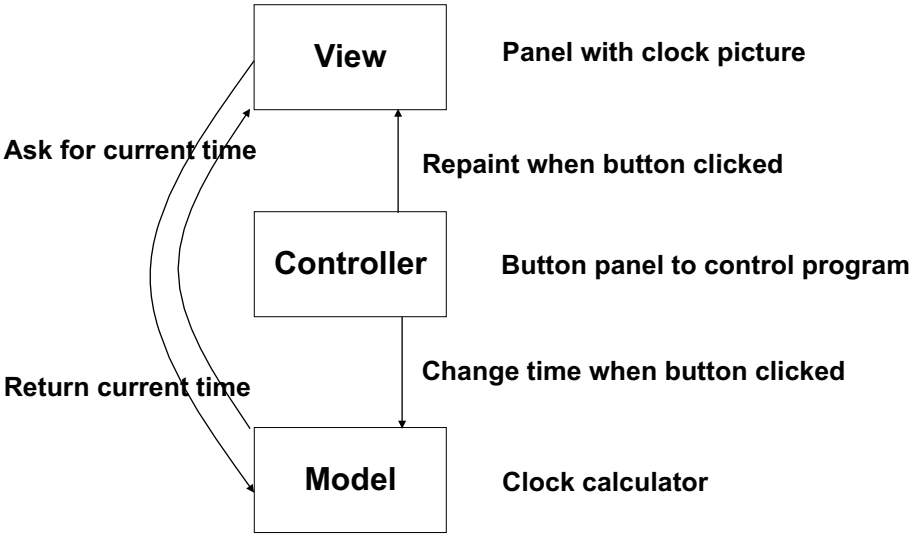
1.00 Lecture 20

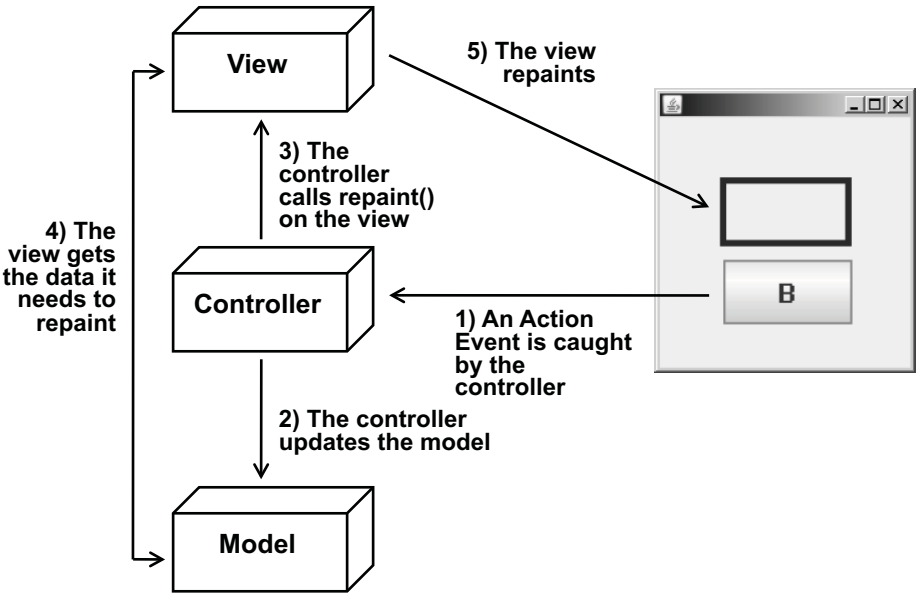
Model-View-Controller
Timers
More Swing Components

Reading for next time: None
Pick up sensor kits (Phidgets) at office hrs Wed-Thu

Clock: Model-view-controller



Model-view-controller operation



Clock Model

```
// Notice we don't 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 void advance() {
        minutes++;
        return;
   }
}
```

Clock View

```
import java.awt.* ;                  // No knowledge of events
import java.awt.geom.*; 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);
    Graphics2D g2= (Graphics2D) g;
    double minutes= model.getMinutes();      // Ask model for time
    Shape e= new Ellipse2D.Double(100, 0, 100, 100);
    g2.draw(e);
    double hourAngle = 2 * Math.PI * (minutes - 3 * 60) / (12 * 60);
    double minuteAngle = 2 * Math.PI * (minutes - 15) / 60;
    Line2D.Double hour= new Line2D.Double(150, 50, 150 + (int) (30 *
       Math.cos(hourAngle)), 50 + (int) (30 * Math.sin(hourAngle)));
    g2.draw(hour);
    Line2D.Double m= new Line2D.Double(150, 50, 150 + (int) (45 *
        Math.cos(minuteAngle)), 50 +(int)(45* Math.sin(minuteAngle)));
    g2.draw(m);
} }
```

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 contentPane;
   private ClockView view;            // Reference to view
   private ClockModel clock;          // Reference to model

   public ClockController() {
        contentPane = getContentPane();
        setSize(200, 300);  setTitle( "MVC Clock" );
        buttonHolder = new JPanel();  // Create button holder
        contentPane.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
      view= new ClockView(clock);          // Creates view object
      contentPane.add( view, BorderLayout.CENTER );

      tickButton.addActionListener(new ActionListener() {
        public void actionPerformed(ActionEvent ae) {
          clock.advance(); view.repaint();   // Use model, view
          setLabels(); }
      });

      resetButton.addActionListener(new ActionListener() {
        public void actionPerformed(ActionEvent ae) {
          clock.setMinutes(720); view.repaint(); // Use model, view
          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

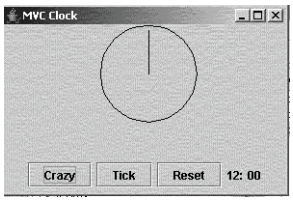
- Model: computational
  - Only knows how to compute the solution
  - Doesn't know how to draw
  - Doesn't know about events, or the GUI at all
- View: purely display of results
  - Only knows how to draw
  - Doesn't know how to compute the solution
  - Doesn't know about events
- Controller: manages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

Exercise 1

- Download and modify the clock MVC code (ClockModel, ClockView, ClockController)
  - Do NOT use ClockFrame or ClockPanel
  - Add a randomAdvance() method to the appropriate class:

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

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



Timers

- Swing allows you to create a timer (class `Timer`) that will send an event once or repeatedly after a timeout that you set.
- The event is an `ActionEvent`. The listener must implement the `ActionListener` interface as if it were waiting for a button press.
- How to create a `Timer`:

```
Timer t = new Timer(intervalInMillisecs,
  listener);
```

- How to start and stop a `Timer`:

```
t.start(); t.stop();
```

- How to tell a `Timer` to fire only once:

```
t.setRepeats(false);
```

Exercise 2

- Remove the tick button
- Add a Timer object to ClockController
  - Remember to start() it after creating it
- Create a class to listen to the timer and advance the clock
  - It must implement ActionListener and must have an actionPerformed() method
  - actionPerformed() must increment the clock by one minute every second (1000 milliseconds)
  - Use an anonymous inner class, similar to those that listen to the buttons
    - You can use the tickButton actionPerformed() logic
- No changes in ClockView or ClockModel

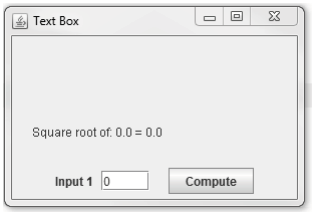
JComponent Size

- There are three set methods that allow you to change a component's *size hints*.
  - `public Dimension setMinimumSize(Dimension d)`
  - `public Dimension setPreferredSize(Dimension d)`
  - `public Dimension setMaximumSize(Dimension d)`
- Where a Dimension argument, d, is created via:
  - `Dimension d = new Dimension(int width, int height)`
- You can also ask components for their size hints:
  - `public Dimension getMinimumSize()`
  - `public Dimension getPreferredSize()`
  - `public Dimension getMaximumSize()`

Exercise 3

- Remove setSize() in controller constructor
- Make the reset and crazy buttons, and the panel that holds them, larger
  - JButton and JPanel are JComponents, which have setPreferredSize() and related methods
  - Use 100 by 50 for the buttons, and 300 by 100 for the panel
- Set the preferred size for the view to 125 by 125
- In the controller constructor, after adding all the components to the panel and pane:
  - Call the JFrame pack() method (no arguments)
  - pack() causes the frame to be sized to fit the preferred size and layouts of its subcomponents
- No changes to model or view

JTextBox Example



You'll use text boxes in homework 6/7

JTextBox Example

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

public class TextBoxController extends JFrame {
   private TextBoxView view;
   private TextBoxModel model;
   private Container contentPane ;
   private JPanel buttonHolder ;
   private JTextField field1;
   private JButton compute;

   public TextBoxController() {
        contentPane = getContentPane() ;
        setTitle("TextBox");
        setSize(300, 200);
        buttonHolder = new JPanel() ;
        contentPane.add( buttonHolder, BorderLayout.SOUTH );
        buttonHolder.add( new JLabel("Input 1 "));
        field1= new JTextField(4);    // 4 columns wide
        field1.setText("0");
        buttonHolder.add( field1 );
```

JTextBox Example, p.2

```
        buttonHolder.add( new JLabel( "   " ));
        compute= new JButton("Compute");
        compute.addActionListener(new ActionListener() {
            public void actionPerformed( ActionEvent e ) {
                model.setNumber(Double.parseDouble(field1.getText()));
                view.repaint();
        }});
        buttonHolder.add( compute );
        model= new TextBoxModel();
        view= new TextBoxView(model);
        view.setPreferredSize(new Dimension(100, 100));
        contentPane.add(view, BorderLayout.CENTER );
        contentPane.repaint();
    }
    public static void main( String[] args ) {
        TextBoxController c= new TextBoxController();
        c.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE) ;
        c.setVisible(true) ;
    }
}
```

JTextBox Example, p.3

```
import javax.swing.*; import java.awt.*;
public class TextBoxView extends JPanel {
    private TextBoxModel model ;
    public TextBoxView( TextBoxModel m ) {
        model = m ; }
    public void paintComponent(Graphics g) {
        super.paintComponent(g);
        Graphics2D g2= (Graphics2D) g;
        double n= model.getNumber();
        double sq= model.getSqrt();
        g2.drawString("Square root of "+ n + " = " +sq, 20, 100);
    }
}
```

```
public class TextBoxModel {
   private double number;
   public TextBoxModel() {};
   public void setNumber(double n) { number= n;}
   public double getNumber() { return number;}
   public double getSqrt() { return Math.sqrt(Math.abs(number));
}
```

Exercise 4: Formatting doubles

- **To display a fixed number of digits after the decimal point:**

```
// At top of file, typically your view class:
import java.text.*;

// In your code, typically in the view:
DecimalFormat f= new DecimalFormat();
f.setMaximumFractionDigits(3);  // Or other desired value
…
x= Math.sin(1.3224);
g2.drawString("" + f.format(x), 150, 60);  // 3 decimal places
```

- **Exercise:**
  - **Modify TextBoxView to show only 3 digits in the square root**

1.00 / 1.001 / 1.002 Introduction to Computers and Engineering Problem Solving
Spring 2012