Instructor: Bruce Reynolds

Introduction to Applications in C# Class 7

Concepts from Last Week

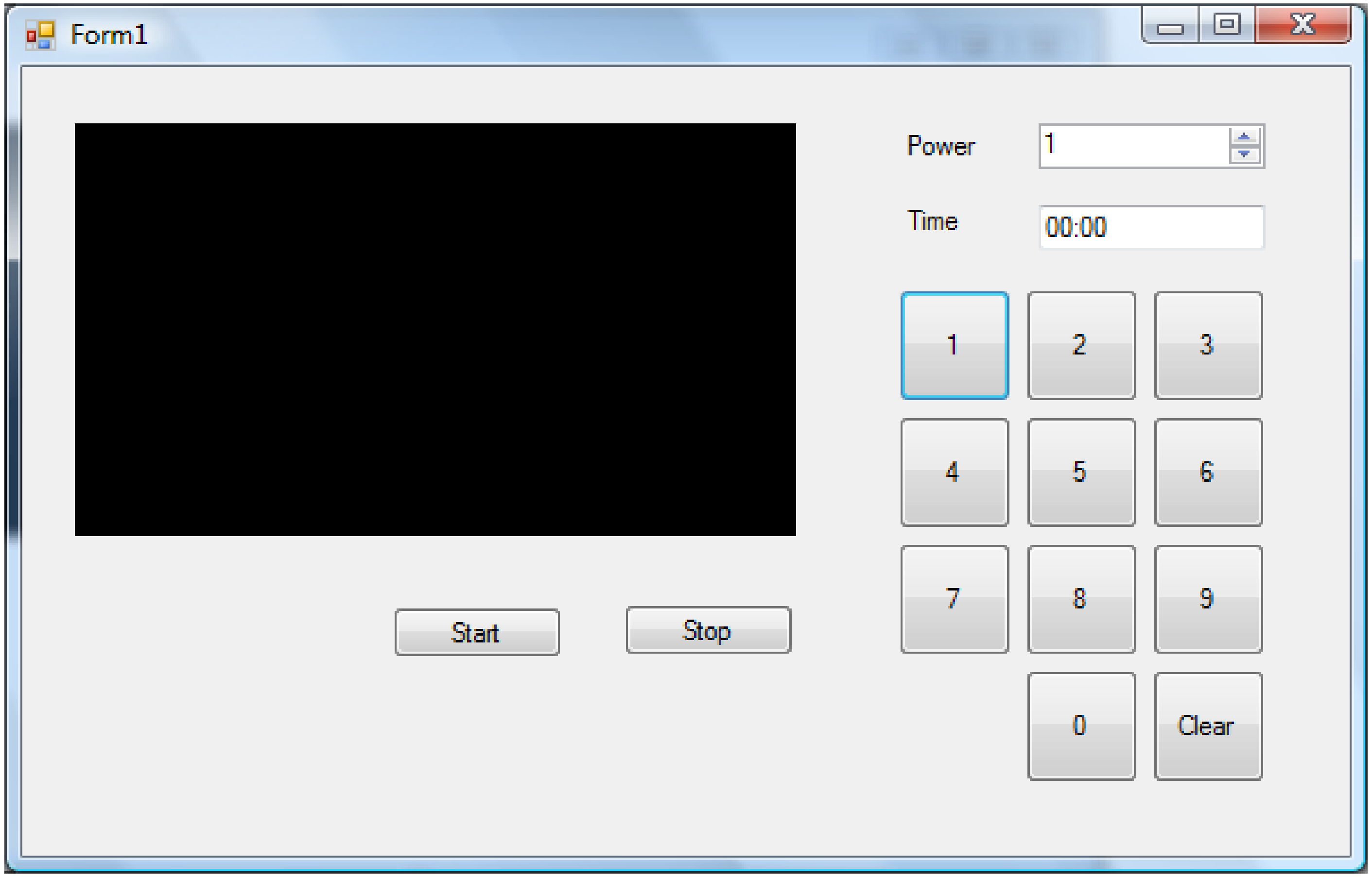
- Windows Forms
  - Windows Controls
    - Button, NumericUpDown, MaskedTextBox
    - ListBox, RadioButton, PictureBox, GroupBox
  - Event Handlers
  - Smart Tags
    - ListBox string editor
    - MaskedTextBox text format

Homework Review

Concepts for Week 7

- Windows Forms
  - Controls
    - Timer, TreeView, Dialogs, Menus
    - PictureBox, ImageList, ComboBox
  - Multicast delegates
  - Images and handling resources
  - Adding another form
  - DataSource property

Microwave Improvements



Keypad on Microwave

- Add ten Button controls.
  - Set the Text property to “0”, “1”, “2”, etc.
  - Name them “keypad1”, “keypad2”, etc.
- Add a MaskedTextBox control.
  - Set the Mask property to “90:00”
- Keypad code
  - We could add code to the click event of each control, but that code would be very repetitive. Repetitive code is prone to error – cut and paste and updating.

You could write this…

```
private void button1_Click(object sender, EventArgs e)
{
   string t = txtTime.Text;
   t = t.Substring(0, 2) + t.Substring(3, 2);
   t = t + '1';
   if (t.Length > 4)
       t = t.Substring(1, 4);
    txtTime.Text = t.Substring(0, 2) + ":" + t.Substring(2, 2);
}

private void button2_Click(object sender, EventArgs e)
{
   string t = txtTime.Text;
   t = t.Substring(0, 2) + t.Substring(3, 2);
   t = t + '2';
   if (t.Length > 4) t =
       t.Substring(1, 4);
    txtTime.Text = t.Substring(0, 2) + ":" + t.Substring(2, 2);
}
```

- But the only difference between the methods is the third line of code.

Factor Out the Method

- Use the Refactor feature to pull out the method and then look for commonalities:

```
private void UpdateTime(char digit)
{
   string t = txtTime.Text;
   t = t.Substring(0, 2) + t.Substring(3, 2);
   t = t + digit;
   if (t.Length > 4)
       t = t.Substring(1, 4);
   txtTime.Text = t.Substring(0, 2) + ":" + t.Substring(2,2);
}
```

One Better…

- Though even this algorithm has limitations.

```
private void UpdateTime(string numeral)
{
    input += numeral;
    input = input.Remove(0, 1);
    string strTime = input.Substring(0, 2) + ":"
        + input.Substring(2, 2);
    DateTime time;
    if (DateTime.TryParse(strTime, out time))
    {
            txtTime.Text = input;
    }
}
```

Call the Method

- Using the new method, we have:

```
private void ctlKeypad8_Click(object sender, EventArgs e)
{
   UpdateTime("8");
}

private void ctlKeypad9_Click(object sender, EventArgs e)
{
   UpdateTime("9");
}

private void ctlKeypad0_Click(object sender, EventArgs e)
{
   UpdateTime("0");
}
```

Use Multicast Delegates

- Add this code:

```
private void KeypadClick(object sender, EventArgs e)
{
    Button key = (Button)sender;
    UpdateTime(key.Text);
}
```

- Cast the “sender” parameter as a Button.
- Use the Properties windows to set this as the Click method for all the buttons.
- We’ve gone from 10 Click methods to 1 Click method and one UpdateTime method.

Timer Control

- The Timer control is a component and sits in the Component Tray. It does not appear on the form at runtime.
- The Timer raises a Tick event at intervals determined by the Interval property.
- You need to set the Enabled property to true to start the timer.

Timer Control and Microwave

- Drag a Timer control on the form.
  - Set Interval to 1000 (1 second).
  - Set Enabled to false (it won't generate events.)
- What we want to happen:
  - Start button – starts the timer
  - Tick event – reduce the time by one second
    - We need to keep track of the number of seconds left
  - Seconds = 0 – turn off the timer

Add a Form (Class) Variable

- Declare this variable in the form, outside of any methods.

```
decimal timeLeft = 0;
```

- Add this code to the Start button:

```
DateTime time;
if (DateTime.TryParse(txtTime.Text, out time))
{
    timeLeft = time.Hour + time.Minute;
    timer1.Enabled = true;
}
```

Add the Timer Event Handler

- Add this code to the Stop button:

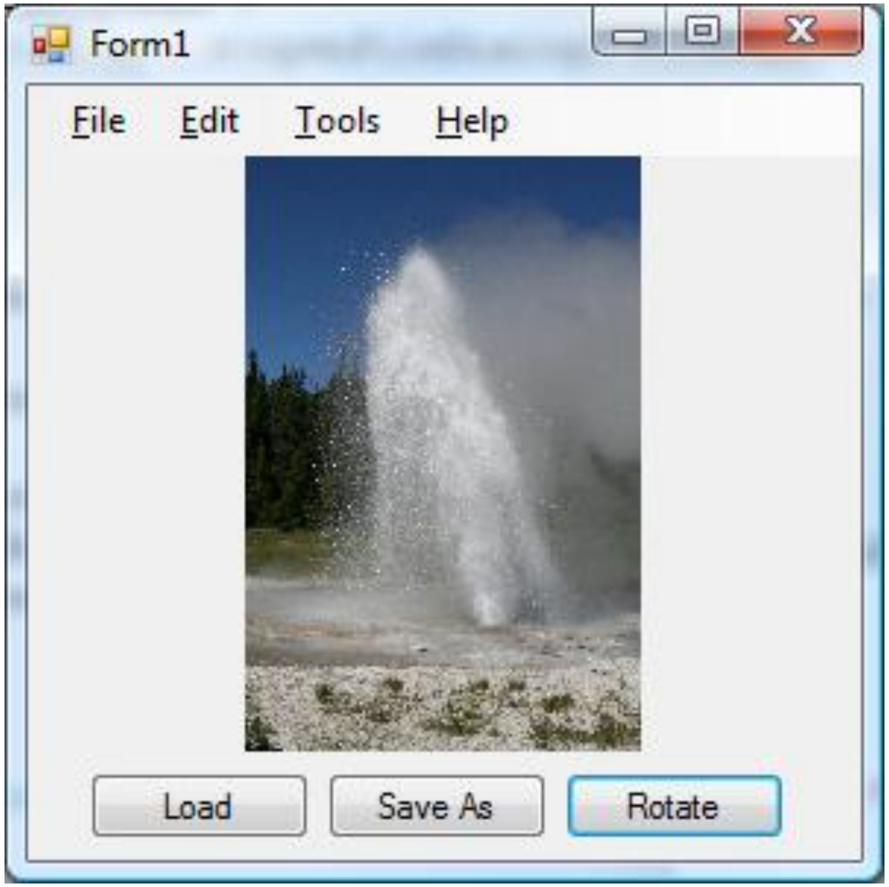
```
timeLeft = 0;
timer1.Enabled = false;
```

- Add this code to the Timer Tick event:

```
timeLeft -= 1;
if (timeLeft == 0)
{
    ovenDoor.BackColor = Color.Black;
    timer1.Enabled = false;
}
```

Dialog Controls

Display Image



Dialog Controls

- These controls pop up standard Windows dialog boxes
  - ColorDialog
  - FolderBrowserDialog
  - FontDialog
  - OpenFileDialog
  - SaveFileDialog

Open and Save a File

- In this demo, we’ll prompt the user for an image file, open it and display it. We’ll rotate it and then save it.

Demo: Open A File

- Create a new Windows Application project named DisplayImage.
- Add these controls:

Control	Property	Value
PictureBox	Name	pictureBox1
	SizeMode	Zoom
Button	Name	btnLoad
	Text	Load
Button	Name	btnSave
	Text	Save As
Button	Name	btnRotate
	Text	Rotate

Demo: Open A File

- Add an OpenFileDialog.

Control	Property	Value
OpenFileDialog	Name	openFileDialog1
	Filter	Image files|*.jpg; *.bmp
	InitialDirectory	Wherever the pictures are

- The default values for these will work:
  - CheckFileExists
  - CheckPathExists

Demo: Load Picture

- Double-click the Load button and add the code:

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

- Image.FromFile is a static method of the Image class.

Demo: Rotate Picture

- Double-click the Rotate button and add the code:

```
private void btnRotate_Click(object sender, EventArgs e)
{
    if (pictureBox1.Image != null)
    {
        Image picture = pictureBox1.Image;
        picture.RotateFlip(RotateFlipType.Rotate90FlipX);
        pictureBox1.Refresh();
    }
    else
    {
        MessageBox.Show("There is no image to rotate.");
    }
}
```

- Why the null check?

Demo: Save Picture

- Add a SaveFileDialog.

Control	Property	Value
SaveFileDialog	Name	saveFileDialog1
	CreatePrompt	true
	Filter	Image files|*.jpg; *.bmp
	InitialDirectory	Wherever the pictures are

- The default values for these will work:
  - AddExtension
  - CheckPathExists
  - CheckFileExists
  - OverwritePrompt

Demo: Save Picture

- Double-click the Save button and add the code:

```
private void btnSave_Click(object sender, EventArgs e)
{
    if (saveFileDialog1.ShowDialog() == DialogResult.OK)
    {
        pictureBox1.Image.Save(saveFileDialog1.FileName,
            System.Drawing.Imaging.ImageFormat.Jpeg);
    }
}
```

Menus and Toolbars

Menus and Toolbars

- These are:
  - ContextMenuStrip
  - MenuStrip
  - StatusStrip
  - ToolStrip
  - ToolStripContainer

Demo: Add MenuStrip

- Add a MenuStrip control to the form of the DisplayImage project.
- Rearrange the controls to fit.
- Click the SmartTag on the MenuStrip and select InsertStandardItems.

Add Code… Sort of

- Click the File/Open menu item. Do not double-click.
- Select the Events button in the Properties window.
- Select the Click event and choose the btnLoad_Click method.

Save As

- Just as you did for the File/Open menu, click the File/Save As menu item.
- Select the Events button in the Properties window.
- Select the Click event and choose the btnSave_Click method.

Exit the Application

- Double-click the File/Exit menu item, and add the code:

```
private void exitToolStripMenuItem_Click(object sender, EventArgs e)
{
    Application.Exit();
}
```

- Application.Exit
  - This ends the application and raises the FormClosing and FormClosed events on every form open in the application.

Rotate the Picture

- Add a new item to the menu, Edit/Rotate.
- Click the File/Rotate menu item.
- Select the Events button in the Properties window.
- Select the Click event and choose the btnRotate_Click method.

Other MenuStrip Settings

- Add hot keys to the menu items using &.
- Add shortcut keys with the ShortcutKeys and ShowShortcutKeys properties.
- Rearrange the menu items.
- Add checks.
- Show images.
- Add ToolTipText.
- Add TextBox and ComboBox controls to the menu.
- Use the SmartTag to access the Items Collection Editor.

Second Form

Adding a Second Form

- To add a second form to your application:
  - Create it in your project.
  - Design it.
  - In the “parent” form, create an instance of the form.
  - Call the ShowDialog method on the instance.

Add a Help Dialog

- Right-click the project in the Solution Explorer.
- Select Add / Windows Form.
- Name the new form HelpForm.
- Add a multiline TextBox control or a RichTextBox control and add some help text to the control. Dock it to Fill the form.

Display the Help Dialog

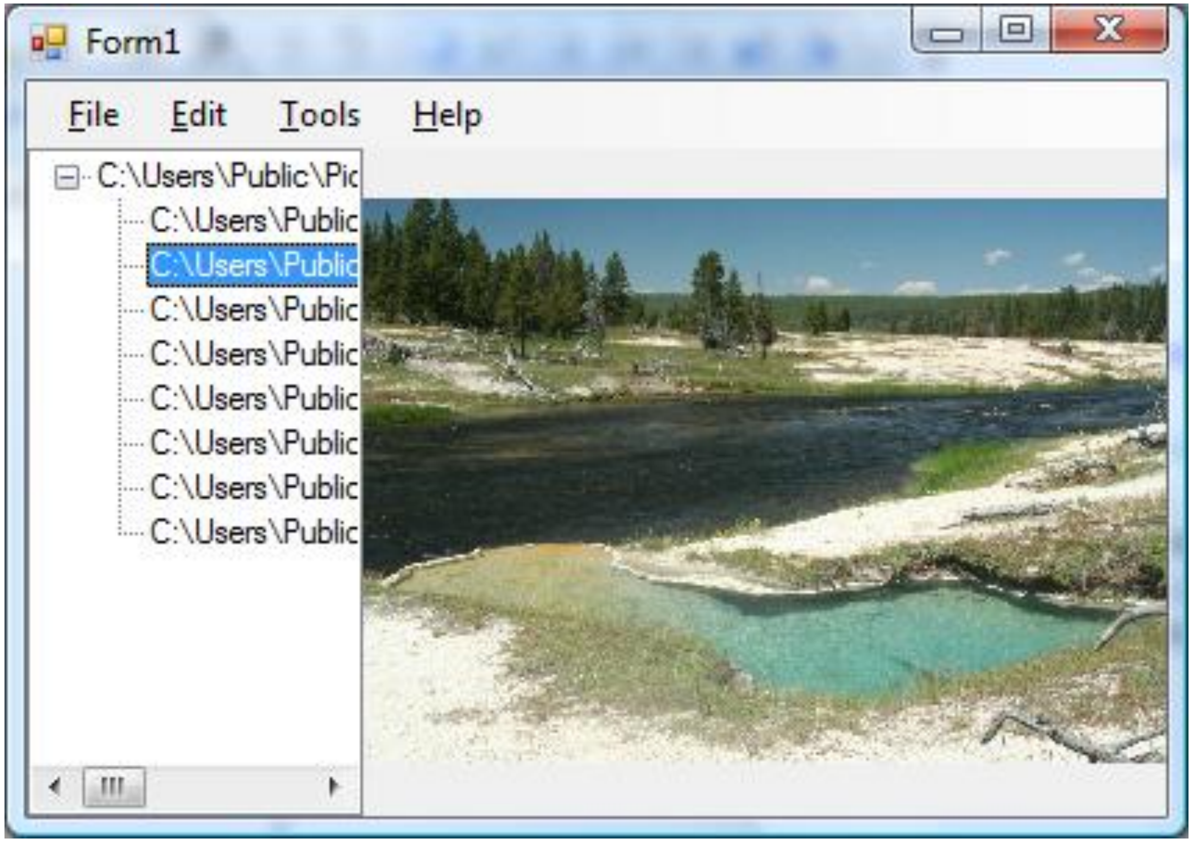
- Double-click the Help/Contents menu item and add the code:

```
private void contentsToolStripMenuItem_Click(object sender,
   EventArgs e)
{
    HelpForm helpForm = new HelpForm();
    helpForm.ShowDialog();
}
```

- This displays a *modal* dialog box. The user can only interact with that dialog.
  - Call Show() to get a non-modal dialog. Users can then access both forms.
  - Application.Exit closes both forms.

TreeView Control

TreeViewDemo



TreeView Control

- The TreeView is use to show hierarchical data, such as directories and files.
- We’ll use the TreeView and FolderBrowserDialog to display the images files.

TreeView Control

- Create a new project, TreeViewDemo. Don’t name the project TreeView.
- Add a TreeView control to the left side of the form.
- Set the Docking property so that it expands vertically.
- Click the SmartTag and experiment with the TreeNode Editor.
- Clear out the nodes, we’ll add new ones at runtime.

FolderBrowserDialog

- Add a FolderBrowserDialog.
- Add a MenuStrip and the standard items.
  - Add a new menu item, File/File &Location.
  - Double-click the menu item to generate the Click method.
- Add a PictureBox control.
  - Set the SizeMode to Zoom.
  - Dock the picture to Fill.

Getting the Files in the Folder

- Add the code to the Click method:

```
private void fileLocationToolStripMenuItem_Click(object sender,
   EventArgs e)
{
    if (folderBrowserDialog1.ShowDialog() == DialogResult.OK)
    {
        string path = folderBrowserDialog1.SelectedPath;
        TreeNode root = treeView1.Nodes.Add(path);
        string[] files = System.IO.Directory.GetFiles(path);
        foreach (string file in files)
        {
            root.Nodes.Add(file);
        }
    }
}
```

Displaying An Image

- We want to display an image if the file is selected in the TreeNode control.
- Double-click the TreeView.
- Notice the EventArgs, TreeViewEventArgs:
  - Action property – did the user click a node or expand/collapse?
  - Node property – which node did the user click?

Displaying an Image

- Add the code to the AfterSelect method:

```
private void treeView1_AfterSelect(object sender,
   TreeViewEventArgs e)
{
    if (e.Node.Text.EndsWith(".jpg"))
    {
        pictureBox1.Image = Image.FromFile(e.Node.Text);
    }
}
```

Editing the Image

- The Process component lets you start up a new process.
- Add a new menu item, Edit/&Image.
- Add a Process component to the form (component tray).
  - Name: process1
  - StartInfo.FileName: browse and find C:\Windows\System32\mspaint.exe.

Editing the Image

- Double-click the Edit/Image menu item and add the code:

```
private void imageToolStripMenuItem_Click(object sender, EventArgs e)
{
    if ((treeView1.SelectedNode != null)
    && (treeView1.SelectedNode.Text.EndsWith(".jpg")))
    {
        process1.StartInfo.Arguments =
            "\"" + treeView1.SelectedNode.Text + "\"";
        process1.Start();
    }
}
```

- You have to save with another file name, though, because the image file is locked as long as the image is displayed. (Code later.)

ContextMenuStrip

- The ContextMenu lets you add a small menu to a control on the form.
- We'll add a ContextMenu to the PictureBox to Edit the image.

ContextMenuStrip

- Add a ContextMenuStrip to the form.
  - It's added to the Component Tray.
- Add an Edit item.
- Set the ContextMenuStrip property of the pictureBox1 control to contextMenuStrip1.

ContextMenuStrip

- Double-click the Edit menu item and add code:

```
private void editToolStripMenuItem1_Click(object sender, EventArgs e)
{
    if ((treeView1.SelectedNode != null)
    && (treeView1.SelectedNode.Text.EndsWith(".jpg")))
    {
        process1.StartInfo.Arguments =
            "\"" + treeView1.SelectedNode.Text + "\"";
        process1.Start();
    }
}
```

Dispose of the Image

- When you switch to another image, remember to dispose any currently-displayed image.
- After a new image is selected, add this code to dispose of the previous Image instance:

```
if (pictureBox1.Image != null)
{
    Image toDispose = pictureBox1.Image;
    pictureBox1.Image = null;
    toDispose.Dispose();
}
```

Progress Bar & Timer

Progress Bar & Timer

- In this application, we’ll pretend to load an image from a file, and display a progress bar as we “load” the image.

Create the User Interface

- Add these controls and set the properties:

Control	Properties
Timer (Components tab)	Name: timer1 Enabled: false
ProgressBar	Name: progressBar1
PictureBox	Name: pictureBox1 BackColor: pick something Visible: false
Button	Name: loadPicture Text: Load

Load Event

- Add this code to the Click event and run:

```
private void loadPicture_Click(object sender,
                EventArgs e)
{
  pictureBox1.Visible = true;
}
```

- Now change it to this:

```
private void loadPicture_Click(object sender,
                EventArgs e)
{
  timer1.Enabled = true;
}
```

Timer Control

- The Timer control is a component and sits in the Component Tray. It does not appear on the form at runtime.
- The Timer raises a Tick event at intervals determined by the Interval property.
  - Set the Enabled property to true to start the timer.
  - Set the Interval to 1000 (1 second).
  - Set Enabled to false.

Timer Control

- Double-click the Timer control and add this code to the Tick event:

```
int seconds = 0;
private void timer1_Tick(object sender, EventArgs e)
{
    seconds++;
    progressBar1.Value = seconds;
    if (seconds == 10)
    {
        pictureBox1.Visible = true;
        timer1.Enabled = false;
        seconds = 0;
    }
}
```

Timer Control

- Hints for working with the Timer control:
  - Watch your counters carefully.
  - Don’t throw up messages boxes in each Tick event.
  - Controls may not update in the order you think.
  - Disable the Timer or the Tick event will keep happening.
  - Do not poll for time. That is, do not set up a loop and keep checking the time.
  - The timer is not exact.

Progress Bar

- The ProgressBar control displays a green bar as determined by the code you write.
  - Set Minimum to 0
  - Set Maximum to 10
  - Set the Step to 1
- We’ll update the ProgressBar every 1 second for 10 seconds, then display the image.

PictureBox Control

- Set up the PictureBox control
  - Set the Image property to an image on your PC.
  - Set the SizeMode to Zoom.
  - Set the Visible property to false.
    - We'll set it to true after 10 seconds.

Progress Bar

- With ProgessBar.Step=10, progress is a bit coarse.
- Make the progress smoother:
  - Set Step to 1.
  - Set Timer.Interval to 100.
  - Rename “seconds” variable to “secondsTenths”.
  - Quit when secondsTenths gets to 101.

Using Resources

Using Resources

- Resources are:
  - Images
  - Strings
  - Icons
  - Text files
- Rather than having to store and open files, you can store them as part of your application.

Add Images

- Download the card images from here (http://www.waste.org/~oxymoron/cards/) and unzip. (Or grab off the course website.)
- Double-click Properties of the project in the Solution Explorer.
- Click the Resources tab.
  - The Resources tab lets you add resources to your project.
  - They are automatically installed with your app.

Add Images

- Select Add Resource.
- Select Add Existing File.
- Select all the files and add them.
- Note the naming scheme:
  - _3d - three of diamonds
  - Kh – king of hearts
  - The Ace of Spades is a special case.

Access Images from the Designer

- Add a PictureBox control.
  - Size: 73, 97
- Set the Image property.
  - The images will appear in the Select Resource dialog box.

Access the Images from Code

- Add a button with text “Load King of Hearts”.
- Add code to the Click method:

```
private void button1_Click(object sender, EventArgs e)
{
    pictureBox1.Image =
        ResourceDemo.Properties.Resources.kh;
}
```

Access the Images from Code

- But you wouldn't want to write code like this:

```
switch (suit)
{
    case "hearts" :
        switch (value)
        {
            case "ace":
                pictureBox1.Image =
                ResourceDemo.Properties.Resources.ah;
                break;
            case "2":
                pictureBox1.Image =
                ResourceDemo.Properties.Resources._2h;
                break;
        }
        break;
}
```

Access the Images from Code

- Add two ComboBox controls:
  - suitList
    - Items: h, d, s, c
  - valueList
    - Items: a, _2, _3, _4, _5, _6, _7, _8, _9, t, j, q, k
- Add a button:
  - Name: loadCard
  - Text: Load Card

Access the Images from Code

- Add code to the Click event:

```
private void loadCard_Click(object sender, EventArgs e)
{
    if ((valueList.SelectedItem != null) &&
        (suitList.SelectedItem != null))
    {
        string resourceName =
            valueList.SelectedItem.ToString()
            + suitList.SelectedItem.ToString();
        pictureBox1.Image =
           (Image) ResourceDemo.Properties.Resources.
                ResourceManager.GetObject(resourceName);
    }
}
```

Access the Images from Code

- What about that special case? The name of the Ace of Spades includes an underscore.

```
if (suite == "s" && value == "a") value = "_a";
```

The ImageList Control

- Add an ImageList to the form
  - Drag from the toolbar
  - Select the Images property
  - Add the card images
- Add a button
  - Name: loadImageList
  - Text: Load Image List

The ImageList Control

- Create the handler for the imageList button.

```
private void loadImageList_Click(object sender, EventArgs e)
{
    if ((valueList.SelectedItem != null)
       && (suitList.SelectedItem != null))
    {
        string resourceName =
            valueList.SelectedItem.ToString()
            + suitList.SelectedItem.ToString() + ".gif";
        resourceName = resourceName.Replace("_", "");
        pictureBox1.Image = imageList1.Images[resourceName];
    }
}
```

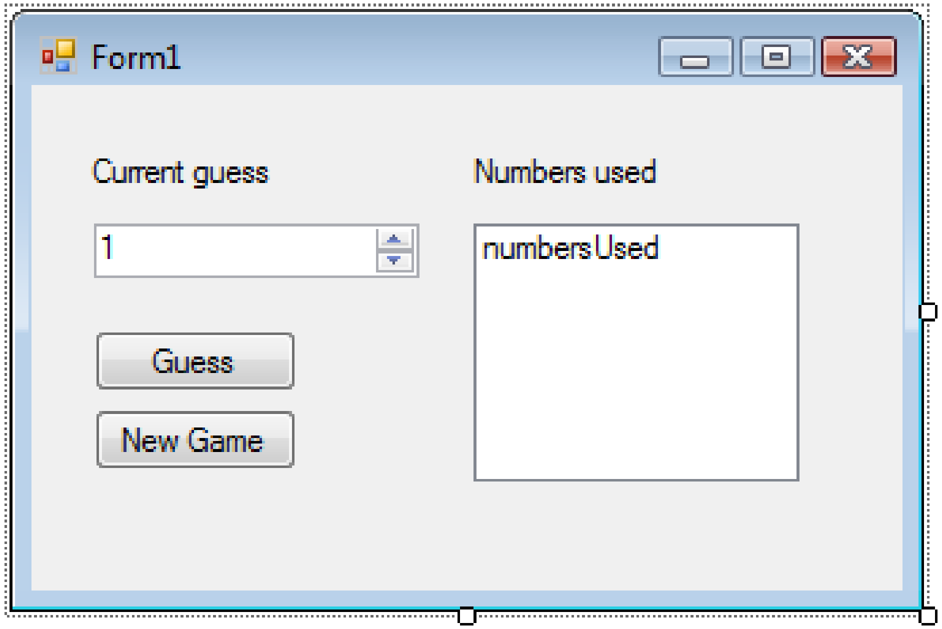
- The ImageList control holds images and allows you access them by an index value.

Guess The Number

Guess the Number

- Write an application program that plays "guess the number".
  - Your program should pick a random number between 0 and 10.
  - When the user enters a guess, your program should the user give a hint on whether the guess is too high or too low.
  - When the user finally guesses the correct answer, reward the user with "Congratulations!" or some other message.

User Interface



Create the User Interface

- Add these controls and set the properties:

Control	Properties
Label	Text: Current guess
Label	Text: Numbers used
NumericUpDown	Name: ctlGuessNumber Minimum: 1 Maximum: 10 Value: 1
ListBox	Name: ctlNumbersUsed
Button	Name: btnGuess Text: Guess
Button	Name: btnNewGame Text: New Game

Start a New Game

- Add this code to the Form1 class:

```
int secretNumber = 0;
Random rand = new Random();

private void StartNewGame()
{
    secretNumber = rand.Next(1, 11);
    ctlNumbersUsed.Items.Clear();
    ctlGuessNumber.Value = 1;
}
```

- But, how do we run that code when the application starts?

Load Event

- The Load event is raised when the form is about to be displayed.
  - Put your startup code here.
- To create the method, double-click the form in the designer.
- Add this code:

```
private void Form1_Load(object sender, EventArgs e)
{
    StartNewGame();
}
```

Add Code

- Add this code to the Click event of the New Game button:

```
private void btnNewGame_Click(object sender, EventArgs e)
{
    StartNewGame();
}
```

Add Code

- Add this code to the Click event of the Guess button:

```
private void btnGuess_Click(object sender, EventArgs e)
{
    int currentGuess = (int)ctlGuessNumber.Value;
    if (currentGuess == secretNumber)
    {
        MessageBox.Show("You Win!");
    }
    else if (currentGuess > secretNumber)
    {
        ctlNumbersUsed.Items.Add(currentGuess);
        MessageBox.Show("Your guess is too high.");
    }
    else // guess is too low
    {
        ctlNumbersUsed.Items.Add(currentGuess);
        MessageBox.Show("Your guess is too low.");
    }
}
```

Focus and Defaults

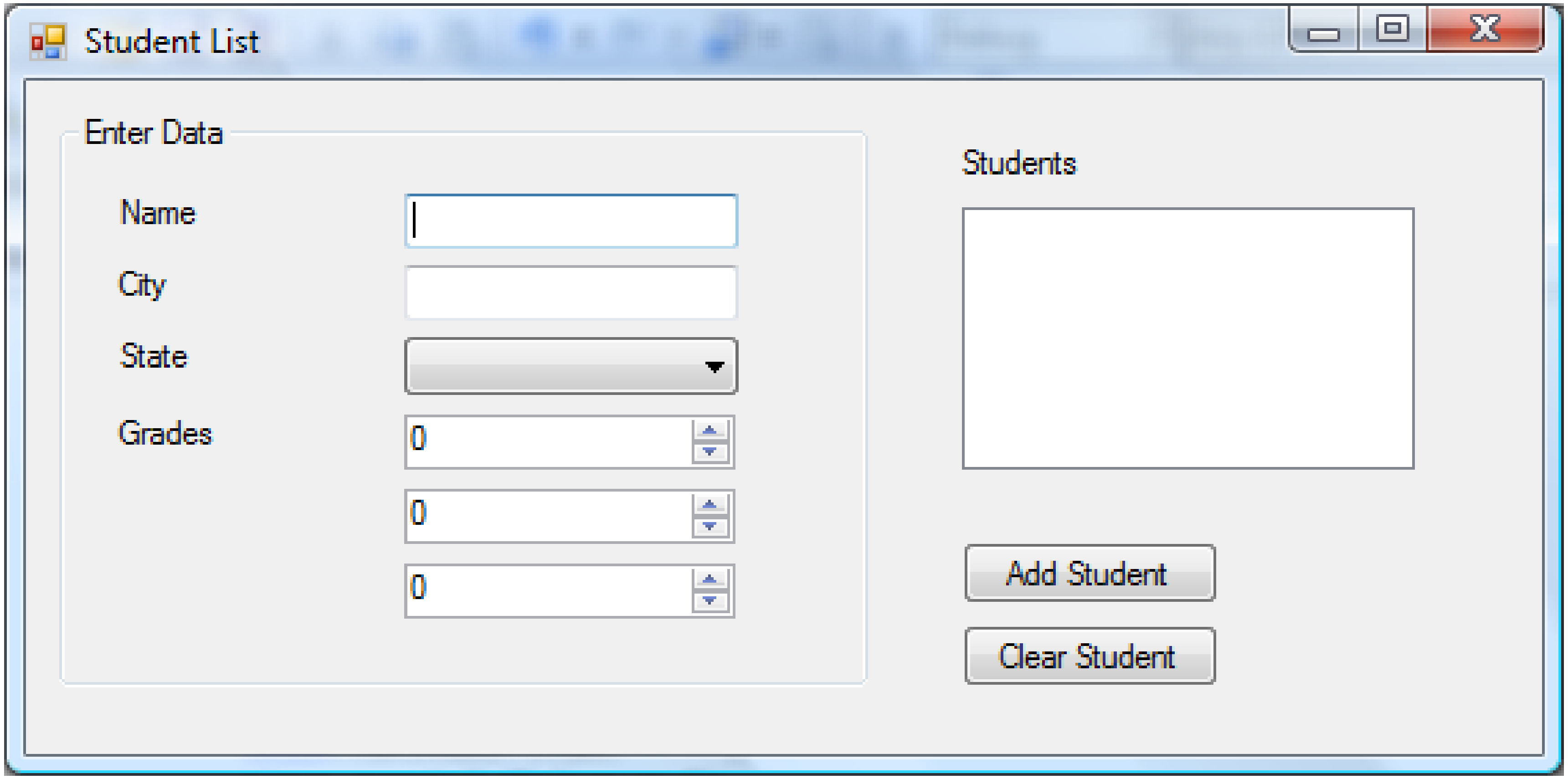
- Set the form’s AcceptButton property to be the Guess button.
  - After setting the NumericUpDown, “Enter” will invoke the Guess button.

Student Example

The Student Application

- In this application, we’ll use Windows Forms controls to help create, display, and update the data from Student class instances.

User Interface



Setup

- Create a new Windows Forms application.
- Add existing items from the shared drive.
  - Student.cs
    - ToString was changed to:

      ```
      return this.Name;
      ```
  - Address.cs
  - Both have the namespaces “fixed.”
- We’ll add a feature at a time.

Entering Students

- Add a GroupBox to the form. To the GroupBox, add:

Control	Name	Purpose
TextBox	txtName	Name
TextBox	txtCity	City
ComboBox	cboState	State

NumericUpDown (3)	`grade0`
	`grade1`
	`grade2`
		Grades

Labels		For Name, City, State, Grades

- Drop them ON the GroupBox.

ComboBox

- Set these properties:
  - DropDownStyle – DropDownList
    - This will enable autocomplete.
  - Items – Washington, Oregon, Idaho, Montana, Wyoming (use the SmartTag)

More controls

- Add these controls to the form:

Control	Properties
Button	Name: btnAddStudent Text: Add
Button	Name: btnClearStudent Text: Clear
ListBox	Name: ctlStudentList
Labels	As needed

ClearStudentData Method

- The ClearStudentData method resets all the controls in the GroupBox. Add this to the Form1 class.

```
private void ClearStudentData()
{
    this.txtName.Text = "";
    this.grade0.Value = 0;
    this.grade1.Value = 0;
    this.grade2.Value = 0;
    this.txtCity.Text = "";
    this.cboState.SelectedIndex = 0;
}
```

The Clear Button

- The Clear button calls the ClearStudentData method.

```
private void btnClearStudent_Click(object sender, EventArgs e)
{
    ClearStudentData();
}
```

The Add Method

- The Add button creates a new instance of Student and adds it to the Items collection of the ListBox.

```
private void btnAddStudent_Click(object sender, EventArgs e)
{
    Student newStudent = new Student(this.txtName.Text);
    newStudent.Grades[0] = (int)this.grade0.Value;
    newStudent.Grades[1] = (int)this.grade1.Value;
    newStudent.Grades[2] = (int)this.grade2.Value;
    newStudent.Address = new Address(txtCity.Text, cboState.Text);
    ctlStudentList.Items.Add(newStudent);
    ClearStudentData();
}
```

Items property

- Use the Items property of the ListBox to access the collection of items in the ListBox
  - It is possible to add or remove items from the collection, or to clear all items from the collection.
  - Items in the collection are Objects
    - Any type of Object can be stored in the collection
    - Might need to cast the object to the proper type when it is extracted from the collection

ListBox – SelectedIndexChanged

- When a student is selected in the ListBox control, the students' data is displayed in the GroupBox.

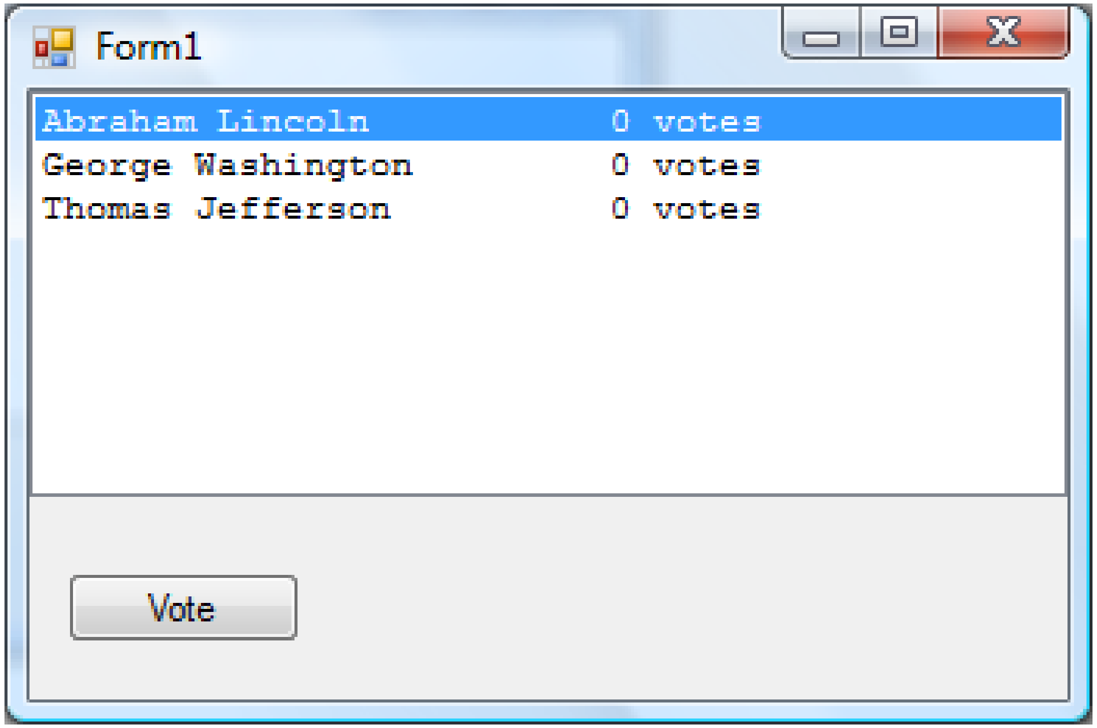
```
private void studentList_SelectedIndexChanged(object sender, EventArgs e)
{
    Student selectedStudent = (Student)ctlStudentList.SelectedItem;
    this.txtName.Text = selectedStudent.Name;
    this.grade0.Value = selectedStudent[0];
    this.grade1.Value = selectedStudent[1];
    this.grade2.Value = selectedStudent[2];
    this.txtCity.Text = selectedStudent.Address.City;
    this.cboState.Text = selectedStudent.Address.State.ToString();
}
```

Data Source

Election Example

- In this example, we'll implement the Voting application from assignment 2 using:
  - A Candidate class.
  - A ListBox with a data source.

User Interface



```
Abraham Lincoln            0 votes
George Washington          0 votes
Thomas Jefferson           0 votes
```

Controls

- Add these controls to the form:

Control	Properties
ListBox	Name: listBox1 Dock: Top FormattingEnabled: True
Button	Name: btnVote Text: Vote

Candidate Class

- Add a Candidate class and this code:

```
class Candidate
{
        public string Name { get; set; }

        private int m_votes = 0;
        public int Votes
        {
            get { return m_votes; }
        }

        public int AddVote()
        {
            m_votes++;
            return m_votes;
        }

        public override string ToString()
        {
            return string.Format("{0, -25} {1} votes", Name, Votes);
        }

}
```

Create A Data Source

- Add this code to the Form 1 class:

```
Candidate[] candidates = {
    new Candidate() { Name = "Abraham Lincoln" },
    new Candidate() { Name = "George Washington" },
    new Candidate() { Name = "Thomas Jefferson" }
};
```

- This code uses object initializers.
- We'll use the candidates array as a data source for the ListBox control.

Form Load

- Add this code to load the candidates into the ListBox control when the form loads:

```
private void Form1_Load(object sender, EventArgs e)
{
    listBox1.DataSource = candidates;
}
```

- Because the candidates variable is an array, the runtime can enumerate through the array and add each Candidate object to the Items collection of the ListBox.
  - Check it out in the debugger.

Vote Button Click

- Add this code to the btnVote Click event:

```
private void btnVote_Click(object sender, EventArgs e)
{
    Candidate selectedCandidate =
         (Candidate)listBox1.SelectedItem;
    selectedCandidate.AddVote();
    listBox1.DataSource = null;
    listBox1.DataSource = candidates;
}
```

Reading 7

- Deitel & Deitel
  - Chapter 14 – GUI with Windows Forms, Part 1
  - Chapter 15 – GUI with Windows Forms, Part 2
- MSDN: Windows Forms (http://msdn.microsoft.com/en-us/library/dd30h2yb.aspx)

Reading 7 (continued)

Topic	Section
Dialog controls	http://msdn.microsoft.com/en-us/library/6t3a1fcc.aspx
TreeView control	http://msdn.microsoft.com/en-us/library/ch6etkw4.aspx
Other tasks	http://msdn.microsoft.com/en-us/library/zftbwa2b.aspx

Assignment 7