

Graphical User Interfaces

(Lecture 5)

By G. Sheblé (Spring 2004), modified slightly
by D. Berleant (Fall 2004) with permission

© 2004 by G. Sheblé

Java GUIs

- Java SDK contains two Graphical User Interfaces (GUIs)
 - Abstract Windowing Toolkit (AWT)
 - original Java GUI
 - Swing package
 - newer, more flexible

How GUIs Work

- GUIs provide a familiar environment (push buttons, menus, drop-down lists, text fields, etc.)
- GUI-based programs must be ready for mouse clicks or keyboard input to any component at any time
- Mouse clicks or keyboard inputs are *events*, thus GUI-based programs are *event-driven*

GUI Elements

- GUIs:
 - **Components**, elements on the screen (push buttons, text fields, etc.)
 - **Container**, holds components (`JPanel`, `JFrame`).
 - **Layout Manager**, controls placement components within container.
 - **Event handlers** to respond to events (mouse clicks or keyboard inputs)

Creating

- Create a container class to hold components
- Create a layout manager
- Create components
- Create “listener” objects to detect and respond to events
- Register listeners with components
- Create `JFrame` object, place container in **content pane**

Events

- **Event** is object created by external action (mouse click, key press, etc.)
- Event occurrence, *Java automatically sends event to that GUI object.*

Event Handling

- When event is received, object checks for listener object (*programmer-created*) to receive event, forwards event to predefined method name (but method body is written by you)
- This method is the **event handler** (performs whatever steps are required).

Event Handling

- Example:
 - mouse click on button is event
 - event is sent to `JButton` object
 - `JButton` object sends event to handler method of the `ButtonHandler` object
 - calls method `updateLabel` (callback)
 - `updateLabel` updates the label

JLabel

- **label** is object that displays a single line of *read-only text and/or an image*.
 - It does *not* respond
- **Constructors:**

```
public JLabel();  
public JLabel(String s);  
public JLabel(String s, int horizontalAlignment);  
public JLabel(Icon image);  
public JLabel(Icon image, int horizontalAlignment);  
public JLabel(String s, Icon image, int  
    horizontalAlignment);
```

JLabel

Method	Description
<code>public Icon getIcon()</code>	returns image from JLabel
<code>public String getText()</code>	returns text from JLabel
<code>public void setIcon(Icon image)</code>	sets JLabel image
<code>public void setText(String s)</code>	set JLabel text
<code>public void setHorizontalAlignment(int alignment)</code>	sets horizontal alignment of JLabel text and image (left, center, right)
<code>public void setHorizontalTextPosition(int textPosition)</code>	sets position of text to image (left, center, right)
<code>public void setVerticalAlignment(int alignment)</code>	sets vertical alignment of JLabel text and images (top, center, bottom)
<code>public void setVerticalTextPosition(int textPosition)</code>	sets position of text relative to image (top, center, bottom)

JButton

- JButton **class creates pushbutton**
 - When one clicks on button, an event is generated and sent to any registered listeners.
- **Constructors:**

```
public JButton();
```

```
public JButton(String s);
```

```
public JButton(Icon image);
```

```
public JButton(String s, Icon image);
```

ActionEvent

- **JButtons generate** `ActionEvent` **objects** when an event occurs.
- **Listener class for these events must implement** `ActionListener` **interface and** `actionPerformed` **method.**
- **Listener object must be registered.**
- **When event occurs,** `actionPerformed` **method of corresponding listener object is called**

JTextField

- `JTextField` class creates text field.
 - When one types text and presses ENTER key, `ActionEvent` is generated and sent to any registered listeners.
- **Constructors:**

```
public JTextField();  
public JTextField(int cols);  
public JTextField(String s);  
public JTextField(String s, int cols);
```

JPasswordField

- JPasswordField **class** = JPasswordField class, except text is not visible. Instead, string of asterisks are shown.
- **Constructors:**

```
public JPasswordField();  
public JPasswordField(int cols);  
public JPasswordField(String s);  
public JPasswordField(String s, int cols);
```

JComboBox

- **JComboBox** class one can either type text or select a choice from drop down list.
- **Constructors:**

```
public JComboBox();  
public JComboBox( Object[] );  
public JComboBox( Vector );
```
- **User is forced to select from drop down list only by using method**
`setEditable(false)`

JCheckBox

- `JCheckBox` class creates button that toggles between “on” and “off”.
- Small box with a check mark
- Selected Constructors:

```
public JCheckBox( String s, boolean state );  
public JCheckBox( Icon image, boolean state );  
public JCheckBox( String s, Icon image, boolean state  
    );
```

JRadioButton

- JRadioButton **class creates radio button, small circle with dot in center when “on”.**
- **Selected Constructors:**

```
public JRadioButton( String s, boolean state );  
public JRadioButton( Icon image, boolean state );  
public JRadioButton( String s, Icon image, boolean  
state );
```

Button Groups

- Radio buttons
 - grouped together are *button groups*.
 - One button may be on at a time (others forced off)

- Constructor:

```
public ButtonGroup();
```

- Methods:

```
public void add(AbstractButton b);    // Add  
    button to group
```

```
public void remove(AbstractButton b); // Remove  
    from group
```

Layout Managers

Element	Description
BorderLayout	lays out elements in central region and four surrounding regions (default for JFrame)
BoxLayout	lays out elements in row horizontally or vertically, no wrap around (default for Box)
CardLayout	stack components like deck of cards, only top visible
FlowLayout	lays out elements left-to-right and top-to-bottom within container (default for JPanel)
GridBagLayout	lays out elements in flexible grid, size of each element may vary
GridLayout	lays out elements in a rigid grid

BorderLayout

- BorderLayout manager puts components into one of five regions (*North, South, East, West, and Center*)

- Constructors:

```
public BorderLayout();
```

```
public BorderLayout(int horizontalGap, int  
verticalGap);
```

- Layout manager is associated:

```
setLayout( new BorderLayout() );
```

- Components added to specific regions:

```
add(new Button("North"), BorderLayout.NORTH);
```

- Default layout manager for JFrame

FlowLayout

- **FlowLayout** manager puts component order from left to right and top to bottom
- **Constructors:**

```
public FlowLayout();  
public FlowLayout(int align);  
public FlowLayout(int align, int horizontalGap,  
                  int verticalGap);
```
- **Alignment** (LEFT, RIGHT, or CENTER)
- **Default layout manager for JPanel**

GridLayout

- GridLayout manager puts components into rectangular grid structure.

- Constructors:

```
public GridLayout(int rows, int cols);
```

```
public GridLayout(int rows, int cols, int horizGap,  
                 int vertGap);
```

- Components added from *left to right* and *top to bottom*

BoxLayout

- The `BoxLayout` manager puts components in single row or single column.
- Spacing and alignment of each element is individually controlled.
- Containers can be nested inside each other
- Constructor:

```
public BoxLayout(Container c, int direction);
```
- Direction is either `X_AXIS` or `Y_AXIS`
- Rigid areas and glue regions are used to space out components

Combining Layout Managers

- Use multiple containers inside, each with separate layout manager
- Top-level panel might be horizontal box, it may contain two or more panels of vertical boxes
- Result is component control of spacing in *both* directions