1 **Windows Forms in C#**

CST242

2 **Visual C# Windows Forms Applications**

- A user interface that is designed for running Windows-based "Desktop" applications
  - *Not* on the Internet, *not* in a browser window
- A window with a Title bar, border, minimum and maximum buttons, and close button
- The Form becomes a running "Windows" dialog with which a user may interact, and which can be moved or resized

4 **Controls**

- A new "Windows Forms Application" starts with a single Form module file and the IDE presents a "Toolbox" with basic visual objects
  - These are controls that are drawn (dragged and dropped) onto the Form
- The controls include:
  - Button, CheckBox, ComboBox, Label, ListBox, MenuStrip and MenuItems, RadioButton, StatusStrip, TextBox, ToolStrip, etc.

5 **Starting a New Windows Forms Application (Page 1)**

- To create a new "Windows Forms" application when launching Visual Studio select "Create a new project"

6 **Starting a New Windows Forms Application (Page 2)**

- If already in Visual Studio:
  1. Click the File command on the menu bar
  2. Click New from the "File" drop-down menu
  3. Click Project... from the "New" submenu

7 **Starting a New Windows Forms Application (Page 3)**

- Alternately:
  1. Click the <New Project> button (if it exists) on the "Standard" toolbar
  2. Click New Project... from the "New Project" drop-down menu

8 **Starting a New Windows Forms Application (Page 4)**

- In the "Create a new project" dialog window:
  1. Select "C#" as the language
  2. Select "Windows" as the operating system
  3. Select "Desktop" as the environment
  4. Select "Windows Forms App (.NET Framework)" as the application type
  5. Click the <Next> button

9 **Starting a New Windows Forms Application (Page 5)**

- In the "Configure your new project" dialog window:
  1. Type the project "Project name" and select its "Location" (drive and folder where it will be saved)

    - Leave "Solution name" the same as the "Project name"
  2. Leave "Place solution and project in the same directory" checkbox unchecked
  3. Click the <Next> button

10 **Starting a New Windows Forms Application (Page 6)**

- In the "Additional information" dialog window:
  1. Leave ".NET 6.0 (Long-term support) selected and click <Create> button

11 **The Label Control (Page 1)**

- The Label control positions *unattached* text anywhere on the form
- Often placed near TextBoxes (or other controls) which do not have labels of their own
- The caption may be modified during design-time by changing the Text property
- User is *cannot type* into a Label during run-time (unlike a TextBox)

12 **The Label Control (Page 2)**

- The AutoSize property for a Label determines if object *increases* or *decreases* in size to fit the amount of Text entered
  - Values:
    - True—AutoSize is "on" (default for a Label)
    - False—AutoSize is "off"
  - A Label with *no Text* and its AutoSize property set to True can be hard to find on the Form

13 **Properties (Page 1)**

- Properties for many items (including controls on the Form) are updated within the "Properties" window
- Click on an item (e.g. a Label) to view its properties

14 **Properties (Page 2)**

- The Text property for a Label control modifies the text displayed for that item
  - Type a new value for property of a Label and text for the control is updated
- Many other control properties exist including:
  - Size (Width and Height) of an object
  - ForeColor (text color) and BackColor
  - (DataBindings) to "bind" a control to data from a database

15 **The Text Property (Page 1)**

- Determines what text is displayed in (or on or beside) an object, e.g. a Label or a TextBox
- Might be modified in "Properties" window, by the user at run-time, or in a code statement
- The Text property is type String
  - Numeric values assigned to the Text property of an object during run-time must be converted to String

16 **The Text Property (Page 2)**

- Run-time code format:

object.Text = *value/variable/formula*;

- Example:

  label1.Text = "My first C# Windows program";

17 **The TextBox Control (Page 1)**

- The TextBox control allows users to *input* text directly into the Windows Forms application
- Any characters can be entered, but user can be forced to entered numeric character only
- Derived from base class TextBoxBase which provides functionality including selecting text, cutting and pasting, etc.

18 **The TextBox Control (Page 2)**

- The Text property determines what text is displayed and can be modified during:
  - Design-time
    - Changing the value in the Properties window
  - Run-time
    - Values *typed* into the TextBox by a user
    - An assignment statement, e.g.

      textBoxKilometers.Text = (miles * 1.614).ToString();

19 **The TextBox Control (Page 3)**

- Other properties:
  - MaxLength: Maximum number of characters that may entered into TextBox
  - Multiline: If set to True, the Height of TextBox is increased and more than one line of text can be entered
  - ReadOnly: If set to True, the user may not key new values into the TextBox; at run-time code statements may assign new values to control

20 **The TextBox Control (Page 4)**

- Other properties (*con.*):
  - ScrollBars: For multi-line TextBoxes—None, Vertical, Horizontal (only if the WordWrap property is set to False) or Both
  - Text.Length—returns number of characters currently in the TextBox (including spaces)
  - WordWrap: For multi-line TextBoxes set to either True or False

21 **The TextBox Control (Page 5)**

- Methods:
  - Clear: erases the content of a TextBox
    - Format:

      textBoxName.Clear();
  - Focus: places the insertion point into the TextBox so that it has the focus
    - Most controls can receive the focus
    - Format:

      textBoxName.Focus();

22 **The TextBox Control (Page 6)**

- Events:
  - TextChanged: the TextBox's *default* event which fires every time one (or more) characters is typed or erased
  - KeyPress—fires each time keystroke is typed
  - KeyDown and KeyUp—fires when the either user presses or releases a key
  - Validating and Validated—fires when control is loosing focus (only if the CausesValidation property is set to True)

24 **The RadioButton Control (Page 1)**

- The RadioButton is an option control that can be turned on and off
- The Checked property is *Boolean* and represents the its value
  - True (it is on)
  - False (it is off)
- Text property is a label to the right of the RadioButton

25 **The RadioButton Control (Page 2)**

- Clicking *on* one RadioButton turns *off* any other that was on (*mutually exclusive*)
  - Cannot click RadioButton once to turn it "on" and then again to turn it "off"
- By default only one RadioButton within a *container* can be turned on at a time …
  - A GroupBox control is a standard container for grouping of RadioButtons

26 **The RadioButton Control (Page 3)**

- Run-time format:
  radioButtonName.Checked [= True/False]
- Examples:
  if (radioButtonMale.Checked)
  if (radioButtonMale.Checked = True)

27 **The RadioButton Control (Page 4)**

- CheckedChanged is the default event
  - It "fires" when the Checked property for a radio button changes (whether it is clicked or not) from True to False, or vice versa

28 **The GroupBox Control**

- The GroupBox is an object in the Container group of the "Toolbox" that provides for the grouping of controls
- Provides both *visual* and *functional* grouping
- When GroupBox is moved, all controls inside it move along with it
- The Text property represents text label on its upper-left border

30 **The CheckBox Control (Page 1)**

- The CheckBox control lets the user display a check mark ☑ in the box when selected (clicked)
- The Checked property is *Boolean* and represents CheckBox's value

- True—it is "on"
- False—it is "off"

- Text property is a label to the right of the CheckBox

31 **The CheckBox Control (Page 2)**

- CheckBox and RadioButton differences:
  - CheckBox can be clicked once and it is turned "on"; when clicked again it is turned "off"
  - More than one CheckBox may be "on" at one time within same container

32 **The CheckBox Control (Page 3)**

- Run-time format:
  checkBoxName.Checked [= True/False]
- Examples:
  if (checkBoxNonSmoker.Checked)
  if (checkBoxNonSmoker.Checked = True)

33 **The CheckBox Control (Page 4)**

- CheckedChanged is the default event
  - It "fires" when the Checked property for a check box changes (whether it is clicked or not) from True to False, or vice versa

35 **The ComboBox Control (Page 1)**

- The ComboBox is an input object with a list of Items (a collection) from which a user may *select* one or more
- It displays as a single text line and then when selected *drops down* to display the items (e.g. a "drop-down list")
- A scroll bar automatically is enabled if there are more items than will fit when the list drops down

36 **The ComboBox Control (Page 2)**

- Properties:
  - Items: A reference to the "Items" collection (the list of items displayed in the box) …
    - Clicking ellipse […] on the property line opens the "String Collection Editor" in which the item lines may be typed
    - Items also may be inserted using Add method (will see this later with the ListBox control)

37 **The ComboBox Control (Page 3)**

- Properties (*con.*):
  - SelectedIndex: Returns the index, starting at zero (0), of the item currently selected by the user; if no item is selected returns -1
  - SelectedItem: Returns of the currently item as an object; use the ToString method to convert that object to Text value, e.g.
    comboBoxObject.SelectedItem.ToString()

39 **The ListBox Control (Page 1)**

- The ListBox is an input object with a list of Items (a collection) from which a user may *select* one or more
- ListBox items displayed on lines in a region of a specific *fixed* size
- A scroll bar automatically is enabled if there are more items than will fit inside the control

40 **The ListBox Control (Page 2)**

- Properties:
  - Dock: Locks a control (such as a ListBox) to one or more of the edges of the Form window
    - The drop-down editor is displayed as a series of graphical boxes representing the edges and the center of the form
    - If the form is resized, the control automatically resizes to fit the boundaries of the docked edge
    - Values are Top, Bottom, Left, Right, Fill (fill entire Form) and None

41 **The ListBox Control (Page 3)**

- Properties:
  - Items: A reference to the Items collection (the list of items displayed in the box) …
    - Clicking the ellipse […] on the property line opens the "String Collection Editor" in which the item lines may be typed
    - Items also may be inserted using Add method (next slide)

42 **The ListBox Control (Page 4)**

- Methods:
  - Items.Add: *Inserts* an Item (a line of text) into the list
    - Format:
      listBoxName.Items.Add(*text_to_insert*);
    - Example:
      listBoxQuote.Items.Add(TextBoxName.Text);

43 **The ListBox Control (Page 5)**

- Methods (con.):
  - Items.Clear: *Deletes* all items from the list
    - Format:
      listBoxName.Items.Clear();
    - Example:
      listBoxQuote.Items.Clear();

45 **The Button Control**

- The Button is an event-oriented control *clicked* by user as needed to execute code operations
- Click is default event for a Button but it also can be programmed to respond to other

events:
  - E.g. DoubleClick, MouseDown, MouseUp, MouseHover ...
  - Determines what the user has done with the mouse and/or keyboard
- User decides *when* the event is executed
- Text property for Button control sets text displayed on the control

46 **Event-Driven Programming**

- The process whereby an application responds to *user actions* is called event handling
- *Double-click* on a Button to create an event handler method that responds to its Click event
- Example method header:
  private void buttonGetQuote_Click(object sender, EventArgs e)
  - Place all statements to be executed for the object and its event inside the {braces} of this method

48 **Solution Explorer**

- The "Solution Explorer" window located is in the upper-right corner of the IDE and is like "home" for Visual Studio developers
- In a tree view layout, it lists:
  - All projects
  - Filenames: source code, images, databases, etc.
  - Other resources and items that are part of the Visual Studio "solution"
- The window is quite sophisticated and it is likely that developers will not use all the power of the tool

50 **The ".Designer.cs" File (Page 1)**

- Drawing the Form at run-time, like all Visual C# operations, requires coded instructions
- This code resides in a file *associated* with the ".cs" file with the extension ".Designer.cs" which is *generated automatically* by creating the Form

51 **The ".Designer.cs" File (Page 2)**

- To find this file, in "Solution Explorer":
  - "Drill down" to the files associated with the Form by clicking the box with the arrow symbol (▷) that precedes the Form's filename
  - Double-click on ".Designer.cs" filename to open it in Source Code Editor

52 **The ".Designer.cs" File (Page 3)**

- The "Windows Form Designer generated code" is initially hidden within the editor
  - Click the box with plus (+) that is in front of the hidden code placeholder to expand the region that contains this code
  - Click the same box, now with a minus (-), to hide the code again

53 **Form Properties (Page 1)**

- FormBorderStyle:
  - None: No border so it is not sizeable; no Title bar so it is not moveable

- Fixed Single or Fixed3D: Only may be resized using Maximize and Minimize buttons on Control box at right end of Title bar (the MaximizeBox and MinimizeBox properties must be set to True)
- Sizeable (default): May be resized by dragging mouse on border

54 **Form Properties (Page 2)**

- FormBorderStyle (*con.*):
  - FixedToolWindow: Tool windows have a smaller text font size on title bar; not sizeable; has no Minimize nor Maximize buttons
  - SizeableToolWindow: Same as FixedToolWindow but resizeable

55 **Form Properties (Page 3)**

- ControlBox—on the right of "Title Bar"
  - It always contains a Close button; it also may display Maximize and Minimize buttons
  - Values are True (it is visible) or False (not visible)
  - MaximizeBox and MinimizeBox properties must be set to True for those buttons to be visible

56 **Form Properties (Page 4)**

- StartPosition—location displayed when the Form initially opens:
  - Manual: Determined by values of the Location property settings
  - CenterOwner: Center in *parent* Form
  - CenterScreen: On *entire* screen
  - WindowsDefaultLocation (default): Operating system determines best location based on the Win32 value known as CW_USEDEFAULT from the MS Windows "Registry"

57 **Form Properties (Page 5)**

- Location—window's absolute position on the screen (monitor)
  - Two sub-properties–X (Left) and Y (Top)
  - Only renders if the StartPosition property is set to Manual
- Size—of the form
  - Two sub-properties--Width and Height
- Text—text displayed on Form's "Title Bar"

58 **Form Properties (Page 6)**

- WindowState—its size when it first opens
  - Normal (default): Same size as *designed* (based upon the settings for the Size property)
  - Maximized: Full screen
  - Minimized: As an icon on the taskbar

60 **Events and Event Handlers**

- For almost every object on a Form, the Microsoft Windows® operating system can respond to many separate *mouse* and *keyboard* actions
- Some Visual C# Events are Click, MouseDown, MouseMove, MouseOver, KeyPress,

Validating, Validated, etc. (there are dozens)

61 **Syntax of Event Handler Header (Page 1)**

- Format:

  private void *methodName*(object sender, *EventClassType* e)

  - The *methodName* by default consists of the concatenation of the *object name* and *event type*

62 **Syntax of Event Handler Header (Page 2)**

- Format (*con*):

  private void *methodName*(object sender, *EventClassType* e)

  - Control procedures have two parameters:
    - *sender*—a reference which identifies the *object* (control) which initiated the call to the method
    - *e*—object variable by which all arguments for an *event* are passed to the called method; the *EventClassType* varies based upon type of event that executed

63 **Syntax of Event Handler Header (Page 3)**

- Format:

  private void *methodName*(object sender, *EventClassType* e)

- Examples:

  private void Form1_Load(object sender, EventArgs e)

  private void buttonGetQuote_Click(object sender, EventArgs e)

64 **Form Events**

- The Load event is the default event for a Form
  - Once the Form is loaded into memory but before it becomes visible
  - *Double-click* on any blank area of the Form to create its Load event handler method
- The Form also responds to events common to other controls such as Click and DoubleClick

65 **The MessageBox.Show Method**

- Displays a message in a separate dialog window
  - Click the <OK> button to close the window
- Format:

  MessageBox.Show("*Display String*", "*Title Bar String*");

- Example:

  MessageBox.Show("My very first C# Windows program", "Let's Go");

67 **C# Comment Style (Page 1)**

- Comments having a special form that are used to direct third-party documentation generators to produce XML from those comments and the source code
  - This documentation is formatted and designed to be readable in a browser
- All comments must immediately precede a user-defined type, e.g. a class, a method, a field, an event, a property, etc.

68 **C# Comment Style (Page 2)**

- C# specific comments are in one of two formats:
  - Single-line comments start with three slashes (///) :
    /// *comment*
  - Delimited (multi-line) comments start with a slash and two stars (/**) and end with a star and slash (*/):
    /**
    * *comment*
    * (*etc.*)
    */

69 **C# Comment Style (Page 3)**

- There are a set of commonly used (recommended) *tags* that are built into C# including:
  <summary>
  <param>
  <returns>
  <exception>
- The documentation generators can accept and process any tag that follows valid XML rules
  - However other programmer-defined tags are possible

70 **C# Comment Style (Page 4)**

- The <summary> tag can be used to describe the type (class, method, etc.) itself
- Example:

```
/**
 * <summary>
 * Calculates an insurance quote from the 'Name' and
 * 'Age' TextBoxes, 'Female' and 'Male' RadioButtons,
 * 'NonSmoker' CheckBox and 'Region' ComboBox
 * </summary>
 */
```

71 **C# Comment Style (Page 5)**

- The <param> tag is used to describe a parameter for a method
- It should include a name attribute to "name" the parameter and a *description* inside the tags
- Example:

```
/**
 * ...
 * <param name="age">The age of the person for the quote</param>
 */
```

72 **C# Comment Style (Page 6)**

- The <returns> tag is used to describe the return value of a method
- Example:

```
/**
 * ...
 * <returns>The portion of the insurance quote based on age</returns>
 */
```

73 **C# Comment Style (Page 7)**

- The <exception> tag provides a way to document the exceptions a method can throw
- It should include a cref attribute to "name" the Exception type and a *description* inside the tags
- Example:

```
/**
 * ...
 * <exception cref="ArgumentException">If age parameter is negative</exception>
 */
```

75 **Declaring Constants**

- A *constant* is an identifier assigned a value that cannot be changed
- Keyword const is used to declare the constant
- Format:

  const *type CONSTANT_NAME = value*;
  - The C# standard *naming convention* is the same as Java, all uppercase letters and underscores
- Example:

```
const int BASE_RATE = 250;
```

76 **The Convert Class**

- The static methods of the Convert class are designed to convert data of one type to another
  - There are several of these methods
- Format:

  Convert.To*ChangeType*(*value*)
  - *value* can be almost any type
  - *ChangeType* is type being converted to (the return type)
- Examples:

```
int age = Convert.ToInt32(TextBoxAge.Text);
```

78 **"Windows Forms" File Structure**

- Three levels of files:
  - *Solution* file (.sln)
    - A single solution can consist of *several projects* including Visual C#, Visual Basic, Visual C++, etc.
  - C# *project* files (.csproj):

- Information about each module (including filename and relative locations/path) that make up each project
- *Module* files (e.g. the *code* and the Form) (.cs & .Designer.cs):
  - Information about the modules in the project

80 **The Solution (.sln) File**

- Information in the Solution file includes:
  - References to the path and filename of the specific "projects" that make up the solution
  - Details regarding which configuration files that are used to compile the entire application
- A new "Solution" is created whenever a new "Project" is started
  - Multiple projects of different types can exist within the solution, e.g. Visual C#, Visual Basic, Visual C++, etc.

81 **The Project (.csproj) Files (Page 1)**

- *Each* Project file within the solution contains:
  - *Which type of project* this is, e.g.:
    - Windows Forms application, Console application, Web Form application
  - References to path and filename of each module that make up project:
    - Form files, code files, and other modules
  - Lists configuration files unique to the *compilation* of each project
  - References to the project's namespaces and assemblies

82 **The Project (.csproj) Files (Page 2)**

- *Some* of the several types of C# projects:
  - Windows Forms Application:
    - Project that executes inside a Microsoft Windows® Form on the desktop
  - Console Application
    - Project that runs in a command (console/terminal) window
  - ASP.NET Web Form Application (Active Server Pages)
    - Project that runs in a web browser and often interacts with a database to provide *dynamic* web content