET Language Integrated Query

LINQ to
Objects

LINQ to
DataSets

LINQ to
SQL

LINQ to
Entities

LINQ to
XML



Objects



Relational

```
<book>  
  <title/>  
  
  <author/>  
  <year/>  
  <price/>  
</book>
```

XML

LINQ to XML

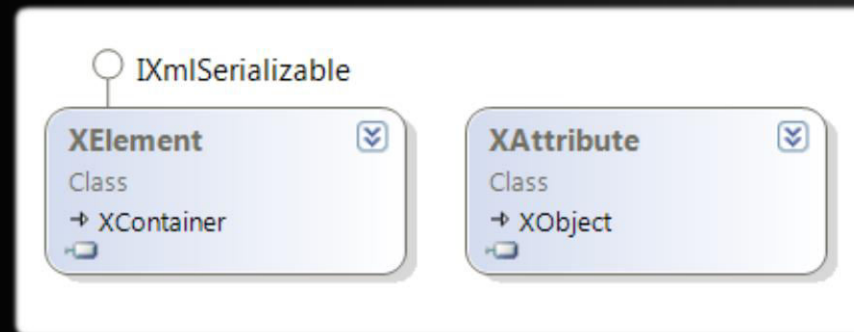
- Large Improvement Over Existing Model
- Supports:
 - Creating XML
 - Loading & querying XML
 - Modifying & saving XML
 - Streaming, Schema, Annotations, Events

LINQ to XML

- New XML API implemented in v3.5 assembly
 - `System.Xml.Linq.dll`
- Namespaces
 - `System.Xml.Linq`
 - `System.Xml.Schema`
 - `System.Xml.XPath`
- Can be used independently of LINQ

Key Classes in System.Xml.Linq

- System.Xml.Linq is a “DOM like” API
 - Manipulates an XML tree in memory
- Naturally work with both XML **documents** and **fragments**
- The two key classes in System.Xml.Linq



Loading Xml Content

- Loading Xml is performed with;
 - **XElement.Load**
 - **XDocument.Load**
- Both support loading from
 - **URI, XmlReader, TextReader**

```
XmlReader reader = XmlReader.Create("myData.xml");  
XElement element = XElement.Load(reader);
```

Modifying XML

- XML tree exposed by **XElement** et al. is modifiable
- Modifications through methods such as:
 - **XElement.Add()**
 - **XElement.Remove()**
 - **XElement.ReplaceWith()**
- Modified tree can be persisted via
 - **XElement.Save(), XDocument.Save()**
 - Both supporting filename, TextWriter, XmlWriter.

```
XElement element = new XElement("foo");  
element.Save(@"c:\temp\foo.xml");|
```

Creating an XML Document

```
XNamespace ns = "http://example.books.com";
XDocument books = new XDocument(
    new XElement(ns + "bookstore",
        new XElement(ns + "book",
            new XAttribute("ISBN", isbn),
            new XElement(ns + "title", "ASP.NET Book"),
            new XElement(ns + "author",
                new XElement(ns + "first-name", a.FirstName),
                new XElement(ns + "last-name", a.LastName)
            )
        )
    );

books.Save(@"C:\Books.xml");
```

VB 9: XML Literals

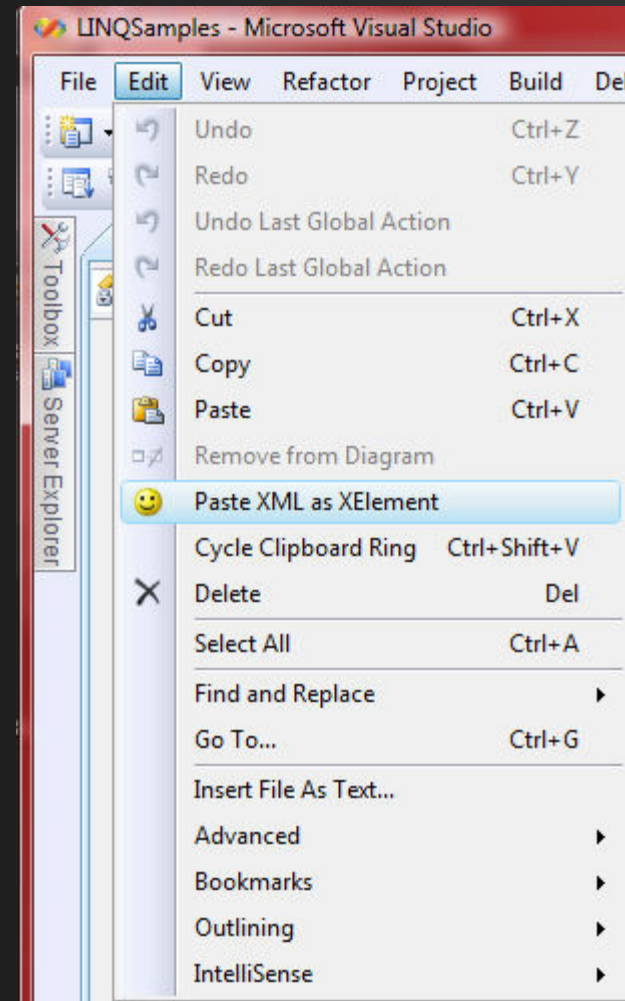
- XML Can Be Used Inline
- Compiled to *XElement* Expression
- Can Be Code-Driven and Dynamic
 - Including LINQ Expressions

```
Dim books = <bookstore xmlns="http://examples.books.com">
    <book ISBN=<%= isbn %>>
        <title>ASP.NET Book</title>
        <author>
            <first-name><%= a.FirstName %></first-name>
            <last-name><%= a.LastName %></last-name>
        </author>
    </book>
</bookstore>

books.Save("C:\Books.xml");
```


...Meanwhile in C#

- No XML Literals, But There's Something to Close the Gap
- “Paste XML as XElement” Add-in
 - Add XML to Clipboard
 - Edit -> Past XML as XElement
- Included in VS2008 Samples
 - Help -> Samples



LINQ to Entities

C# 3.0

Visual Basic 9.0

Others

.NET Language Integrated Query

LINQ to
Objects

LINQ to
DataSets

LINQ to
SQL

LINQ to
Entities

LINQ to
XML



Objects



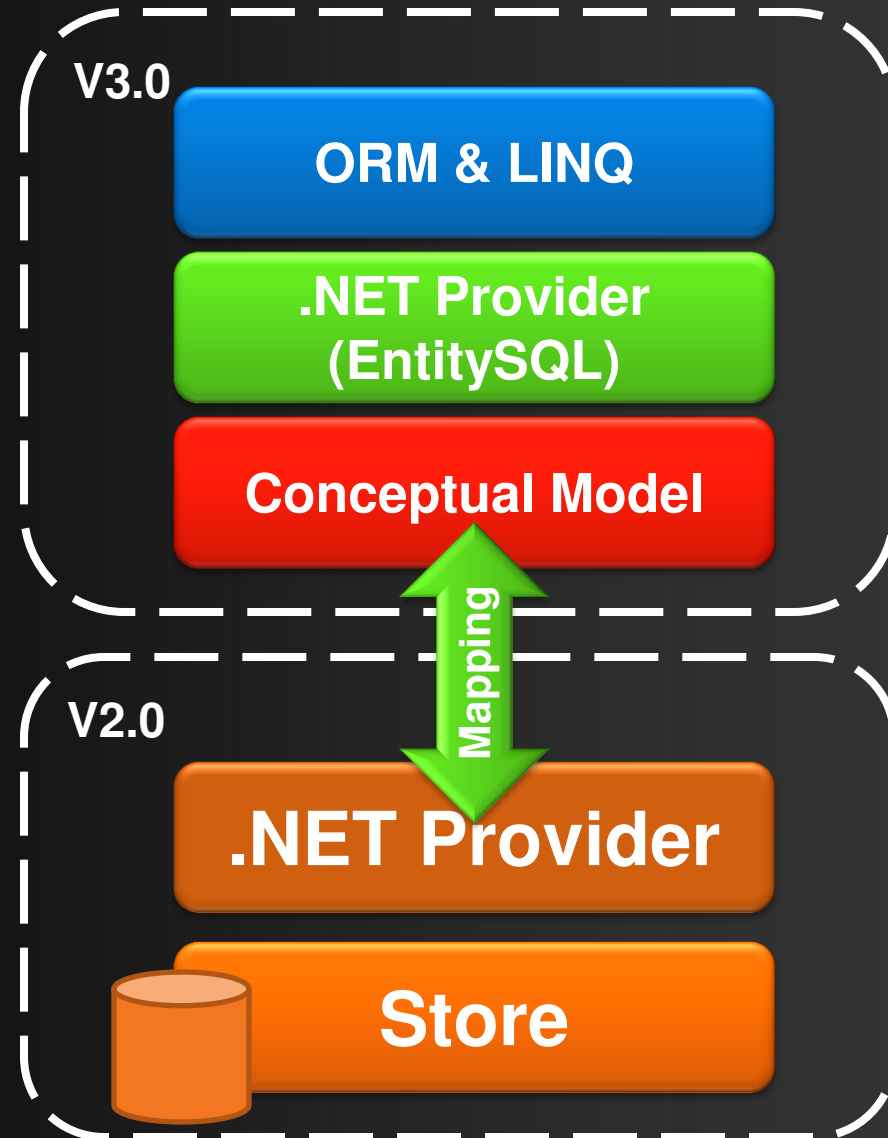
Relational

```
<book>  
  <title/>  
  
  <author/>  
    <year/>  
    <price/>  
</book>
```

XML

ADO.NET Entity Framework

- Schema and Store independence
- Higher-Level Constructs
 - Relationships
 - Inheritance
- Functionality
 - Flexibility of Entity Framework model & mapping
 - Productivity of LINQ
- Ships post-VS 2008
 - At Beta 2 Today
 - Tooling at CTP 1



LINQ to Entities

- New Data Access API implemented in assembly
 - `System.Data.Entity.dll`
 - `System.Data.Entity.Design.dll`
- Namespaces
 - `System.Data.Entity`
 - `System.Data.Objects`
 - and more...

Searching in Collections

- Begin with a simple array of, say, Customers.

```
Customer[] customers = new Customer[30];  
customers[0] = new Customer(...);  
...  
customers[29] = new Customer(...);
```

Searching in Collections: The Old Way

- Find the names of all London customers:

```
List<string> londoners = new List<string>();  
  
foreach (Customer c in customers) {  
    if (c.City == "London") {  
        londoners.add(c.Name);  
    }  
}
```

Searching in Collections: The LINQ Way

```
string[] londoners =  
    from c in customers  
    where c.City == "London"  
    select c.Name;
```

No loops!

SQL-like!

Declarative!

Syntax Translation Example

```
string[] londoners =  
    from c in customers  
    where c.City == "London"  
    select c.Name;
```

```
string[] londoners =  
    customers.  
    Where (expression) .  
    Select (expression) ;
```



Expressions == Methods?

- `Where()` wants a Boolean *method*.
- The method acts as a *filter*.
- Likewise for `Select()`: a *translation* method.

Some LINQ Examples

```
from m in typeof(string).getMethods
where !m.IsStatic
orderby m.name
select m.Name
groupby m.name
into g
select new {
    Method = g.Key,
    Overloads = g.Group.Count
};
```

var []

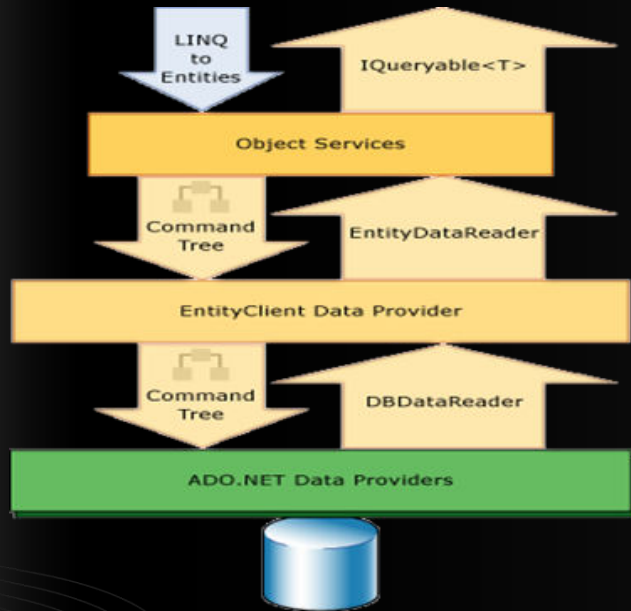
Name = Clone	Overloads = 1
Name = Compare	Overloads = 2
Name = Copy	Overloads = 1
Name = CopyTo	Overloads = 1
Name = IndexOf	Overloads = 5
Name = Insert	Overloads = 3
Name = Substring	Overloads = 6
...	...

Updates with LINQ?

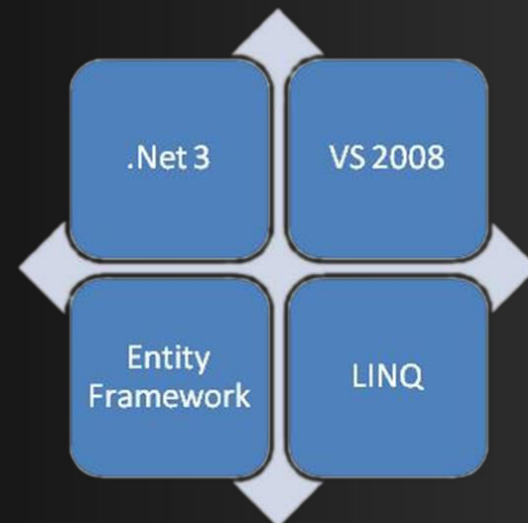
- Future feature: updating capabilities.

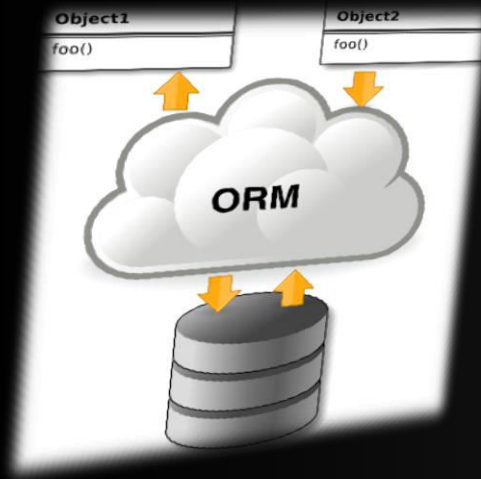
```
from c in customers
where c.City == "Peking"
set c.City = "Beijing";    ??
```

ADO.NET Entity Framework



ORM Concepts ADO.NET Entity Framework,ObjectContext





Introduction to ORM

Object-Relational Mapping (ORM) Technologies

ORM Technologies

- Object-Relational Mapping (ORM) is a programming technique for automatic mapping and converting data
 - Between relational database tables and object-oriented classes and objects
- ORM creates a “virtual object database”
 - Which can be used from within the programming language, e.g. C# or Java
- ORM frameworks automate ORM process
 - A.k.a. object-relational persistence frameworks

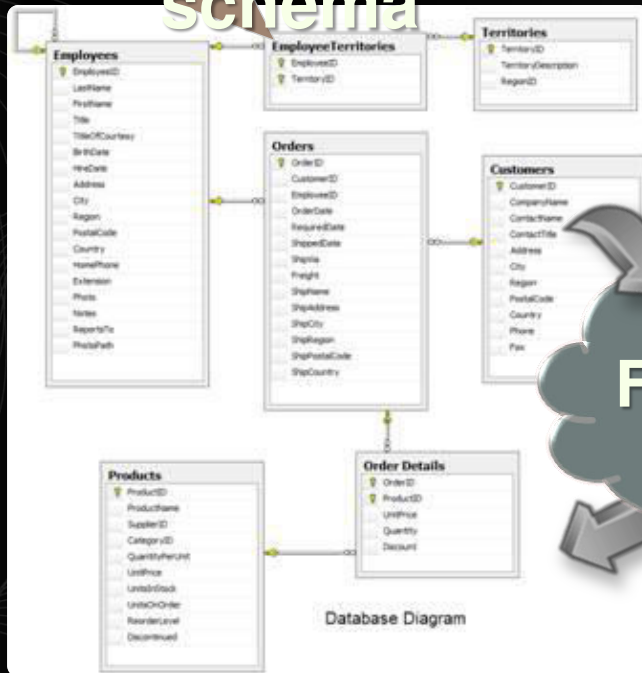
ORM Frameworks

- ORM frameworks typically provide the following functionality:
 - Creating object model by database schema
 - Creating database schema by object model
 - Querying data by object-oriented API
 - Data manipulation operations
 - **CRUD** – create, retrieve, update, delete
- ORM frameworks automatically generate SQL to perform the requested data operations

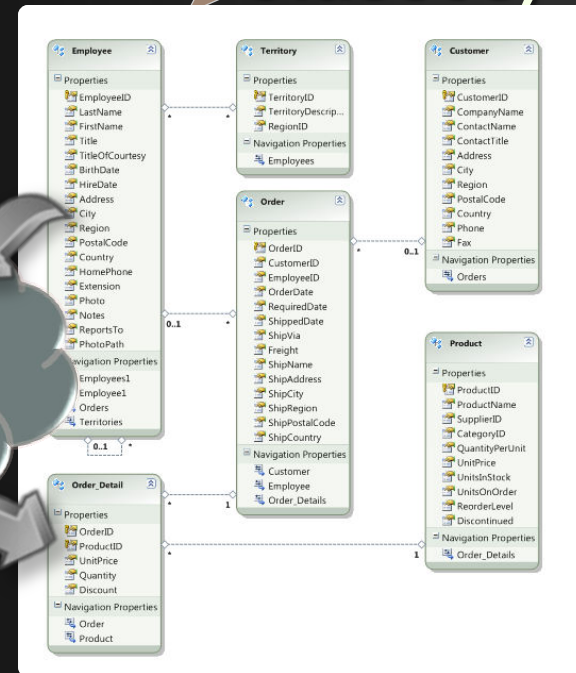
ORM Mapping – Example

- Database and Entities mapping diagrams for a subset of the Northwind database

Relational database schema



ORM Entities (C# Classes)



ORM
Framework

ORM Advantages

- Object-relational mapping advantages
 - Developer productivity
 - Writing less code
 - Abstract from differences between object and relational world
 - Complexity hidden within ORM
 - Manageability of the CRUD operations for complex relationships
 - Easier maintainability

ORM Frameworks in .NET

- Built-in ORM tools in .NET Framework and VS
 - ADO.NET Entity Framework
 - LINQ to SQL
 - Both combine entity class mappings and code generation, SQL is generated at runtime
- Third party ORM tools
 - NHibernate – the old daddy of ORM

ADO.NET Entity Framework

Object Relation Mapping Framework

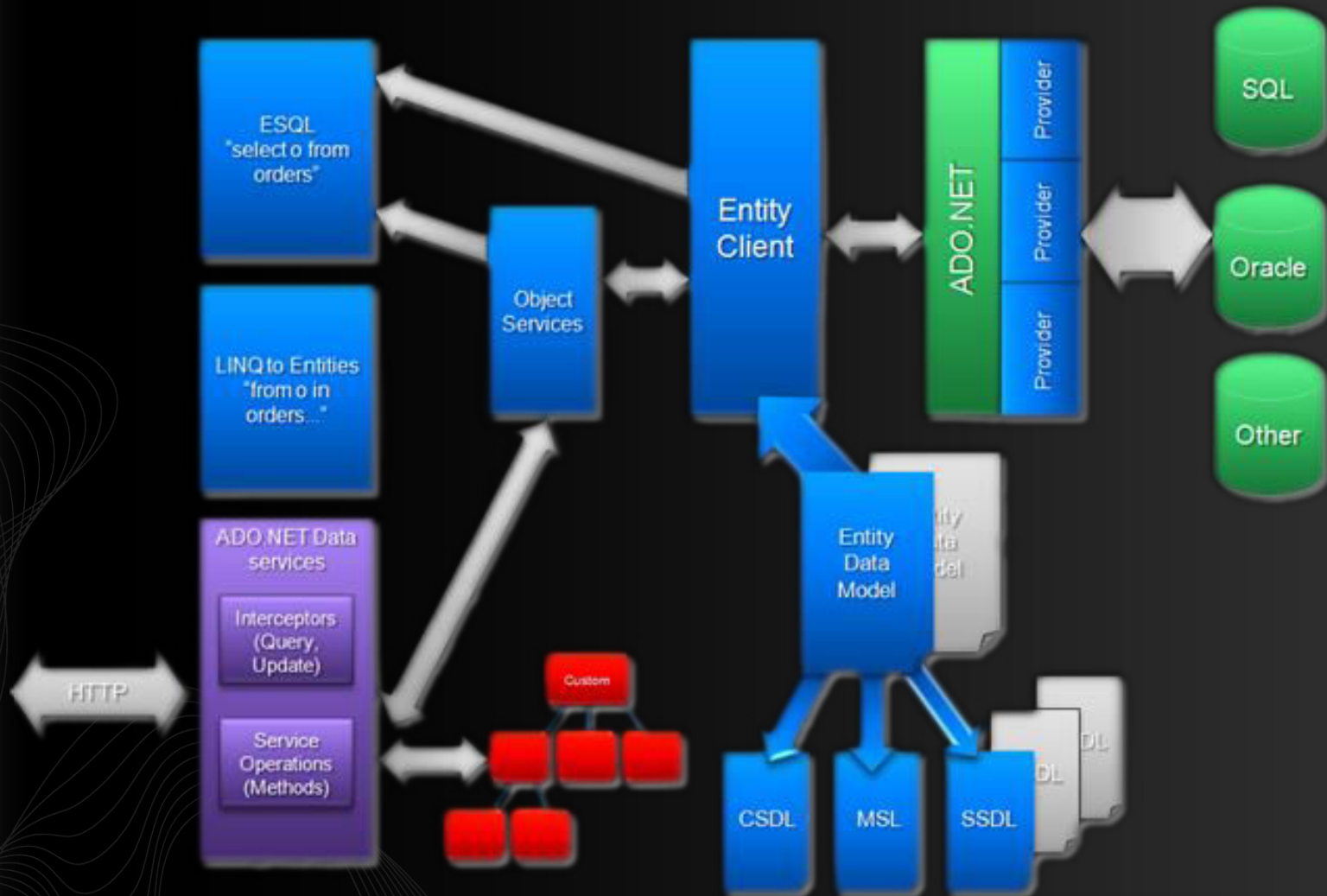
Overview of ADO.NET

- Entity Framework is a standard ORM framework, part of .NET
 - Provides a run-time infrastructure for managing SQL Database data as .NET objects
- The relational database schema is mapped to an object model (classes and associations)
 - Visual Studio has built-in tools for generating Entity Framework SQL data mappings
 - Data mappings consist of C# classes and XML
 - A standard data manipulation API is provided

Overview of ADO.NET (2)

- Entity Framework is an application programming interface (API)
 - For working with Database Servers
 - Built on the top of ADO.NET and LINQ
- LINQ to Entity is Microsoft's entry-level LINQ-enabled ORM implementation for Database Servers
 - Works with SQL Server and SQL Server Express
 - Works with MySQL, Oracle etc.
 - Maps tables and one-to-many relationships
 - Maps many-to-many relationships

ADO.NET Entity Framework Architecture



Entity Framework Features

- Entity Framework standard features:
 - Map tables, views, stored procedures and functions as objects
 - Provides LINQ-bases data queries
 - Executed as SQL SELECTs on the Database server
 - CRUD operations – Create/Read/Update/Delete
 - Create compiled queries – for executing the same parameterized query multiple times
 - Creating or deleting the database schema

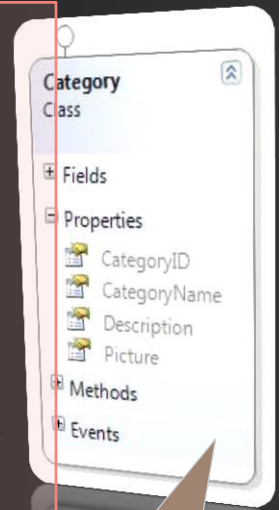
Entity Framework Lifecycle

- When the application starts
 - Entity Framework translates into SQL the language-integrated queries in the object model
 - Sends them to the database for later execution

LINQ to Entity Files – Example

- EDMX mappings file for the Categories table from Northwind database in SQL Server

```
<EntityType Name="Categories">
  <Key>
    <PropertyRef Name="CategoryID" />
  </Key>
  <Property Name="CategoryID" Nullable="false"
    Type="int" StoreGeneratedPattern="Identity" />
  <Property Name="CategoryName" Type="nvarchar"
    Nullable="false" MaxLength="15" />
  <Property Name="Description" Type="ntext" />
  <Property Name="Picture" Type="image" />
</EntityType>
```



**Entity
class
Category**

TheObjectContext Class

- ObjectContext class is generated by the VS designer
- ObjectContext provides:
 - Ability to manipulate Database data through entity classes (read, modify, delete, insert)
 - Easily navigate through the table relationships
 - Executing LINQ queries as corresponding native SQL queries
 - Create new databases using its database schema

UsingObjectContext Class

- First create instance of ObjectContext
- In the constructor you can pass a Database connection and mapping source
- ObjectContext properties
 - Connection – the SqlConnection to be used
 - CommandTimeout – timeout for Database language execution
 - All entity classes (tables) are listed as Properties

```
NorthwindEntities northwindEntities =  
    new NorthwindEntities();
```

Logging the Native Queries

- To print the native Database SQL queries call the cast the object to `ObjectQuery` and call the method `ToTraceString()`;

```
var p = CountriesEntities.Countries;  
Console.WriteLine((query as  
ObjectQuery).ToTraceString());
```

- ◆ **This will print the queries executed to select the Countries**
 - ◆ Can be printed to file using `StreamWriter` class instead of `Console` class

Creating New Data

- To create a new row use the method `AddObject(...)` of the corresponding collection

```
// Create new order object
Order order = new Order()
{
    OrderDate = DateTime.Now, ShipName = "Titanic",
    ShippedDate = new DateTime(1912, 4, 15),
    ShipCity = "Bottom Of The Ocean"
};
// Mark the object for inserting
context.Orders.AddObject(order);
context.SaveChanges();
```

This will
execute an
SQL INSERT

- ◆ **`SaveChanges()` method call is required to perform the insert actions**

Creating New Data (2)

- Creating new row can also be done by using the `AddTo+The_Entity_Name` method directly on the `ObjectContext`
 - This method is deprecated
 - Better use the other one

```
// Mark the object for inserting  
context.AddToOrders(order);  
context.SaveChanges();
```

Creating New Data with Cascading

- We can also add data cascading in the database:

```
Country spain = new Country();  
spain.Name = "Spain";  
spain.Population = "46 030 10";  
spain.Cities.Add( new City { Name = "Barcelona" } );  
spain.Cities.Add( new City { Name = "Madrid" } );  
countryEntities.Countries.AddObject(spain);  
countryEntities.SaveChanges();
```

- ◆ **This way we don't have to add each City individually**
 - ◆ They will be added when the **Country Entity (spain)** is inserted in the database

Reading Data with LINQ

- Reading data from EF entity:

```
NorthwindEntities northwindEntities =  
    new NorthwindEntities();  
var customers =  
    from c in context.Customers  
    where c.City == "London"
```

Transformed to SQL
and is executed at
the sever

- Customers property in theObjectContext:

```
public ObjectSet<Customer> Customers  
{  
    get{  
        if ((_Customers == null))  
        { _Customers  
            =base.CreateObjectSet<Customer>("Customers");  
        }  
        return _Customers;  
    }  
}
```


Updating Data

- `ObjectContext` allows modifying object properties and persisting them in the Database
 - Just load an entity, modify it and call `SaveChanges()`
- The `ObjectContext` automatically tracks all changes made on its entity objects

```
Order order = northwindEntities.Orders.First();  
order.OrderDate = DateTime.Now;  
context.SaveChanges();
```

This will
execute an
SQL UPDATE

This will execute an SQL
SELECT to load the first
order

Deleting Data

- Delete is done by `DeleteObject()` on the current entity collection
- `SaveChanges()` method performs the delete action

```
Order order = northwindEntities.Orders.First();  
// Mark the entity for deleting on the next save  
northwindEntities.Orders.DeleteObject(order);  
northwindEntities.SaveChanges();
```

**This will
execute an
SQL DELETE**

Executing Native Queries

- Examples are shown in SQL Server but the same can be done for all the other databases
- In EF native queries can be executed as well

```
string queryString = @"SELECT VALUE  
    Customer FROM NorthwindEntities.Customers AS Customer  
    WHERE Customer.City=='London'";  
  
var customerQuery = new ObjectQuery<Customer>(queryString, northwindEntities);
```

- ◆ **The native SQL Query is in the queryString**

Executing Native Queries

- Native queries can also be parameterized
 - Can be done by using `Parameters.Add(...)`;

```
string queryString = @"SELECT VALUE Customer FROM
NorthwindEntities.Customers AS Customer
WHERE Customer.City== @city";

var customerQuery = new ObjectQuery<Customer>(
queryString, northwindEntities);

customerQuery.Parameters.Add(
new ObjectParameter("city", "London"));
```



Executing Native SQL Queries

Live Demo



The N+1 Query Problem

What is the N+1 Query Problem?

How to avoid it?



The N+1 Query Problem

- What is the N+1 Query Problem?
 - Imagine a Database that contains tables Countries and Cities
 - They are connected with one-to-many relationship
 - We want to print each Country with the its Cities:

```
foreach (var country in countriesEntities.Countries)
{
    Console.WriteLine(country.CountryName + "\nCities:");
    foreach (var city in country.Cities)
    {
        Console.WriteLine("{0}", city.CityName);
    }
}
```

The N+1 Query Problem

**N Queries to the Database
(one for each Country)**

**One Query to
the Database**

```
foreach (var country in CountriesEntities.Countries)
{
    Console.WriteLine(country.CountryName + "\nCities:");
    foreach (var city in country.Cities)
    {
        Console.WriteLine("{0}", city.CityName);
    }
}
```

- All in all N+1 queries to the Database
- Imagine there are 100 Countries and each Country has at least 100 Cities
 - That is at least 10000 queries
 - Too many queries

Resolving the N+1 Query Problem

- Fortunate **Using Include(...) method only one query is made to the Database**

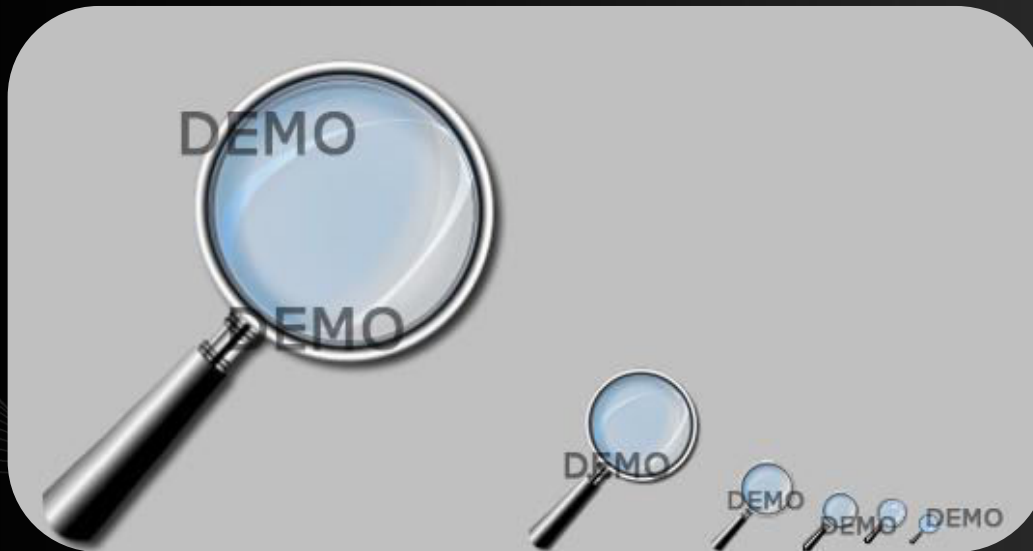
```
foreach (var country in countries.Entities.Countries.  
        Include("Cities"))  
{  
    foreach (var city in country.Cities)  
    {  
        Console.WriteLine(" {0}", city.CityName);  
    }  
}
```

**No Additional
Queries are**

made

Resolving the N+1 Query Problem

Live Demo



Joining and Grouping Tables

Join and Group Using LINQ

Joining Tables

- When working with EF using LINQ is available
 - That means that joining two Entities is the same as joining two collections

Using Join Extension Method

```
var customerSupplier =  
    from customer in northwindEntities.Customers  
    join supplier in northwindEntities.Suppliers  
    on customer.Country equals supplier.Country  
    select new {  
        CustomerName = customer.CompanyName,  
        Supplier = supplier.CompanyName,  
        Country = customer.Country  
    };
```

```
northwindEntities.Customers.  
Join(northwindEntities.Suppliers,  
    (c=>c.Country), (s=>s.Country), (c,s)=>  
    new {Customer = c.CompanyName, Supplier =  
        s.CompanyName, Country = c.Country });
```

Grouping Tables

- Grouping also can be done by using LINQ
 - Again grouping is the same as with a collection in LINQ

◆ Using LINQ

```
var groupedCustomers =  
    from customer in  
northwindEntities.Customers  
    group customer by Customer.Country;
```

◆ Using Extension Methods

```
var groupedCustomers =  
    northwindEntities.Customers.GroupBy(  
        customer => customer.Country);
```

Joining and Grouping Tables

Live Demo





Attaching and Detaching Objects

- In the Entity Framework, objects can be attached to or detached from an object context
- Attached objects are tracked and managed by the `ObjectContext`
- Detached objects are not referenced by the `ObjectContext`

Attaching Detached Objects

- When a query is executed inside an `ObjectContext`, the returned objects are automatically attached to it
 - You might attach objects that have been previously detached

Detaching Objects

- When an object is detached?
 - When we obtain the object from an `ObjectContext` and the `Dispose` it
 - Detach the object manually
 - By calling the `Detach(...)` method of the entity set

```
Product GetProduct(int id)
{
    using (NorthwindEntities northwindEntities =
        new NorthwindEntities())
    {
        return northwindEntities.Products.First(
            p => p.ProductID == id);
    }
}
```

Attaching Objects

- When we want to update the detached object we need to reattach it and the update it
- Done by the `Attach(...)` method of the entity set

```
void UpdateProduct(Product product)
{
    using (NorthwindEntities northwindEntities =
        new NorthwindEntities())
    {
        northwindEntities.Products.Attach(product);
        product.UnitPrice = 123m;
        northwindEntities.SaveChanges();
    }
}
```

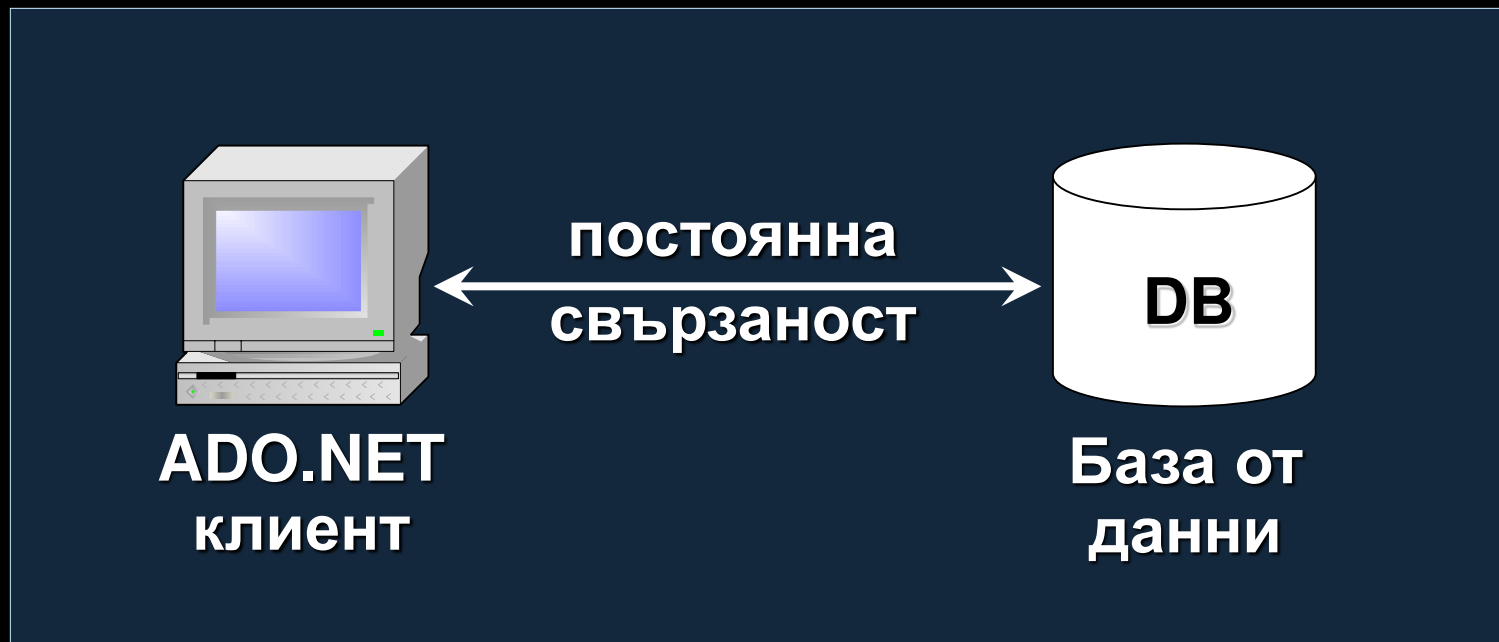
ADO.NET

Модели за работа с данни

- ◆ **Свързан модел (connected model)**
 - ❖ **Постоянна връзка с данните (online)**
 - ❖ **Случаи на използване**
- ◆ **Несвързан модел (disconnected model)**
 - ❖ **Връзката с данните се осъществява offline – за изтегляне на данни и нанасяне на промени по данните**
 - ❖ **Случаи на използване**
 - ❖ **Примери**
 - ❖ **Достъп до данни чрез Web услуга**
 - ❖ **Интеграция с XML**

Свързан модел

- ◆ Реализация на достъп до данни в среда, в която винаги има връзка до източника на данните



Свързан модел – за и против

◆ Предимства:

- ❖ Средата е по-лесна за подsigуряване (по-малко усилия за разработчика)
- ❖ Контролът върху конкурентният достъп се упражнява по-лесно
- ❖ По-добра вероятност за работа с текущата версия на данните

◆ Недостатъци:

- ❖ Нужда от постоянна мрежова връзка
- ❖ Проблеми при нужда от скалируемост

Несвързан модел

- ◆ Подмножество на данните от централната система за съхранение на данните се копира локално при клиента



Несвързан модел – за и против

◆ Предимства:

- ❖ Клиентът се свързва, когато има нужда, а в останалото време работи без връзка с централната база от данни
- ❖ Други потребители могат да се свързват междувременно
- ❖ Скалируемостта е доста добра

◆ Недостатъци:

- ❖ Данните не винаги са текущи
- ❖ Допълнителни усилия за решаване на конфликтите между различните версии

Еволюция на приложенията

- ◆ **Еднослойни приложения**
 - ❖ **Най-често работи само един потребител**
 - ❖ **Предимства**
 - ❖ **Всички компоненти са на едно място**
 - ❖ **Недостатъци**
 - ❖ **Промяна на функционалността изисква преинсталация**
 - ❖ **Пример**
 - ❖ **Приложение базирано на MS Access**

Еволюция на приложенията

- ◆ **Двуслойни приложения (клиент-сървър)**
 - ❖ Потребителският интерфейс и бизнес правилата се дефинират на едно място
 - ❖ Данните се съхраняват във втория слой
 - ❖ Предимства
 - ❖ Има разделяне на функционалността
 - ❖ Недостатъци
 - ❖ Лоша скалируемост – проблеми с поддръжката на голям брой клиенти
 - ❖ Примери
 - ❖ MS SQL Server ↔ MS Query Analyzer
 - ❖ MS Exchange ↔ MS Outlook

Еволюция на приложенията

◆ Трислойни приложения

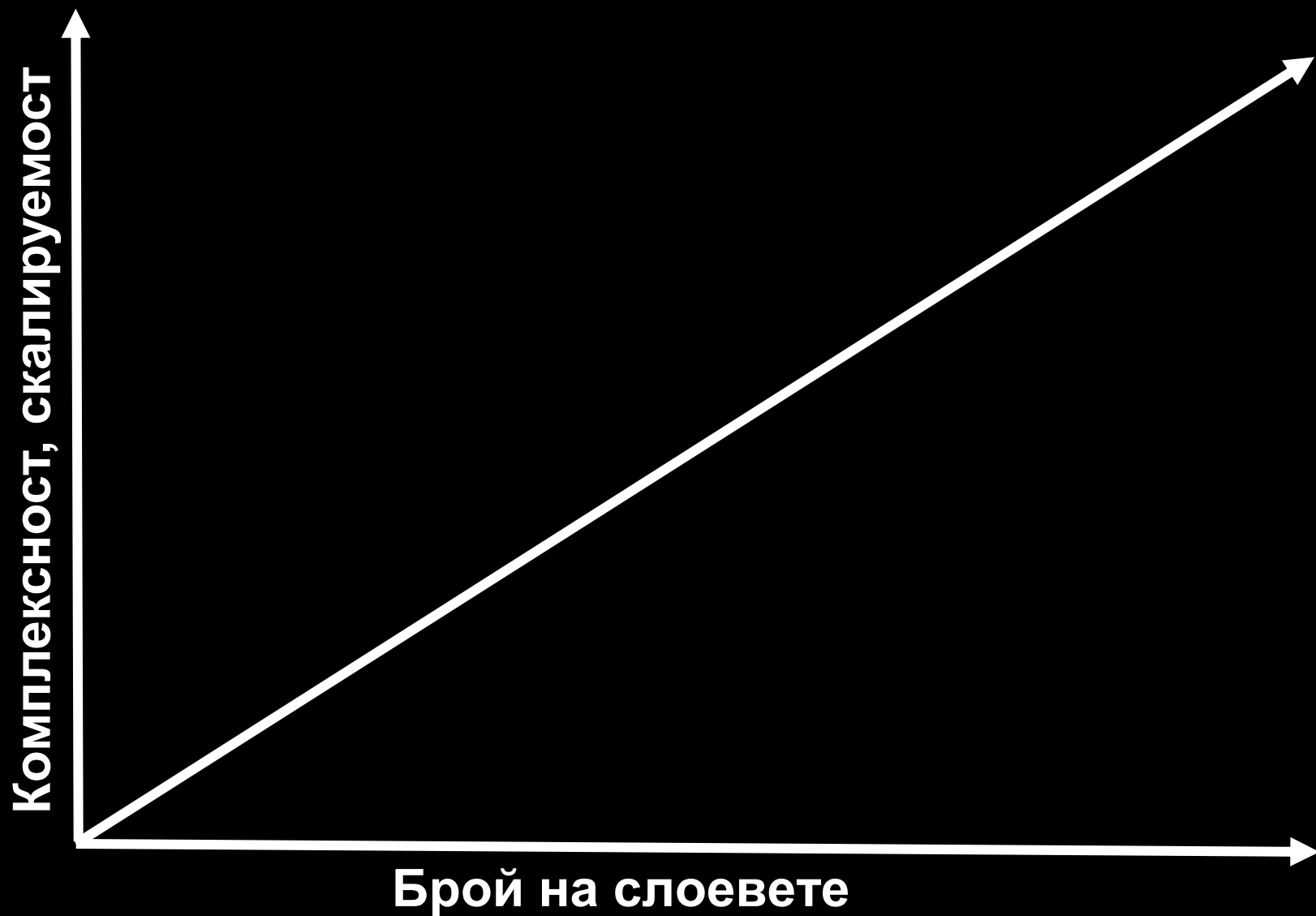
- ❖ Различните типове функционалност са в различни слоеве
- ❖ Предимства
 - ❖ Отделяне на функционалността между потребителски интерфейс, бизнес правила и съхранение / достъп до данните
- ❖ Недостатъци
 - ❖ По-трудна поддръжка
 - ❖ Повече усилия за осигуряване на сигурността
- ❖ Пример
 - ❖ ASP.NET Web-приложение ↔ ASP.NET Web услуга ↔ MS SQL Server

Еволюция на приложенията

◆ Многослойни приложения

- ❖ Системи с повече от 3 логически слоя
- ❖ Възможност за добавяне на още слоеве за разширяване на функционалността
- ❖ Предимства
 - ❖ Възможност за различни приложения да достъпват части от функционалността през отворени протоколи
- ❖ Недостатъци
 - ❖ Много труден процес по дефиниране и реализация на правила за сигурен достъп
 - ❖ Изискват повече планиране и по-големи срокове за разработка

В търсене на баланса



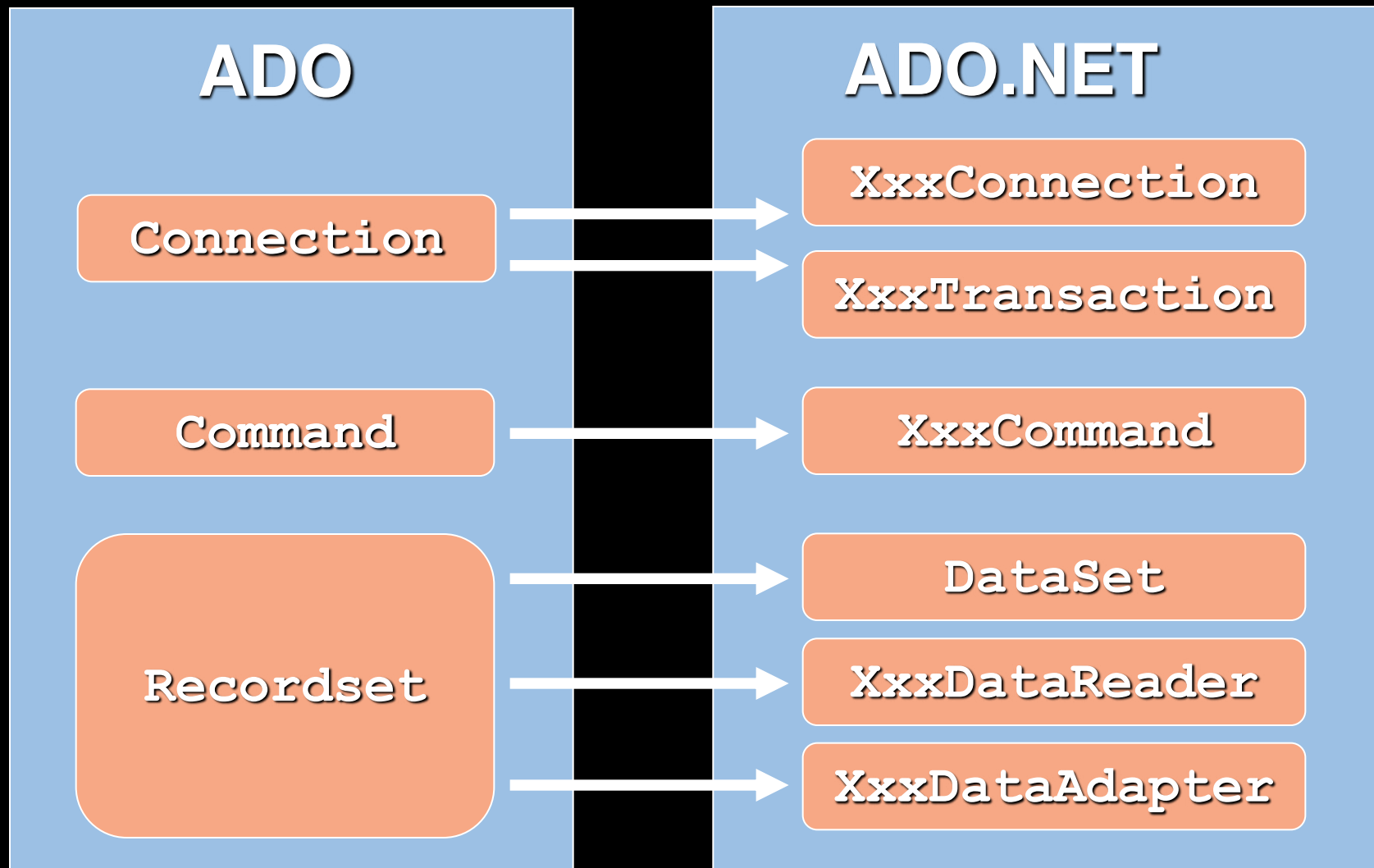
Какво е ADO.NET?

- ◆ Набор от класове за работа с данни
 - ❖ Набор от класове, интерфейси, структури и други типове за достъп до данни през изцяло .NET базирана реализация
 - ❖ Програмен модел за работа с данни
 - ❖ Осигурява възможност за работа в несвързана среда
 - ❖ Осигурява връзка с XML
 - ❖ Наследник на ADO (Windows технология за достъп до бази от данни)

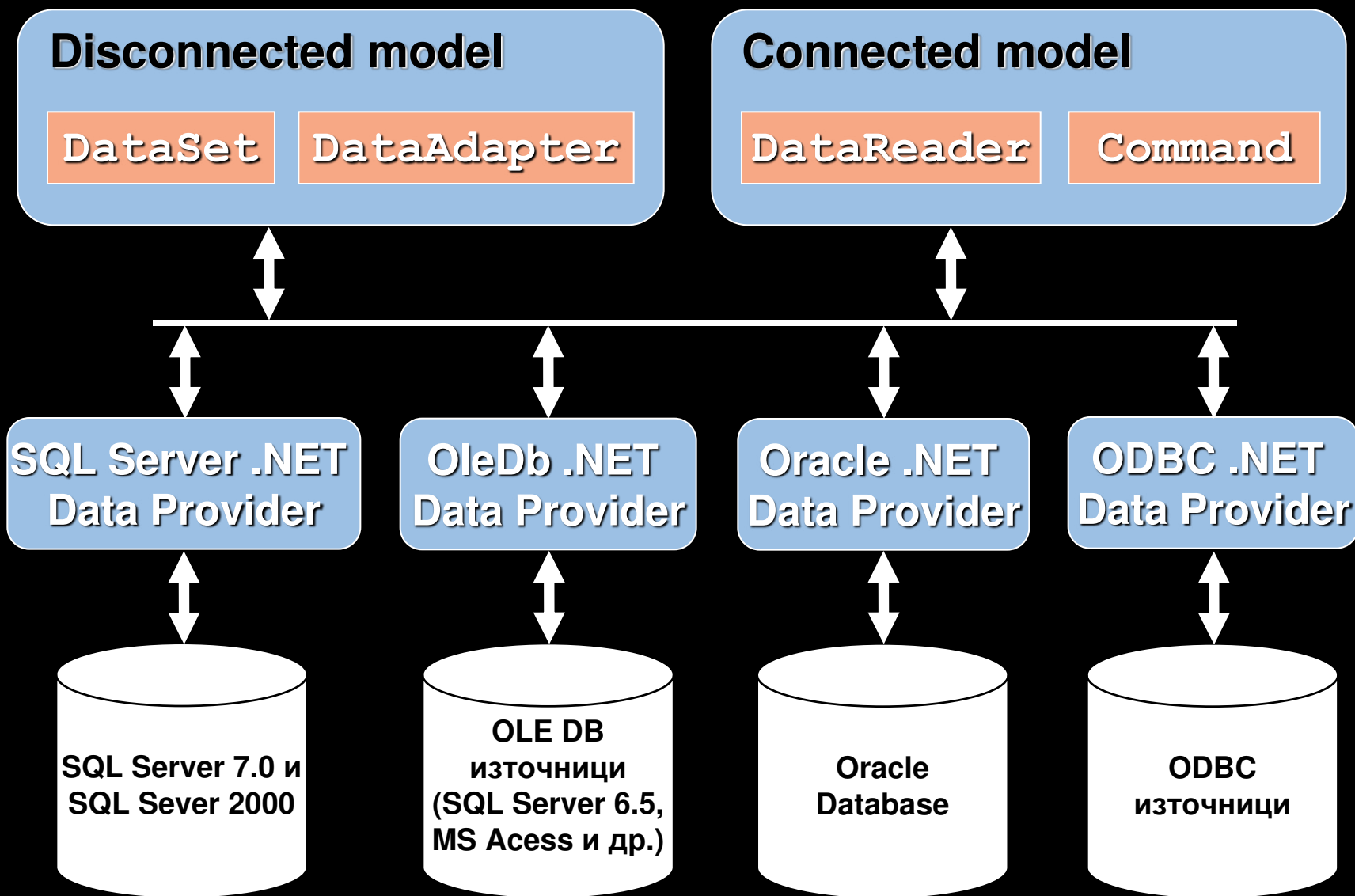
Namespace-и на ADO.NET

- ◆ Пространства от имена от ADO.NET
 - ❖ `System.Data` – основни архитектурни класове на ADO.NET
 - ❖ `System.Data.Common` – общи класове за всички data Provider-и
 - ❖ `System.Data.SqlClient` и `System.Data.SqlTypes` – Data Provider класове за достъп до SQL Server
 - ❖ `System.Data.OleDb` – връзка с OleDb
 - ❖ `System.Data.Odbc` – връзка с ODBC
 - ❖ `System.Xml` – връзка с XML

Еволюция на ADO към ADO.NET



Компоненти на ADO.NET



Data Provider-и в ADO.NET

- ◆ Data Provider-ите са съвкупности от класове, които осигуряват връзка с различни бази от данни
 - ❖ За различните RDBMS системи се използват различни Data Provider-и
 - ❖ Различните производители използват различни протоколи за връзка със сървърите за данни
 - ❖ Дефинират се от 4 основни обекта:
 - ❖ `Connection` – за връзка с базата
 - ❖ `Command` – за изпълнение на SQL
 - ❖ `DataReader` – за извличане на данни
 - ❖ `DataAdapter` – за връзка с `DataSet`

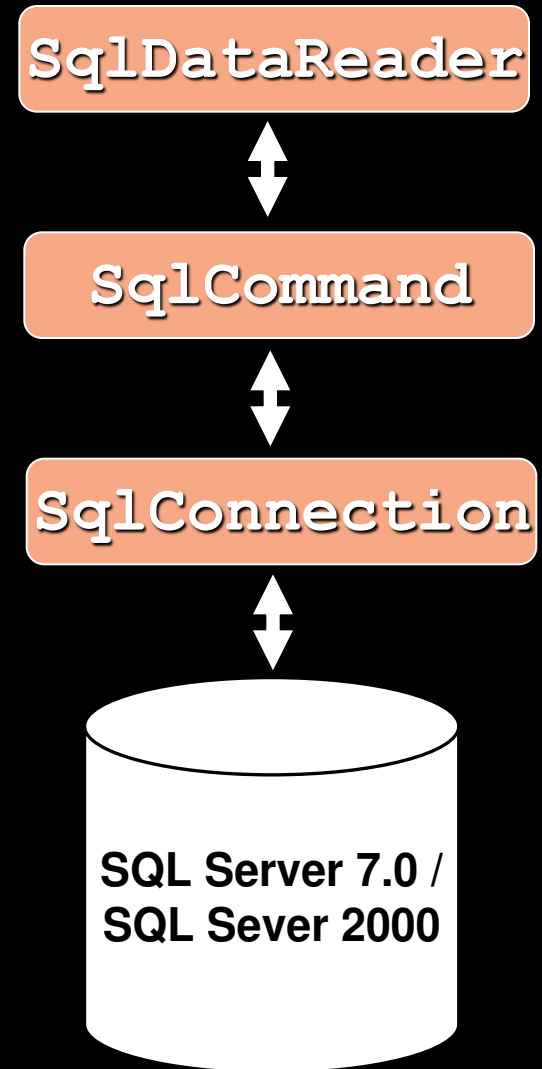
Data Provider-и в ADO.NET

- ◆ В ADO.NET има няколко стандартни Data Provider-a
 - ❖ `SqlClient` – за връзка със SQL Server
 - ❖ `OleDb` – за връзка със стандарта OleDb
 - ❖ `Odbc` – за връзка със стандарта ODBC
 - ❖ `Oracle` – за връзка с Oracle
- ◆ Трети доставчици предлагат Data Provider-и за връзки с други RDBMS:
 - ❖ IBM DB2
 - ❖ MySQL
 - ❖ PostgreSQL
 - ❖ Borland Interbase / Firebird

ADO.NET в свързана среда

◆ Данните са на сървъра до затваряне на връзката

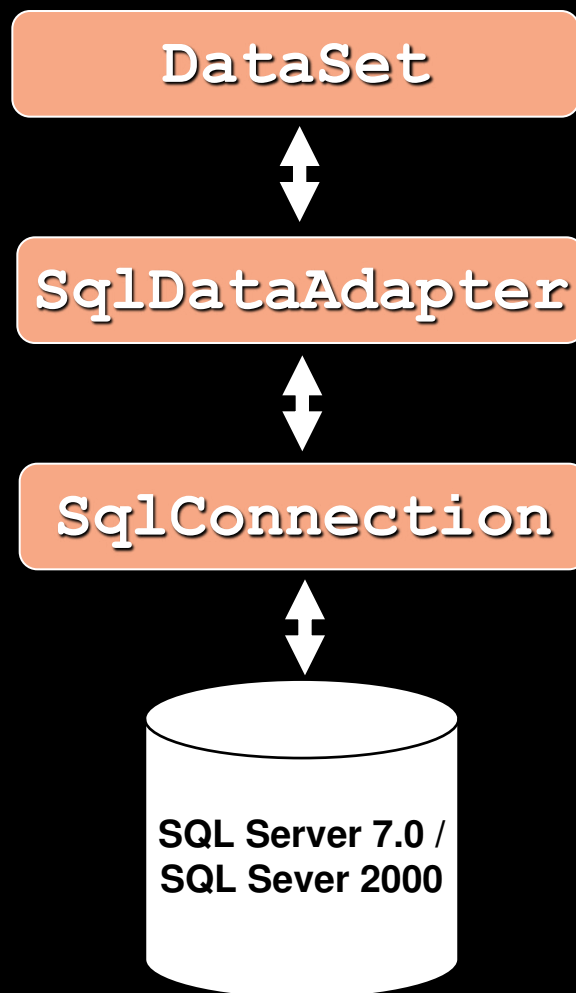
1. Отваряне на връзка (SqlConnection)
2. Изпълнение на команда / команди (SqlCommand)
3. Обработка на редовете получени като резултат от заявката чрез четец (SqlDataReader)
4. Затваряне на четеца
5. Затваряне на връзката



ADO.NET в несвързана среда

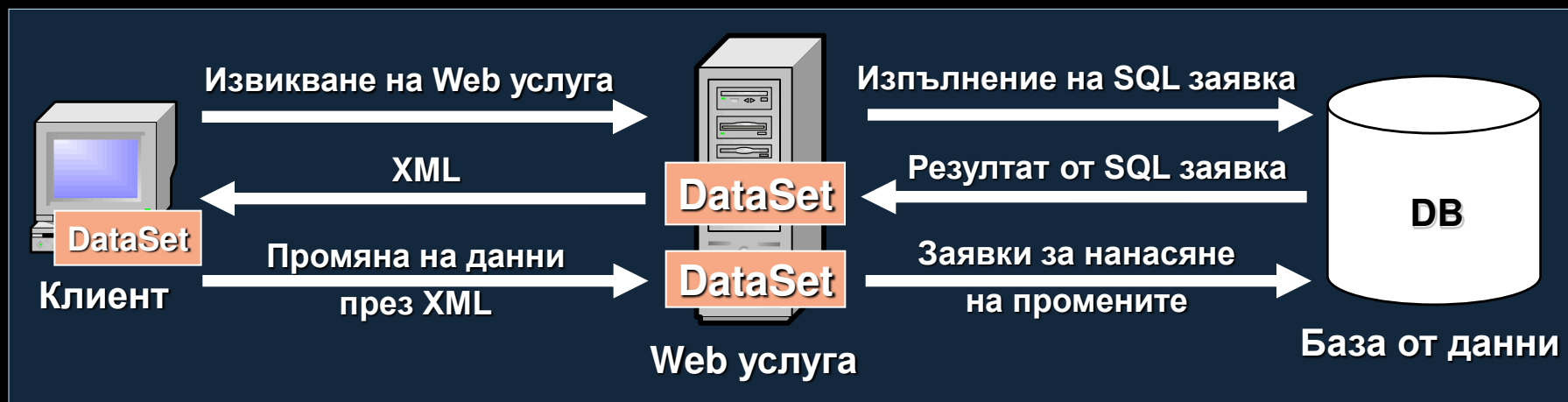
- ◆ Данните се кешират в DataSet обект и връзката се преустановява

1. Отваряне на връзка (SqlConnection)
2. Пълнене на DataSet (чрез SqlDataAdapter)
3. Затваряне на връзката
4. Работа със DataSet-а
5. Отваряне на връзка
6. Нанасяне на промени по данните по сървъра
7. Затваряне на връзката



ADO.NET, XML и Web услуги

- ◆ ADO.NET е тясно интегрирано с XML
 - ❖ Често се използва в несвързан сценарий посредством Web услуги
 - ❖ Web-услугата реализира бизнес слоя на трислойните приложения
 - ❖ Извършва бизнес операциите над данните



SqlClient Data Provider

- ◆ **SqlConnection** – осъществява връзката с MS SQL Server
- ◆ **SqlCommand** – изпълнява команди върху SQL Server-а през вече установена връзка
- ◆ **SqlDataReader** – служи за извличане на данни от SQL Server-а
 - ❖ Данните са резултат от изпълнена команда
- ◆ **SqlDataAdapter** – обменя данни между DataSet обекти и SQL Server
 - ❖ Осигурява зареждане на DataSet с данни и обновяване на променени данни
 - ❖ Може да се грижи сам за състоянието на връзката с базата данни

Система за сигурност в SQL Server

- ◆ Видове автентикация
 - ❖ Windows (чрез потребител в ОС)
 - ❖ Сигурна валидация и криптиране
 - ❖ Одитиране на достъп
 - ❖ Политики за пароли
 - ❖ Заклучване на акаунти
 - ❖ Смесена (Windows и SQL Server)
 - ❖ Основно за съвместимост с предишни версии
 - ❖ Малка разлика в скоростта

Символен низ за връзка с база от данни (Connection String)

- ◆ Дефинира параметрите, необходими за осъществяване на връзка с базата данни
- ◆ Основни параметри
 - ❖ Provider – име на драйвера за достъп
 - ❖ Data Source – идентификатор на базата
 - ❖ User ID/Password
 - ❖ Integrated Security
 - ❖ Persist Security Info (False)
- ◆ Пример:

```
Server=localhost; Database=Pubs;  
Integrated Security=true; Persist  
Security Info=false
```

SqlConnection – пример

```
const string CONNECTION_STRING =  
    "Server=localhost; Database=Northwind; " +  
    "Integrated Security=true; " +  
    "Persist Security Info=false";  
  
// Create the connection  
SqlConnection con =  
    new SqlConnection(CONNECTION_STRING);  
using (con)  
{  
    // Open connection  
    con.Open();  
  
    // Use the connection here  
    // ...  
}
```

Connection Pooling

- ◆ "Database Connection Pooling" е програмна техника, която:
 - ❖ Осигурява по-ефективно използване на връзките към базата от данни
 - ❖ Подобрява производителността
- ◆ Пулингът работи така:
 - ❖ При отваряне на връзка тя се взима готова от т. нар. "пул" (connection pool)
 - ❖ Ако там няма свободна се отваря нова
 - ❖ При затваряне на връзка тя се връща в пула без да се прекъсва физически
- ◆ По подразбиране SqlConnection ползва пул

Още за работата със `SqlConnection`

- ◆ Експлицитно отваряне и затваряне на връзката (работи с пула)
 - ❖ `Open()` и `Close()` методите
- ◆ Имплицитно отваряне и затваряне на връзката (работи с пула)
 - ❖ Възможно е само при работа с `DataAdapter`-и
- ◆ Използване на `Dispose()` метода
 - ❖ Извиква се ако ползваме `using (con)`
 - ❖ Връща връзката в `connection pool`-а

Събитията на SqlConnection

- ◆ Събития възникващи от връзката:
 - ❖ StateChange и InfoMessage
- ◆ StateChangeEventArgs класът
 - ❖ Дава информация за това какво се е случило с връзката към базата данни
 - ❖ CurrentState и OriginalState
- ◆ SqlInfoMessageEventArgs
 - ❖ Errors – SqlErrorCollection колекция от съобщения за грешки и предупреждения
 - ❖ Message – съобщението за грешка
 - ❖ Source – къде е възникнала грешката/предупреждението

StateChange – пример

```
private const string CONNECTION_STRING = "Server=.;" +  
    " Database=Pubs; Integrated Security=true";  
  
private SqlConnection mConn;  
  
private void InitConnection()  
{  
    mConn = new SqlConnection(CONNECTION_STRING);  
    mConn.StateChange +=  
        new StateChangeEventHandler(ConnStateChange);  
    mConn.Open();  
}  
  
private void ConnStateChange(object sender,  
    StateChangeEventArgs e)  
{  
    Debug.WriteLine("SQL Server connection: " +  
        e.OriginalState.ToString() + " --> " +  
        e.CurrentState.ToString());  
}
```

InfoMessage – пример

```
private const string CONNECTION_STRING = "Server=.; " +
    "Database=Pubs; Integrated Security=true";

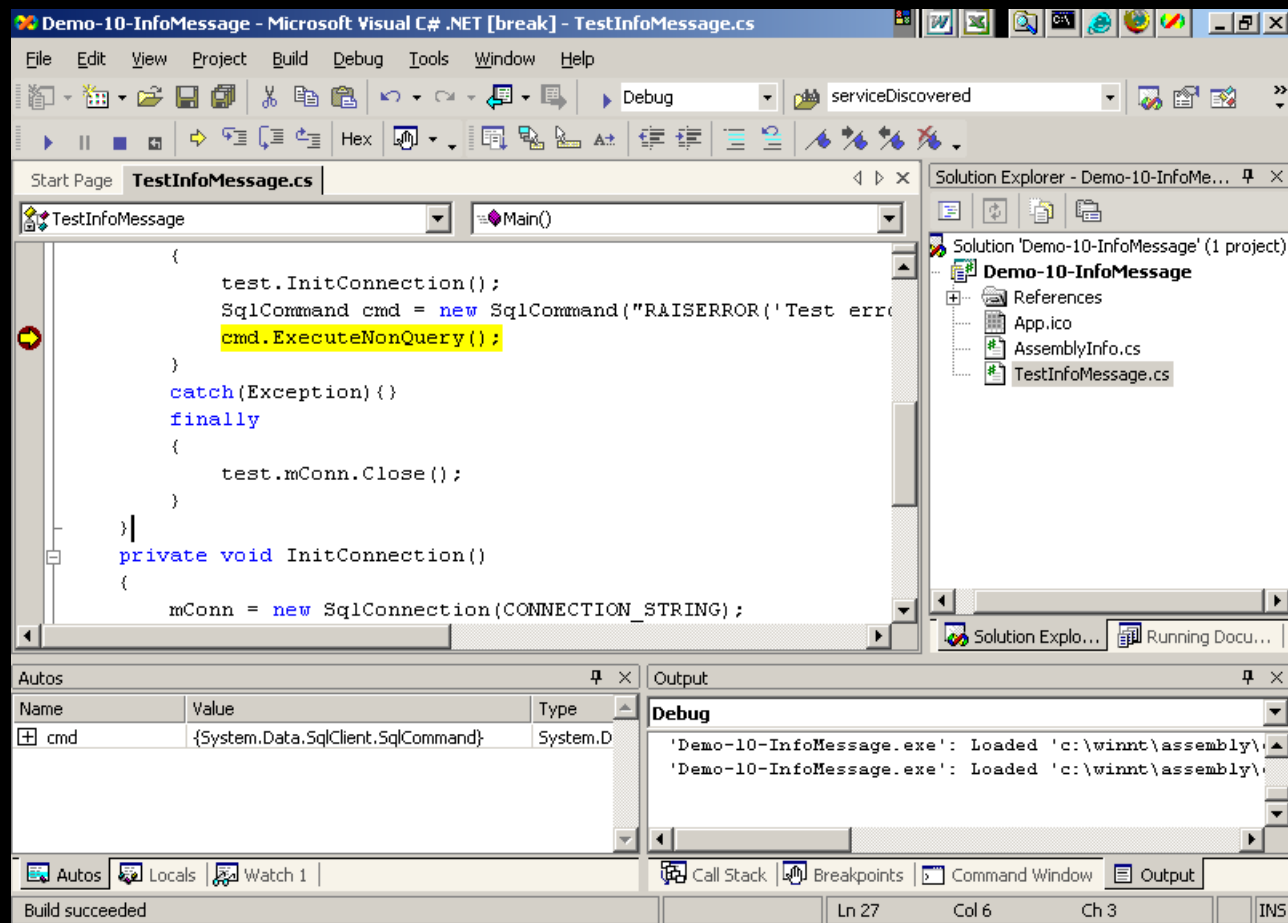
private SqlConnection mConn;

private void InitConnection()
{
    mConn = new SqlConnection(CONNECTION_STRING);
    mConn.InfoMessage +=
        new SqlInfoMessageEventHandler(ConnInfoMessage);
    mConn.Open();
}

private void ConnInfoMessage(object sender,
    SqlInfoMessageEventArgs e)
{
    Debug.WriteLine("SQL Server message: " +
        e.Message + "    Source: " + e.Source);
}
```

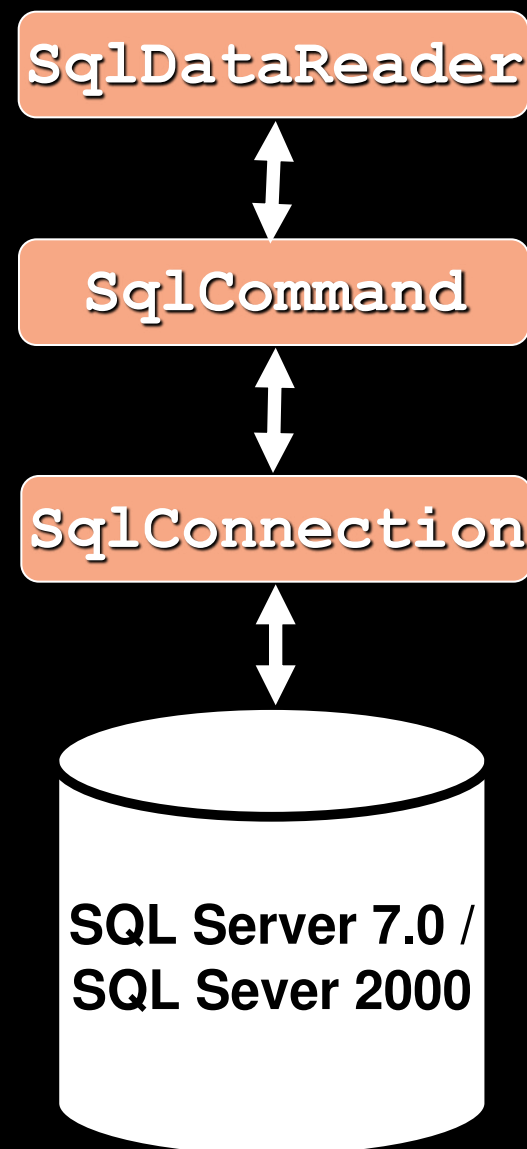

Демонстрация

◆ Проследяване на събитията StateChange и InfoMessage

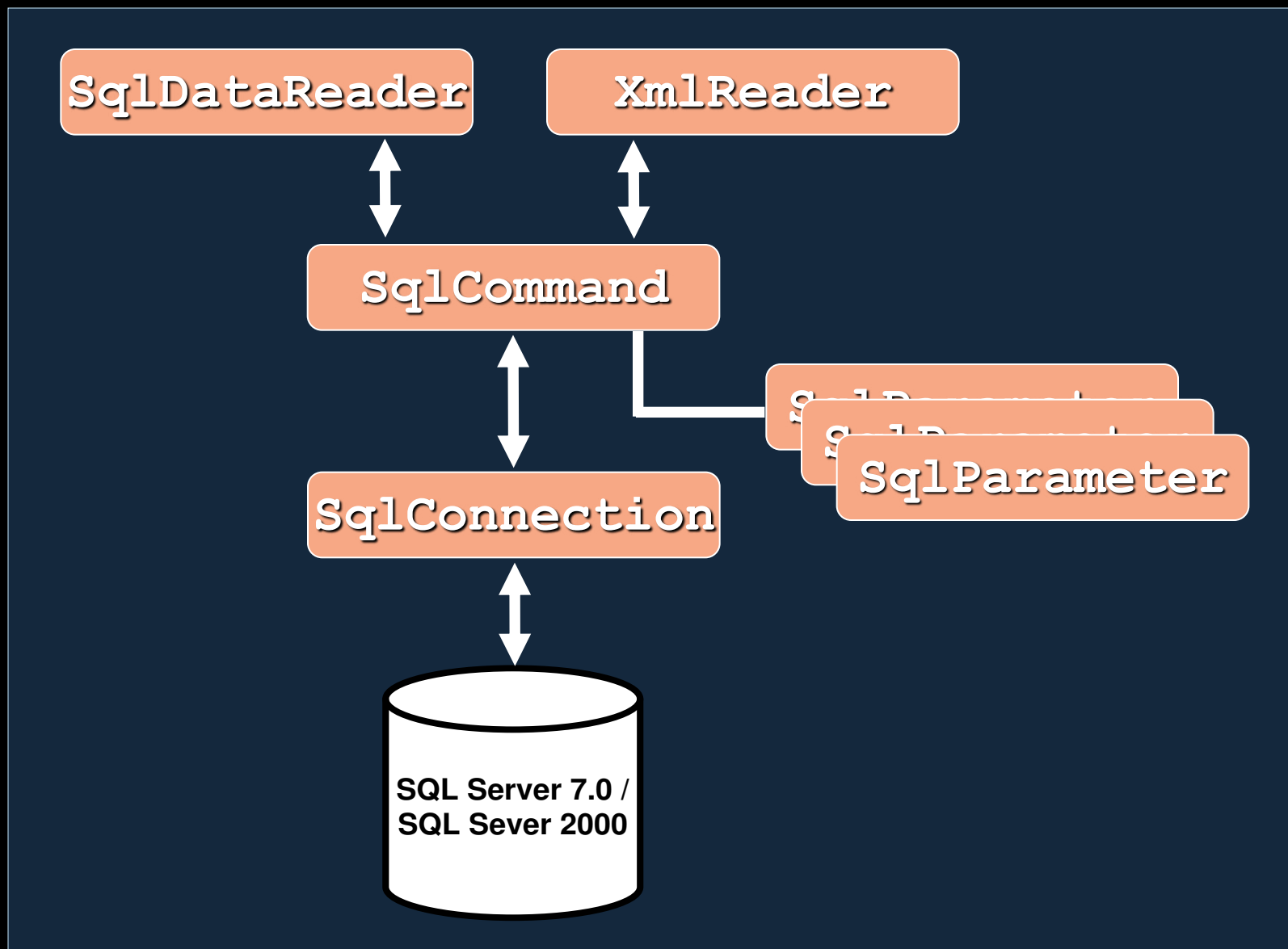


Реализация на свързан модел

- ◆ Данните са на сървъра до затваряне на връзката
 1. Отваряне на връзка (SqlConnection)
 2. Изпълнение на команда / команди (SqlCommand)
 3. Обработка на редовете получени като резултат от заявката чрез четец (SqlDataReader)
 4. Затваряне на четеца
 5. Затваряне на връзката



Класовете в свързана среда



Класът SqlCommand

- ◆ Изпълнява обръщение към SQL заявка или съхранена процедура
 - ❖ По-важни свойства
 - ❖ `Connection` – връща / задава `SqlConnection`-а на командата
 - ❖ `CommandType` – тип команда
 - ❖ `CommandType.StoredProcedure`
 - ❖ `CommandType.TableDirect`
 - ❖ `CommandType.Text`
 - ❖ `CommandText` – SQL заявка или име на съхранена процедура
 - ❖ `Parameters` – параметри

Класът SqlCommand

◆ По-важни методи

❖ ExecuteScalar()

- ❖ Връща единична стойност (първата колона от първия ред от резултата)
- ❖ Върнатата стойност е `System.Object`

❖ ExecuteReader()

- ❖ Връща курсор (`SqlDataReader`)
- ❖ `CommandBehavior` задава настройки

❖ ExecuteNonQuery()

- ❖ Връща броя на засегнатите записи (`int`)

❖ ExecuteXmlReader()

- ❖ Връща `XmlReader` по резултата
- ❖ Поддържа се само в `SqlClient`

Класът SqlDataReader

- ◆ Извлича последователност от записи (курсор) – резултат от изпълнена команда
 - ❖ Достъпът е само за четене (read-only)
 - ❖ Достъпът е еднопосочен (forward-only)
- ◆ По-важни методи и свойства:
 - ❖ `Read()` – придвижва курсора напред и връща `false` ако няма следващ запис
 - ❖ `Item` (индексатор) – извлича стойността на колона по име или индекс
 - ❖ `Close()` – затваря курсора – трябва задължително да се вика!

Свързан модел – пример

```
using System;
using System.Data;
using System.Data.SqlClient;

class TestSqlCommand
{
    private const string CONNECTION_STRING = "Server=.;" +
        " Database=pubs; Integrated Security=true";
    private const string COMMAND_SELECT_AUTHORS =
        "SELECT au_fname, au_lname, phone FROM authors";

    static void Main()
    {
        SqlConnection con =
            new SqlConnection(CONNECTION_STRING);
        con.Open();
        try
        {
```

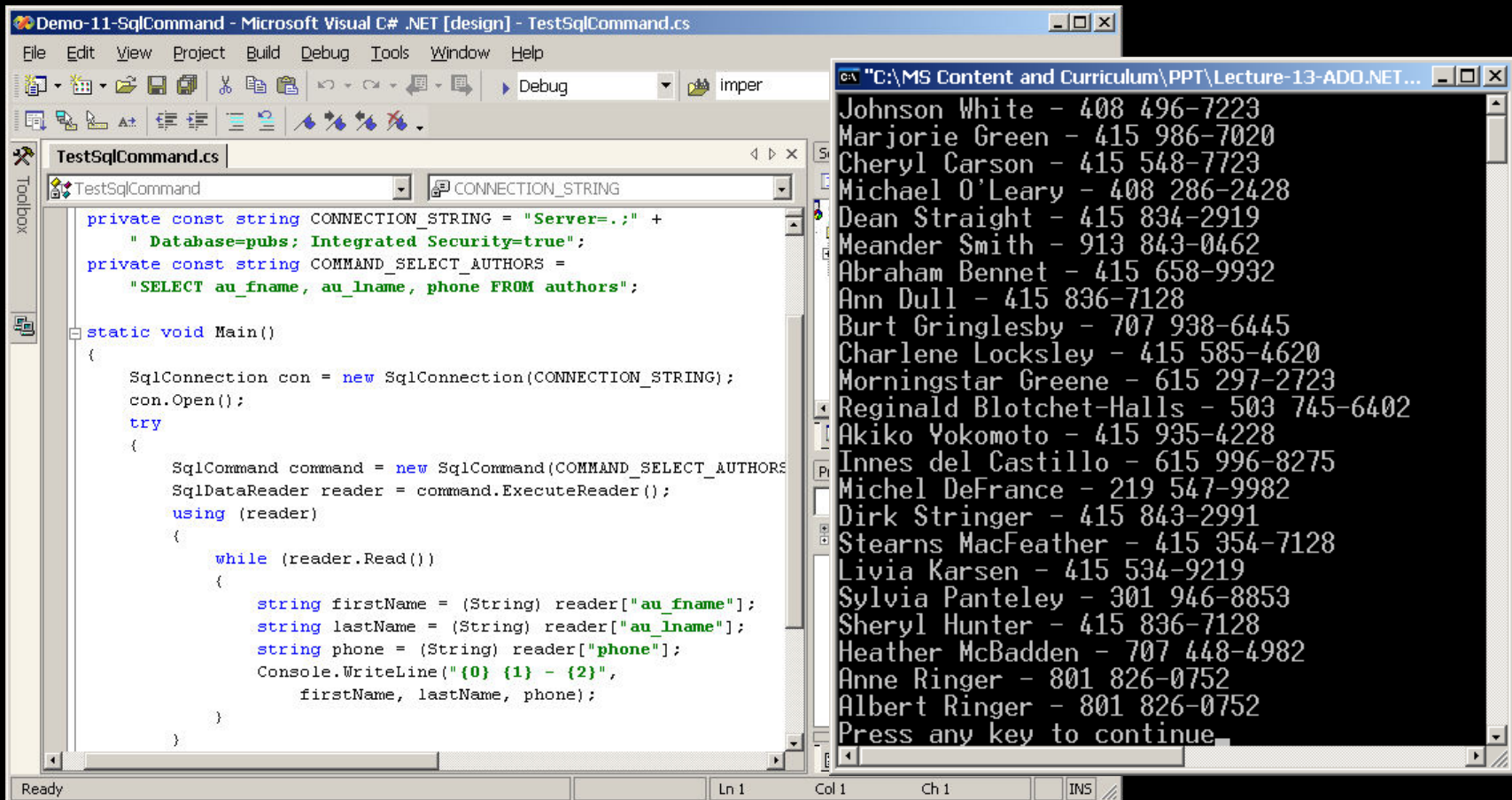
(примерът продължава)

Свързан модел – пример

```
SqlCommand command =  
    new SqlCommand(COMMAND_SELECT_AUTHORS, con);  
SqlDataReader reader = command.ExecuteReader();  
using (reader)  
{  
    while (reader.Read())  
    {  
        string firstName = (String) reader["au_fname"];  
        string lastName = (String) reader["au_lname"];  
        string phone = (String) reader["phone"];  
        Console.WriteLine("{0} {1} - {2}",  
            firstName, lastName, phone);  
    }  
}  
finally  
{  
    con.Close();  
}  
}
```


Демонстрация

◆ Реализация на свързан модел – SqlCommand и SqlDataReader



The screenshot displays the Microsoft Visual C# .NET [design] environment. The main window, titled "TestSqlCommand.cs", contains the following C# code:

```
private const string CONNECTION_STRING = "Server=.;" +  
    " Database=pubs; Integrated Security=true";  
private const string COMMAND_SELECT_AUTHORS =  
    "SELECT au_fname, au_lname, phone FROM authors";  
  
static void Main()  
{  
    SqlConnection con = new SqlConnection(CONNECTION_STRING);  
    con.Open();  
    try  
    {  
        SqlCommand command = new SqlCommand(COMMAND_SELECT_AUTHORS,  
            con);  
        SqlDataReader reader = command.ExecuteReader();  
        using (reader)  
        {  
            while (reader.Read())  
            {  
                string firstName = (String) reader["au_fname"];  
                string lastName = (String) reader["au_lname"];  
                string phone = (String) reader["phone"];  
                Console.WriteLine("{0} {1} - {2}",  
                    firstName, lastName, phone);  
            }  
        }  
    }  
}
```

The output window on the right, titled "C:\MS Content and Curriculum\PPT\Lecture-13-ADD.NET...", displays the results of the query:

```
Johnson White - 408 496-7223  
Marjorie Green - 415 986-7020  
Cheryl Carson - 415 548-7723  
Michael O'Leary - 408 286-2428  
Dean Straight - 415 834-2919  
Meander Smith - 913 843-0462  
Abraham Bennet - 415 658-9932  
Ann Dull - 415 836-7128  
Burt Gringlesby - 707 938-6445  
Charlene Locksley - 415 585-4620  
Morningstar Greene - 615 297-2723  
Reginald Blotchet-Halls - 503 745-6402  
Akiko Yokomoto - 415 935-4228  
Innes del Castillo - 615 996-8275  
Michel DeFrance - 219 547-9982  
Dirk Stringer - 415 843-2991  
Stearns MacFeather - 415 354-7128  
Livia Karsen - 415 534-9219  
Sylvia Panteley - 301 946-8853  
Sheryl Hunter - 415 836-7128  
Heather McBadden - 707 448-4982  
Anne Ringer - 801 826-0752  
Albert Ringer - 801 826-0752  
Press any key to continue
```

ASPX ADO.NET

```
<%@ Import Namespace="System.Data.OleDb" %>
```

```
<script runat="server">
```

```
sub Page_Load
```

```
dim dbconn,sql,dbcomm,dbread
```

```
dbconn=New OleDbConnection
```

```
("Provider=Microsoft.Jet.OLEDB.4.0;
```

```
data source=" & server.mappath("northwind.mdb"))
```

```
dbconn.Open()
```

```
sql="SELECT * FROM customers"
```

```
dbcomm=New OleDbCommand(sql,dbconn)
```

```
dbread=dbcomm.ExecuteReader()
```

```
customers.DataSource=dbread
```

```
customers.DataBind()
```

```
dbread.Close()
```

```
dbconn.Close()
```

```
end sub
```

```
</script>
```

```
<html>
```

```
<body>
```

```
<form runat="server">
```

```
<asp:Repeater id="customers" runat="server">
```

```
<HeaderTemplate>
```

```
<table border="1" width="100%">
```

```
<tr>
```

```
<th>Companyname</th>
```

```
<th>Contactname</th>
```

```
<th>Address</th>
```

```
<th>City</th>
```

```
</tr>
```

```
</HeaderTemplate>
```

```
<ItemTemplate>
```

```
<tr>
```

```
<td><%#Container.DataItem("companyname")%></td>
```

```
<td><%#Container.DataItem("contactname")%></td>
```

```
<td><%#Container.DataItem("address")%></td>
```

```
<td><%#Container.DataItem("city")%></td>
```

```
</tr>
```

```
</ItemTemplate>
```

```
<FooterTemplate>
```

```
</table>
```

```
</FooterTemplate>
```

```
</asp:Repeater>
```

```
</form>
```

```
</body>
```

```
</html>
```

Създаване на SqlCommand

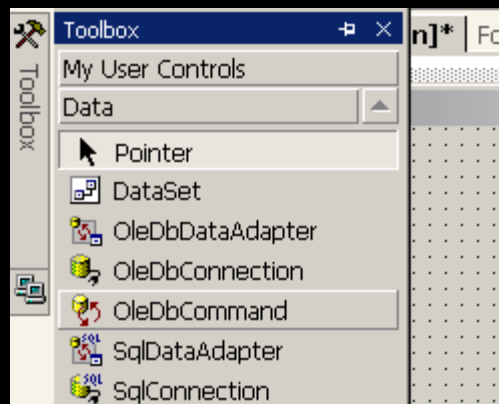
- ◆ Можете да създадете SqlCommand обект по няколко начина:

- ❖ Програмно

```
SqlCommand cmd = new SqlCommand(  
    "SELECT * FROM Products, con);
```

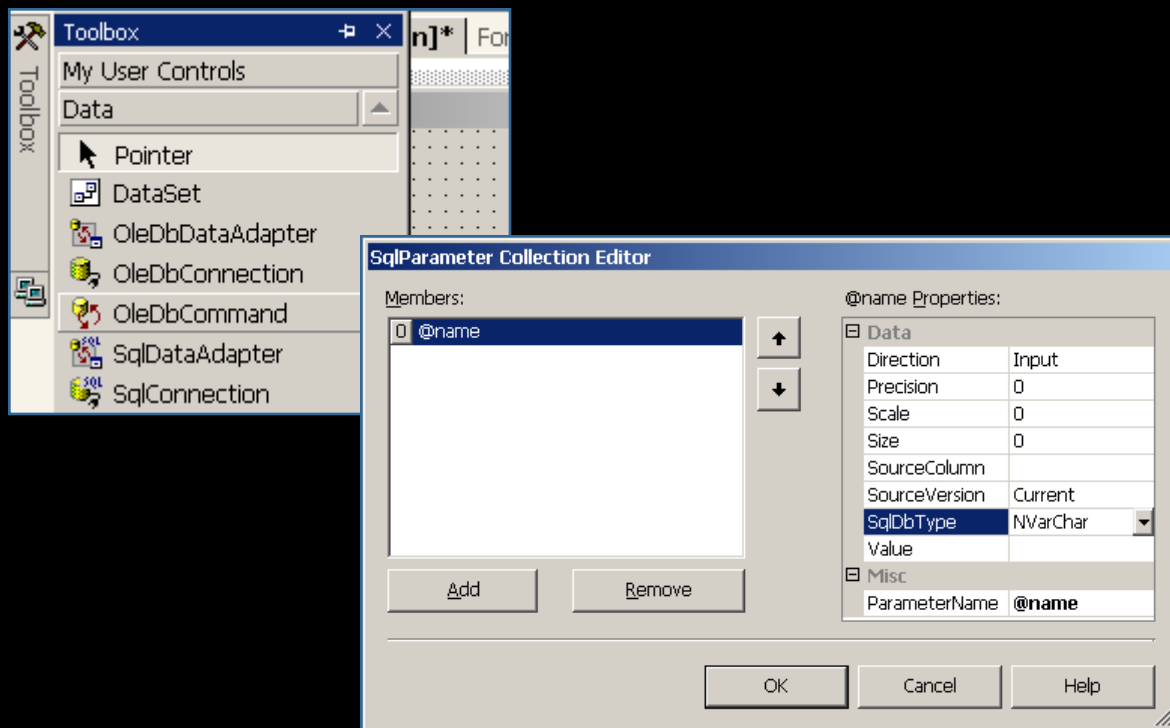
- ❖ От Server Explorer във VS.NET

- ❖ От Toolbox във VS.NET



Демонстрация

- ◆ Създаване на SqlCommand във VS.NET чрез Server Explorer и чрез Data компонентите от Toolbox-a



Класът SqlParameter

- ◆ Какво представляват SqlParameter-ите?
 - ❖ SQL заявките и съхранени процедури могат да имат входящи и изходящи параметри
 - ❖ Те се достъпват чрез Parameters колекцията на SqlCommand класа
- ◆ По-важни свойства:
 - ❖ ParameterName – име на параметъра
 - ❖ DbType – тип (NVarChar, Timestamp, ...)
 - ❖ Size – размер на типа (ако има)
 - ❖ Direction – входен, изходен, ...

Параметрични заявки – пример

```
private void InsertShipper(string aName, string aPhone)
{
    SqlCommand cmdInsertShipper = new SqlCommand(
        "INSERT INTO Shippers(CompanyName, Phone) " +
        "VALUES (@Name, @Phone)", dbConnection);

    SqlParameter paramName =
        new SqlParameter("@Name", SqlDbType.NVarChar);
    paramName.Value = aName;
    cmdInsertShipper.Parameters.Add(paramName);

    SqlParameter paramPhone =
        new SqlParameter("@Phone", SqlDbType.NVarChar);
    paramPhone.Value = aPhone;
    cmdInsertShipper.Parameters.Add(paramPhone);

    cmdInsertShipper.ExecuteNonQuery();
}
```

Първичен ключ – извличане

- ◆ Извличането на автоматично-генериран първичен ключ е специфично за всеки database сървър
- ◆ При SQL Server и MS Access се използват Identity и AutoNumber колони
 - ❖ Стойността им се излича със заявката:

```
SELECT @@Identity
```

- ◆ Пример за извличане на автоматично-генерирания първичен ключ от ADO.NET:

```
SqlCommand cmdSelectIdentity =  
    new SqlCommand("SELECT @@Identity", dbCon);  
decimal insertedRecordId =  
    (decimal) cmdSelectIdentity.ExecuteScalar();
```

Демонстрация

- ◆ Работа с параметрични SQL заявки
 - ❖ Извличане на превозвач по номер
 - ❖ Добавяне на превозвач и извличане на генерирания за него първичен ключ

```
private decimal InsertShipper(string aName, string aPhone)
{
    SqlCommand cmdInsertShipper = new SqlCommand(
        "INSERT INTO Shippers(CompanyName, Phone) " +
        "VALUES (@Name, @Phone)", mDbCon);

    SqlParameter paramName =
        new SqlParameter("@Name", SqlDbType.NVarChar);
    paramName.Value = aName;
    cmdInsertShipper.Parameters.Add(paramName);

    SqlParameter paramPhone =
        new SqlParameter("@Phone", SqlDbType.NVarChar);
    paramPhone.Value = aPhone;
    cmdInsertShipper.Parameters.Add(paramPhone);

    cmdInsertShipper.ExecuteNonQuery();

    SqlCommand cmdSelectIdentity =
        new SqlCommand("SELECT @@Identity", mDbCon);

    decimal insertedRecordId =
        (decimal) cmdSelectIdentity.ExecuteScalar();
    return insertedRecordId;
}
```

Shipper(Id=1, Name=Speedy Express, Phone=(503) 555-9831)
Inserted new shipper. ShipperID = 33
Press any key to continue.

Използване на транзакции

◆ Работа с транзакции в SQL Server:

```
BEGIN TRANSACTION

DECLARE
    @orderDetailsError int,
    @productError int

DELETE FROM "Order Details"
WHERE ProductID=42
SELECT @orderDetailsError = @@ERROR

DELETE FROM Products WHERE ProductID=42
SELECT @productError = @@ERROR

IF @orderDetailsError = 0 AND @productError = 0
    COMMIT TRANS
ELSE
    ROLLBACK TRANS
```

Използване на транзакции

◆ Работа с транзакции в ADO.NET

❖ Започване на транзакция:

```
SqlTransaction trans =  
    dbConnection.BeginTransaction();
```

❖ Въвличане на команда в дадена транзакция:

```
command.Transaction = trans;
```

❖ Потвърждаване / анулиране на транзакция:

```
trans.Commit();  
trans.Rollback();
```

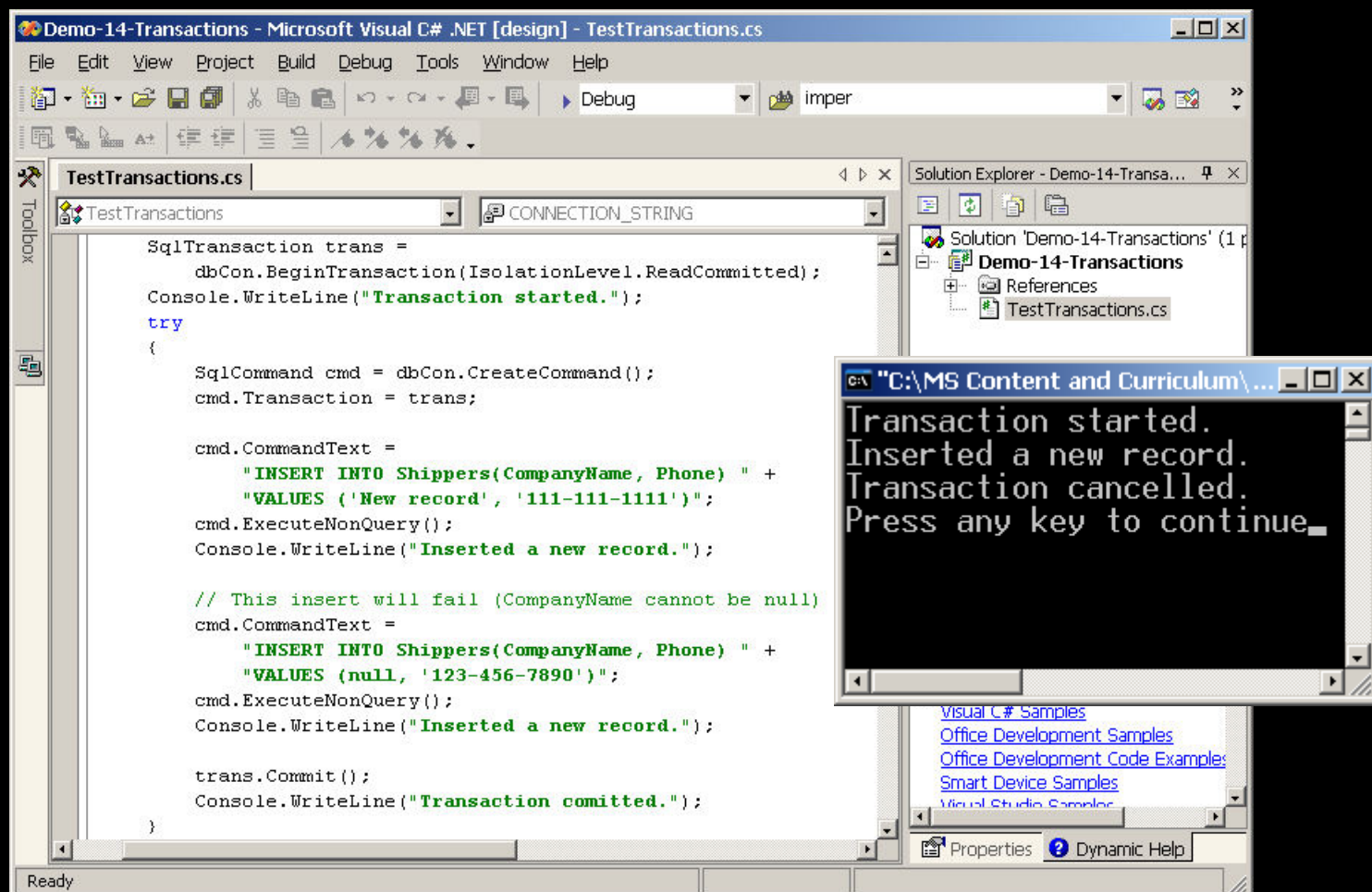
Използване на транзакции

- ◆ Нивото на изолация се дефинира с енумерацията `IsolationLevel`
- ◆ Нива на изолация на `SqlTransaction`
 - ❖ `ReadUncommitted`
 - ❖ `ReadCommitted`
 - ❖ `RepeatableRead`
 - ❖ `Serializable`
 - ❖ `Unspecified`
- ◆ Пример:

```
SqlTransaction trans = dbConnection.  
    BeginTransaction(IsolationLevel.Serializable);
```

Демонстрация

◆ Транзакции с ADO.NET



Връзка с други бази от данни

- ◆ ADO.NET поддържа връзка с различни бази от данни (освен SQL Server) чрез съответни Data Providers:
 - ❖ OLE DB – поддържа се стандартно
 - ❖ Oracle – поддържа се стандартно
 - ❖ MySQL – инсталира се допълнително
 - ❖ PostgreSQL – инсталира се допълнително
- ◆ Реализирани са интерфейсите:
 - ❖ `IDbConnection`
 - ❖ `IDbCommand`, `IDataParameter`
 - ❖ `IDataReader`
 - ❖ `IDbDataAdapter`

OLE DB Data Provider

- ◆ **OleDbConnection** – осъществява връзка с OLE DB източник на данни

```
OleDbConnection dbConn = new OleDbConnection(  
    @"Provider=Microsoft.Jet.OLEDB.4.0;Data  
    Source=C:\MyDB.mdb;Persist Security Info=False");
```

- ◆ **OleDbCommand** – изпълнява SQL команди върху OLE DB връзка към база данни
- ◆ **OleDbParameter** – параметър на команда
- ◆ **OleDbDataReader** – за извличане на данни от команда, изпълнена през OLE DB
- ◆ **OleDbDataAdapter** – обменя данни между DataSet обекти и OLE DB база данни

Връзка с OLE DB – пример

- ◆ Имаме база данни C:\Library.mdb, създадена с MS Access

- ◆ В нея има таблица Users:

Users : Table		
	Field Name	Data Type
🔑	id	AutoNumber
	username	Text
	password	Text
▶		

- ◆ Използваме "Microsoft Jet 4.0 Provider" за връзка от ADO.NET през OLE DB
- ◆ Създаваме Connection String чрез OleDbConnection компонентата от Toolbox-а на VS.NET:

```
Provider=Microsoft.Jet.OLEDB.4.0;Data Source=C:\Library.mdb;Persist Security Info=False
```

Връзка с OLE DB – пример

```
OleDbConnection dbConn = new OleDbConnection(
    @"Provider=Microsoft.Jet.OLEDB.4.0;Data Source" +
    @"=C:\Library.mdb;Persist Security Info=False");
dbConn.Open();

OleDbCommand cmd = new OleDbCommand(
    "INSERT INTO Users ([username], [password]) " +
    "VALUES (@user, @pass)", dbConn);
cmd.Parameters.Add("@user",
    OleDbType.VarChar).Value = "new user name";
cmd.Parameters.Add("@pass",
    OleDbType.VarChar).Value = "secret password";
int affected = cmd.ExecuteNonQuery();

Console.WriteLine("{0} records were inserted",
    affected);
```


Демонстрация

◆ Връзка с MS Access през OLE DB

The screenshot displays the Visual Studio .NET IDE with a C# project named 'OleDbConnectionDemo'. The code in 'Main()' uses 'OleDbCommand' to execute an 'INSERT INTO' statement on a 'Users' table. The table structure is as follows:

Field Name	Data Type
id	AutoNumber
username	Text
password	Text

The code snippet is as follows:

```
dbConn.Open();
using (dbConn)
{
    OleDbCommand cmd = new OleDbCommand(
        "INSERT INTO Users ([username], [password]) VALUES (@user, @pass)", dbConn);

    cmd.Parameters.Add("@user", OleDbType.VarChar).Value = "new user name";
    cmd.Parameters.Add("@pass", OleDbType.VarChar).Value = "secret password";

    // Execute command
    int rowsAffected = cmd.ExecuteNonQuery();

    if (rowsAffected > 0)
    {
        Console.WriteLine("{0} rows were affected.", rowsAffected);
    }
    else
    {
        Console.WriteLine("Operation failed.");
    }
}
```

The console output shows:

```
1 rows were affected. Operation was successfull
Inderted row id = 18.
Press any key to continue.
```

The status bar at the bottom indicates 'Build succeeded'.

Правилна работа с дати

◆ Препоръки:

- ❖ Използвайте вградените типове на базата данни, с която работите, а не стринг
- ❖ Някои бази от данни поддържат повече от един тип за дати
 - ❖ В MS SQL Server 2000 има `datetime` (8 байта) и `smalldatetime` (4 байта)
- ❖ При работа с данни от тип дата, използвайте стринг само за визуализация към потребителя

Правилна работа с дати

- ◆ Използвайте `System.DateTime` структурата за работа с дати в .NET
- ◆ Използвайте параметрични заявки за предаване на дати към базата данни
- ◆ При нужда от конвертиране предавайте `IFormatProvider` за дефиниране на правилата за конвертиране
- ◆ При необходимост използвайте неутрални културни настройки: `CultureInfo.InvariantCulture`

Работа с дати – пример

```
CREATE TABLE Messages
```

```
(
```

```
    MsgId int identity not null primary key,
```

```
    MsgText nvarchar(1000),
```

```
    MsgDate datetime -- Don't use varchar for dates!
```

```
)
```

```
public void AddMsg(string aText, DateTime aDate)
```

```
{
```

```
    SqlCommand cmdInsertMsg = new SqlCommand(
```

```
        "INSERT INTO Messages(MsgText, MsgDate) " +
```

```
        "VALUES (@MsgText, @MsgDate)", mDbCon);
```

(примерът продължава)

Работа с дати – пример

```
SqlParameter paramMsgText = new SqlParameter(  
    "@MsgText", SqlDbType.NVarChar);  
paramMsgText.Value = aText;  
cmdInsertMsg.Parameters.Add(paramMsgText);  
  
SqlParameter paramMsgDate = new SqlParameter(  
    "@MsgDate", SqlDbType.DateTime);  
paramMsgDate.Value = aDate;  
cmdInsertMsg.Parameters.Add(paramMsgDate);  
  
cmdInsertMsg.ExecuteNonQuery();  
}
```

Демонстрация

◆ Работа с дати в MS SQL Server

The screenshot shows the Visual Studio .NET IDE with the 'DatesDemo' project open. The 'DisplayAllMessages' method is being executed, and the console window displays the following output:

```
Inserted record in Messages table.
Inserted record in Messages table.
Inserted record in Messages table.
08-Юни-2005 22:15:02 - Test message 1
08-Юни-2005 22:15:02 - Test message 2
08-Юни-2005 22:15:02 - Test message 3
Press any key to continue.
```

The code in the 'DatesDemo.cs' file is as follows:

```
public void DisplayAllMessages(CultureInfo aCult
{
    SqlCommand cmdSelectMsgs = new SqlCommand(
        "SELECT MsgText, MsgDate FROM Messages",
        SqlDataReader reader = cmdSelectMsgs.Execute
        using (reader)
        {
            while (reader.Read())
            {
                string msgText = (string) reader["MsgText"];
                DateTime msgDate = (DateTime) reader["MsgDate"];
                string msgDateFormatted = msgDate.ToString(aFormat, aCultureInfo);
                Console.WriteLine("{0} - {1}", msgDateFormatted, msgText);
            }
        }
    }

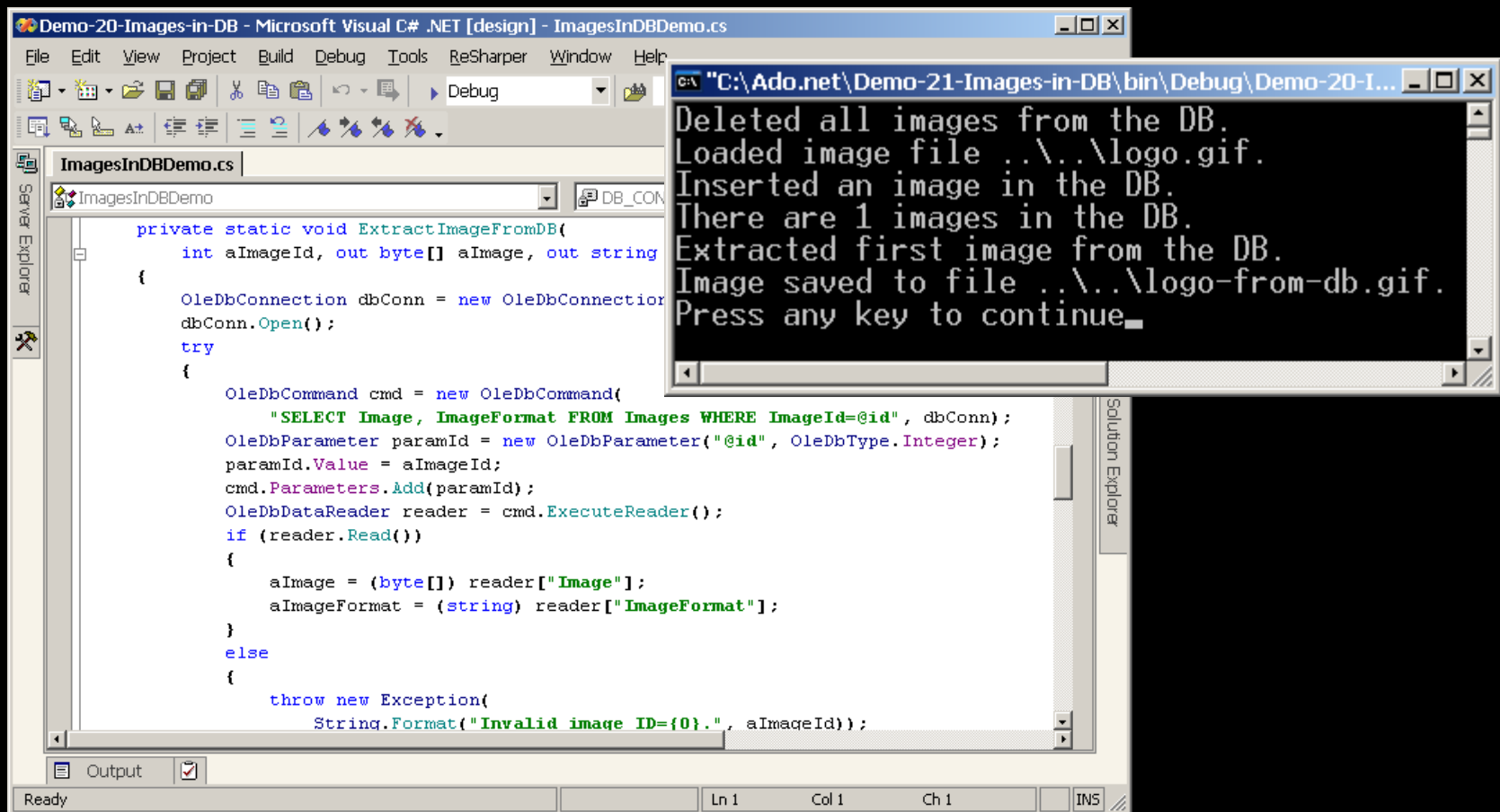
    public void DropMessagesTable()
    {
        SqlCommand cmdCreateMsgTable = new SqlCommand(
            @"IF OBJECT_ID('Messages') IS NOT NULL
            DROP TABLE Messages",
```

Работа с картинки в БД

- ◆ За записването на графични изображения в бази от данни се използват бинарни полета:
 - ❖ тип "image" в MS SQL Server
 - ❖ тип "blob" в Oracle
 - ❖ тип "OLE Object" в MS Access
- ◆ При обемни файлове трябва да се използва работа с потоци

Демонстрация

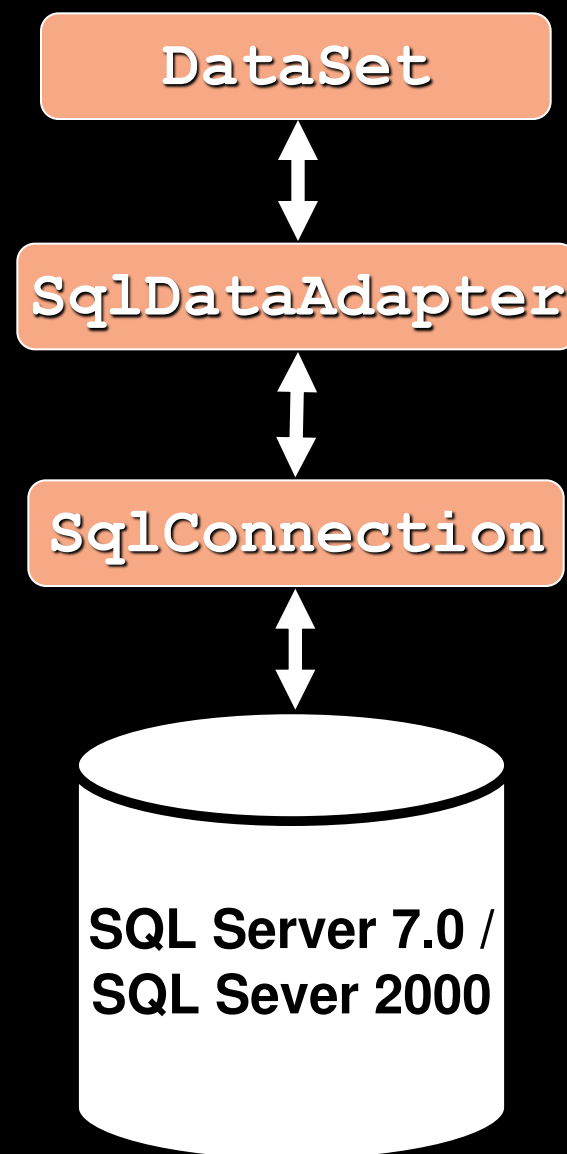
◆ Съхранение на графични изображения в база от данни



ADO.NET в несвързана среда

- ◆ Данните се кешират в DataSet обект и връзката се преустановява

1. Отваряне на връзка (SqlConnection)
2. Пълнене на DataSet (чрез SqlDataAdapter)
3. Затваряне на връзката
4. Работа със DataSet-a
5. Отваряне на връзка
6. Нанасяне на промени по данните по сървъра
7. Затваряне на връзката



Работа в несвързана среда

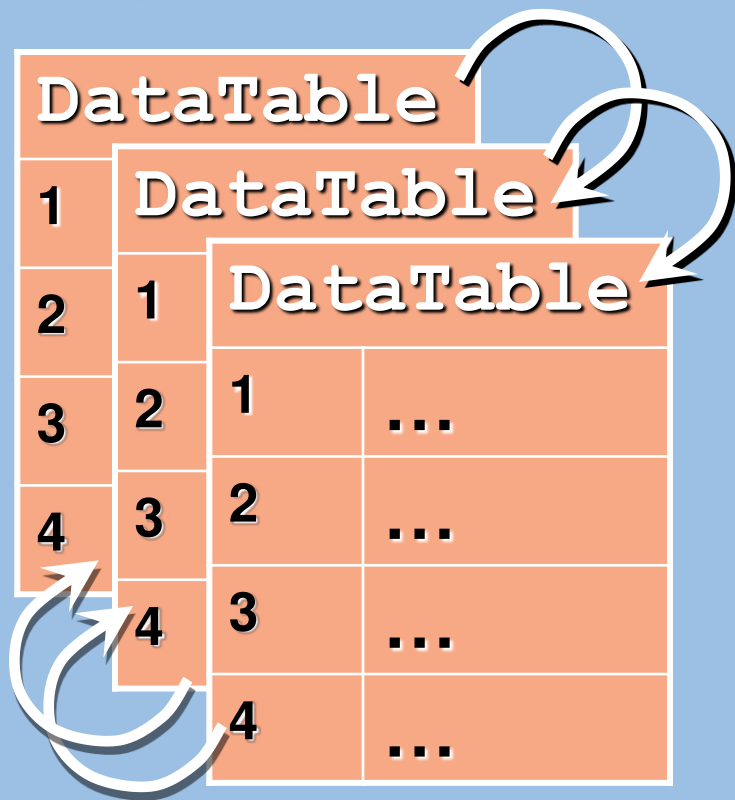
- ◆ Типични сценарии, в които се използва работата в несвързана среда
- ◆ Класове реализиращи достъпа до данните
 - ❖ `DataSet`
 - ❖ Контейнер клас – съдържа таблици, релации, `constraints` и други обекти
 - ❖ `DataTable` – съдържа таблица
 - ❖ `XxxDataAdapter` – осъществява достъпа до данните чрез `XxxCommand`, `XxxConnection` класове
 - ❖ `DataRelation` – връзка между таблици

DataSet – обектен модел

- ◆ **Колекции в DataSet**
 - ❖ **Tables** – съдържа таблиците
 - ❖ **Relations** – съдържа релациите между таблиците
- ◆ **Поддръжка на автоматично свързване (data binding)**
- ◆ **Схема на DataSet**
 - ❖ Описва структурата на DataSet-а
 - ❖ Дефинира се или програмно
 - ❖ или чрез XSD схема

Модел на данните DataSet

DataSet



```
<?xml version="1.0" standalone="yes"?>
<xs:schema id="DataSetUsers"
targetNamespace="http://www.tempuri.org/DataSetUsers.xsd"
xmlns:mstns="http://www.tempuri.org/DataSet1.xsd"
xmlns="http://www.tempuri.org/DataSet1.xsd"
xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:msdata="urn:schemas-microsoft-com:xml-msdata"
attributeFormDefault="qualified"
elementFormDefault="qualified">
  <xs:element name="DataSetUsers" msdata:IsDataSet="true"
msdata:Locale="bg-BG">
    <xs:complexType>
      <xs:choice maxOccurs="unbounded">
        <xs:element name="Users">
          <xs:complexType>
            <xs:sequence>
              <xs:element name="username"
type="xs:string" minOccurs="0" />
              <xs:element name="password"
type="xs:string" minOccurs="0" />
            </xs:sequence>
          </xs:complexType>
        </xs:element>
      </xs:choice>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

XML Schema

Силно-типизирани DataSets

- ◆ В .NET Framework се поддържат силно-типизирани DataSet-и
 - ❖ Силно-типизираните DataSet-и са класове, наследници на DataSet
 - ❖ Дефинират свойства за достъп до таблиците, техните редове и колони
- ◆ Достъп до нетипизиран DataSet:

```
string username = (string) dsUsers.  
    Tables["Users"].Rows[0]["username"];
```

- ◆ Достъп до силно-типизиран DataSet:

```
string username = dsUsers.Users[0].username;
```

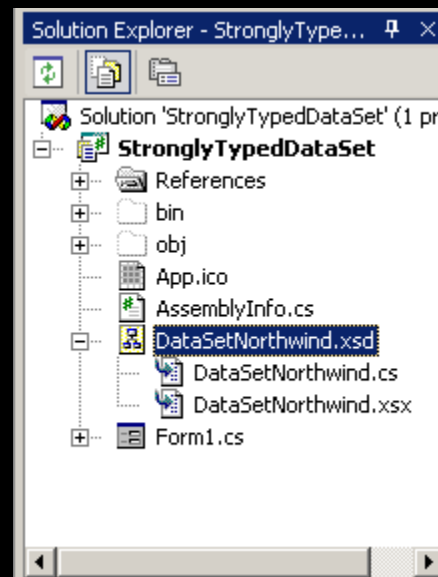
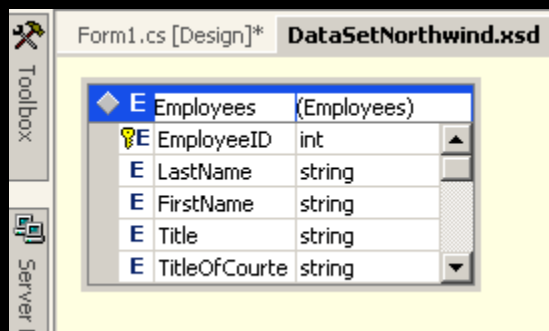
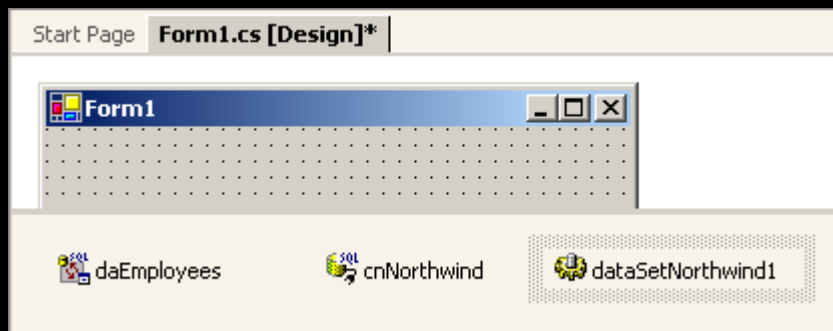
Създаване на DataSet

- ◆ **Създаване на силно-типизиран DataSet**
 - ❖ През XSD дизайнера на VS.NET 2003
 - ❖ File | Add New Item | Data Set
 - ❖ При записване на XSD файла, се генерира съответният C# клас
 - ❖ С инструмента `xsd.exe` по XSD схема:

```
xsd.exe MyDataSet.xsd /dataset/
```
 - ❖ Генерираният `MyDataSet.cs` съдържа класове и свойства за таблиците, редовете и колоните на DataSet-a

Демонстрация

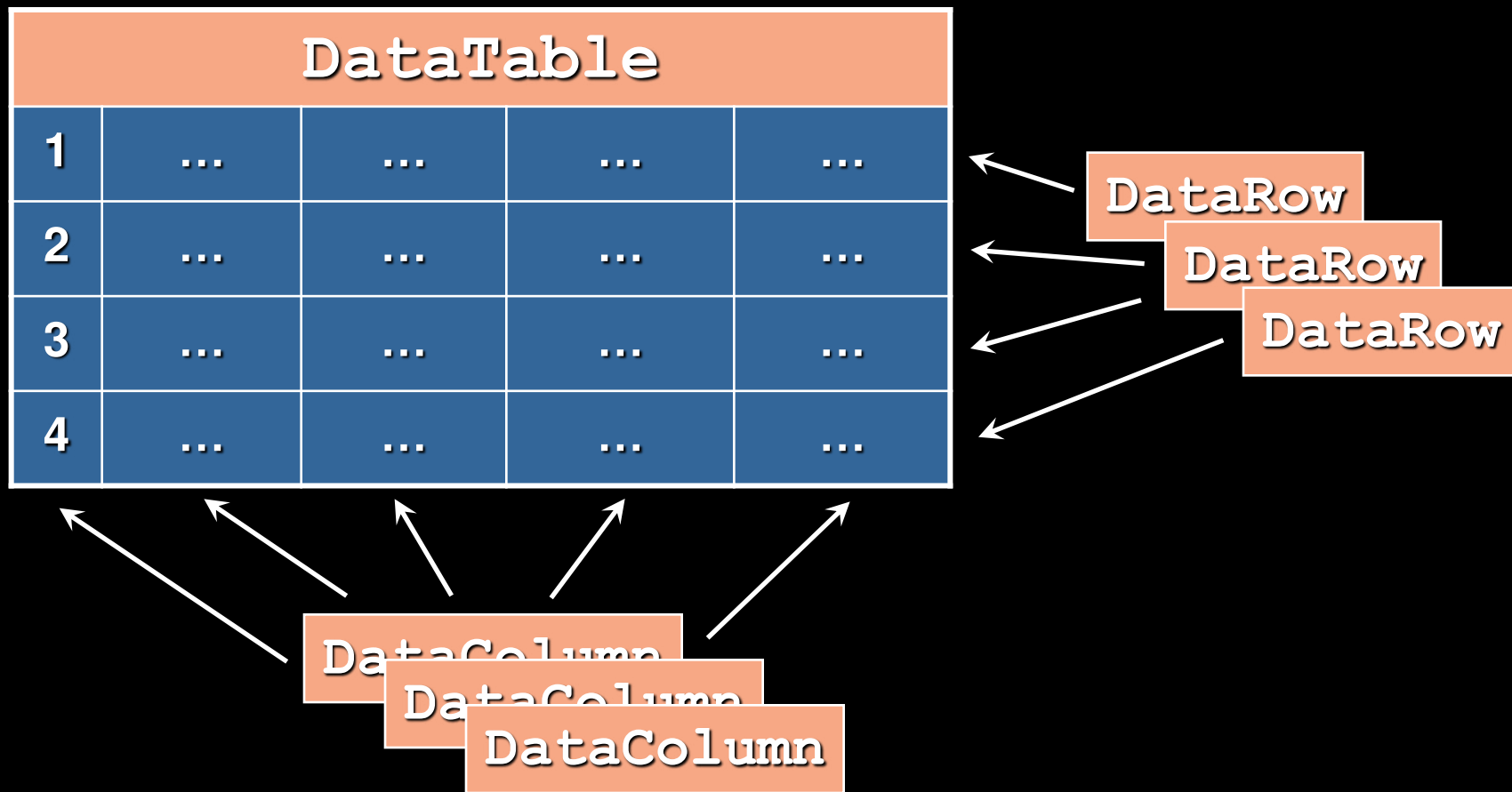
- ◆ Създаване на силно-типизиран DataSet чрез VS.NET



Класът DataTable

- ◆ Като таблиците в базите от данни
- ◆ Съставна част на DataSet класа
- ◆ Кеширано копие на данните в паметта
- ◆ Основни методи и свойства:
 - ❖ Rows [index] – достъп до редовете по номер
 - ❖ Columns [...] – достъп до колоните (DataColumn) по име или индекс
 - ❖ NewRow () – създава нов ред (DataRow), но не го добавя в таблицата
 - ❖ Rows .Add (DataRow) – добавя ред в таблицата
 - ❖ Columns .Add (DataColumn) – добавя колона
 - ❖ PrimaryKey – задава/извлича първичния ключ

Модел на данните DataTable



Класове за работа с DataTable

◆ Събития на класа DataTable

- ❖ RowChanging – преди промяна на ред
- ❖ RowChanged – след промяна на ред
- ❖ RowDeleting – преди изтриване на ред
- ❖ RowDeleted – след изтриване на ред

◆ Класът DataRow представя ред от таблица

- ❖ `this[]` – дава достъп до стойността в дадена колона (по име или индекс)
- ❖ `Delete` – маркира реда като изтрит

◆ Класът DataColumn представя колона

- ❖ `Name` – име на колоната
- ❖ `DataType` – тип на колоната (`int`, `string`, ...)

Работа с DataTable – пример

```
DataTable tbl = new DataTable("Authors");

tbl.Columns.Add("au_id", typeof(int));
tbl.Columns.Add("au_fname", typeof(string));
tbl.Columns.Add("au_lname", typeof(string));
tbl.Columns.Add("au_phone", typeof(string));

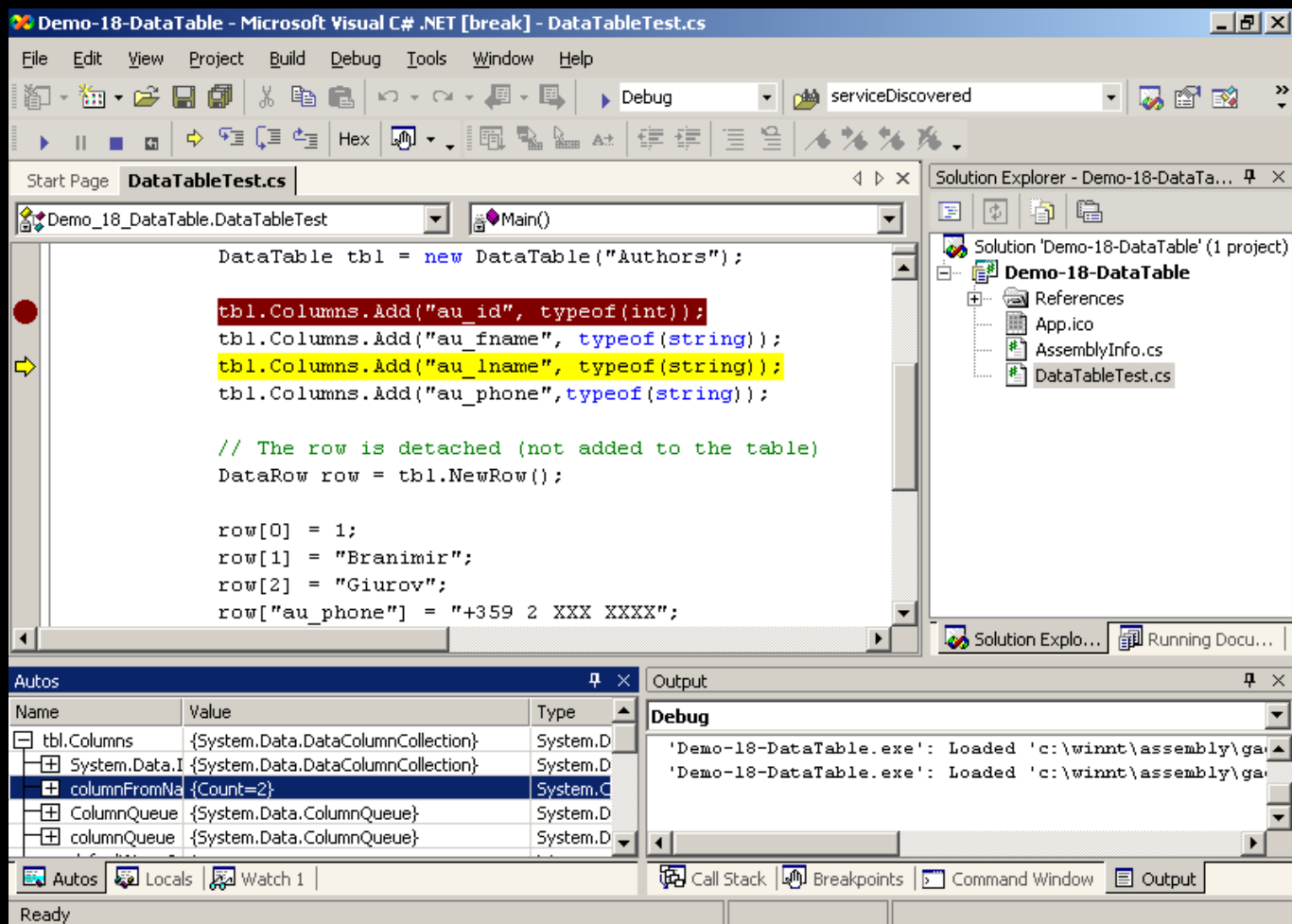
// The row is detached (not added to the table)
DataRow row = tbl.NewRow();

row[0] = 1;
row[1] = "Branimir";
row[2] = "Giurov";
row["phone"] = "+359 2 XXX XXXX";

tbl.Rows.Add(row);
```

Демонстрация

◆ Работа с класа DataTable



The screenshot displays the Microsoft Visual Studio .NET IDE. The main window shows the code for `DataTableTest.cs` in the `Main()` method. The code creates a new `DataTable` object, adds columns, and adds a new row.

```
DataTable tbl = new DataTable("authors");  
tbl.Columns.Add("au_id", typeof(int));  
tbl.Columns.Add("au_fname", typeof(string));  
tbl.Columns.Add("au_lname", typeof(string));  
tbl.Columns.Add("au_phone", typeof(string));  
  
// The row is detached (not added to the table)  
DataRow row = tbl.NewRow();  
  
row[0] = 1;  
row[1] = "Branimir";  
row[2] = "Giurov";  
row["au_phone"] = "+359 2 XXX XXXX";
```

The Solution Explorer on the right shows the project structure for `Demo-18-DataTable`, including `References`, `App.ico`, `AssemblyInfo.cs`, and `DataTableTest.cs`.

The Autos window at the bottom left shows the state of the `tbl` object, which is a `System.Data.DataTable`. The `tbl.Columns` property is expanded, showing a `System.Data.DataColumnCollection` with a count of 2.

The Output window at the bottom right shows the debug output, indicating that the application has loaded successfully.

Използване на ограничения (constraints)

◆ Дефиниране на първичен ключ (Primary key)

```
// Single column PK
dtCustomers.PrimaryKey = new DataColumn()
{
    dtCustomers.Columns("CustomerID")
}

// Multiple columns PK
dtEmployees.PrimaryKey = new DataColumn()
{
    dtEmployees.Columns("LastName"),
    dtEmployees.Columns("FirstName")
}
```

Използване на ограничения (constraints)

◆ Дефиниране на UniqueConstraint и ForeignKeyConstraint

```
ds.Tables["Product"].Constraints.Add(  
    new UniqueConstraint("UC_ProductName",  
        ds.Tables["Product"].Columns["ProductName"]));  
  
ForeignKeyConstraint custOrderFK =  
    new ForeignKeyConstraint("CustOrderFK",  
        ds.Tables["CustTable"].Columns["CustomerID"],  
        ds.Tables["OrdersTable"].Columns["CustomerID"]);  
  
custOrderFK.DeleteRule = Rule.None;  
// Cannot delete a customer value  
// that has associated existing orders.  
  
ds.Tables["OrdersTable"].Constraints.Add(custOrderFK);
```

Потребителски изрази

- ◆ Изразите са формули, дефинирани от програмиста
- ◆ Дефинират се с `Expression` свойството на `DataColumn` класа
 - ❖ `Sum([Unit Price] * [Quantity])`
- ◆ Агрегатните функции могат да използват родител/наследник релациите
 - ❖ `Avg, Count, Sum, Max, Min`

Потребителски изрази

- ◆ Използват за създаване на:
 - ❖ калкулирана колона
 - ❖ агрегирана колона
 - ❖ конкатенация на една или повече колони
 - ❖ обръщение към таблица родител или наследник
- ◆ Пример:

```
// Calculated field - VAT  
DataColumn priceVat = new DataColumn(  
    "Price (VAT)", typeof(decimal));  
priceVat.Expression = "Price * 1.2";  
productsTable.Columns.Add(priceVat);
```


DataRelation обекти

- ◆ Дефинират релации между родител / наследник колони в таблици с навигационна цел
- ◆ Не създават ограничения (constraints)
- ◆ Могат да се дефинират и към същата таблица
- ◆ Често се използват за навигация през потребителския интерфейс
- ◆ Използват се от колоните с потребителски изрази за дефиниране на агрегатни изрази
- ◆ Съдържат се в Relations колекцията на DataSet обекта

DataRelation обекти

- ◆ **GetChildRows ()** – връща редовете наследници на даден ред

```
dsNorthwind.Relations.Add("FK_CustomersOrders",  
    dtCustomers.Columns["CustomerID"],  
    dtOrders.Columns["CustomerID"], true);  
  
// Process all customers and their orders  
foreach (DataRow drCustomer in  
    dsNorthwind.Tables["Customers"].Rows)  
{  
    foreach (DataRow drOrder in drCustomer.  
        GetChildRows("FK_CustomerOrders"))  
    {  
        // Do some work with the rows  
    }  
}
```

DataView клас

- ◆ Аналогия на изгледите (View обектите) в базите от данни
 - ❖ Динамичен изглед на данните базиран на израз за филтриране
 - ❖ Филтриране по версията на данните
 - ❖ Сортиране по колона или колони

```
DataTable usersTable = ...;  
  
DataView usersView = new DataView(usersTable);  
  
// Show only modified versions of current rows and new rows  
usersView.RowStateFilter =  
    DataViewRowState.ModifiedCurrent |  
    DataViewRowState.Added;
```

Запазване и зареждане на данните от DataSet

- ◆ Два основни метода
 - ❖ `ReadXml (...)` – зарежда DataSet от XML
 - ❖ `WriteXml (...)` – записва DataSet в XML
 - ❖ Пътят до файла се подава като параметър
- ◆ `ReadXml (...)` хвърля изключение ако файлът не съществува или потребителят няма права
- ◆ `WriteXml (...)` припокрива съществуващи файлове

DataSet.ReadXml()

- ◆ Приема източник на XML данните:
 - ❖ Stream, TextReader – отворен за четене ПОТОК
 - ❖ String – път до файл или URL
 - ❖ XmlReader – XML четец
- ◆ Може да се задава режим на четене (XmlReadMode)
 - ❖ Auto, DiffGram, Fragment, IgnoreSchema, InferSchema, ReadSchema
 - ❖ Текущата схема на данните може да се запазва, да се прочете наново или да се генерира от самите данни

DataSet.WriteXml()

- ◆ **Възможност за записване на данните и схемата на DataSet в XML формат**
- ◆ **Синхронизация със XmlDocument класа**
- ◆ **Може да се задава режим на записване (XmlWriteMode)**
 - ❖ **DiffGram** – записва оригиналните данни и нанесените в тях промени
 - ❖ **IgnoreSchema** – не записва схемата
 - ❖ **WriteSchema** – записва и схемата

ReadXml и WriteXml – пример

```
DataSet dsStudents = new DataSet();  
// A string that contains the XML data  
string xmlStudentData = "<students><student>" +  
    "<name>Petar Petrov</name><fn>12345</fn>" +  
    "</student><student>" +  
    "<name>Ivan Ivanov</name><fn>54321</fn>" +  
    "</student></students>";  
  
// A StringReader to pass the XML to ReadXml(...)  
StringReader srXmlStudents = new  
    StringReader(xmlStudentData);  
dsStudents.ReadXml(srXmlStudents,  
    XmlReadMode.InferSchema);  
srXmlStudents.Close();
```

(примерът продължава)

ReadXml и WriteXml – пример

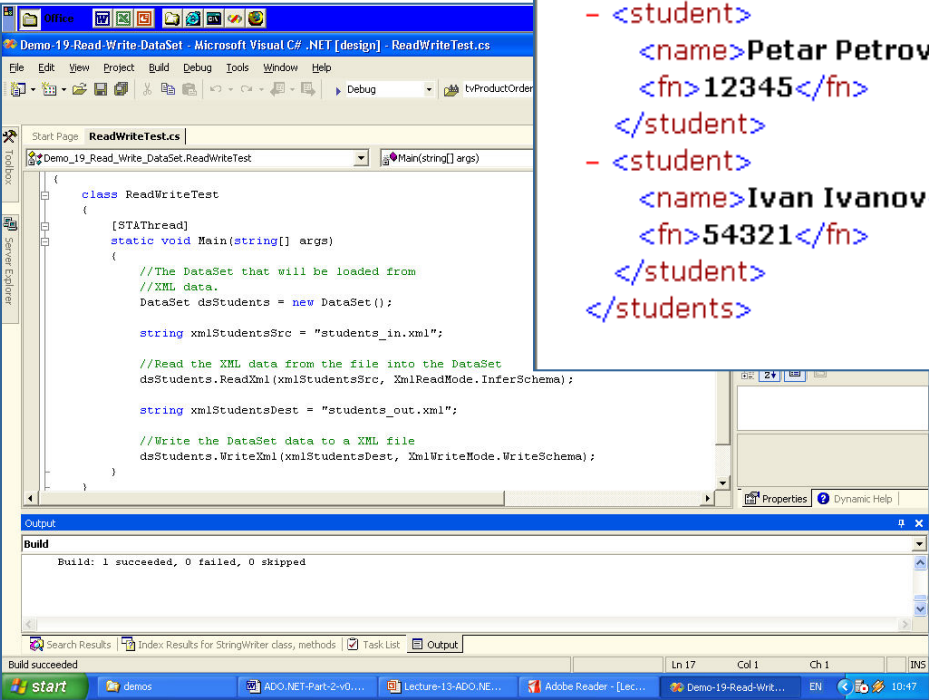
```
// A StringWriter to store the data from the DataSet
StringWriter swStudents = new StringWriter();
dsStudents.WriteXml(swStudents,
    XmlWriteMode.WriteSchema);
string strStudents = swStudents.ToString();
swStudents.Close();

// Print to the console the XML before reading it
// in the DataSet, and the XML produced by the
// DataSet's WriteXml(...) method
Console.WriteLine("XML before:\n" +
    xmlStudentData + "\n\nXML after:\n " +
    strStudents);
```


Демонстрация

◆ Чете и писане на DataSet от XML файл

```
<?xml version="1.0" standalone="yes" ?>
- <students>
- <xs:schema id="students" xmlns="" xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns:msdata="urn:schemas-microsoft-com:xml-msdata">
- <xs:element name="students" msdata:IsDataSet="true" msdata:Locale="bg-BG">
- <xs:complexType>
- <xs:choice maxOccurs="unbounded">
- <xs:element name="student">
- <xs:complexType>
- <xs:sequence>
  <xs:element name="name" type="xs:string"/>
  <xs:element name="fn" type="xs:string"/>
</xs:sequence>
</xs:complexType>
</xs:element>
</xs:choice>
</xs:complexType>
</xs:element>
</xs:schema>
- <student>
  <name>Petar Petrov</name>
  <fn>12345</fn>
</student>
- <student>
  <name>Ivan Ivanov</name>
  <fn>54321</fn>
</student>
</students>
```



The screenshot shows the Visual Studio IDE with the file `ReadWriteTest.cs` open. The code in the file is as follows:

```
class ReadWriteTest
{
    [STAThread]
    static void Main(string[] args)
    {
        //The DataSet that will be loaded from
        //XML data.
        DataSet dsStudents = new DataSet();

        string xmlStudentsSrc = "students_in.xml";

        //Read the XML data from the file into the DataSet
        dsStudents.ReadXml(xmlStudentsSrc, XmlReadMode.InferSchema);

        string xmlStudentsDest = "students_out.xml";

        //Write the DataSet data to a XML file
        dsStudents.WriteXml(xmlStudentsDest, XmlWriteMode.WriteSchema);
    }
}
```

The Output window at the bottom shows the build results:

```
Build
Build: 1 succeeded, 0 failed, 0 skipped
```

```
<?xml version="1.0" standalone="yes" ?>
- <students>
- <student>
  <name>Petar Petrov</name>
  <fn>12345</fn>
</student>
- <student>
  <name>Ivan Ivanov</name>
  <fn>54321</fn>
</student>
</students>
```

Използване на DataAdapter

- ◆ DataAdapter свързва DataSet с източника на данни
- ◆ Използва се за:
 - ❖ Запълване на DataSet с данни
 - ❖ Обновяване на данните в източника след извършване на промени в тяхното копие в DataSet обекта
- ◆ Метод Fill () на DataAdapter
 - ❖ Запълва таблица в DataSet
 - ❖ Запълва DataTable обект
 - ❖ Използва SelectCommand

Използване на DataAdapter

- ◆ Свойство MissingSchemaAction
- ◆ Запълване на повече от една таблица в DataSet
- ◆ Задаване на съответствие между таблици и колони в източника на данни и DataSet
 - ❖ Свойство TableMappings на DataAdapter
- ◆ Метод FillSchema() на DataAdapter – извличане на схемата от източника

Използване на DataAdapter

◆ Свойства на DataAdapter

- ❖ AcceptChangesDuringFill
- ❖ ContinueUpdateOnError

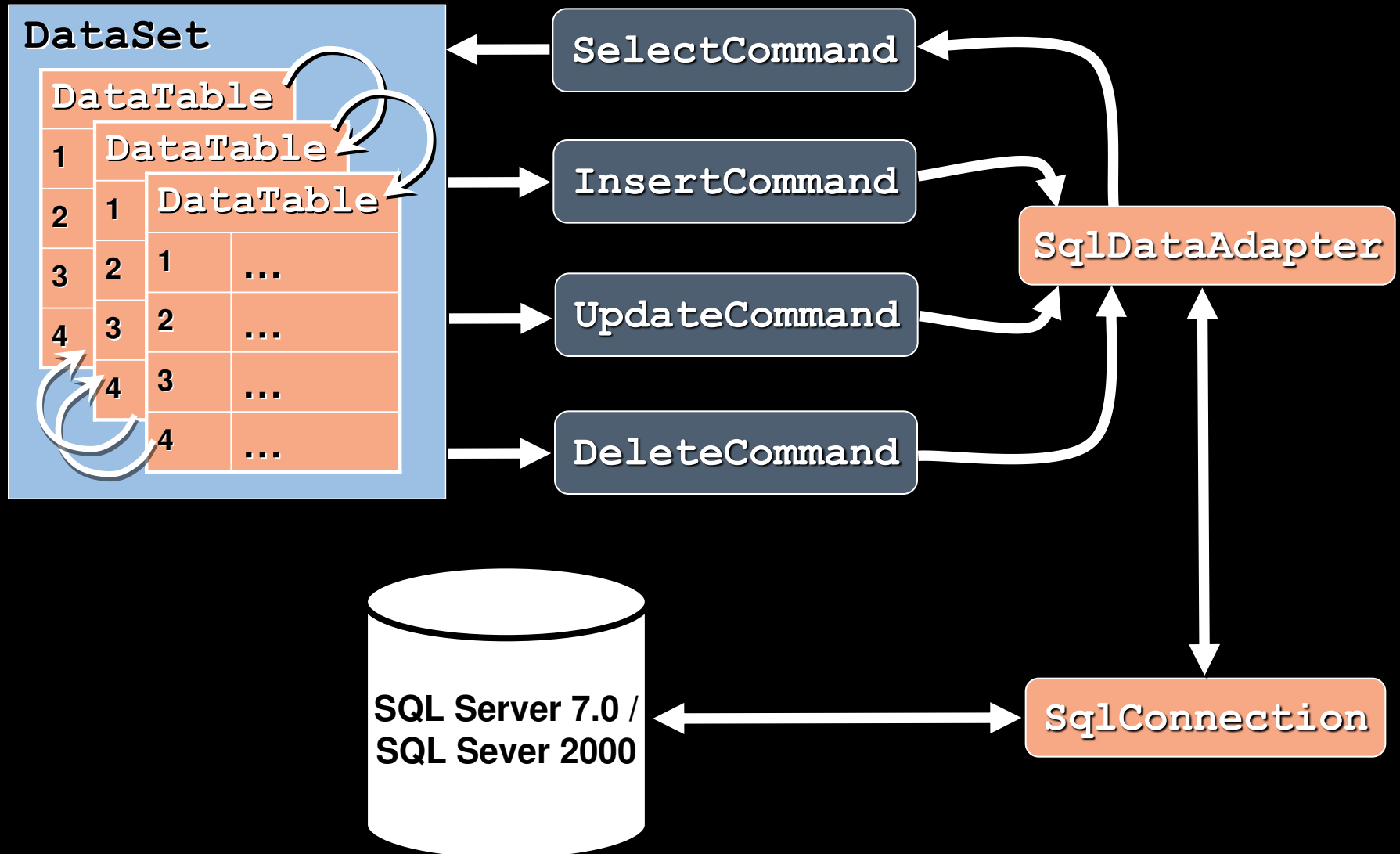
◆ Събития на DataAdapter

- ❖ FillError
- ❖ RowUpdating
- ❖ RowUpdated

Използване на DataAdapter

- ◆ Обновяване на данните в източника
 - ❖ Свойства InsertCommand, UpdateCommand, DeleteCommand
- ◆ Класът SqlCommandBuilder
- ◆ Обновяване на източника с потребителска логика
- ◆ Извличане на обновени стойности в DataSet
- ◆ Обновяване на свързани таблици

DataAdapter – архитектура



DataAdapter – пример

```
static void Main()
{
    string strCon = "Data Source=(local);" +
        "Integrated Security=SSPI;Database=Northwind";
    SqlConnection cnNorthwind = new SqlConnection(strCon);
    SqlCommand cmdSelect = CreateSelectCommand(cnNorthwind);
    SqlDataAdapter daEmployees = new
        SqlDataAdapter(cmdSelect);
    DataSet dsNorthwind = new DataSet();
    daEmployees.Fill(dsNorthwind, "Employees");
    // Set the AutoIncrement property of EmployeeID column
    DataTable employeesTable =
        dsNorthwind.Tables["Employees"];
    DataColumn columnEmployeeId =
        employeesTable.Columns["EmployeeID"];
    columnEmployeeId.AutoIncrement = true;
    columnEmployeeId.AutoIncrementSeed = -1;
    columnEmployeeId.AutoIncrementStep = -1;
}
```

(примерът продължава)

DataAdapter – пример

```
// Create the commands for the data adapter
daEmployees.InsertCommand = CreateInsertCommand(cnNorthwind);
daEmployees.DeleteCommand = CreateDeleteCommand(cnNorthwind);
daEmployees.UpdateCommand = CreateUpdateCommand(cnNorthwind);

// Add new record and update the database
DataRow row = employeesTable.NewRow();
row["LastName"] = "Ivanov";
row["FirstName"] = "Ivan";
employeesTable.Rows.Add(row);
daEmployees.Update(dsNorthwind, "Employees");
Console.WriteLine("Inserted row id={0}.", row["EmployeeID"]);

// Change the added record and update the database
row["LastName"] = "Petrov";
daEmployees.Update(dsNorthwind, "Employees");
Console.WriteLine("Updated the row.");
```

(примерът продължава)

DataAdapter – пример

```
// Delete the added record and update the database
row.Delete();
daEmployees.Update(dsNorthwind, "Employees");
Console.WriteLine("Deleted the row.");
}

static SqlCommand CreateSelectCommand(SqlConnection aConnection)
{
    string strSelect = "SELECT EmployeeID, LastName, FirstName" +
        " FROM Employees";

    SqlCommand cmdSelect = new SqlCommand(strSelect, aConnection);
    return cmdSelect;
}

static SqlCommand CreateInsertCommand(SqlConnection aConnection)
{
    string strInsert = "INSERT Employees(LastName, FirstName) " +
        "VALUES(@LastName, @FirstName);" +
        "SET @EmployeeID=Scope_Identity()";
}
```

(примерът продължава)

DataAdapter – пример

```
SqlCommand cmdInsert = new SqlCommand(strInsert, aConnection);
SqlParameterCollection cparams = cmdInsert.Parameters;
SqlParameter empID = cparams.Add("@EmployeeID",
    SqlDbType.Int, 0, "EmployeeID");
empID.Direction = ParameterDirection.Output;
cparams.Add("@LastName", SqlDbType.NVarChar, 20, "LastName");
cparams.Add("@FirstName", SqlDbType.NVarChar, 10, "FirstName");
return cmdInsert;
}

static SqlCommand CreateUpdateCommand(SqlConnection aConnection)
{
    string strUpdate = "UPDATE Employees SET " +
        "LastName=@LastName, FirstName=@FirstName " +
        "WHERE EmployeeID=@EmployeeID";
    SqlCommand cmdUpdate = new SqlCommand(strUpdate, aConnection);
    SqlParameterCollection cparams = cmdUpdate.Parameters;
    SqlCommand cmdUpdate = new SqlCommand(strUpdate, aConnection);
    SqlParameterCollection cparams = cmdUpdate.Parameters;
```

(примерът продължава)

DataAdapter – пример

```
        SqlParameter empID = cparams.Add("@EmployeeID",
            SqlDbType.Int, 0, "EmployeeID");
        empID.SourceVersion = DataRowVersion.Original;
        cparams.Add("@LastName", SqlDbType.NVarChar, 20, "LastName");
        cparams.Add("@FirstName", SqlDbType.NVarChar, 10,
            "FirstName");
        return cmdUpdate;
    }

    static SqlCommand CreateDeleteCommand(SqlConnection
        aConnection)
    {
        string strDelete = "DELETE FROM Employees " +
            "WHERE EmployeeID = @EmployeeID";

        SqlCommand cmdDelete = new SqlCommand(strDelete,
            aConnection);

        SqlParameter empID = cmdDelete.Parameters.Add(
            "@EmployeeID", SqlDbType.Int, 0, "EmployeeID");
        empID.SourceVersion = DataRowVersion.Original;
        return cmdDelete;
    }
```

DataSet.GetChanges () и DataSet.HasChanges ()

- ◆ Могат да приемат параметър за версията на данните от тип DataRowState
 - ❖ Added – добавени редове
 - ❖ Deleted – изтрити редове
 - ❖ Detached – разкачени от таблицата редове
 - ❖ Modified – променени редове
 - ❖ Unchanged – непроменени редове
- ◆ GetChanges () връща копие на DataSet-а, съдържащо промените след последното обръщение на AcceptChanges () метода

DataSet.GetChanges() – пример

```
if (! myDataSet.HasChanges (DataRowState.Modified) )  
    return;  
  
// Create temporary DataSet variable  
  
DataSet modifiedDataSet;  
  
// GetChanges for modified rows only  
modifiedDataSet = myDataSet.GetChanges (  
    DataRowState.Modified) ;  
  
// Check the DataSet for errors  
if(modifiedDataSet.HasErrors) {  
    // Insert code to resolve errors  
}  
  
// After fixing errors, update the data source  
// with the DataAdapter used to create the DataSet  
adp.Update(modifiedDataSet);
```

Несвързан модел – типичен сценарий на работа

1. Зареждаме данните в DataSet чрез `DataAdapter.Fill()` или по друг начин:

```
userDataAdapter.Fill(dsUsers);
```

2. Обработваме данните – променяме, добавяме и изтриваме записи
3. Извличаме направените промени:

```
DataSet dsChanges = dsUsers.GetChanges();
```

`GetChanges()` връща `null` ако няма промени в DataSet-a

Несвързан модел – типичен сценарий на работа

4. Прилагаме направените промени и разрешаваме конфликтите

```
userDataAdapter.RowUpdated +=  
    new SqlRowUpdatedEventHandler(OnRowUpdated) ;  
private void OnRowUpdated(object sender,  
    SqlRowUpdatedEventArgs e)  
{  
    // Handle the conflict ...  
    e.Status = UpdateStatus.Continue;  
}  
userDataAdapter.Update(dsChanges) ;
```

5. Зареждаме отново DataSet-а от базата данни, за да работим с актуални данни

```
userDataAdapter.Fill(dsUsers) ;
```


Демонстрация #20

◆ Реализация на несвързан модел с DataSet и DataAdapter

The screenshot displays the Microsoft Visual Studio .NET IDE. The main window shows the file `DataAdapterTest.cs` with the following C# code:

```
string strSelect = "SELECT Id, Name FROM Alabala";

//Create an adapter and generate its commands
//with a CommandBuilder
SqlDataAdapter daAlabala = new SqlDataAdapter(strSelect, cnNorthwind);
SqlCommandBuilder cbAlabala = new SqlCommandBuilder(daAlabala);
//Add an event handler for the RowUpdated event
daAlabala.RowUpdated += new SqlRowUpdatedEventHandler(OnRowUpdated);

//Create and fill a DataSet
DataSet dsNorthwind = new DataSet();
daAlabala.Fill(dsNorthwind, "Alabala");

PrintAlabala(dsNorthwind);

//Add some data to the DataSet
MakeChanges(dsNorthwind);

PrintAlabala(dsNorthwind);

daAlabala.Update(dsNorthwind, "Alabala");
```

The console window, titled `"C:\temp\ConsoleApplication1\bin\Debug\ConsoleApplication1.exe"`, shows the output of the application:

```
Table after adding some records:
ID Name
1 Ivan
2 Pescho
3 Goscho
1 Sasho
4 Toshio

Trying to update the table...
There were conflicts!

Row "1 Sasho" conflicts with some row in the database:
Violation of PRIMARY KEY constraint 'PK_Alabala_151B244E'.
cate key in object 'Alabala'.
The statement has been terminated.

Trying to update the table again...
No errors during update.
The table contents from the database:
ID Name
1 Ivan
2 Pescho
3 Goscho
4 Toshio
```


Класът XmlDocument

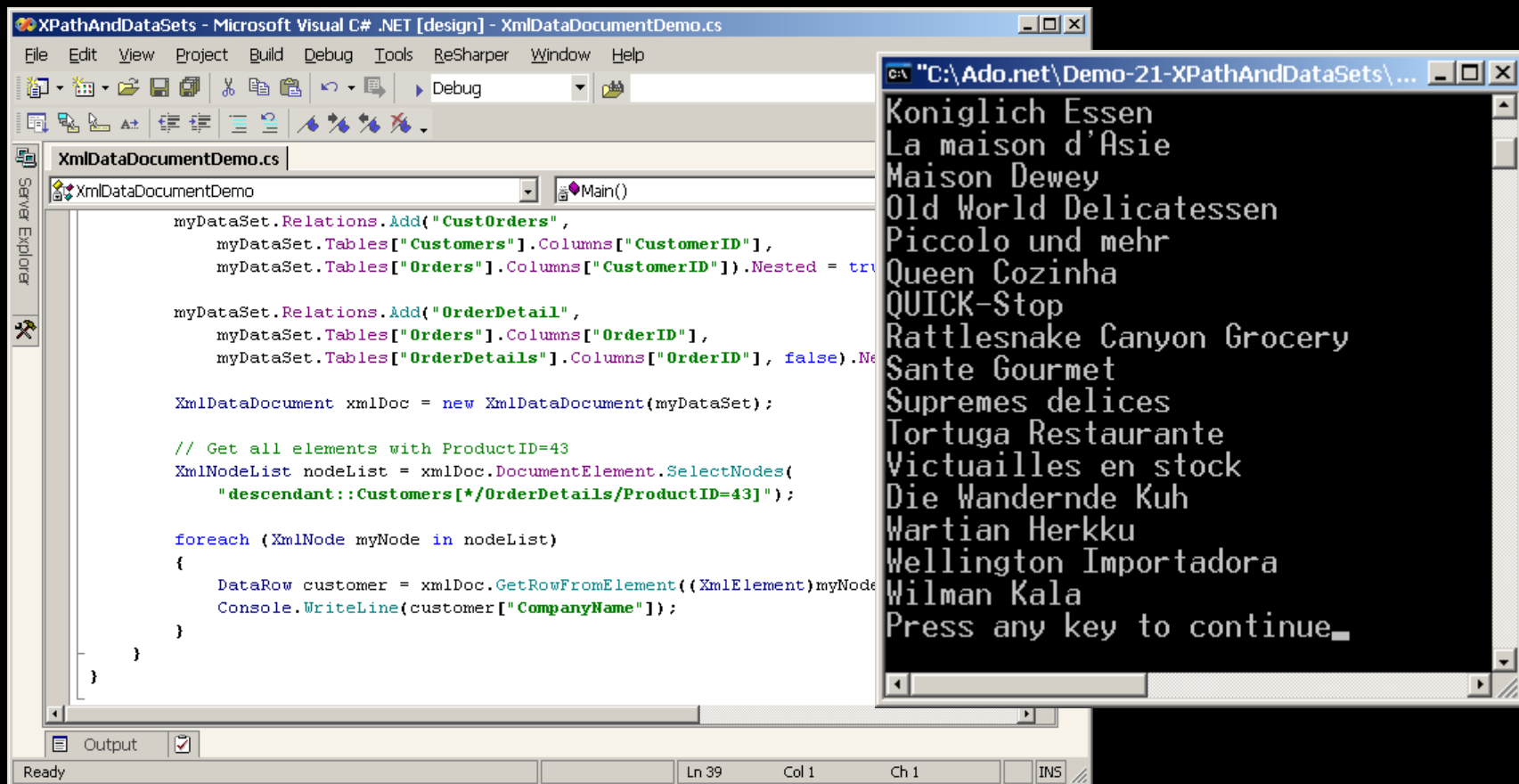
- ◆ Интеграция и поддръжка на XML през XmlDocument класа
- ◆ Класът XmlDocument
 - ❖ Представява DOM дърво
 - ❖ Наследник на XmlDocument
 - ❖ Реализира автоматична синхронизация на DataSet с DOM дърво
 - ❖ При промяна в DOM дървото се променя DataSet-а
 - ❖ При промяна на DataSet-а се променя DOM дървото

XmlDataDocument – пример

```
DataSet myDataSet = new DataSet();  
  
// ... Fill the DataSet ...  
  
XmlDataDocument xmlDoc =  
    new XmlDataDocument(myDataSet);  
  
// Get all elements with ProductID = 43  
XmlNodeList nodeList = xmlDoc.DocumentElement.  
SelectNodes("descendant::Customers[*]/OrderDetails/  
ProductID=43");  
  
foreach (XmlNode myNode in nodeList)  
{  
    DataRow customer = xmlDoc.GetRowFromElement(  
        (XmlElement)myNode);  
    Console.WriteLine(customer["CompanyName"]);  
}
```

Демонстрация #21

◆ Използване на XPath за търсене в DataSet чрез XmlDocument



Сигурността при работа с бази от данни

- ◆ Да не се сглобяват SQL команди с "+"
- ❖ Използване на параметрични команди
- ◆ Connection pooling и сигурност
- ◆ Съхраняване на connection string
- ◆ Криптиране на данните

Въведение

... в Java EE технологиите



Петьо Димитров

01 Април 2013

Съдържание на лекциите

Въведение в Java EE технологиите

Презентация на данните

Бизнес логика

Съхранение на данните

Съдържание

Същност на Java EE

Многослойно приложение

Контейнери

Основни спецификации

Демонстрация: FleaMarket

Защо ни трябва?

- Когато програмист иска да работи с набор от телефонни номера:
 - не създава своя имплементация на свързан списък
 - използва API-то за колекции, част от Java Standard Edition
- Когато програмист се нуждае от сигурно, разпределено приложение, поддържащо транзакции, което може да работи върху няколко платформи:
 - не започва да работи по собствено решение за тези стандартни проблеми
 - използва Java EE функционалностите
 - фокусира се върху бизнес проблемите, а не стандартните операции

Същност на Java Enterprise Edition

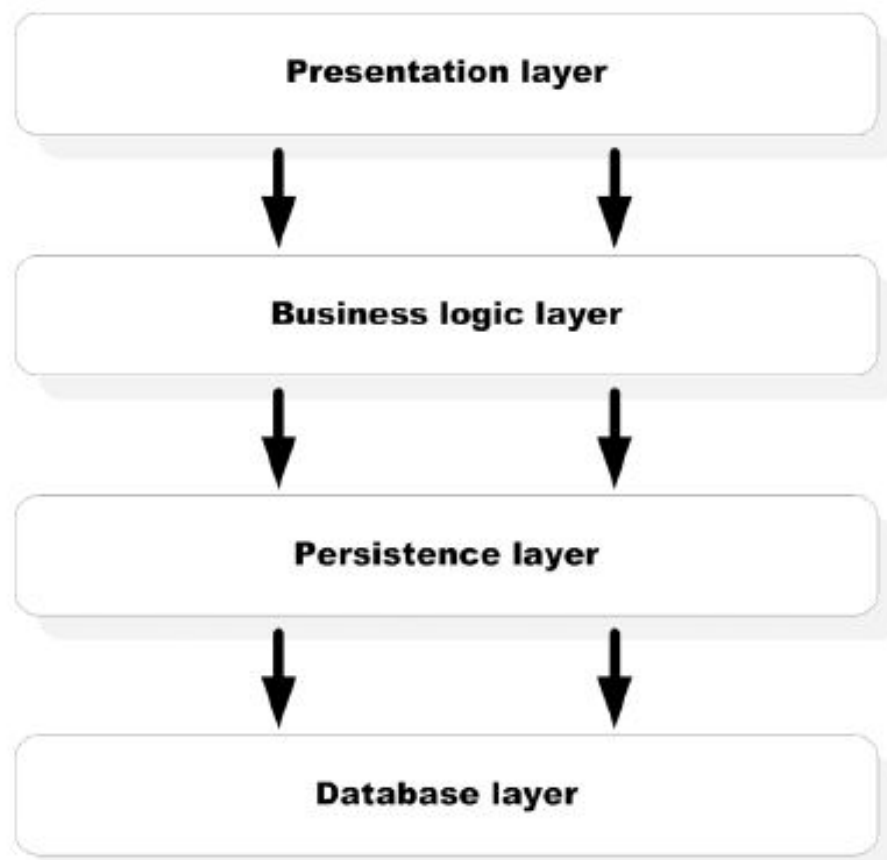
- набор от API-та и runtime среда за разработка и изпълнение на реални бизнес приложения (enterprise applications)
- разширение на Java SE (Standard Edition) предлагащо възможности за ORM мапинг, отдалечени извиквания, уеб услуги и много други
- дизайн базиран на сътрудничащи си отделни *компоненти* за създаване на многослойна архитектура
- залага на "*convention over configuration*"

Многослойно приложение

- стандартен pattern за създаване на сървърни приложения
- базира се на групи от компоненти с обща цел наречени слоеве (tiers) - като части от асемблиния
- всеки слой изпълнява функциите си и делегира останалата работа на следващия слой

Традиционна 4-слойна архитектура

1. *презентационния слой* е отговорен за графичния интерфейс на клиента
2. *бизнес логиката* описва процесите и правилата за приложението
3. *persistence слоя* предлага ОО интерфейс към базата данни
4. *база данни* – RDBMS като Oracle, DB2, и т.н.



Java EE компоненти

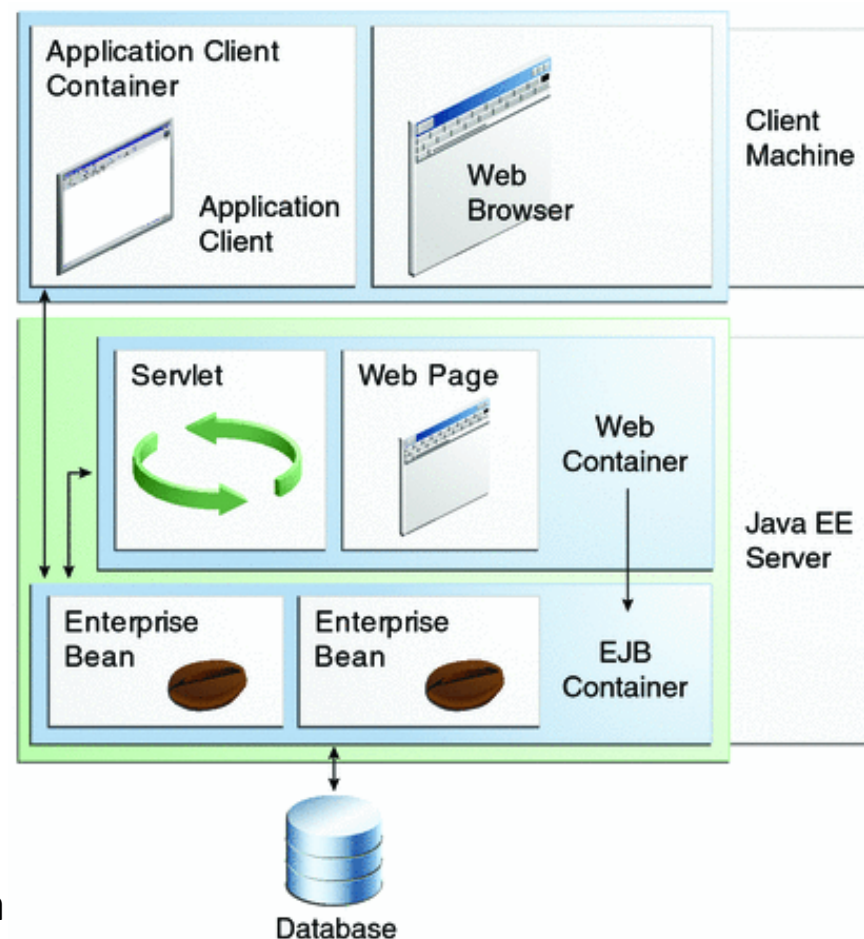
- presentation layer:
 - servlet, jsp, facelet, listener, filter, custom tag
- business logic layer:
 - ejb (session, message-driven), interceptor, timer
- persistence layer:
 - entity, listener
- database layer – извън обхвата на Java

Java EE контейнер

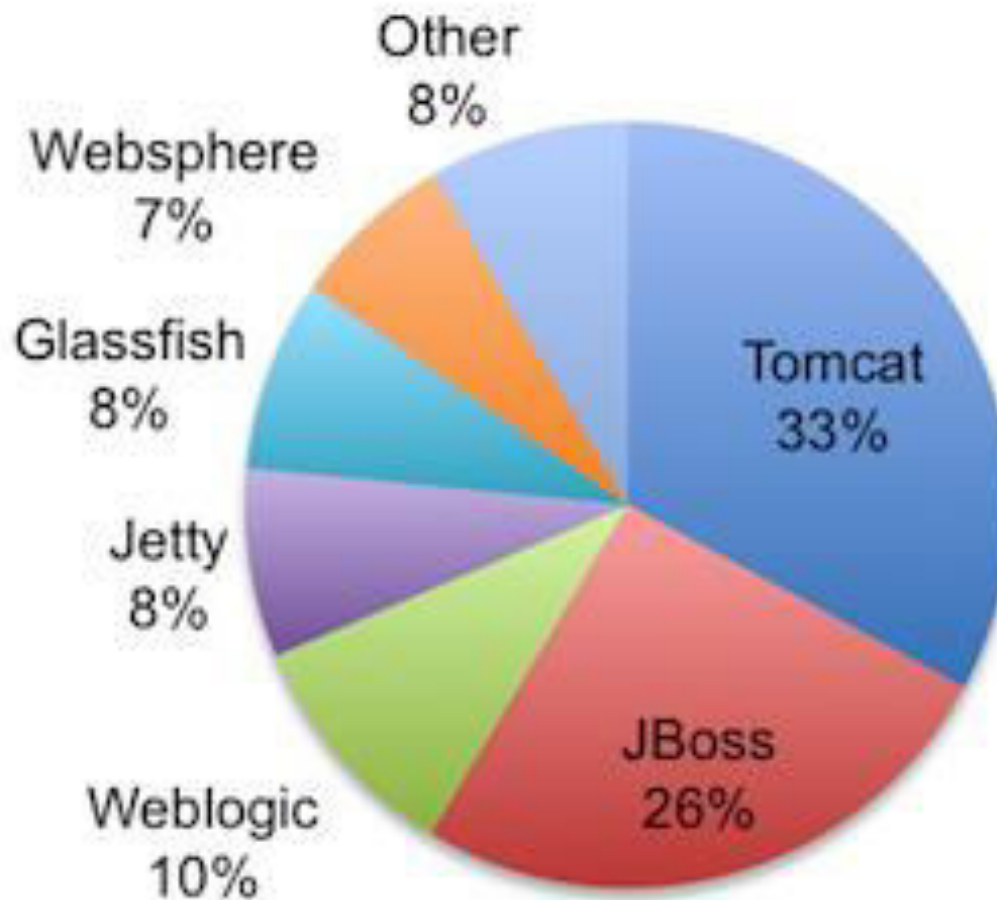
- интерфейс между компонент и API-то от по-ниско ниво
- контейнера обвива всеки нов компонент и предлага набор от услуги:
 - управление на жизнения цикъл
 - сигурност
 - транзакции
 - IoC (inversion of control)
 - и други
- грижи се за компонент, както JVM-а се грижи за Java програма

Видове Java EE контейнери

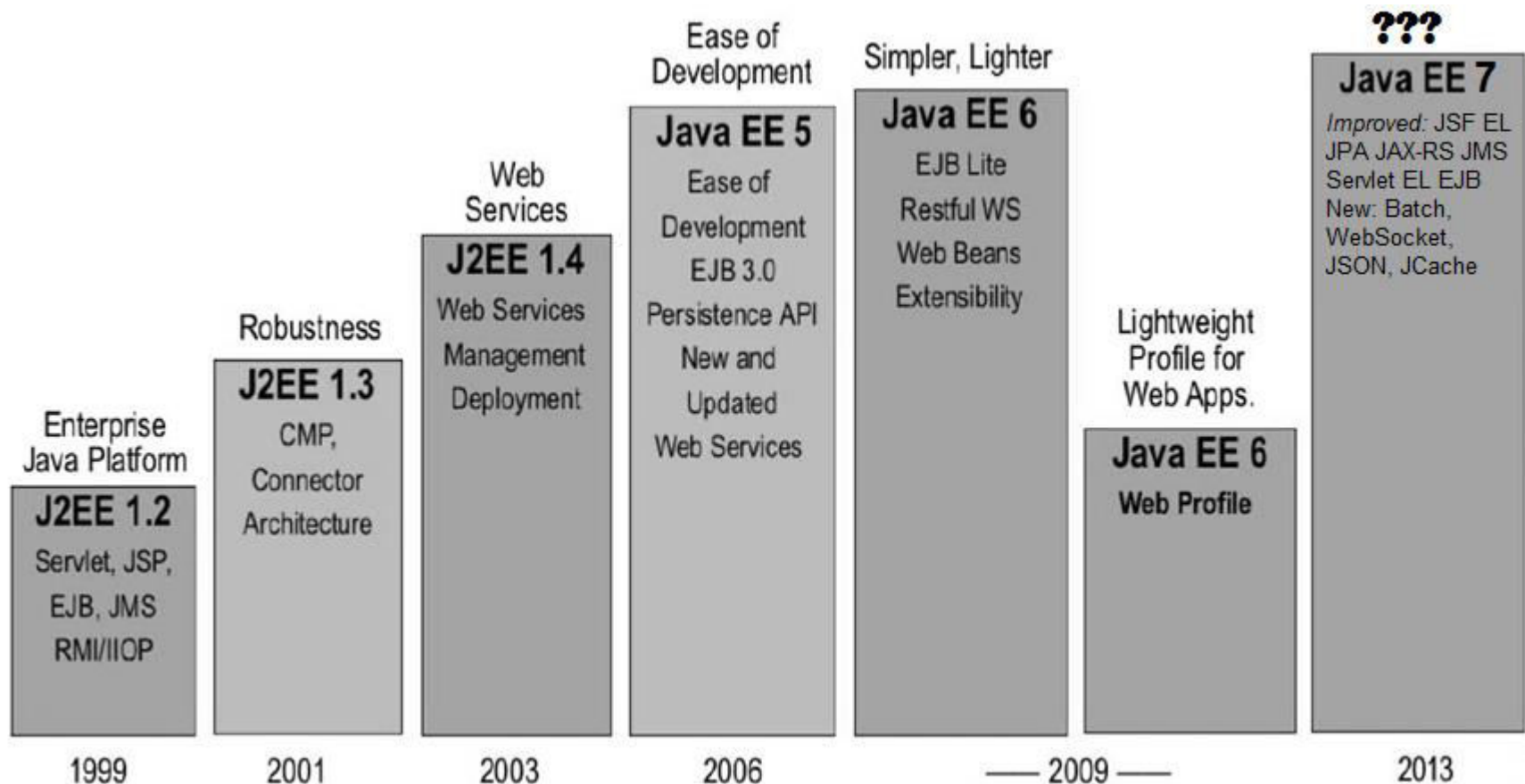
- **Java EE сървър** - среда за изпълнение на Java EE приложения; предлага EJB и Web контейнери
- **Enterprise JavaBeans (EJB) контейнер** - управлява изпълнението на enterprise beans
- **Web контейнер** - управлява изпълнението на уеб страници, servlet-и, jsp-та и т.н.
- **Application client контейнер** - управлява изпълнението на Java EE клиентски приложения
- **Applet контейнер** - управлява изпълнението на applet-и; представлява Java Plug-in инсталиращ се в уеб браузъра



Популярни Java EE сървъри



История на Java EE



Java EE спецификации

- много и различни спецификации, резултат от:
 - дългогодишната история и еволюция на Java EE
 - разнообразните функционалности нужни на едно приложение
- ще разгледаме само най-съществените



Java EE спецификации (продължение)

Web Application Technologies

Java Servlet 3.0

JSR 315

JavaServer Faces 2.0

JSR 314

JavaServer Pages 2.2/Expression Language 2.2

JSR 245

Standard Tag Library for JavaServer Pages (JSTL) 1.2

JSR 52



Java EE спецификации (продължение)

Enterprise Application Technologies

Bean Validation 1.0 JSR 303

Enterprise JavaBeans 3.1 JSR 318

Java EE Connector Architecture 1.6 JSR 322

Java Persistence 2.0 JSR 317

Common Annotations for the Java Platform 1.1 JSR 250

Java Message Service API 1.1 JSR 914

Java Transaction API (JTA) 1.1 JSR 907

JavaMail 1.4 JSR 919



BALIN



DWALIN



BOFUR



FILI



KILI

Java EE спецификации (продължение)

Web Services Technologies

Java API for RESTful Web Services (JAX-RS) 1.1	JSR 311
Java API for XML-Based Web Services (JAX-WS) 2.2	JSR 224
Java Architecture for XML Binding (JAXB) 2.2	JSR 222
Web Services Metadata for the Java Platform	JSR 181
Java APIs for XML Messaging 1.3	JSR 67
Java API for XML Registries (JAXR) 1.0	JSR 93
Java API for XML Processing (JAXP) 1.3	JSR 206
Streaming API for XML (StAX) 1.0	JSR 173



DORI



NORI

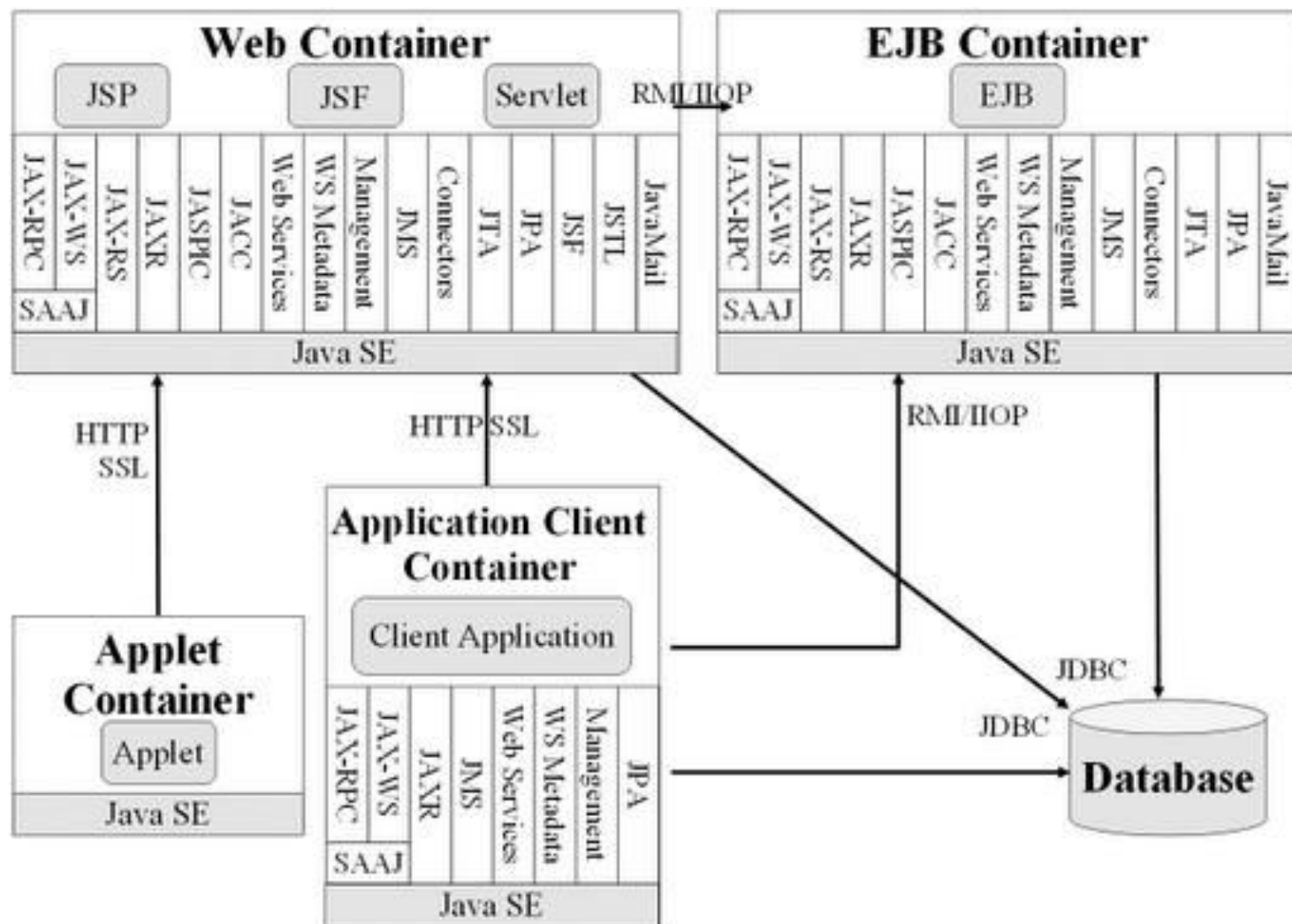


BIFUR



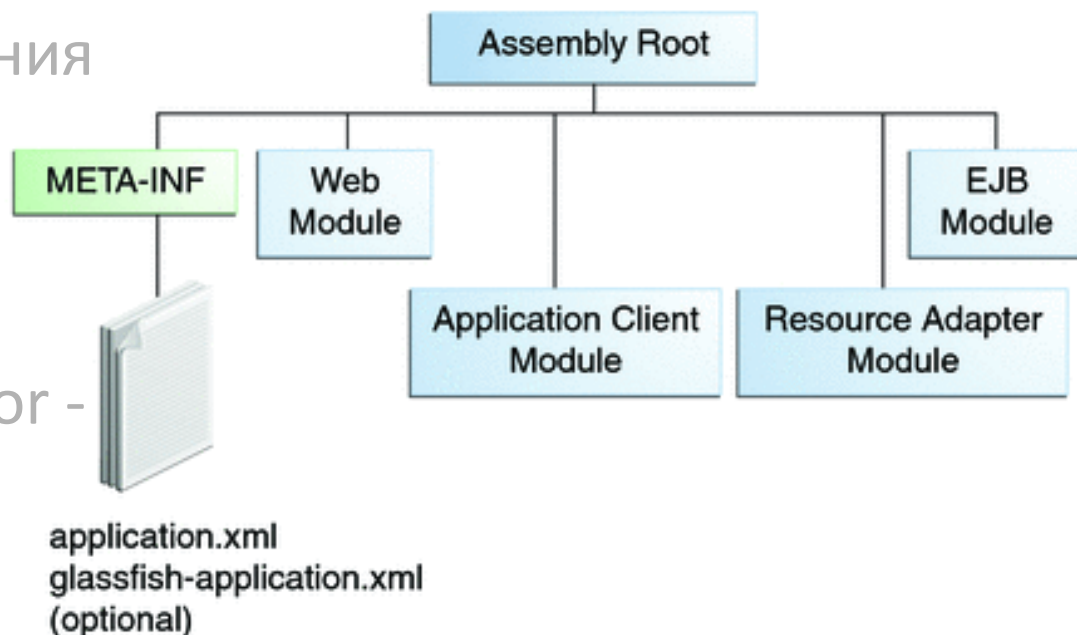
BOMBUR

Java EE спецификации и контейнеры



Пакетиране

- enterprise приложенията се пакетират в EAR (или WAR) и може да съдържат:
 - презентационни компоненти (пакетирани в WAR)
 - бизнес логиката под формата на EJB модули (в JAR-ове)
 - клиентски приложения (пакетирани в JAR)
 - ресурс адаптери* (в RAR файлове)
 - deployment descriptor - xml документ с конфигурация



Роли на разработчиците

- теоретично разделение позволяващо на различни разработчици (или подизпълнители) да отговарят за различни части от процеса:
 - Enterprise Bean Developer
 - Web Component Developer
 - Application Client Developer
 - Application Assembler
 - Application Deployer and Administrator

Алтернативни технологии

- Spring
- Struts
- Hibernate (non-JPA API)

FleaMarket

- примерно приложение онаглеждаващо представяните концепции
- достатъчно сложно за да притежава характеристики на реално приложение

Flea Market

where all the cool fleas go

[home](#) [items](#) [my bids](#) [my won bids](#) [logout \(bidder\)](#)

Bids:

March 18, 2013	tiem 2 - 50.0 BGN / 30.0 BGN Status: WINNER
March 18, 2013	tiem 2 - 40.0 BGN / 30.0 BGN Status: NEW
March 18, 2013	item 1 - 22.0 BGN / 20.0 BGN Status: NEW
March 18, 2013	item 1 - 21.0 BGN / 20.0 BGN Status: NEW

Technical University - Sofia
Faculty for Computer Systems and Technology

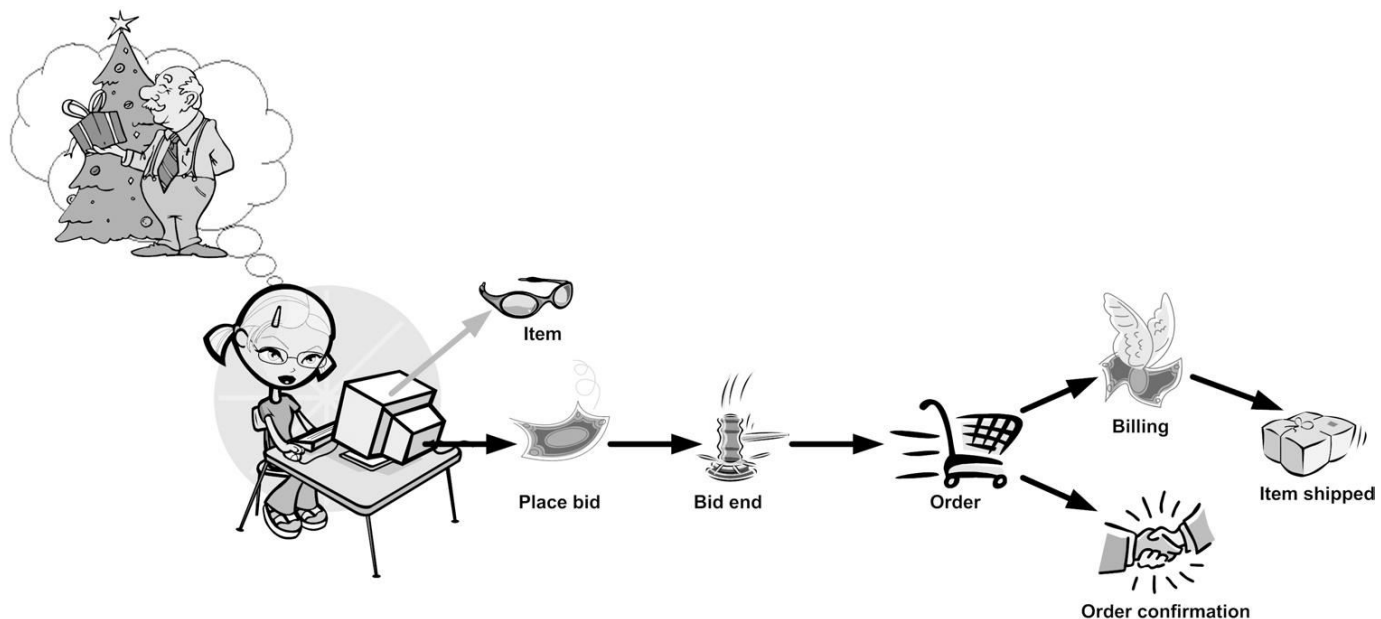


Flea Market is a demonstration of the capabilities of Java EE for creating a multi-tiered enterprise application, containing presentation, business logic and persistence layers.

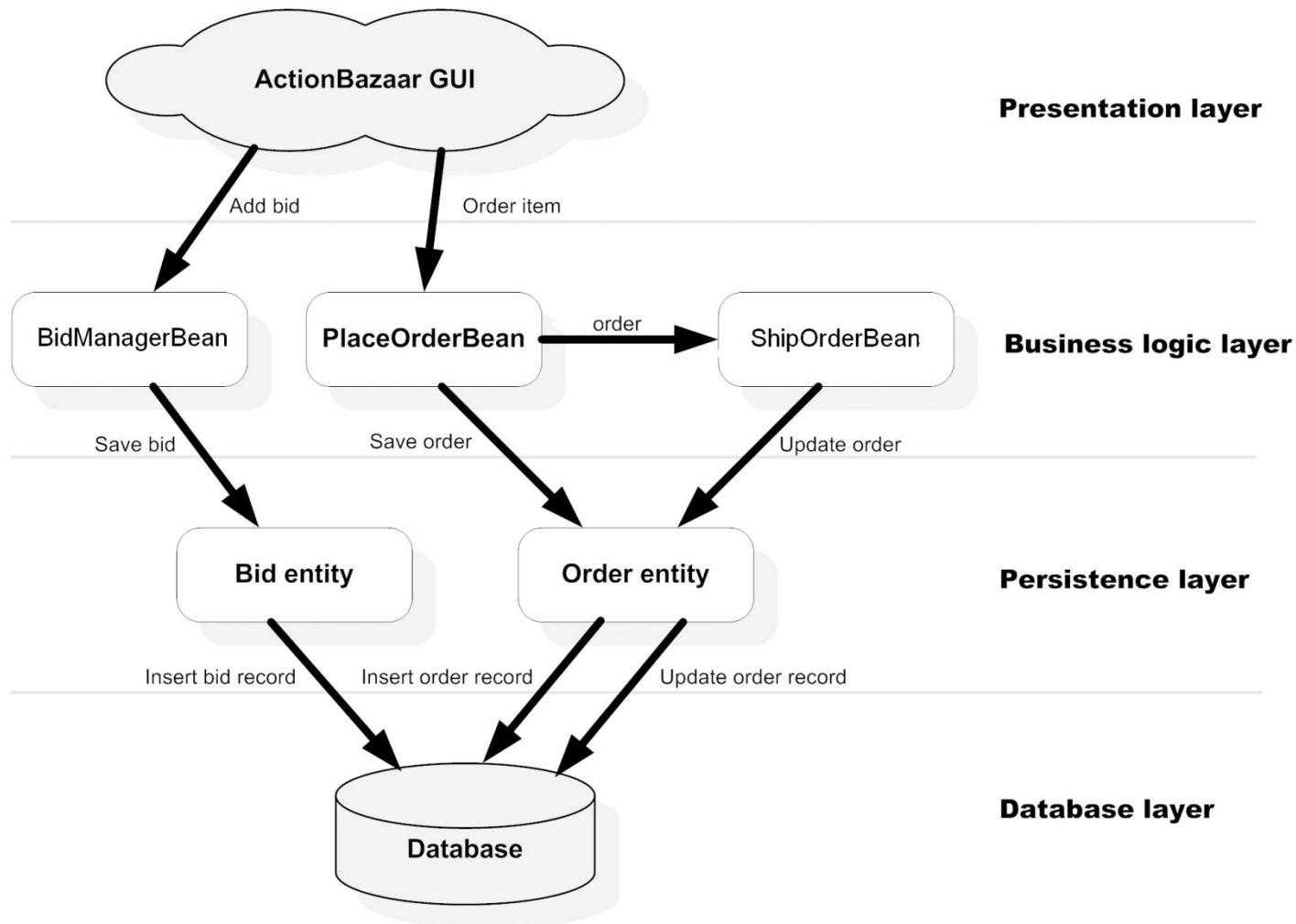
2013 - created by Petyo Dimitrov

FleaMarket – основна идея

- приложение за онлайн търгове
- продавачите създават артикули за търгуване
- залагащите наддават до приключване на търга
- победителя поръчва артикула и той бива изпратен



FleaMarket – основен дизайн



FleaMarket – технически изисквания

- Git – за достъп до кода
- Eclipse* – за разглеждане/модифициране на кода
- GlassFish* – за деплой на приложението, плюс конфигурация на jdbc-domain
- Derby* – за съхранение на данните (част от GlassFish)
- Mail server – за изпращане на имейли

Демонстрация

THE FOLLOWING **PREVIEW** HAS BEEN APPROVED FOR
ALL AUDIENCES

BY THE MOTION PICTURE ASSOCIATION OF AMERICA, INC.

THE FILM ADVERTISED HAS BEEN RATED



www.filmratings.com

www.mpaa.org

Out of scope

- детайли за разглежданите технологии
- Java EE Web Service спецификациите (друг курс)
- ОО концепции и базови Java API-та (анотации, generics, колекции, многонишковост и т.н.)
- SQL концепции и синтаксис
- основни принципи на markup (XML, HTML...)



Ресурси

- The Java EE 6 Tutorial, 2013
- EJB 3 in Action, 2007
- Head First: Servlets and JSPs, 2008
- Pro JPA 2 Mastering the Java Persistence API, 2009
- JavaServer Faces 2.0, The Complete Reference, 2009
- Head First: Design Patterns, 2004

Въпроси



Благодаря за вниманието



Презентация на данните

... чрез Servlet, JSP, JSTL и EL технологиите



Петьо Димитров

01 Април 2013

Съдържание на лекциите

Въведение в Java EE технологиите

Презентация на данните

Бизнес логика

Съхранение на данните

Съдържание

- Сървлет - същност, жизнен цикъл и особености
- JSP - същност, жизнен цикъл, JSP конструкции
- Тагове - потребителски, JSTL, таг файлове
- Expression Language - същност и употреба
- MVC pattern (JSF)



Какво е сървлет?

- сървлетът "сервира" информация като отговор от получена заявка
- програмно API позволяващо прихващане на заявка обслужвана от сървъра и генериране на отговор
- клас имплементиращ Servlet интерфейса
- може да се използва за всякакъв вид заявки (GenericServlet), но най-често се използва заедно с HTTP протокола (HttpServlet)



Приложение на сървлети

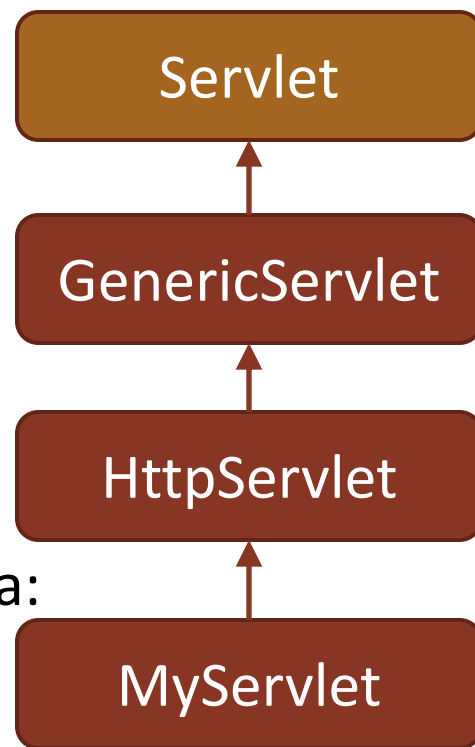
- динамично генериране на съдържание
- обработка на HTTP заявки:
 - GET – за представяне на данни
 - POST – за получаване на данни от HTML форми
- обработка на множество паралелни заявки (по-добро scalability спрямо CGI)
- платформено независимо сървърно API
- позволява препращане на заявки към други сървлети (дава възможност за load balancing)
- основа на JSP и JSF технологиите

Пример за сървлет

```
@WebServlet("/dynamic")
public class MyServlet extends HttpServlet {
    protected void doGet(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException {
        response.setContentType("text/html");
        Integer count = (Integer) request.getSession().getAttribute("count");
        if (count == null) {
            count = 0;
        }
        request.getSession().setAttribute("count", ++count);
        response.getWriter().println(formatDynamicContent((Integer) request.getSession().getAttribute("count")));
    }
    private String formatDynamicContent(int count) {
        StringBuilder builder = new StringBuilder();
        builder.append("<html>");
        builder.append("<head><title>").append("Dynamic Servlet").append("</title></head>");
        builder.append("<body>");
        builder.append("<h2>Timestamp: " + new Date() + "<br>").append("Times: " + count + "</h2>");
        builder.append("</body></html>");
        return builder.toString();
    }
}
```

Servlet интерфейс и базови класове

- Servlet интерфейсът дефинира базовите **методи**:
 - `init(ServletConfig)`
 - `service(ServletRequest, ServletResponse)`
 - `destroy()`
- GenericServlet класът имплементира базовата сървърна функционалност:
 - `getInitParameter(String)`
 - `getServletContext()`
- HttpServlet имплементира HTTP протокола:
 - `doGet()`, `doPost()`, `doPut()`, `doDelete()`...
- MyServlet - потребителските сървлети най-често наследяват HttpServlet



Жизнен цикъл на сървлет

- **load** - контейнерът зарежда сървлет класа при стартиране на уеб модула или при първа заявка
- **instantiate** - контейнерът създава инстанция от сървлет класа използвайки конструктора му (например `new MyServlet()`)
- **init** - преди сървлетът да може да обслужва заявки, контейнерът го инициализира (чрез метод `init()`) и подава параметрите от `web.xml`-а



Жизнен цикъл на сървлет

- **service** - след успешна инициализация, сървлетът може да обслужва заявки; контейнерът създава *отделна нишка* за всяка заявка и вика `service()` метода на сървлета
- **destroy** - когато сървлетът вече не е нужен, контейнерът вика метод `destroy()`; подобно на `init()` това се случва веднъж в жизнения цикъл
- **unload** – сървлет класът се освобождава от JVM-а на контейнера

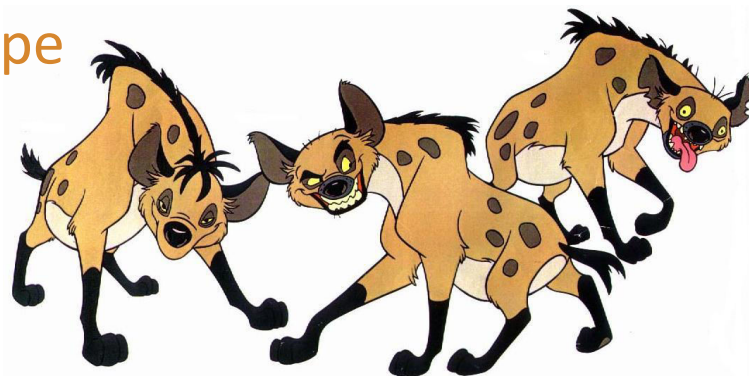


Споделяне на информация

- уеб компонентите споделят информация под формата на обекти складирани като атрибути на `scope` обекта
- те се манипулират чрез методи `setAttribute()` и `getAttribute()` на `scope`-а
- `scope`-овете са (с намаляваща видимост):
 - *application* (`javax.servlet.ServletContext`) - данните ще са видими от всички уеб компоненти в приложението
 - *session* (`javax.servlet.http.HttpSession`) - данните ще са видими за уеб компоненти в текущата сесия (може да се използва за прехвърляне на данни между заявки)
 - *request* (`javax.servlet.HttpServletRequest`) - данните ще са видими за уеб компоненти от текущата заявка
 - *page* (`javax.servlet.jsp.JspContext`) - само в дадената JSP страница

Concurrency при сървлети

- multithreading-а е заложен в сървлет спецификацията
 - създава се само една инстанция от даден сървлет клас
 - всяка заявка се обработва в отделна нишка
 - всички нишки работят върху един и същ обект
- паралелен достъп може да възникне в следните случаи:
 - достъп до атрибути в application scope
 - достъп до атрибути в session scope
 - достъп до член променливи на сървлета
- предпазване от грешки?



Request & Response

- `ServletRequest`-а дава достъп до:
 - HTTP хедърите на заявката
 - HTML form данни и параметри на заявката
 - други клиентски данни (cookie-та, път и т.н.)
- `ServletResponse`-а отговаря за комуникацията от сървлета обратно към клиента:
 - задаване на `content length` и MIME тип
 - събиране на отговора (чрез `ServletOutputStream` или `Writer`)



Извикване на други уеб компоненти

- `RequestDispatcher` – обект използван за директно и индиректно извикване на други компоненти; може да се вземе от заявката (`ServiceRequest`) или от контекста на приложението (`ServletContext`)
 - `include(ServletRequest, ServletResonse)` – **ВМЪКВА съдържанието на друг статичен или динамичен компонент**
 - `forward(ServletRequest, ServletResonse)` – **предава изпълнението на друг компонент (не трябва да е връщан отговор преди това)**
- `HttpServletResponse` – за разлика от диспечера, не прави обработка на сървъра, а връща пренасочващ отговор
 - `sendRedirect(String)` – **праща HTTP 302 с новия адрес**

Съхранение на състояние

- сесията се използва за корелация на поредица от заявки
- представя се като `HttpSession` обект достъпен от `request` обекта, чрез метод `getSession()`
- най-лесният начин за следене на сесията е чрез подаване на уникален идентификатор:
 - идентификаторът може да се подава като *cookie*
 - уеб контейнерът може да включи идентификатора като параметър на всяка заявка (URL rewriting)

Демонстрация

THE FOLLOWING **PREVIEW** HAS BEEN APPROVED FOR
ALL AUDIENCES
BY THE MOTION PICTURE ASSOCIATION OF AMERICA, INC.

THE FILM ADVERTISED HAS BEEN RATED



www.filmratings.com

www.mpaa.org

JavaServer Pages (JSP)

- технология за създаване на статични или динамични view-та
- използва HTML синтаксис и лесно се интегрира с JavaScript и CSS
- транслира се до сървлет, който се изпълнява на сървъра и връща генерираната репрезентация
- по-малко писане на Java код
- голям набор от Java EE сървъри поддържащи JSP спецификацията (дава portability)



JSP versus ASP

- много Java EE сървъри
- платформено независим
- поддържа Java (JavaScript)
- компилира се
- *модел*: JavaBeans и EJB-та
- *DB достъп*: чрез JDBC
- *разширяване*: чрез custom tag библиотеки



- Internet Information Services (IIS)
- Windows XP ↑
- поддържа VBScript (JScript)
- интерпретира се*
- *модел*: Wins 32 COM
- *DB достъп*: чрез ADO
- *разширяване*: не позволява

Идеята зад JSP

- JSP се базира на сървлет технологията
- всяко JSP в основата си е HTML страница с вложени специални JSP тагове (в които може да има Java код)
- JSP файлът има разширение .jsp (или.jspf за фрагменти)
- JSP engine-ът парсва JSP файла и създава Java сървлет. След това сървлетът се компилира до class файл.

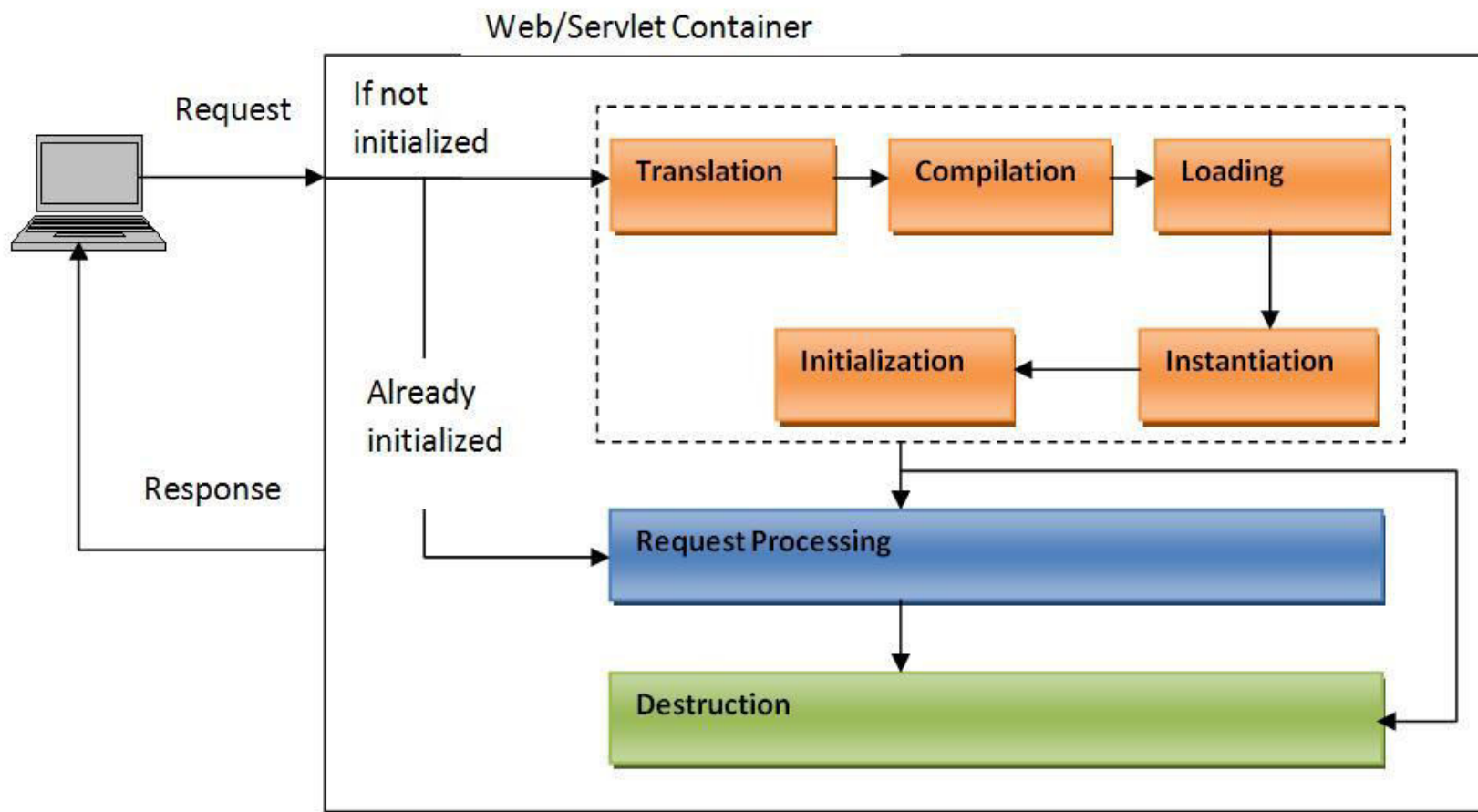


Примерно JSP

- при всяка заявка, ще се връща текущата дата и час

```
<html>
  <head><title>Date JSP example</title></head>
  <body>
    The date is:
    <% out.println(new java.util.Date()); %>
  </body>
</html>
```

Жизнен цикъл на JSP



- `jspInit()`, `jspDestroy()`
- `_jspService(HttpServletRequest, HttpServletResponse)`

JSP променливи по подразбиране

- следните променливи могат да се ползват в JSP-та:
 - `request` – за текущия `HttpServletRequest`
 - `response` – за текущия `HttpServletResponse`
 - `session` – текущата `HttpSession` (ако има такава)
 - `out` – текстов поток за JSP резултата (`PrintWriter`)
 - `application` – `ServletContext`-а на приложението
 - `exception` – за достъп до възникнала грешка
 - `config`, `pageContext`, `page`

JSP директиви (directive tags)

- JSP директивите влияят на цялостната структура на генерирания сървлет клас
- синтаксис: `<%@директива атрибут="стойност" %>`
`<jsp:directive.име атрибут="стойност" />`
- **page** директива - дефинира атрибути за JSP-то
 - дефинира import пакети:
`<%@ page import="java.util.*" %>`
 - MIME тип на резултата:
`<%@ page contentType="text/plain" %>`
 - дефиниране на имплицитна сесия:
`<%@ page session="true" %>`
 - задаване на JSP URL за грешки:
`<%@ page errorPage="url" %>`



JSP директиви (directive tags)

- **include** директива - позволява да включат други страници преди JSP-то да се конвертира до сървлет

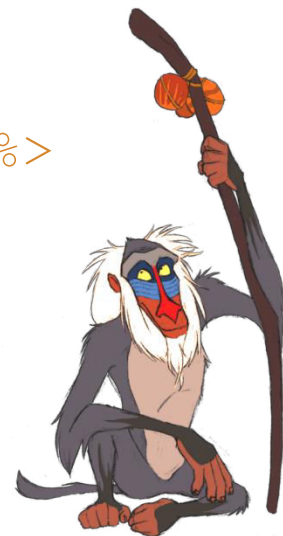
- **пример:** `<%@ include file="/opa/header.jsp" %>`

- **taglib** директива - декларира използването на библиотека с custom тагове

- **пример:** `<%@ taglib uri="uri" prefix="op" %>`

- XML формат на директивите:

- `<jsp:directive.page attribute="value" />`
 - `<jsp:directive.include file="url" />`
 - `<jsp:directive.taglib uri="uri" prefix="op"/>`



JSP действия (actions)

- XML елементи контролиращи поведението на сървлет engine-a
- синтаксис: `<jsp:име атрибут="стойност" />`
- примери:
 - `<jsp:include page="relative URL" flush="true" />`
 - `<jsp:forward page="url" />`
 - `<jsp:useBean id="opa" />`
 - `<jsp:setProperty name="opa" property="prop" value="val" />`
 - `<jsp:getProperty name="opa" property="prop" />`
 - `<jsp:text>Template data</jsp:text>`

JSP декларации (declaration tags)

- JSP декларации позволяват дефинирането на методи или член променливи в тялото на генерирания сървлет клас
- синтаксис: `<%! Java декларации; %>`
- обикновено не генерират съдържание, а се използват заедно с JSP изрази и скриплетите

- пример:

```
<%!  
    long counter = 0;  
    public void getCounter() {return counter;}  
%>
```



JSP изрази (expression tags)

- JSP изразите се използват за директно вмъкване на резултата от Java израз в изходния резултат от JSP-то
- синтаксис: `<%= Java израз %>`
- примери:
 - `<%= new java.util.Date() %>`
 - `<%= Math.PI %>`
 - `<%= request.getRemoteHost() %>`
 - `<%= session.getMaxInactiveInterval() %>`

JSP скриплетите (scriptlets)

- JSP скриплетите позволяват директното влягане на Java код в `_jspService()` метода на JSP-то
- синтаксис: `<% Java код; %>`
- скриплетите имат достъп до стандартните JSP променливи (`request`, `response`, `session`, ...)
- пример:

```
<% for (int i=0; i<10; i++) { %>  
    <%= i %> * <%= i %> = <%= i*i %>  
    <br>  
<% } %>
```



Демонстрация

THE FOLLOWING **PREVIEW** HAS BEEN APPROVED FOR
ALL AUDIENCES
BY THE MOTION PICTURE ASSOCIATION OF AMERICA, INC.

THE FILM ADVERTISED HAS BEEN RATED



www.filmratings.com

www.mpaa.org

Потребителски тагове (custom tags)

- компоненти за многократно използване позволяващи компонентно-ориентирана разработка в Java уеб приложенията
- всеки разработчик може да създаде custom tag
- скриват сложността на визуализацията
- изглеждат като HTML тагове, което улеснява уеб дизайнерите и разработчиците
- пример:

```
<mytags:opa someattr="value" />
```

Библиотека от тагове (tag library)

- съвкупност от пакетирани потребителски тагове
- всяка библиотека има префикс и URI идентификатор
- всяка таг библиотека се състои от:
 - TLD дескриптор описващ таговете
 - JAR с компилираните класове и ресурси на таговете
- използване:
 - TLD дескрипторът се слага в /WEB-INF
 - JAR-ът се поставя в /WEB-INF/lib
 - регистриране библиотеката в JSP-то:
`<%@ taglib prefix="mytags" uri="WEB-INF/my.tld" %>`
 - използване на тага: `<mytags:opa someattr="value" />`

JavaServer Pages Standard Tag Library (JSTL)

- представлява набор от потребителски тагове (custom tags) за многократно използване
- стандартна библиотека с тагове част от JSP 2.0
- имплементира често използвани функционалности:
 - *JSTL Core* - основни функции за работа с променливи, условия, цикли, операции за вход/изход, ...
 - *JSTL Format* - форматиране и интернационализация
 - *JSTL XML* - четене на XML данни , XSL трансформации
 - *JSTL SQL* - изпълнение на SQL заявки (!)
 - *JSTL Functions* - извикване на стандартни функции

Предимства на JSTL

- позволява JSP страниците да съдържат само XML
 - избягваме влягане на Java код в view-тата (чрез скриплети)
 - кодът става по лесен за четене и поддръжка
- спестяват се усилия на разработчика, защото може наготово да се използва често срещана функционалност
- лесно използване:
`<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>`

JSTL Core тагове

- работа с променливи:

- `<c:set var="n1" value="v1" scope="page" />`
- `<c:set var="n2" value="v2" scope="request" />`
- `<c:set var="n3" value="v3" scope="session" />`
- `<c:set var="n4" value="v4" scope="application" />`
- `<c:remove var="n1" scope="page" />`

- извеждане на данни:

- `<c:out value="{user.age}" escapeXml="false"/>`

- прихващане на грешки:

- `<c:catch var="DivideByZeroException">`
 `<% int x = 10/0; %>`
 `</c:catch>`

JSTL Core тагове (продължение)

- **условни операции:**

- `<c:if test='${param.p} == "val" '>`
Пропърти p на param е равно на "val"
`</c:if>`
- `<c:choose>`
 `<c:when test='${param.p}=="val" '>оп</c:when>`
 `<c:otherwise>няма оп</c:otherwise>`
`</c:choose>`

- **ЦИКЛИ:**

- `<c:forEach var='item' begin='1' end='10'>`
 `<c:out value='${item}' />`
`</c:forEach>`
- `<c:forEach var='item' items='${itemsList}'>`
 `<li><c:out value='${item}' /></li>`
`</c:forEach>`

Таг файлове

- файл съдържащ JSP фрагмент, който може да се използва многократно като потребителски таг
- препоръчва се използването на разширение .tag
- използване:
 - за разделяне на JSP-то на отделни модули
 - за повторно използване на логика
- видове:
 - самостоятелни (без тяло): `<m:ora />`
 - поддържащи други тагове (с тяло): `<m:ora>X</m:ora>`

Примерен таг файл

- файл с име opa.tag (самостоятелен):

```
<%@ tag body-content="empty" %>  
<%@ attribute name="firstName" required="true"%>  
<h1>Hello, <%= firstName %> !</h1>
```

- поставяне на opa.tag в /WEB-INF/tags

- деклариране:

```
<%@ taglib prefix="m" tagdir="/WEB-INF/tags" %>
```

- употреба:

```
<m:opa firstName="Penn"/>
```

Unified Expression Language (UEL)

- позволява лесното достъпване на обекти, техните пропъртита и методи от JSP и JSF страници
- синтаксис: `${ el израз }`
- замества JSP action таговете:
 - `<jsp:useBean id="user" scope="request" />`
 - `<jsp:setProperty name="user" property="name" value="Penn" />`
 - `<jsp:getProperty name="user" property="name"/>`
- пример:
 - `${opa.prop = "Penn"}`
 - `${opa.prop}`



Предимства на UEL

- дава кратка нотация за достъп до данни в scope-ове page, request, session и application

`${course}`

- лесен достъп до елементи на колекция

`${students[2]}`

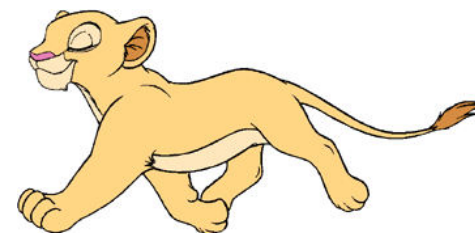
- лесен достъп до пропъртите на bean обекти:

`${course.presenter.name}`

- достъп до параметри на заявката, хедъри, cookies, ...

`${param["studentCount"]}`

`${cookie["dateOfLastVisit"]}`



Предимства на UEL (продължение)

- набор от прости оператори (+, -, *, /, =, <, >, ==, &&, ||, ?:, empty, not):

```
${ (2 + 5) * 3 }
```

```
${course.studentCount - 1}
```

```
${not empty presenter ? "тук съм" : "успях се" }
```

- автоматично конвертиране на типове към стринг
- възможност за извикване на функции:

```
${fn:length("It's the cycle of life...")}
```

- връща празен стринг вместо exception:

```
${thisIsNull.prop} → ""
```



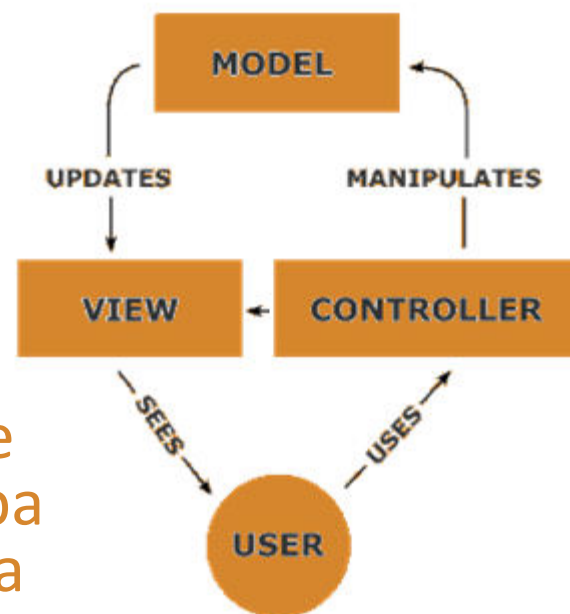
Достъп до обекти при UEL

- атрибутите могат да се търсят в конкретен score:
`${session.course}`
- ако името на атрибута не е валиден Java идентификатор, се използва алтернативната нотация:
`${session[course.with.dots]}`
- ако не се зададе score, търсенето на обекта се извършва от най-малкия към най-големия score:
 - `pageContext`
 - `request`
 - `session`
 - `application`



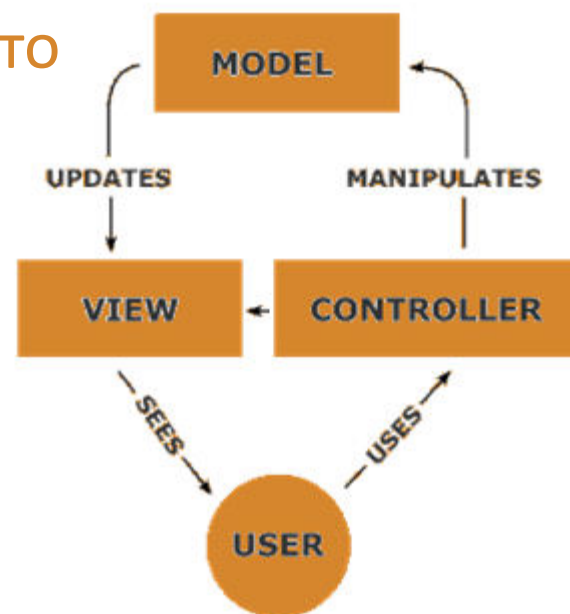
Model-View-Controller (MVC)

- софтуерен *pattern* разделящ презентацията на информация от работата на потребителя с нея
- използва се при визуални приложения
- разделя приложението на:
 - **model** - съхранява данните, състоянието и бизнес логиката; информира view-то при промяна на състоянието *
 - **view** - представя визуално данните от модела
 - **controller** - приема потребителските команди и на тяхна база манипулира данните в модела; праща сигнали за промяна на view-то



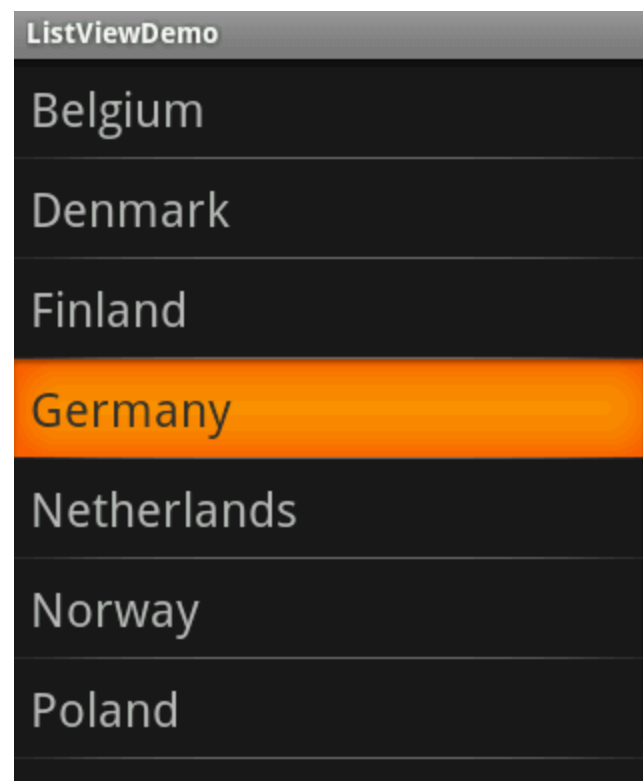
Model-View-Controller (продължение)

- ВИДИМОСТ МЕЖДУ КОМПОНЕНТИТЕ:
 - моделът не знае за останалите компоненти
 - view-то знае само за модела
 - контролерът знае за модела и view-то
- съставни части на *pattern-a*:
 - composite pattern (view)
 - strategy pattern (view и контролер)
 - observer pattern (view и модел)



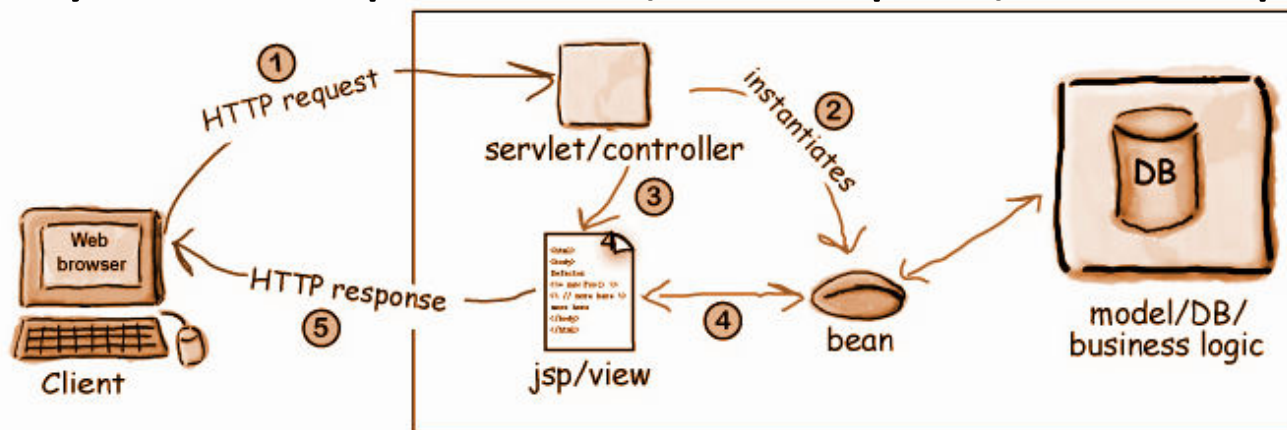
Model-View-Controller - пример

- потребителят изпраща команда за добавяне на елемент в списъка:
 - контролерът получава командата и модифицира модела
 - моделът информира view-то за промяната
 - view-то визуализира новия елемент
- потребителят изпраща команда за скролиране:
 - контролерът я получава и праща сигнал до view-то
 - view-то скролира списъка



MVC и мрежата (model 2)

- 1 потребителят праща заявка на даден URL
- 2 контролерът извиква модела и получава данни*
- 3 контролерът асоциира данните с view-то и предава контрола на view-то
- 4 view-то използва данните и изгражда презентация
- 5 получената презентация се връща на потребителя



Демонстрация

THE FOLLOWING **PREVIEW** HAS BEEN APPROVED FOR
ALL AUDIENCES
BY THE MOTION PICTURE ASSOCIATION OF AMERICA, INC.

THE FILM ADVERTISED HAS BEEN RATED



www.filmratings.com

www.mpaa.org

JavaServer Faces (JSF)

- Java framework за създаване на компонентно-базирани уеб приложения
- използва request-driven MVC:
 - модел - managed bean-ове, еjb-та и entity-та (beans)
 - view - JSF UI тагове и Facelets (xml файлове)
 - контролер - рутира заявки и избира view, което да се визуализира;
контролера е стандартен - `FacesServlet`



Out of scope

- За сървлети: filter-и, listener-и, web.xml дескриптор, сървлет анотации, изпращане на файлове, асинхронни сървлети, частични дескриптори и други
- За JSP-та: детайли за JSP директивите и JSTL тагове (Format, SQL, XML, функции), създаване на потребителски тагове, deferred UEL изрази
- За JSF технологията: почти всичко 😊



Въпроси



Благодаря за вниманието

