

HCI JAVA SWING SOLNS 99

- (a) *What is the Model-View-Controller (MVC) paradigm and why is it important to Graphical User Interface Design? Briefly describe the function of each of the MVC units.*

The Model-View-Controller architecture (MVC) is a fundamental design behind each of iGUI components.

- MVC breaks GUI components into 3 elements :

Model

- State data for each component.
Different for each component
E.g. Scrollbar -- Max and minimum Values, Current Position, width of thumb position relative to values
E.g. Menu -- Simple List of items.
Model data is always independent of visual representation.

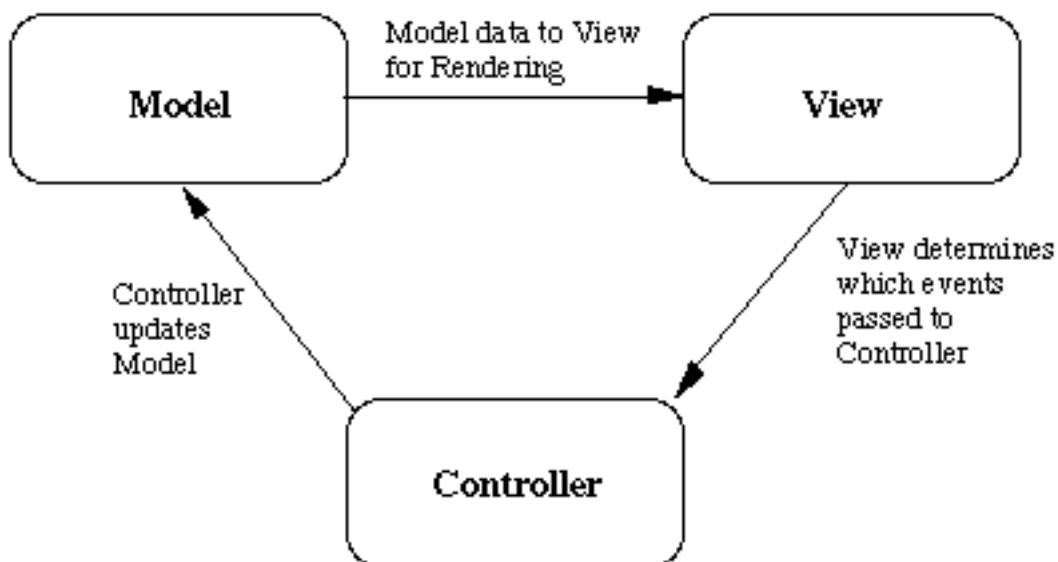
View

- How component is painted on the screen
Can vary between platforms/look and feel

Controller

- dictates how component interacts with events
Many forms of events -- mouse clicks, keyboard, focus, repaint etc..
Controller decides how component reacts to an event -- if at all

The three elements interact as follows:



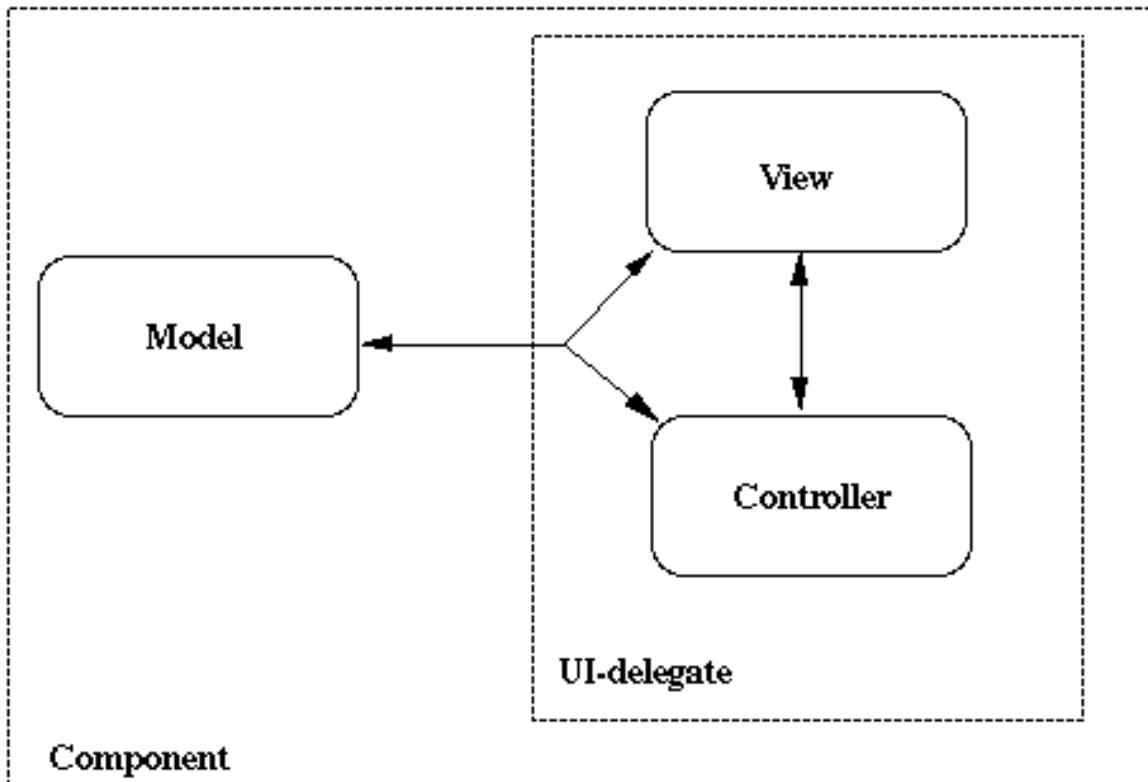
- Each elements plays an important role in how component behaves

- Allows for Portability etc.

5 MARKS ---- Bookwork

(b) How do the MVC Architectures of Smalltalk and Java Swing differ? What is the reason for the difference.

Swing uses a simplified model of the basic smalltalk MVC design called the *model-delegate* architecture



The MVC Swing Model

Combines View and Controller into a single element -- the UI-delegate

Easy to bundle graphics and event handling in Java -- AWT.

Model delegate interaction is now Two way.

Model -- maintains information

UI-delegate -- Draws and reacts to events that propagate through component.

4 MARKS ---- Draws together aspects of BOTH Halves of this Module

(c) *What are the main features of Java Swing Components? What is meant by the term lightweight components in relation to Swing? How is customised rendering of the look and feel of Swing Components facilitated by the MVC architecture?*

Swing Features

- Pluggable look-and feels
- Lightweight components
 - Do not depend on native peers to render themselves.
 - Simplified graphics to paint on screen
 - Similar behaviour across all platforms
 - Portable look and feel
 - Only a few top level containers not lightweight.
- A wide variety of GUI components – basic items plus trees, tables, sliders, progress bars, frames, text components. Tooltips -- textual popup to give additional help
- arbitrary keyboard event binding
- Debugging support

All lightweight contain `paint()`, `repaint()` methods as part of their description --- contain very basic Java graphics. These methods are overridden for appropriate look and feels.

4 MARKS ---- Applied Bookwork

(d) *For most applications the default Look-and-Feel libraries provided by Swing are adequate. However, there may be some occasions when a customised Look-and-Feel is more appropriate.*

Give two such situations with reasons why a customised Look-and-Feel should be adopted

A few possible situations (possibly more):

1. Design of a corporate or other standard user interface --- customisation of applications so that they look and feel the same way on many platforms on different systems/intranets. Reduce training costs create corporate identity.
2. Ease Portability --- write once run everywhere. Design custom libraries that port more easily, ease transfer of code
3. New component creation --- add to Swing's existing set but still use useful features of swing controller/viewer portions of similar components etc.

7 Marks --- Totally Unseen

4 (a) *What are actions in relation to Java Swing Components? Give an example application where actions might typically be used.*

Actions are a popular addition to Swing

- Bundles commonly used procedure and properties in a common Class
- Especially useful when application needs to call a function from multiple sources

e.g. Save data to disk from Menu Item, Save Button, Dialog action (after a quit) etc. (see Section for a full program example)

Marks --- Bookwork

(b) An editor application wishes to provide a *save as* facility from a menu item and also as an option from a save icon in a toolbar which may be displayed at certain times in a program

Show with aid of suitable diagrams and Java Swing code fragments *only*, how you would achieve the above application interface and how you would facilitate appropriate user interaction:

Your answer should show how following features in the application are provided:

- The creation of an appropriate menu item
- The creation of an appropriate tool bar item
- The interface between the events that trigger a save operation from either of these components

Do not give details of:

- The design or creation of other interface components
- The actual code that would save the data to a file

SOLUTION

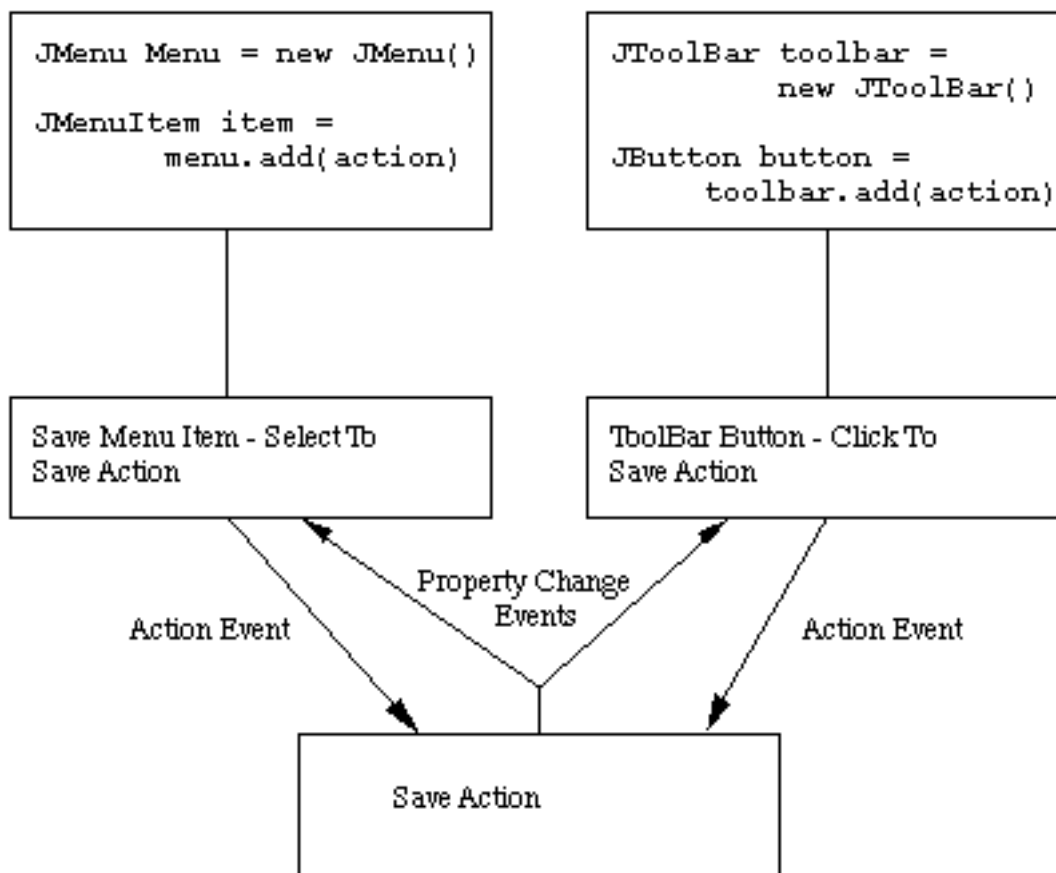
The solution is sketch of many possible attempted solutions

Important points is the use of the ACTION CLASS to provide an interface to same save function

Marking scheme approx 5 marks for each point asked for in question, but allow leeway for alternatives.

BASICALLY WE DO

- create a Save Menu Item
- create a ToolBar Button
- Both invoke the same save event via an action:



Full Code Listing (NOT EXPECTED) but illustrates main points required:

```
// ActionExample.java
//
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import javax.swing.border.*;

public class ActionExample extends JPanel {

    public JMenuBar menuBar;
```

```

public JToolBar toolBar;

public ActionExample() {
    super(true);

    // Create a menu bar and give it a bevel
border
    menuBar = new JMenuBar();
    menuBar.setBorder(new
BevelBorder(BevelBorder.RAISED));

    // Create a menu and add it to the menu bar
JMenu menu = new JMenu("Menu");
    menuBar.add(menu);

    // Create a toolbar and give it an etched
border
    toolBar = new JToolBar();
    toolBar.setBorder(new EtchedBorder());

    // Instantiate a sample action with the NAME
property of
    // "Save" and the appropriate SMALL_ICON
property.
    SampleAction exampleAction = new
SampleAction("Save");

    // Finally, add the sample action to the menu
and the toolbar.
    menu.add(exampleAction);
    toolBar.add(exampleAction);
}

class SampleAction extends AbstractAction {

    // This is our sample action. It must have an
actionPerformed() method,
    // which is called when the action should be
invoked.
    public SampleAction(String text, Icon icon) {
        super(text, icon);
    }

    public void actionPerformed(ActionEvent e) {

// Save code goes HERE --- NOT REQUIRED    }
    }
}

```

```
public static void main(String s[]) {
    ActionExample example = new ActionExample();
    JFrame frame = new JFrame("Action Example");
    frame.addWindowListener(new
BasicWindowMonitor());
    frame.setJMenuBar(example.menuBar);
    frame.getContentPane().add(example.toolBar,
BorderLayout.NORTH);
    frame.setSize(200,200);
    frame.setVisible(true);
}
}
```

15 Marks --- Unseen