Delegates and Events

Hussam Abu-Libdeh
CS 2026, Spring 2009

Announcements

- Homework assignment due today
  - Submit in CMS
  - Only upload you .cs C# source code file

Previously Covered

- Function parameters: `ref, out, params`
- Iterators
- Advanced C# topics:
  - Nullable types
  - Partial Classes
  - Generics

Today's Agenda

- Delegates
- Anonymous methods
- Events

Motivation – Function Pointers

- Treat functions as “first-class” objects
- Pass functions to other functions
  - Ocaml:

```
map (fun x -> x*x) [1; 2; 3] ;;
```

  - C/C++:

```
typedef int (*fptr) (int);
int apply(fptr f, int var) {
   return f(var);
}
int F(int var) { .. }
fptr f = F;
apply(f,10);  //same as F(10)
```

Delegates

- An objectified function
  - Is a type that references a method
  - Behaves like a C/C++ function pointer
  - Inherits from `System.Delegate`
  - Sealed implicitly
    - i.e. It can not be inherited from
- e.g. `delegate int Foo(int x)`
  - defines a new type `Foo`, takes `int`, returns `int`

Delegates Example

```
delegate int Foo(ref int x);
int Increment(ref int x) {return x++;}
int Decrement(ref int x) {return x--;}
Foo f1 = new Foo(Increment);
f1 += Decrement;
x = 10;
Console.WriteLine(f1(x));
Console.WriteLine(x);
```

- Delegate calls methods in order
  - ref values updated between calls
  - return value is the value of the last call

Delegates Usage Pattern

- Declared like a function
- Instantiated like a reference type
  - Takes a method parameter in constructor
- Modified with +, -, +=, -=
  - Can add multiple instances of a method
  - Removes the last instance of the method in the list
- Called like a function
  - Invoking a delegate that has not been assigned a list of methods causes an exception

List Mapping Example

- ```
  delegate int Foo(int x);

  List<int> Map(Foo f, List<int> list) {
     List<int> result = new List<int>();
     foreach (int element in list) {
        result.Add(f(element));
     }
     return result;
  }
  ```
- Mapper.cs
  ```

Anonymous Methods

- ```
  // f is a delegate
  int y = 10;
  f += delegate(int x) { return x+y; }
  ```
- Creates a method and adds it to delegate
  - Treated the same as other methods
- Variables captured by anonymous method
  - Outer variables
  - e.g. y in the previous example
  ```

Outer Variables

- Local variables declared outside the scope of an anonymous method
  - Captured & remain for the lifetime of the delegate
- Outer variables values are captured once per delegate
- OuterVariables.cs

Events Motivation

- Event-based programming
  - Events are raised by run-time
    - Indirectly through external actions, function calls
  - Client code registers *event handles* to be invoked
    - Also called *callbacks*
    - Allows asynchronous computation
  - e.g. GUI programming

Events in C#

- In C#
  - Events; special delegates
  - Event handles; functions
- Created from delegates using `event` keyword
  - Declares a class member, enabling the class to *raise* events (i.e. to invoke the event delegate)

Events Example

- ```
  public delegate void EventHandler (object source,
  EventArgs e);

  class Room {
     public event EventHandler Enter;
     public void RegisterGuest (object source,
                                EventArgs e) { .. }
     public static void Main(string[] args) {
        Enter += new EventHandler (RegisterGuest);
        if (Enter != null) {
           Enter(this, new EventArgs());
        }
     }
  }
  ```
  ```

Events Usage Pattern

- Enter is an object of type delegate
  - When event is “raised” each delegate called
  - C# allows any delegate to be attached to an event

Events vs Delegates

- Differences from regular delegates
  - Delegates can not be declared in interfaces; events can
  - Can only raise an event in its defining a class
  - Outside can only do += and -=
- To raise events from outside the class
  - Normally methods are used
  - e.g. `Button.OnClick`

Events Accessors

- `add` and `remove` accessors
  - Like `get` and `set` for properties
  - Invoked by += and -= operations
  - Can be explicitly defined for events
  - Normally generated by compiler
- Example
  - When you want to control the space used for storage
  - Or to control accessibility