1.00 Lecture 20

Anonymous Inner Classes
Model-View-Controller

Reading for next time: Big Java: review sections 4.3-4.10

Anonymous Inner Classes

- **Java offers a shortcut for creating new classes**
  - **Used when writing small, simple classes.**
  - **Helpful for writing event listeners. Because they are very common, this approach makes writing them simpler.**
- **To see how anonymous inner classes can be used, look at the `SwitchButton` class.**

SwitchButtonFrame

```
import javax.swing.*;
import java.awt.*;

public class SwitchButtonFrame extends JFrame {
   public SwitchButtonFrame() {
        SwitchButton b = new SwitchButton("Click me!",
                Color.BLUE, Color.WHITE);
        Container c = getContentPane();
        c.add(b);
        pack();    // Pack has JFrame resize to fit components
   }
   public static void main(String[] args) {
        SwitchButtonFrame f = new SwitchButtonFrame();
        f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        f.setVisible(true);
   }
}
```

SwitchButton

```
import javax.swing.*;
import java.awt.event.*;
import java.awt.*;

public class SwitchButton extends JButton implements
   ActionListener {
   public SwitchButton(String text, Color b, Color f) {
        super(text);
        setBackground(b);
        setForeground(f);
        addActionListener(this);
   }
   // Button listens to itself (no panel here to do it)
   public void actionPerformed(ActionEvent ae) {
        Color fore = getForeground();
        setForeground(getBackground());
        setBackground(fore);
   }
}
```

AnonSwitchButton

- This code is functionally equivalent and can be used with the same `SwitchButtonFrame`, but it uses an anonymous inner class to serve as the `ActionListener`.

```
// Same import statements
public class AnonSwitchButton extends JButton {
   public AnonSwitchButton(String t, Color b, Color f) {
     super(t);
     setBackground(b);
     setForeground(f);
     addActionListener(new ActionListener() {
       public void actionPerformed(ActionEvent ae) {
         Color fore = getForeground();
         setForeground(getBackground());
         setBackground(fore);
     });
  }
}
```

SwitchButton

- `SwitchButton` no longer implements `ActionListener` or possesses an `actionPerformed` method.
- The anonymous inner classes takes over that function.
- It looks as if we are newing an interface, which would be illegal. Actually we are creating an nameless class that will only have a single instance, the one we create right here for the `addActionListener()` method.
- The new constructor call can NOT have arguments.

```
addActionListener(
  new ActionListener() { . . . }
);
```

- The anonymous inner class has access to its enclosing class' data members and methods, so it doesn't need arguments.
- Anonymous inner classes are helpful when there are many event sources—we can have one class per event source

Clock: Anonymous Inner Classes

- Rewrite Clock example from lecture 15 using two anonymous inner classes.
  - Download ClockFrame and ClockPanel
- Create an anonymous inner class for tickButton and for resetButton.
- ClockPanel no longer needs to implement ActionListener
  - Remove actionPerformed()
- Once you have a solution, run ClockFrame and verify that it works.

ClockFrame

```
import java.awt.*;
import javax.swing.*;

// Nothing needs to be changed in ClockFrame
public class ClockFrame extends JFrame{
    public ClockFrame() {
        super("Clock Test");
        Container contentPane= getContentPane();
        ClockPanel clock = new ClockPanel();
        contentPane.add(clock);
    }

    public static void main(String[] args) {
        ClockFrame frame = new ClockFrame();
        frame.setSize(300, 200);
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setVisible(true);
    }
}
```

ClockPanel

```
import java.awt.*;
import javax.swing.*;
import java.awt.event.*;

public class ClockPanel extends JPanel implements ActionListener {
    private JButton tickButton, resetButton;
    private JLabel hourLabel, minuteLabel;
    private int minutes = 720;                // 12 noon

    public ClockPanel(){
        JPanel panel = new JPanel();
        tickButton = new JButton("Tick");
        resetButton = new JButton("Reset");
        hourLabel = new JLabel("12:");
        minuteLabel = new JLabel("00");
        panel.add(tickButton);
        panel.add(resetButton);
        panel.add(hourLabel);
        panel.add(minuteLabel);
        setLayout(new BorderLayout());
        add(panel, BorderLayout.SOUTH);
        tickButton.addActionListener(this); // Change these 2 lines
        resetButton.addActionListener(this);
```

Remove

ClockPanel, p.2

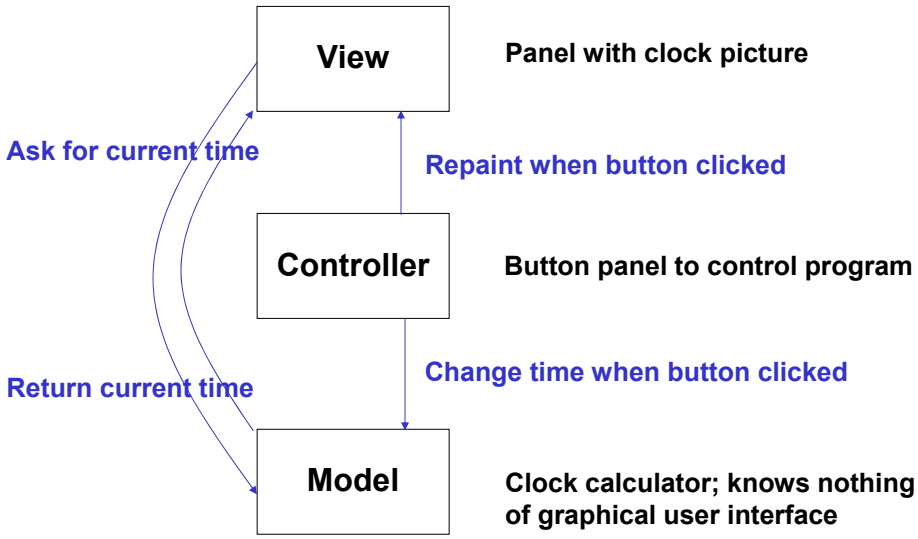
```
// Remove this method and replace it with anonymous classes
public void actionPerformed(ActionEvent e) {
   public void actionPerformed(ActionEvent e) {
        if(e.getSource().equals(tickButton))
                minutes++;
        else
                minutes = 720;          // 12 noon
        repaint();      // Repaint redraws circle and lines
        setLabels();    // setLabels resets hour, minute text
   }
```

ClockPanel, p.3

```
// These two methods are unchanged
public void paintComponent(Graphics g) {
        super.paintComponent(g);
        g.drawOval(100, 0, 100, 100);
        double hourAngle = 2 * Math.PI * (minutes - 3 * 60)/ (12 * 60);
        double minuteAngle = 2 * Math.PI * (minutes - 15)/ 60;
        g.drawLine(150, 50, 150 + (int)(30 * Math.cos(hourAngle)),
                50 + (int)(30 * Math.sin(hourAngle)));
        g.drawLine(150, 50, 150 + (int)(45 * Math.cos(minuteAngle)),
                50 + (int)(45 * Math.sin(minuteAngle)));
 }

 public void setLabels(){             // Doesn't handle midnight
        int hours = minutes/60;
        int min = minutes - hours*60;
        hourLabel.setText(hours+ ":");
        if(min<10)                    // Minutes should be two digits
            minuteLabel.setText("0" + min);
        else
            minuteLabel.setText("" + min);  }
}
```

Clock: Model-view-controller



Clock Model

```
// Notice no import javax.swing.*; or java.awt.*;
// No references or knowledge of view or controller

public class ClockModel {
   private int minutes;

   public ClockModel(int m) {
        minutes = m;
   }
   public int getMinutes() {
        return minutes;
   }
   public void setMinutes(int m) {
        minutes = m;
   }
   public int advance() {
        minutes++;
        return minutes;
   }
}
```

Clock View

```
import java.awt.* ;
import javax.swing.* ;

public class ClockView extends JPanel {
    private ClockModel model ;       // Needs reference to model!
    public ClockView( ClockModel cm ) { model = cm ;}

    public void paintComponent(Graphics g) {
        super.paintComponent(g);
        double minutes= model.getMinutes();  // Uses model
        g.drawOval(100, 0, 100, 100);
        double hourAngle = 2 * Math.PI * (minutes - 3*60) / (12 * 60);
        double minuteAngle = 2 * Math.PI * (minutes - 15) / 60;
        g.drawLine(150,50,150 + (int) (30 * Math.cos(hourAngle)),
                50 + (int) (30 * Math.sin(hourAngle)));
        g.drawLine(150,50,150 + (int) (45 * Math.cos(minuteAngle)),
                50 + (int) (45 * Math.sin(minuteAngle)));
   }
}
```

Clock Controller, p.1

```
import java.awt.* ;
import java.awt.event.* ;
import javax.swing.* ;

public class ClockController extends JFrame {
   private JLabel hourLabel, minuteLabel ;
   private JButton tickButton, resetButton ;
   private JPanel buttonHolder ;
   private Container canvas ;
   private ClockView drawArea ;    // Reference to view
   private ClockModel clock ;      // Reference to model

   public ClockController() {
        canvas = getContentPane() ;
        canvas.setLayout( new BorderLayout() ) ;
        setSize(200, 300 ) ;  setTitle( "MVC Clock" ) ;
        buttonHolder = new JPanel() ;  // Create button holder
        canvas.add( buttonHolder, BorderLayout.SOUTH ) ;
        tickButton = new JButton("Tick");
        resetButton = new JButton("Reset");
        hourLabel = new JLabel("12:");
        minuteLabel = new JLabel("00");
```

Clock Controller, p.2

```
      buttonHolder.add(tickButton);
      buttonHolder.add(resetButton);
      buttonHolder.add(hourLabel);
      buttonHolder.add(minuteLabel);

      clock= new ClockModel(720);          // Creates model object
      drawArea= new ClockView(clock);      // Creates view object
      canvas.add( drawArea, BorderLayout.CENTER ) ;
      drawArea.repaint() ;                 // Adds view to canvas,
                                           // same as buttonHolder
      tickButton.addActionListener(new ActionListener() {
        public void actionPerformed(ActionEvent ae) {
          clock.advance(); drawArea.repaint(); // Use model,view
          setLabels(); }
      });

      resetButton.addActionListener(new ActionListener() {
        public void actionPerformed(ActionEvent ae) {
          clock.setMinutes(720); drawArea.repaint(); // Use model,vw
          setLabels(); }
      });
}          // End constructor
```

Clock Controller, p.3

```
    public void setLabels() {  // Doesn't handle midnight
         int hours = clock.getMinutes() / 60;
         int min = clock.getMinutes() - hours * 60;
         hourLabel.setText(hours + ":");
         if (min < 10)          // Minutes should be two digits
                minuteLabel.setText("0" + min);
         else
                minuteLabel.setText("" + min);
    }

    public static void main(String[] args) {
      ClockController application = new ClockController() ;
      application.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE) ;
      application.setVisible(true) ;
    }
}
```

Model-view-controller

- **Model: computational (CatenaryModel in hw 6-7)**
  - **Only knows how to compute the solution**
  - **Doesn't know how to draw**
  - **Doesn't know about events, or the GUI at all**
- **View: purely display of results(CatenaryView in hw)**
  - **Only knows how to draw**
  - **Doesn't know how to compute the solution**
  - **Doesn't know about events**
- **Controller: manages events (CatenaryController…)**
  - **Manages startup (construction), object creation, events, repaints, label refreshes, exit, …**
  - **Doesn't know how to draw**
  - **Doesn't know how to compute**

Exercise

- **Download and modify the clock MVC code:**
  - **Add a randomAdvance() method to the appropriate class:**

```
minutes+= Math.random()*MAX_ADVANCE;
// Math.random() returns double between 0.0 and 1.0
// Store MAX_ADVANCE appropriately (use 20 minutes)
```

  - **"Crazy" button:**
    - **Declare it, create it, add to to the appropriate panel**
    - **Write an `ActionListener()` for it to increment the time by `randomAdvance()` when the `crazyButton` is clicked**
  - **Save/compile and run your program**