Swing II: Layout and Inner Classes
Announcements

• Friday is the BONUS lecture

• HW10 is available:
  – due Tuesday, April 23rd at 11:59:59pm
HW 10: Game projects
Swing Programming Demo

Layout & Wiring
Inner Classes
Inner Classes

• Useful in situations where two objects require “deep access” to each other’s internals

• Replaces tangled workarounds like “owner object” (as in the drawing example)
  – Solution with inner classes is easier to read
  – No need to allow public access to instance variables of outer class

• Also called “dynamic nested classes”
Key idea: Classes can be *members* of other classes...

```java
public class Outer {
    private int outerVar;
    public Outer () {
        outerVar = 6;
    }

class Inner {  
    private int innerVar;
    public Inner(int z) {
        innerVar = outerVar + z;
    }
}
}
```

Name of this class is `Outer.Inner` (which is also the static type of objects that this class creates)

Reference from inner class to instance variable bound in outer class
Object Creation

• Inner classes can refer to the instance variables and methods of the outer class

• Inner class instances usually created by the methods/constructors of the outer class

```java
public Outer () {
    Inner b = new Inner ();
}
```

• Inner class instances *cannot* be created independently of a containing class instance.

```java
Outer.Inner b = new Outer.Inner();
Outer a = new Outer();
Outer.Inner b = a.new Inner();
Outer.Inner b = (new Outer()).new Inner();
```

Actually this.new
Inner classes

DrawingExample Constructor

```java
b1.addActionListener(new DrawingButtonListener(b1));
b2.addActionListener(new DrawingButtonListener(b2));
```

Inner Class

```java
class DrawingButtonListener implements ActionListener{
    JButton button;
    DrawingButtonListener(JButton b)
    {
        button = b;
    }

    public void actionPerformed(ActionEvent e) {
        // Find out which button generated the event
        if (button.equals(b1)) {
            shapes.add(new Line());
        } else if (button.equals(b2)) {
            shapes.add(new Square());
        }
    }
}
```

Button action code far from button creation

Awkward logic to avoid one class per button
Anonymous Inner Classes

- Define a class and create an object from it all at once, inside a method

```java
b1.addActionListener(new ActionListener() {  
    public void actionPerformed(ActionEvent e) {  
        shapes.add(new Line());  
        drawingCanvas.repaint();  
    }  
});
```

- Can access fields and methods of outer class

```java
b2.addActionListener(new ActionListener() {  
    public void actionPerformed(ActionEvent e) {  
        shapes.add(new Square());  
        drawingCanvas.repaint();  
    }  
});
```

- Puts button action right with button definition
- Each button gets its own inner class
Anonymous Inner class

- New *expression* form: define a class and create an object from it all at once

```java
new InterfaceOrClassName() {
    public void method1(int x) {
        // code for method1
    }
    public void method2(char y) {
        // code for method2
    }
}
```

- Static type of the expression is the Interface/superclass used to create it
- Dynamic class of the created object is anonymous!
- Can't really refer to it.
Like first-class functions

• Anonymous inner classes are the Java equivalent of Ocaml first-class functions

• Both create "delayed computation" that can be stored in a data structure and run later
  – Code stored by the event / action listener
  – Code only runs when the button is pressed
  – Could run once, many times, or not at all

• Both sorts of computation can refer to variables in the current scope
  – OCaml: Any available variable
  – Java: only instance variables (fields) and variables marked final