



Core Java Course
Java 1.2 User Interfaces
(Chapter 9)
Prof. Jeff Blessing
MSOE



Agenda

- ❖ Homework
- ❖ Model-View-Controller Design Pattern
- ❖ Implementing UIs in Java
- ❖ Canvases
- ❖ Text Components
 - text boxes, etc.
- ❖ User Selection Components
 - check boxes, radio buttons, etc.
- ❖ Scrolling
- ❖ Layout Managers
- ❖ Menus
- ❖ Homework Assignment


2



Model-View-Controller (MVC)

- ❖ A *design pattern* popular with UIs
- ❖ Model
 - the underlying representation of the component
- ❖ View
 - the way the user sees the component
- ❖ Controller
 - a *handler* which interacts with the user and the component


3



MVC Example

- ❖ Text inside a text box
 - Model
 - ◆ a string in memory which holds the characters
 - View
 - ◆ the font and point size in which the string is currently displayed
 - Controller
 - ◆ the callback routine (like `actionPerformed()`) which implements some behavior of the object
- ❖ Changing the font doesn't change the model


4



User Interface Components

- ❖ UI Components in Java are common elements of GUI applications
- ❖ Each component has a platform independent purpose and common behavior. (Certain details/features may be platform dependent, so that components would look familiar to users)
- ❖ Only components available for ALL platforms are included into AWT
- ❖ Example: buttons, lists, scroll bars, text fields, etc.

5



Extending Standard Components

- ❖ Custom components can be built by subclassing the "standard" components.
- ❖ Custom components (also called Lightweight Components) don't have to use native OS interfaces and allow for a uniform look across multiple platforms.
- ❖ Example: Extending Java AWT by using Java Foundation Classes (JFC, SunSoft, Netscape) and Application Foundation Classes (AFC, Microsoft)

6

Example: Building User Interface

- ❖ Design the form layout
- ❖ Select proper components
- ❖ Add components to a container (a Frame or a Panel)
- ❖ Position components (and sub-containers) inside the container using a Layout Manager
- ❖ In order to process user input, add code for handling component events
- ❖ BuildingUI Code Example

7

Canvases

- ❖ An AWT component (replaced by JFrame in Swing)
- ❖ Canvases (class Canvas) allow one to isolate graphics output from other components or different graphics outputs between themselves
- ❖ In order to use a Canvas component we need to perform the following steps
 - Derive a class from class Canvas
 - Override the paint method in the new class in order to provide appropriate graphics output
 - Instantiate the derived class
 - Add the instance to the container (e.g., frame)

8

Text Input Components

- ❖ Text input components are subclasses of `TextComponent` (an abstract class)
 - `void setText(String text);`
 - `String getText();`
 - `void setEditable(boolean editable);`
- ❖ `TextField` - single line text input component
 - `TextField(String text, int columns);`
 - `setColumns(int columns); // call validate();`
- ❖ `TextArea` - multiple line text input
 - `TextArea(String text, int rows, int columns, int scrollbars);`

9

Swing Text Input Components

- ❖ Text input components are subclasses of `JTextComponent` (an abstract class)
 - `void setText(String text);`
 - `String getText();`
 - `void setEditable(boolean editable);`
- ❖ `JTextField` - single line text input component
 - `JTextField(String text, int columns);`
 - `setColumns(int columns); // call validate();`
- ❖ `JTextArea` - multiple line text input
 - `JTextArea(String text, int rows, int cols)`
 - ◆ scrollbars replaced by putting `JTextArea` in a `JScrollPane`

10

Text Input Components (cont.)

- ❖ Text in a `TextComponent` can be selected programmatically
 - `void selectAll();`
 - `void select(int startSel, int endSel);`
- ❖ To check if there is selected text, one can use
 - `int getSelectionStart();`
 - `int getSelectionEnd();`
 - `String getSelectedText();`
- ❖ Labels (class `Label`) are for displaying static text
 - Swing has class `JLabel`
- ❖ `TextTest` and `TextAreaTest` Code Examples

11

User Selection Components

- ❖ Check Box component (`Checkbox` class)
- ❖ Radio Button components (`Checkbox` and `CheckboxGroup` classes)
- ❖ Choice Box or Drop Down List component (`Choice` class)
- ❖ List component (`List` class)

12

Swing Selection Components

- ❖ Check Box component (JCheckBox class)
- ❖ Radio Button components (JRadioButton and JRadioButtonMenuItem classes)
- ❖ Choice Box or Drop Down List component (JComboBox class)
- ❖ List component (JList class)
- ❖ CheckBoxTest.java Code Example

13

User Selection Components (cont.)

- ❖ Checkbox class methods
 - Checkbox(String lbl, boolean state);
 - boolean getState();
 - void setState(boolean state);
- ❖ CheckBoxTest Example
- ❖ CheckboxGroup class example
 - CheckboxGroup grp = new CheckboxGroup();
 - Checkbox btnDark = new Checkbox("Dark", grp, true);
 - Checkbox btnLight = new Checkbox("Light", grp, false);

14

Radio Buttons

- ❖ Mutually exclusive set of check boxes
 - only one can be selected from the set
- ❖ The AWT does not contain radio buttons
 - Swing does implement two radio button classes
- ❖ JRadioButton class creates new buttons
- ❖ ButtonGroup forms the set of related radio buttons (from which **one** is selected)
- ❖ RadioButtonTest.java Code Example

15

User Selection Components (cont.)

- ❖ Choice class
 - Choice();
 - void add(String item);
 - void insert(String item, int index);
 - void remove(String item);
 - void remove(int index);
 - void removeAll();

16

Swing Choices

- ❖ There is no JChoice class!
 - instead, there is a JComboBox class for dealing with choices
- ❖ JComboBox -- a combination of a text field and drop-down list that lets the user either type in a value or select it from a list that is displayed when the user asks for it. The editing capability can also be disabled so that the JComboBox acts only as a drop down list.

17

User Selection Components (cont.)

- ❖ List class
 - List(int rows, boolean multSel);
 - void add(String item, int index);
 - String getItem(int index);
 - void select(int index);
 - String[] getSelectedItems();
 - void makeVisible(int index);

18

Swing List Components

- ❖ JList class
 - JList(Object items[]);
 - void addListSelectionListener(ListSelectionListener);
- ❖ A separate model, ListModel, represents the contents of the list
- ❖ Best explained by an example!
- ❖ LongListTest.java Code Example

19

JList Model Example

```
//Create a JList that displays the strings in data[]
String[] data = {"one", "two", "free", "four"};
JList dataList = new JList(data);

//The value JList model property is an object that
//provides a read-only view of the data. It was
//constructed automatically.
for(int i = 0; i < dataList.getModel().getSize(); i++)
{ System.out.println(dataList.getModel().get(i));
}
...
```

20

Scrolling JLists

- ❖ JList doesn't support scrolling directly. To create a scrolling list you make the JList the *viewport* view of a JScrollPane

```
...
JScrollPane scrollPane = new JScrollPane(dataList);
// Or in two steps:
JScrollPane scrollPane = new JScrollPane();
scrollPane.getViewport().setView(dataList);
```

21

Item Events

- ❖ When selection changes the listener's method `itemStateChanged(ItemEvent evt)` is called
- ❖ To obtain more information about the changed item and its current state the following methods can be invoked on the `evt` object
 - String label = (String) evt.getItem();
 - Checkbox src = (Checkbox) evt.getSource();
 - boolean state = src.getState();
- ❖ There is no JItem class

22

Scrollbar Components

- ❖ Scrollbars can be used for changing an input value within a given range and for scrolling a graphics image when it is larger than the output window
- ❖ Class Scrollbar
 - Scrollbar(int orientation);
 - Scrollbar(int orientation, int val, int visibleSize, int min, int max);
 - void setValue(int value);
 - void setValues(val, vis, min, max);
 - int getValue();
 - void setUnitIncrement(int increment);
 - void setBlockIncrement(int increment);

23

Scrollbar Events

- ❖ Scrollbar component generates AdjustmentEvents. Its listener has `adjustmentValueChanged(AdjustmentEvent evt)` method called every time a scrollbar is adjusted
- ❖ To collect information about the change we can use the event object
 - int value = evt.getValue();
 - int adjType = evt.getAdjustmentType();
 - Adjustment types: TRACK,
 - UNIT_INCREMENT, UNIT_DECREMENT,
 - BLOCK_INCREMENT, BLOCK_DECREMENT

24

Swing Scrollbars

- ❖ `JScrollBar` implements most of the same functionality of AWT Scrollbars
- ❖ Typically, as the position of the knob in the scrollbar changes, a corresponding change is made to the position of the `JViewport` on the underlying view, changing the contents of the `JViewport`
- ❖ `ColorSelect.java` Code Example

25

Scrolling Panes (Java 1.1)

- ❖ Even though scrolling a window can be implemented using Scrollbar components, it is much easier to achieve the same goal with Scrolling Panes (`ScrollPane` class)
- ❖ Example:
 - `ScrollPane sp = new ScrollPane();`
 - `sp.add(TextCanvas);`
 - `sp.SetSize(maxWidth, maxHeight);`
 - `add(sp);`
 - `pack();` // method on Window class

26

Layout Managers

- ❖ Layout Manager is an object provided by AWT. Layout Manager allows the user to arrange UI components inside every container.
- ❖ Layout Manager classes
 - `FlowLayout`
 - `BorderLayout`
 - `CardLayout`
 - `GridLayout`
 - `GridBagLayout`
- ❖ You can create custom Layout Manager or no Layout Manager at all (use `setBounds()` method)

27

Layout Managers (cont.)

- ❖ Class `FlowLayout`
 - `FlowLayout(int align);`
 - // Values: `CENTER`, `LEFT`, `RIGHT`
 - `FlowLayout(int alignment, int hgap, int vgap);`
 - `GridLayout`
 - `GridBagLayout`
- ❖ Class `BorderLayout`
 - `BorderLayout(int hgap, int vgap);`
 - `add(Component cmp, String position);`
 - // Position Values: `"North"`, `"South"`, `"West"`, `"East"`, `"Center"`

28

Layout Managers (cont.)


- ❖ Class `GridLayout`
 - `GridLayout(int hdim, int vdim);`
- ❖ Class `CardLayout`
 - Card Layout can be used to simulate Windows Property Sheets/Property Pages
 - `CardLayoutTest` Code Example

29

Menus

- ❖ Three important classes define menus:
 - `JMenuItem`, `JMenu`, `JMenuBar`
- ❖ `JMenuItem` contains text and an optional icon to be displayed in the menu
 - a listener is associated with each `JMenuItem`
- ❖ `JMenu` is a drop-down menu of items
- ❖ `JMenuBar` is a horizontal bar across the top of the frame

30



Menu Example

```
JMenuBar menuBar = new JMenuBar();
JMenu menu = new JMenu("Edit");
item = new JMenuItem("Cut");
item.addActionListener(this);
menu.add(item);
item = new JMenuItem("Copy");
item.addActionListener(this);
menu.add(item);
item = new JMenuItem("Paste");
item.addActionListener(this);
menu.add(item);
menu.addSeparator();
...
menuBar.add(menu);
```

31



Other Predefined Dialogs

- ❖ JOptionPane has static methods:
 - showMessageDialog(), showConfirmDialog(), showOptionDialog(), showInputDialog()
- ❖ Password Dialog Box
 - see DataExchangeTest.java
- ❖ File Dialog Box
 - JFileChooser class
 - also FileView and FileFilter classes

32



Homework Assignment

- ❖ Implement ShapeTest, employing different user interface components (panels, buttons, check boxes, radio buttons, lists, drop-down lists, scrolling panes)
- ❖ Try different Layout Managers (at least FlowLayout, BorderLayout, GridLayout)

33



34