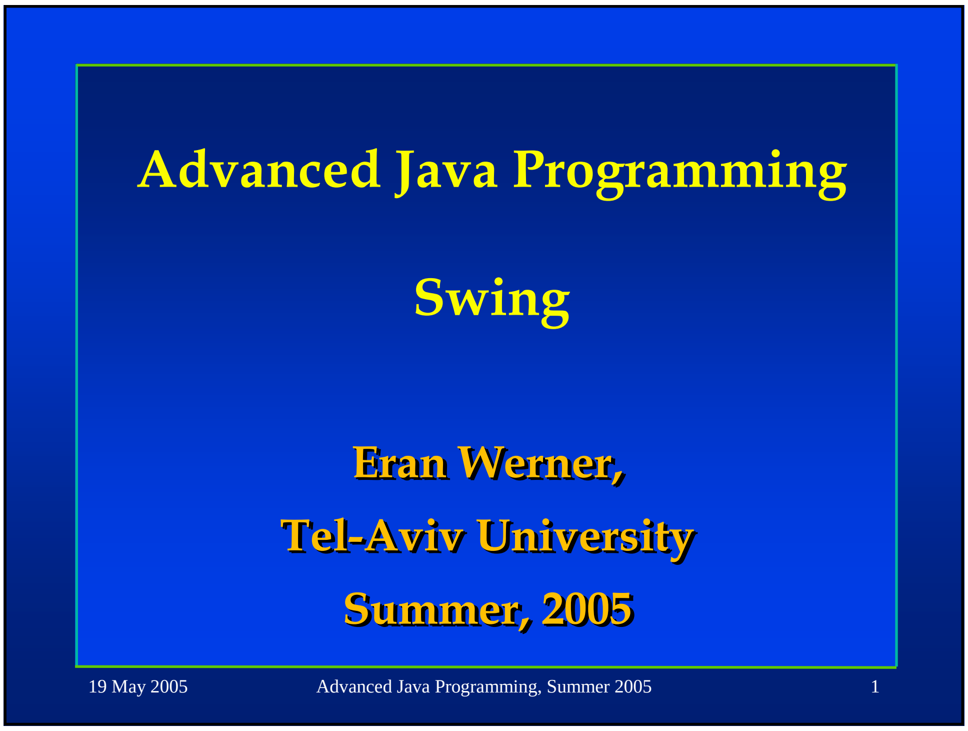


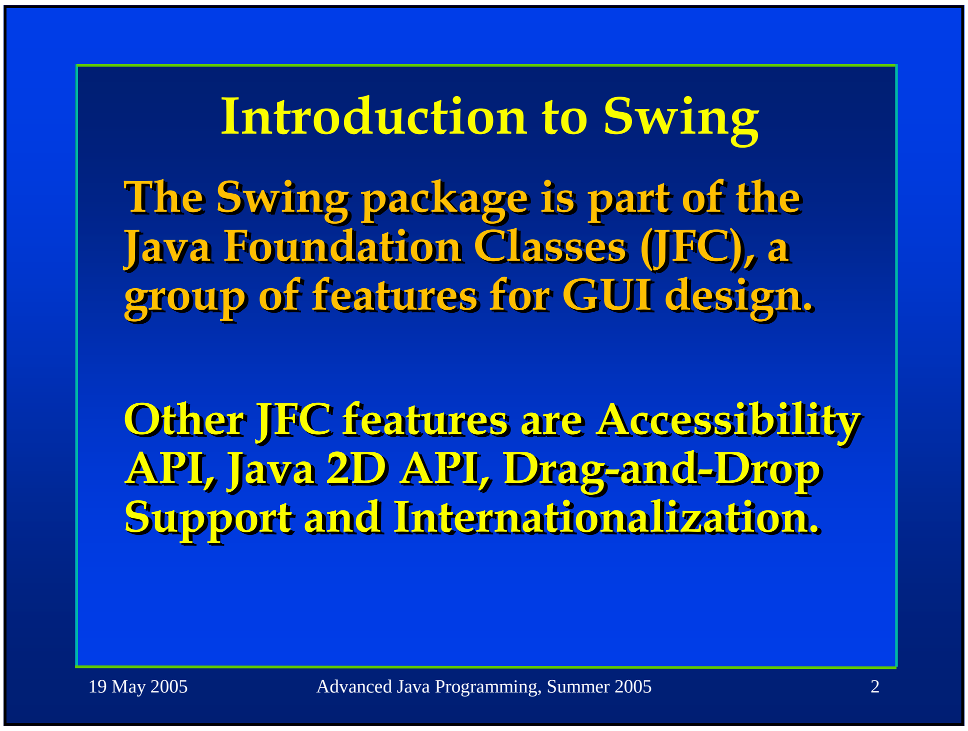
Advanced Java Programming

Swing

Eran Werner,

Tel-Aviv University

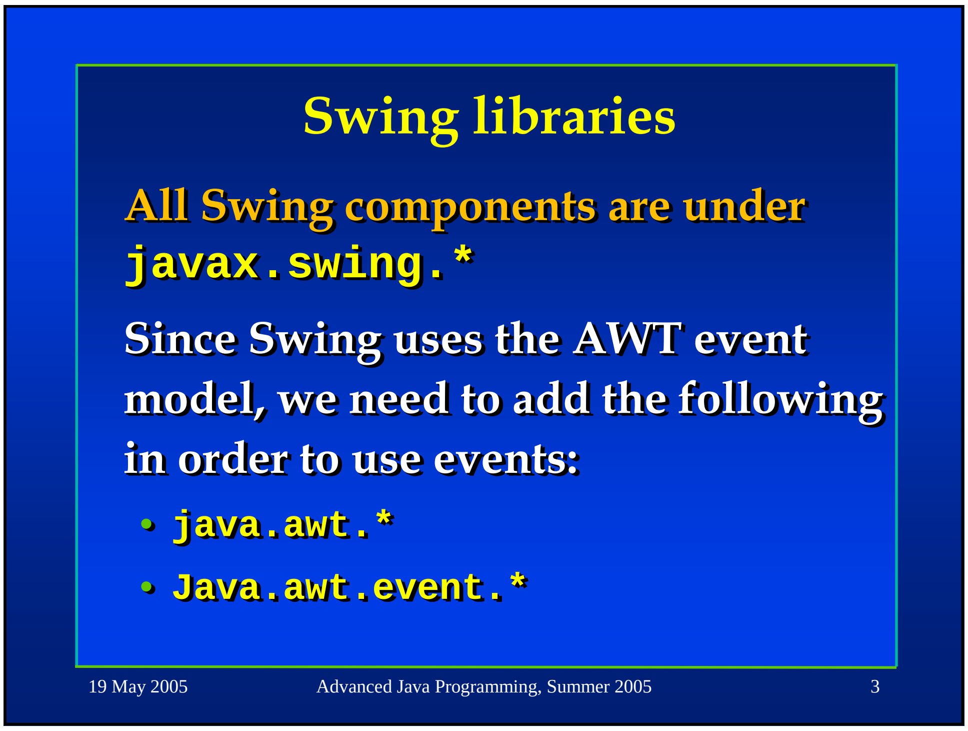
Summer, 2005



Introduction to Swing

The Swing package is part of the Java Foundation Classes (JFC), a group of features for GUI design.

Other JFC features are Accessibility API, Java 2D API, Drag-and-Drop Support and Internationalization.

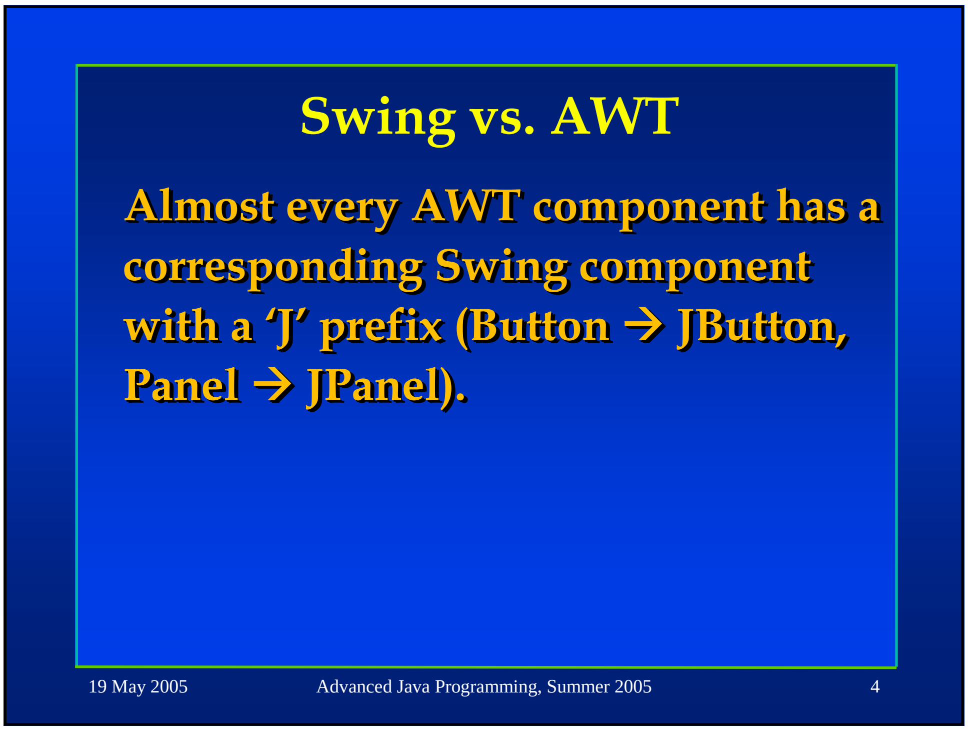


Swing libraries

All Swing components are under `javax.swing.*`

Since Swing uses the AWT event model, we need to add the following in order to use events:

- `java.awt.*`
- `Java.awt.event.*`

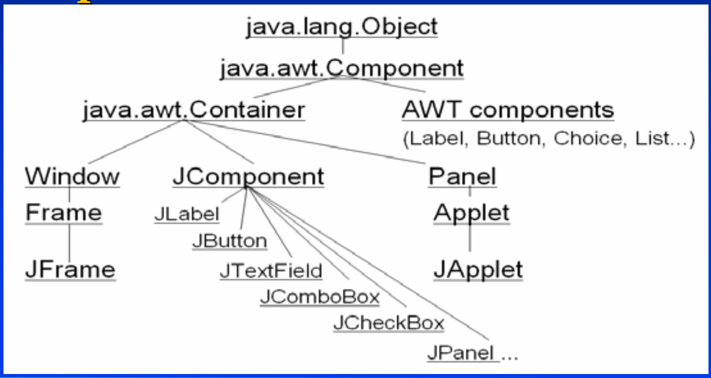


Swing vs. AWT

Almost every AWT component has a corresponding Swing component with a 'J' prefix (Button → JButton, Panel → JPanel).

Swing Vs. AWT

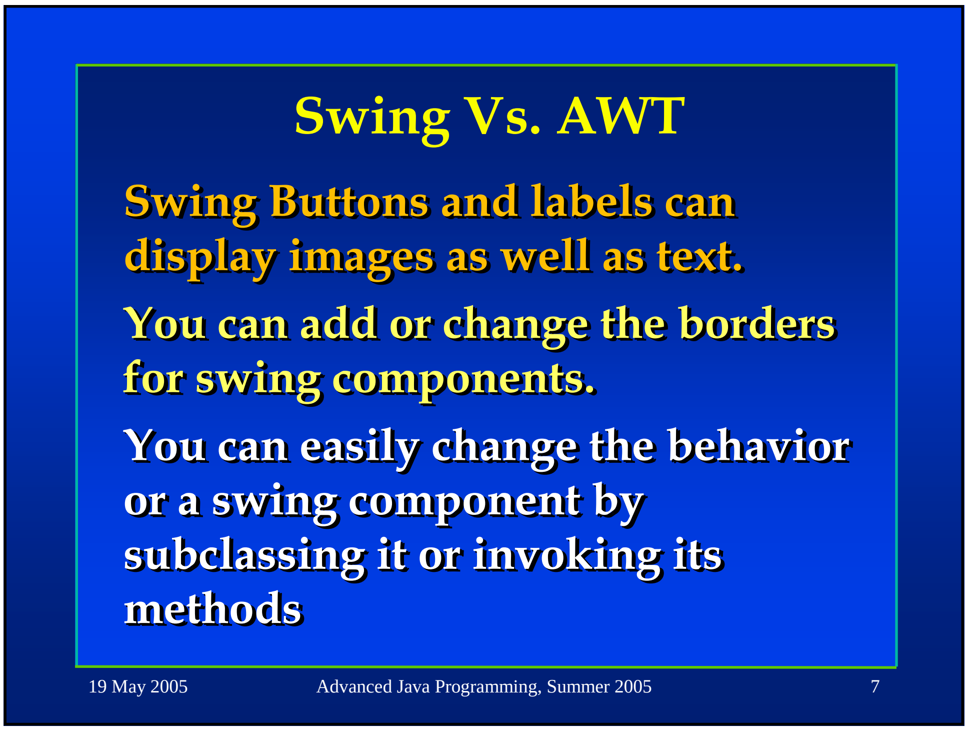
Lightweight components, platform independent.



Swing Vs. AWT

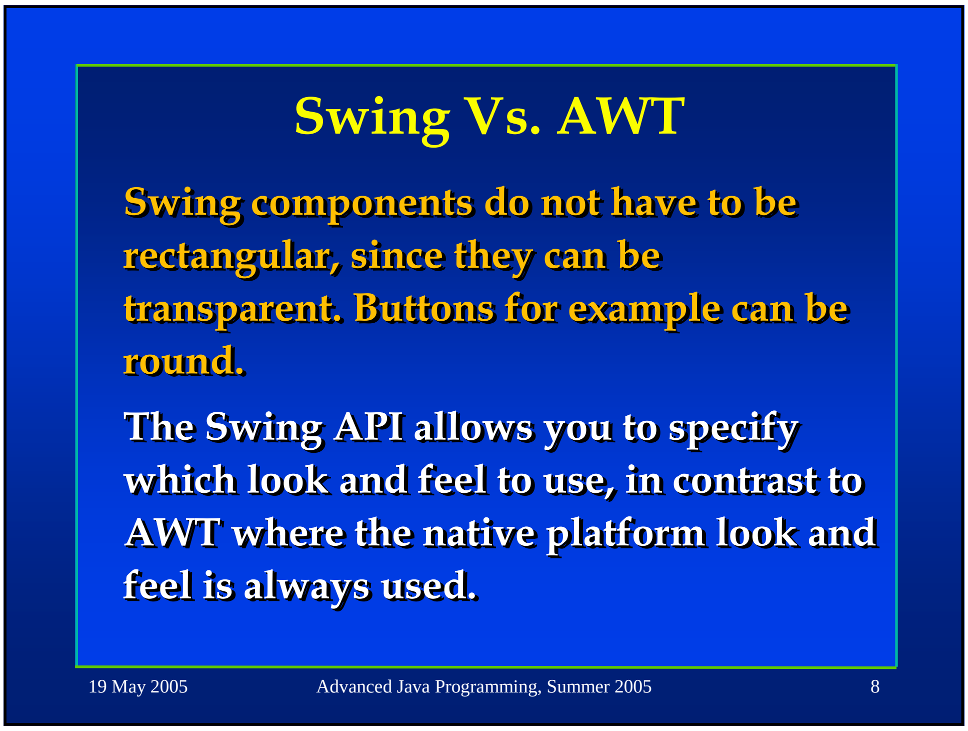
The main difference between AWT and Swing components is that swing components are implemented with absolutely no native code.

Swing components aren’t restricted to the features presented in every platform and therefore can have more functionality.



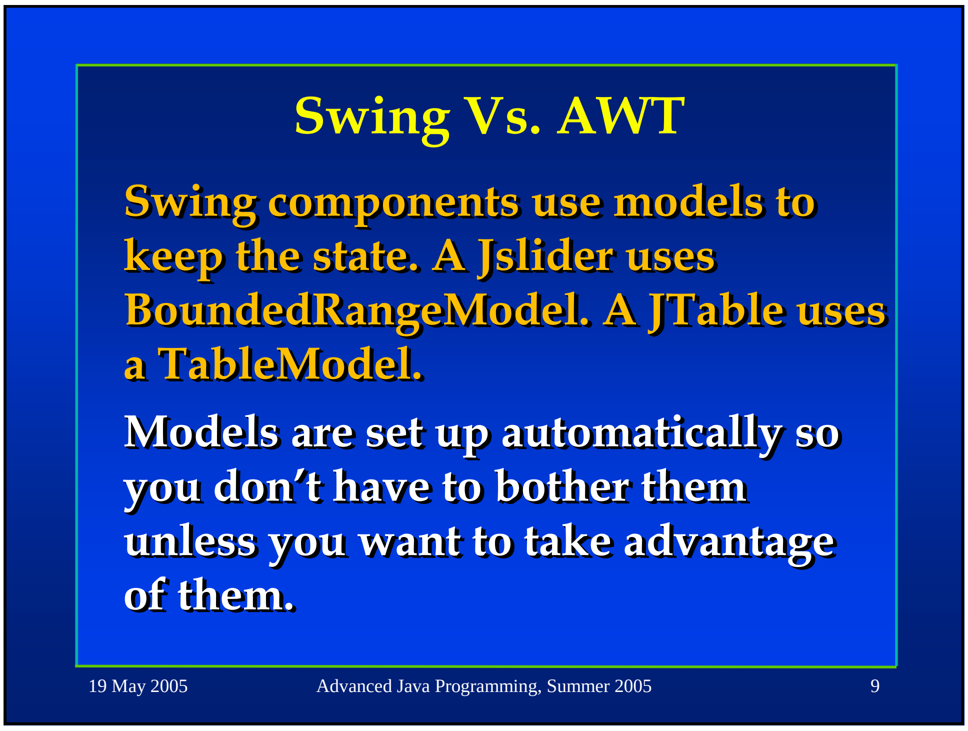
Swing Vs. AWT

- Swing Buttons and labels can display images as well as text.
- You can add or change the borders for swing components.
- You can easily change the behavior or a swing component by subclassing it or invoking its methods



Swing Vs. AWT

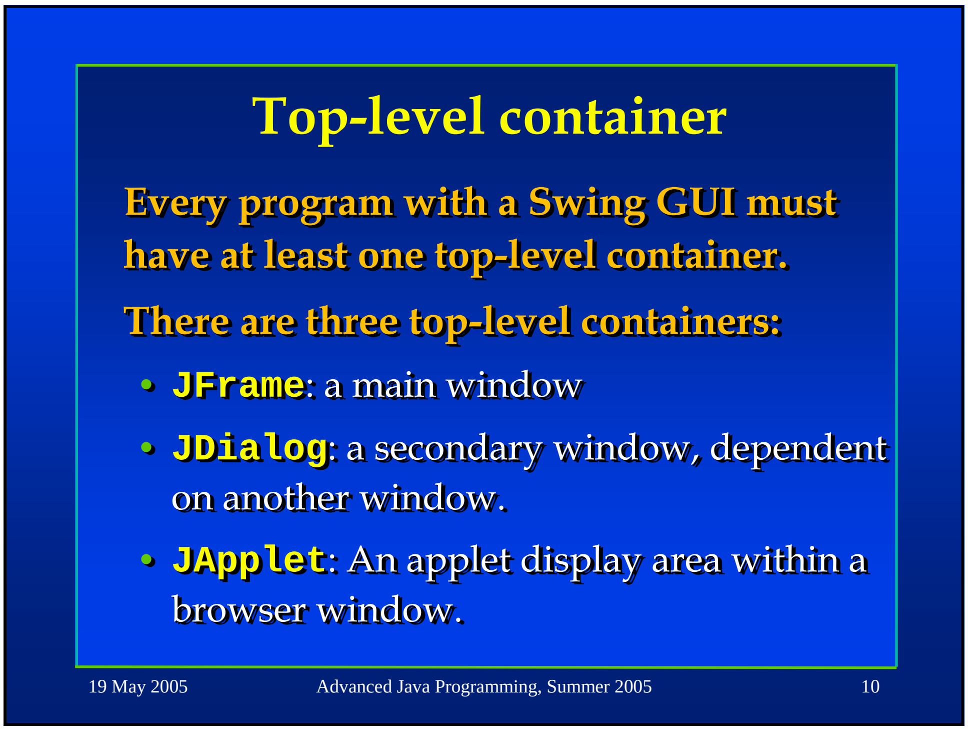
- Swing components do not have to be rectangular, since they can be transparent. Buttons for example can be round.
- The Swing API allows you to specify which look and feel to use, in contrast to AWT where the native platform look and feel is always used.



Swing Vs. AWT

Swing components use models to keep the state. A Jslider uses BoundedRangeModel. A JTable uses a TableModel.

Models are set up automatically so you don't have to bother them unless you want to take advantage of them.



Top-level container

Every program with a Swing GUI must have at least one top-level container.

There are three top-level containers:

- `JFrame`: a main window
- `JDialog`: a secondary window, dependent on another window.
- `JApplet`: An applet display area within a browser window.

JFrame

Setting up a frame:

```
JFrame frame = new JFrame("HelloWorldSwing");

// ... Add components to the frame

frame.pack();
frame.setVisible(true);
```

Adding a component to a frame:

```
frame.getContentPane().add(label);
```

Closing a frame:

```
frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
```

JLabel

- A component that displays text.
- Can also display an image.
- Does not react to input events.
- Cannot get the keyboard focus.

```
JLabel label = new JLabel("Hello World");
frame.getContentPane().add(label);
```

The JComponent Class

All Swing components whose names begin with "J" descend from the JComponent (except JFrame and JDialog – top level containers) .

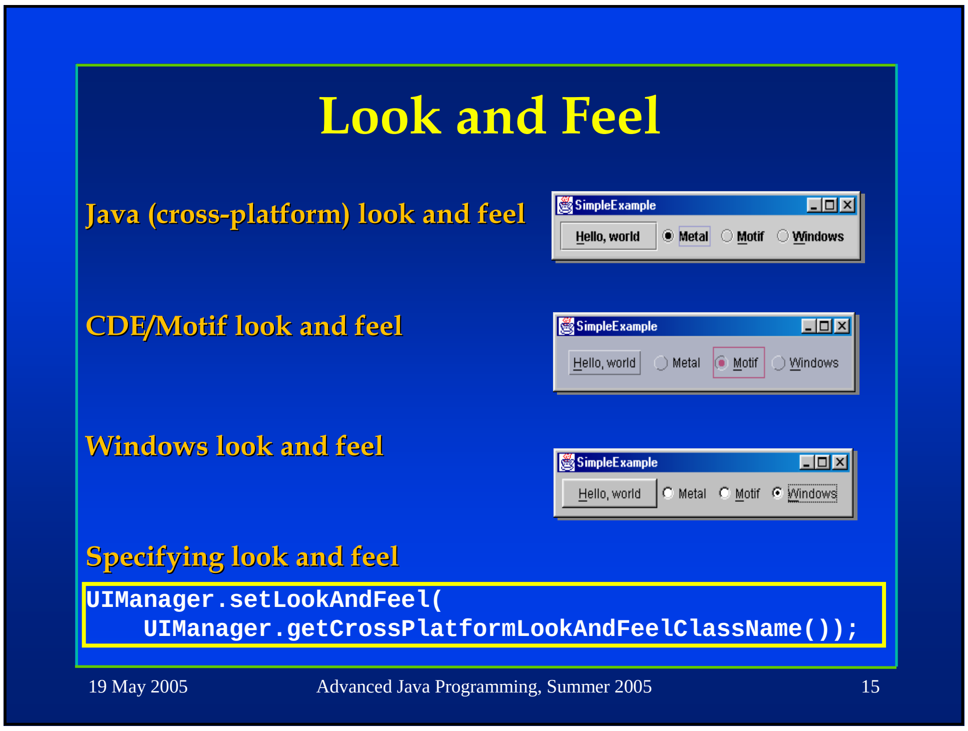
For example, JPanel, JScrollPane, JButton, and JTable.

JComponent extends java.awt.Container

The JComponent Class

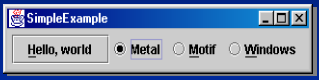
JComponent Features

- Tool tips
- Painting and borders
- Application-wide pluggable look and feel
- Support for drag and drop
- Double buffering
- Key bindings

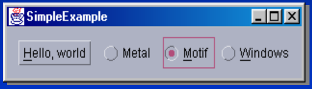


Look and Feel

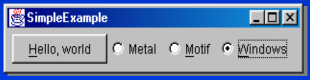
Java (cross-platform) look and feel



CDE/Motif look and feel

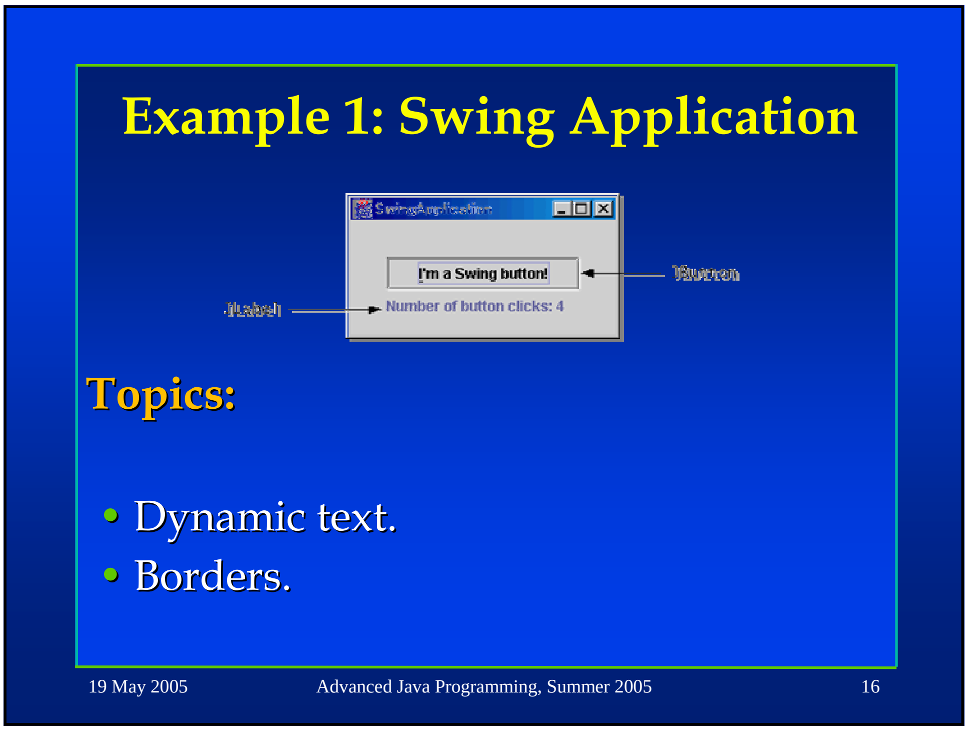


Windows look and feel

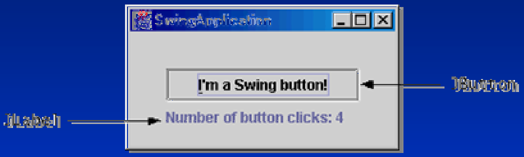


Specifying look and feel

```
UIManager.setLookAndFeel(
    UIManager.getCrossPlatformLookAndFeelClassName());
```



Example 1: Swing Application



Topics:

- Dynamic text.
- Borders.

Dynamic text

Creating a button

- The mnemonic functions as a hot key.
- The event handler updates the label's text when the button is clicked.

```
JButton button = new JButton("I'm a Swing button!");
button.setMnemonic('i');
button.addActionListener(new ActionListener() {
  public void actionPerformed(ActionEvent e) {
    numClicks++;
    label.setText(labelPrefix + numClicks);
  }
});
```

Borders

Every JComponent can have one or more borders.

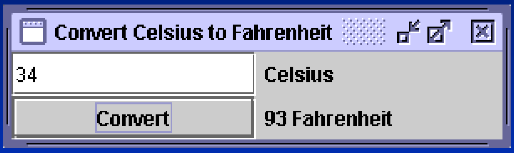
Borders are incredibly useful objects that, while not themselves components, know how to draw the edges of Swing components.

Borders

To put a border around a JComponent, you use its setBorder method. You can use the BorderFactory class to create most of the borders that Swing provides.

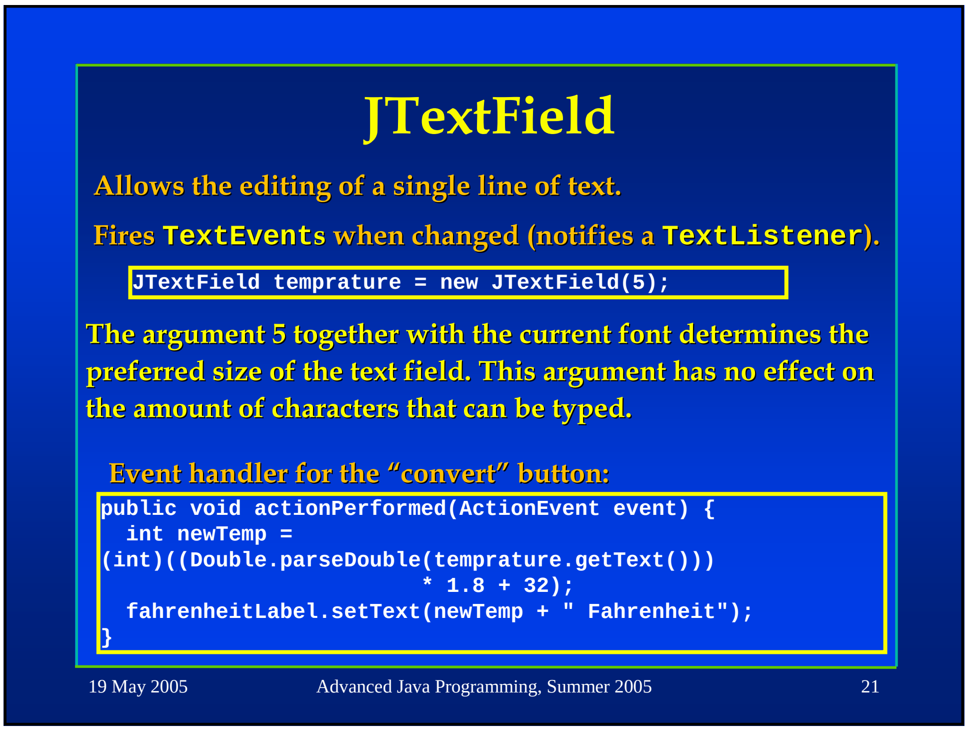
```
panel.setBorder
        (BorderFactory.createEmptyBorder(30,   //top
                                         30,   //left
                                         10,   //bottom
                                         30)); //right
```

Example 2: Celsius Converter



Topics:

- The JTextField component.
- The default button.
- Adding HTML.
- Icons.



JTextField

Allows the editing of a single line of text.

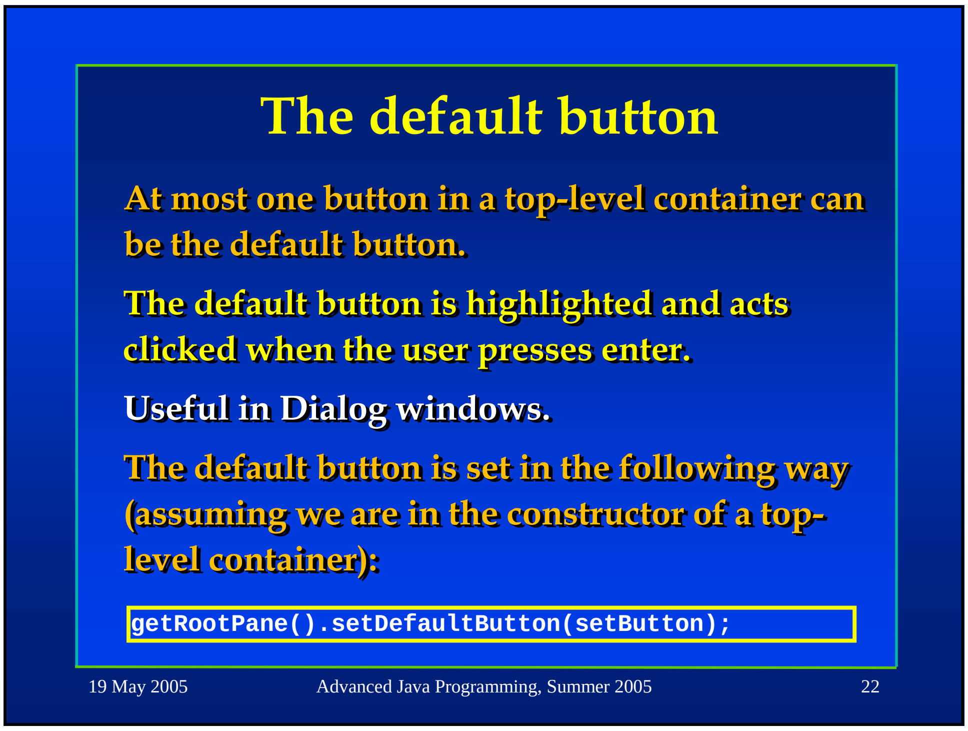
Fires `TextEvents` when changed (notifies a `TextListener`).

```
JTextField temprature = new JTextField(5);
```

The argument 5 together with the current font determines the preferred size of the text field. This argument has no effect on the amount of characters that can be typed.

Event handler for the “convert” button:

```
public void actionPerformed(ActionEvent event) {
  int newTemp =
(int)((Double.parseDouble(temprature.getText()))
                         * 1.8 + 32);
  fahrenheitLabel.setText(newTemp + " Fahrenheit");
}
```



The default button

At most one button in a top-level container can be the default button.

The default button is highlighted and acts clicked when the user presses enter.

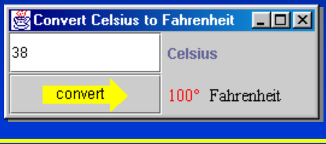
Useful in Dialog windows.

The default button is set in the following way (assuming we are in the constructor of a top-level container):

```
getRootPane().setDefaultButton(setButton);
```

Adding HTML

To add HTML to a component, use the <html>…</html> tag. HTML is useful for controlling fonts, colors, line breaks, etc.



```
if (tempFahr <= 32) {
  fahrenheitLabel.setText("<html><font color=blue>" + tempFahr
                        + "&#176  Fahrenheit </font></html>");
} else {
  fahrenheitLabel.setText("<html><font color=red>" + tempFahr
                        + "&#176  Fahrenheit </font></html>");
}
```

Icons

An icon usually refers to a descriptive fixed-size image.

Some components can be decorated with an icon.

Swing provides an interface called Icon.

It also provides a useful implementation of this interface: `ImageIcon`.

Icons

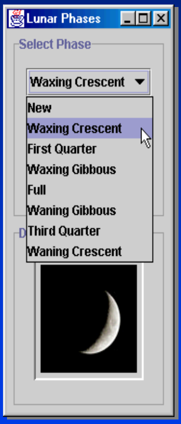
`ImageIcon` constructs an icon from a GIF or JPEG image.

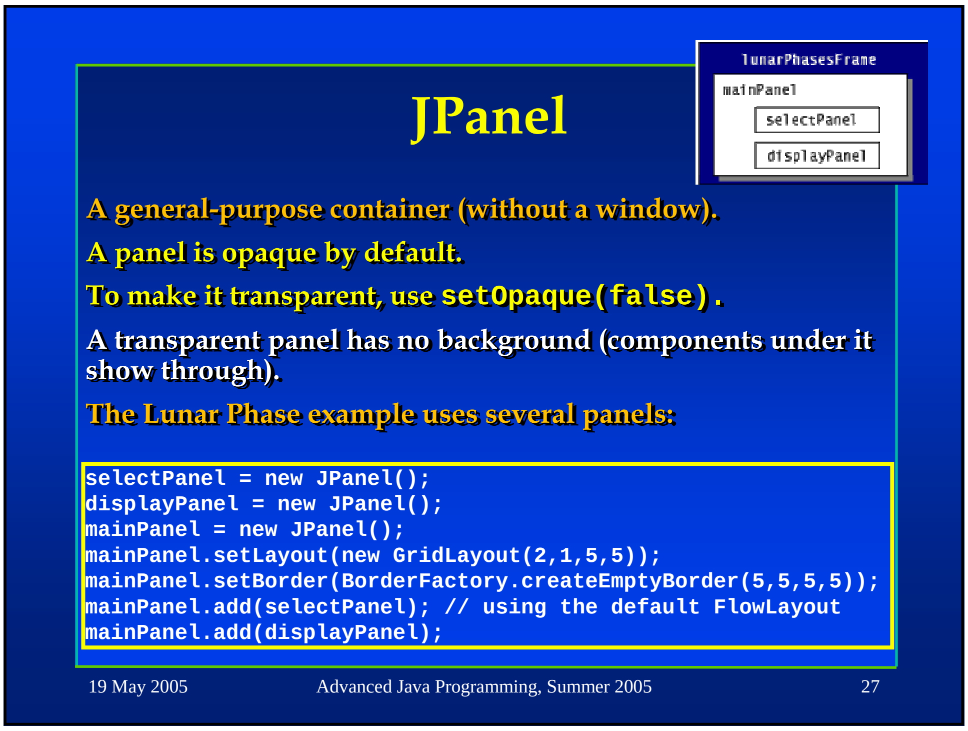
The following code adds the arrow icon to the “convert” button:

```
ImageIcon icon = new ImageIcon("images/convert.gif",
                              "Convert temperature");
JButton convertButton = new JButton(icon);
```

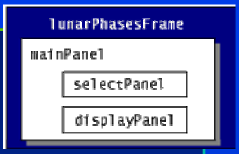
Example 3: Lunar Phases

- The JPanel component.
- Compound borders.
- The `JComboBox` component.
- Using multiple images.





JPanel



A general-purpose container (without a window).

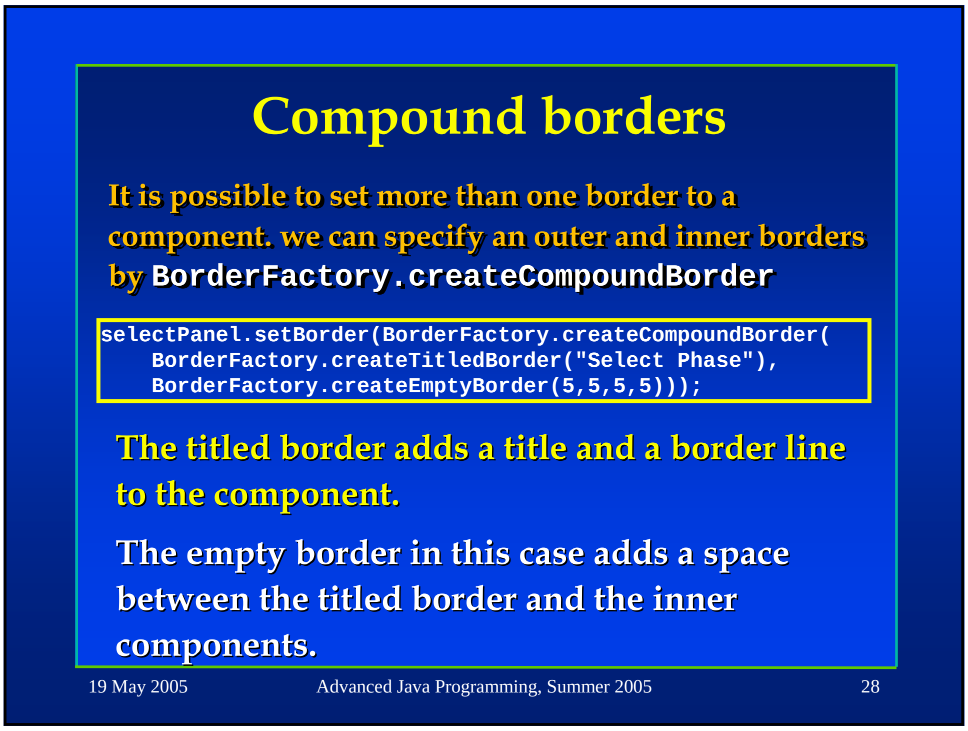
A panel is opaque by default.

To make it transparent, use `setOpaque(false)`.

A transparent panel has no background (components under it show through).

The Lunar Phase example uses several panels:

```
selectPanel = new JPanel();
displayPanel = new JPanel();
mainPanel = new JPanel();
mainPanel.setLayout(new GridLayout(2,1,5,5));
mainPanel.setBorder(BorderFactory.createEmptyBorder(5,5,5,5));
mainPanel.add(selectPanel); // using the default FlowLayout
mainPanel.add(displayPanel);
```



Compound borders

It is possible to set more than one border to a component. we can specify an outer and inner borders by `BorderFactory.createCompoundBorder`

```
selectPanel.setBorder(BorderFactory.createCompoundBorder(
    BorderFactory.createTitledBorder("Select Phase"),
    BorderFactory.createEmptyBorder(5,5,5,5)));
```

The titled border adds a title and a border line to the component.

The empty border in this case adds a space between the titled border and the inner components.

JComboBox

A component that enables user choice.

Can be editable allowing to dynamically add choices.

Constructed with an array of Strings. Icons can also be added.

An initial item can be selected using the `setSelectedIndex` method.

The selection is done by the item index.

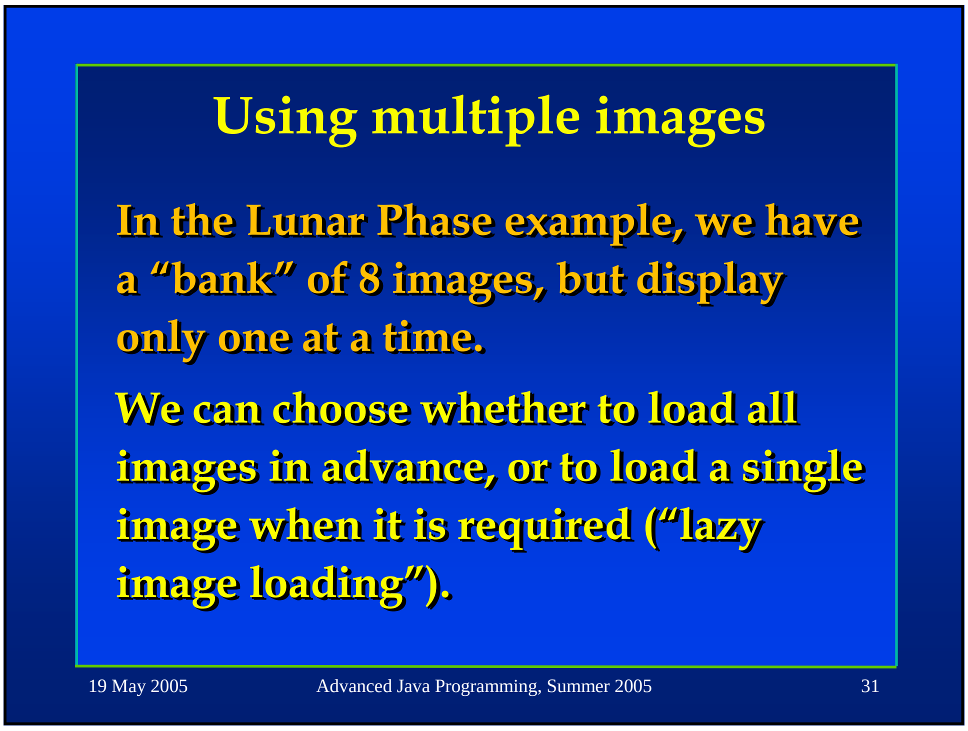
When the user starts writing an item the selection changes accordingly.

JComboBox

```
String[] phases = { "New", "Waxing Crescent“,
                   "First Quarter", "Waxing Gibbous",
                   "Full", "Waning Gibbous",
                   "Third Quarter", "Waning Crescent“
                  };
JComboBox phaseChoices = new JComboBox(phases);
phaseChoices.setSelectedIndex(START_INDEX);
```

An event handler for `ActionEvents` fired from a combo box.

```
public void actionPerformed(ActionEvent event) {
  if ("comboBoxChanged".
        equals(event.getActionCommand()))
    phaseIconLabel.setIcon(
        images[phaseChoices.getSelectedIndex()]);
  }
}
```

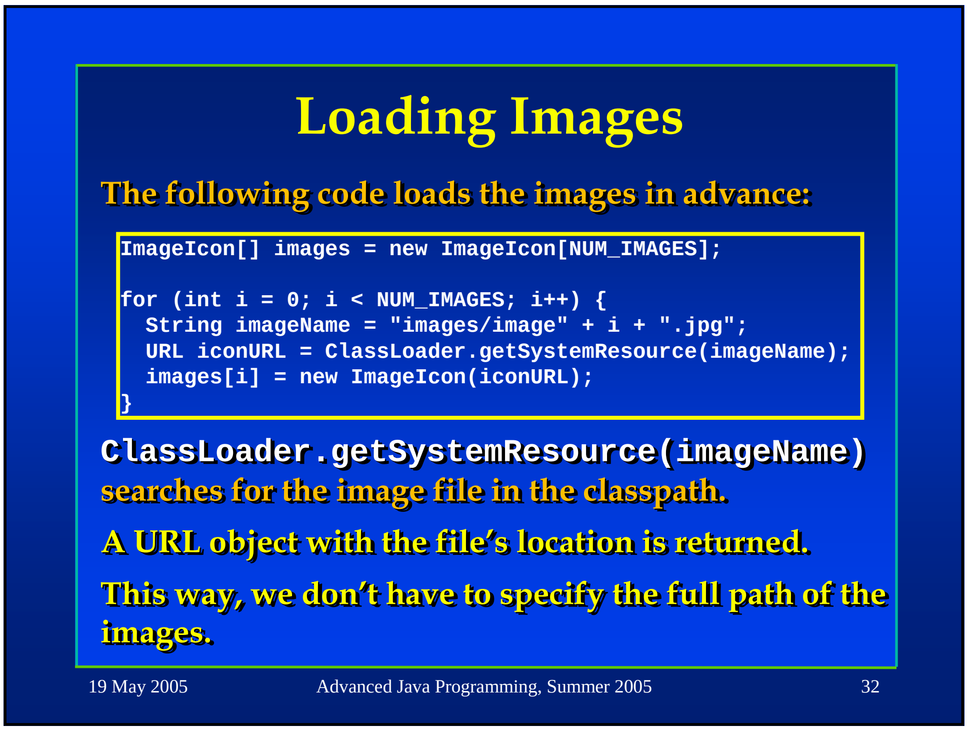


Using multiple images

In the Lunar Phase example, we have a “bank” of 8 images, but display only one at a time.

We can choose whether to load all images in advance, or to load a single image when it is required (“lazy image loading”).

19 May 2005 Advanced Java Programming, Summer 2005 31



Loading Images

The following code loads the images in advance:

```
ImageIcon[] images = new ImageIcon[NUM_IMAGES];

for (int i = 0; i < NUM_IMAGES; i++) {
  String imageName = "images/image" + i + ".jpg";
  URL iconURL = ClassLoader.getSystemResource(imageName);
  images[i] = new ImageIcon(iconURL);
}
```

`ClassLoader.getSystemResource(imageName)` searches for the image file in the classpath.

A URL object with the file’s location is returned.

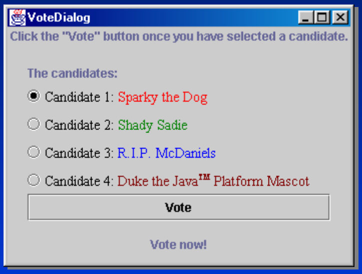
This way, we don’t have to specify the full path of the images.

19 May 2005 Advanced Java Programming, Summer 2005 32

Example 4: Vote Dialog

Topics:

- The `JRadioButton` component.
- Dialogs.
  - Displaying and customizing dialogs.
  - Receiving user input from a dialog.



Click the "Vote" button once you have selected a candidate.

The candidates:

[● Candidate 1: Sparky the Dog]

○ Candidate 2: Shady Sadie

○ Candidate 3: R.I.P. McDaniels

○ Candidate 4: Duke the Java™ Platform Mascot

Vote now!

JRadioButton

An item that can be selected or deselected.

For each group of radio buttons, you need to create a ButtonGroup instance and add each radio button to it.

ButtonGroup takes care of unselecting the previously selected button when the user selects another one in the group.

JRadioButton

```
JRadioButton[] radioButtons = new JRadioButton[numButtons];
ButtonGroup group = new ButtonGroup();

radioButtons[0] = new JRadioButton("<html>Candidate 1:
      <font color=red>Sparky the Dog</font></html>");
radioButtons[0].setActionCommand(CANDIDATE1_STRING);

radioButtons[1] = new JRadioButton("<html>Candidate 2:
      <font color=green>Shady Sadie</font></html>");
radioButtons[1].setActionCommand(CANDIDATE2_STRING);
...
for (int i = 0; i < numButtons; i++)
    group.add(radioButtons[i]);

radioButtons[0].setSelected(true);
```

Dialogs

A top-level window with a title and a border. used to get some input from the user.

Must have a frame or another dialog as its "owner".

- When the owner is destroyed, so is the dialog.
- When the owner is minimized, so is the dialog.

Can be modal (disables all input to other top-level windows).

Can be used to create a custom dialog (many ready made dialogs are available in JOptionPane).

JOptionPane

Enables creation and customization of several kinds of modal dialogs.

Dialogs are created by invoking one of the static creation methods in `JOptionPane`

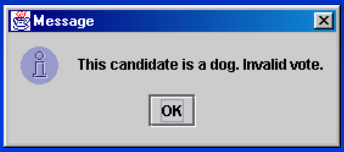
Customization options:

- Choosing an icon.
- Setting the title and text.
- Setting the button text.

showMessageDialog

Displays a modal dialog with one button labeled “ok”.

The title, text and icon are customizable.

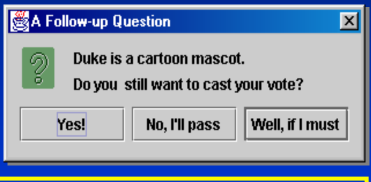


This candidate is a dog. Invalid vote.

```
JOptionPane.showMessageDialog
(frame,"This candidate is a dog. " + "Invalid vote.");
```

showOptionDialog

Displays a modal dialog with specified buttons, title, text and icon.


A Follow-up Question

Duke is a cartoon mascot.
Do you still want to cast your vote?

```
Object[] options = {"Yes!", "No, I'll pass", "Well, if I must"};
int n = JOptionPane.showOptionDialog(
             frame,                              // the owner frame
             "Duke is a cartoon mascot... \n“,   // message text
             "A Follow-up Question",             // title
             JOptionPane.YES_NO_CANCEL_OPTION,   // button format
             JOptionPane.QUESTION_MESSAGE,       // message type
             null,                               // custom icon
             options,                            // button names
             options[2]);                        // default selection
```

User input from a dialog

The **`showMessageDialog`** and **`showOptionDialog`** methods both return an integer indicating the user’s choice.

The possible returned values are:

- YES_OPTION
- NO_OPTION
- CANCEL_OPTION
- OK_OPTION
- CLOSED_OPTION (dialog closed without clicking a button)

The value is returned according to the clicked button and the button format of the dialog (DEFAULT, YES_NO, YES_NO_CANCEL, OK).

The buttons’ text doesn’t affect the returned value.

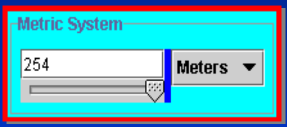
Swing components

The rest of this presentation contains a short description of most Swing components:

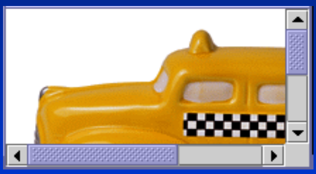
- General-purpose containers.
- Special-purpose containers.
- Basic controls.
- Uneditable information displays.
- Editable displays of formatted information.

General-purpose containers

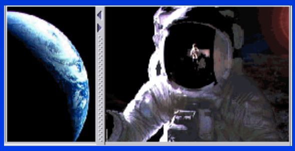
Panel



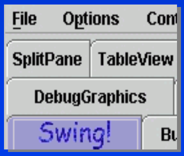
Scroll pane



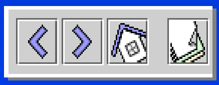
Split pane



Tabbed pane

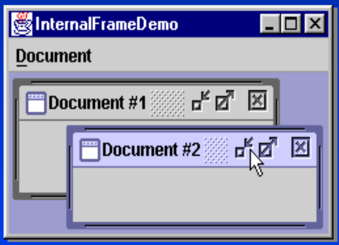


Tool bar

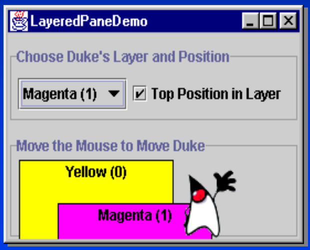


Special-purpose containers

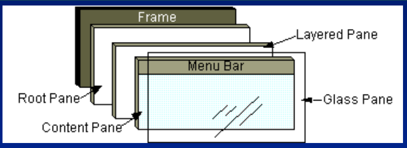
Internal frame



Layered pane



The Root pane



Root pane:
- Created automatically by every top-level (and internal) container.
- Contains a Layered pane.

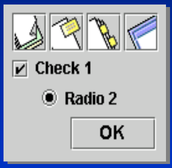
Layered pane:
- Holds components in a specified depth order.
- Initially contains the Content pane and the optional Menu bar.

Content pane:
- Contains all the Root pane's visible components (excluding the Menu bar).

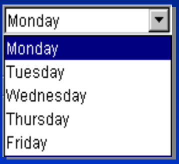
Glass pane:
- A hidden panel that intercepts input events for the Root pane.
- Can be made visible and drawn on by implementing its `paint()` method.

Basic controls

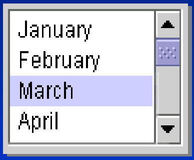
Buttons



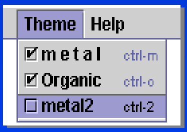
Combo box



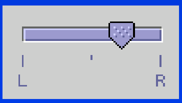
List



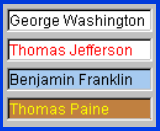
Menu



Slider



Text fields

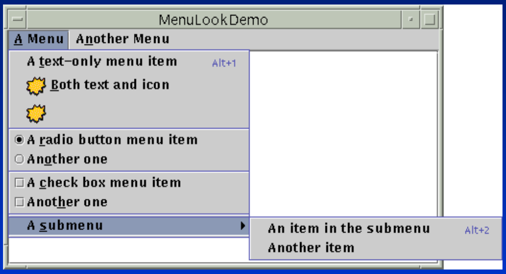


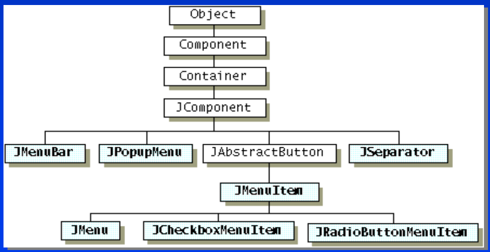
Buttons

The following list contains all button types (all are subclasses of `AbstractButton`):

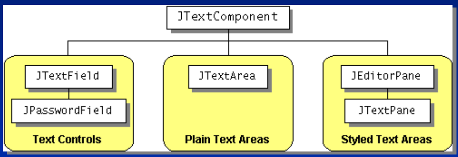
- `JButton`: a common button.
- `JCheckBox`: a check box button.
- `JRadioButton`: one of a group of radio buttons.
- `JMenuItem`: an item in a menu.
- `JCheckBoxMenuItem`: a menu item that has a check box.
- `JRadioButtonMenuItem`: a menu item that has a radio button.
- `JToggleButton`: a two-state button.

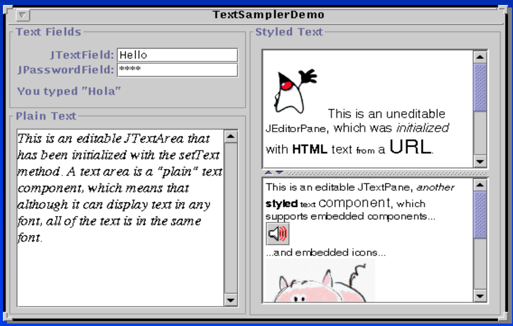
Menus





Text components





You typed "Hola"

This is an editable JTextArea that has been initialized with the setText method. A text area is a "plain" text component, which means that although it can display text in any font, all of the text is in the same font.

This is an uneditable JEditorPane, which was *initialized* with **HTML** text from a URL.

This is an editable JTextPane, *another* **styled** text component, which supports embedded components...

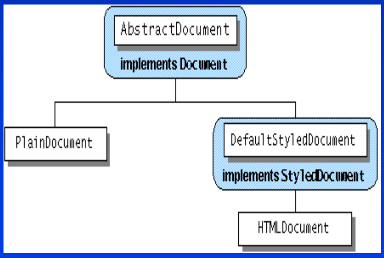
...and embedded icons...

Documents

All Swing components separate their data (or model) from the view of the data.

Text components use a `Document` as their model:

- Contains the text itself (including style info).
- Provides support for editing the text.
- Notifies document listeners on changes to the text.



Editor kits

Each text component holds an editor kit:

- Manages editing actions (cut, paste, etc) for the text component.
- Reads and writes documents of a particular format.

`DefaultEditorKit:`

- Reads and writes plain text.
- Provides a basic set of editing commands.
- The super class of all other editor kits.

`StyledEditorKit:`

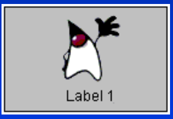
- Reads and writes styled text.
- Provides editing commands for styled text.

`HTMLEditorKit:`

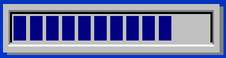
- Reads, writes and edits HTML.
- Subclass of `StyledEditorKit`.

Uneditable information displays

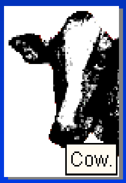
Label



Progress bar

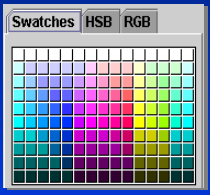


Tool tip

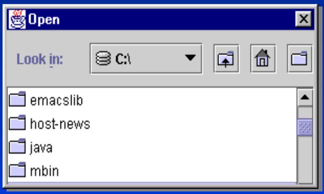


Editable displays of formatted information

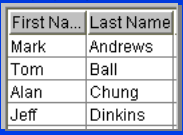
Color chooser



File chooser

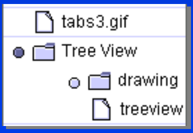


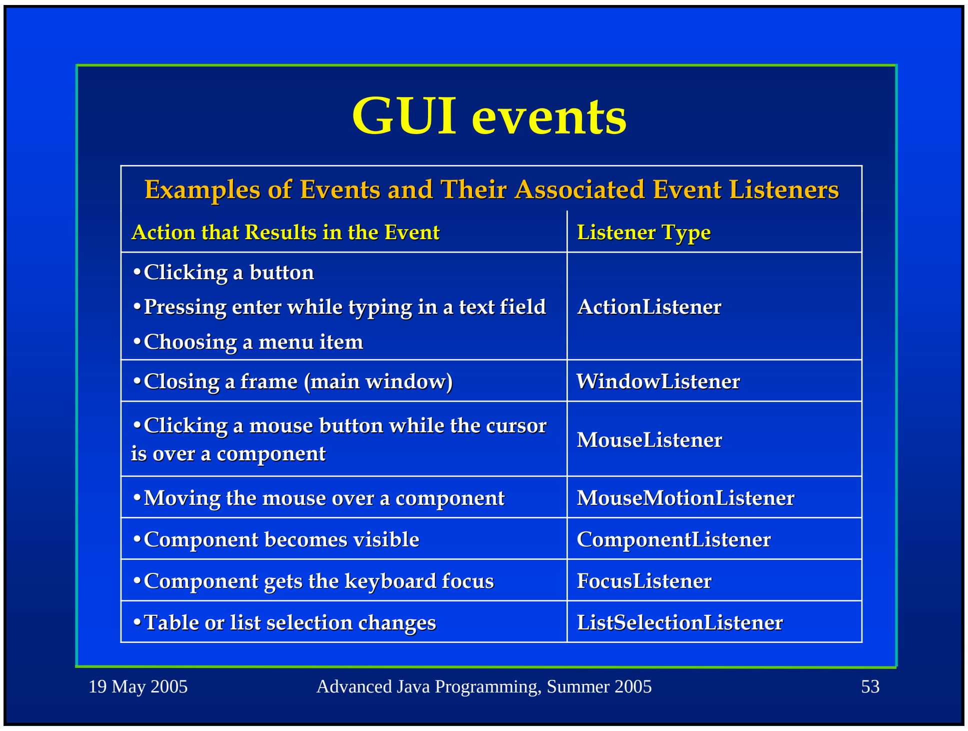
Table



First Na...	Last Name
Mark	Andrews
Tom	Ball
Alan	Chung
Jeff	Dinkins

Tree

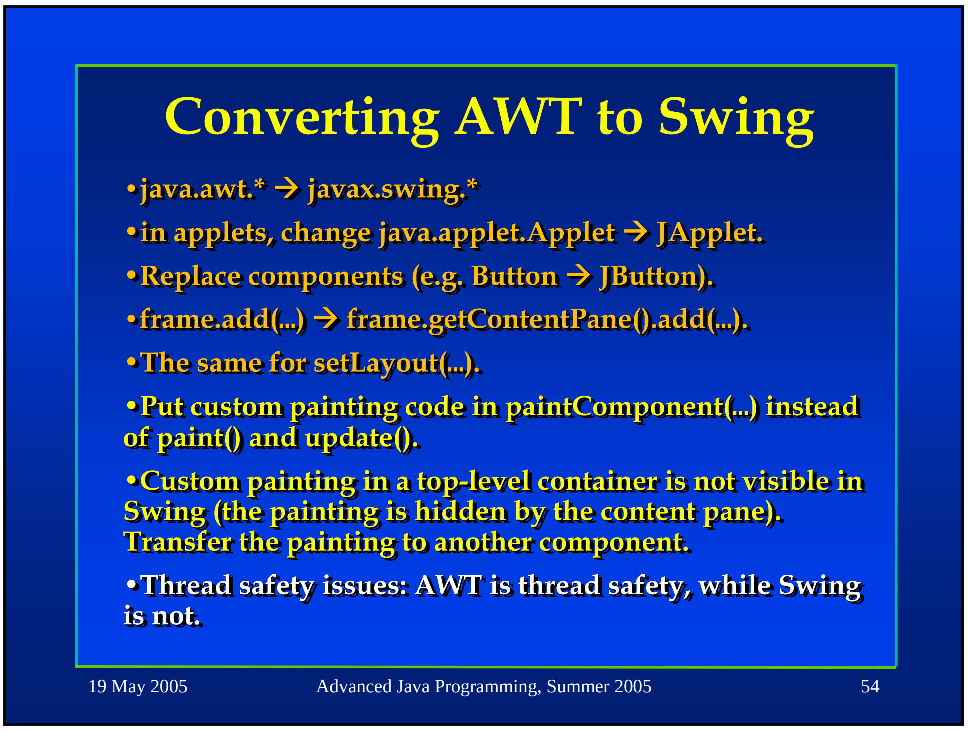




GUI events

Examples of Events and Their Associated Event Listeners

Action that Results in the Event	Listener Type
•Clicking a button •Pressing enter while typing in a text field •Choosing a menu item	ActionListener
•Closing a frame (main window)	WindowListener
•Clicking a mouse button while the cursor is over a component	MouseListener
•Moving the mouse over a component	MouseMotionListener
•Component becomes visible	ComponentListener
•Component gets the keyboard focus	FocusListener
•Table or list selection changes	ListSelectionListener



Converting AWT to Swing

- java.awt.* → javax.swing.*
- in applets, change java.applet.Applet → JApplet.
- Replace components (e.g. Button → JButton).
- frame.add(...) → frame.getContentPane().add(...).
- The same for setLayout(...).
- Put custom painting code in paintComponent(...) instead of paint() and update().
- Custom painting in a top-level container is not visible in Swing (the painting is hidden by the content pane). Transfer the painting to another component.
- Thread safety issues: AWT is thread safety, while Swing is not.

Converting AWT to Swing

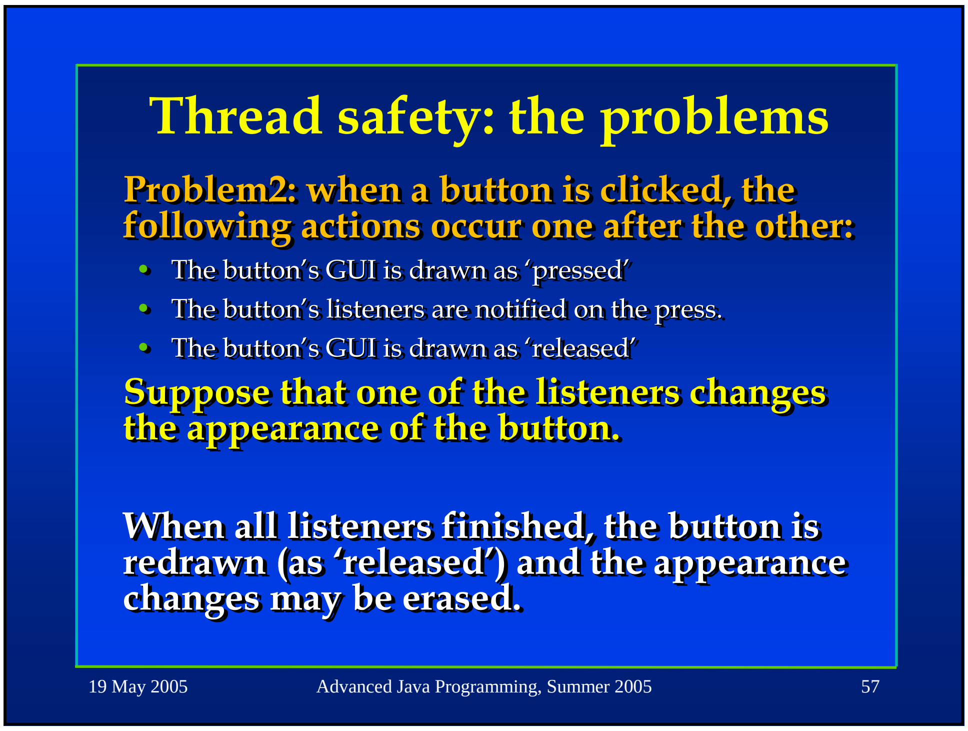
The containment hierarchy for any window or applet containing swing components must have a swing top level container at the root of the hierarchy. For example the main window should be a JFrame rather than a Frame.

Thread safety: the problems

Swing GUI components are updated in an event dispatching mechanism

In Swing, once a component is created, it can be updated only through the event dispatching mechanism.

Problem 1: What happens if we want to update the GUI from another thread?



Thread safety: the problems

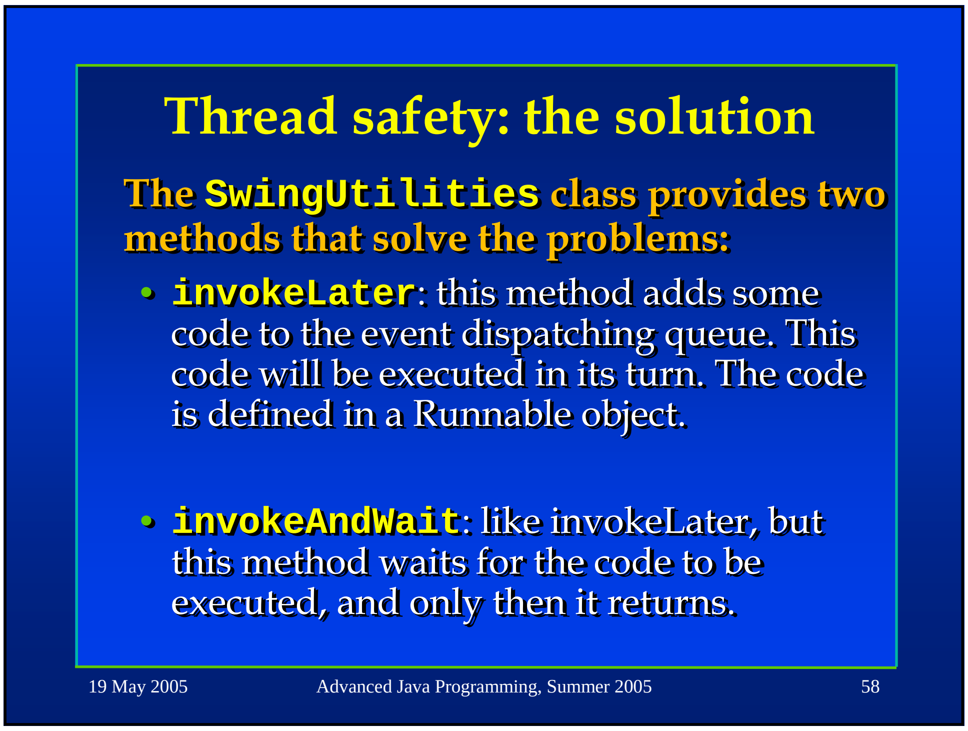
Problem2: when a button is clicked, the following actions occur one after the other:

- The button's GUI is drawn as 'pressed'
- The button's listeners are notified on the press.
- The button's GUI is drawn as 'released'

Suppose that one of the listeners changes the appearance of the button.

When all listeners finished, the button is redrawn (as 'released') and the appearance changes may be erased.

19 May 2005 Advanced Java Programming, Summer 2005 57



Thread safety: the solution

The `SwingUtilities` class provides two methods that solve the problems:

- **`invokeLater`**: this method adds some code to the event dispatching queue. This code will be executed in its turn. The code is defined in a Runnable object.
- **`invokeAndWait`**: like invokeLater, but this method waits for the code to be executed, and only then it returns.

19 May 2005 Advanced Java Programming, Summer 2005 58