Programming Languages and Techniques (CIS120e)

Lecture 30
Nov. 22, 2010

Swing II

HW10

• Available Wednesday; Due Friday, Dec 10
• We give you code for a simple Pong game
• Goal: Build a GUI for a game of your choice
• Two options:
  – Do a few reasonably straightforward, well-specified extensions of Pong, for up to 70% credit
  – Or, for more points, go a little further and design your own game...

Some ideas...

Breakout
Some ideas...

Space Invaders

Some ideas...

Reversi / Connect Four

Some ideas...

Tetris

Layouts
Flow Layout

- Items just “flow”
- Layout manager places them one after another
- Swing’s default layout

This and following examples taken from “The Swing Tutorial”

Grid Layout

- Places components in a grid of cells.
- Each component takes all the available space within its cell,
- Each cell is exactly the same size.

Border Layout

More Sophisticated Managers

- Include:
  - BoxLayout
  - CardLayout
  - GridBagLayout
  - GroupLayout
  - SpringLayout
- These are harder to use when coding by hand
- Better way: Use a layout tool such as NetBeans
  - GUIs that write code for GUIs!
Time for some more demos...

Nested Classes

Key idea: Classes can be *members* of other classes...

```java
public class Outer {
    public int outerVar;
    public Outer () {
        outerVar1 = 6;
    }
    public class Inner {
        public int innerVar;
        public Inner(int z) {
            innerVar = outerVar+z;
        }
    }
}
```

Reference from inner class to variable bound in outer class
**Dynamic vs. Static Nested Classes**

Two distinct kinds of nested classes...

- **Dynamic nested classes (also called inner classes):**
  ```java
class A {
    class B { ... }
    ...
}
```

- **Static nested classes:**
  ```java
class A {
    static class B { ... }
    ...
}
```

**Static nested classes**

- Not very different from plain old classes...
  - Logically group classes that are only used in one place
  - More fine-grained access control than public/private
  - Put definitions of small classes near where they are used

- We won’t say more
  - *Interesting and somewhat subtle scope issues*...

**Dynamic Nested 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
Details

- Inner class instances cannot be created independently of a containing class instance.
  
  ```java
  Outer.Inner b = new Outer.Inner();
  ```

  ![Error]

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

  ![Correctness]

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

Anonymous Inner Classes

- Remember OCaml’s “anonymous functions”?
  
  ```ocaml
  map (fun x -> x^3) [1;2;3]
  ```

- Java has something similar: *anonymous inner classes*

- Same motivation in both cases:
  
  - local function definitions (in OCaml) and inner classes (in Java) are often used in only one place
  
  - ... so it’s silly to give them a name — better just to just directly write the function/class *itself* at the point where it’s needed!

Recall...

```java
class TimerAction implements ActionListener {
  JButton button;
  TimerAction (JButton b) { button = b; }
  public void actionPerformed(ActionEvent e) {
    Color f = button.getBackground();
    Color b = button.getForeground();
    button.setForeground(f);
    button.setBackground(b);
  }
}
```

```java
class OnOff6 {
  public OnOff6() {
    Timer timer = new Timer(1000, new TimerAction(button));
    timer.start();
  }
  ...
}
```
Better...

class AnonExample {
    public AnonExample() {
        final JButton button = new JButton("On/Off");
        ...
        Timer timer = new Timer(1000, new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                Color f = button.getBackground();
                Color b = button.getForeground();
                button.setForeground(f);
                button.setBackground(b);
            }
        });
        timer.start();
    }
    ...
}

Points to note...

- Syntax:
  
  ```java
  new interfacename() {
      method definitions
  }
  ```
  
  - Other forms also available, but this is enough for present purposes

- Local variables (like `button` here) that are referenced from the inner class must be declared `final`
  
  - reasons have to do with the fact that inner classes were added to Java late in its development

Back to the demos...

See DrawingExample2.java and friends