



COMPSCI 280 S1 2011 Enterprise Software Development

Programming Fundamentals
Delegates



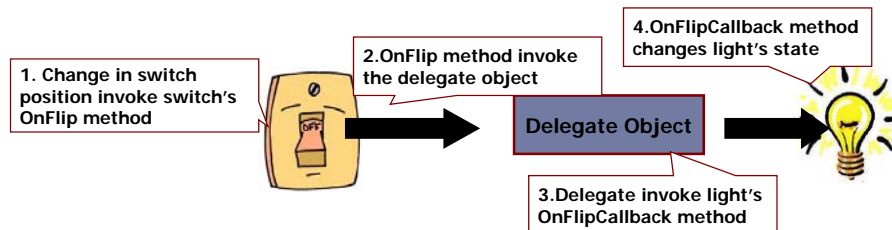
Agenda & Reading

- ▶ **Agenda:**
 - ▶ Delegates
 - ▶ Overview
 - ▶ Creating Delegates
 - ▶ Using Delegates
 - Using Array of Delegates
 - Named & Anonymous Methods
 - Multicast Delegates
 - ▶ Events
 - ▶ Overview
 - ▶ Event Handling
- ▶ **Recommended Reading:**
- ▶ **Hands-On Lab:**
 - ▶ Lecture 13 Lab



Delegates Overview

- ▶ A delegate is a reference type that defines a method signature
- ▶ Delegate are object-oriented, type-safe, and secure function pointers
 - ▶ Example:
 - ▶ The switch object models an electric switch
 - ▶ The light object models an electric light
 - ▶ The delegate object encapsulates a reference to a light object's OnFlipCallback method

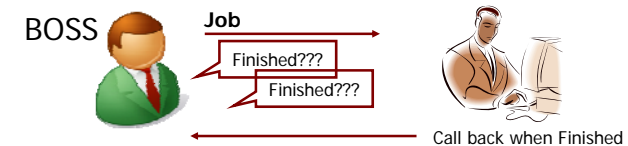


- ▶ The switch object code could be written without a delegate by referring directly to the specific light object's method. However, then it can't dynamically connect/disconnect/reconnect various light object method to the switch object.



Characteristics

- ▶ Delegate are object-oriented, type-safe, and secure function pointers
 - ▶ A delegate instance holds one or more methods
 - ▶ Methods can be static or non-static
- ▶ Delegates allow methods to be passed as parameters.
- ▶ Delegates can be used to define callback methods.



- ▶ Delegates can be chained together; for example, multiple methods can be called on a single event.
- ▶ It is a foundation of event handling.



Creating Delegates

- ▶ The type of a delegate is defined by the name of the delegate

```
public delegate void Print(string s);
```

- ▶ Each delegate is limited to referencing methods of a particular kind only. The type is indicated by the delegate declaration – the input parameters and return type

- ▶ Example:

- ▶ Two methods with the same signature as the delegate declared above

```
public static void printToLower (string s) {
    Console.WriteLine("static:" + s.ToLower());
}
public void printToUpper (string s) {
    Console.WriteLine("instance:" + s.ToUpper());
}
```

Static method

- ▶ Creating a New Delegate Object

- ▶ Use the new operator
- ▶ The argument is the method call, but without the arguments to the method
- ▶ Delegate objects are immutable. (can't change)
- ▶ A delegate can reference static or instance method

```
Print v1 = new Print(printToLower);
Print v2 = new Print(new Program().printToUpper );
```

5

COMPSCI280

Handout13



Using Delegates

- ▶ Once a delegate is instantiated, a method call made to the delegate will be passed by the delegate to that method, and call the underlying method

- ▶ Use the name of the delegate, followed by the parenthesized arguments to be passed to the delegate

Static method

```
v1("This is a test");
v2("This is a test");
```

output:
static:this is a test

output:
instance:THIS IS A TEST

- ▶ It is also possible to programmatically change method calls, and also plug new code into existing classes. As long as you know the delegate's signature, you can assign your own delegated method.

6

COMPSCI280

Handout13



Using Arrays

- ▶ Create an array of delegate objects
- ▶ Instantiate each delegate object with various instance method defined above.
- ▶ Note: In C#, if we reference a method on an object (omitting the parentheses), C# instead treats the method name like a field, returning the object representing that method.
- ▶ Call each delegate object by using a loop -> invoke the underlying method

```
public delegate void Print(string s);
```

```
Print[] arr = new Print[2];
arr[0] = new Print(printToLower);
arr[1] = new Print(new Program().printToUpper);
foreach (Print p in arr)
    p("In an array");
```

static:in an array
instance:IN AN ARRAY

7

COMPSCI280

Handout13



Named & Anonymous methods

- ▶ Named Method

```
public delegate void Print(string s);
```

- ▶ A delegate can be associated with a named method.
- ▶ When you instantiate a delegate using a named method, the method is passed as a parameter
 - The method that you pass as a delegate parameter must have the same signature as the delegate declaration.
 - A delegate instance may encapsulate either static or instance method.

```
Print v1 = printToLower;
v1("This is another test");
Print v2 = new Program(). printToUpper;
v2("This is another test");
```

Named method

- ▶ Anonymous Method

- ▶ In C# 2.0, you can declare a delegate using an anonymous method

```
Print a1 = delegate(string j){
    Console.WriteLine("Anonymous:" + j);
};
a1("Well done");
```

Anonymous method

8

COMPSCI280

Handout13



Multicast delegates

- ▶ Delegate objects can be assigned to one delegate instance to be multicast using the + operator.
- ▶ A delegate can simultaneously reference multiple methods, and it will invoke all underlying methods
 - ▶ Methods are invoked sequentially, in the order added.
- ▶ Conditions:
 - ▶ Only delegates of the same type can be composed.
- ▶ The - operator can be used to remove a component delegate from a composed delegate.

```
public delegate void Print(string s);
```

A delegate object

```
Print mp2 = null;
mp2 += printToLower;
mp2 += new Program().printToUpper;
mp2("Hello World");
```

Output:
static:Hello World
Instance: Hello World

9

COMPSCI280

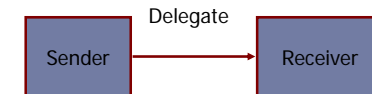
Handout13



Event

Event Basic:

- ▶ An event is a message sent by an object to signal the occurrence of an action.
- ▶ The action could be caused by user interaction, such as a mouse click, or it could be triggered by some other program logic.
- ▶ The object that raises the event is called the **event sender**.
- ▶ The object that captures the event and responds to it is called the **event receiver**.
- ▶ The event sender class does not know which object or method will receive (handle) the events it raises. What is needed is an intermediary (or pointer-like mechanism, delegate) between the source and the receiver.
- ▶ An event is a member of a delegate type that enables an object or class to provide notifications.



10

COMPSCI280

Handout13



Example:

- ▶ Declare a delegate which take a two parameters:
 - ▶ the source that raised the event , and
 - ▶ the data for the event

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

In the Button class

- ▶ Defines a click event of type EventHandler
- ▶ Inside the Button class, the click member is exactly like a field of type EventHandler
- ▶ Outside the Button class, the click member can only be used on the left-hand side of the += and -= operators.
 - ▶ The += operator adds a handler for the event and -= removes a handler for the event

```
public class Button {
    public event EventHandler Click;
    public void Reset() {
        Click = null;
    }
    ...
}
```

11

COMPSCI280

Handout13



Example:

In the Form1 class (windows application)

- ▶ Create a button object
- ▶ Create a event handler method
- ▶ Connect the event handler method to the click event of the button object

Event handler method

```
public Form1() {
    ...
    button1.Click += new EventHandler(button1_Click);
}

public void button1_Click(object sender, EventArgs e){
    Console.WriteLine("Clicked");
}
```

- ▶ You can also reuse the same method for multiple events

```
button2.Click += new EventHandler(button1_Click);
button3.Click += new EventHandler(button1_Click);
```

12

COMPSCI280

Handout13