Windows Programming

Form Programming

Contrasting Windows and Console Applications

- Console applications
  - Each line in Main() method executed sequentially – then the program halts
  - Method calls might branch to different locations in the program, however control always returns back to Main
  - Program initiates interaction with the user by calling the OS to get data using `ReadLine()` method
  - Program calls OS to output data through method calls like `WriteLine()` or `Write()`
  - Console applications run IPO (Input-Process-Output) model of computing process

Contrasting Windows and Console Applications

- Windows applications
  - Instead of the program executing sequential statements from top to bottom, the application, once launched, sits in what is called a process loop and waits for an event
  - Sits in a process loop, waiting for event to execute
  - Windows applications run Event-driven model of a computing process

Contrasting Windows and Console Applications

- Event: notification from OS that an action, such as the user clicking the mouse or the user pressing a key, has occurred
- Instead of calling the OS with IO request as in console applications, Windows applications receive messages from OS that event has occurred
- It is must to write methods, called Event Handlers to indicate what should be done when an event occurs

Graphical User Interfaces

- Windows applications also look different from console applications
- User Interface: front end of a program
  - Visual image you see when you run a program
  - Algorithmic functionality stays behind GUI
  - Often users of programs actually identify the interface of the program itself, when in reality, the interface is just one facet of the program.
  - Algorithmic complexity is hidden behind GUI.
Graphical User Interfaces

- Graphical user interface (GUI) includes:
  - Menus, labels, text boxes, list boxes,
  - Other controls (pictures, buttons, etc.),
  - Text in many different colors and sizes
- Controls are objects that can display and respond to user interaction.
- Controls are placed in a container, called form, which is an object as well. It is instance of a class derived from predefined class Form.

Windows Applications

- Reference and import
  System.Windows.Forms namespace
- Class heading definition
  - Includes not only the class name, but a colon followed by another class name
  - Derived class (first class)
  - Colon (:) – delimiter, separator
  - Base class (second class)
  - public class Form1 : Form
- Derived classes inherit from base class

Windows Applications (continued)

- Text
  - A property for setting/getting title bar caption
  - Can be used in constructor
- Windows forms/controls offer many properties including Text, Color, Font, and Location
- Execution begins in Main() method
  - Main() is located in Program.cs file for the application
  - Call to Run() method places application in process loop

Elements of Good Design

- Appearance matters
  - Human-computer interaction (HCI) research
- Design considerations
  - Consistency
  - Alignment
  - Avoid clutter
  - Color
  - Target audience
Using C# & Visual Studio to Create Windows based Applications

- **Primitive approach**
  - Manually create Win based application in C# using a text editor like Notepad, or Textpad etc

- **Professional approach**
  - Using IDE like MS Visual Studio
  - Select File Menu: New >> Project
  - Select Visual C#, Windows and Windows Forms Application from Visual Studio Installed Templates pan

Use Visual Studio to Create Windows-Based Applications

- Select File New Project
- Browse to location to store your work
- Name
- Windows Forms Application template
- Figure 9-2 Visual Studio New Windows application

Windows-Based Applications

- Toolbox
- Initial design screen
- Design View
- Properties Window

Windows-Based Applications (continued)

- Dockable windows
- Pushpin icon and dockable panes (auto hide state)

Windows Based Applications

- MS VS automatically generates code to create a blank Windows Form object.
- If you see Code Editor pane instead, you press Shift+F7 to view Designer.
- Other panes used at application design time: Toolbox, Solution Explorer, Properties
- Other panes used at application run time: Error List pane and Output pane
- Extensive collection of Control classes
- Top-level window for an application is called a Form
- Each control has large collection of properties and methods
  - Select property from an alphabetized list (Properties window)
  - Change property by clicking in the box and selecting or typing the new entry
Windows Based Applications

- How to run Windows based application?
- Like console applications
  - Select from Debug menu
  - Start Debugging (shortcut F5)
  - Or
  - Start Without Debugging (Ctrl+F5)

The Form as a Platform

- A Windows application *form* can act as a *container* for additional objects.
- Objects that can be contained by forms (and are coded by existing classes in the Foundation Class Library) are called *controls*.

Windows Forms

- Task: Modify the *PureFCLForm* application
- Find appropriate property and type a new value
  - Change the following Form properties:
    - Text
    - BackColor
    - Size
    - FormBorderStyle

System.Windows.Forms

- This namespace contains all of the controls used on the average Windows interface
- A control is a higher-level object composed of
  - A window in which the control is drawn
  - Visual parts of the control which are drawn in the window
  - A set of delegates which are triggered when various events occur
Form Methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Activate</td>
<td>Activates the window and gives it focus</td>
</tr>
<tr>
<td>Close</td>
<td>Closes the form</td>
</tr>
<tr>
<td>Show</td>
<td>Makes the form visible</td>
</tr>
<tr>
<td>BringToFront</td>
<td>Moves to top of stacking order</td>
</tr>
<tr>
<td>Hide</td>
<td>Makes the form invisible</td>
</tr>
<tr>
<td>Focus</td>
<td>Gives the form focus</td>
</tr>
</tbody>
</table>

Windows Form Properties

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Property value</th>
</tr>
</thead>
<tbody>
<tr>
<td>AutoScaleDimensions</td>
<td>DPI resolution of display it was built for. Will be scaled to look correct on other displays.</td>
<td></td>
</tr>
<tr>
<td>BackColor</td>
<td>Background color</td>
<td></td>
</tr>
<tr>
<td>ForeColor</td>
<td>Foreground or drawing color</td>
<td></td>
</tr>
<tr>
<td>Text</td>
<td>Text displayed or caption</td>
<td></td>
</tr>
<tr>
<td>Location</td>
<td>Point of to left corner</td>
<td></td>
</tr>
<tr>
<td>MinimumSize</td>
<td>Minimum size window can be resized to</td>
<td></td>
</tr>
<tr>
<td>MaximumSize</td>
<td>Maximum size window can be resized to</td>
<td></td>
</tr>
<tr>
<td>DefaultSize</td>
<td>Size when initially resized</td>
<td></td>
</tr>
<tr>
<td>ClientSize</td>
<td>Size of drawing area without borders or scrollbars</td>
<td></td>
</tr>
<tr>
<td>Text</td>
<td>Typed “First Windows Application”</td>
<td></td>
</tr>
</tbody>
</table>

Windows Form Events

- Add code to respond to events, like button clicks
- From the Properties window, select the lightning bolt (Events)
  - Double-click on the event name to generate code
  - Registers the event as being of interest
  - Adds a heading for event-handler method

Form Events

- Forms provide support for a large number of events
- You add one or more delegates to these events
- When the event happens, the delegates are invoked
- The delegates must have the signature of an event handler
  ```csharp
  void EventHandler(object sender, EventArgs e)
  ```
Form Events

<table>
<thead>
<tr>
<th>Event</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Load</td>
<td>Just before form is loaded the first time</td>
</tr>
<tr>
<td>Closing</td>
<td>Just before the form is closed</td>
</tr>
<tr>
<td>Closed</td>
<td>When the form is actually closed</td>
</tr>
<tr>
<td>Shown</td>
<td>Occurs when a form is first displayed</td>
</tr>
<tr>
<td>ResizeBegin</td>
<td>Resize operation has begun</td>
</tr>
<tr>
<td>ResizeEnd</td>
<td>Resize operation has ended</td>
</tr>
</tbody>
</table>

Windows Form Properties (continued)

Windows Form Events (continued)

Expand Form1.cs node to reveal the Form1.Designer.cs file

Solution Explorer window

Windows Form – Closing Event

Code automatically added to register event
```
this.Closing += new System.ComponentModel.CancelEventHandler
    (this.Forms_Closing);
```

Code automatically added for method heading
```
private void Forms_Closing(object sender,
    System.ComponentModel.CancelEventArgs e) {
```

You add statement to event-handler method body
```
    MessageBox.Show("Hope you are having fun!");
```

Simple Windows Application

IDE separates the source code into three separate files
- Form1.cs: normally this is the only one you edit
- Form1.Designer.cs: holds the auto generated code
- Program.cs: contains the Main() method, where execution always begins

Form1.cs and Form1.Designer.cs both include partial class definitions for the Form1 class

Your First C# Windows Form Program 07-01
Controls are all classes
- Button, Label, TextBox, ComboBox, MainMenu, ListBox, CheckBox, PictureBox, MenuStrip, RadioButton, and MonthCalendar

Each comes with its own predefined properties and methods

Each fires events

Each is derived from the System.Windows.Forms.Control class

Standard Controls

Windows Forms controls

Properties of the Control Class

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Anchor</td>
<td>Gets or sets which edges of the control are anchored to the edges of its container.</td>
</tr>
<tr>
<td>BackColor</td>
<td>Gets or sets the background color for the control.</td>
</tr>
<tr>
<td>BackgroundImage</td>
<td>Gets or sets the background image displayed in the control.</td>
</tr>
<tr>
<td>CanRaiseFocus</td>
<td>Gets a value indicating whether the control can receive input focus.</td>
</tr>
<tr>
<td>CanSelect</td>
<td>Gets a value indicating whether the control can be selected.</td>
</tr>
<tr>
<td>Enabled</td>
<td>Gets or sets a value indicating whether the control can respond to user interaction.</td>
</tr>
<tr>
<td>Focusable</td>
<td>Gets a value indicating whether the control can have input focus.</td>
</tr>
<tr>
<td>Font</td>
<td>Gets or sets the font of the text displayed by the control.</td>
</tr>
<tr>
<td>ForeColor</td>
<td>Gets or sets the foreground color of the control.</td>
</tr>
<tr>
<td>Location</td>
<td>Gets or sets the coordinates of the upper-left corner of the control relative to the upper-left corner of its container.</td>
</tr>
<tr>
<td>Name</td>
<td>Gets or sets the name of the control.</td>
</tr>
<tr>
<td>Size</td>
<td>Gets or sets the height and width of the control.</td>
</tr>
<tr>
<td>TagIndex</td>
<td>Gets or sets the tab order of the control within its container.</td>
</tr>
<tr>
<td>Dock</td>
<td>Gets or sets the text associated with the control.</td>
</tr>
<tr>
<td>Visible</td>
<td>Gets or sets a value indicating whether the control is displayed.</td>
</tr>
</tbody>
</table>

Methods of the Control Class

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Focus( )</td>
<td>Sets the input focus to the control.</td>
</tr>
<tr>
<td>Hide( )</td>
<td>Conceals the control from the user.</td>
</tr>
<tr>
<td>Select( )</td>
<td>Activates a control.</td>
</tr>
<tr>
<td>Show( )</td>
<td>Displays the control to the user.</td>
</tr>
</tbody>
</table>
Label Objects

- Provide descriptive text or labels for other controls
- Instantiate object
  Label labelName = new Label();
- Add control to Form
  this.Controls.Add(labelName);
- Set property values (some from Control class)
  - Text; TextAlign; Font; Location

Creating a TaxApp

Creating a TaxApp Form

Formatting Label objects

Adding Labels to TaxApp Form

Table 9-4 TaxApp Form properties

<table>
<thead>
<tr>
<th>Property</th>
<th>Changes with explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>BackgroundColor</td>
<td>Selected blue from the Custom tab could have typed 192, 192, 255 (RGB code for that selection)</td>
</tr>
<tr>
<td>Font</td>
<td>Selected Bookman Old Style, regular 12 point; changed the font on the form, so that all controls added to this container would have this font set</td>
</tr>
<tr>
<td>Size</td>
<td>Changed the size of the window from 300, 300 to 300, 260</td>
</tr>
<tr>
<td>Text</td>
<td>Typed “Windows Tax App” for the title bar caption</td>
</tr>
</tbody>
</table>

TextBox Objects

- Used to enter data or display text during run time
  - Used for both input and output
- Instantiate object
  TextBox textBoxName = new TextBox();
- Add control to Form
  this.Controls.Add(textBoxName);
- Interesting properties
  - MultiLine, ScrollBars, MaxLength, PasswordChar, CharacterCasing

Creating a TaxApp

Creating a TaxApp Form

Formatting Label objects

Adding Labels to TaxApp Form

Table 9-5 TaxApp label object properties

<table>
<thead>
<tr>
<th>Property</th>
<th>Changes with explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>BackColor</td>
<td>Selected medium blue from Custom tab could have typed 50, 50, 192</td>
</tr>
<tr>
<td>ForeColor</td>
<td>Selected style of bold italic</td>
</tr>
<tr>
<td>Font</td>
<td>Selected light blue from Custom tab could have typed 128, 128, 255 (RGB code for that selection)</td>
</tr>
<tr>
<td>Text</td>
<td>Typed “<em>1.4</em>”</td>
</tr>
</tbody>
</table>

TextBox Objects (continued)

Table 9-6 TextBox properties

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AcceptsReturn</td>
<td>Gets or sets a value indicating whether the Enter key inserts a newline of text in a multiline text box control</td>
</tr>
<tr>
<td>AcceptsTab</td>
<td>Gets or sets a value indicating whether the Tab key inserts a tab into text of a multiline text box control</td>
</tr>
<tr>
<td>CharacterCasing</td>
<td>Gets or sets whether the text box control modifies the case of the characters as they are typed</td>
</tr>
<tr>
<td>MaxLength</td>
<td>Gets or sets the maximum number of characters the user can type or paste into the text box control</td>
</tr>
<tr>
<td>Modified</td>
<td>Gets or sets a value indicating that the text box control has been modified since creation or when its contents were last set</td>
</tr>
<tr>
<td>MultiLine</td>
<td>Gets or sets a value indicating whether this is a multiline text box control</td>
</tr>
<tr>
<td>PasswordChar</td>
<td>Gets or sets the character used to mask characters in a single-line text box control</td>
</tr>
<tr>
<td>ReadOnly</td>
<td>Gets or sets a value indicating whether text in the text box is read-only</td>
</tr>
<tr>
<td>Scrollbars</td>
<td>Gets or sets which scroll bars should appear in a multiline text box control</td>
</tr>
<tr>
<td>TextAlign</td>
<td>Gets or sets how text is aligned in a text box control</td>
</tr>
<tr>
<td>WordWrap</td>
<td>Indicates whether a multiline text box control automatically wraps words to the beginning of the next line</td>
</tr>
</tbody>
</table>
Adding TextBox Objects to TaxApp Form

Add TextBox objects, then set their property values

<table>
<thead>
<tr>
<th>Object</th>
<th>Property</th>
<th>Changes with explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>txtBox1</td>
<td>Name</td>
<td>Typed txtPurchase</td>
</tr>
<tr>
<td>txtBox2</td>
<td>Name</td>
<td>Typed txtTotalDue</td>
</tr>
<tr>
<td>txtPurchase</td>
<td>TextAlign</td>
<td>Selected right</td>
</tr>
<tr>
<td>txtTotalDue</td>
<td>TextAlign</td>
<td>Selected right</td>
</tr>
</tbody>
</table>

Adding Button Objects to TaxApp Form

• Enables user to click button to perform task
  • If button has event-handler method and is registered as an event to which your program is planning to respond, event-handler method is called automatically when button clicked
  • Button object’s properties, methods, and events
    • Inherits from Control
      • Text, Enabled, Focused, TabIndex

Adding Button Objects to TaxApp Form (continued)

<table>
<thead>
<tr>
<th>Property</th>
<th>Changes with explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>Typed btnCompute</td>
</tr>
<tr>
<td>BackColor</td>
<td>Selected Navy</td>
</tr>
<tr>
<td>Font</td>
<td>Selected style of bold, point size of 14</td>
</tr>
<tr>
<td>ForeColor</td>
<td>Selected yellow from Custom tab; could have typed 255, 255, 128 (RGB code for that selection)</td>
</tr>
<tr>
<td>TabIndex 2</td>
<td>textbox.Text</td>
</tr>
<tr>
<td>Text</td>
<td>“Compute”</td>
</tr>
</tbody>
</table>

Adding Button Objects to TaxApp Form (continued)

• When you double-click on event, an event-handler method is created:

```csharp
private void btnCompute_Click(object sender, System.EventArgs e)
{
    string inValue;
    double purchaseAmt, percent, ans;
    inValue = txtPurchase.Text;
    purchaseAmt = Int32.Parse(inValue);
    inValue = label5.Text; // inValue previously declared as string
    inValue = inValue.Remove(inValue.Length -1, 1);
    percent = double.Parse(label5.Text.Remove(label5.Text.Length -1, 1)) / 100;
    ans = (double.Parse(label5.Text.Remove(label5.Text.Length -1, 1))) / 100;
    txtTotalDue.Text = String.Format("{0:C}", ans).ToString();
}
```

• AND registers click event:

```csharp
this.btnCompute.Click += new System.EventHandler(this.btnCompute_Click);
```
Creating Windows Applications

- We will demonstrate how to build a simple GUI interface using a text editor
- Most of the time, the designer in Visual Studio will be used
- Doing it by hand
  - Shows how it works under the hood
  - Provides an understanding so you can create your own controls

Creating Windows Applications

- In creating a GUI application we will use
  - Application – a class with static methods to control operation of an application
  - Label – a widget that can display static text or an image
  - Button – a push button with a textual or image displayed. Able to respond to mouse clicks.

Visual Studio Designer

- This is a drag and drop interface for drawing a GUI
- The code is automatically generated
- You can hook event handlers onto the events and write the code for them
- It speeds writing code
- You cannot make major modifications to the code it generates

A Form with One Button

- Using one of the 4 described techniques, we will add a button to a new form.
- We will program a handler function that will respond to the Click event, generated when the mouse is clicked while its pointer is positioned over the button.
- This event is termed a Button.Click event.

CheckBoxes

- Labeled boxes which can be checked or unchecked
  - Checked – get/set Boolean to determine if box is checked
  - CheckedChanged – delegate called when the box is checked or unchecked
**GroupBox**

- Displays a border around a group of controls
- Can have optional label controlled by Text property
- Controls can be added by
  - Placing them within the group box in the designer
  - Adding to the Controls list programmatically

**Radio Buttons**

- Radio buttons are similar to checkboxes, but
  - Appear slightly different
  - Allow buttons to be grouped so that only one can be checked at a time
- A group is formed when the radio buttons are in the same container – usually a group box or panel

**Radio Buttons**

- Checked – get/set Boolean indicating if the button is checked
- CheckedChanged – delegate invoked when the button is checked or unchecked

**Panels**

- A panel is like a group box but does not have a text label
- It contains a group of controls just like group box
  - BorderStyle – get/set border style as
    - BorderStyle.Fixed3D
    - BorderStyle.FixedSingle
    - BorderStyle.None

**TextBox**

- This is a single line or multi-line text editor
  - Multiline – get/set Boolean to make multiline
  - AcceptsReturn – in a multiline box, if true then pressing Return will create a new line. If false then the button referenced by the AcceptButton property of the form, will be clicked.
  - PasswordChar – if this is set to a char, then the box becomes a password box

Examples 07-
TextBox

- **ReadOnly** – if true, the control is grayed out and will not accept user input
- **ScrollBars** – determines which scrollbars will be used: ScrollBars.None, Vertical, Horizontal, Both
- **TextAlign** – get/set
  - HorizontalAlignment.Left, Center, or Right
- **TextChanged** – event raised when the text is changed

File Dialog

- **InitialDirectory** – string representing the directory to start in
- **Filter** – a string indicating the different types of files to be displayed
  - A set of pairs of display name and pattern separated by vertical bars
    - Windows Bitmap|*.bmp|JPEG|*.jpg|GIF|*.gif
- **FilterIndex** – the filter to use as an origin index

File Dialog

- **FileName** – the name of the file selected
- **ShowDialog** – a method to show the dialog and block until cancel or OK is clicked

```csharp
if (openDialog.ShowDialog() == DialogResult.OK)
{
    Image img = Image.FromFile(openDialog.FileName);
    pictureBox1.Image = img;
}
```

Image Class

- An abstract class that can store an image
- Several concrete classes are used for image types such as BMP, GIF, or JPG
  - **FromFile** (string `fname`) – loads any supported image format from a file
  - **FromStream** (stream) – loads an image from a stream
- **Height** – image height
- **Width** – image width

PictureBox Class

- This displays an image
  - **Image** – assigned an Image object to display
  - **SizeMode** – determines what to do if the image does not fit into the window
    - Normal
    - StretchImage
    - AutoSize
    - CenterImage
    - Zoom
These are the small pop-up boxes which explain the purpose of a control.

To use:
- Create a new tooltip in the designer
- Drop the tooltip onto the form
- The tooltip will appear on a tray below the form

After the tooltip appears in the tray, a new tooltip property appears for every component.
This can be assigned different text for each component.
That text will be displayed when the mouse hovers over that component.

This allows the selection of an integer from a limited range.
Also called a spinner.
- Minimum – smallest selectable value
- Maximum – largest selectable value
- Increment – size of increment per click
- Value – the selected value
- ValueChanged – event raised when the value changes

A control which displays a calendar for the selection of a range of dates.
- MinDate – the first selectable date
- MaxDate – the last selectable date
- SelectionStart – DateTime of start of selection
- SelectionEnd – DateTime of end of selection
- DateChanged – event raised when date is changed
DateTimePicker

- Similar to a month calendar but
  - Calendar pulls down and selection displayed
  - More configurable
  - Selects a single value, not a range
- Properties/methods
  - Format – Long, Short, Time, Custom
  - Value – DateTime value selected
  - ValueChanged – event which fires when date or time changes

ListBox

- The ListBox presents a list of items which can be selected
- A scrollbar is displayed if needed
- MultiColumn – displays list as multiple columns
- SelectedIndex – index of selected item
- SelectedIndices – collection of selected indices
- SelectedItem – the selected item
- SelectedItems – collection of selected items
- SelectionMode – how items can be selected
  - None – no selection
  - One – single selection
  - MultiSimple – each click selects additional item
  - MultiExtended – uses shift and control keys
  - Sorted – if true the items will be sorted alphabetically
- Items – a collection of items in the list box
- ClearSelected – method to clear selection
- GetSelected – returns true if the parameter passed is selected
- SelectedIndexChanged – event when selection changes

Populating a ListBox

- Any object can be placed into a ListBox
- The display is generated by ToString()

```csharp
for(int i = 0; i < 50; i++) {
    listBox1.Items.Add(
        "Item " + i.ToString());
}
```
A combo box is like a list but lets you display a selected value.

The list pulls down when a selection is being made.

Options allow the selected text to be editable or to require it to be selected from the dropdown list.

**ComboBox**

- **DropDownStyle** –
  - **Simple** – text is editable & list always visible
  - **DropDown** – default indicating text is editable & user must click to see list
  - **DropDownList** – value is not editable & user must click to see list

- **Items** – the collection of items in the list

**ComboBox**

- **MaxDropDownItems** – max number of items in pulldown before scrollbar used
- **SelectedIndex** – index of selection
- **SelectedItem** – selected item
- **Sorted** – whether entries are sorted
- **SelectedIndexChanged** – event raised when selection changes