CS133: Developing Programming Principles

Lecture 21

Buttons, Text Fields, and Containers

Overview

Last Day: graphics, drawing, mouse clicks, and the **`Board`** class

Today: buttons, action listeners, container classes and text fields

Button Example

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;

public class PushMe
   extends JFrame
   implements ActionListener
{
  // on next two slides
}
```

Button Example

```
public PushMe()
{
  JButton pushMe = new JButton("Push Me");
  pushMe.addActionListener(this);

  Container content = getContentPane();
  content.add(pushMe);

  setDefaultCloseOperation(EXIT_ON_CLOSE);
  pack();
  setVisible(true);
}
```

Button Example

```
public void actionPerformed(ActionEvent e)
{
  System.out.println("Stop that!");
}


public static void main(String[] args)
{
  PushMe test = new PushMe();
}
```


Class **JButton**

The **JButton** class is a Swing component that provides a simple “push button”:

public JButton(String label)

When the button is pressed an event is delivered to its ***listener***.

PushMe implements ActionListener

In this example, the **PushMe** class handles events for the button by specifying itself as the button's **actionListener:**

```
pushMe.addActionListener(this)
```


Implementing ActionListener

Classes implementing the **ActionListener** interface must support the method:

```
public void actionPerformed(ActionEvent e)
{
   …
}
```

Two Buttons

```
JButton pushMe = new JButton("Push Me");
pushMe.addActionListener(this);

JButton pushMe2 = new JButton("Push Me Too");
pushMe2.addActionListener(this);

JPanel buttonPanel = new JPanel();
buttonPanel.setLayout(new FlowLayout());
buttonPanel.add(pushMe);
buttonPanel.add(pushMe2);
```

Two Buttons

```
public void actionPerformed(ActionEvent e)
{
  if (e.getActionCommand().equals("Push Me"))
  {  System.out.println("Stop that!");
  }
  else if (
    e.getActionCommand().equals("Push Me Too"))
  {
     System.out.println("Ouch!");
  }
}
```

Layout Managers

A ***layout manager*** is a special kind of object that is responsible for organizing the contents of a **`JFrame`**, **`JPanel`**, or other container class. Three useful layout managers are:

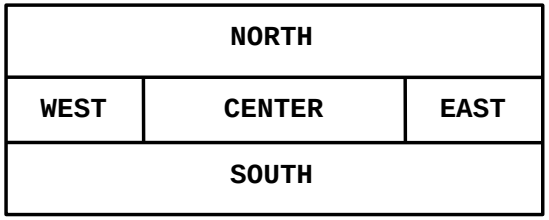
- **`FlowLayout`**
- **`BorderLayout`**
- **`GridLayout`**

`FlowLayout`

- A **`FlowLayout`** arranges components in a line from left to right in the order they were added, until the line is full.
- Lines are centered.
- Each component is allowed to assume its preferred size.

BorderLayout

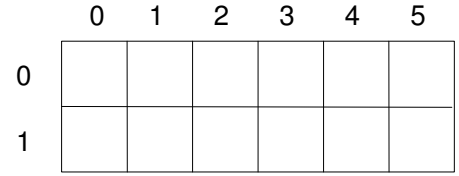
A **BorderLayout** organizes components into one of five regions:



GridLayout

A **GridLayout** organizes components into a rectangular NxM grid:

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



`ActionEvent`

An **`ActionEvent`** is generated by a component to describe an event, such as a button when it is pressed.

`public String getActionCommand()`

Method to return the command **`String`** associated with this action. For a button, this is usually the label.

Adding a Text Field

```
public class PushMe
             extends JFrame
             implements ActionListener
{
  private JTextField text;

  // etc.
}
```

Adding a Text Field

```
JPanel buttonPanel = new JPanel();
buttonPanel.setLayout(new FlowLayout());
buttonPanel.add(pushMe);
buttonPanel.add(pushMe2);

text = new JTextField(40);
text.setEditable(false);

Container content = getContentPane();
content.setLayout(new BorderLayout());
content.add(text, BorderLayout.NORTH);
content.add(buttonPanel, BorderLayout.SOUTH);
```


Adding a Text Field

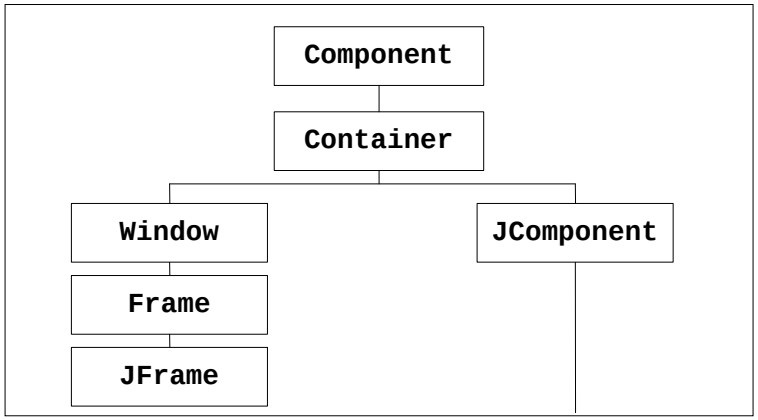
```
public void actionPerformed(ActionEvent e)
{
 if (e.getActionCommand().equals("Push Me")){
     text.setText("Stop that!");
 } else if (
  e.getActionCommand().equals("Push Me Too")){
     text.setText("Ouch!");
 }
}
```

JTextField

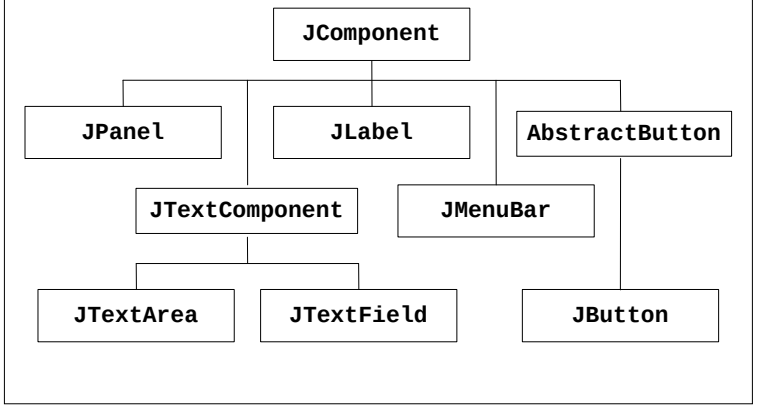
A **JTextField** component provides for the display and editing of a single line of text.

```
public JTextField(int columns)
public String getText()
public void setText(String text)
public void setEditable(boolean b)
```


Component Class Hierarchy



JComponent
Class Hierarchy



JLabel

A **JLabel** component provides for the display (but not editing) of a single line of text.

```
public JLabel(String text)
public String getText()
public void setText(String text)
```

JTextArea

A **JTextArea** component provides for the display and editing of multiple lines of text.

```
public JTextArea(int rows,int columns)
public String getText()
public void setText(String text)
public void setEditable(boolean b)
public void setLineWrap(boolean wrap)
```


Summary

- Buttons
- Action listeners
- Layout managers
- Text Fields
- The Swing class hierarchy

ERROR: undefined
OFFENDING COMMAND:

STACK: