
CSE 143 Java

Adapter Classes & Inner Classes

Reading: Ch. 17

10/20/2004

(c) 2001-4, University of Washington

09-1

Overview

- Adapter Classes
- Inner Classes
 - Named
 - Anonymous

10/20/2004

(c) 2001-4, University of Washington

09-2

Event Handling Evaluated

- The bouncing ball simulator had a couple of awkward features
 - The class implementing `MouseListener` had to implement all of the methods in that interface, even though it was only interested in mouse click events
 - The mouse and button listeners were separate classes from the controller, yet they were closely intertwined (high coupling)
 - All had to know about the simulated world
 - The listener classes introduced unneeded top-level class names
- Can clean this up considerably using adapter classes and inner classes

10/20/2004

(c) 2001-4, University of Washington

09-3

Adapter Classes

- Problem: Many of the event handling interfaces have several methods (`MouseReleased`, `MouseClicked...`), but user may only be interested in 1 or 2, not all 5 or 10
- Solution: Most of these interfaces have an associated *adapter class* that contains empty implementations of all the methods in the interface
 - (Not provided for `ActionListener`, since it has only one method)
- Idea: Extend the adapter class and override the interesting methods
 - Inherit the empty implementations of the methods you don't care about

10/20/2004

(c) 2001-4, University of Washington

09-4

MouseListener & Mouse Adapter

- Old code

```
class SimMouseListener
    implements MouseListener {
    /** process mouse click */
    public void mouseClicked(MouseEvent e) {
        world.add(randomBall(e.getX(), e.getY()));
    }
    // other events in mousetlistener
    public void mouseEntered(MouseEvent e) {}
    public void mouseExited(MouseEvent e) {}
    public void mousePressed(MouseEvent e) {}
    public void mouseReleased(MouseEvent e) {}
}
```

- New code

```
class SimMouseListener
    extends MouseAdapter {
    /** process mouse click */
    public void mouseClicked(MouseEvent e) {
        world.add(randomBall(e.getX(), e.getY()));
    }
}
```

- Same functionality, less typing

10/20/2004

(c) 2001-4, University of Washington

09-5

Inner Classes

- The mouse and button listeners are tightly coupled to the BallSimControl class
- Idea: would like these listeners to have direct access to the instance variables of BallSimControl
- Solution: *inner classes*
 - Declare the mouse and button listener classes *inside* the BallSimControl class
 - Code in inner classes has the same access to instance variables as code in methods in class BallSimControl
 - If the inner class is an implementation detail of the outer class, make the inner class *private*

10/20/2004

(c) 2001-4, University of Washington

09-6

Mouse Listener as an Inner Class

```
/** Viewer/controller for ball world simulation */
public class BallSimControl extends JPanel implements SimView {
    // instance variables
    private SimModel world; // the simulation world we are controlling
    ...
    /** Handle mouse events */
    private class SimMouseListener extends MouseAdapter {
        /** process mouse clicks */
        public void mouseClicked(MouseEvent e) {
            world.add(randomBall(e.getX(), e.getY()));
        }
    }
}
```

- Notice direct use of `world` – no separate instance variable needed in SimMouseListener to keep track of this

10/20/2004

(c) 2001-4, University of Washington

09-7

Anonymous Inner Classes

- We only create one instance of the mouse listener

```
SimMouseListener mouseListener = new SimMouseListener();
viewPane.addMouseListener(mouseListener);
```
- Maybe we don't even need to give this class a name(!)
- In Java you can create *anonymous* inner classes
 - Particularly useful in situations where we want to extend an adapter and create a "function object" – an object that encapsulates a function like a MouseClick listener method

- **WARNING!!!** Ghastly syntax ahead

10/20/2004

(c) 2001-4, University of Washington

09-8

Syntax for an Anonymous Inner Class

- Idea: a single construct replaces both the class definition and the “new” operation that creates a single instance of it

```
new classname ( constructor_parameters_if_any ) {  
    methods  
}
```

- This creates a new instance of an anonymous class that extends *classname*
- The methods in the class body can override methods declared in *classname*

10/20/2004

(c) 2001-4, University of Washington

09-9

Anonymous Inner Class for Mouse Listener

- Instead of defining `SimMouseListener` and creating an instance, replace

```
viewPane.addMouseListener(new SimMouseListener());
```

in `BallSimControl` with

```
viewPane.addMouseListener(  
    new MouseListener() { // anon. inner class extending MouseListener  
        public void mouseClicked(MouseEvent e) { // override mouseClicked  
            world.add(randomBall(e.getX(), e.getY()));  
        }  
    } // end of anon. inner class  
);
```

- This is the conventional indentation; helps readability a bit

10/20/2004

(c) 2001-4, University of Washington

09-10

Summary

- Adapter classes – empty implementations of interfaces that can be extended when only a few methods in the interface are needed
- Inner classes
 - Powerful programming technique – allows tightly coupled classes (“helper classes”) to interact cleanly
 - Can be named or anonymous (if extending some other class)
 - Can be abused to create horribly complex code
 - My advice: use when (and **only** when) they simplify things (for you, for the reader)

10/20/2004

(c) 2001-4, University of Washington

09-11