Programming Languages and Techniques (CIS120)

Lecture 34
April 11, 2016

Swing II: Inner Classes and Layout
Announcements (Review)

• Final exam: May 2, 3-5PM
  – If you have two finals at the same time, you can postpone one
  – If you have 3 finals on one day, you may postpone the middle one
  – If either case applies to you, add your name to the form on the website
  – Must discuss all other exam conflicts with course instructors
Game project (Review)

• Game Design Proposal Milestone Due: (12 points)  
  Friday April 13th at NOON!!!!
  – (Should take about 1 hour)
  – Submit on GRADESCOPE
  – TAs will give you feedback over the weekend

• Final Program Due: (88 points)  
  Wednesday, April 25th at 11:59pm
  – Submit zipfile online, submission only checks if your code compiles
  – Eclipse is STRONGLY recommended for this project
  – May distribute your game (after the deadline) if you do not use any of our code

• Grade based on demo with your TA during reading days
  – Grading rubric on the assignment website
  – Recommendation: don’t be too ambitious.

• **NO LATE SUBMISSIONS PERMITTED**
How to have first-class computation?

class ButtonListener implements ActionListener {
    private LightBulb bulb;
    public ButtonListener (LightBulb b) {
        bulb = b;
    }
    @Override
    public void actionPerformed(ActionEvent e) {
        bulb.flip();
        bulb.repaint();
    }
}

// somewhere in run ...
LightBulb bulb = new LightBulb();
JButton button = new JButton("On/Off");
button.addActionListener(new ButtonListener(bulb));

let bulb, bulb_flip = make_bulb ()
let onoff,_, bnc = button "ON/Off"
;; bnc.add_event_listener (mouseclick_listener bulb_flip)
public void run() {
    // ... other code omitted
    LightBulb bulb = new LightBulb();
    panel.add(bulb);

    JButton button = new JButton("On/Off");
    panel.add(button);

    button.addActionListener(new ButtonListener(bulb));

    // ... other code omitted
}
Too much boilerplate code...

• ButtonListener really only needs to do bulb.flip()

• But we need all this boilerplate code to build the class

• Often we will only instantiate one instance of a given Listener class in a GUI

```java
class ButtonListener implements ActionListener {
    private LightBulb bulb;
    public ButtonListener (LightBulb b) {
        bulb = b;
    }

    @Override
    public void actionPerformed(ActionEvent e) {
        bulb.flip();
        bulb.repaint();
    }
}
```
Inner Classes
Inner Classes

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

• Replaces tangled workarounds like “owner object”
  – 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
class Outer {
    private int outerVar;
    public Outer () {
        outerVar = 6;
    }
    public class Inner {
        private int innerVar;
        public Inner(int z) {
            innerVar = outerVar + z;
        }
        public int getInnerVar() {
            return innerVar;
        }
    }
}
```

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
Constructing Inner Class Objects

Based on your understanding of the Java object model, which of the following make sense as ways to construct an object of an inner class type?

1. `Outer.Inner obj = new Outer.Inner(2);`
2. `Outer.Inner obj = (new Outer()).new Inner(2);`
3. `Outer.Inner obj = new Inner(2);`
4. `Outer.Inner obj = Outer.Inner.new(2);`

Answer: 2 – the inner class instances can refer to non-static fields of the outer class (even in the constructor), so the invocation of "new" must be relative to an existing instance of the Outer class.

```java
class Outer {
    private int outerVar;
    public Outer () {
        outerVar = 6;
    }
    public class Inner {
        private int innerVar;
        public Inner(int z) {
            innerVar = outerVar + z;
        }
        public int getInnerVar() {
            return innerVar;
        }
    }
}
```
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

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

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

    Outer.Inner b = new Outer.Inner();

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

    Outer.Inner b = (new Outer()).new Inner();
public class Demo {
    private JLabel label1 = new JLabel("a label");

    void m(JLabel label2) {
        JLabel label3 = new JLabel("another label");

        JButton button = new JButton("button");
        button.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {

                label1.setText("label1"); // 1
                label2.setText("label2"); // 2
                label3.setText("label3"); // 3

            }
        });
    }
}
Anonymous Inner Classes

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

```
final LightBulb bulb = new LightBulb();
JButton button = new JButton("On/Off");

button.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        bulb.flip();
        bulb.repaint();
    }
});
```

Puts button action right with button definition

Can access fields and methods of outer class, as well as final local variables
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 refer to it.
Like first-class functions

• Anonymous inner classes are the real 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
Lambda expressions: Java 8

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

```java
final LightBulb bulb = new LightBulb();
JButton button = new JButton("On/Off");

button.addActionListener((e) -> {
    bulb.flip();
    bulb.repaint();
});
```

Can access fields and methods of outer class, as well as final local variables
Swing Programming Demo

LayoutDemo.java
What components would you use for this app?
GameCourt
subclass of
JPanel
(court)

JPanel
(control_panel)

JPanel
(status_panel)

JButton
(reset)

JLabel (status)
Did you attend lecture today?

1. yes
2. yes
3. yes
4. yes
What layout would you use for this app? What components would you use?
Canvas subclass of JPanel (canvas)

JPanel (toolbar)

JRadioButton (point, line)

JCheckbox (thick)

JButton (quit)