CSE1030 – Introduction to Computer Science II

Lecture #13

Graphical User Interface II

Goals for Today

- Theoretical
  - Model View Controller architecture
- Practical
  - Introduction to Games

CSE1030 – Lecture #13

- Review
- MVC
- Game Programming
- We're Done!

Java GUI Programming Goals

- The objective is to become "familiar" with the parts of a Java GUI program
- The size of the API makes learning GUI programming difficult
  - There's a whole 3rd year course just on Java GUI programming (CSE3461)
- For now,
  - Tinker with the Demo programs
  - Use the Demo programs as a basis for your GUI programs

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

public class DemoSwing extends JFrame implements ActionListener
{
   public static void main(String[] args)
   {
      DemoSwingFrame frame = new DemoSwingFrame();
      frame.setTitle("Swing Demo");
      frame.pack();
      frame.setVisible(true);
   }

   private int clickCount;
   private JLabel tally;
   private JButton pressMeButton;
   private JButton exitButton;
```

Important imports

Setup the Frame

setVisible()!

Define Widgets

```
   public DemoSwing()
   {
      // -------------------------------------
      // declare and initialize local variables
      // -------------------------------------

      clickCount = 0;

      // ------------------------------
      // create and configure components
      // ------------------------------

      tally = new JLabel("Click count: 0");
      tally.setHorizontalAlignment(SwingConstants.CENTER);
      pressMeButton = new JButton("Press me!");
      exitButton = new JButton("Exit");

      // ------------
      // add listeners
      // ------------

      pressMeButton.addActionListener(this);
      exitButton.addActionListener(this);
      this.addWindowListener(new WindowCloser());
```

Setup the widgets we declared earlier

listeners

(so we can get input)

```
      // -----------------
      // arrange components
      // -----------------

      // put components in a panel

      JPanel panel = new JPanel();
      panel.setBorder(BorderFactory.createEmptyBorder(10,
                                                      10, 10, 10));
      panel.setLayout(new GridLayout(3, 1));
      panel.add(pressMeButton);
      panel.add(tally);
      panel.add(exitButton);

      // make the panel this extended JFrame's content pane

      setContentPane(panel);
   }
```

Layout the component in a JPanel with a LayoutManager

The ContentPane is the main area of the application window

```
   // ------------------------------
   // implement ActionListener method
   // ------------------------------

   public void actionPerformed(ActionEvent ae)
   {
      if (ae.getSource() == pressMeButton)
      {
         clickCount++;
         tally.setText("Click count: " + clickCount);
      }
      else if (ae.getSource() == exitButton)
         System.exit(0);
   }


   private class WindowCloser extends WindowAdapter
   {
      public void windowClosing(WindowEvent event)
      {
         System.exit(0);
      }
   }
}
```

Listeners get us our input

Sometimes we use inner classes

CSE1030 – Lecture #13

- Review
- MVC
- Game Programming
- We’re Done!

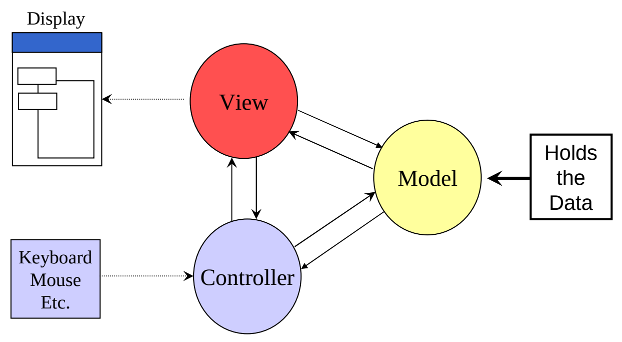
Background

- The model-view controller (MVC) paradigm was developed at the *Xerox Palo Alto Research Center* (PARC).
- MVC was central to the architecture of the multi-windowed *Smalltalk* environment used to create the first graphical user interfaces.
- The approach was borrowed by the developers of the Apple *Macintosh* and many other imitators.
- In such an interface, input is primarily via the mouse and keyboard; output is a mix of graphics and textual components as appropriate.
- MVC is elegant and simple, but unlike the approach of traditional application programs.

MVC Paradigm

- Traditional paradigm…
  - Input $\rightarrow$ processing $\rightarrow$ output
- MVC paradigm…
  - Controller $\rightarrow$ model $\rightarrow$ view

MVC Schematic



Controller Tasks

- Receive user inputs from mouse and keyboard
- Map these into commands that are sent to the model and/or viewport to effect changes in the view
- E.g., detect that a button has been pressed and inform the model that the button stated has changed

Model Tasks

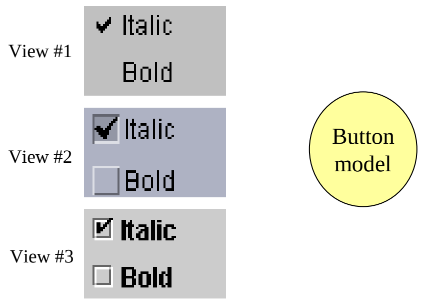
- Store and manage data elements, such as state information
- Respond to queries about its state
- Respond to instructions to change its state
- E.g., the model for a button can be queried to determine if the button is pressed

View tasks

- Implements a visual display of the model
- E.g., a button has a coloured background, appears in a raised perspective, and contains an icon and text; the text is rendered in a certain font in a certain colour

MVC Concepts – *multiple views*

- Any number of views can subscribe to the model



MVC Concepts - *Model Changes*

- What happens when the model changes?
- E.g., a button is pressed (the state of the button has changed!)
- The model must notify the view
- The view changes the visual presentation of the model on the screen

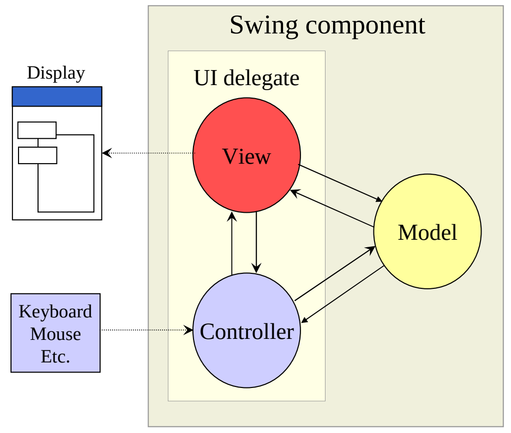
Benefits of MVC Architecture

- Improved maintainability
  - Due to modularity of software components
- Promotes code reuse
  - Due to OO approach (e.g., subclassing, inheritance)
- Model independence
  - Designers can enhance and/or optimize model without changing the view or controller
- Plug-able look and feel
  - New L&F without changing model
  - Multiple views use the same data

MVC and Swing

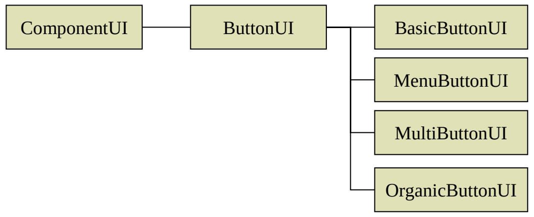
- Swing designers found it difficult to write a generic controller that didn't know the specifics about the view
- So, they collapsed the view and controller into a single UI (user interface) object known as a delegate (the UI is *delegated* to this object)
- This object is known as a *UI delegate*

MVC and Swing (2)



ComponentUI Class

- The delegate part of a component is derived from an abstract class named ComponentUI
- Naming convention: remove the “J” from the component’s class name, then add “UI” to the end (e.g., JButton → ButtonUI)

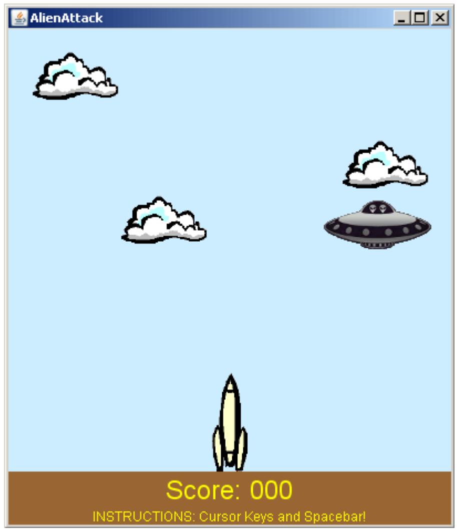


Swing Models

- In Swing, many models exist as interfaces
  - Eg., ButtonModel, BoundedRangeModel, ComboBoxModel, ListModel, ListSelectionModel, TableModel, Document
- The interface is implemented in model classes
- Usually there is a default model class that is automatically associated with a component (whew!)
  - E.g., DefaultButtonModel implements ButtonModel
  - E.g, AbstractDocument implements Document (PlainDocument is a subclass of AbstractDocument)

CSE1030 – Lecture #13

- Review
- MVC
- Game Programming
- We’re Done!



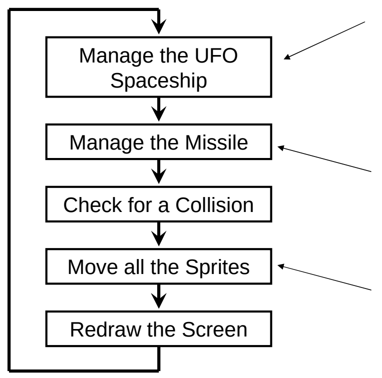
AlienAttack

- Controllable Missile
- Randomised UFO spaceship
- Animated Explosion
- Points (Score)

Features of the Game

- Three things have to happen simultaneously:
  1. Moving images (Sprites!)
  2. User Input controls the Missile
  3. Missile and UFO spaceship collision
- Animation requires that the screen be redrawn at least 20 time a second, but faster is even better

The Main Loop

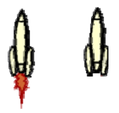


1. Wait for random delay
2. Start new screen transit, with random: side, height, speed
3. Need special handling of the explosion

If the missile is moving, wait for it to hit the top of the screen, and then stop it

Need a way to track all of the sprites, with their speeds and directions

Media

- This is the most important part of the game:
  -
  -
  - 

AlienAttack

- Let’s Look at the Code…

CSE1030 – Lecture #13

- Review
- MVC
- Game Programming
- We’re Done!

Next topic…

Graphical User Interface III