Programming Languages and Techniques (CIS120)

Lecture 34

April 10, 2012

Swing III: Inner Classes and OO refactoring

Announcements

- HW 9 due tonight at 11:59:59pm
  - Lab today is only GUI lab (i.e. save HW questions for OH)

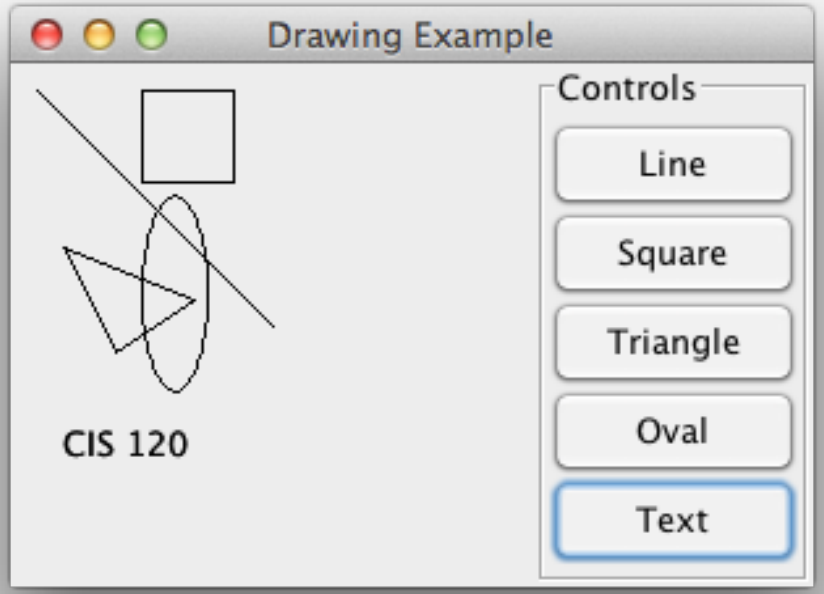
- HW 10 (Game Project) is available
  - Due Tuesday April 24th at 11:59:59pm (last day of classes)
  - Demo code includes inner classes (covered today) and mouse/ keyboard input (covered on Monday)

- Bonus Lecture on Friday
  - *Consequences of "Code is Data"*
  - Not covered on homework or final exam

- Max Scheiber's band is playing at Fling, 2:45 in the quad

Swing Example

A case study in organizing GUI Applications



GUI Design Pattern

- Separate Graphical Applications into three components
- Model
  - The “state” or data of the Application
  - “Toplevel” class, such as “DrawingExample”
- View
  - How that state is presented to the user (could be in different ways)
  - DrawingExampleCanvas
- Controller
  - How users interact with the model
  - Swing components such as buttons and their event listeners
  - DrawingExampleListener

Refactoring for Extensibility

Already saw one example of refactoring:

We replaced these five fields of DrawingExample:

```
public boolean drawLine     = false;
public boolean drawSquare   = false;
public boolean drawTriangle = false;
public boolean drawOval     = false;
public boolean drawText     = false;
```

with this one:

```
public List<Shape> shapes = new LinkedList<Shape>();
```

Make canvas more flexible

```
public void paintComponent(Graphics gc) {
    super.paintComponent(gc);
    if (owner.drawLine) {
        gc.drawLine(10, 10, 100, 100);
    }
    if (owner.drawSquare) {
        gc.drawRect(50, 10, 35, 35);
    }
    …
```

Canvas does not need to know how to draw all of the shapes.

```
public void paintComponent(Graphics gc) {
    super.paintComponent(gc);
    for (Shape shape : owner.shapes) {
        shape.draw(gc);
    }
}
```

What about the action listener?

```
public void actionPerformed(ActionEvent e) {
    // Find out which button generated the event,
    if (button.equals(owner.b1)) {
        owner.shapes.add(new Line());
    } else if (button.equals(owner.b2)) {
        owner.shapes.add(new Square());
    } else if (button.equals(owner.b3)) {
        owner.shapes.add(new Triangle());
    } else if (button.equals(owner.b4)) {
        owner.shapes.add(new Oval());
    } else if (button.equals(owner.b5)) {
        owner.shapes.add(new Text());
    }
    // Notify Swing that the drawing panel needs to
      be repainted
    owner.drawingCanvas.repaint();
}
```

What about redundant code?

```
// Create the buttons
JButton b1, b2, b3, b4, b5;
b1 = new JButton("Line");
b2 = new JButton("Square");
b3 = new JButton("Triangle");
b4 = new JButton("Oval");
b5 = new JButton("Text");

// Attach actions to the buttons.
b1.addActionListener(
   new DrawingExample1bListener(this, new Line()));
b2.addActionListener(
   new DrawingExample1bListener(this, new Square()));
b3.addActionListener(
   new DrawingExample1bListener(this, new Triangle()));
b4.addActionListener(
   new DrawingExample1bListener(this, new Oval()));
b5.addActionListener(
   new DrawingExample1bListener(this, new Text()));
```

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”

```
public class DrawingExample implements Runnable {
   public List<Shape> shapes = new LinkedList<Shape>();
   private DrawingPanel drawingPanel;
   public void run() {
       JFrame frame = new JFrame("Drawing Example");
       drawingPanel = new DrawingPanel(this);
   }
}
class DrawingCanvas extends JComponent {
   private DrawingExample owner;
   public DrawingCanvas (DrawingExample p) { owner = p; }
   public void paintComponent(Graphics gc) {
        super.paintComponent(gc);
        for (Shape shape : owner.shapes) {
            shape.draw(gc);
        }
   }
}
```

Without Inner classes

Each class has a reference to the other

Needs to access toplevel field

```
public class DrawingExample implements Runnable {
   public List<Shape> shapes = new LinkedList<Shape>();
   private DrawingPanel drawingPanel;
   public void run() {
       JFrame frame = new JFrame("Drawing Example");
       drawingPanel = new DrawingPanel();
   }

   class DrawingCanvas extends JComponent {
    public DrawingCanvas () {  }
    public void paintComponent(Graphics gc) {
        super.paintComponent(gc);
        for (Shape shape : shapes) {
            shape.draw(gc);
        }
   }
}
```

With Inner classes

shapes is private

No explicit reference to frame from canvas

Inner class can access toplevel private members directly

Basic Example

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

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

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

Actually this.new

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

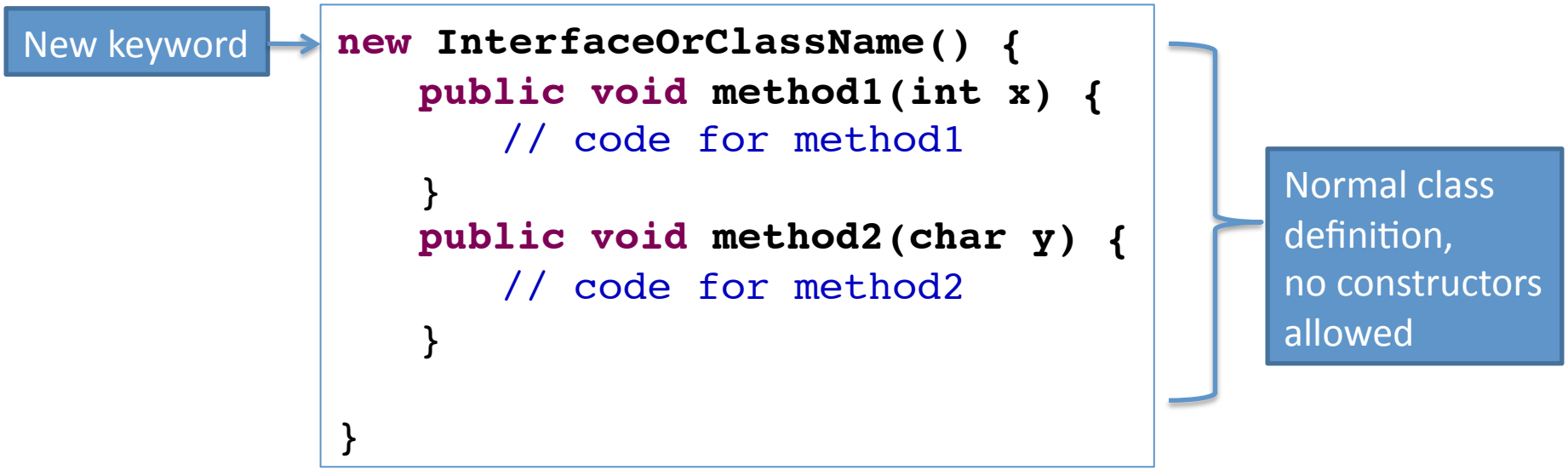
```
Outer.Inner b = new Outer.Inner()
```

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

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

Anonymous Inner class

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



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