ISE 582: Information Technology for Industrial Engineers

University of Southern California
Department of Industrial and Systems Engineering

Lecture 9
JAVA Cup 8: Applets

Handouts & Announcements

• Lecture 9 slides
• READ Winston & Narasimhan:
  – Chapters 42-46 (pp 275-304)
• Homework 5 (posted Saturday)

JAVA Cup 8

• Standalone Observers / Listeners
• Defining Applets
• Calling Applets from Web Browsers
• Resource Allocators
• Making Lists for Selection
• Adding Images to Applets
### Hierarchy of Swing Classes

- **Component**
  - **Container**
    - **Window**
    - **Panel**
    - **JComponent**
      - **Frame**
      - **Applet**
      - **JPanel**
      - **JList**
      - **JLabel**
      - **JTextField**
      - **JButton**

### Review the Model-View Approach

- **Application**
  - **Model**
  - **View**

### Standalone Observers / Listeners

- **Goal**: move Observers / Listeners into independent files
- **Changes**:
  - Establish a MovieApplication instance variable in the class
  - Include the application instance as an argument to the constructor
  - Assign the application instance variable in the constructor
Example: Old format

```java
import java.util.*;
public class MovieObserver implements Observer {
    public void update (Observable observable, Object object) {
        getMeter().setValue(getMovie().rating());  // BUG!
        getMeter().setTitle(getMovie().getTitle());  // BUG!
    }
}
```

Example: New Format

```java
import java.util.*;
public class MovieObserver implements Observer {
    private MovieApplication applet;
    public MovieObserver (MovieApplication a) {
        applet = a;
    }
    public void update (Observable observable, Object object) {
        applet.getMeter().setValue(applet.getMovie().rating());
        applet.getMeter().setTitle(applet.getMovie().getTitle());
    }
}
```

Similar Changes in MeterListener

```java
import java.awt.event.*;
public class MeterListener extends MouseAdapter {
    private MovieApplication applet;
    public MeterListener (MovieApplication a) {
        applet = a;
    }
    public void mouseClicked (MouseEvent e) {
        int x = e.getX();
        int y = e.getY();
        int v = (int) Math.round(applet.getMeter().getValueAtCoordinates (x, y) / 3.0 );
        applet.getMovie().setScript(v);
        applet.getMovie().setActing(v);
        applet.getMovie().setDirection(v);
    }
}
```

In setMovie:
```
movie.addObserver(new MovieObserver(this));
```

In setMeter:
```
meter.addMouseListener(new MeterListener(this));
```
Applets

- Properties
- Browser Conventions
- What Applets Cannot Do
- What Applets Can Do
- Accessing Applets from Web Browsers

Applets: Properties

- Applets have no main methods.
- The applet's size is determined by the html file (no calls to setSize, show).
- There are no WindowAdapter subclass and no windowClosing method in JApplet classes.

Changing to an Applet

- extend JApplet instead of JFrame
- Remove main method
- Remove WindowListener and call to addWindowListener, do not import java.awt.event package
- Constructor need not pass title to super(), eliminate calls to setSize, show
Browser Conventions

- When a browser loads an applet, it calls:
  - the constructor
  - the init method (default init does nothing)
  - the start method (contains computations to be performed to start the applet)
- When the applet page is replaced, it calls:
  - the stop method (also called before destroy)
- When the applet page is abandoned completely, it calls:
  - the destroy method

What Applets Cannot Do

- Load libraries or define native methods.
- Read/write files on the host that's executing it.
- Make network connections except to the host that it came from.
- Start any program on the host that's executing it.
- Read certain system properties.
- Windows that an applet brings up look different than windows that an application brings up.

What Applets Can Do

- Play Sounds
- Make network connections to the host they came from.
- Cause HTML documents to be displayed.
- Invoke public methods of other applets on the same page.
- Applets that are loaded from the local file system have none of the restrictions that applets loaded over the network do.
- Keep running after you leave their page.
Calling an Applet from an HTML file

- When a Java-enabled browser encounters an `<APPLET>` tag, it:
  - reserves a display area of the specified width and height for the applet,
  - loads the bytecodes for the specified Applet subclass,
  - creates an instance of the subclass, and then
  - calls the instance's init and start methods.

- `<applet code="?.class" width=? height=?>
  </applet>`

Using Resource Locators

- Why use Resource Locators?
- The Class class
- Using the Class class
- How Resource Locators work
- Accessing Image Files

Why Use Resource Locators

- They find files even after you have moved your program from one directory to another
- They find files properly not only for standalone applications, but also for browser applications
The Class class

- When the JVM loads a class / interface,
  – it creates a companion “Class” instance
- The “Class” instance provides
  – information about the class’s name and
    instance variables
  – access to methods for locating resources
    such as text and image files

Using the Class class

```java
public class MovieAuxiliaries {
    public static Vector readMovieFile(String fileName) {
        Vector v = new Vector();
        URL url = MovieAuxiliaries.class.getResource(fileName);
        InputStream stream = (InputStream) (url.getContent());
        InputStreamReader reader = new InputStreamReader(stream);
        ...   }
    }
```

How Resource Locators Work

- The resource locator starts looking for the file
  in the same location as the class
- If the filename includes a relative path, the
  resource locator will look for the file in the
  subdirectory of the directory holding that
  class
- If the class is part of a browser, this
  determines the security rules employed
- Why insist on the same-place restriction?
**Accessing Image Files**

```java
import java.net.*; import java.awt.image.*; Blah blah blah...
public static Image readMovieImage(String fileName) {
    try {
        URL url = MovieAuxiliaries.class.getResource(fileName);
        if (url == null) {return null;} // If URL is null, return null
        ImageProducer producer = (ImageProducer) url.getContent();
        if (producer == null) {return null;} // If producer is null, return null
        Toolkit tk = Toolkit.getDefaultToolkit();
        Image image = tk.createImage(producer);
        return image;
    } catch (IOException e) { System.out.println(e); } // Print exception message
    return null;
} Blah blah blah...
```

**Using Choice Lists**

- A new MovieData class
- Modifications to the Model side
- Modifications to the View side
- Linking the new model and new view
- Final changes
- Using a ScrollPane

**MovieDataInterface**

```java
import java.util.*;
public interface MovieDataInterface {
    // Put movies into data source:
    public abstract void setMovieVector  (Vector v) ;
    // Extract movies from data source:
    public abstract Vector getMovieVector  () ;
    // Find a particular movie at given index:
    public abstract Movie getMovie(int index) ;
    // Miscellaneous method:
    public abstract void changed () ;
}
```
Modifications to the Model

```java
private MovieData movieData;

public MovieData getMovieData () {
  if (movieData == null) { setMovieData(new MovieData ()); }
  return movieData;
}

public void setMovieData (MovieData m) {
  movieData = m;
  movieData.addObserver(new MovieDataObserver(this));
  movieData.changed();
}
```

Modifications to the View

```java
private JList jList;

public MovieApplication() {
  getMovie(); getMovieData();
  getContentPane().add("Center", getMeter());
  getContentPane().add("East", getJList());
}

public JList getJList () {
  if (jList == null) {setJList(new JList());}
  return jList; }

public void setJList (JList j) {
  jList = j; }
```

Linking Model and View

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

public class MovieDataObserver implements Observer {
  private MovieApplication applet;
  public MovieDataObserver (MovieApplication a) { applet = a; }

  public void update (Observable observable, Object object) {
    Vector titles = new Vector();
    for (Iterator i = applet.getMovieData().getMovieVector().iterator();
         i.hasNext(); ) {
      Movie movie = (Movie) i.next();
      titles.add(movie.getTitle());
    }

    applet.getJList().setListData(titles);
  }
```
Linking Model and View

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

public class MovieListListener implements ListSelectionListener {
    private MovieApplication applet;
    public MovieListListener(MovieApplication a) {
        applet = a;
    }
    public void valueChanged(ListSelectionEvent e) {
        int index = applet.getJList().getSelectedIndex();
        applet.setMovie(applet.getMovieData().getMovie(index));
    }
}
```

Final Changes

```java
public void setMovieData(MovieData m) {
    movieData = m;
    movieData.addObserver(new MovieDataObserver(this));
    movieData.changed();
}
public void setJList(JList j) {
    jList = j;
    jList.addListSelectionListener(new MovieListListener(this));
}
```

Using a Scroll Pane

- Instead of
  - getContentPane().add("East",getJList());
- Use
  - getContentPane().add("East",new JScrollPane(getJList()));