CMPE 1666- Intermediate Programming

Lecture 9-

Delegates And Modeless Dialogs

Acknowledgements J.D Silver

Introduction To Delegates

- Note: The earlier Slides focus on syntax, not contextual use of delegates!
- The later slides discuss Modeless Dialogs and more contextually relevant use of delegates.

Introduction To Delegates

- Delegates are references to methods.
- What does this mean?
- Consider that we have

  int x=25.
- Here x an instance of an integer value. This value can be changed.
- We can have reference to an object of a class. E.g. **class1 x= new class1();**
- Similarly, a **delegate** creates an object that can be a reference to a **method**.
  - The method can be changed
- However, the way we use delegates is a bit different from other object references.

Introduction To Delegates

- To use delegates, we create a delegate type.
- Here, we specify the signature of the delegate type, i.e number and types of parameters and the type of the returned value.
  - Objects of this delegate type can only reference methods with the same signature.
- E.g. The statement below creates a delegate type that has a single string parameter and returns void:

```
delegate void delSingleString(string str);
```

Introduction To Delegates

- Once we have the delegate type, we can create delegate objects and assign it to compatible methods-i.e methods with the same signature

- Note: Generally, we use delegates with callback methods

Creating an instance of the delegate type and assigning a function

- Let's have a method called displayString() defined as follows:

```
void displayString(string str){
        Console.WriteLine(str);
    }
```

- We note that the method signature is compatible with the delegate type **delSingleString**
- We can create a delegate object and assign the method displayString() to it.

```
delSingleString delStr1= new delSingleString(displayString)
```

- Here delStr1 is a delegate object and it references an instance of displayString.

Creating an instance of the delegate type and assigning a function

- To execute the method we use the invoke() method of the delegate class.
- E.g. delStr1.Invoke(“Hello- How are you?”);

Lecture 9- demo1 – Creating a delegate type and instantiating a delegate object

1. Create a console (.Net Framework) application.
2. In the program class, above the Main() function, create a delegate type called **delSingleString()**. It should have one string parameter and return type void.
3. Below the Main() method, create the following method:

```
void displayString(string str){
    Console.WriteLine(str);
}
```

4. In the Main() method, create an object called **firstObject**, of type **delSingleString** and assign it to a new object of type **delSingleString**, passing **displayString** as the parameter to the constructor.
5. Call firstObject.Invoke() passing “Hello! How are you?” as the parameter.

Lecture 9- demo1(b) – Assigning different functions to a delegate reference

1. To demo1, Add a method called **displayUpper** that has as parameter a string object and it returns type void.
2. Add the required code to displayUpper so that it displays the value of its parameter in uppercase.
3. Create a variable of type **delSingleString** called **secondObject**. Assign it to a new object of class **delSingleString** passing displayUpper as parameter.

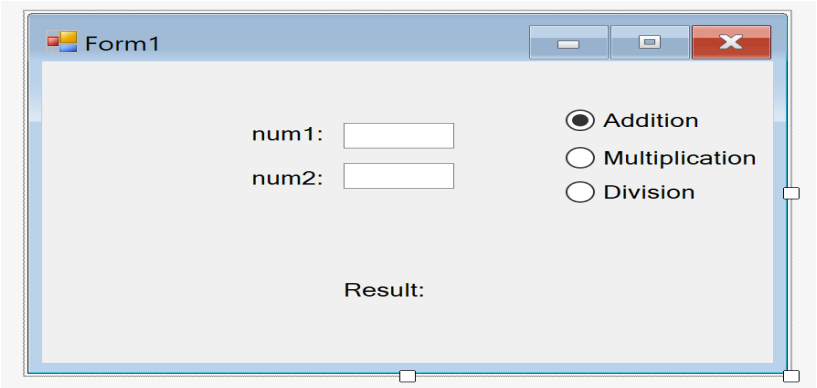
Lecture 9- Exercise 1

1. Create a Console (.NET Framework) Application.
2. In the **program** class, create a delegate type called **delBinaryOp**. It must have 2 integer parameters and return an integer value.
3. Below the Main() method create 3 methods as below, where each method has 2 integer parameters and returns an integer result:
   - Addition that returns the sum of its parameters
   - Multiplication that returns the product of its parameters
   - Division that returns the result of the integer division of the first parameter by the second one
4. Test the 3 methods by creating appropriate delegate objects and using the Invoke method.

Lecture 9 – Exercise 2

- Now We'll use our delegates in Windows Form applications.

1. Create a new Windows Form application
2. Place 3 radio buttons, 2 text boxes with corresponding labels and a label for the result as shown below



3. Use event consolidation that calls a single event handler when any one of the radio buttons is clicked or the value of any of the text boxes changes.
4. Create 3 methods for addition, multiplication and division as in exercise 1.
5. Using delegates invoke one of the methods based on which of the radio buttons is checked.
6. The Result should be recalculated whenever any radio button is changed or the text in the textboxes changes.

Shorthand forms of delegate use

- The compiler allows some shorthand forms of delegate use as below:
  - We can create a delegate object without the use of the **new** keyword
  - Instead of **delBinaryOp delObject= new delBinaryOp(Addition);**
  - we can use **delBinaryOp delObject= Addition**

  We can then invoke the method in the usual way using the delegate:

  E.g. delObject.Invoke(5,6);

Shorthand forms of delegate use

- Another shorthand form allowed by the compiler is to operate the delegate as it was the referenced method:
  - E.g. delBinaryOp delObject=Addition

    delObject(5,6);

Referencing Anonymous Methods with delegates

- A delegate object need not reference a formal method
- Recent versions of C# allow delegate objects to reference anonymous functions
  - E.g. delBinaryOp delObject= delegate (int x, int y){return x*y;};

Lecture9 – Exercise3

- In this exercise, we will illustrate the use of the 2 shorthand methods and the use of delegate with an anonymous function.
- Create a console program.
- Within the program class, create a delegate type that can accept 2 integer parameters and returns an integer value.
- Create a method Addition that has 2 integer parameters x and y and returns the sum of x and y.
- Within the Main method, create 2 delegate objects called delObj1 and delObj2.
- Initialize delObj1 to **Addition** and delObj2 to **new delBinaryOp(Addition)**
- On delObj1, use the Invoke method to find the result of Addition of 5 and 6
- Using delObj2, find the sum of 5 and 6, without using Invoke()
- Create a 3rd delegate object, delObj3, of the same type, for an anonymous method that returns the result of the multiplication of its parameters.
- Invoke delObj3 passing 5 and 8 as parameters.
- In each case, display the result of the invocation

Modeless Dialogs

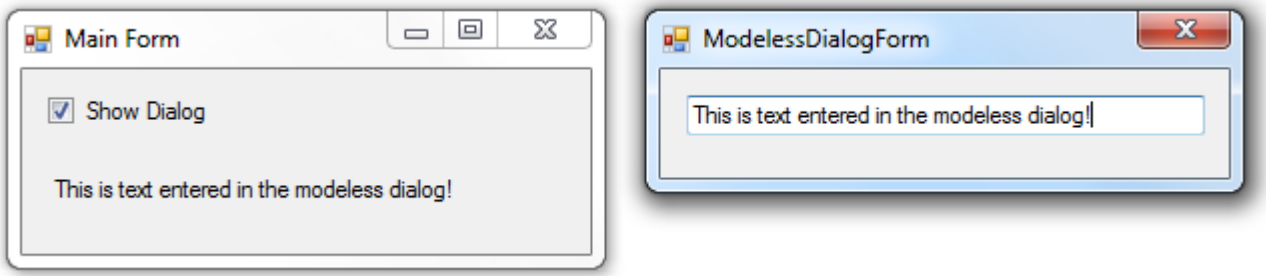
- A modeless dialog may remain present during the entire life of the calling code or application.
- This type of dialog is usually used to display dynamic, context-based information.
- A modeless dialog is created as a form, then displayed using the **Show** method rather than the ShowDialog method.
- Use the **Hide** method to have a modeless dialog disappear.

Modeless Dialogs

- The calling code can manipulate the dialog through its reference, and public methods created within the dialog.
- Problems arise when the dialog wishes to manipulate items within the calling code.
- The dialog does not have a reference to the calling code.
- A solution is provided by the use of delegates in the modeless dialog, which can hold a reference to a method located in the calling code.
- The main method will contain Callbacks that can be referenced by the delegates

Lecture9-demo2 (ModelessDialogExample)

- To illustrate the use of delegates with Modeless Dialogs, we'll create an application with the following features:
  - It will have a main form + a modeless dialog form, as shown below:
  - The user will input a text in the modeless dialog. As the text is being typed, the text also appears in the main form.
- The main form contains a checkbox used to show or hide the modeless dialog.
- The main form contains a label which displays text entered in the modeless dialog as it changes in the modeless dialog.
- The modeless dialog contains a textbox; text entered in the textbox immediately appears in the main form's label.



Lecture9-demo2 (ModelessDialogExample)

- Step 1- Create the components

Component	Name
Check Box (on Main Form)	UI_ShowDialog_Cbx
Label (on Main Form)	UI_DialogText_Lbl
Modeless Dialog Form	ModelessDialogForm
Text Input	UI_Input_Lbl

Step 2- Modeless Dialog Code

- The modeless dialog defines the delegate type and has a member variable that will reference the callback method in the main form.

```
namespace ModelessDialogExample
{
    //delegate for sending notifications to the main form
    public delegate void delVoidString(string s);

    public partial class ModelessDialogForm : Form
    {
        //delegate reference will contain the callback
        public delVoidString _dTextChanged = null;
```

Step 3- Main Form Code

- The main form declares a variable for the modeless dialog

```
ModelessDialogForm dlg = null;
```

- The checkbox **checkChange** event handler in the main form creates the modeless dialog, then assigns the modeless dialog delegate member a reference to a callback method.

```
dlg = new ModelessDialogForm();
dlg._dTextChanged = new delVoidString(CallbackTextChanged);
```

Main Form Code

```
3 references
public partial class Form1 : Form
{
    ModelessDialogForm dlg = null;
    1 reference
    public Form1()
    {
        InitializeComponent();
    }

//When the checkbox is checked for the first time,
//the modeless dialog is created and shown.
//Subsequently, whenever the checkbox is checked the dialog is shown
//when the checkbox is unchecked the dialog is hidden
1 reference
private void UI_ShowDialog_Cbx_CheckedChanged(object sender, EventArgs e)
{
    if (UI_ShowDialog_Cbx.Checked)
    {
        if (dlg == null)
        {
            dlg = new ModelessDialogForm();
            dlg._dTextChanged = CallBackTextChanged;

        }
        dlg.Show();
    }
    else dlg.Hide();
}
```

Step 4- Create Main Form Callback Method

- The callback method in the main form will update the main form's label with its parameter.
- This method will be invoked through the delegate from the modeless dialog

```
//Callback to update the label
1 reference
private void CallBackTextChanged(string str)
{
    UI_DialogText_Lbl.Text = str;
}
```

Step 5- Invoking the delegate from the Modeless dialog

- When the text is changed in the textbox, the callback method is invoked to update the label on the main form.

```
//We invoke the _dTextChanged delegate to invoke the corresponding callback at the caller
1 reference
private void UI_Input_Tbx_TextChanged(object sender, EventArgs e)
{
    if (_dTextChanged != null)
        _dTextChanged.Invoke(UI_Input_Tbx.Text);
}
```

Modeless Dialog Code

```
namespace ModelessDialogExample
{   public delegate void delVoidString(string str);


    4 references
    public partial class ModelessDialogForm : Form
    {   public delVoidString _dTextChanged=null;

        1 reference
        public ModelessDialogForm()
        {
            InitializeComponent();
        }

        1 reference
        private void UI_Input_Tbx_TextChanged(object sender, EventArgs e)
        {
            if (_dTextChanged != null)
                _dTextChanged(UI_Input_Tbx.Text);
        }
```

Closing the Modeless Dialog

- In the given demo, a checkbox is used to display the modeless dialog.
- When the dialog is closed, the checkbox should be cleared.
- A second delegate will be used to invoke a callback method in the main form to clear the checkbox when the dialog is closing.

Closing the Modeless Dialog

- In the dialog, a second delegate type is defined.
- A delegate member variable is declared to hold a reference to a callback method in the main form to uncheck the checkbox.
- The callback method is invoked when the dialog form is closing.

Modeless Dialog Code

```
namespace ModelessDialogExample
{   public delegate void delVoidString(string str);
    public delegate void delVoidVoid();

    4 references
    public partial class ModelessDialogForm : Form
    {   public delVoidString _dTextChanged=null;
        public delVoidVoid _dFormClosing = null;

        1 reference
        public ModelessDialogForm()
        {
            InitializeComponent();
        }

        1 reference
        private void UI_Input_Tbx_TextChanged(object sender, EventArgs e)
        {
            if (_dTextChanged != null)
                _dTextChanged(UI_Input_Tbx.Text);
        }
```

Modeless Dialog Code

```
        private void ModelessDialogForm_FormClosing(object sender, FormClosingEventArgs e)
        {
            //check to ensure that the delegate reference has been initialized
            if (null != _dFormClosing)
            {
                //you can invoke a delegate like it was a method
                _dFormClosing();
            }

        }
    }
}
```

Main Form Code

```
3 references
public partial class Form1 : Form
{
    //We create a variable to reference the Modeless Dialog Form
    private ModelessDialogForm dlg = null;
    1 reference
    public Form1()
    {
        InitializeComponent();
    }

    /* When the checkbox is checked, the modeless dialog is shown.
     * When it's unchecked, the dialog is hidden.
     * Note that when the checkbox is checked for the first time,
     * the modal dialog object does not exist yet. It has to be created
     */
    1 reference
    private void UI_ShowDialog_Cbx_CheckedChanged(object sender, EventArgs e)
    {
        if (UI_ShowDialog_Cbx.Checked)
        {
            if (dlg == null)
            {
                dlg = new ModelessDialogForm();
                dlg._dTextChanged = CallBackTextChanged;
                dlg._dFormClosing = CallbackFormClosing;
            }

            dlg.Show();
        }
        else
            dlg.Hide();
    }

    //Callback to update the label
    1 reference
    private void CallBackTextChanged(string str)
    {
        UI_DialogText_Lbl.Text = str;
    }

    1 reference
    private void CallbackFormClosing()
    {
        UI_ShowDialog_Cbx.Checked = false;
    }
```

The Dialog Close Button

- The dialog close button 'X' will normally close the dialog and dispose of it from memory.
- In the case of a modeless dialog, you probably just want to hide the dialog.
- You can intercept the form closing event, check for the reason, and override the closure and hide the form.

The Dialog Close Button

```
private void ModelessDialogForm_FormClosing(object sender, FormClosingEventArgs e)
{
    //is the dialog being closed by the user?
    if (e.CloseReason == CloseReason.UserClosing)
    {
        //check to ensure that the delegate reference has been initialized
        if (null != _dFormClosing)
        {
            //you can invoke a delegate like it was a method
            _dFormClosing();
        }
        // stop the close from happening
        e.Cancel = true;

        //hide the dialog
        Hide();
    }

}
```

Summary of Steps

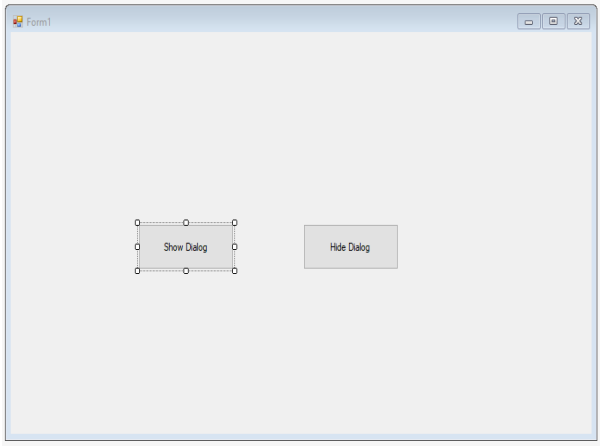
1. The namespace contains the public declaration for the delegate types.
2. The main form adds callback methods to handle events from the modeless dialog.
3. The modeless dialog contains delegate references for all callback events, set to null.
4. The main form creates the modeless dialog.
5. The main form registers its callback functions with the delegates in the dialog.
6. As events occur in the dialog, the dialog invokes the functions registered in the delegates.

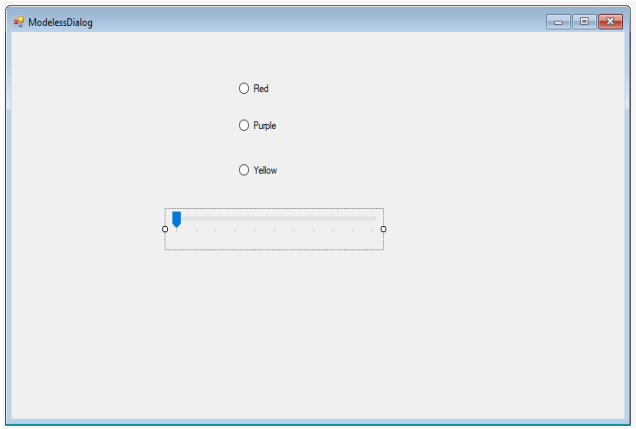
Lecture9- Exercise4

1. Create a Windows form-based project
2. On the main form add 2 buttons as shown (on the next slide). Assign the buttons the names UI_**Show_Btn** and UI_**Hide_Btn**, respectively.
3. Add a Modeless Dialog form (with name ModelessDialog) Containing 3 Radio buttons, and a trackbar as shown on next slide. To the radio buttons, assign the names **UI_Red_Radio**, **UI_Purple_Radio** and **UI_Yellow_Radio.** Assign the name **UI_OP_Track** to the trackbar
4. Add required event handlers and delegate objects, such that when you click on the Show Dialog button the modeless dialog appears and when you click on the Hide Dialog button it disappears.
5. The color of the main form will vary Red, purple or yellow depending on which Radio button is chosen. Use event consolidation.
6. The Opacity of the main form will depend on the value of the trackbar.
7. Set the track bar to vary from 0-25. Set the initial value to 25.

Lecture 9- Exercise4 (contd)

8. Use the **Opacity** property of the form to set the opacity value. Note that opacity values can be in the range 0.0-1.0.
9. Add the required event handler to the Form-Closing event of the modeless dialog, so that when the user closes the form using the X button, the form is hidden instead of being destroyed.





Generic Delegate Types

- C# defines a few generic delegate types to avoid us the task of creating a new custom delegate type for each use.
- We'll look at 2 of these here:
  - Action – Has zero or more input parameters and no output parameter (i.e it refers to a void type method)
  - Func – Has zero or more input parameters and one output parameter (or return type)

Example of Using an Action delegate type

- The Action delegate type is defined as:

public delegate void Action<in T>(T obj);

- Consider that we have the following method:

void display(string str)

And we want to call it through a delegate.

We can have:

public delegate void delVoidString(string str);

- Then we can create an instance of the delegate type by using:

public delVoidString delDisplayString=new delDisplaySTring(display);

delDisplayString.Invoke("Hello- How are you?");

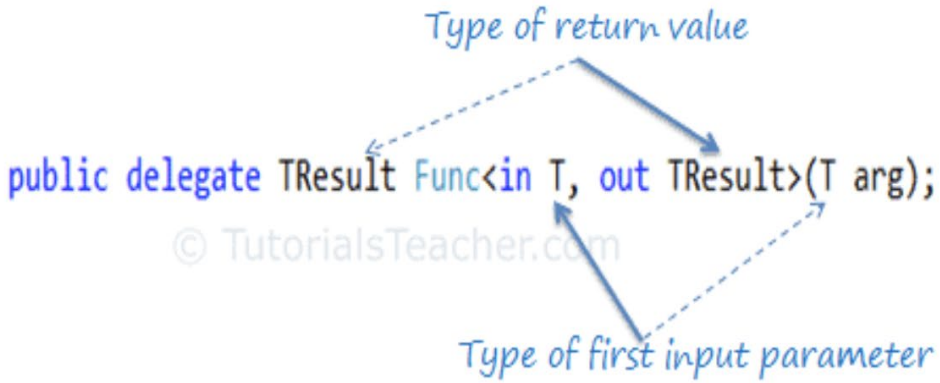
- Instead we can have:

public Action<string> delDisplayString= new Action<string>(display);

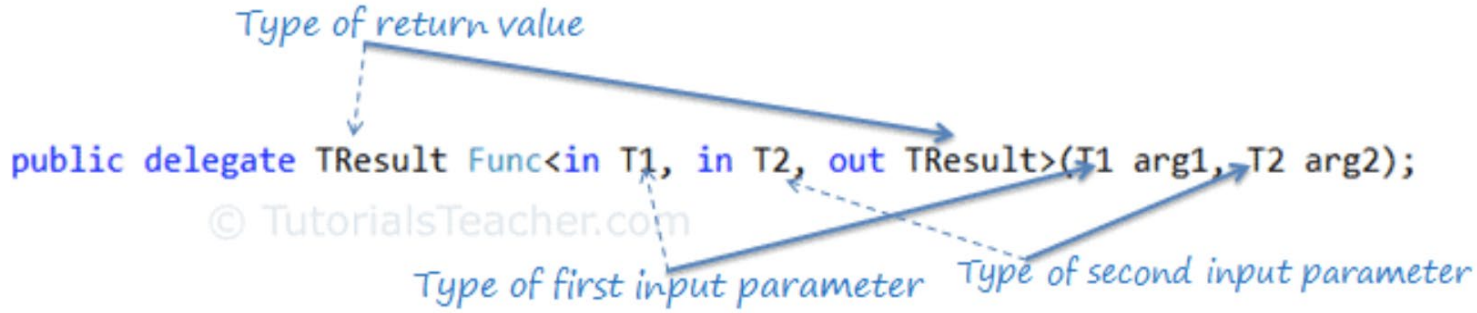
delDisplayString.Invoke("Hello- How are you?");

The **Func** Delegate Type

- A **Func** Delegate type can have any number of input parameters but has one output parameter.
- In its simplest form, the Func delegate type is as below:



- For a 2 input-parameter delegate, we have



Example of Using the Func delegate type

- Consider that we have the following method:

  int Sum(int x, int y)

And we want to call it through a delegate.

We can have:

public delegate int delIntInt(int x, int y);

- Then we can create an instance of the delegate type by using:

public delIntInt delBinaryOp=new delIntInt(Sum);

int result= delBinaryOp.Invoke(5,4);

- Instead we can have:

public Func<int, int,int> delBinaryOp= new Func<int, int, int>(Sum);

int result= delBinaryOp.Invoke(5,4);

Lecture9- Exercise5

- Make a copy of Exercise 4 and name it Lecture9-Exercise5
- Remove the declaration of the 2 custom delegate types
- Use Action delegates, instead, for the delegate objects.