# CS 683 Emerging Technologies
# Spring Semester, 2003
# Doc 26 C# Delegates & Events
# Contents



## References

Programming C#, Jesse Liberty, O'Reilly, Chapters 9, 11

C# Language Specification, http://download.microsoft.com/download/0/a/c/0acb3585-3f3f-4169-ad61-efc9f0176788/CSharp.zip

C# Class Library Detailed Specifications, http://download.microsoft.com/download/3/c/2/3c26966b-ce53-4d78-9076-5cb8331844f1/TypeLibrary.zip

# Delegates

You can think of delegates as references to member methods

```
using System;

public delegate void D(int x);

class Test
	{
	public void M(int i)
		{
		Console.WriteLine("Test.M: " + i);
		}
	}

public class Tester
	{
	public static void Main()
		{
		Test t = new Test();
		D sample = new D(t.M);
		sample(10);
		tryIt( sample);
		}
	public static void tryIt(D aDelegate)
		{
		aDelegate(5);
		}
	}
```

## Output

```
Test.M: 10
Test.M: 5
```

# Method and Delegate Matching

A method and a delegate type are compatible if both of the following are true:

- They have the same number or parameters, with the same types, in the same order, with the same parameter modifiers.
- Their return-types are the same.

Delegate types are name equivalent,

Two different delegate types that have the same parameter lists and return type are considered **different** delegate types.

```
delegate int D1(int i, double d);
class A
{
  public static int M1(int a, double b) {…}
}
class B
{
  delegate int D2(int c, double d);
  public static int M1(int f, double g) {…}
  public static void M2(int k, double l) {…}
  public static int M3(int g) {…}
  public static void M4(int g) {…}
}
```

# Combining Delegates

Delegates can be combined using + or +=

Delegates can be removed from a combination using – or -=

When removing a delegate the last occurrence is removed

```
delegate void D(int x);
class Test
{
  public static void M1(int i) {…}
  public static void M2(int i) {…}
}
class Demo
{
  static void Main() {
    D cd1 = new D(Test.M1);       // M1
    D cd2 = new D(Test.M2);       // M2
    D cd3 = cd1 + cd2;            // M1 + M2
    D cd4 = cd3 + cd1;            // M1 + M2 + M1
    D cd5 = cd4 + cd3;            // M1 + M2 + M1 + M1 + M2
    D cd6 = cd5 – cd1;            // M1 + M2 + M1 + M2
  }
}
```

# Another Example

```
using System;
delegate void D(int x);
class Test
{
   public static void M1(int i) {
      Console.WriteLine("Test.M1: " + i);
   }
   public static void M2(int i) {
      Console.WriteLine("Test.M2: " + i);
   }
   public void M3(int i) {
      Console.WriteLine("Test.M3: " + i);
   }
}
```

```
class Demo
{
   static void Main() {
      D cd1 = new D(Test.M1);
      cd1(-1);      // call M1
      D cd2 = new D(Test.M2);
      cd2(-2);      // call M2
      D cd3 = cd1 + cd2;
      cd3(10);      // call M1 then M2

      cd3 += cd1;
      cd3(20);      // call M1, M2, then M1
      Test t = new Test();
      D cd4 = new D(t.M3);
      cd3 += cd4;
      cd3(30);      // call M1, M2, M1, then M3
      cd3 -= cd1; // remove last M1
      cd3(40);      // call M1, M2, then M3
      cd3 -= cd4;
      cd3(50);      // call M1 then M2
      cd3 -= cd2;
      cd3(60);      // call M1
      cd3 -= cd2; // impossible removal is benign
      cd3(60);      // call M1
      cd3 -= cd1; // invocation list is empty
//    cd3(70);      // System.NullReferenceException thrown
      cd3 -= cd1; // impossible removal is benign
   }
}
```

# Combination of Delegates

The same parameters are sent to each delegate in a combination in order

Changes to a reference parameter done in one delegate is known to the remaining delegates

If delegates return a value the value returned in a combination is the result of the last delegate in the list

If an unhandled exception occurs in a delegate in a combination the remaining delegates are not run

# Delegates are Subclasses of System.Delegate

## Methods & Properties

public virtual object Clone()

public static Delegate Combine(Delegate a, Delegate b)
public static Delegate Combine(Delegate[] delegates)

public static Delegate CreateDelegate(Type type, object target, string method)
public static Delegate CreateDelegate(Type type, Type target, string method)
public static Delegate CreateDelegate(Type type, MethodInfo method)

public object DynamicInvoke(object[] args)

public override bool Equals(object obj)
public override int GetHashCode()
public virtual Delegate[] GetInvocationList()

public static bool operator ==(Delegate d1, Delegate d2)
public static bool operator !=(Delegate d1, Delegate d2)

public static Delegate Remove(Delegate source, Delegate value)
public static Delegate RemoveAll(Delegate source, Delegate value)

public MethodInfo Method { get; }
Gets the last method in a delegate's invocation list.

public object Target { get; }
Gets the last object upon which a delegate invokes an instance method.

```
using System;
class MyClass {
 public int Increment(ref int i) {
   Console.WriteLine("Incrementing {0}",i);
   return (i++);
 }
 public int Negate(ref int i) {
   Console.WriteLine("Negating {0}",i);
   i = i * -1;
   return i;
 }
}

public delegate int DelegatedMethod(ref int i);
class TestClass {
 public static void Main() {
   MyClass myInstance = new MyClass();
   DelegatedMethod delIncrementer = new DelegatedMethod(myInstance.Increment);
   DelegatedMethod delNegater = new DelegatedMethod(myInstance.Negate);
   DelegatedMethod d = (DelegatedMethod) Delegate.Combine(delIncrementer,
delNegater);
   int i = 1;
   Console.WriteLine("Invoking delegate using ref value {0}",i);
   int retvalue = d(ref i);
   Console.WriteLine("After Invoking delegate i = {0} return value is {1}",i, retvalue);
  }
}
```

## Output

```
Invoking delegate using ref value 1
Incrementing 1
Negating 2
After Invoking delegate i = -2 return value is -2
```

# Events

An ***event*** is a member that enables an object or class to provide notifications

```
using System;

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

public class Button
{
   public event EventHandler Click;

   protected void OnClick(EventArgs e) {
      if (Click != null) Click(this, e);
   }

   public void Reset() {
      Click = null;
   }
}
```

# Registering For the Event

```
public class Form1
{
   Button Button1 = new Button();

   public Form1() {
      // Add Button1_Click as an event handler for Button1's Click event
      Button1.Click += new EventHandler(Button1_Click);
   }

   void Button1_Click(object sender, EventArgs e) {
      Console.WriteLine("Button1 was clicked!");
   }

   public void Disconnect() {
      Button1.Click -= new EventHandler(Button1_Click);
   }
}
```

# Event Operations

Operations permitted outside of defining class

- **+=**
- **-=**

# Complete Example

```
using System;

public class EndOfSemesterEventArgs : EventArgs {
	public readonly string grade;

	public EndOfSemesterEventArgs(string grade ) {
		this.grade = grade;
	}
}

public class Course {
	public delegate void EndSemesterHandler(object course,
		EndOfSemesterEventArgs grade);

	public event EndSemesterHandler EndSemester;

	public void TriggerEnd() {
		if (EndSemester != null)
			EndSemester(this, new EndOfSemesterEventArgs( "A++"));
	}
}
```

```
public class Student {
	public void Subscribe(Course aCourse) {
		aCourse.EndSemester+=
			new Course.EndSemesterHandler(Grades);
	}

	public void Unsubscribe(Course aCourse) {
		aCourse.EndSemester -= new Course.EndSemesterHandler(Grades);
	}

	public void Grades(object course, EndOfSemesterEventArgs grade) {
		Console.WriteLine( "My Grade is: {0}", grade.grade);
	}
}

public class Tester
	{
	public static void Main()
		{

		Course cs683 = new Course();
		Student sam = new Student();
		Student pete = new Student();
		sam.Subscribe(cs683);
		pete.Subscribe(cs683);
		cs683.TriggerEnd();
		}
	}
```

# Where is the list of Observers?

The compiler

- Automatically creates storage to hold the delegate

- Creates accessors for the event that add or remove event handlers to the delegate field

```
class X
{
  public event D Ev;
}
```

Compiles to something like:

```
class X
{
  private D __Ev;  // field to hold the delegate
  public event D Ev {
    add {
      lock(this) { __Ev = __Ev + value; }
    }
    remove {
      lock(this) { __Ev = __Ev - value; }
    }
  }
}
```

# Static Events

```
class X
{
   public static event D Ev;
}
```

Compiles to something like:

```
class X
{
   private static D __Ev;  // field to hold the delegate
   public static event D Ev {
      add {
         lock(typeof(X)) { __Ev = __Ev + value; }
      }
      remove {
         lock(typeof(X)) { __Ev = __Ev - value; }
      }
   }
}
```

## More Control over adding/removing Observers

Each event has an add and remove method

```
public class Button
{
    private EventHandler handler;

    public event EventHandler Click {

      add { handler += value; }

      remove { handler -= value; }
    }
}
```

# Web Services

The easy way:

In Visual Studio .NET create a C# Web Service project

In the class created add methods

Marking a method with [WebMethod] exposes the method as a SOAP method

```
public class Example : System.Web.Services.WebService {

   // stuff removed

   [WebMethod]
   public int Add(int a, int b) {
      return a + b;
   }
}
```