320341 Programming in Java

JACOBS
UNIVERSITY

Fall Semester 2014

Lecture 14: Introduction - Swing User Interface Components

Instructor: Jürgen Schönwälder

Slides: Bendick Mahleko

Objectives

JACOBS UNIVERSITY

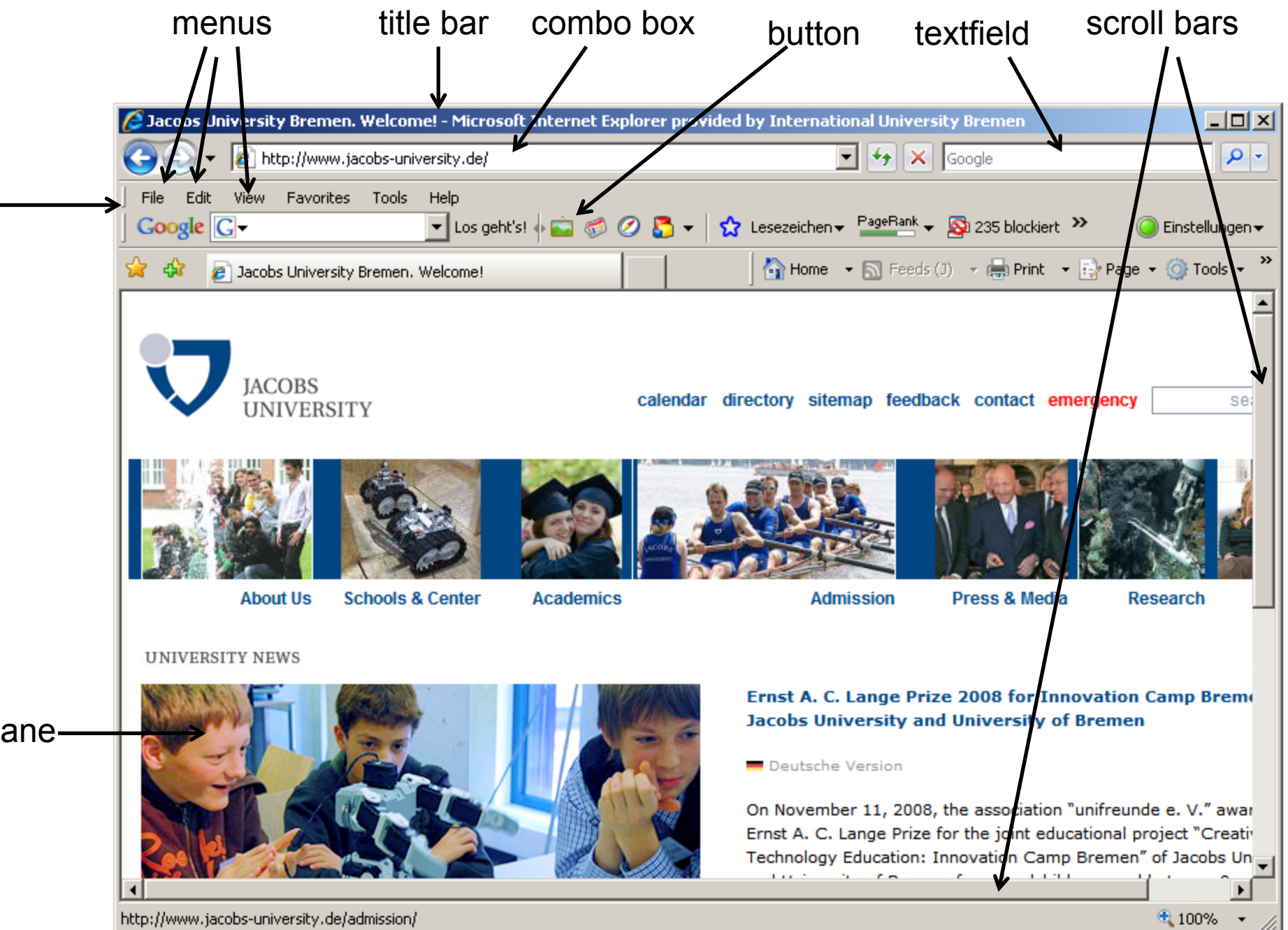
This lecture presents the following

- The **Model-View-Controller (MVC)** design pattern
- Basic **Layout Managers**
- **Text** input **Components**
- **Choice Components**

Introduction

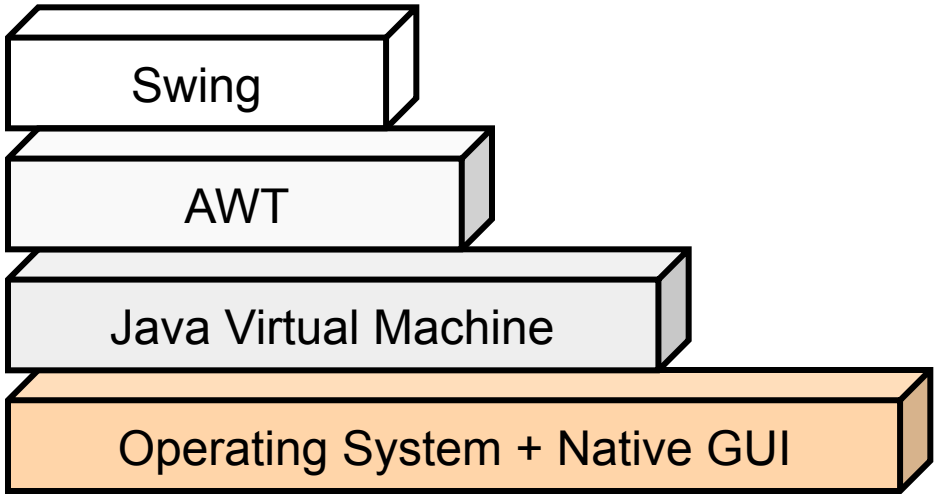
Abstract Window Toolkit (AWT)

- The original goal of AWT was to allow building *GUIs that look good on all platforms – this goal was not achieved as AWT had implementation problems*
- AWT drawing used “native peers” (Linux, Mac, Win32) – to be synchronized
- **Java Foundation Classes (JFC)** replaced old AWT in Java 2
- The Graphical User Interface (GUI) portion of JFC is called **Swing**
- GUIs are built from GUI components called widgets (window gadgets)
- A widget is an object with which a user interacts via mouse, keyboard, voice ..


menus
title bar
combo box
button
textfield
scroll bars
Jacobs University Bremen. Welcome! - Microsoft Internet Explorer provided by International University Bremen
http://www.jacobs-university.de/
Google
File
Edit
View
Favorites
Tools
Help
Los geht's!
Lesezeichen
PageRank
235 blockiert
Einstellungen
Jacobs University Bremen. Welcome!
Home
Feeds (J)
Print
Page
Tools
JACOBS
UNIVERSITY
calendar
directory
sitemap
feedback
contact
emergency
About Us
Schools & Center
Academics
Admission
Press & Media
Research
UNIVERSITY NEWS
Ernst A. C. Lange Prize 2008 for Innovation Camp Breme
Jacobs University and University of Bremen
Deutsche Version
On November 11, 2008, the association "unifreunde e. V." awar
Ernst A. C. Lange Prize for the joint educational project "Creati
Technology Education: Innovation Camp Bremen" of Jacobs Un
http://www.jacobs-university.de/admission/
100%
ane

Swing

- Implemented in Java (rt.jar file contains java code for Swing)
- Built on the **AWT** primitives, but done right
- Has **pluggable look-and-feel** (look-and-feel is the appearance & way in which the user interacts with the application)



- Some old AWT classes (e.g., event model) are still used in Swing

Model-View-Controller (MVC) Design Pattern

JACOBS UNIVERSITY

According to **MVC**, every component has three characteristics

- **Content** (e.g., state of a button, text in a text field)
- **Visual appearance** (e.g., color, size)
- **Behavior** (reaction to events)

MVC pattern distributes responsibilities to three specialized classes

- The **model** stores the content or data
- The **view** displays the content
- The **controller** handles user input

Model-View-Controller Design Pattern

JACOBS UNIVERSITY

Text Field Example

Model	**View**	**Controller**
The quick brown fox jumps over the lazy dog	**The quick brown**	Handles user input events (e.g., mouse clicks and keystrokes)

- The **model** must implement methods to change the contents and to discover what the content is (*add*, *remove*, *getText*)
- The model is completely non-visual
- The controller handles user-input events

Model-View-Controller Design Pattern

JACOBS UNIVERSITY

Wrapper classes (e.g., **JButton**, **JTextField**) store the ***model*** and ***view***

Queries about the **model** and **view** are handled by wrapper classes

It is possible to retrieve the model and work directly with it

The **Look-and-Feel** is responsible for the view

- Visual representation depends on the UI design of a particular look-and-feel
- Nimbus is an example Look and Feel used in Java (from Java SE 6)

MVC on Swing Buttons

JACOBS UNIVERSITY

Model

- The model class implements an interface with name ending in **Model**
- For the button, this is the **ButtonModel** interface

Method	Description
getActionCommand()	The action command string associated with this button
getMnemonic()	The keyboard mnemonic for this button
isEnabled()	true if the button is selectable
…	

Each button stores a button model, which you can retrieve

```
JButton button = new JButton(“Blue“);
ButtonModel model = button.getModel();
```

Basic GUI Components

JACOBS UNIVERSITY

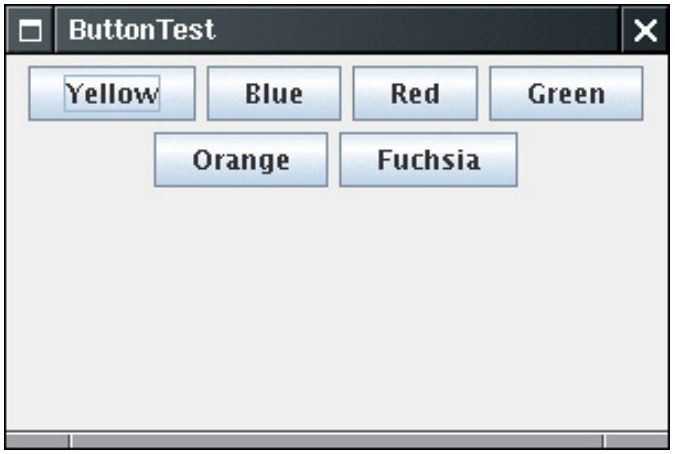
JLabel	Displays uneditable text or icons
JTextField	Enables user to enter data from the keyboard
JButton	Triggers an event when clicked with a mouse
JCheckBox	Specifies an option that can be selected or not selected
JComboBox	Provides a dropdown list of items from which the user can select
JList	Provides a *list of items* from which the user make a selection by clicking on any item in the list. Allows multiple element selection
JPanel	Provides an *area in which components can be placed and organized*. Can also be a drawing area for graphics

Introduction to Layout Managers

All components in a container are positioned by a **layout manager**

FlowLayout Manager

- The **FlowLayout** is the *default* layout manager for a panel
- Lines the components horizontally until there is no more room; and then starts a new row of components



The **FlowLayout** manager automatically reflows the components on resizing

Introduction to Layout Managers

JACOBS UNIVERSITY

FlowLayout Manager cont…

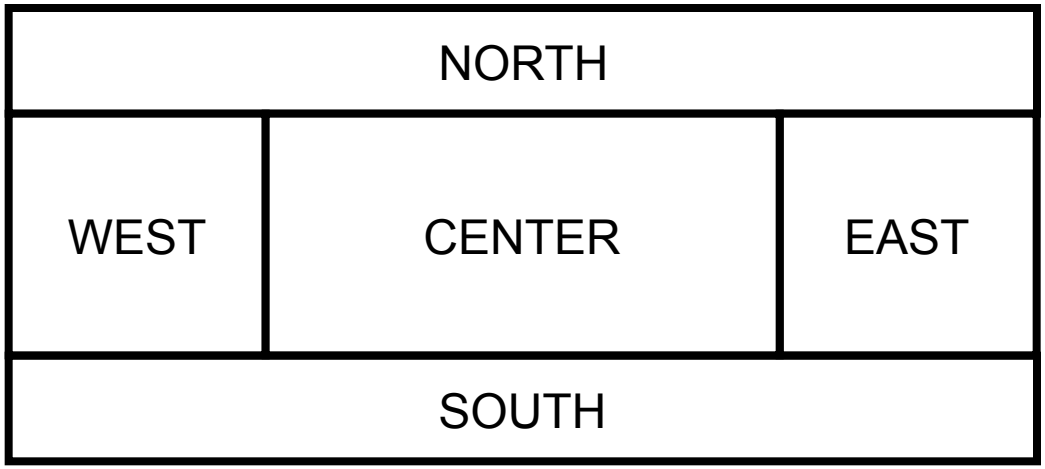
- Choose how to arrange components in a row
- Choose **LEFT**, **RIGHT** (default is **CENTER**)

Example

```
Panel.setLayout(new FlowLayout(FlowLayout.LEFT));
```

BorderLayout Manager

- The default layout manager of the content pane of every **JFrame**
- This manager lets you choose where you want to place components
- Components can be placed in the **CENTER**, **NORTH**, **SOUTH**, **EAST**, **WEST** of the content pane



JACOBS UNIVERSITY

BorderLayout Manager Example

```
panel.setLayout(new BorderLayout());
panel.add(yellowButton, BorderLayout.SOUTH);
```

BorderLayout Manager Features

- Edge components are laid out first; remaining space is occupied by center
- On resizing, edge components don't change, but center components do
- Border layout grows all components to fill available space

Introduction to Layout Managers

JACOBS UNIVERSITY

Panels

- A border layout is not very useful on its own
- Components grow to fill entire region
- Example

- Button fills entire southern region
- Adding another displaces the first one

Nest panels inside containers and use different layout managers

- Ex. Create a panel in the southern region to hold buttons
- Ex. Create another panel in the center to hold text

Introduction to Layout Managers

JACOBS UNIVERSITY

Example

- A panel with three buttons

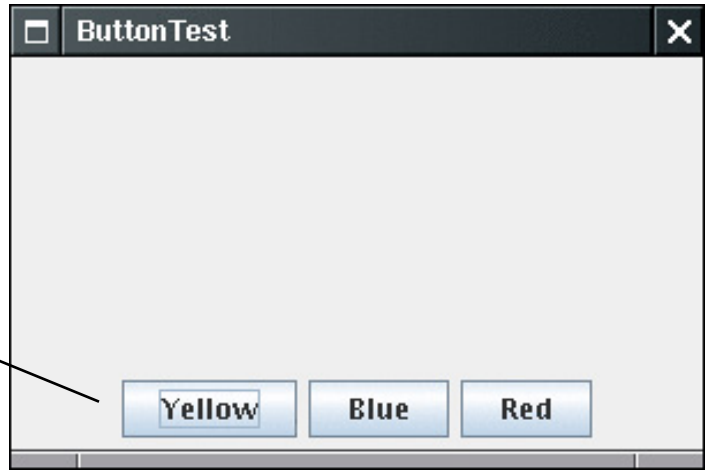
Create **JPanel** object

Buttons under **FlowLayout**

```
JPanel panel = new JPanel();
panel.add(yellowButton);
panel.add(blueButton);
panel.add(redButton);
frame.add(panel, BorderLayout.SOUTH);
```

- Panel nested inside frame

- Buttons centered in panel and do not fill entire panel area



Introduction to Layout Managers

JACOBS UNIVERSITY

GridLayout

- Arranges all components in rows and columns like a spreadsheet
- Cells are always the same size
- Buttons grow and shrink upon window resize
- All buttons have the same size

GridLayout Constructor

Sets panel to 5 rows, 4 cols

```
panel.setLayout(new GridLayout(5, 4));
```

- Components are added row-wise

Introduction to Layout Managers

GridLayout Example (Calculator)

Matrix of equally sized cells

Buttons must be attached to event listeners

Basic Swings UI Components

JACOBS UNIVERSITY

Component Classes

- Classes are found in the `javax.swing` package
- Component class names all begin with "**J**"
- The remainder of the class name is the name of the component
- Examples: **JButton**, **JTextField**, **JTextArea**, **JScrollBar** classes

- All Swing components are subclasses of `java.awt.Container`
- Most Swing components emit *events* from the `java.awt.event` package

Basic Swings Components

Container Components

- These are components that can contain other components (including other containers)

- **Containers** use layout managers to determine size and position of their child components

Examples of **Container** Components

- **JFrame**
- **JPanel**

Basic Swings UI Components

JACOBS UNIVERSITY

Container Components - **JFrame**

- A **JFrame** is an independent window that can be moved around on the screen independently of any other GUI windows

- Any application that requires a GUI must use one or more frames to contain the desired components

Example

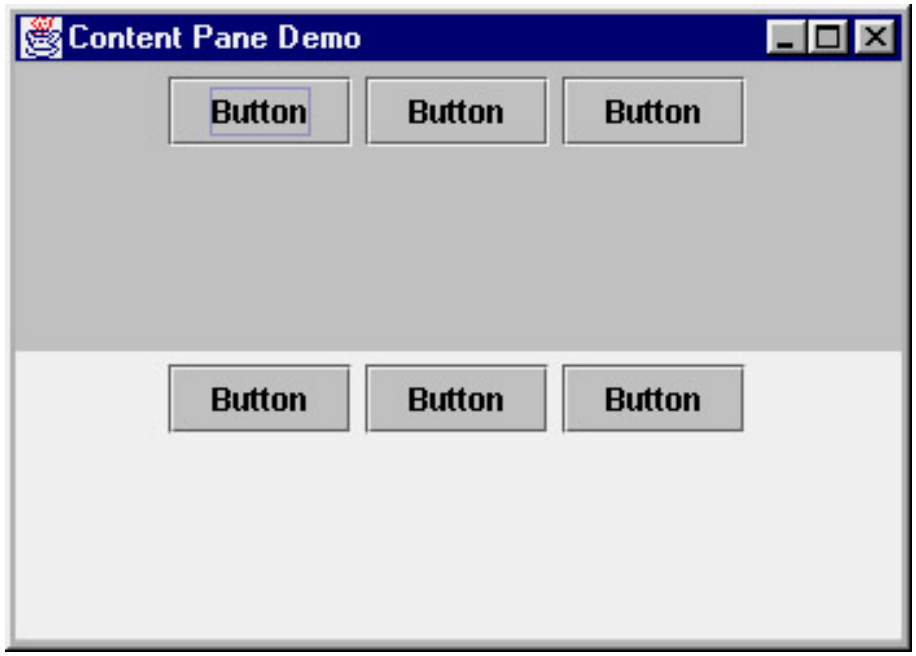
```
import javax.swing.JFrame;

 public class FrameDemo {

  public static void main(String[] args) {
        JFrame frame = new JFrame("Frame Demo");
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setSize(350, 250);
        frame.setVisible(true);
  }
 }
```

Container Components – JPanel

- A **JPanel** is a blank rectangular component that can contain other components
- Example : Two panels inside a frame, containing three buttons each



JTextField and JTextArea

JACOBS UNIVERSITY

Text Input

- Lets the user input and edit text
- Two components: **JTextField** and **JTextArea** components
- Both classes inherit from **JTextComponent** abstract class
- **JTextComponent** text manipulation methods:

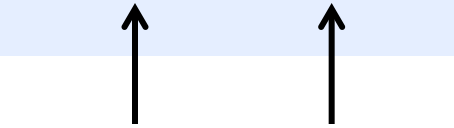
```
void setText (String t)
String getText ()
void setEditable(boolean b)
```

Example

Text Input – **Text Fields**

- To add a text field to a window, add the text field to a panel (or other container)

```
JPanel panel = new JPanel();
JTextField textField = new JTextField("Default input", 20);
panel.add(textField);
```



Text Fields usually start off blank

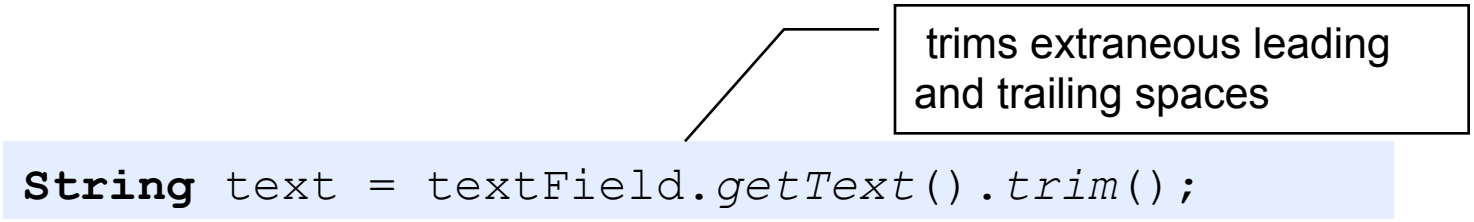
- Use constructor without text portion

```
JPanel panel = new JPanel();
JTextField textField = new JTextField(20);
panel.add(textField);
```

Example

Reading Text Fields

- Use *getText*() method to read what the user typed



Labels

JACOBS UNIVERSITY

Labels and Labeling Components

- Labels are components that hold text and used to identify components
- They do not react to user input

Constructing a **JLabel**

```
JLabel label = new JLabel("Minutes", JLabel.RIGHT);
```

- Labels can be positioned inside a container like any other component

Password Fields

Password Fields

- A special kind of text field
- Typed characters are represented by an echo character (*)

- Swing supplies a **JPasswordField** to implement the password text field

JPasswordField (**String** text, int columns)
char[] *getPassword()*
void *setEchoChar(char echo)*

Example:

JFormattedTextField

JACOBS UNIVERSITY

Formatted Input Fields

- Use **JFormattedTextField** class to validate user input

- Example – Integer input validation

```
JFormattedTextField intField = new
        JFormattedTextField(NumberFormat.getIntegerInstance());
```

JFormattedTextField

- The **NumberFormat**.*getIntegerInstance*() returns a formatter object that formats integers, using the current *locale*

- Reading the values from integer formatted text field

```
Number value = (Number) intField.getValue();
int v = value.intValue();
```

FormattedTextFieldVerifier

JACOBS UNIVERSITY

Verifiers

- Verifiers are useful for alerting users to invalid input
- A verifier can be attached to any **JComponent**
- The verifier extends the abstract **InputVerifier** class & define *verify* method

```
class FormattedTextFieldVerifier extends InputVerifier {
  public boolean verify(JComponent component)    {
       JFormattedTextField field = (JFormattedTextField) component;
    return field.isEditValid();
  }
}
```

Attach verifier to **JFormattedTextField** as follows

```
intField.setInputVerifier(new FormattedTextFieldVerifier());
```

Standard Formatters

- getNumberInstance
- getCurrencyInstance
- getPercentInstance

Editing Dates (call of one of the static methods of the **DateFormat** class)

- getDateInstance
- getTimeInstance
- getDateTimeInstance

DefaultFormatter

DefaultFormatter

- Can format any class that has a constructor with a string parameter and a matching `toString` method like for example

```
DefaultFormatter formatter = new DefaultFormatter();
formatter.setOverwriteMode(false);
JFormattedTextField urlField = new JFormattedTextField(formatter);
urlField.setValue(new URL("http://java.sun.com"));
```

MaskFormatter

- Used to format fixed-size patterns

#	Digit	A	Letter or digit
?	Letter	H	Hexadecimal digit[0-9A-Fa-f]
U	Upper case letter	*	Any character
L	Lowercase letter	‘	Escape character

JTextArea

- The **JTextArea** class allows a user to enter any number of lines
- Specify the number of rows and columns when calling the constructor

```
JTextArea textArea = new JTextArea(8, 40); // 8 lines of 40 columns each
```

- Avoid clipping long lines by turning on line wrapping

```
textArea.setLineWrap(true); // long lines are wrapped
```

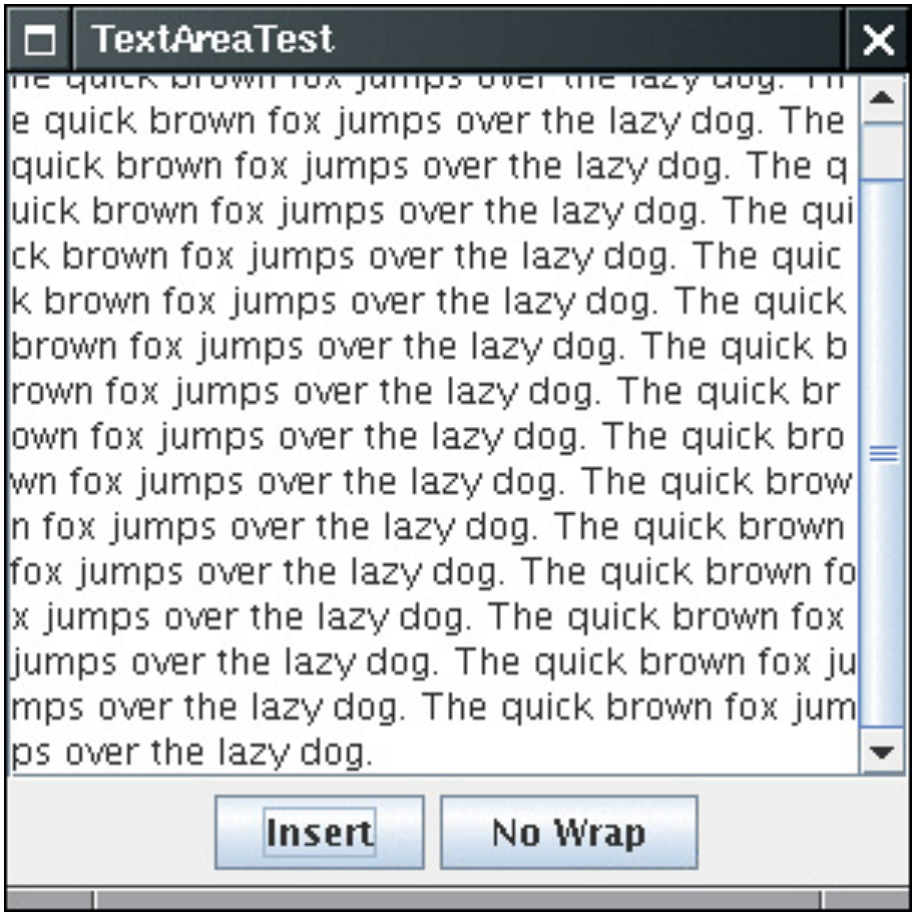
- Insert text area inside a scroll pane

```
textArea = new JTextArea(8, 40);
JScrollPane scrollPane = new JScrollPane(textArea);
```

Basic Swings UI Components

JACOBS UNIVERSITY

Text Areas - Example



CheckBoxes

JACOBS UNIVERSITY

Choice Components

- Allow to give users a finite set of choices

Checkboxes

- Checkbox component used to collect “yes” or “no” input
- Constructor

```
JCheckBox  bold = new JCheckBox("Bold");
```

- Use *setSelected* method to turn a checkbox on or off

```
bold.setSelected(true);
```

CheckBoxes

JACOBS UNIVERSITY

Choice Components - Checkboxes

- When a checkbox is clicked, an **ActionEvent** is triggered
- Attach an action listener to the checkbox

```
ActionListener listener = . . .
bold.addActionListener(listener);
italic.addActionListener(listener);
```

- The *actionPerformed* method of the **ActionListener** interface queries the state of the source events

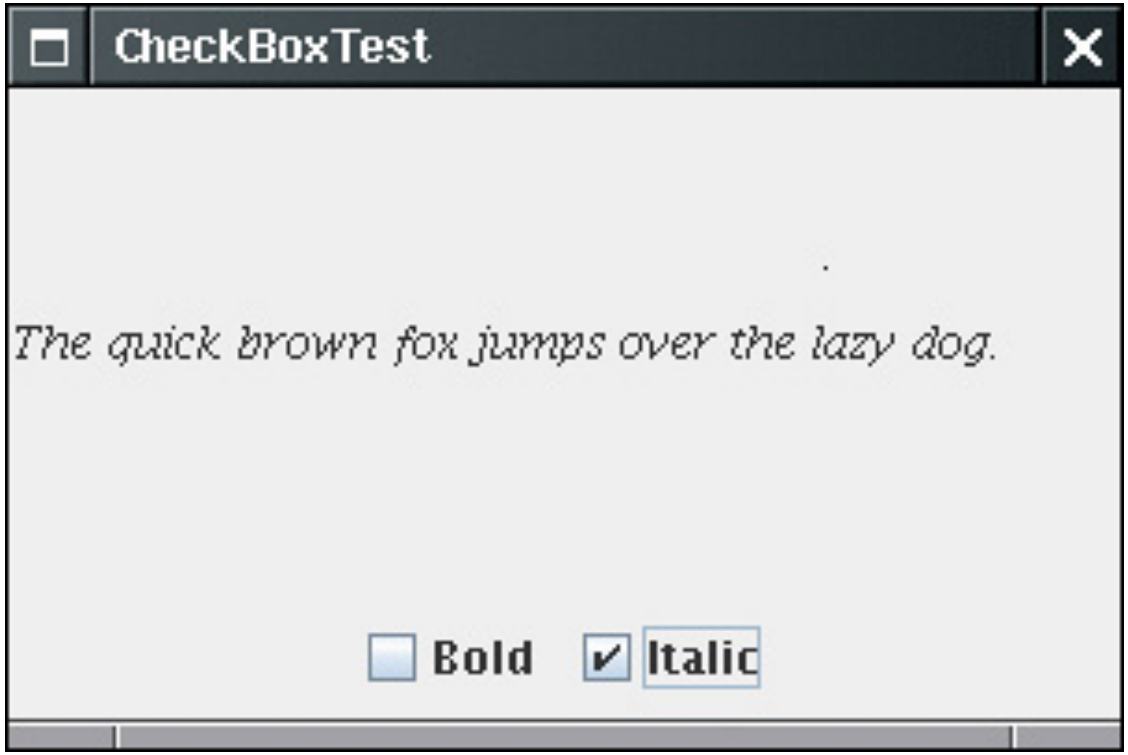
```
public void actionPerformed(ActionEvent event) {
    int mode = 0;
    if (bold.isSelected()) mode += Font.BOLD;
    if (italic.isSelected()) mode += Font.ITALIC;
    label.setFont(new Font("Serif", mode, FONTSIZE));
}
```

CheckBoxes

JACOBS UNIVERSITY

Choice Components - Checkboxes

- Example



Radio Buttons

Choice Components – Radio Buttons

- Used when user must select only one of several options
- Construct one object of type **ButtonGroup** for every group of buttons

```
ButtonGroup group = new ButtonGroup();
JRadioButton smallButton = new JRadioButton("Small", false);
group.add(smallButton);
JRadioButton mediumButton = new JRadioButton("Medium", true);
group.add(mediumButton);
. . .
```

Event Notification

```
ActionListener listener = new   ActionListener()   {
      public void actionPerformed(ActionEvent event)  {
            // size refers to the final parameter of the addRadioButton method
            label.setFont(new Font("Serif", Font.PLAIN, size));
      }
};
```

Basic Swings UI Components

JACOBS UNIVERSITY

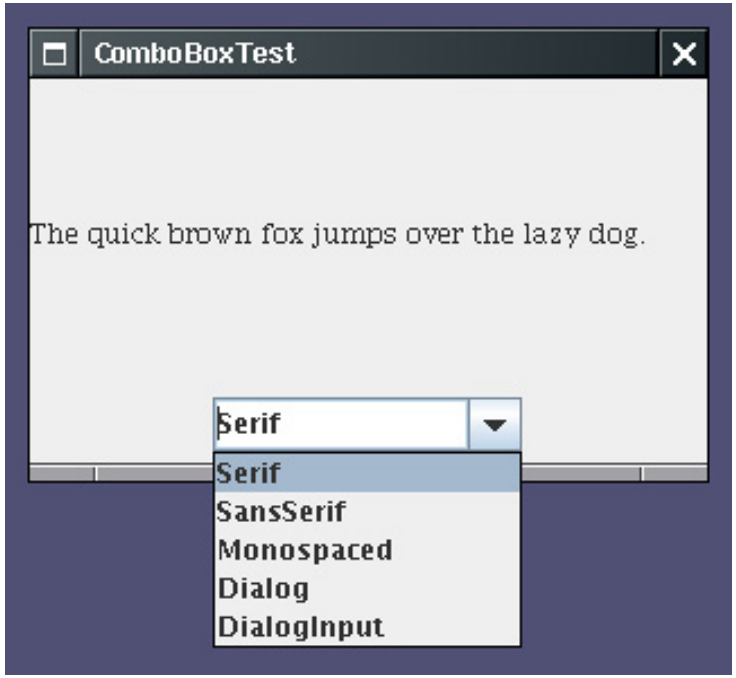
Choice Components – Radio Buttons

- Example

Combo Boxes

Choice Components – Combo Boxes

- Used to minimize the amount of space required for choice components
- When user clicks the combo box, a list of choices drops down



- The user can select one option

Combo Boxes

JACOBS UNIVERSITY

Choice Components – Combo Boxes (A generic class as of Java SE 7

- Creating a Combo Box

```
JComboBox<String>  faceCombo = new JComboBox<String>();
faceCombo.setEditable(true);
faceCombo.addItem("Serif");
faceCombo.addItem("SansSerif");. . .
```

Reading Current Selection

- Use the *getSelectedItem* method

To remove items from Combo Box

- Use *removeItem()*, *removeItemsAt()*, *removeAllItems()*

Combo Boxes

JACOBS UNIVERSITY

You can also insert items at specific positions

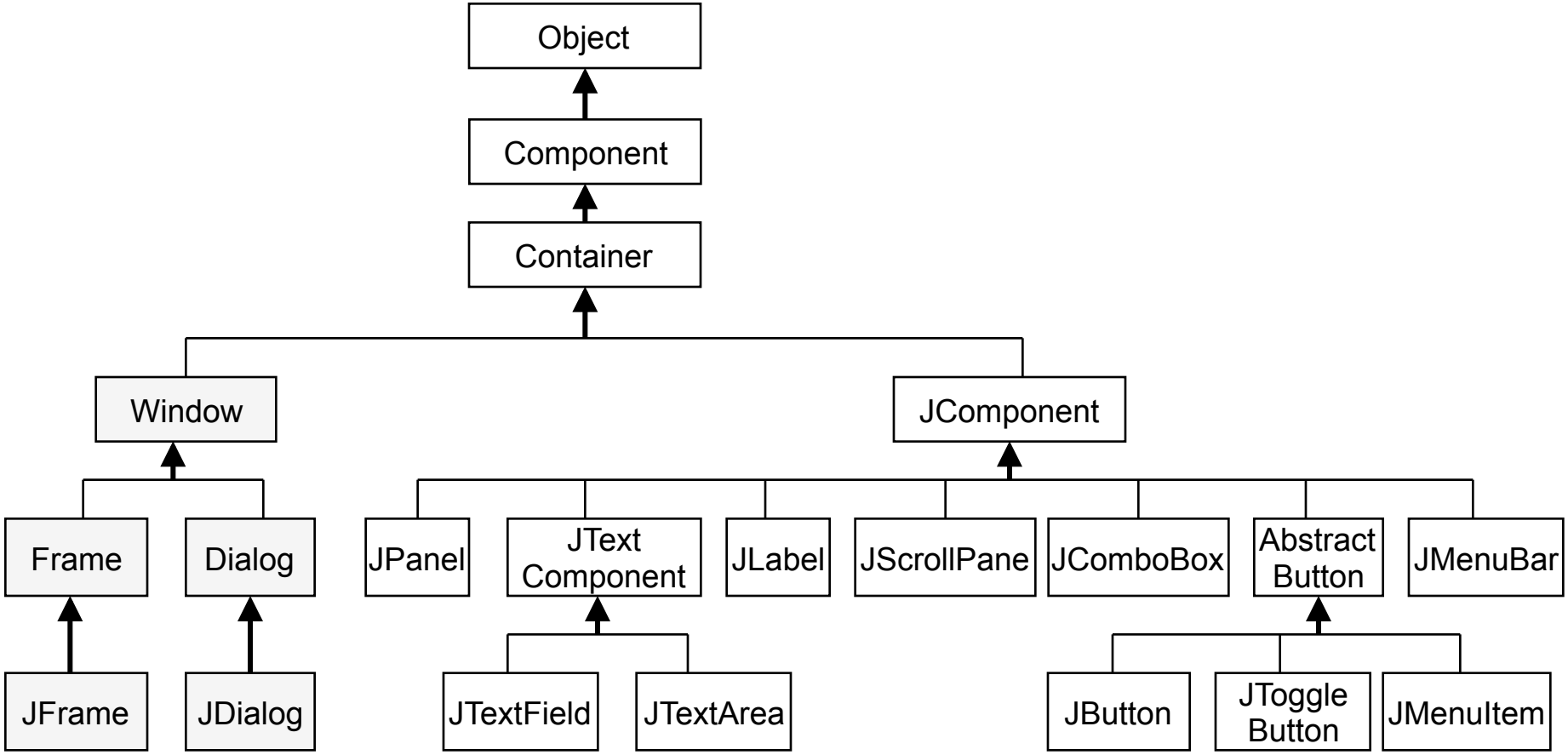
- Example

```
faceCombo.insertItemAt("Monospaced", 0); // add at the beginning
```

Layout Managers Overview

Basic Principles

- Inheritance hierarchy for the **Component** class



Reading Assignment

Horstmann & Cornell (2013) Core Java 2 Volume I, Chapter 9. 9th ed. User Interface Components with Swing by

Deitel, P., Deitel, H., Mukherjee, S. & Bhattacharjee, A. K. (2011) Java™: How to Program, Chapter 14. 9th Edition. Pearson Higher Education.