PV178: Programming for the CLI Environment

Seminar: Week 6

Tomáš Pochop

Institute of Computer Science and **Faculty of Informatics**
Masaryk University

April 15, 2007

Seminar outline

Delegates

1. Delegate is a class that holds a reference to a method
2. Allows to pass a reference to a method just like variable
3. Any method with the signature (return type and parameters) that match to the delegate declaration can be passed
4. Declaration:

   `delegate` *returnType DelegateName*(*parameters*);

Simple delegates example

```
//declare a delegate
public delegate int MyDelegate(int a, int b);
//method that match the delegate
public int MyMethod(int x, int y)
{
  //do something
}
//declaration of a new delegate variable
MyDelegate del;
//del now points to the MyMethod
del = new MyDelegate(MyMethod);
//execute the MyMethod method via the del delegate
int result = del(2,4);
```

Delegates task

Declare one delegate `int MathOperation(int, int)` and implement four methods that match the delegate declaration. These methods perform basic integer math operation (+,-,*,/). In the Main method create one instance of the MathOperation delegate and perform those operations via the delegate using methods above.

Events

1. Events are a special form of delegates
2. An event is a message sent by an object to signal that some action occurs
3. An object that raises an event is called event generator
4. An object that is interesting in an event is called consumer
5. The consumer contains a method that is performed when an event occurs and must be registred to the event, this method is called event handler
6. The method is registred to an event via delegate

Simple events example

1. An event delegate declaration:
   `public delegate void MyEventDelegate(int i, string s);`
2. An event declaration:
   `public event MyEventDelegate MyEvent;`
   The MyEvent event match the MyEventDelegate signature and all its event handlers must match that signature too.
3. Raising an event:
   `MyEvent(1, "some text");`
   The event is called in the same way as a method
4. An event handler definition:
   `public void MyEventHandler(int i, string s){ }`
5. Registration of an event handler:
   `MyEvent += new MyEventDelegate(MyEventHandler);`
   It can be registred one or more event handlers
6. See the SimpleEvents example

Windows application task

Write a simple text editor windows application. The application allows simple text edition, clipboard functions (copy, cut, paste), save/load to/from a file (simple Unicode text), change font and text color (affects only editor environment).
Application is build of one window form, one menu strip to access all functions (new file, load file, save file, close application, cut/copy selected text, paste text, font change, color change), one tool strip and one status bar (there show the path of the current file). For save and load file allow user to choose the file in standard dialog, the same for font and color changing.

Form

1. `System.Windows.Forms.Form`
2. Represents a window or dialog box that makes up an application's user interface.

RichTextBox

1. `System.Windows.Forms.RichTextBox`
2. Represents a Windows rich text box control, a multiline text area.

MenuStrip, ToolStripMenuItem

1. `System.Windows.Forms.MenuStrip`
2. Provides a menu system for a form.

1. `System.Windows.Forms.ToolStripMenuItem`
2. Represents a selectable option displayed on a `MenuStrip` or `ContextMenuStrip`.

ToolStrip, ToolStripItem

1. `System.Windows.Forms.ToolStrip`
2. Provides a container for Windows toolbar objects.

1. `System.Windows.Forms.ToolStripItem`
2. A `ToolStripItem` is an element such as a button, combo box, text box, or label that can be contained in a `ToolStrip` control.

StatusStrip

1. `System.Windows.Forms.StatusStrip`
2. Represents a Windows status bar control.
3. A `StatusStrip` control displays information about an object being viewed on a Form, the object's components, or contextual information that relates to that object's operation within the application.
4. The StatusStrip can contain `ToolStripStatusLabel`, `ToolStripDropDownButton`, `ToolStripSplitButton`, and `ToolStripProgressBar` controls.
5. The default StatusStrip has no panels.

OpenFileDialog

1. `System.Windows.Forms.OpenFileDialog`
2. Prompts the user to open a file.
3. `public string FileName`
   Gets or sets a string containing the file name selected in the file dialog box.
4. `public DialogResult ShowDialog()`
   Shows the dialog box, returns `DialogResult.OK` if the user clicks OK button; otherwise, `DialogResult.Cancel`.
5. `public string Filter`
   Gets or sets the current file name filter string, which determines the choices that appear in the "Save as file type" or "Files of type" box in the dialog box.

SaveFileDialog

1. `System.Windows.Forms.SaveFileDialog`
2. Prompts the user to select a location for saving a file.
3. Similar to the OpenFileDialog

FontDialog

1. `System.Windows.Forms.FontDialog`
2. Prompts the user to choose a font from among those installed on the local computer.
3. `public Font Font`
   Gets or sets the selected font.
4. `public DialogResult ShowDialog()`
   Shows the dialog box, returns `DialogResult.OK` if the user clicks OK button; otherwise, `DialogResult.Cancel`.

ColorDialog

1. `System.Windows.Forms.ColorDialog`
2. Represents a common dialog box that displays available colors along with controls that enable the user to define custom colors.
3. `public Color Color`
   Gets or sets the color selected by the user.
4. `public DialogResult ShowDialog()`
   Shows the dialog box, returns `DialogResult.OK` if the user clicks OK button; otherwise, `DialogResult.Cancel`.