Programming III

Centennial College

Week#1 & #2
Winter 2020

Topic: Delegates and Event Handling

Objectives

- Understand what is delegate, the purpose of the delegate and how to use delegate in the code
- Demonstrate a good understanding of **event-handling mechanism** in .NET
- Practice the use of delegates to implement event handling in .NET, and **create dynamic forms** by registering Windows controls with their event handlers by code

What is Delegate

- A delegate is a class that represents references to methods with a particular parameter list and return type
- Delegates are used to pass methods as arguments to other methods
- The following example declares a delegate named ***Del*** that can encapsulate a method that takes a string as an parameter and returns void

public ***delegate*** *void Del(string message);*

More on Delegate

- There are two terms for class, class and object; however with delegates, there is only one term, **delegate** which can refer to both the class and the object. In other words, an instance of a delegate is also referred to as a delegate
- A delegate allows the programmer to **encapsulate a reference to a method inside an instance of a delegate**. And the referenced method can be invoked during the execution time **without having to know at compile time**
- **Delegates** are type-safe classes that define the return type and types of parameters

Declare Delegates

- A delegate type is declared using the **delegate** keyword along with the method signature of the delegate and the return type of the method that it represents

  delegate return-type identifier ([parameters]);

- Examples
  - *public delegate void SimpleDelegate ();*
    - Defines a delegate named *SimpleDelegate*, which will encapsulate any method that has no parameter and returns no value.
  - *public delegate string AnotherDelegate (object o1, object o2)*
    - Defines a delegate named *AnotherDelegate*, which will encapsulate any method that takes two objects as parameters and returns a string.

Delegate Example

```
namespace SimpleDelegate
{
    public class SimpleDelegate
    {
        public delegate void Del(string message);

        static void Main(string[] args)
        {
            // Instantiate the delegate.
            Del handler = DelegateMethod;

            // Call the delegate.
            handler("Hello World");
            handler.Invoke("this is through invoke: Hello World");

            Console.ReadKey();
        }

        public static void DelegateMethod(string message)
        {
            System.Console.WriteLine(message);
        }
    }
}
```

Delegates & the Event-Handling Mechanism

- The control that generates an event is the event sender
- The event-handling method (a.k.a event handler) responds to a particular event (e.g., click a button) that a control generates
- When an event occurs, the event sender calls its event handler to perform a task
- Event handlers are connected to a control's events through **delegates**.
- A **delegate** holds a reference to a method.
- A delegate of type `EventHandler` must return `void` and receive two parameters—one of type `object` and one of type `EventArgs`:

```
public delegate void EventHandler( object sender, EventArgs e );
```

GUI example

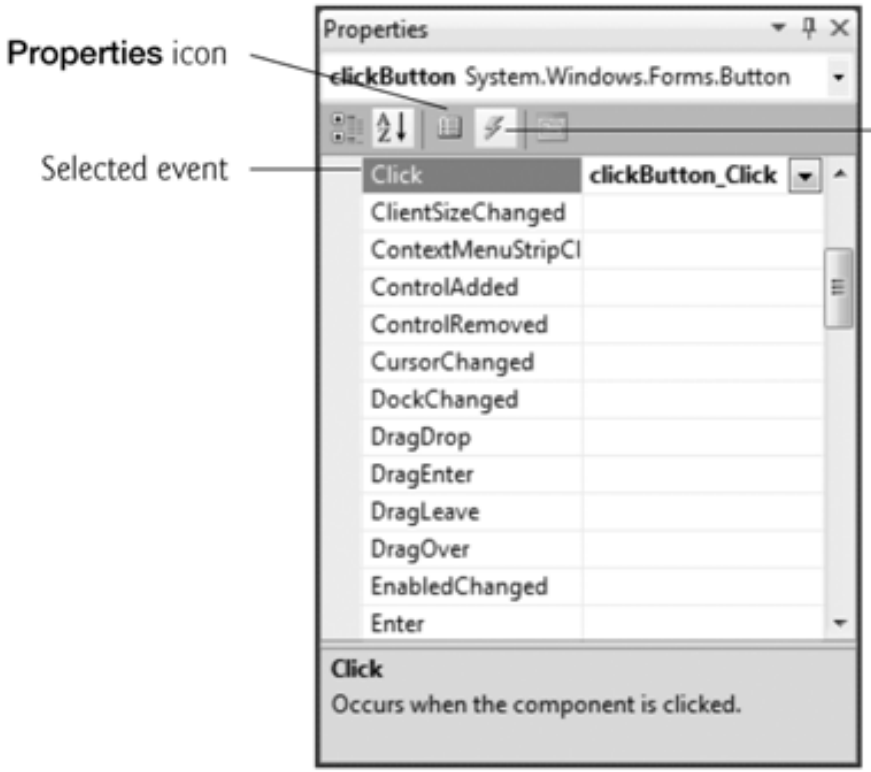
```
private void InitializeComponent()
{
    this.clickButton = new System.Windows.Forms.Button();
    this.SuspendLayout();
    //
    // clickButton
    //
    this.clickButton.Location = new System.Drawing.Point(104, 37);
    this.clickButton.Name = "clickButton";
    this.clickButton.Size = new System.Drawing.Size(75, 23);
    this.clickButton.TabIndex = 0;
    this.clickButton.Text = "Click Me";
    this.clickButton.UseVisualStyleBackColor = true;
    this.clickButton.Click += new System.EventHandler(this.clickButton_Click);
```

Create an EventHandler delegate instance and Initializes it with clickButton_click method

8 Add the delegate to Button's click event handler delegate.
In other words, register the control (i.e. the clickButton) with the event handler

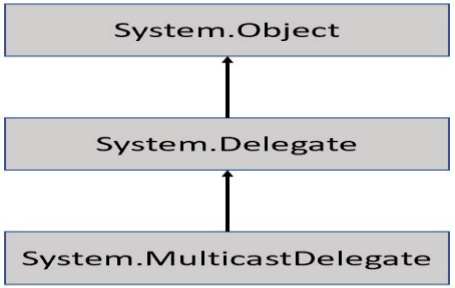
Another Way to Create Event Handlers

- Controls can generate many different events.
- Clic[illegible] the **Events** icon (the lightning-bolt icor[illegible]ays all the



Multicast Delegates

- Developer can specify that several different methods to be invoked in response to an event by adding delegates to the event
- Adding more than one delegates to an event is called **multicast delegates**
- **Add method** to multicast delegate: + or +=
- **Remove method** calls from multicast delegates: - or -=
- **Inheritance hierarchy**



Multicast Delegates Example

- Any object can *publish* a set of events to which other classes can *subscribe*. When the publishing class raises an event, all the subscribed classes are notified
  - With this mechanism, the publish object can say, "Here are things I can notify you about," and other classes might sign up, saying, "Yes, let me know when that happens."
    - For example, a button might notify any number of interested observers when it is clicked. The button is called the *publisher* because the button publishes the Click event and the other classes are the *subscribers* because they subscribe to the Click event. Note that the publishing class does not know or care who (if anyone) subscribes; it just raises the event. Who responds to that event, and how they respond, is not the concern of the publishing class.
- The publisher and the subscribers are decoupled by the delegate.

Publish/subscribe Example

```
public class Publisher
{
        //Declare Delegate
        public delegate void PublishMessageDel(string msg);

        //Declare an instance variable which is a Delegate object
        public PublishMessageDel publishmsg = null;

        //Method used to Invoke Delegate
        public void PublishMessage(string message)
        {
                //Invoke Delegate
                publishmsg.Invoke(message);
        }
}
```

Publish/subscribe Example (Con't)

```
public class SendViaEmail
{
        private String emailAddr;

        public SendViaEmail() { }

        public SendViaEmail(String emailAddr)
        {
            this.emailAddr = emailAddr;
        }

        public void setEmailAddr(String emailAddr)
        {
            this.emailAddr = emailAddr;
        }

        public String getEmailAddr()
        {
            return emailAddr;
        }

        public void sendEmail(string msg)
        {
                Console.WriteLine("The message" + "\""+ msg+ "\" has already sent to "+ emailAddr);
        }

        public void Subscribe(Publisher pub)
        {
                pub.publishmsg += sendEmail;
        }
}
```

Publish/subscribe Example (Con't)

```
class Driver
{
        public static void Main(string[] args)
        {
                Publisher publisher = new Publisher();

                SendViaMobile send2Mobile = new SendViaMobile("416 1234567");
                SendViaEmail send2Email = new SendViaEmail("li@my.centennialcollege.ca");

                //Subscribing for Mobile notifications
                send2Mobile.Subscribe(publisher);

                //Emails are not subscribed so it wont receive notifications via Email
                send2Email.Subscribe(publisher);

                //Invoking the delegate Only Mobile will receive notifications.
                publisher.PublishMessage("Hello You Have New Notifications");
                Console.ReadKey();
        }
}
```

Lambda Expressions(1/2)

- A lambda expression is an anonymous function that is used to create delegates

- To create a lambda expression, the input parameters (if any) is located on the left side of the lambda operator => , and the expression or statement block is put on the other side

Lambda Expressions(2/2)

```
1   using System;
2   using System.Collections.Generic;
3   using System.Linq;
4   using System.Text;
5   using System.Threading.Tasks;
6
7   namespace DemoLambdas
8   {
        0 references
9       class Program
10      {
11          private delegate int del(int x);
12
            0 references
13          static void Main(string[] args)
14          {
15              del myDelegate = x => x * x;
16
17              Console.WriteLine("The result = {0}",myDelegate(5));
18
19              Console.ReadKey();
20          }
21      }
22  }
```

Action<T> and Func<T> Delegates

- Action<T> delegate is to refer a method with void return
  - Can pass up to 16 different parameter types
- Func<T> delegate is to refer a method with a return type
  - Can pass up to 16 different types
  - Func(in T1, in T2, in T3, in T4, out TResult) is a method with **4** parameters

Example of Func<T> & Lambda

```
static void Main(string[] args)
{
    string mid = ",middle part, ";

    Func<string, string> del = message =>
        {
            message += mid;
            message += " and this was added to the message ";
            return message;
        };

    Console.WriteLine(del("Start of string"));
}
```

References

- .NET Documentation