Programming Languages and Techniques (CIS120e)

Lecture 32

Nov. 29, 2010

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 {
    public int outerVar;
    public Outer () {
        outerVar = 6;
    }

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

```java
... Outer o = new Outer();
Outer.Inner i = o.new Inner();
...
```

Reference from inner class to variable bound in outer class

The type of inner-class instances
• Inner classes can refer to the instance variables and methods of the outer class
Creation

- Each inner class instance is created relative to a containing class instance.

  ```java
  Outer.Inner b = new Outer.Inner(); // Incorrect
  Outer a = new Outer();
  Outer.Inner b = a.new Inner(); // Correct
  Outer.Inner b = (new Outer()).new Inner(); // Correct
  ```
Inner Classes at Run Time

heap

Outer$Inner

owner:

innerVar: 7

Outer

outerVar: 6

Outer$Inner

owner:

innerVar: 4

Inner class instances with implicit pointers to "owning" outer class object

Outer class instance
Anonymous Inner Classes

• Remember OCaml’s “anonymous functions”?
  \[
  \text{map \ (fun x -> x*3) \ [1;2;3]}
  \]

• Java has something similar: \textit{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 \textit{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.setForeground();
        button.setForeground(f);
        button.setBackground(b);
    }
}

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