CSE203 – OBJECT ORIENTED PROGRAMMING

Introduction to GUI Programming in Java

Elements of GUI Programming

❑ Components
- Visual objects that appear on the screen

❑ Layouts
- Control over the positioning of components within a *container*

❑ Events
- Responses to user actions

❑ Graphics
- Lines, shapes, colors, fonts, etc.

❑ All are supported in Java library packages.

Components/widgets

Two categories of Java Component classes:

❑ AWT – Abstract Windows Toolkit (java.awt package)

- The older version of the components
- Rely on "peer architecture"…drawing done by the OS platform on which the application/applet is running
- Platform-dependent look.
- Considered to be "heavy-weight"

❑ Swing (javax.swing package)

- Newer version of the components
- No "peer architecture"…components draw themselves
- Most are considered to be "lightweight"

Swing VS. AWT

❑ *Never* mix Swing and AWT components

❑ If you know AWT, put 'J' in front of everything

▪ AWT: **Button**

▪ Swing: **JButton**

❑ Swing does all that AWT does, but better.

Using a GUI Component

1. Create it
2. Configure it
3. Add children (if container)
4. Add to parent (if not JFrame)
5. Listen to it

order
important

Containers

❑ Descendents of the java.awt.Container class

❑ Components that can contain other components.

❑ Use a layout manager to position and size the components contained in them.

❑ Components are added to a container using one of the various forms of its **add** method

  ▪ Depending on which layout manager is used by the container

Container Hierarchy

❑Top-level container:
- place for other Swing components to paint themselves
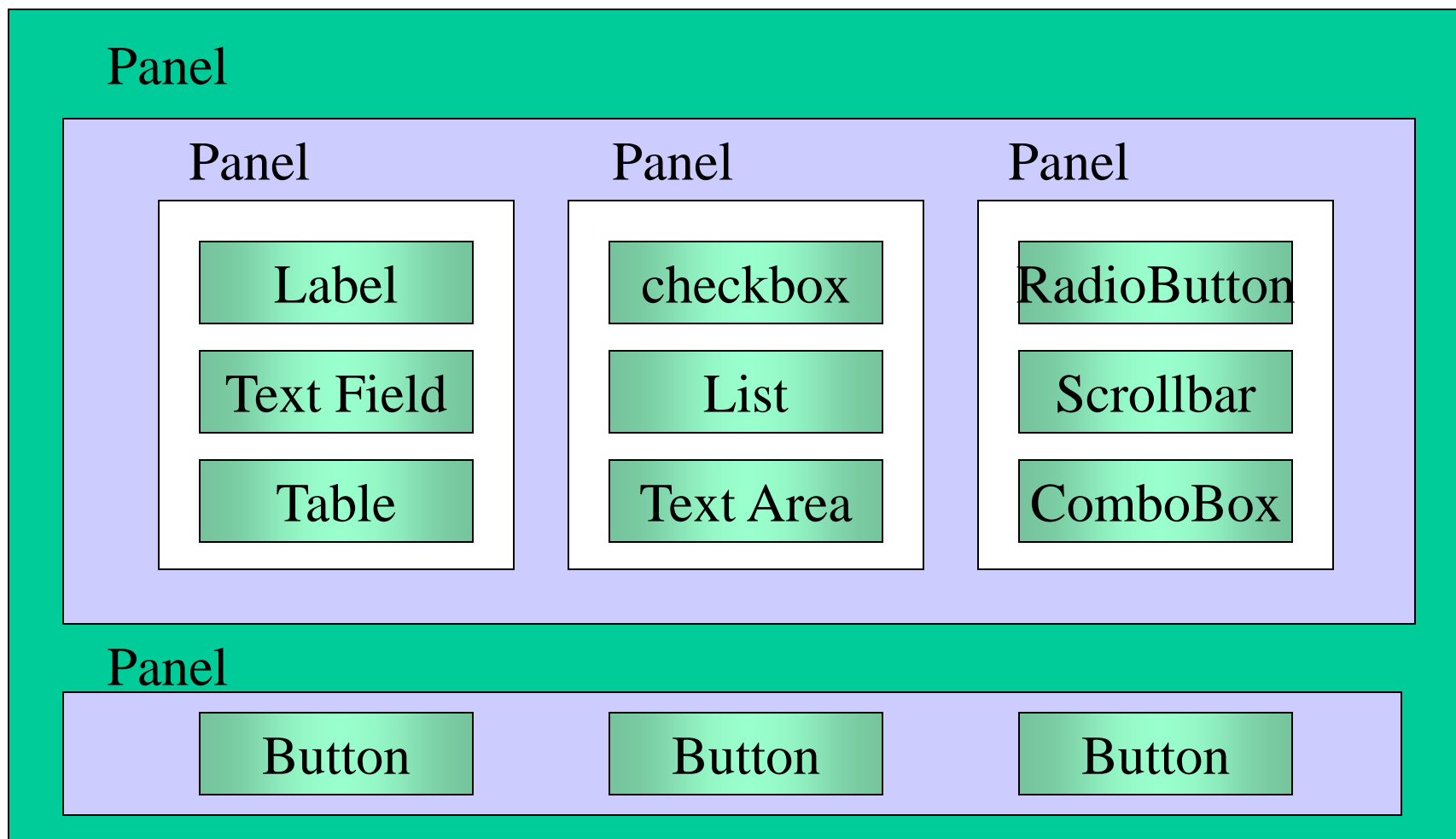- e.g., JFrame, JDialog, Japplet

❑Intermediate container:

- simplify positioning of atomic components
- e.g., JPanel, JSplitPane, JTabbedPane

❑Atomic components:
- self-sufficient components that present information to and get input from the user
- e.g., JButton, JLabel, JComboBox, JTextField, JTable

Hierarchy Sample

Frame

Panel

Panel

Label

Text Field

Table

Panel

checkbox

List

Text Area

Panel

RadioButton

Scrollbar

ComboBox

Panel

Button

Button

Button

Top Level Containers

❑Every program that presents a Swing GUI contains **at least one top-level container.**

❑A Top level container provides the support that Swing components need to perform their painting and event-handling.

❑Swing provides three top-level containers:

- ▪ JFrame (Main window)
- ▪ JDialog (Secondary window)
- ▪ JApplet (An applet display area within a browser window)

JFrame

❑ A frame implemented as an instance of the JFrame class, is a window that has decorations such as a border, a title and buttons for closing and iconifying the window.

▪ The decorations on a frame are platform dependent.

❑ Applications with a GUI typically use at least one frame.

Some Jframe methods

JFrame() - Constructs a new frame that is initially invisible.

JFrame(String title)

setSize(width, height);

pack();

setLocation(x, y);

setVisible(true);

setResizable(false);

getContentPane();

Content Pane
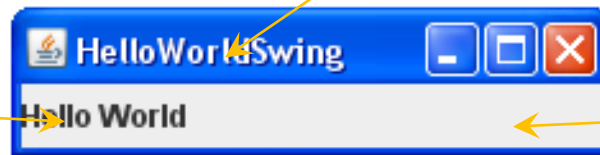
❑ When a frame is created, the content pane is created with it

❑ To add a component to the content pane (and thus to the frame), use:

- ▪ frameName.getContentPane().add(component name);
- ▪ where frameName is the name of the frame

Example 1

```java
import javax.swing.*;
public class HelloWorldSwing {
    public static void main(String[] args) {
        JFrame frame = new JFrame("HelloWorldSwing");
        JLabel label = new JLabel("Hello World");
        frame.getContentPane().add(label);
        frame.setVisible(true);
    }
}
```

Title

JFrame

JLabel

HelloWorldSwing

Hello World

This gray area is the content pane of this frame.

Example 2

```java
import javax.swing.*;
public class HelloWorldFrame extends JFrame {
    public HelloWorldFrame() {
        super("HelloWorldSwing");
        JLabel label = new JLabel("Hello World");
        getContentPane().add(label);
        setVisible(true);
    }
    public static void main(String[] args) {
        HelloWorldFrame frame = new HelloWorldFrame();
    }
}
```
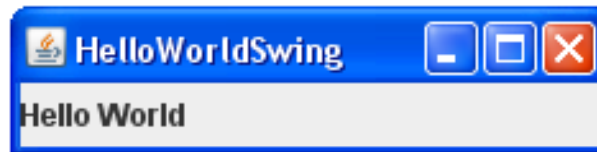
JComponents
(also called "widgets")

❑ JComponent is the base class for all Swing components except top-level containers.

▪ JLabel, JButton, JList, JPanel, JTable, ...

❑ To use a component that inherits from JComponent, it must be placed in a top-level container.

JPanels/Panes

- ❑ The terms "pane" and "panel" are used interchangeably in Java
- ❑ Intermediate Level Containers.
- ❑ Panes hold a window's GUI components
- ❑ Every frame has *at least* one pane, the default "Content Pane"
- ❑ Useful for layout
  - ▪ If you want to group certain GUI components together, put them inside a pane, then add that pane to the frame
- ❑ Needed to add components to the frame
  - ▪ Nothing can be added directly to the frame; instead, everything, including other panes, is added to the frame's content pane

JPanel methods

- **JPanel**() - Creates a new JPanel with a double buffer and a flow layout.
- **JPanel**(boolean isDoubleBuffered) - Creates a new JPanel with FlowLayout and the specified buffering strategy.
- **JPanel**(LayoutManager layout) - Create a new buffered JPanel with the specified layout manager.
- **JPanel**(LayoutManager layout, boolean isDoubleBuffered) - Creates a new JPanel with the specified layout manager and buffering strategy.

JLabel

- ❑ A display area for a short text string or an image, or both.
- ❑ A label does not react to input events.
- ❑ Methods:
  - **JLabel**() - Creates a JLabel instance with no image and with an empty string for the title.
  - **JLabel**(String text) - Creates a JLabel instance with the specified text.
  - **JLabel**(String text, alignment)
   - Alignemnt is eigher JLabel.LEFT, JLabel.CENTER, JLabel.RIGHT

JLabel

JButton

❑Java class that allows you to define a button

❑Multiple constructors allow you to initialize a button with a predefined label and/or a predefined icon

❑Although the button's "action" can be defined in the constructor, defining a button's action can take many lines of code and should be done separately

❑JButton methods

    ❑**JButton**() - Creates a button.

    ❑**JButton**(String text) - Creates a button with text.
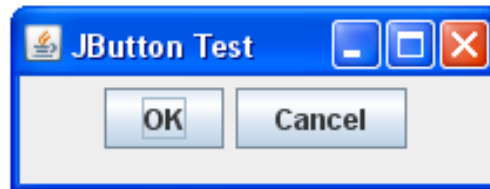
JTextField

❑ A Java text field is essentially the same as a text area, only limited to one line

❑ Very similar set of methods

❑ JPasswordField is the same as JTextField, only the contents are hidden

❑ Different constructors allow you to predefine the number of columns and/or the default text

JLabel   Years: 30   JTextField
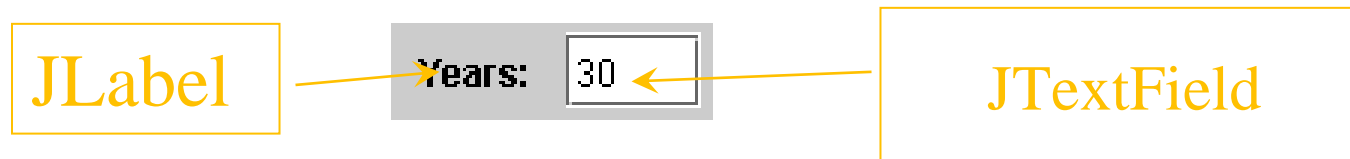
JTextField methods

❑ **JTextField**() - Constructs a new TextField.

❑ **JTextField**(String text) - Constructs a new TextField initialized with the specified text.

❑ **JTextField**(String text, int columns) - Constructs a new TextField initialized with the specified text and columns.

❑ **getText**() - Return the text of this field as a string object

❑ void **setHorizontalAlignment**(int alignment)
- JTextField.LEFT
- JTextField.CENTER
- JTextField.RIGHT
- JTextField.LEADING
- JTextField.TRAILING

JTextAreas

❑A text area is just a white space of variable size that can hold text

❑If text goes out of the area's bounds, it will exist but some of it will not be seen

▪ Wrap the text area in a scrollable pane
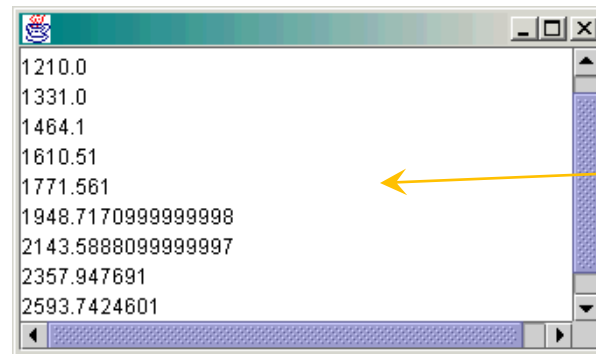
JTextArea with Scroll Bars

JTextarea Methods

- ❑ JTextArea() - Constructs a new TextArea.

- ❑ JTextArea(int rows, int columns) - Constructs a new empty TextArea with the specified number of rows and columns.

- ❑ **JTextArea**(String text) - Constructs a new TextArea with the specified text displayed.

- ❑ **JTextArea**(String text, int rows, int columns) - Constructs a new TextArea with the specified text and number of rows and columns.

- ❑ textarea.setText(String);

- ❑ textarea.getText(String);

- ❑ textarea.append(String);

- ❑ textarea.setEditable(boolean);

JScrollPane

❑JScrollPane basically consists of JScrollBars, a JViewport, a column header and a row header.

❑Methods
- ▪ JTextArea textArea = new JTextArea("Type here",5, 20);
- ▪ JScrollPane scrollPane = new JScrollPane(textArea);
- ▪ frame.setContentPane(scrollPane);
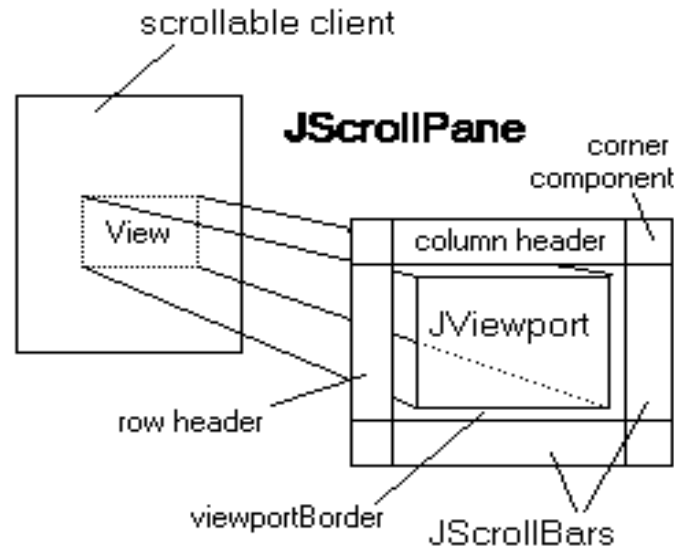
JList

❑ A simple GUI object design to hold lists of objects and allow users to make selections from the list

❑ Can be created from a ListModel, a Vector, or an array (all essentially lists themselves)

❑ Jlist methods

  ❑ **JList**() - Constructs a JList with an empty model.

  ❑ **JList**(Object[] listData) - Constructs a JList that displays the elements in the specified array.
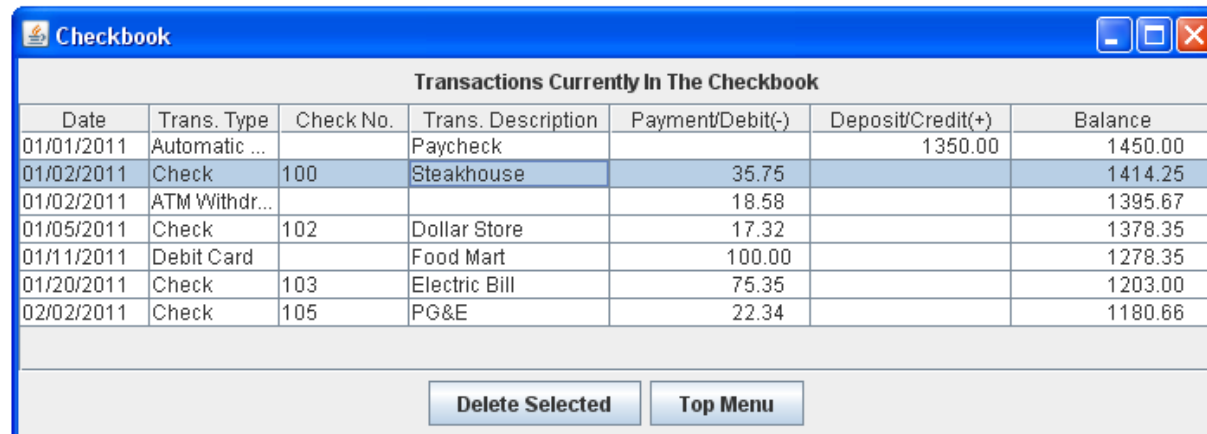
JComboBox

- Drop-down box
- Can be:
  - User-editable:
    - allows user to type in a value in the field similar to a JTextField
    - list.setEditable(true);
  - User-un-editable:
    - user must select an entry from the drop-down list
    - default
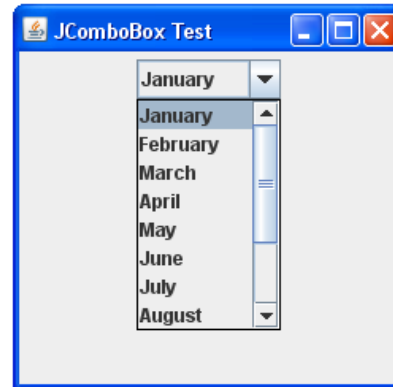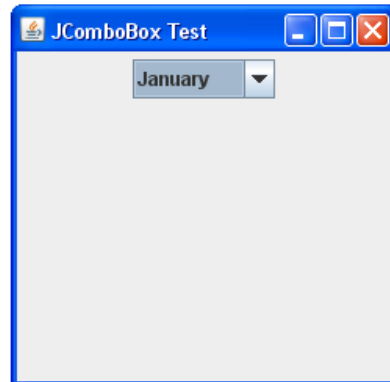
JTable

❑ Usually created from a DefaultTableModel

  ▪ Can also be created from an array of arrays or a Vector of Vectors, or can have no initial data

❑ Create a DefaultTableModel, then initialize a table from the DefaultTableModel

❑ Some methods

  ▪ Jtable()

  ▪ JTable(int numRows, int numColumns)

Checkbook

Transactions Currently In The Checkbook

Date	Trans. Type	Check No.	Trans. Description	Payment/Debit(-)	Deposit/Credit(+)	Balance
01/01/2011	Automatic ...		Paycheck		1350.00	1450.00
01/02/2011	Check	100	Steakhouse	35.75		1414.25
01/02/2011	ATM Withdr...			18.58		1395.67
01/05/2011	Check	102	Dollar Store	17.32		1378.35
01/11/2011	Debit Card		Food Mart	100.00		1278.35
01/20/2011	Check	103	Electric Bill	75.35		1203.00
02/02/2011	Check	105	PG&E	22.34		1180.66

← JTable

Delete Selected   Top Menu

RadioButton

- Come in groups – only 1 selected per group
- Make radiobuttons
- Make group
- Add radiobuttons to group
- ActionListener

JCheckBox

- A *check box* can be toggled checked or unchecked

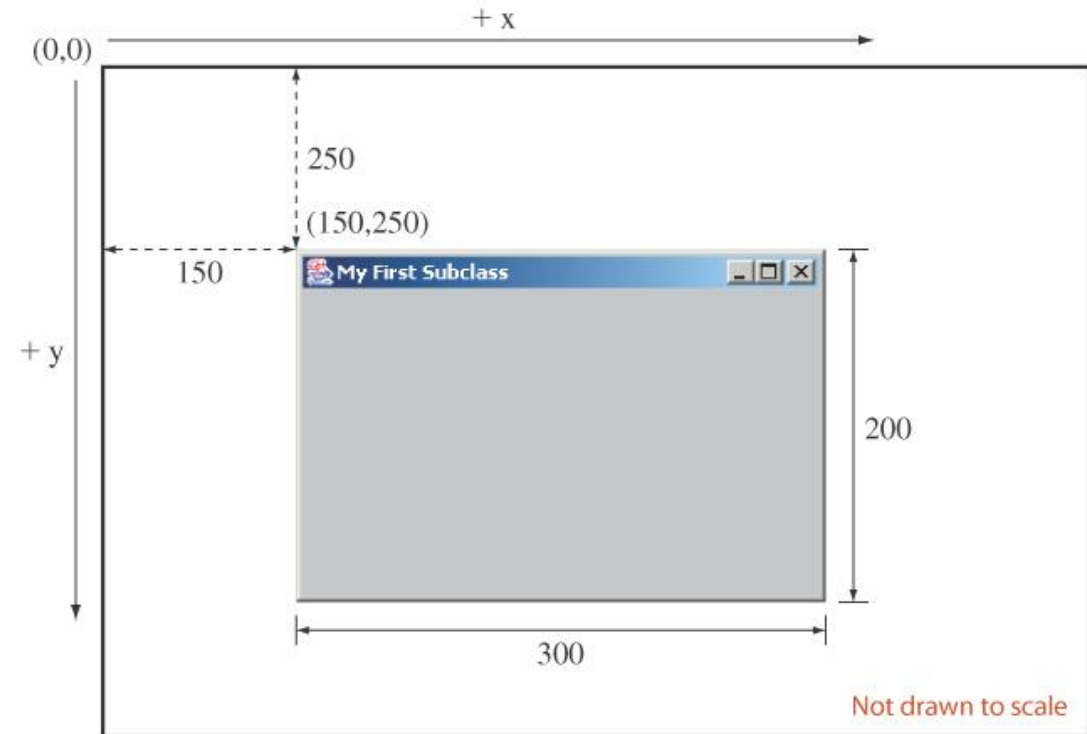- Can have one or more checkboxes selected (as opposed to radio buttons)

How to Layout Widgets?

❑ Two approaches
  ▪ Manually specify absolute positions
  ▪ Use layout managers

Manually specify absolute positions

Use component methods such as:
- public void setLocation(Point p)
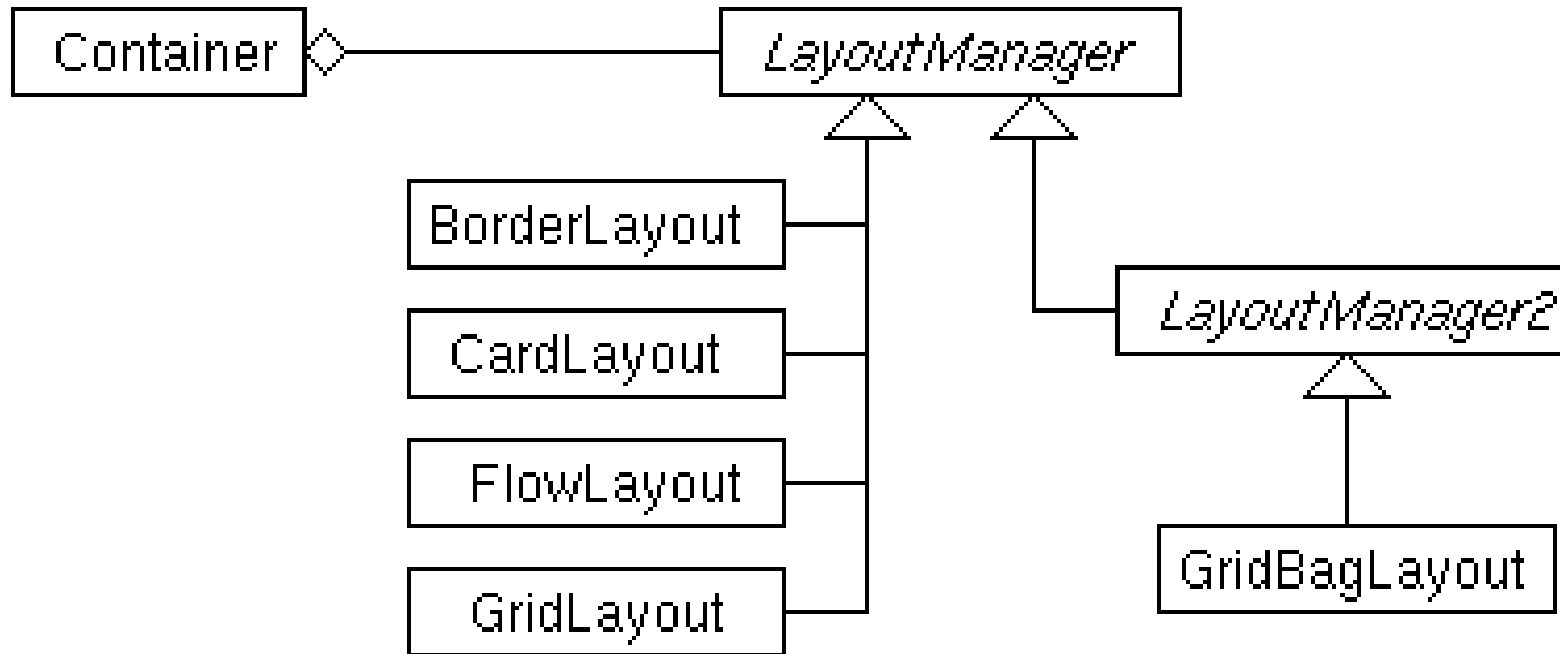- public void setSize(int width, int height)

Layout Managers

❑Layout managers are AWT software components which have the ability to lay out widgets/components by their relative positions.

❑Associated with containers.

❑Automate the layout of elements

- When elements are added to the container
- When the window is resized
  - automatically adjust the positions and sizes of the elements.

Hierarchy of Layout Managers

```
┌───────────┐      ┌──────────────┐
│ Container │◇─────│ LayoutManager │
└───────────┘      └──────────────┘
                      △        △
┌──────────────┐       │        │
│ BorderLayout │───────┤        │
└──────────────┘       │   ┌──────────────┐
┌──────────────┐       │   │ LayoutManager2 │
│  CardLayout  │───────┤   └──────────────┘
└──────────────┘       │        △
┌──────────────┐       │        │
│  FlowLayout  │───────┤        │
└──────────────┘       │   ┌──────────────┐
┌──────────────┐       │   │ GridBagLayout │
│  GridLayout  │───────┘   └──────────────┘
└──────────────┘
```

Using Layout Managers

| Method	Description
setLayout(lm)	Set lm as the layout manager
add(comp)	Add a component
add(comp, cst)	Add a component with constraint

setLayout(new FlowLayout());
add(new JButton("Increment"));
add(new JButton("Decrement"));

Layout Management

❑ When using the *add* method to put a component in a container, the container's layout manager must be taken into account.

- Relative position (BorderLayout)

  panel.add(component, BorderLayout.CENTER);

- Order of addition (BoxLayout, GridLayout, ...)

  panel.add(component);

FlowLayout

❑Places components from left to right, starting new rows if necessary.

❑Default LayoutManager of JPanel

Left to right,
Top to bottom
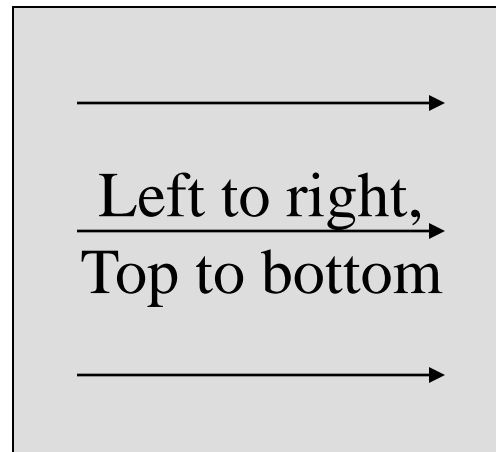
FlowLayout example

```java
import javax.swing.*;
import java.awt.*;
public class flow {
    public static void main(String[] args) {
        JFrame frm = new JFrame("Flow Layout Test");
        Container contentPane = frm.getContentPane();
        contentPane.setLayout(new FlowLayout());
        for (int i=0; i<=9; ++i)
                contentPane.add(new JButton(""+i));
        frm.pack();
        frm.setSize(318,220);
        //frm.setResizable(false);
        frm.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frm.setVisible(true);
    }
}
```

BorderLayout

❑ Has five areas available to hold components

▪ north, south, east, west and center

❑ All extra space is placed in the center area

▪ Only the center area is affected when the container is resized.

❑ Default layout manager of content panes.
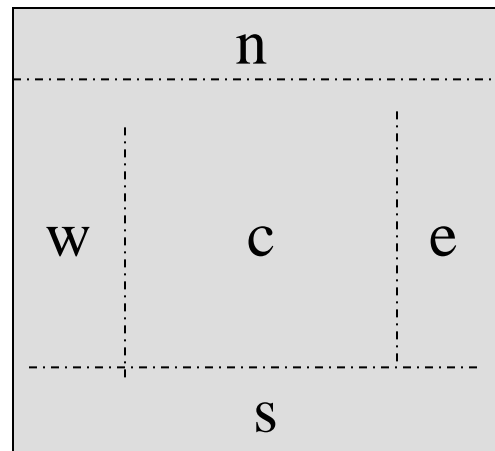
BorderLayout example

```java
import javax.swing.*;
import java.awt.*;
public class border {
    public static void main(String[] args) {
        JFrame frm = new JFrame("Border Latout Test");
        Container contentPane = frm.getContentPane();
        contentPane.setLayout(new BorderLayout());
        contentPane.add(new JButton("  0  "), BorderLayout.NORTH);
        contentPane.add(new JButton("  1  "), BorderLayout.SOUTH);
        contentPane.add(new JButton("  2  "), BorderLayout.EAST);
        contentPane.add(new JButton("  3  "), BorderLayout.WEST);
        contentPane.add(new JButton("  4  "), BorderLayout.CENTER);
        frm.pack();
        frm.setSize(318,220);
        frm.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frm.setVisible(true);
    }
}
```

BoxLayout
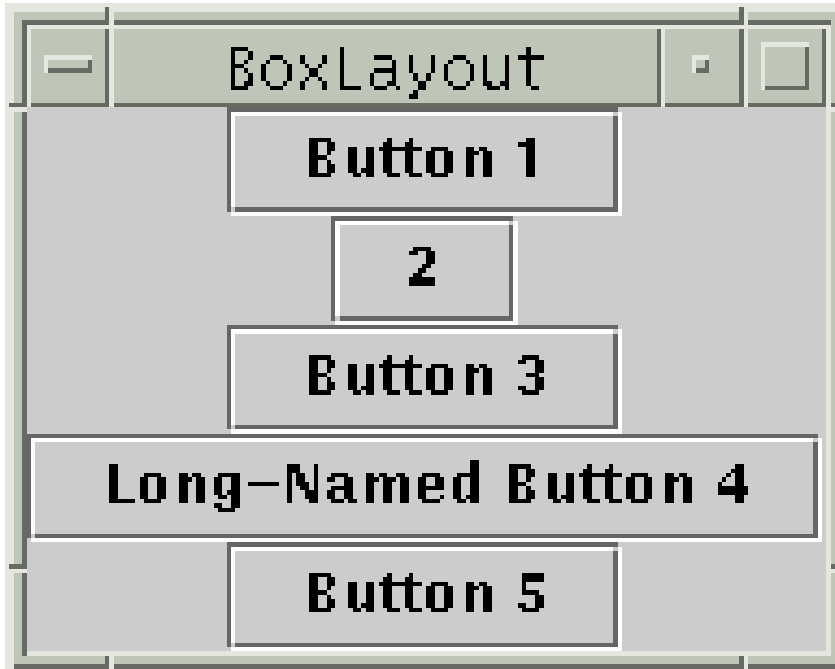
- The `BoxLayout` class puts components in a single row or column.

- It respects the components' requested maximum sizes, and also lets you align components.

GridLayout

❑ Places components in a requested number of rows and columns.

❑ Components are placed left-to-right and top-to-bottom.

❑ Forces all components to be the same size
  - as wide as the widest component's preferred width
 - as high as the highest component's preferred height
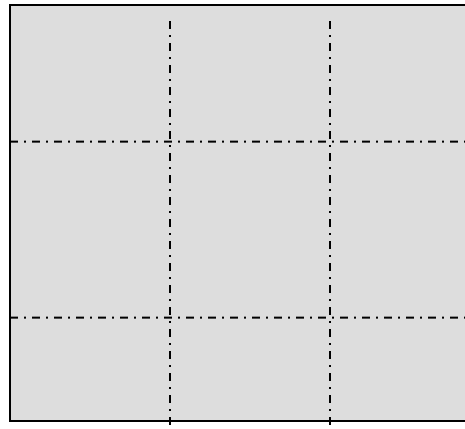
GridLayout example

```java
import javax.swing.*;
import java.awt.*;
public class grid {
    public static void main(String[] args) {
            JFrame frm = new JFrame("Grid Layout Test");
            Container contentPane = frm.getContentPane();
            contentPane.setLayout(new GridLayout(3, 4));
            for (int i=0; i<=9; ++i)
                    contentPane.add(new JButton(""+i));
            frm.pack();
            frm.setSize(318,220);
            //frm.setResizable(false);
            frm.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
            frm.setVisible(true);
    }
}
```

CardLayout

❑Components are organized as a deck of cards.

❑Can only be shown one at a time.

❑Methods:

- void add(String name, Component comp)
- void first(Container parent)
- void show(Container parent, String name)

One at a time
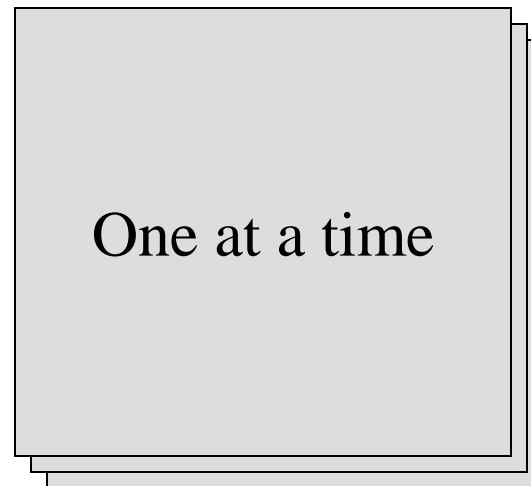
CardLayout example

```java
public class card implements ActionListener {
    static CardLayout contentPaneLayout;
    static JButton B1 = new JButton("To Card 2");
    static JButton B2 = new JButton("To Card 1");
    static JPanel contentPane;

    public void actionPerformed(ActionEvent e) {
        Object source = e.getSource();
        if (source==B1) contentPaneLayout.show(contentPane, "Card 2");
        if (source==B2) contentPaneLayout.show(contentPane, "Card 1");
}
```

```java
public static void main(String[] args) {
    JFrame frm = new JFrame("Card Layout Test");
    contentPane = (JPanel) frm.getContentPane();
    contentPane.setLayout(contentPaneLayout=new CardLayout());
    JPanel card1 = new JPanel();
    JPanel card2 = new JPanel();
    card1.add(new Label("This is card 1"));
    card2.add(new JTextField("This is Crad 2", 20));
    card1.add(B1); card2.add(B2);
    contentPane.add("Card 1", card1);
     contentPane.add("Card 2", card2) ;
    contentPaneLayout. show(contentPane, "Card 1");
    ActionListener AL = new card() ;
    B1.addActionListener(AL); B2.addActionListener(AL);
    frm.pack();
    frm.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    frm.setVisible(true);
    }
}
```

JOPTIONPANE
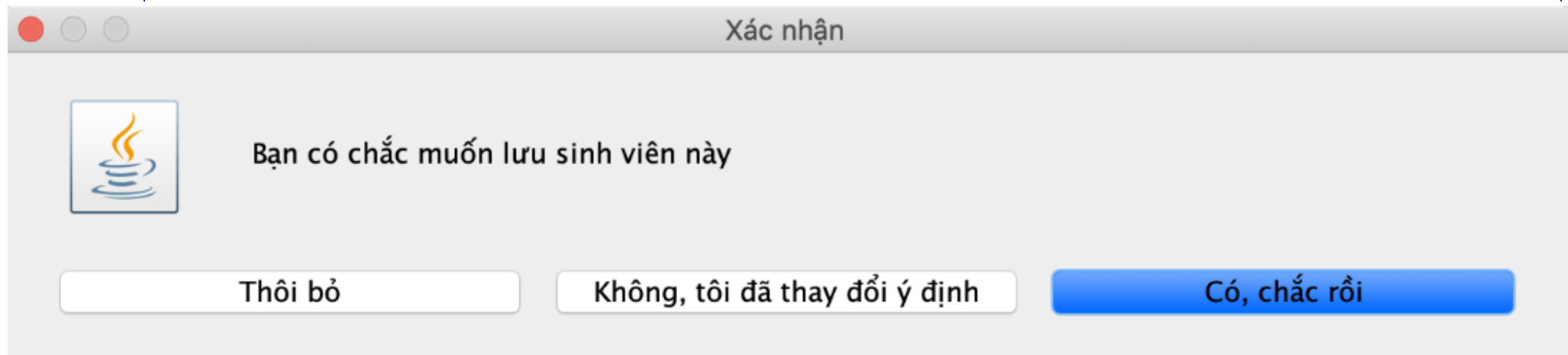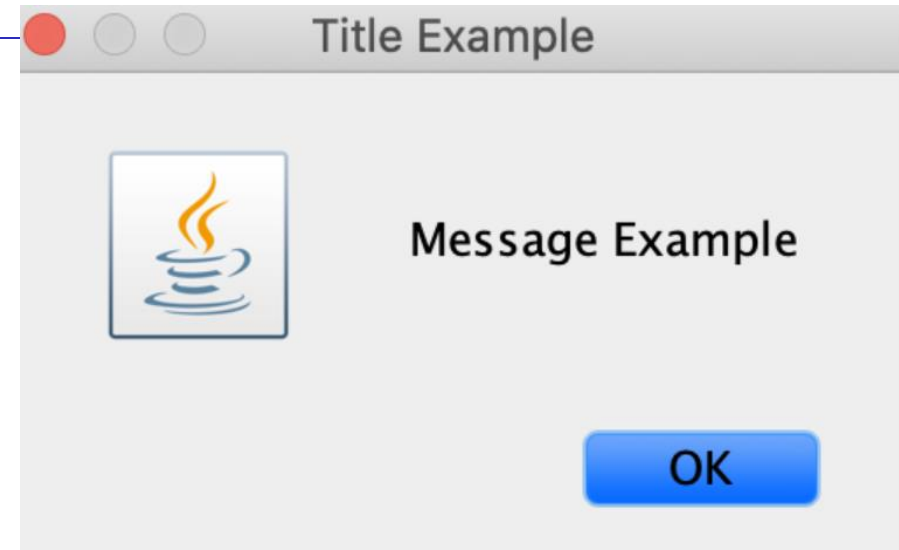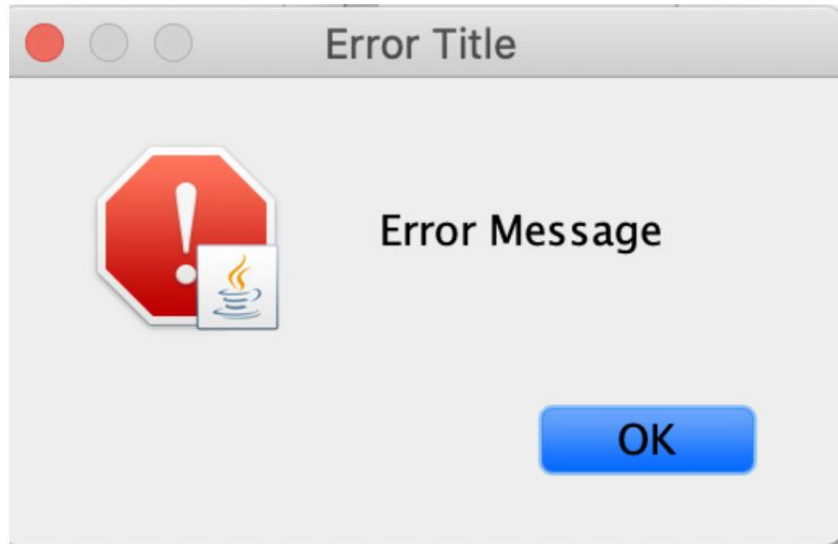
JOPTIONPANE

JOptionPane allows us to quickly create dialog boxes with different texts, icons, titles.

import javax.swing.JOptionPane;

Constuctors

`JOptionPane()`

Creates a JOptionPane with a test message.

`JOptionPane(Object message)`

Creates a instance of JOptionPane to display a message using the plain-message message type and the default options delivered by the UI.

`JOptionPane(Object message, int messageType)`

Creates an instance of JOptionPane to display a message with the specified message type and the default options,

`JOptionPane(Object message, int messageType, int optionType)`

Creates an instance of JOptionPane to display a message with the specified message type and options.

`JOptionPane(Object message, int messageType, int optionType, Icon icon)`

Creates an instance of JOptionPane to display a message with the specified message type, options, and icon.

`JOptionPane(Object message, int messageType, int optionType, Icon icon, Object[] options)`

Creates an instance of JOptionPane to display a message with the specified message type, icon, and options.

`JOptionPane(Object message, int messageType, int optionType, Icon icon, Object[] options, Object initialValue)`

Creates an instance of JOptionPane to display a message with the specified message type, icon, and options, with the initially-selected option specified.

JOPTIONPANE

JOptionPane allows us to quickly create dialog boxes with different texts, icons, titles.

import javax.swing.JOptionPane;

Method Name	Description
showConfirmDialog	Asks a confirming question, like yes/no/cancel.
showInputDialog	Prompt for some input.
showMessageDialog	Tell the user about something that has happened.
showOptionDialog	The Grand Unification of the above three.

JOPTIONPANE

Show an error dialog that displays the message, 'alert':

JOptionPane.showMessageDialog(null, "alert", "alert", JOptionPane.ERROR_MESSAGE);

Show an internal information dialog with the message, 'information':

JOptionPane.showInternalMessageDialog(frame, "information",
        "information", JOptionPane.INFORMATION_MESSAGE);

Show an information panel with the options yes/no and message 'choose one':

JOptionPane.showConfirmDialog(null,
        "choose one", "choose one", JOptionPane.YES_NO_OPTION);

Show an internal information dialog with the options yes/no/cancel and message 'please choose one' and title information:

JOptionPane.showInternalConfirmDialog(frame,
        "please choose one", "information",
        JOptionPane.YES_NO_CANCEL_OPTION, JOptionPane.INFORMATION_MESSAGE);

JOPTIONPANE

Show a warning dialog with the options OK, CANCEL, title 'Warning', and message 'Click OK to continue':

```
Object[] options = { "OK", "CANCEL" };
JOptionPane.showOptionDialog(null, "Click OK to continue", "Warning",
        JOptionPane.DEFAULT_OPTION, JOptionPane.WARNING_MESSAGE,
        null, options, options[0]);
```

Show a dialog asking the user to type in a String:

```
String inputValue = JOptionPane.showInputDialog("Please input a value");
```

Show a dialog asking the user to select a String:

```
Object[] possibleValues = { "First", "Second", "Third" };

Object selectedValue = JOptionPane.showInputDialog(null,
        "Choose one", "Input",
        JOptionPane.INFORMATION_MESSAGE, null,
        possibleValues, possibleValues[0]);
```

Show a dialog with Icon:

```
final ImageIcon icon = new ImageIcon(new URL("http://www.java2s.com/style/download.png"));
    JOptionPane.showMessageDialog(null, "Blah blah blah", "About", JOptionPane.INFORMATION_MESSAGE,
icon);
```

JFILECHOOSER

JFileChooser provides a simple mechanism for the user to choose a file.
import javax.swing.JFileChooser;

```java
public JFileChooserExam() throws HeadlessException {
    JFileChooser chooser = new JFileChooser();
    FileNameExtensionFilter filter = new FileNameExtensionFilter("JPG & GIF Images", "jpg", "gif");
    chooser.setFileFilter(filter);
    int returnVal = chooser.showOpenDialog(null);
    if (returnVal == JFileChooser.APPROVE_OPTION) {
        System.out.println("You chose to open this file: "
            + chooser.getSelectedFile().getName());
    }
}
public static void main(String[] args) {
    new JFileChooserExam();
}
```
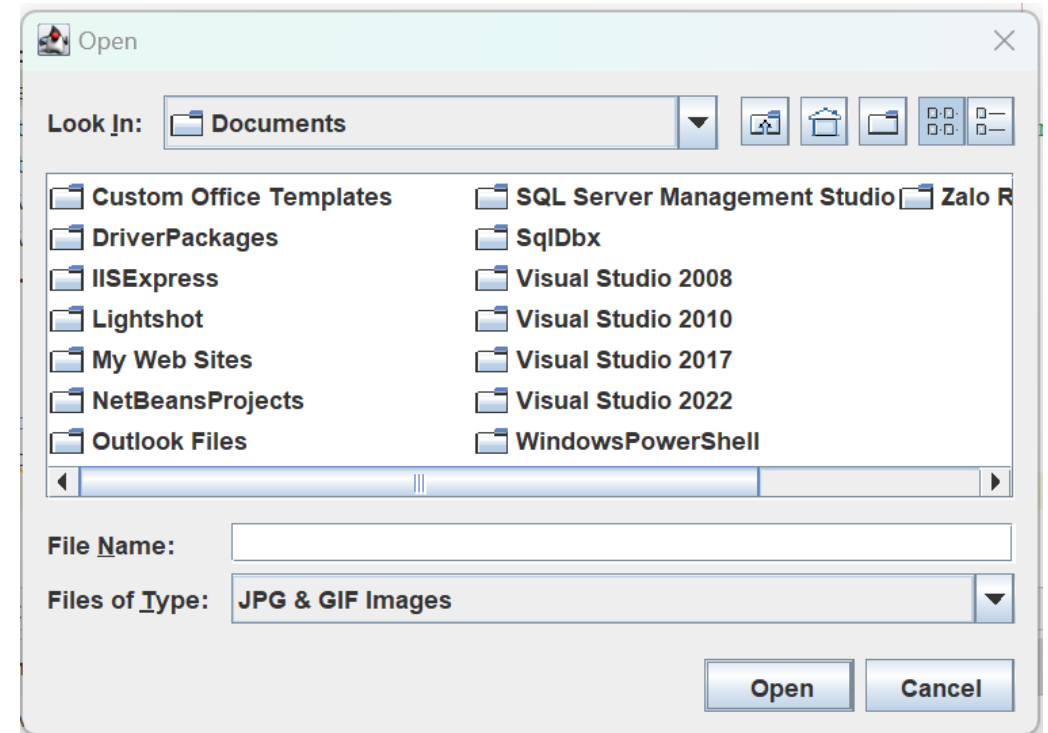
JFILECHOOSER

Show save file dialog using JFileChooser

```java
// parent component of the dialog
JFrame parentFrame = new JFrame();

JFileChooser fileChooser = new JFileChooser();
fileChooser.setDialogTitle("Specify a file to save");

int userSelection = fileChooser.showSaveDialog(parentFrame);

if (userSelection == JFileChooser.APPROVE_OPTION) {
    File fileToSave = fileChooser.getSelectedFile();
    System.out.println("Save as file: " + fileToSave.getAbsolutePath());
}
```
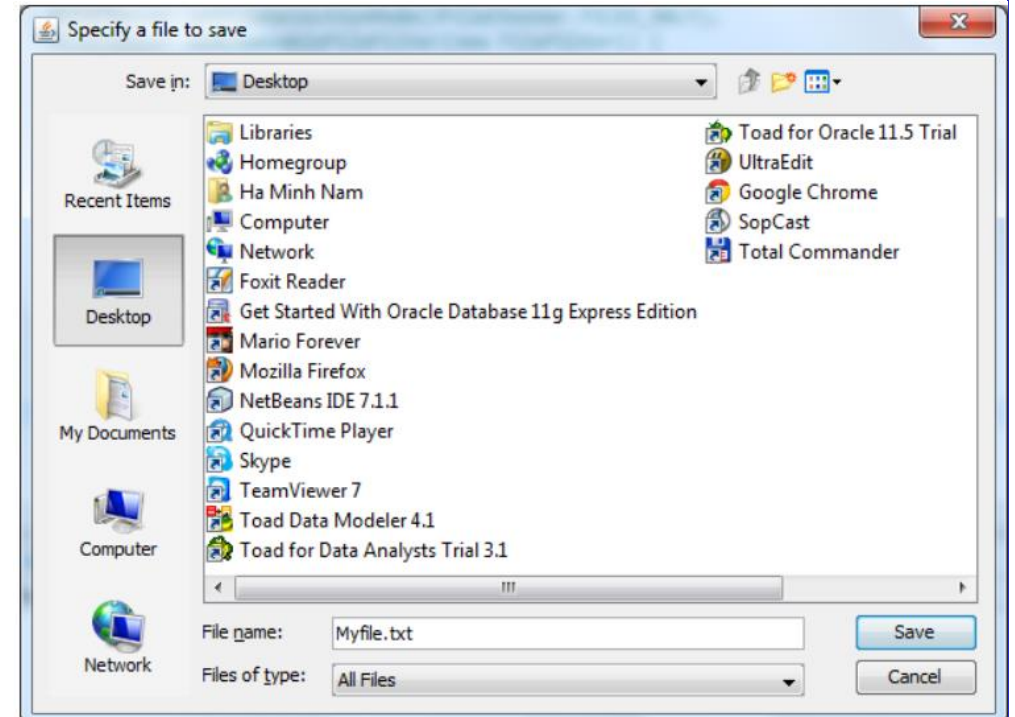
JFILECHOOSER

Show simple open file dialog using JFileChooser

```java
JFileChooser fileChooser = new JFileChooser();
fileChooser.setCurrentDirectory(new File(System.getProperty("user.home")));
int result = fileChooser.showOpenDialog(this);
if (result == JFileChooser.APPROVE_OPTION) {
    File selectedFile = fileChooser.getSelectedFile();
    System.out.println("Selected file: " + selectedFile.getAbsolutePath());
}
```

EVENT

An event in Swing is an action taken by a user. For example, pressing a button, pressing a key down/up on the keyboard.

The source of an event is the component that generates the event. For example, when we press a JButton, the clicked event occurs on that JButton. In this case, the JButton is the source of the clicked event.

An event represents the action that takes place on the source component.

EVENT

Steps Involved in Event Handling

Step 1 − The user clicks the button and the event is generated.

Step 2 − The object of concerned event class is created automatically and information about the source and the event get populated within the same object.

Step 3 − Event object is forwarded to the method of the registered listener class.

Step 4 − The method is gets executed and returns.

EVENT

Event Listeners

Sr.No.	Class & Description
1	**ActionListener** This interface is used for receiving the action events.
2	**ComponentListener** This interface is used for receiving the component events.
3	**ItemListener** This interface is used for receiving the item events.
4	**KeyListener** This interface is used for receiving the key events.
5	**MouseListener** This interface is used for receiving the mouse events.
6	**WindowListener** This interface is used for receiving the window events.
7	**AdjustmentListener** This interface is used for receiving the adjustment events.
8	**ContainerListener** This interface is used for receiving the container events.
9	**MouseMotionListener** This interface is used for receiving the mouse motion events.
10	**FocusListener** This interface is used for receiving the focus events.

EVENT

Example

```
private void showEventDemo(){
    headerLabel.setText("Control in action: Button");

    JButton okButton = new JButton("OK");
    JButton submitButton = new JButton("Submit");
    JButton cancelButton = new JButton("Cancel");

    okButton.setActionCommand("OK");
    submitButton.setActionCommand("Submit");
    cancelButton.setActionCommand("Cancel");

    okButton.addActionListener(new ButtonClickListener());
    submitButton.addActionListener(new ButtonClickListener());
    cancelButton.addActionListener(new ButtonClickListener());

    controlPanel.add(okButton);
    controlPanel.add(submitButton);
    controlPanel.add(cancelButton);

    mainFrame.setVisible(true);
}
```
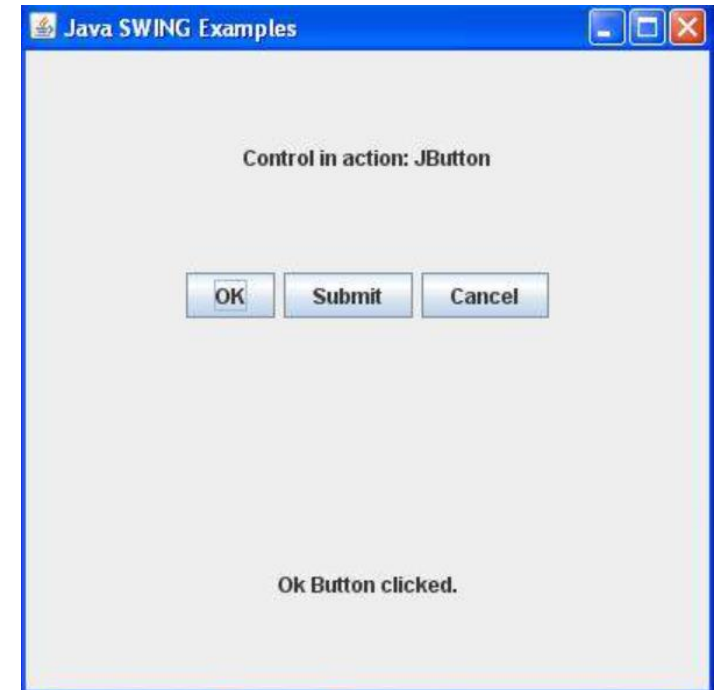
EVENT

Example

```
private class ButtonClickListener implements ActionListener{
    public void actionPerformed(ActionEvent e) {
        String command = e.getActionCommand();

        if( command.equals( "OK" ))  {
            statusLabel.setText("Ok Button clicked.");
        } else if( command.equals( "Submit" ) )  {
            statusLabel.setText("Submit Button clicked.");
        } else {
            statusLabel.setText("Cancel Button clicked.");
        }
    }
}
```
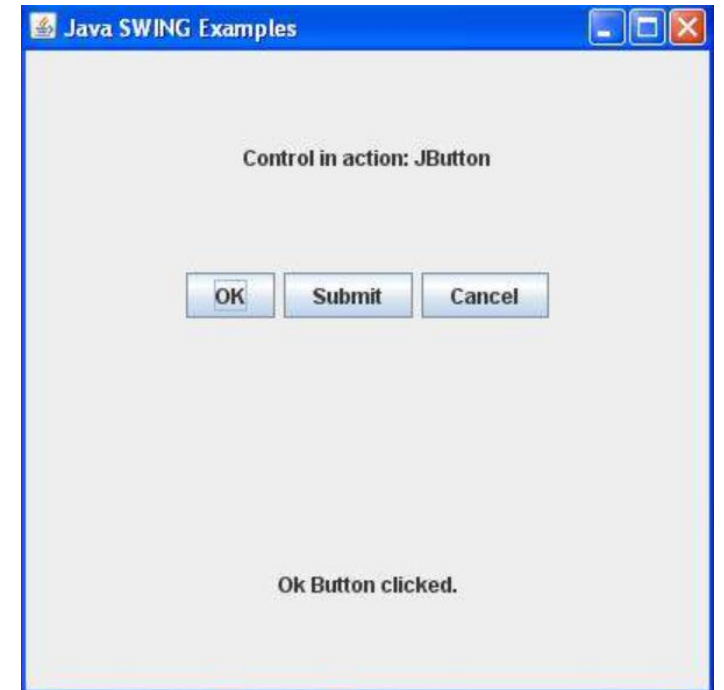
GRAPHICS

java.awt.Graphics class provides many methods for graphics programming
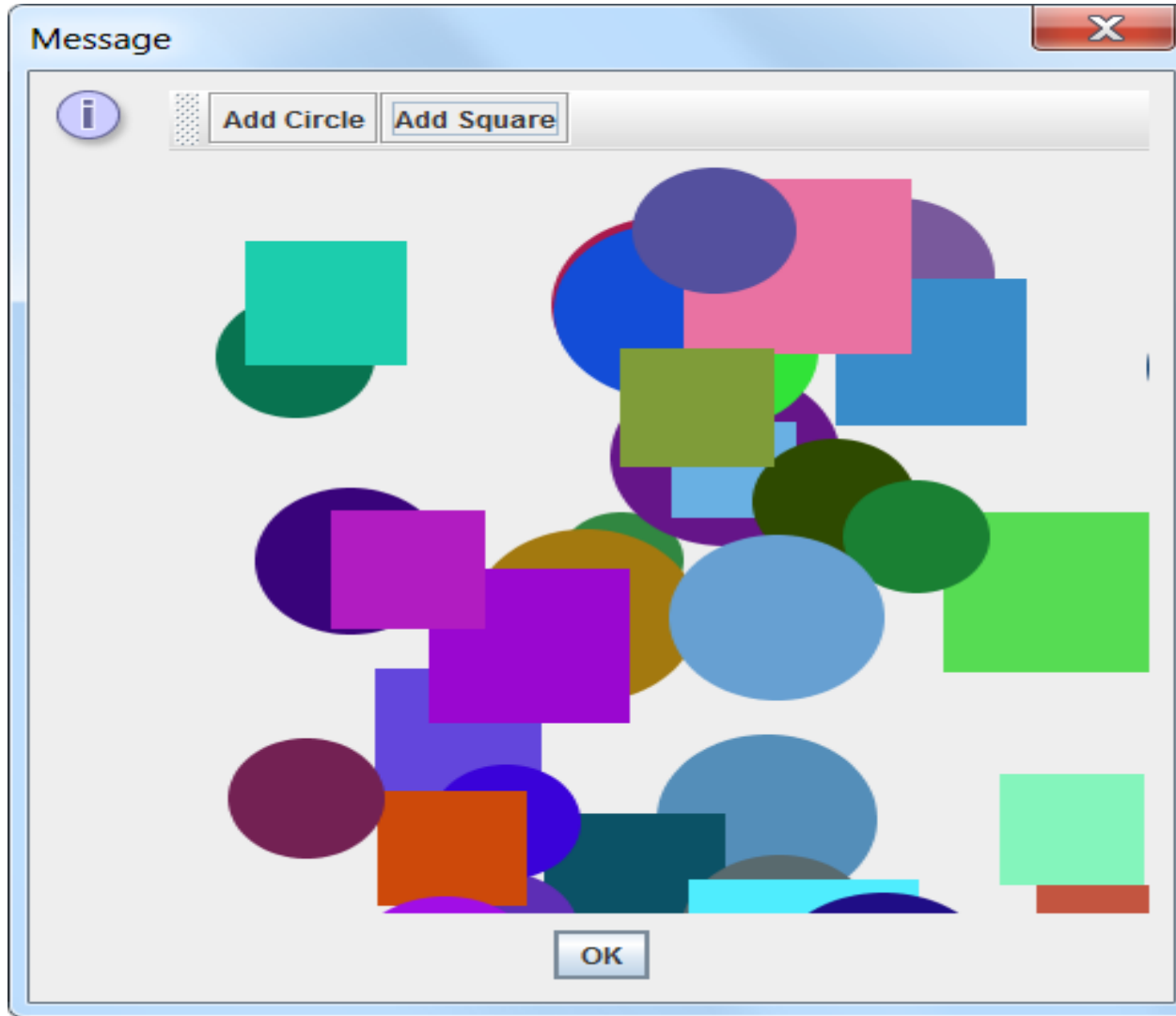
GRAPHICS

Methods of Graphics:

public abstract void drawString(String str, int x, int y): is used to draw the specified string.

public void drawRect(int x, int y, int width, int height): draws a rectangle with the specified width and height.

public abstract void fillRect(int x, int y, int width, int height): is used to fill rectangle with the default color and specified width and height.

public abstract void drawOval(int x, int y, int width, int height): is used to draw oval with the specified width and height.

public abstract void fillOval(int x, int y, int width, int height): is used to fill oval with the default color and specified width and height.

public abstract void drawLine(int x1, int y1, int x2, int y2): is used to draw line between the points(x1, y1) and (x2, y2).

public abstract boolean drawImage(Image img, int x, int y, ImageObserver observer): is used draw the specified image.

public abstract void drawArc(int x, int y, int width, int height, int startAngle, int arcAngle): is used draw a circular or elliptical arc.

public abstract void fillArc(int x, int y, int width, int height, int startAngle, int arcAngle): is used to fill a circular or elliptical arc.

public abstract void setColor(Color c): is used to set the graphics current color to the specified color.

public abstract void setFont(Font font): is used to set the graphics current font to the specified font.
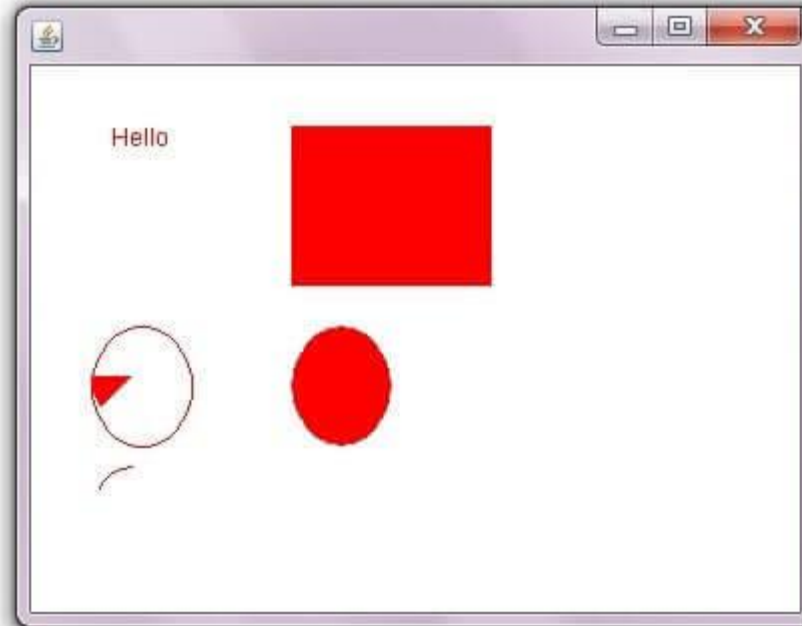
GRAPHICS

Example

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

public class DisplayGraphics extends Canvas{

   public void paint(Graphics g) {
      g.drawString("Hello",40,40);
      setBackground(Color.WHITE);
      g.fillRect(130, 30,100, 80);
      g.drawOval(30,130,50, 60);
      setForeground(Color.RED);
      g.fillOval(130,130,50, 60);
      g.drawArc(30, 200, 40,50,90,60);
      g.fillArc(30, 130, 40,50,180,40);
   }
      public static void main(String[] args) {
      DisplayGraphics m=new DisplayGraphics();
      JFrame f=new JFrame();
      f.add(m);
      f.setSize(400,400);
      //f.setLayout(null);
      f.setVisible(true);
   }
}
```

Assignments

Assignments

Assignments

Student Management System — Students

First Name	Last Name	Birthday	Grade	Class	Major
John	Doe	1999-01-01	A	C01	IT
Jane	Smith	2000-05-15	B	C02	Economic

Student Management System — Input

First Name:

Last Name:

Birthday:

Grade:

Class: C01

Major: IT

Add   Edit   Delete