
MAHARISHI UNIVERSITY of MANAGEMENT
Engaging the Managing Intelligence of Nature
Computer Science Department
CS401 Modern Programming Practices (MPP)
Professor Paul Corazza

Lecture 6:
Advanced Swing

Wholeness of the Lesson

Swing is a windowing toolkit that allows developers to create GUIs that are rich in content and functionality. The ultimate provider of tools for the creation of beautiful and functional content is pure intelligence itself; all creativity arises from this field's self-interacting dynamics.

Introduction to Java's UI Libraries

- ***Java Swing*** is a part of Java Foundation Classes (JFC) that is used to create window-based applications. It is built on the top of AWT (Abstract Windowing Toolkit) API and entirely written in java.

- ***Sun's AWT.*** The original version of Java (jdk1.0) came with a primitive windowing toolkit (the AWT) for making simple GUIs. GUI components were built by using the native GUI toolkit of the target platform (Windows, MacIntosh, Solaris, etc). It is platform dependent.

- Unlike AWT, Java Swing provides platform-independent and lightweight components.

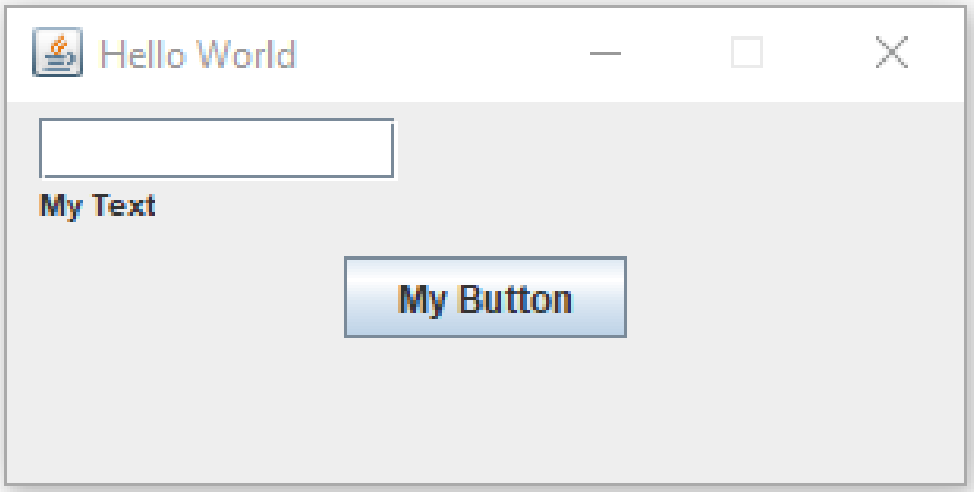
- ***AWT Still Used.*** Swing components still make use of aspects of the AWT – Swing is built "on top of" the old AWT. In particular, handling of events relies on the old event-handling model.

- ***JavaFX.*** In 2014, Oracle declared that Swing libraries would be developed no further, and that the windowing toolkit of choice had become JavaFX. JavaFX has more modern-looking components and has a more flexible API.

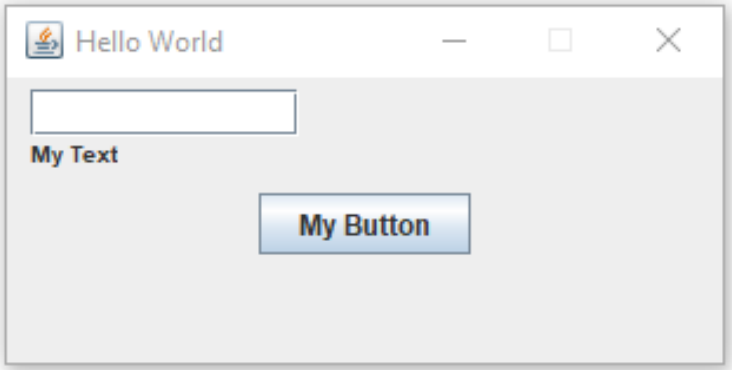
- ***Return of Swing.*** In 2018, Oracle announced that, starting with JDK 11, JavaFX will no longer be bundled with the JDK, but will be available through a separate download. On the other hand, Oracle has announced that it will resume support of Swing (along with AWT) in JDK 8 and 11 and for the foreseeable future.

Swing Review

- We work through the Swing code needed to produce the following UI:

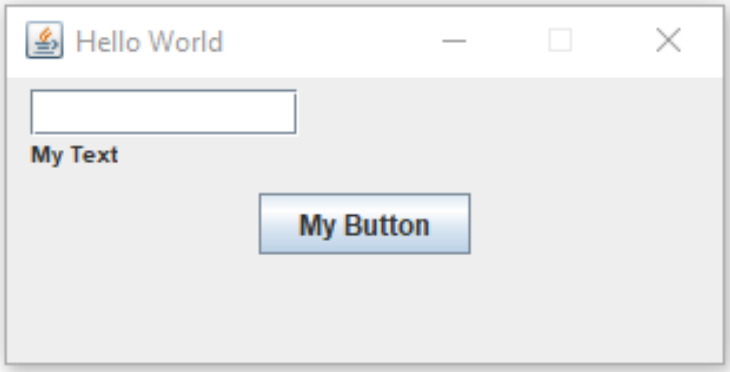


Swing Example



- BorderLayout is used
- NORTH panel contains a specially constructed textPanel (to arrange a Textfield with a Jlabel)
- CENTER panel contains the JButton
- The textPanel is a JPanel with a BorderLayout. In the NORTH is a JTextField. In the CENTER is a JLabel

Swing Example



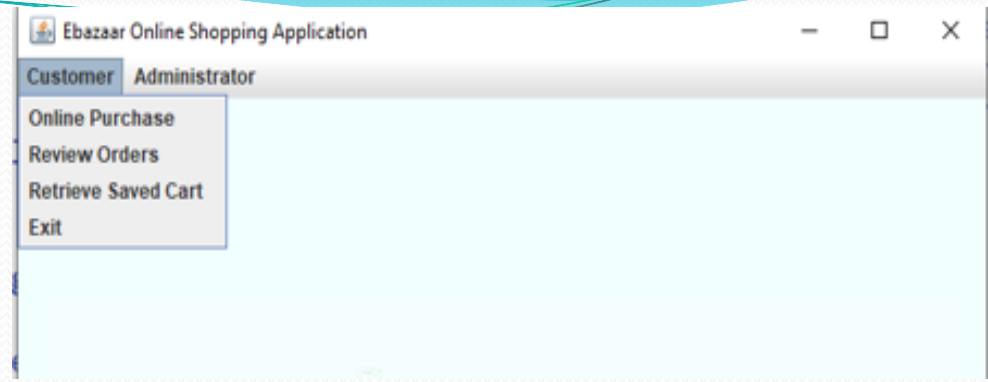
- An ActionListener is attached to the JButton using an inner class:

```
class MyButtonListener implements ActionListener {
   public void actionPerformed(ActionEvent evt){
        String textVal = text.getText();
        final String youWrote = "You wrote: ";
        text.setText(youWrote+"\""+textVal+"\".");
  }
}
```

```
button = new JButton("My Button");
button.addActionListener(new MyButtonListener());
```

Using Menus

See demo:
lesson6.lecture.menus.gui.Start



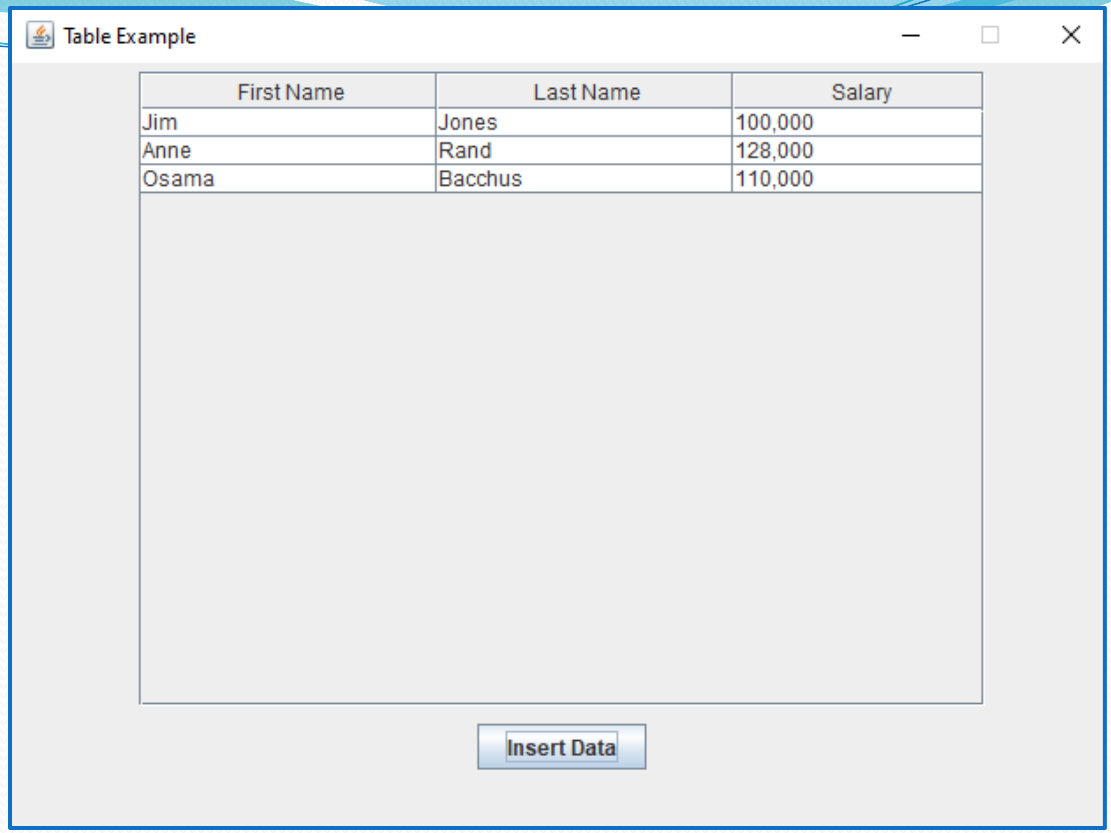
Menus in Swing are implemented using three Swing classes: JMenuBar, JMenu, JMenuItem

```
JMenuBar menuBar;
JMenu menuCustomer, menuAdministrator;
JMenuItem menuItemPurchaseOnline, menuItemMaintainProduct
```

```
menuCustomer = new JMenu(CUSTOMER);
menuAdministrator = new JMenu(ADMINISTRATOR);
menuBar.add(menuCustomer);
menuBar.add(menuAdministrator);

//purchase online menu item
menuItemPurchaseOnline = new JMenuItem(ONLINE_PURCHASE);
menuItemPurchaseOnline.addActionListener(new PurchaseOnlineActionListener());
menuCustomer.add(menuItemPurchaseOnline);
```

Using Tables

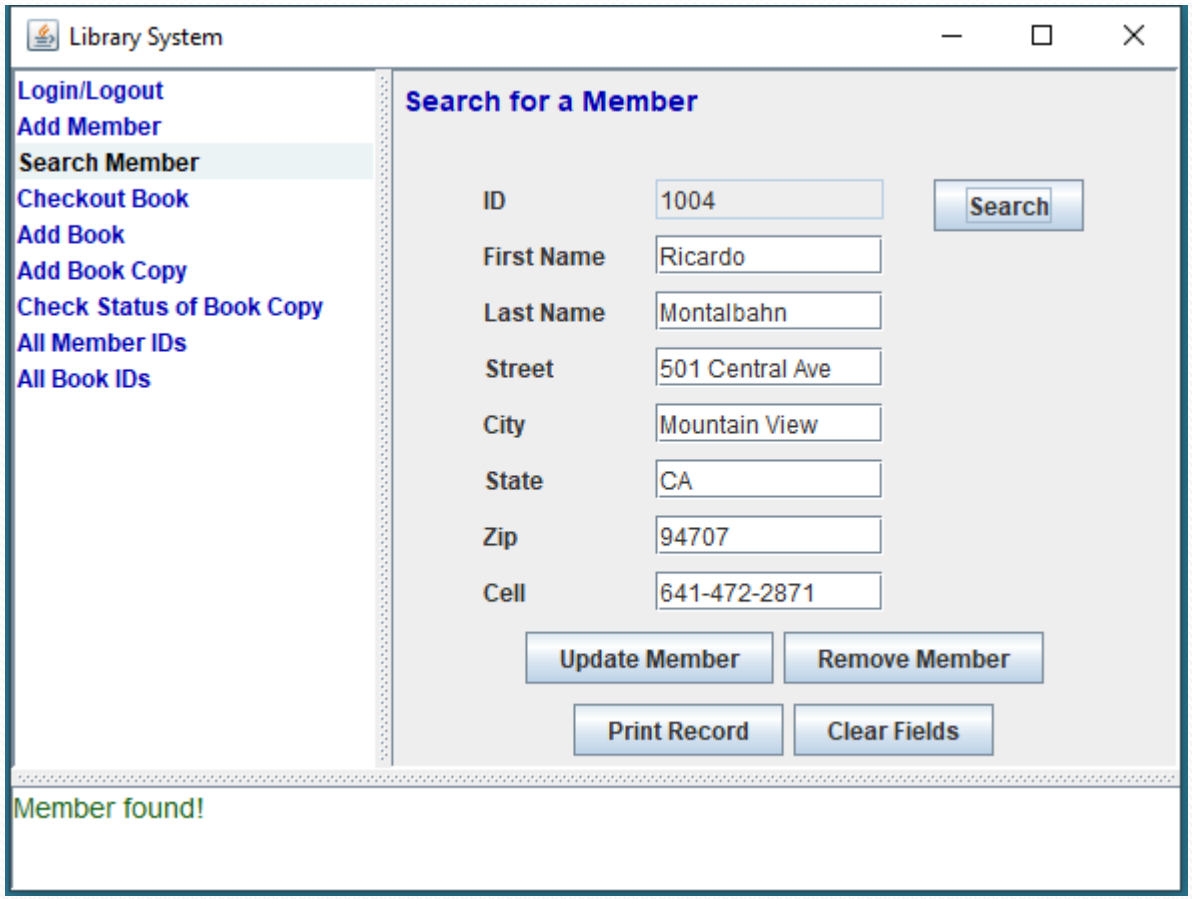


Steps for creating a table in Swing:

1. Create a new JTable
2. Embed the table in a JScrollPane
3. Add the scrollpane to the main panel
4. Set up a table model and insert model into table
5. Provide an updateModel method so that data in the table can be modified as application executes.

See demo: ProjectSwingSampleCode tables.TableSample.java

Split Panes and CardLayout



- Use *SplitPane* to partition main window into *content* (right pane), *control* (left pane), *status bar* (bottom pane).
- *CardLayout* allows you to display different panels within the same area. Here, the content panel that is shown is controlled by a JList in the left pane.
- Use a *CellRenderer* to control highlighting on the JList elements

Working with JLists in Swing

- One versatile component in Swing is a JList, which displays selectable lists.

- JLists are often embedded in a JScrollPane to support changes in the size of the list.

```
mainScroll = new JScrollPane(mainList);
```

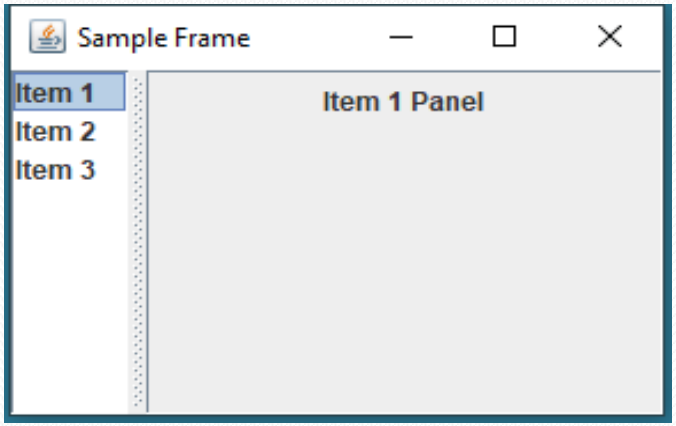
- It is possible to load data for a JList directly, but the best practice is to load it using a *list model*.

```
JList<String> list = new JList<String>(listModel);
```

A list model keeps data separate from its presentation – this supports the MVC design pattern, which allows presentation and data to change independently. For example, you can present the same data in multiple ways.

See the package `lesson6.lecture.jlist`

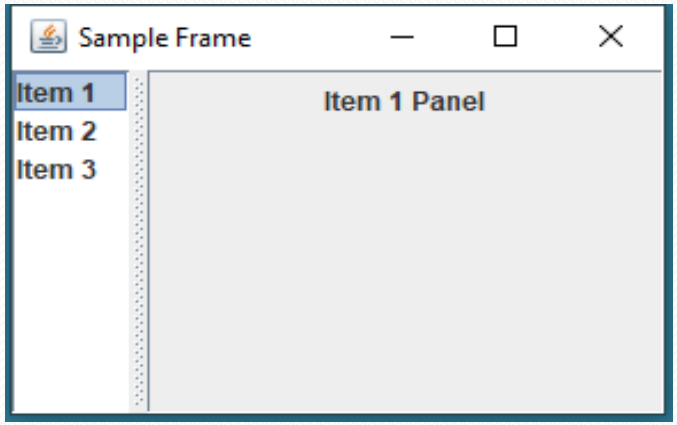
Working with SplitPane



- Create a left and right component to be displayed in the split pane. Here we have a JList on the left and a bundle of JPanels (called cards) on the right
- Create a new SplitPane instance and set the divider location.
- Add your SplitPane to the contentpane

```
String[] items = {"Item 1", "Item 2", "Item 3"};
linkList = new JList<String>(items);
createPanels();
// set up split panes
JSplitPane splitPane = new JSplitPane(
    JSplitPane.HORIZONTAL_SPLIT, linkList, cards);
splitPane.setDividerLocation(50);
add(splitPane, BorderLayout.CENTER);
```

Setting up a CardLayout



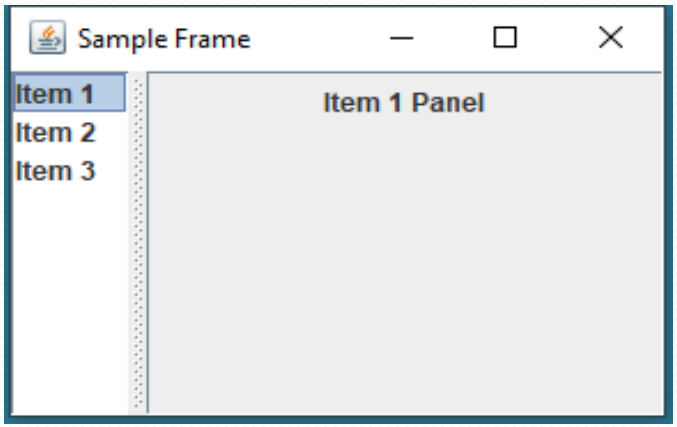

```
JPanel panel1 = new JPanel();
JLabel label1 = new JLabel("Item 1 Panel");
panel1.add(label1);

JPanel panel2 = new JPanel();
JLabel label2 = new JLabel("Item 2 Panel");
panel2.add(label2);

JPanel panel3 = new JPanel();
JLabel label3 = new JLabel("Item 3 Panel");
panel3.add(label3);
cards = new JPanel(new CardLayout());
cards.add(panel1, "Item 1");
cards.add(panel2, "Item 2");
cards.add(panel3, "Item 3");
```

- Create JPanels that you wish to present
- Create a new JPanel that will be given CardLayout as its LayoutManager. (Here, this JPanel is called cards.)
- Add your panels to the CardLayout and specify in the second argument a key that can be used to locate each JPanel in the CardLayout. Here, the keys are "Item 1", "Item 2", and "Item 3".

Controlling CardLayout with JList

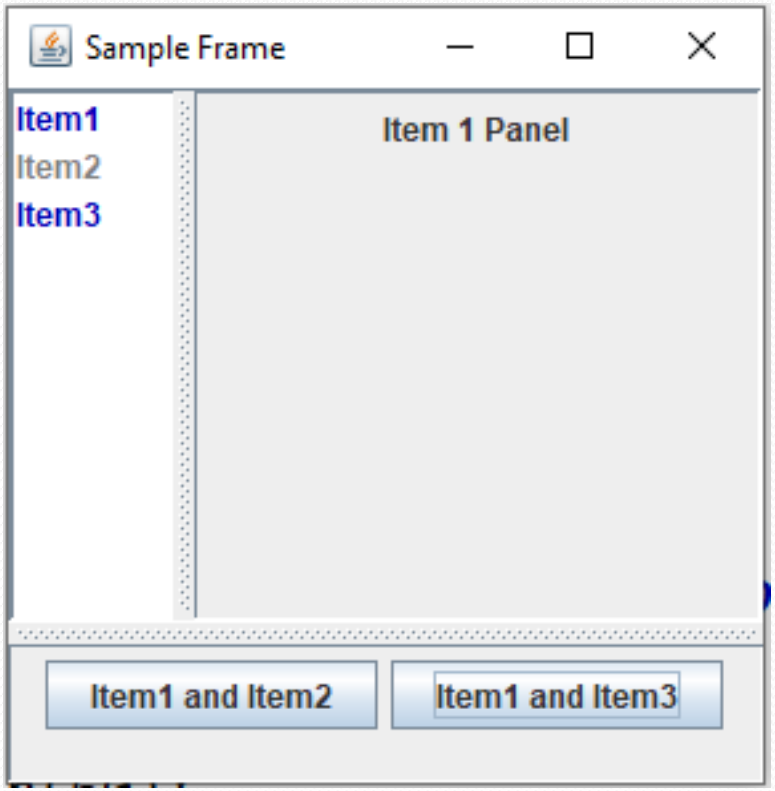


- Use a ListSelectionListener to connect the JList to the CardLayout
- When a list item is selected, read the selected value
- Use the value as a key to ask CardLayout to display the corresponding JPanel.

```
//connect JList elements to CardLayout panels
linkList.addListSelectionListener(event -> {
    String value = linkList.getSelectedValue().toString();
    CardLayout cl = (CardLayout) (cards.getLayout());
    cl.show(cards, value);
});
```

Demo: lesson6.lecture.splitpane

Control Function and Appearance of JList Using a CellRenderer



- A CellRenderer allows you to control the color of the list items and to dynamically disable/enable their function for displaying panels in the right pane.
- For this kind of control, the JList needs to be created using a ListModel and items inserted into the Model will have both a String value (which is displayed in the left panel) and another value that will determine the properties of each list item (its color, and whether enabled or disabled).
- See the demo:
  lesson6.lecture.cellrenderer

Connecting the Parts of Knowledge With the Wholeness of Knowledge

The self-referral dynamics
arising from the reflexive association of container classes

1. In Swing, components are placed and arranged in container classes for attractive display.
2. In Swing, containers are also considered to be components; this makes it possible to place and arrange container classes inside other container classes. These self-referral dynamics support a much broader range of possibilities in the design of GUIs.

---

3. **Transcendental Consciousness:** TC is the self-referral field of all possibilities.
4. **Wholeness moving within itself**: In Unity Consciousness, all activity is appreciated as the self-referral dynamics of one's own Self.