158.212 Application Software Development

Suriadi Suriadi

Lecture 5

- Object Constructors
- Code Refactoring
- Visual Studio IDE
  - Control visibility
  - Access Keys
  - ToolTips
  - Tab Order
  - Containers

Object Oriented Programming

- A Class defines the template or blueprint of an Object.
  - Encapsulation – the grouping of data and related methods into one class
  - Access modifier – private, public, protected
  - Shared/static modifier
- Instances of a Class can be instantiated to create an object
- A **constructor** is needed in the instantiation of an object
- Constructors set up an instance when it is created.

OO - Constructors

Some variables can be created and initialised trivially as:

```
Dim i As Integer = 0
```

```
int i = 0;
```

The above are **value-type** variables or **primitives**.

However, some variables require the explicit use of a **constructor**:

```
Dim t As Timer = New Timer()
```

```
Timer t = new Timer();
```

The above are **reference-type** variables and are instantiations of class types.

Primitives vs. Reference-type

OO - Constructors

- Constructors can take parameters that are used to initialise the instance
    - May be optional – depends on how the constructor is coded.

- Why constructors?
    - Allow the created object to be properly created with correct values, including those private data members.
    - It helps to set the correct baseline properties and behaviours of objects

```
Dim t As Timer = New Timer(1000)
```

```
Timer t = new Timer(1000);
```

OO - Constructors

- Later in the course we will discuss creating your own Classes and constructors.

- For the moment you just need to know how to use them.

“Anyone can write code, but it takes a skilled craftsman
to write code that is easy to maintain
and can be extended over time”

Robert C. Martin

The reality about code…

- Code is hardly ever “right” the first time its written
- Code design is a continuous process
    - Software must evolve with experience, new knowledge gained, and evolving requirements
- Code decay
    - Multiple teams of programmers
    - Different coding standards
    - Code hacking in rush to meet deadlines
    - Incomplete understanding of the problem domain
        - Unclear requirements (both parties – users and developers)

Refactoring

*"Refactoring is a process of changing a software system in such a way that it does not alter the external behaviour of the code, yet **improves** the **internal structure**"*

Martin Fowler

Refactoring

- Refactoring modifies the non-functional aspects of the software.
- Triggered by “code smells”.
- Carried out iteratively using standardised “micro-refactorings” that result in small increments of change to the source code.
- Standard industry practice where software is critical.

Benefits of refactoring…

- Reduction in complexity
- Software architecture that models the business
- Software lifetime
- Developer productivity

What can be refactored…

1. Renaming: variables, methods, classes and packages
2. Componentization
3. Extract methods
4. Extract class

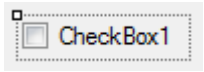
Visual Studio provides Refactoring Tools as well as Third Party plugins.

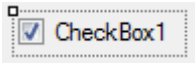
How to refactor code…

- Done in small steps, usually interleaved with bug fixes and feature additions
- Easier to rearrange code if functionality unaltered
    - Interface remains
- Further, it is easier to alter functionality when the code has been refactored
    - Modularized code
- ***Design Patterns*** are ultimately the target of refactoring.

CheckBox

- CheckBox allows the user to select an option or not.
- The vital property of a CheckBox is called **Checked**.
- The property is True if the box has been ticked or False otherwise.
- The **Checked** property of a checkBox1 can be accessed or changed in the code using:






```
Dim isChecked As Boolean
isChecked = checkBox1.Checked
or
checkBox1.Checked = True
```

CheckBox

- The default event (the event that gets added by the designer when you double-click on a control) for a CheckBox is **CheckedChanged**.

- Adding this event handler/listener will create a subroutine that will be called every time the CheckBox is either checked or unchecked.

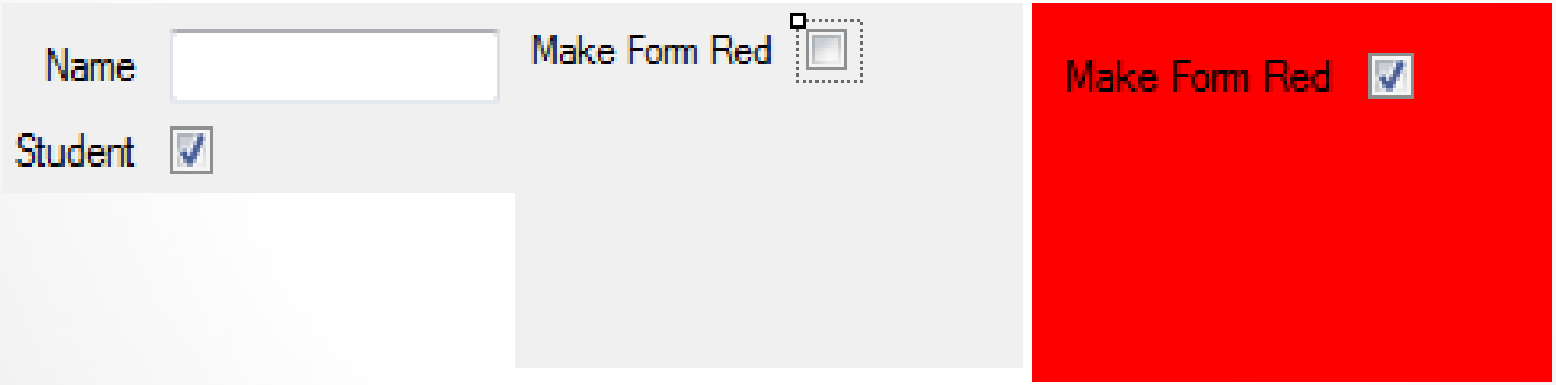
```
Private Sub CheckBox1_CheckedChanged(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles CheckBox1.CheckedChanged

End Sub
```

CheckBox

The two main methods of using a CheckBox are:

1. Selecting an option in an input form as true/false that is read when the input is submitted.
2. Turning an option on/off that has an immediate effect when the CheckBox is changed.



RadioButton

- RadioButton allows the user to select ***one*** option out of ***several options***.
- RadioButtons *must belong to a group*.
- Within a *group*, only one RadioButton may be checked at any point in time.
- RadioButton has a Checked Property that determines if it is selected or not.

```
Dim isChecked As Boolean
isChecked = radioButton1.Checked
or
radioButton1.Checked = True
```

RadioButton

- The default event for a RadioButton is **CheckedChanged**.

- This event will be called every time the RadioButton is selected or unselected.

```
Private Sub RadioButton2_CheckedChanged(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles RadioButton2.CheckedChanged

End Sub
```

RadioButton

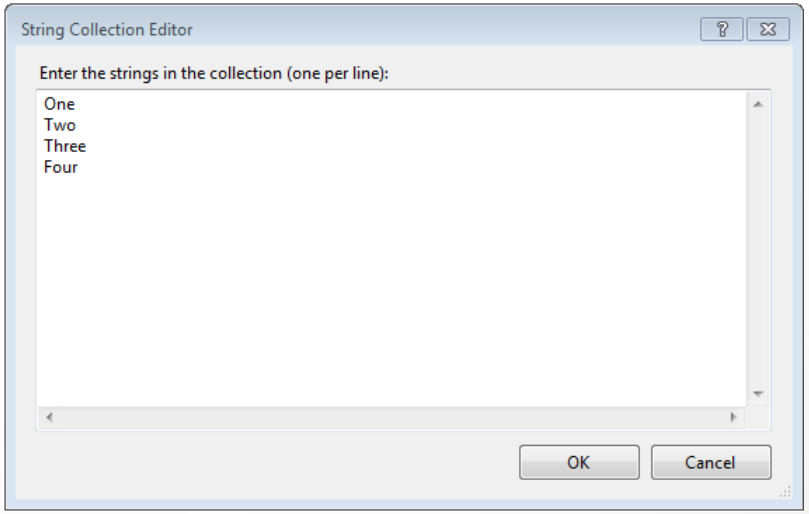
Like the CheckBox, a group of RadioButtons has two main uses:

1. Selecting one of several options on an input form that is read when the form is submitted.
2. Selecting one of several options that has an immediate effect.

ComboBox

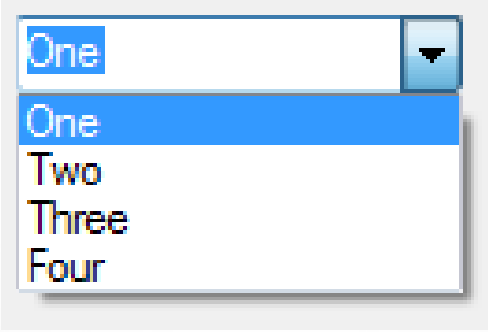
- ComboBox allows the user to select one of several options from a drop-down list.
- ComboBox has a property called **Items** which is a list of the options the user can select.
- The simplest way to set this property is to change the values in **Items** in the properties window. For example:

ItemHeight	13
Items	**(Collection)**
Location	**66, 167**
Locked	False
Margin	3, 3, 3, 3
MaxDropDownItems	8



ComboBox

- In the code, the user-selected item can be accessed through the property **SelectedItem**.
- When you add Items to a ComboBox in the properties window, each option is a String.
- However, Items can hold a list of any type of objects and simply calls the **ToString()** function of each option to display the list.



ComboBox

- The default event for a ComboBox is the **SelectedIndexChanged**.

- The **SelectedIndexChanged** event will be called every time the user selects a different item from the ComboBox.

```
Private Sub ComboBox1_SelectedIndexChanged(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles ComboBox1.SelectedIndexChanged

End Sub
```

TextBox

- TextBox can be used to read input from a user (mainly), though can also be used to output text
- The default event for this Control is **TextChanged**.
- **TextChanged** event will be called every time the text inside the TextBox is modified.

MaskedTextBox

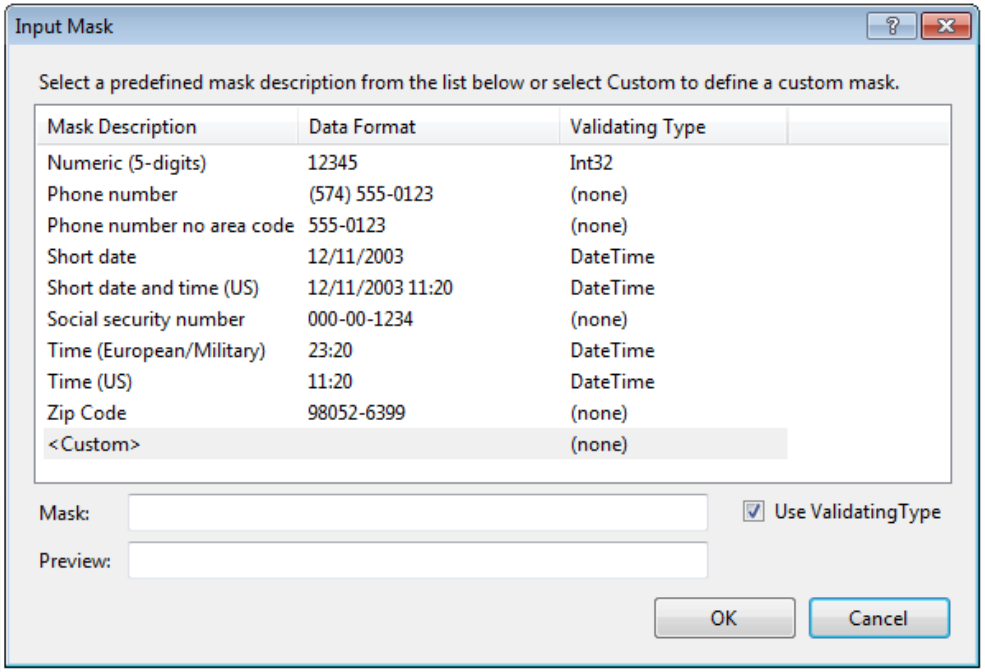
- MaskedTextBox is similar to a TextBox.
- MaskedTextBox can be used to allow the user to enter input.
- However, a MaskedTextBox can be configured ***to restrict the input*** from a user to a specific format. The allowed input format can be controlled through the Mask property.

Locked	False
Margin	3, 3, 3, 3
Mask	
MaximumSize	0, 0
MinimumSize	0, 0

MaskedTextBox

A dialog window is used to set a pre-defined format or to create a custom format for the input restriction.

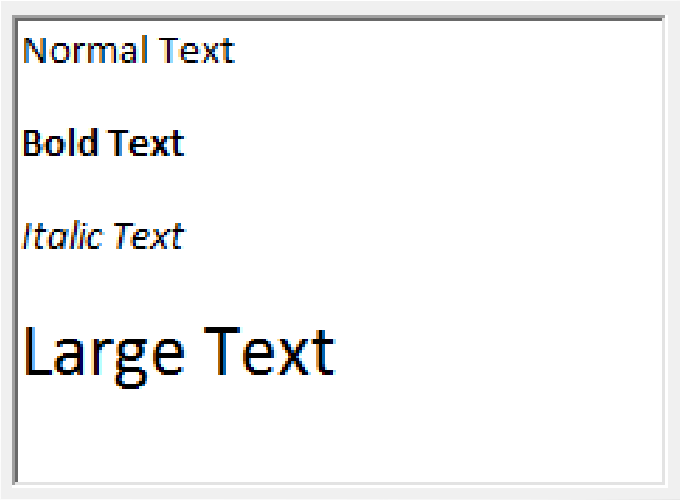
Only input that matches this format can be entered into the MaskedTextBox.



RichTextBox

- RichTextBox is similar to a TextBox but it allows the content text to be formatted.

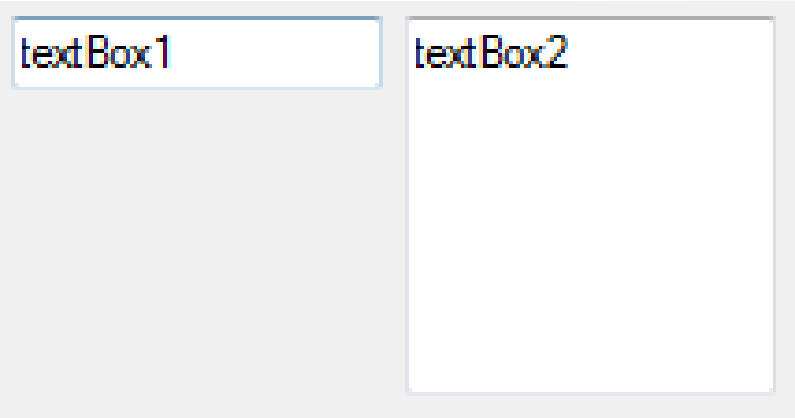
- In a TextBox, all the text is plain and uniform. A RichTextBox allows the user to apply formatting to different characters, words and paragraphs. This is similar to the main input space of a word processor.



Multiple Lines

- All text boxes (TextBox, MaskedTextBox and RichTextBox) can be configured to allow input in multiple lines.

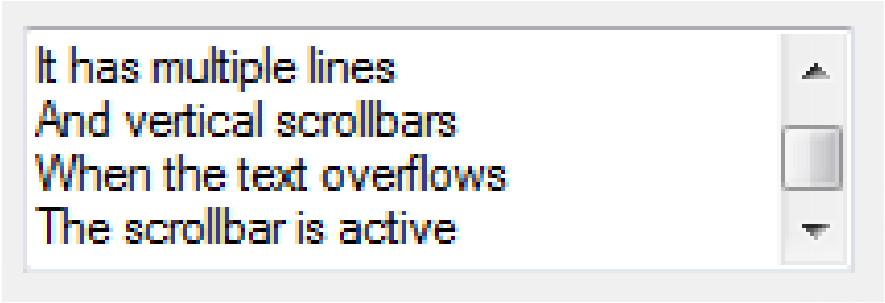
- Changing the **Multiline** property to true enables the user to enter multiple lines of text and will automatically wrap text onto the next line.



ScrollBars

- ScrollBars can be added to areas such as TextBoxes by changing the ScrollBars Property.

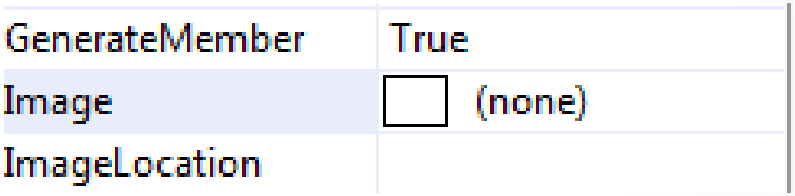
- ScrollBars can be set to Horizontal, Vertical or Both.



PictureBox

- A PictureBox allows an image to be displayed on the form.

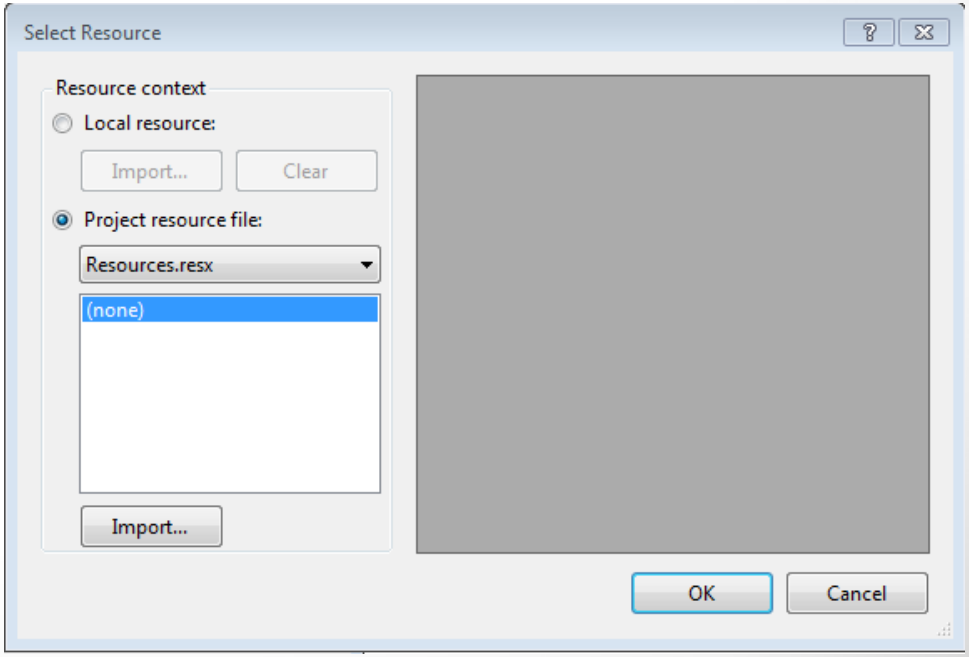
- The PictureBox will initially be blank, but the **Image** property can be set to a specific image.



PictureBox

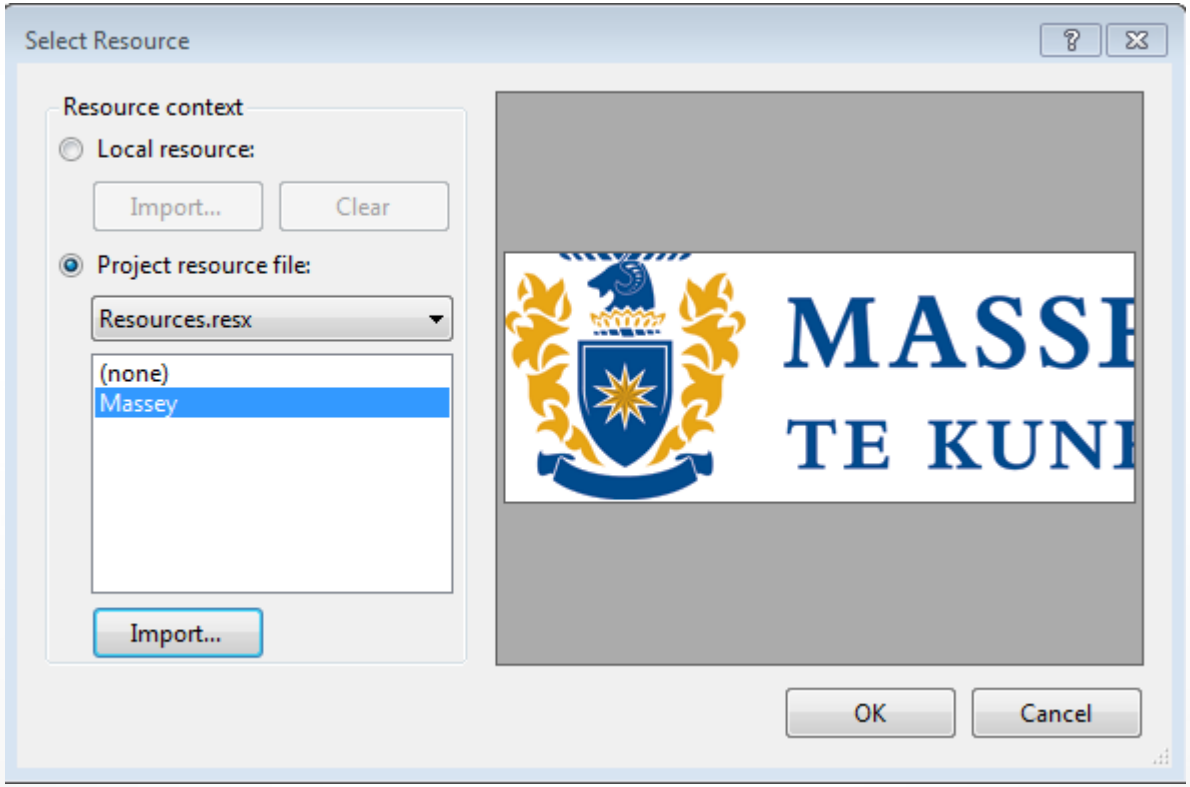
- By selecting a PictureBox and the **Image** property, Visual Studio will prompt for a **Resource**.

- By selecting **Import**, VS will allow the selection of an image to import into the project.



PictureBox

Following this, one can then select a resource for the PictureBox to display.



PictureBox

Resources added to the Project can be accessed in code in:

```
Properties.Resources.(filename)
```

PictureBox

The PictureBox will display the image. But depending on the **SizeMode** Property, the PictureBox will display the image in different ways.

Normal

CenterImage

PictureBox

StretchImage

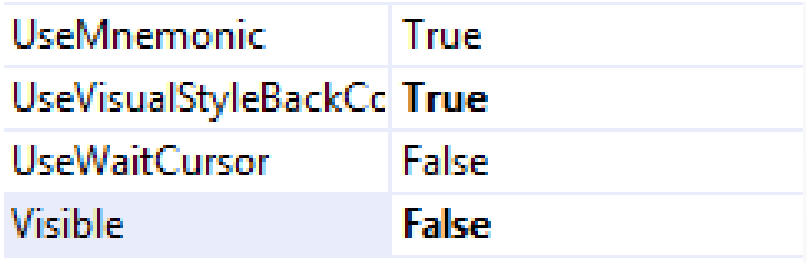
Zoom

AutoSize

Visibility

- Controls that are added to a form are set to be visible by default. However, sometimes it is useful to hide controls and only make them visible later.

- Controls can be made invisible by setting the **Visible** property to false. This can be done either in the Properties Window or in the code.

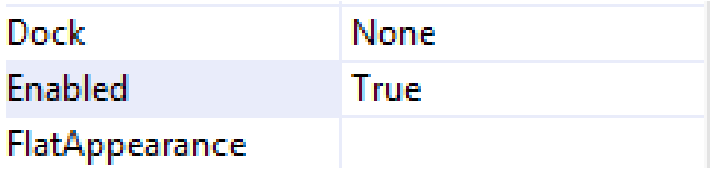


Visibility

Controlling whether a control is visible or enabled is useful for user input forms which require different input depending on the options selected by the user.

Enabled

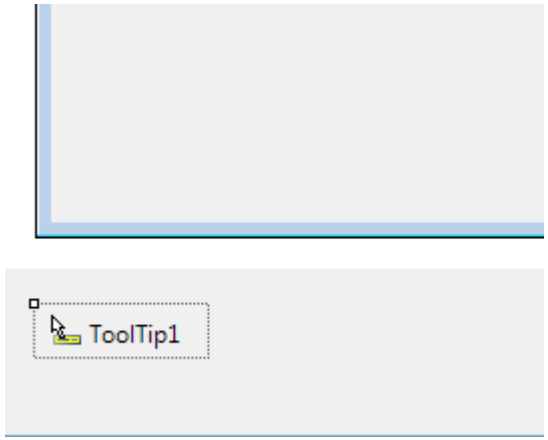
- Controls can also be made unusable by disabling them.
- When a control is disabled it is still visible but cannot be used.
- This can be done by setting the **Enabled** property to false.



ToolTips

- ToolTip is a short description of a control that appears when the user holds the mouse over a control.

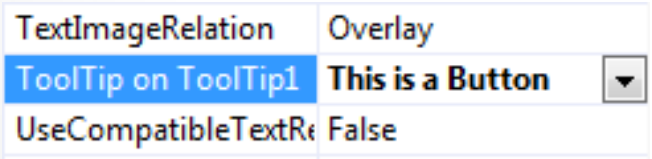
- To add tool tips to controls, first add a ToolTip Control to the form. This will appear as a non-displayed Control in the Form designer.

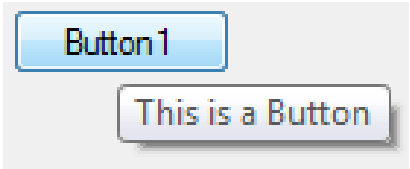


ToolTips

- The Controls will now have a ToolTip Property that can be set to a desired message.

- This message will appear if the user hovers the mouse over the control.



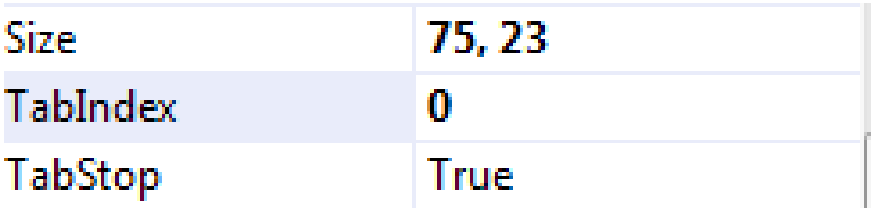


TabIndex

- TabIndex allows the users to cycle through the controls on the form by pressing the Tab key. This is especially useful for user input forms.
- Each control on the form has a TabIndex automatically added based on the order the controls were added to the form.
- The default tab order can be manually altered.
- The **TabIndex starts at 0** and moves up through the controls.
- If the TabStop Property of a control is set to False, that control will not be ‘tabbed to’.

TabIndex

These **TabIndex** properties can be set in the Properties Window.



Access Keys

- Access Key allows the user to select a specific action by using a particular hotkey.

- When the user presses **Alt + O**, the form will look for a key associated with the Access Key O. A control can be associated with an Access Key by putting an ampersand (&) in the Text field. For example:

Tag	
Text	**E&xit**
TextAlign	MiddleCenter

Control Containers

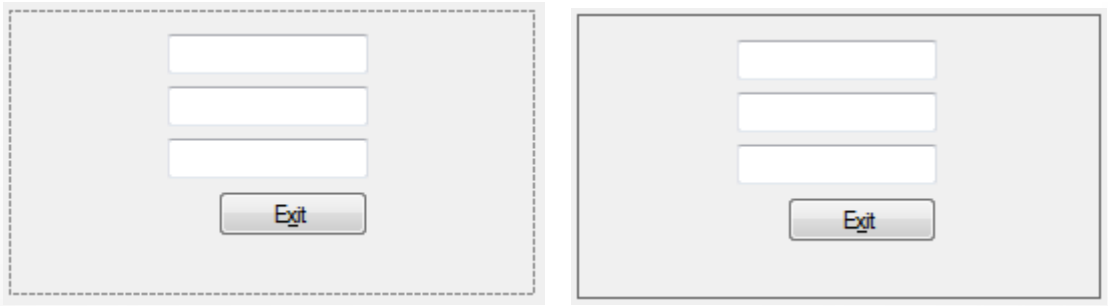
- Control Containers are very useful for grouping other Controls into logical units.

- We will look at four of these containers
  - Panel
  - GroupBox
  - TabControl
  - FlowLayoutPanel

Panel

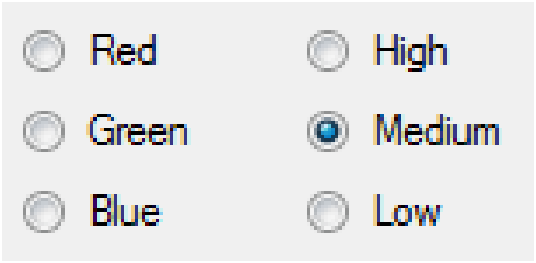
- Panel is a simple area that can be added to a Form or another Panel.

- They can be used simply make an application look better, but more importantly can be used to group Controls. They can be invisible and simply used to move controls together or can be visible by changing the **BorderStyle** Property.



GroupBox

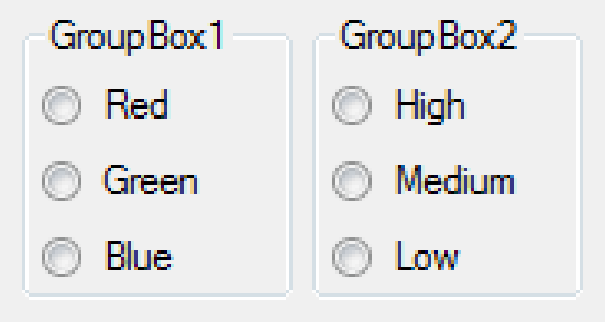
- GroupBox is used as a container for other controls.
- These are particularly important for RadioButtons.
- If multiple RadioButtons are added to a Form, they will all be in same group and **only one** can be checked at one time.



GroupBox

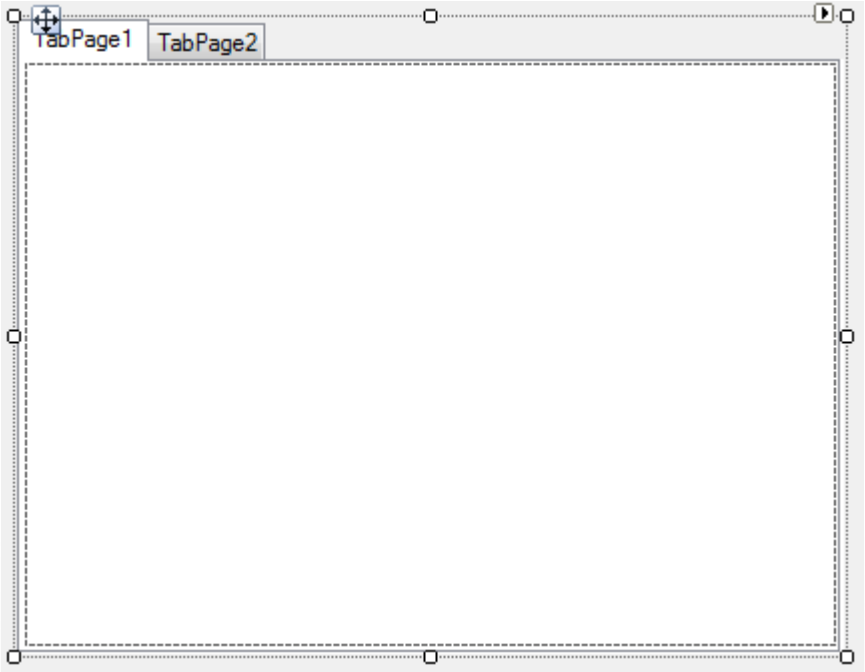
- To allow different groups of RadioButtons, the RadioButtons must be added to different GroupBoxes.

- These GroupBoxes display text in the GroupBox or left blank.



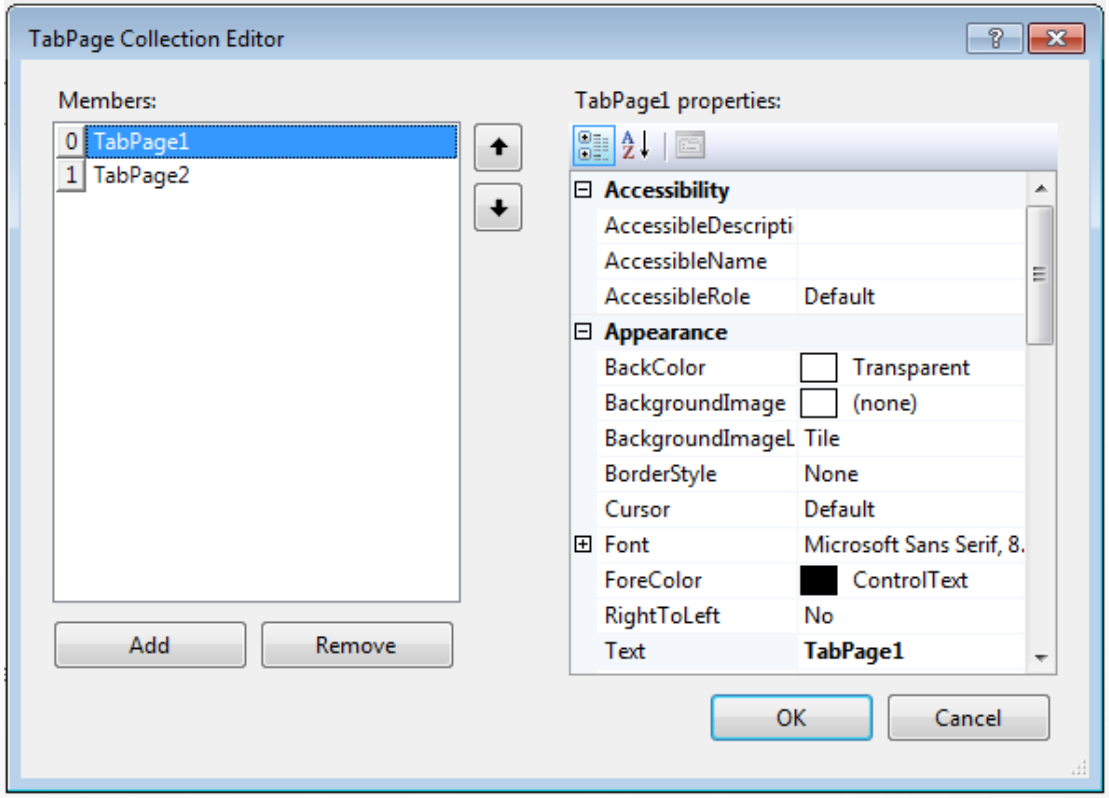
TabControl

TabControl is similar to a set of panels that the user can move between, this is particularly useful for switching between ‘views’ in an application without having to change Forms.



TabControl

Additional TabPages can be added to a TabControl by selecting the **TabPages** property from the TabControl.



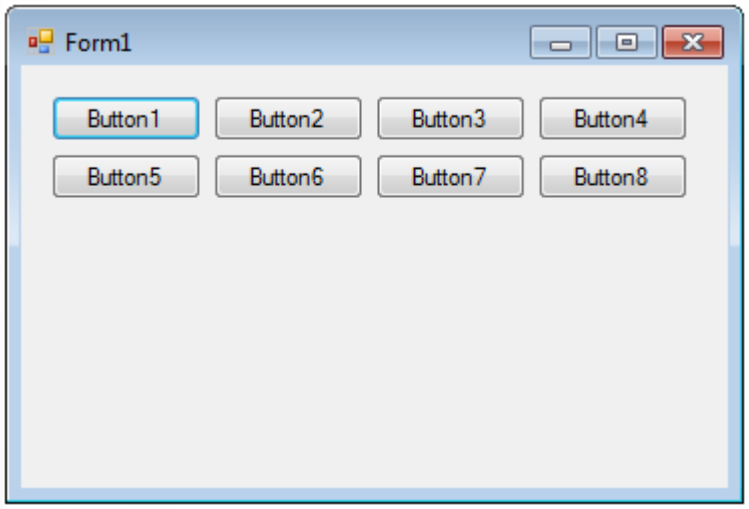
TabControl

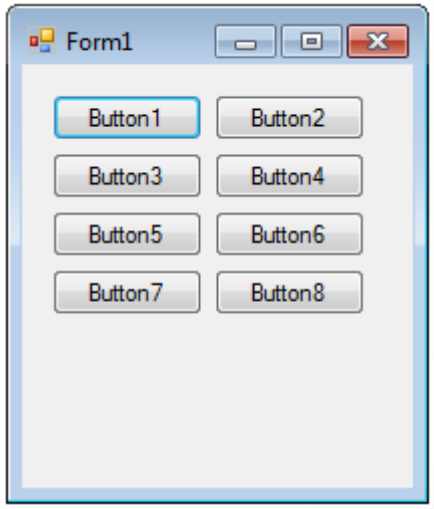
- Selecting a different tab will call the default event for a TabControl - **SelectedIndexChanged**.

- The control will automatically switch tabs but this event can be used to control additional behaviour if needed.

FlowLayoutPanel

FlowLayoutPanel will automatically rearrange the controls to fit in the panel. As the Form is resized the controls will be moved to fit in the panel.





Summary

Constructors allocate memory to an object reference and set values to its properties.

Code refactoring is a crucial component in the maintainability of code.

Code refactoring modifies the code without altering the functionality.

Windows forms: Visibility, Access Keys, ToolTips, Tab Order, Containers