Programming Languages and Techniques (CIS120)

Lecture 34
April 22, 2019

Swing II: Inner Classes and Layout
Chapter 30
Announcements

• Completed Game Projects
  – Due: May 1\textsuperscript{st}
  – \textit{NO LATE SUBMISSIONS}
  – \textit{Grading will be by demo. Stay tuned for details!}
Swing

Java’s GUI Library
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)
Too much “boilerplate”!

- ButtonListener really only needs to do bulb.flip() and repaint
- But we need all this extra 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 objects require “deep access” to each other’s internals

• Replaces tangled workarounds like the “owner object” pattern
  – 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 (i.e., the static type of objects that this class creates) is Outer.Inner

Reference from inner class to field bound in outer class
Java Inner Classes

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;
        }
    }
}

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

Actually `this.new`
Anonymous Inner Classes

• 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(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        bulb.flip();
        bulb.repaint();
    }
});
```
Anonymous Inner Classes

```java
quit.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        System.exit(0);
    }
});
```

Puts button action with button definition

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

Can access fields and methods of outer class, as well as final local variables
Anonymous Inner Classes

- **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 a Java equivalent of OCaml’s first-class functions

• Both create "delayed computations" 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 variables marked final
Lambda Expressions

- Java 8 introduced "lambda expressions" which are simplified syntax for anonymous classes with "functional interfaces" with just one method

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

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

LayoutDemo.java