CS 349 - User Interfaces **Winter 2018**

Lecture 9: January 24, 2018

Lecturer: Keiko Katsuragawa *Notes By: Harsh Mistry*

9.1 Event Binding

Event binding answers how we bind an event event to code after dispatch to a widget

9.1.1 Event-to-Code Binding Approaches

- Event loop + Switch statement "manual" binding
- Switch statement binding
- Inheritance binding
- Listener interface binding
- Listener object binding
- Listener adapter binding.
- Delegate binding (C#)

9.1.1.1 Event Loop and Switch Statement Binding

- All events consumed in one event loop (not by widgets)
- Switch selects window and code to handle the event
- Used in Xlib and many early systems

```
while( true ) {
    XNextEvent(display, &event); // wait next event
    switch(event.type) {
    case Expose:
       // ... handle expose event ...
       cout << event.xexpose.count << endl;
         break;
    case ButtonPress:
    // ... handle button press event ...
       cout << event.xbutton.x << endl;
         break;
    ...
```

Switch Statement Binding: WindowProc

- Each window registers a WindowProc function (Window Procedure) which is called each time an event is dispatched
- The WindowProc uses a switch statement to identify each event that it needs to handle.
  - There are over 100 standard events...

```
LRESULT CALLBACK WindowProc(HWND hwnd, UINT uMsg,
                            WPARAM wParam, LPARAM lParam) {
   switch (uMsg) {
      case WM_CLOSE:
         DestroyWindow (hWnd);
         break;
      case WM_SIZE:
         …
      case WM_KEYDOWN:
         …
```

9.1.1.2 Inheritance Binding

- Event is dispatched to an Object-Oriented (OO) widget
- OO widget inherits from a base widget class with all event handling methods defined a priori onMousePress, onMouseMove, onKeyPress, etc
- The widget overrides methods for events it wishes to handle - Each method handles multiple related events
- Used in Java 1.0

9.1.1.3 Inheritance Problems

- Each widget handles its own events, or the widget container has to check what widget the event is meant for
- Multiple event types are processed through each event method - complex and error-prone, just a switch statement again
- No filtering of events: performance issues
- It doesnt scale well: How to add new events? - e.g. penButtonPress, touchGesture, ...
- Muddies separation between GUI and application logic: event handling application code is in the inherited widget
- Use inheritance for extending functionality, not binding events

9.1.1.4 Delegate Binding

- Interface architecture can be a bit heavyweight

- Can instead have something closer to a simple function callback (a function called when a specific event occurs)
- Delegates in Microsofts .NET are like a C/C++ function pointer for methods, but they:
  - Are object oriented
  - Are completely type checked
  - Are more secure - Support multicasting (able to point to more than one method)

- Using delegates is a way to broadcast and subscribe to events
- .NET has special delegates called events

Using Delegates

1. Declare a delegate using a method signature

```
public delegate void Del(string message);
```

2. Declare a delegate object

```
Del handler;
```

3. Instantiate the delegate with a method

```
// method to delegate (in MyClass)
public static void MyMethod(string message) {
          System.Console.WriteLine(message); }
handler = myClassObject.MyMethod;
```

4. Invoke the delegate

```
handler("Hello World");
```

Multicasting

- Instantiate more than one method for a delegate object

```
handler = MyMethod1 + MyMethod2;
handler += MyMethod3;
```

- Invoke the delegate, calling all the methods

```
handler("Hello World");
```

- Remove method from a delegate object

```
handler -= MyMethod1;
```

- What about this?

```
handler = MyMethod4;
```

Events in .NET

- Events are a delegate with restricted access
- Declare an event object instead of a delegate object:

```
public delegate void Del(Object o, EventArgs e);
event Del handler;
```

- "event" keyword allows enclosing class to use delegate as normal, but outside code can only use the -= and += features of the delegate
- Gives enclosing class exclusive control over the delegate
- Outside code can't wipe out delegate list, can't do this:

```
handler = MyMethod4;
```

- Can have anonymous delegate events (similar to Java style):

```
b.Click += delegate(Object o, EventArgs e) {
      Windows.Forms.MessageBox.Show("Click!");  };
```