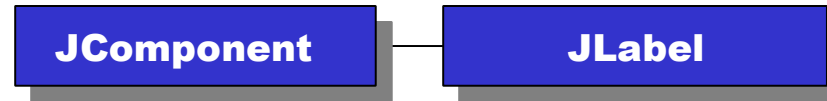


Labels

- Simplest form of user interface
- It is essentially an object that lets text strings be displayed. In AWT, that was about all it could do.
- JLabel is a JComponent, so it supports keyboard accelerators, borders, and so on.
- JLabels support internal ImageIcon objects, which allows graphics to be placed within a label.
- JLabels support fonts and colors, along with a wide range of text alignment
- JLabels now support html as text
 - If the text property of an AbstractButton subclass or a JLabel contains a string that begins with "<html>" the rest of the string be used to create a lightweight HTML document and the text will be rendered by the full Swing styled text engine.

JLabel constructors and methods



- **Constructors**

```
JLabel()
```

```
JLabel(String text)
```

```
JLabel(String text, int horizontalAlignment)
```

```
JLabel(String text, Icon icon, int horizontalAlignment)
```

```
JLabel(Icon icon)
```

```
JLabel(Icon icon, int horizontalAlignment)
```

– **horizontalAlignment**: Left, Right, Leading, Trailing, Center

- **Setting fonts**

```
label.setFont(new Font("Dialog", Font.PLAIN, 12))
```

- **Setting colors**

```
label.setBackground(Color.blue) ????
```

```
label.setForeground(Color.yellow)
```

Setting Fonts in JLabels

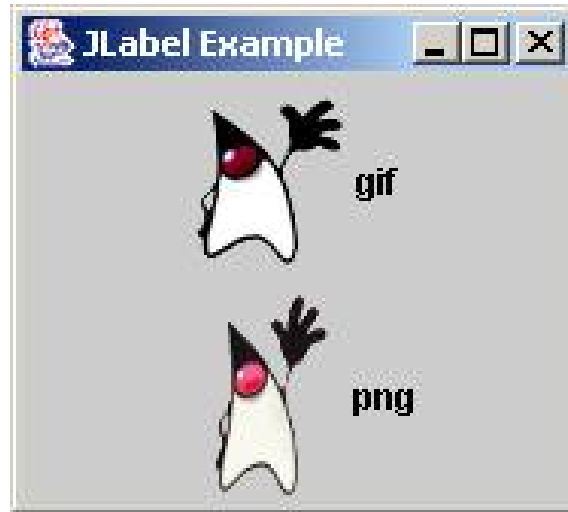
- When you set a font in a JComponent, its size may need to change
- You must call
 - `invalidate()` and `validate()` on the component to ensure it will resize properly
 - `pack()` will also work
- If you don't use the default Java fonts, you need to call `GraphicsEnvironment.getAllFonts()` to load them into the virtual machine
 - Dialog
 - DialogInput
 - Serif
 - SansSerif
 - ZapfDingbats
 - Monospaced
- Code Example, `Fonts.java`



Adding Images to Labels

- Need to add an Icon object
 - `Icon image = new ImageIcon("foo.gif");`
- The icon can be set after label creation by:
 - `label.setIcon(image);`
- Note that icons are not scaled to the label, so once the label is created it may be clipped if you add it afterwards
- Remove an icon with `label.setIcon(null)`
- JLabel, like many other components, supports alternate “disabled” display modes. If you have a different Icon you wish to present when the label is disabled, use
 - `label.setDisabledIcon(image)`
- Swing has a nice utility which can make “grayed-out” images from originals, if you want something other than the default behavior
 - `ImageIcon DImage = new ImageIcon(GrayFilter.createDisabledImage(image.getImage()));`

Code Example, Graphic.java



Implementing custom Icons

- You are not limited to “gif”, “png”, and “jpeg” images on JLabels (or JButtons)

- All you need is a class that implements the Icon interface

```
public interface Icon {  
    public void paintIcon (Component comp, Graphics g, int x, int y);  
    public int getIconWidth();  
    public int getIconHeight();  
}
```

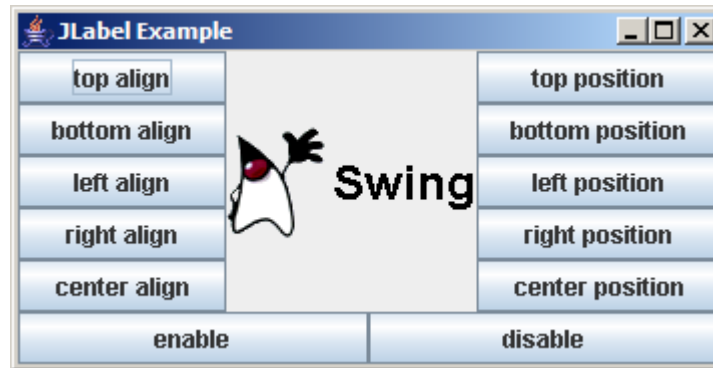
- When the component is painted, it calls the icon’s paintIcon method, passing it the component, a graphics context and the top-left corner of the area allocated to the icon.
 - The background will be cleared to the component’s background
 - Allows for non-rectangular icons

Code Example CustomIcon, CustomIcon2, CustomIconLabel, CustomIconLabel2

- These examples show a custom icon that is animated.
- Two means of causing repaints are shown
 - This first example, shows a reference to the enclosing object being passed to CustomIcon, which is then repainted as necessary
 - The second example shows the use of a property listener to drive the updates on the screen.



Code Example - MyLabel.java



Loading image files - example

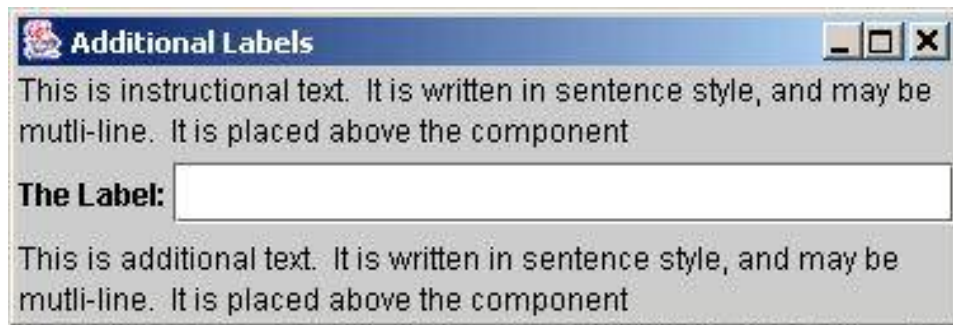
- Assume that your code resides in the `com.foo` package, and the `<filename>` argument of the `getClass().getResource(<filename>)` code is “`images/bar.gif`”
- Suppose that our class was loaded from local file storage using `c:\baz` as the CLASSPATH variable. Then the code actually resides in `c:\baz\com\foo` directory and your gif is actually `c:\baz\com\foo\images\bar.gif`.
 - The URL returned would be `file:///C:/baz/com/foo/images/bar.gif`
- Case 2: if the class file has been downloaded from a host with a CODEBASE of `http://www.myhost.com/java`
 - The URL returned would be `http://www.myhost.com/java/com/foo/images/bar.gif`
- Using the `getResource` method in this way allows the class file to be independent of the user’s current directory when the application is running, and it avoids any assumptions about where the class was loaded from.

Label placement

- Captions should be ended with a colon to clearly identify it as such.
- For single data fields, place the caption to the left of the data field or place the caption, justified upper left to the data field.
- Stay consistent within screen or related screens
- For multiple listings of columnar-oriented data, place the caption above the columnized data fields

Explanatory Labels

- If you need to add instructional text for a component, add it above the label.
 - Use complete sentences with ending punctuation
- Additional information on a component should be placed below the component
 - Use complete sentences with ending punctuation
 - Be brief
- If you need multiple line labels, use JTextArea



Label Placement continued

- **Justification**

- Left justify both captions (labels) and data fields
 - Leave one space between the longest caption and the data field column
 - Disadvantage of caption beginning point is usually farther from the entry field than the right-justified approach. This may greatly increase eye movements.
 - Advantage is the crisp left-justified captions
 - You can use the ellipsis (...) to tie together, but this increases noise of display

- **Justification**

- Left justify data fields and right justify labels
 - Leave one space between each
 - Disadvantage is that section headings using location positioning as the identification element do not stand out as well
 - Advantage is that labels are always positioned close to their related data fields.

- Other justifications are not recommended as they tend to increase screen complexity.

A form with a green background. The label 'Name:' is left-aligned, followed by a light blue input field. Below it, the label 'Address:' is left-aligned, followed by a shorter light blue input field.

Good

A form with a green background. The label 'Name:' is right-aligned, followed by a light blue input field. Below it, the label 'Address:' is right-aligned, followed by a shorter light blue input field.

Good

A form with a green background. The label 'Name:' is left-aligned, followed by a light blue input field. Below it, the label 'Address:' is left-aligned, followed by a shorter light blue input field.

Bad

JComponent Properties`

- Property list is supported on any JComponent
- JLabel has the following:
 - UI, UIClassID, accessibleContext, disabledIcon*, displayedMnemonic*, font, horizontalAlignment*, horizontalTextPosition, icon*, iconTextGap*, labelFor*, text, verticalAlignment*, verticalTextPosition*.
 - Items with a * are bound
- Bound items may have a PropertyChangeListener assigned so that you can react when that property is changed
- MyLabelProperty.java is a modified version of MyLabel.java with a PropertyChangeListener added.

