

CSE 8B—Intro to CS: Java

Winter, 2006

March 2 (Day 16)
Model-View-Controller
Observer/Observed
Automating GUI testing

Instructor: Neil Rhodes

MVC

Traditional Model/View/Controller

- Model holds the data and computations. No user interface.
 - It notifies listeners when it changes
- View displays the data
 - It listens to the model, updating itself as-needed
- Controller
 - Handles user interaction, updating the model

Swing:

- View and controller are bound together
- If you don't provide a model, one is created for you
 - By default, each checkbox is linked to its own model
 - Model actually holds listeners and notifies listeners on change

Biggest advantage

- Multiple views of the same data

2

How the model notifies the views

Change occurs to model

Model notifies views

- Either notification includes what has changed, or:
- View must query back to model to find what has changed

3

Example model

```
ModelTest() {  
    super("ModelTest");  
    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);  
    setSize(300, 100);  
    setLayout(new FlowLayout());  
  
    JCheckBox checkbox = new JCheckBox("Check me");  
    add(checkbox);  
    JCheckBox checkbox2 = new JCheckBox("Check me  
2");  
    add(checkbox2);  
    checkbox.setModel(checkbox2.getModel());  
}
```

4

Observer/Observed

Observer is an interface that specifies this object can be notified of changes to other objects

One method:

- `void notify(Observable o, Object arg)`
- Specifies that o has changed value. Extra information may be passed in arg.

Observable is a class that implements the ability to notify observers of changes to this object

- Methods:
 - `void setChanged();` // specifies that this object // has changed
 - `addObserver(Observer o);` // add given observer
 - `void deleteObserver(Observer o);`
 - `void notifyObservers(Object arg);` // if this is // changed, tells all observers

5

Example Observable

ObservableValue

- Stores an integer which is the current value
 - Also stores high and low bounds
- Can retrieve the value of the integer
- Can change the value of the integer
 - On change, notifies all its observers of the change

6

```
import java.util.Observable; ObservableValue
```

```
public class ObservableValue extends Observable
{
    private int value = 0;
    private int low = 0;
    private int high = 0;

    public ObservableValue(int value, int low, int high) {
        this.value = value;
        this.low = low;
        this.high = high;
    }

    public void setValue(int value) {
        this.value = value;
        setChanged();
        notifyObservers();
    }

    public int getValue() {
        return value;
    }

    public int getLowerBound() {
        return low;
    }

    public int getHigherBound() {
        return high;
    }
}
```

7

TextObserver

```
private ObservableValue ov = null;
private JTextField tf = null;
private Label l = null;
public TextObserver(ObservableValue observableValue)
{
    super("Text Observer Tool");
    this.ov = observableValue;
    ov.addObserver(this);

    setLayout(new GridLayout(0, 1));
    addTextField();

    l = new Label();
    add(l);
    pack();
    setVisible(true);
}

public void update(Observable obs, Object obj) {
    if (obs == ov)
        tf.setText(String.valueOf(ov.getValue()));
}
```

8

TextObserver

```
public void addTextField() {
    tf = new JTextField(String.valueOf(ov.getValue()));
    tf.addActionListener(new ActionListener() {
        public void actionPerformed(ActionEvent e) {
            int n = 0;
            boolean valid = false;
            try {
                n = Integer.parseInt(tf.getText());
                if (n >= ov.getLowerBound() && n <= ov.getHigherBound())
                    valid = true;
            } catch (NumberFormatException nfe) {}
            if (valid) {
                ov.setValue(n);
                l.setText("");
            } else
                l.setText("invalid value -- please try again...");
        }
    });
    add(tf);
}
```

9

ScrollObserver

```
private ObservableValue ov = null;

private JSlider s = null;

public ScrollObserver(ObservableValue observableValue)
{
    super("Scroll Observer Tool");

    this.ov = observableValue;
    ov.addObserver(this);

    setLayout(new GridLayout(0, 1));

    addSlider();
    pack();
    setVisible(true);
}

public void update(Observable obs, Object obj)
{
    if (obs == ov)
        s.setValue(ov.getValue());
}
```

10

ScrollObserver

```
public void addSlider() {
    s = new JSlider(ov.getLowerBound(), ov.getHigherBound(),
        ov.getValue());
    s.addChangeListener(new ChangeListener() {
        public void stateChanged(ChangeEvent e) {
            ov.setValue(s.getValue());
        }
    });
    add(s);
}
```

11

Putting it all together

```
public static void main(String[] args) {
    ObservableValue o = new ObservableValue(50, 0, 100);
    TextObserver to = new TextObserver(o);
    ScrollObserver s = new ScrollObserver(o);
}
```

12

Adding a model to Tic-Tac-Toe

What methods belong in the Tic-Tac-Toe model?

13

Automating GUI Testing

Best idea: make the GUI so simple that there's not much to go wrong

- Make all the interesting part in the model

But, if you need to, there are frameworks to help

- JFCUnit, for example
 - Name each component (`component.setName("name")`)
 - Access component:

```
- finder.setName( "LoginNameTextField" );  
- JTextField userNameField = ( JTextField ) finder.find  
  ( loginScreen, 0 );  
- assertNotNull( "Could not find the userNameField",  
  userNameField );
```
 - Manipulate component:

```
- getHelper().sendString( new StringEventData( this,  
  userNameField, "admin" ) );
```

14