.NET Programming

Delegates using C#

.NET

Contents:

- Defining Delegates
- Publisher/Subscriber Model
- Multicast Delegates
- Synchronous and asynchronous calls

Farid Naisan, farid.naisan@mah.se

Delegates and Callbacks

.NET

- A delegate is class that holds the address of a single or multiple methods of other classes.
- .NET uses delegates to implement callbacks.
- Delegates accomplish the task of callbacks in a safe and object-oriented way.
- A delegate is an intermediary object between a caller and a target object.
- The methods can be invoked later at run time, *synchronously or asynchronously*.

Farid Naisan, farid.naisan@mah.se

Use of Delegates

- Delegates are used to pass methods to other methods.
- Methods are passed as a parameter in another method.
- A delegate can instantiated as other types.
- There are normally three steps in working with single cast delegates.

Implementation of a delegate

- Defining and using a delegate involves three steps:
  - Declaration
  - Instantiation
  - Method passing
  - Invocation

```
public delegate double calculateHandler(double value1, double value2);
```

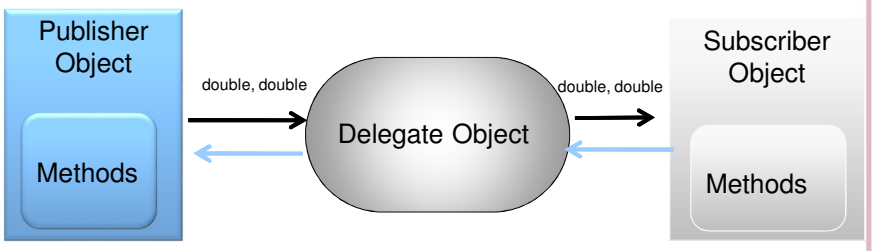
- It is quite common to define the delegate nested in the class that acts as the publisher.

Notifications

- An object's attributes store state of the object.
- Methods change the state.
- There may be objects that would like to be notified when state changes.
  - The object that notifies the "Caller" and more often called as the **Publisher**.
- The objects that are interested in receiving notification are the Target object. Such an object is usually called as the **Subscriber**.

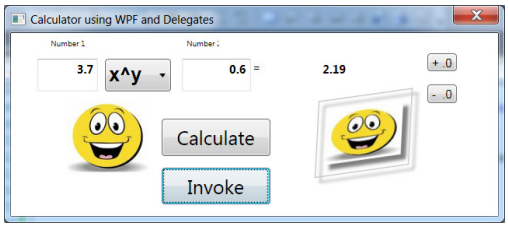
Publish-Subscribe implementation

- The Target registers with the Caller
- The Publisher notifies (calls back) the Subscriber when changes occur.
- The data is passed through the Delegate

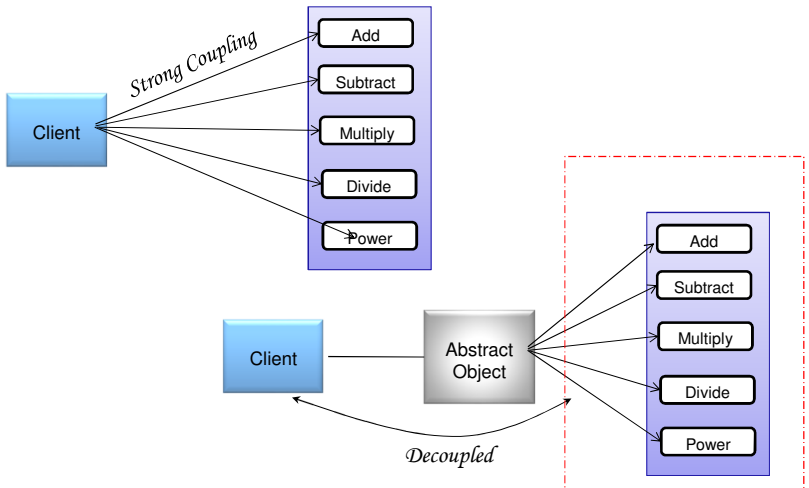


Example

- We would like to program a simple calculator that adds, subtracts, multiplies and divides two double numbers.
- This example is only illustrative and shows the simplest example possible.
- Delegates can be used in much more advanced ways.
  - Sorting with different criteria



Decouple using delegates



A Simple Delegate Example

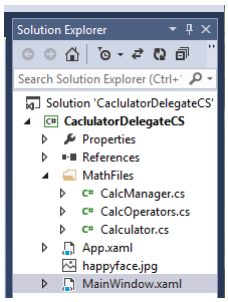
```
public delegate double CalculateHandler(double value1, double value2);

public class CalcManager
{
    // declared but not created
    private CalculateHandler calcDelegate;

    /// Method that performs the required calculation by delegating the job to the delegate CalculateHandler ...
    public double DoCalculate(double value1, double value2, CalcOperators.Operators operation)
    {

        switch (operation)
        {
            case CalcOperators.Operators.Add:
                calcDelegate = new CalculateHandler(Calculator.Add);
                break;
            case CalcOperators.Operators.Substract:
                calcDelegate = new CalculateHandler(Calculator.Subtract);
                break;
            case CalcOperators.Operators.Multiply:
                calcDelegate = new CalculateHandler(Calculator.Multiply);
                break;
            case CalcOperators.Operators.Divide:
                calcDelegate = new CalculateHandler(Calculator.Divide);
                break;
            case CalcOperators.Operators.Power:
                calcDelegate = new CalculateHandler(Calculator.Power);
                break;
            default:
                break;
        }

        //which function? don't know until runtime
        return calcDelegate(value1, value2);
    }
```



Click the Calculate Button!

```
private void btnOK_Click(object sender, RoutedEventArgs e)
{
    if (cmbOperator.SelectedIndex < 0)
        return;

    CalcOperators.Operators operation = (CalcOperators.Operators)cmbOperator.SelectedIndex;

    //Read the user given values
    double value1 = 0.0;
    double value2 = 0.0;

    if (!ReadInput(ref value1, ref value2))
    {
        MessageBox.Show("Values are out of range!");
        return;
    }

    //GUI does not care about how
    calcResult = calcMngr.DoCalculate(value1, value2, operation);
    UpdateResult();

}
```

Using the Invoke method

```
public CalculateHandler GetMethod(CalcOperators.Operators operation)
{
    CalculateHandler calcMethod = null;

    switch (operation)
    {
        case CalcOperators.Operators.Add:
            calcMethod = Calculator.Add;
            break;
        case CalcOperators.Operators.Substract:
            calcMethod = Calculator.Subtract;
            break;
        case CalcOperators.Operators.Multiply:
            calcMethod = Calculator.Multiply;
            break;
        case CalcOperators.Operators.Divide:
            calcMethod = Calculator.Divide;
            break;
        case CalcOperators.Operators.Power:
            calcMethod = Calculator.Power;
            break;
    }

    return calcMethod;
}
```


Declaration

- A delegate declaration includes:
  - The arguments (if any) of this method
  - The return value (if any) of this method

```
public delegate double CalculateHandler(double value1, double value2);
```

- The declaration can appear
  - in a file by itself, or
  - outside a class in another file, or
  - can be nested inside a class

```
public class CalcPublisher {
  public delegate void CalcDataHandler()   'Nested type
```

Single cast and Mulitcast Delegates

.NET

- Delegates can be single cast or multicast
- Single cast delegates point to one method at a time.
- In C#, delegates are multicast meaning that they can point to more than one method at a time.
- The methods are saved in a invocation list.
- The methods all will be called when the delegate is invoked.

Multicast

.NET

- .NET has a whole namespace for multithreading (System.Threading) and the class library provides many facilities
- However, delegates provides this functionality in much a easier and more effective way.
- No need to start or manage a thread to start a process on a secondary thread.

Asynchronous calling

- Use the BeginInvoke and EndInvoke.
- BeginInvoke returns a result which can be used to monitor the progress of the asynchronous call.
- The EndInvoke method retrieves the results of the asynchronous call.
- Both the BeginInvoke and the EndInvoke have parameters that you can use.

Summary

- A delegate is class that holds the address of a single or multiple methods of other classes.
- Delegates are used a lot in .NET.
- In this lesson we talked about a general use of delegates.
- Delegates are the basis of Events.
  - Covered in a separate lesson.