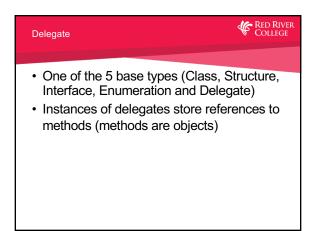
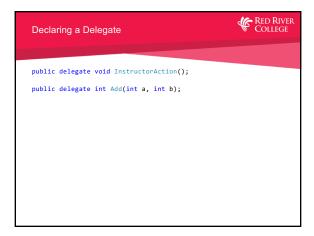


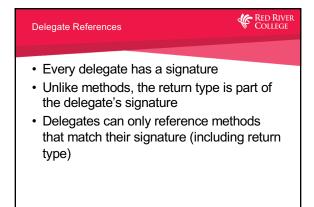
```
public class Student
{
   public void AddTestMark(double percentage)
   {
      this.NumberOfTests++;
      this.SumOfTests += percentage;
      if(this.TestAverage < .65)
      {
            // something needs to happen here
      }
   }
}</pre>
```

```
public class Student
{
   public void AddTestMark(double percentage)
   {
      this.NumberOfTests++;
      this.SumOfTests += percentage;
      if(this.TestAverage < .65)
      {
            console.WriteLine("DANGER!!!");
      }
    }
}</pre>
```

```
public class Student
{
   public void AddTestMark(double percentage, string message)
   {
      this.NumberOfTests++;
      this.SumOfTests += percentage;
      if(this.TestAverage < .65)
      {
            Console.WriteLine(message);
      }
   }
}</pre>
```







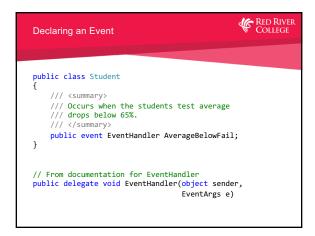
```
public delegate void InstructorAction();
class Program
{
    static void Main(string[] args)
    {
        InstructorAction a = AlertStudent;
    }
    static void AlertStudent()
    {
        Console.WriteLine("DANGERI");
    }
}
```

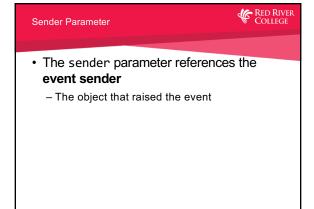
A delegate can reference and invoke 1 or more methods The operators += and -= can be used to add methods to and remove from the delegate's invocation list

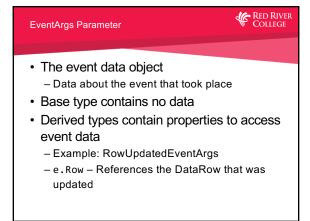




Events are delegates
 Acts as the intermediary between objects that raise events and objects that respond to events.







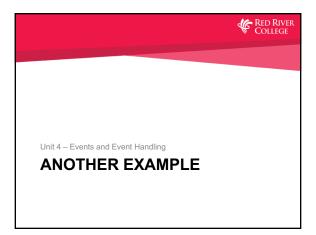
```
public class Student
{
    /// <summary>
    /// Called when a Student's grade drops below
    /// the fail threshold.
    /// </summary>
    protected virtual void OnAverageBelowFail()
    {
        if(AverageBelowFail != null)
        {
            AverageBelowFail(this, new EventArgs());
    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

```
public class Student
{
   public void AddTestMark(double percentage)
   {
     this.NumberOfTests++;
     this.SumOfTests += percentage;
     if (this.TestAverage < .65)
     {
        OnAverageBelowFail();
     }
   }
}</pre>
```

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

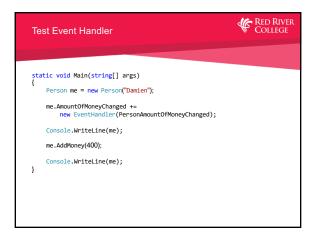


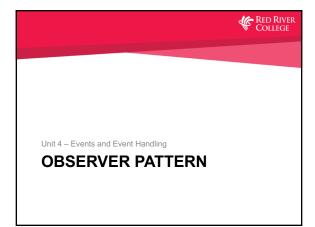
```
public class Person
{
    /// <summary>
    /// Occurs when the amount of money changes.
    /// </summary>
    public event EventHandler AmountOfMoneyChanged;
}
```

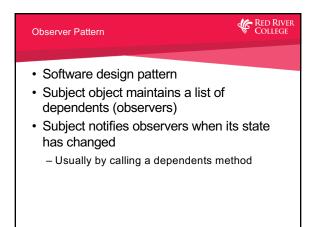
```
public class Person
{
    /// <summary>
    /// Called when a Person's amount of money changes.
    /// </summary>
    protected virtual void OnAmountOfMoneyChanged()
    {
        if (AmountOfMoneyChanged != null)
        {
            AmountOfMoneyChanged(this, new EventArgs());
        }
    }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

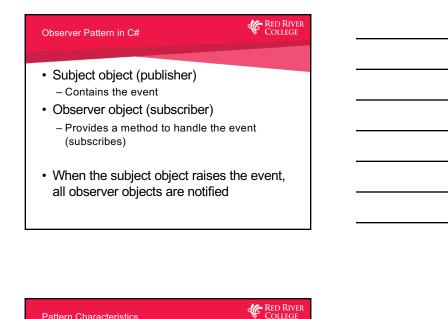
```
public void AddMoney(decimal amountOfMoneyToAdd)
{
    this.AmountOfMoney += amountOfMoneyToAdd;
}
public void SubtractMoney(decimal amountOfMoneyToSubstract)
{
    this.AmountOfMoney -= amountOfMoneyToSubstract;
}
```

```
class Program
{
    static void Main(string[] args)
    {
        static void Person_AmountOfMoneyChanged(object sender, EventArgs e)
        {
            Console.WriteLine("My amount of money changed.");
        }
}
```



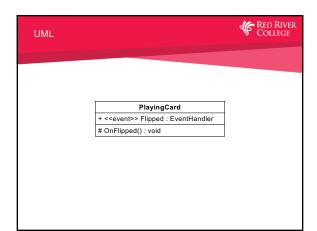






- Subject object determines when event is raised - observer determines the action An event can have multiple observers (multicast) • Events with multiple observers are invoked
- synchronously • Observers can observe one or more events from one or more subjects

Pattern Characteristics



```
[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);
}
}
```

