RED RIVER COLLEGE
OF APPLIED ARTS, SCIENCE AND TECHNOLOGY

RED RIVER COLLEGE

ADEV-2005 Programming 2
Applied Computer Education
Red River College

UNIT 4 - EVENTS AND EVENT HANDLING

Topics

RED RIVER COLLEGE

- What is an Event?
- Why use Events?
- Coding events in classes
- Handling events of objects

What do we do?

```
public class Student
{
    public void AddTestMark(double percentage)
    {
        this.NumberOfTests++;
        this.SumOfTests += percentage;

        if(this.TestAverage < .65)
        {
            // something needs to happen here
        }
    }
}
```

Option 1: Code statement(s)

```
public class Student
{
    public void AddTestMark(double percentage)
    {
        this.NumberOfTests++;
        this.SumOfTests += percentage;

        if(this.TestAverage < .65)
        {
            Console.WriteLine("DANGER!!!");
        }
    }
}
```

Option 2: Dynamic statement(s)

```
public class Student
{
    public void AddTestMark(double percentage, string message)
    {
        this.NumberOfTests++;
        this.SumOfTests += percentage;

        if(this.TestAverage < .65)
        {
            Console.WriteLine(message);
        }
    }
}
```

Option 3: More dynamic statement(s)

```
public class Student
{
    public void AddTestMark(double percentage,
                            string message,
                            MediaMode mode)
    {
        this.NumberOfTests++;
        this.SumOfTests += percentage;

        if(this.TestAverage < .65)
        {
            if(mode == MediaMode.Console)
                Console.WriteLine(message);
            else if(mode == MediaMode.WindowsForm)
                MessageBox.Show(message);
        }
    }
}
```

Can we do this????...I wish

```
public class Student
{
    public void AddTestMark(double percentage,
                            Method myAction)
    {
        this.NumberOfTests++;
        this.SumOfTests += percentage;

        if(this.TestAverage < .65)
        {
            myAction();
        }
    }
}
```

Delegate

- One of the 5 base types (Class, Structure, Interface, Enumeration and Delegate)
- Instances of delegates store references to methods (methods are objects)

Declaring a Delegate

```
public delegate void InstructorAction();

public delegate int Add(int a, int b);
```

Delegate References

- Every delegate has a signature
- Unlike methods, the return type is part of the delegate's signature
- Delegates can only reference methods that match their signature (including return type)

Instantiating a Delegate (1 of 2)

```
public delegate void InstructorAction();

class Program
{
    static void Main(string[] args)
    {
        InstructorAction myAction =
            new InstructorAction(AlertStudent);
    }

    static void AlertStudent()
    {
        Console.WriteLine("DANGER!");
    }
}
```

Instantiating a Delegate (2 of 2)

```
public delegate void InstructorAction();

class Program
{
    static void Main(string[] args)
    {
        InstructorAction a = AlertStudent;
    }

    static void AlertStudent()
    {
        Console.WriteLine("DANGER!");
    }
}
```

Invoking a Delegate

```
public void AddTestMark(double percentage,
                        InstructorAction action)
{
    if (action == null)
       throw new ArgumentNullException();

    this.NumberOfTests++;
    this.SumOfTests += percentage;

    if(this.TestAverage < .65)
    {
        action.Invoke();
        // action();
    }
}
```

Delegate as an Argument

```
class Program
{
    static void Main(string[] args)
    {
        Student s = new Student();

        InstructorAction myAction =
            new InstructorAction(AlertStudent);

        s.AddTestMark(.14, myAction);
    }

    static void AlertStudent()
    {
        Console.WriteLine("DANGER!");
    }
}
```

Multicast Delegates (1 of 2)

- A delegate can reference and invoke 1 or more methods
- The operators += and -= can be used to add methods to and remove from the delegate's **invocation list**

Multicast Delegates

```
class Program
{
    static void Main(string[] args)
    {
        Student s = new Student();

        InstructorAction myAction =
            new InstructorAction(AlertStudent);

        myAction += AlertParents;

        s.AddTestMark(.14, myAction);
    }

    static void AlertParents()
    {
        Console.WriteLine("Emailing parents....SUCCESSFUL!");
    }
}
```

Unit 4 – Events and Event Handling

EVENTS

What is an event?

- A mechanism for an object to notify other objects that something interesting happened to it
- Example:
  - Button Object
    - Click Event
  - Textbox Object
    - TextChanged Event

Typical Event Uses

- Object state changed
- Object action (method) takes place

Relationship of Delegates and Events

- Events are delegates
- Acts as the intermediary between objects that raise events and objects that respond to events.

Declaring an Event

```
public class Student
{
    /// <summary>
    /// Occurs when the students test average
    /// drops below 65%.
    /// </summary>
    public event EventHandler AverageBelowFail;
}


// From documentation for EventHandler
public delegate void EventHandler(object sender,
                                  EventArgs e)
```

Sender Parameter

- The `sender` parameter references the **event sender**
  - The object that raised the event

EventArgs Parameter

- The event data object
  - Data about the event that took place
- Base type contains no data
- Derived types contain properties to access event data
  - Example: RowUpdatedEventArgs
  - `e.Row` – References the DataRow that was updated

On Method

```
public class Student
{
    /// <summary>
    /// Called when a Student's grade drops below
    /// the fail threshold.
    /// </summary>
    protected virtual void OnAverageBelowFail()
    {
        if(AverageBelowFail != null)
        {
            AverageBelowFail(this, new EventArgs());
        }
    }
}
```

Raise the Event

```
public class Student
{
    public void AddTestMark(double percentage)
    {
        this.NumberOfTests++;
        this.SumOfTests += percentage;

        if (this.TestAverage < .65)
        {
            OnAverageBelowFail();
        }
    }
}
```

Code Event Handler Method

```
class Program
{
    static void Main(string[] args)
    {
        Student s = new Student();

    }


    /// <summary>
    /// Handles the AverageBelowFail event of a Student.
    /// </summary>
    private static void Student_AverageBelowFail(object sender,
                                                 EventArgs e)
    {
        Console.WriteLine("Not cool :(");
    }
}
```

Event Wiring (1 of 2)

- Event handler methods must be attached (wired up) to the event
- The += operator adds the method (handler) to the event's invocation list

Event Wiring (2 of 2)

```
class Program
{
    static void Main(string[] args)
    {
        Student s = new Student();

        s.AverageBelowFail += Student_AverageBelowFail;

        s.AddTestMark(.14);
    }

    private static void Student_AverageBelowFail(object sender,
                                                 EventArgs e)
    {
        Console.WriteLine("Not cool :(");
    }
}
```

Unit 4 – Events and Event Handling

ANOTHER EXAMPLE

Declaring an Event

```
public class Person
{
    /// <summary>
    /// Occurs when the amount of money changes.
    /// </summary>
    public event EventHandler AmountOfMoneyChanged;
}
```

On Method

```
public class Person
{
    /// <summary>
    /// Called when a Person's amount of money changes.
    /// </summary>
    protected virtual void OnAmountOfMoneyChanged()
    {
        if (AmountOfMoneyChanged != null)
        {
            AmountOfMoneyChanged(this, new EventArgs());
        }
    }
}
```

Raise the Event

```
public class Person
{
    public decimal AmountOfMoney
    {
        get
        {
            return this.amountOfMoney;
        }

        set
        {
            if (value != this.amountOfMoney)
            {
                this.amountOfMoney = value;
                OnAmountOfMoneyChanged();
            }
        }
    }
}
```

Update Other Code

```
public void AddMoney(decimal amountOfMoneyToAdd)
{
    this.AmountOfMoney += amountOfMoneyToAdd;
}

public void SubtractMoney(decimal amountOfMoneyToSubstract)
{
    this.AmountOfMoney -= amountOfMoneyToSubstract;
}
```

Event Handler Method

```
class Program
{
    static void Main(string[] args)
    {
    }

    static void Person_AmountOfMoneyChanged(object sender, EventArgs e)
    {
        Console.WriteLine("My amount of money changed.");
    }
}
```

Event Wiring

```
class Program
{
    static void Main(string[] args)
    {
        Person me = new Person("Damien");

        me.AmountOfMoneyChanged +=
            new EventHandler(PersonAmountOfMoneyChanged);
    }

    static void PersonAmountOfMoneyChanged(object sender, EventArgs e)
    {
        Console.WriteLine("My amount of money changed.");
    }
}
```

Test Event Handler

```
static void Main(string[] args)
{
    Person me = new Person("Damien");

    me.AmountOfMoneyChanged +=
        new EventHandler(PersonAmountOfMoneyChanged);

    Console.WriteLine(me);

    me.AddMoney(400);

    Console.WriteLine(me);
}
```

Unit 4 – Events and Event Handling

OBSERVER PATTERN

Observer Pattern

- Software design pattern
- Subject object maintains a list of dependents (observers)
- Subject notifies observers when its state has changed
  - Usually by calling a dependents method

Observer Pattern in C#

- Subject object (publisher)
  - Contains the event
- Observer object (subscriber)
  - Provides a method to handle the event (subscribes)
- When the subject object raises the event, all observer objects are notified

Pattern Characteristics

- Subject object determines when event is raised
  - observer determines the action
- An event can have multiple observers (multicast)
- Events with multiple observers are invoked synchronously
- Observers can observe one or more events from one or more subjects

UML

PlayingCard
+ <<event>> Flipped : EventHandler
# OnFlipped() : void

Sample Event Unit Test

```
[TestClass]
public class StudentTest
{
    private string eventActual = null;

    void TestHandler(object sender, EventArgs e)
    {
        eventActual = "Event triggered.";
    }

    [TestMethod]
    public void AverageBelowFailEvent_Test()
    {
        this.eventActual = null;

        Student s = new Student();

        s.AverageBelowFail += TestHandler;

        s.AddTestMark(.14);

        Assert.AreEqual("Event triggered.", eventActual);
    }
}
```

Summary

- Event = object (subject) notify other objects (observers) about state changes
- Observer defines method to handle event
- += attaches the handler method (observer)
- Subject determines when event is raised

QUESTIONS?