Session 11

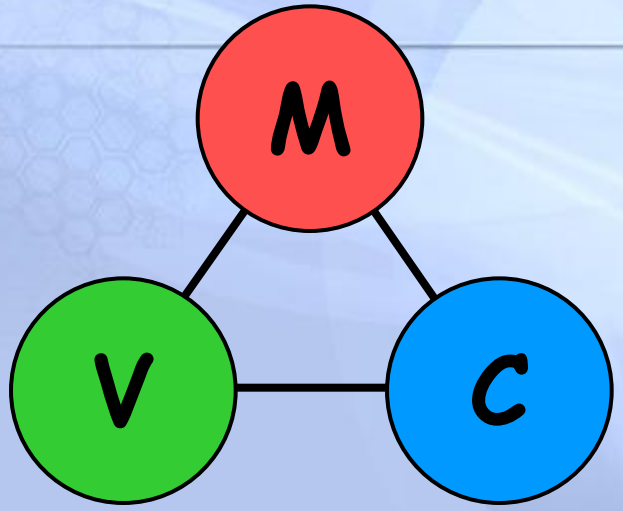
Advanced GUI

Outline

- Model View Controller
- Advanced Swing Components
  - JTree
  - JTable
  - JOptionPane
  - JSplitPane
  - Jslider
  - JProgressBar
  - JDeskTopPane &JInternalFrame

Model-View-Controller (MVC)

- SmallTalk strategy recently promoted by Sun
- Model = data and values, validation, ranges
- View = display
- Controller = what happens when user interacts with the component



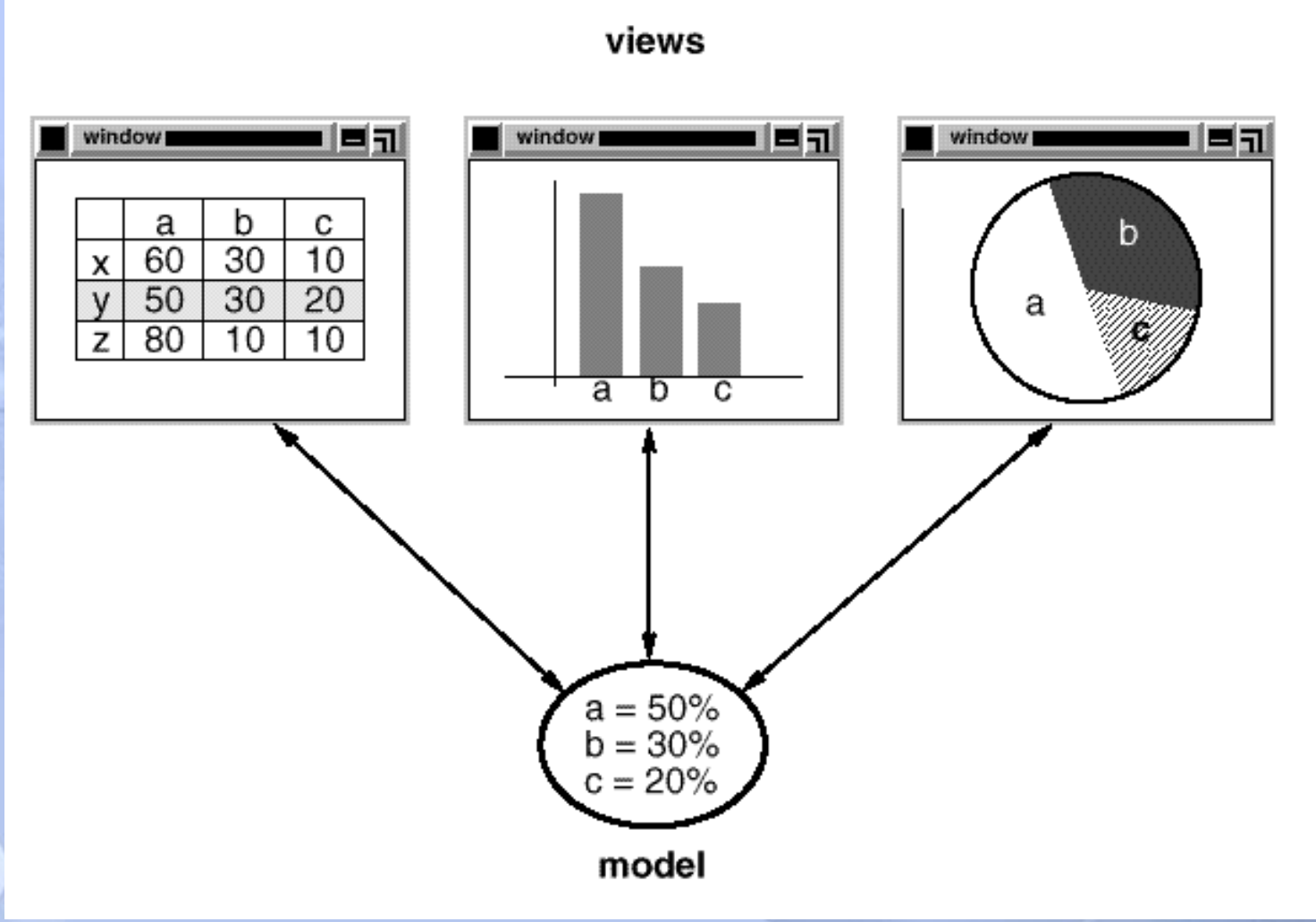
MVC Example: Swing

- Model
  - Standard Java classes
- View
  - JButton
  - JMenuItem, etc
- Controller
  - actionPerformed(..)
  - mouseClicked(..)

MVC Example: Swing

- MVC architecture assumes web applications are built of three types of components:
  - **Model**: represent data, provide data access and data processing methods
  - **View**: responsible for visualization; receive values from Model components and display them on a screen
  - **Controller**: receive user requests, parse the requests, call Model component methods, call View component methods

MVC


views
window
window
window
a b c
x 60 30 10
y 50 30 20
z 80 10 10
a b c
a
b
c
a = 50%
b = 30%
c = 20%
model

Advanced Swing Components

JTable & JTree

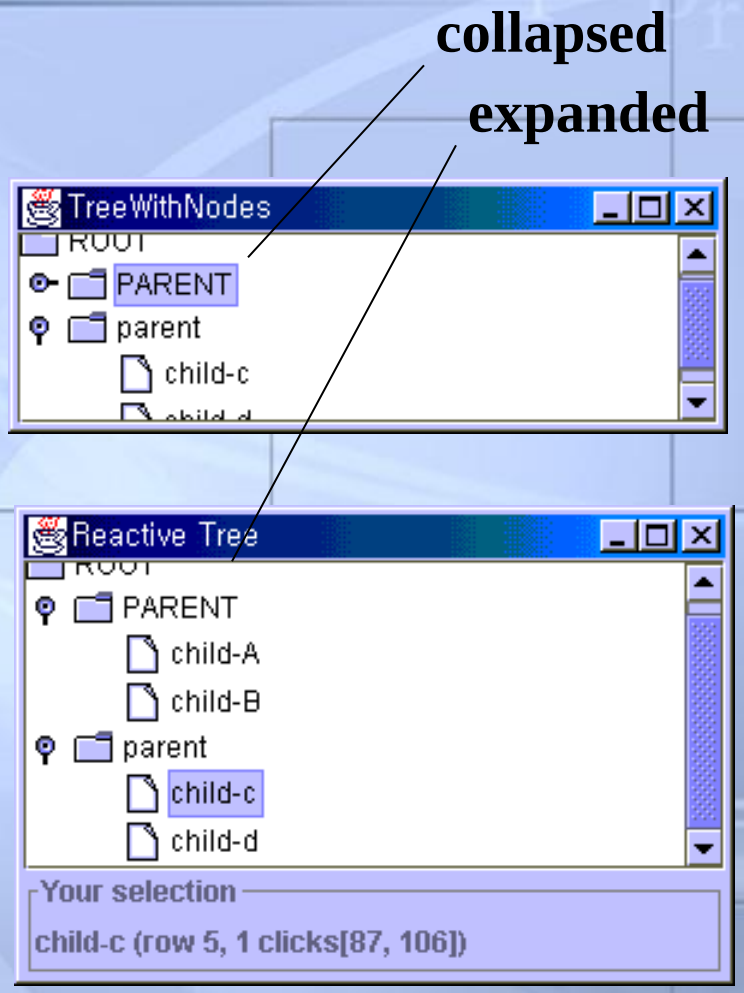
- Swing provides two sophisticated controls for structured information:
  - JTable
  - JTree
- The JTable class is best suited for tabular information.
- The JTree class is ideal for hierarchical information.

JTree

- A tree is a set of hierarchical nodes.
- The top of this hierarchy is called “root”.
- An element that does not have children is called “leaf”.
- **JTree** nodes are managed by a “**TreeModel**”.
- “**TreeCellRenderer**” converts an object to its visual representation in the tree.
- a component that shows data structures of tree organization
- a complex component with many sub-components
- MVC model
  - abstract data model : TreeModel > DefaultTreeModel
  - TreeNode > DefaultMutableTreeNode
  - DefaultTreeCellRenderer

Tree Model

- ROOT
  - PARENT
    - child-A
    - child-B
  - parent
    - child-c
    - child-d



JTree constructors

- JTree()
- JTree(Object[] value)
- JTree(Vector value)
- JTree(Hashtable value)
- JTree(TreeNode root)
- JTree(TreeNode root, boolean asksAllowsChildren)
- JTree(TreeModel newModel)

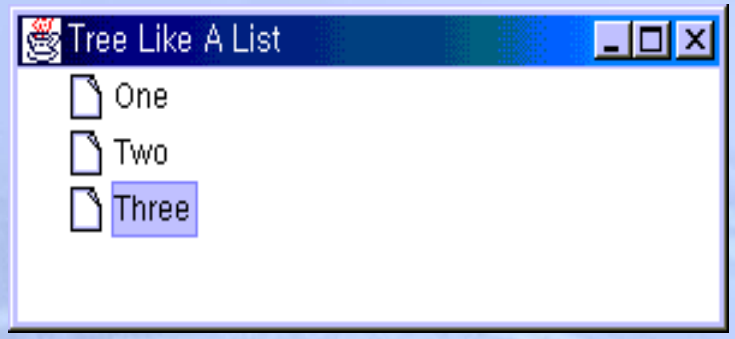
TreeModel Methods

- Object getRoot()
- Object getChild(Object parent, int index)
- int getChildCount(Object parent)
- boolean isLeaf(Object node)
- void valueForPathChanged(TreePath path, Object newVal)
- int getIndexOfChild(Object parent, Object child)
- void addTreeModelListener(TreeModelListener l)
- void removeTreeModelListener(TreeModelListener l)

TreeModel Listener

- TreeModelListener methods
  - void treeNodesChanged(TreeModelEvent e)
  - void treeNodesInserted(TreeModelEvent e)
  - void treeNodesRemoved(TreeModelEvent e)
  - void treeStructureChanged(TreeModelEvent e)
- TreeModelEvent methods
  - TreePath getTreePath()
  - Object[] getPath()
  - Object[] getChildren()
  - int[] getChildIndices()

Simple Example




```
Vector data = new Vector();

data.addElement("One");
data.addElement("Two");
data.addElement("Three");

JTree tree = new JTree(data);

getContentPane().add(tree);
```

TreeNodes

- Default tree model uses objects of TreeNode to represent the nodes in the tree.
- A default implementation of it is **DefaultMutableTreeNode**.
- **TreeNode** methods
  - TreeNode getChildAt(int childIndex)
  - int getChildCount()
  - TreeNode getParent()
  - int getIndex(TreeNode node)
  - boolean getAllowsChildren()
  - boolean isLeaf()
  - Enumeration children()

TreeNodes

- **DefaultMutableTreeNode** constructors
  - DefaultMutableTreeNode()
  - DefaultMutableTreeNode(Object userObject)
  - DefaultMutableTreeNode(Object userObject, boolean allowsChildren)

Output of the following JTree Example

JTree Example

```
import java.awt.*;
import javax.swing.*;
import javax.swing.tree.*;
public class TreeExample extends JFrame {
private JTree jtree;
public TreeExample()
{setSize(400,400);
 // root node
  DefaultMutableTreeNode filenode = new DefaultMutableTreeNode("TreeExample.java");
  // immediate children
  DefaultMutableTreeNode importnode = new DefaultMutableTreeNode("Imports");
  DefaultMutableTreeNode datanode = new DefaultMutableTreeNode("Variables");
  DefaultMutableTreeNode methodnode = new DefaultMutableTreeNode("Methods");
    filenode.add(importnode);
    filenode.add(datanode);
    filenode.add(methodnode);
```

```
DefaultMutableTreeNode awt = new DefaultMutableTreeNode("java.awt.*");
DefaultMutableTreeNode swing = new DefaultMutableTreeNode("javax.swing.*");
 DefaultMutableTreeNode swingtree = new DefaultMutableTreeNode("javax.swing.tree.*");
 importnode.add(awt);
 importnode.add(swing);
 importnode.add(swingtree);
 DefaultMutableTreeNode variable = new DefaultMutableTreeNode("JTree jtree");
 datanode.add(variable);
 DefaultMutableTreeNode method1 = new DefaultMutableTreeNode("JTree()");
 DefaultMutableTreeNode method2 = new DefaultMutableTreeNode("main()");
 methodnode.add(method1);
 methodnode.add(method2);
 DefaultTreeModel model = new DefaultTreeModel(filenode);
 jtree = new JTree(model);
 getContentPane().add("Center",new JScrollPane(jtree));
}
public static void main(String args[]) {
 TreeExample treeexample = new TreeExample();
 treeexample.setVisible(true);
}
```

Listener in JTree

```
public void valueChanged(TreeSelectionEvent e) {

    if(e.getSource()==jtree)
    {
        if(!jtree.isSelectionEmpty())
        {
        System.out.println("Selection Event");
        System.out.println("Path : " + jtree.getSelectionPath().toString());
        }
    }
}
```

```
//to register
                    jtree.addTreeSelectionListener(this);
```

JTable

JTable

- JTable displays tabular data in rows and columns
- Header for the title of each column
- Simple two dimensional display

- *Supports*
  - custom data models
  - custom cell rendering
  - custom header rendering
- Render any component inside the cell

- Highly flexible component
- Can be customized by the programmer
- Class is called **JTable**

JTable Constructors

- Table()
- Table(int *rows*, int *columns*)
- Table(Object[][] *rowData*, Object[] *columnNames*)
- Table(Vector *rowData*, Vector *columnNames*)
- Table(TableModel *model*)
- Table(TableModel *model*, TableColumnModel *tcModel*)
- Table(TableModel *model*, TableColumnModel *tcModel*, ListSelectionModel *lsModel*)

TableModel

- TableModel Methods
  - void addTableModelListener(TableModelListener l)
  - void removeTableModelListener(TableModelListener l)
  - int getRowCount()
  - int getColumnCount()
  - String getColumnName(int columnIndex)
  - Class getColumnClass(int columnIndex)
  - boolean isCellEditable(int rowIndex, int columnIndex)
  - Object getValueAt(int rowIndex, int columnIndex)
  - Object setValueAt(Object aValue, int rowIndex, int columnIndex)

TableModel

- AbstractTableModel (methods that must be implemented)
  - int getRowCount()
  - int getColumnCount()
  - Object getValueAt(int rowIndex,int columnIndex)

TableModelListener

- TableModelListener methods
  - void tableChanged(TableModelEvent evt)

- TableModelEvent methods
  - int getColumn()
  - int getFirstRow()
  - int getLastRow()
  - int getType()

JTable *Example(1)*

```
import java.awt.*;
import javax.swing.*;
public class SimpleTableExample  extends  JFrame {
  public SimpleTableExample() {
    setSize(600,300);

    // create the columns and the values
    String[] columns = {"Name","Telephone","City","Company"};
    String values[][] = {
                    {"Mr. X","703-4228989","Herndon","Bell Atlantic"},
                    {"Mr. Z","301-6748989","Rockville","Artesia Tech"},
                    {"Mr. W","703-4258999","Herndon","Intersect Soft"},
                    {"Mr. A","703-7864456","Herndon","Intelsat"}
                  };
```

JTable *Example(cont.)*

```
  JTable table = new JTable(values,columns);
  JScrollPane pane = new JScrollPane(table);
  // add the table to the frame
  getContentPane().add(pane);
 }

 public static void main(String[] args) {
  SimpleTableExample simpleTableExample = new
   SimpleTableExample();
  simpleTableExample.setVisible(true);
 }

} // end program
```

Output of following JTable Example(1)

- A JTable which shows information about people

Name	Telephone	City	Company
Mr. X	703-4228989	Herndon	Bell Atlantic
Mr. Z	301-6748989	Rockville	Artesia Tech
Mr. W	703-4258999	Herndon	Intersect Software
Mr. A	703-7864456	Herndon	Intelsat

JTable *Example(2)*

```
import java.awt.*;
import javax.swing.*;
import javax.swing.table.*;
public class SimpleTableExample extends Jframe{
String[] title={"Sub1","Sub2","Sub3","Sub4","Sub5"};
String[][] data={{"10","20","30","14","15"},{"16","17","18","29","36"},
        {"9","22","33","40","50"}};
public SimpleTableExample()
{
   setSize(400,300);
   this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
   JTable table = new JTable(new MyTableModel(data,title));
   //table.setShowHorizontalLines(false);
```

```
    table.setRowHeight(30);
    table.setBorder(BorderFactory.createEtchedBorder());
    JScrollPane pane = new JScrollPane(table);
    //pane.getViewport().setBackground(Color.white);
    table.setForeground(Color.blue);
    getContentPane().add(pane);
    this.setVisible(true);
}
public static void main(String[] args) {
     new SimpleTableExample();
}
}
```

```
class MyTableModel extends DefaultTableModel
{
    public MyTableModel(Object data[][],String[] title)
     {
        int colCount=title.length;

        for(int i=0;i<colCount;i++)
          this.addColumn(new String(title[i]));
        this.addColumn("Total");
```

```
        for(int i=0,n=data.length;i<n;i++) {
          int sum=0;
          Object[] obj=new Object[colCount+1];
          int j=0;
          for(j=0;j<colCount;j++) {
                    sum+=Integer.valueOf(data[i][j].toString());
                    obj[j]=data[i][j];
          }
          obj[j]="" + sum;
          this.addRow(obj);
        }
      }
}
```

Output of the JTable Example(2)

Sub1	Sub2	Sub3	Sub4	Sub5	Total
10	20	30	14	15	89
16	17	18	29	36	116
9	22	33	40	50	154

Rendering JTable

- Associating a cell renderer to a column

  TableColumn colum = table.getColumn("Telephone");

  column.**setCellRenderer**(new TelephoneRenderer());

- Table header rendering can be achieved by

  column.**setHeaderRenderer**(<renderer>)

- Table selection events can be observed by implementing the ListSelectionListener interface.

- ListSelectionEvent is generated whenever the selections change.

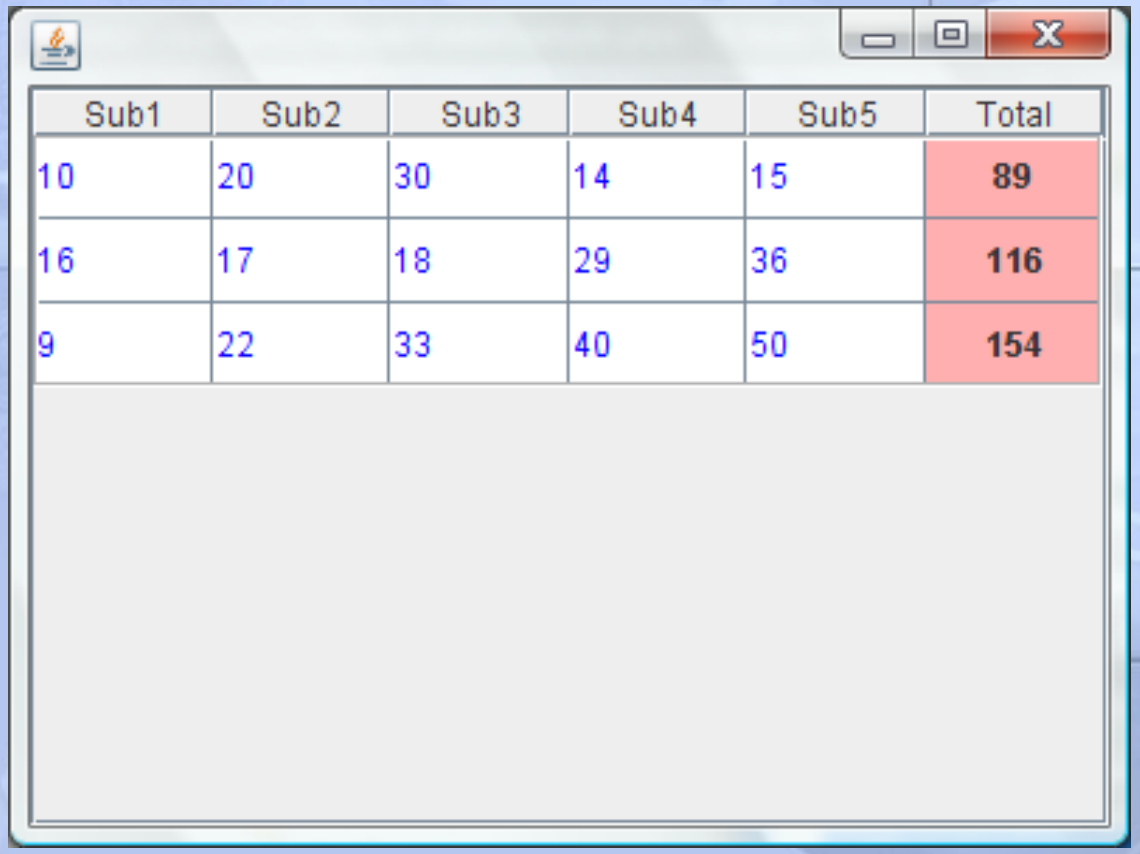
- table.getSelectionModel().**addListSelectionListener**(this);

Rendering JTable

```
class MyCellRenderer extends JLabel implements TableCellRenderer
{
        public MyCellRenderer() {
                setOpaque(true);
        }
        public Component getTableCellRendererComponent(JTable table,
                Object value,boolean isSelected,boolean hasFocus,int row,int
                column)
        {
            this.setHorizontalAlignment(JLabel.CENTER);
            this.setText((String)value);
            this.setBackground(Color.pink);
            return(this);
        }
}
```

Renderer - Output

```
table.getColumnModel().getColumn(5).setCellRenderer(new
    MyCellRenderer());
```



Sub1	Sub2	Sub3	Sub4	Sub5	Total
10	20	30	14	15	89
16	17	18	29	36	116
9	22	33	40	50	154

Listener - JTable

- Table selection events can be observed by implementing the **ListSelectionListener** interface.
- **ListSelectionEvent** is generated whenever the selections change.
- table.getSelectionModel().**addListSelectionListener**(this);

```
public void valueChanged(ListSelectionEvent event) {
   if (event.getSource() == table.getSelectionModel() && !event.getValueIsAdjusting()) {
      int row = table.getSelectedRow();
      int col=table.getSelectedColumn();
      if (row >= 0 && row < table.getRowCount()) {
        MyTableModel model = (MyTableModel)table.getModel();
        String value = (String)model.getValueAt(row,col);
        JOptionPane.showMessageDialog(this,"Selected : " + value);
      }
   }
}
```

Listener - Output

Sub1	Sub2	Sub3	Sub4	Sub5	Total
10	20	30	14	15	89
16	17	18	29	36	116
9	22	33	40	50	154

Message

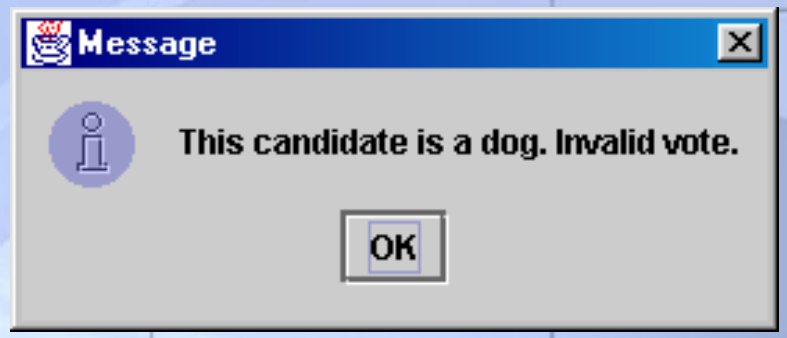
Selected : 30

OK

JOptionPane

JOptionPane

An option pane is a simple dialog box for graphical input/output



- **advantages**:
  - simple
  - flexible (in some ways)
  - looks better than the black box of death
- **disadvantages**:
  - created with static methods;
    not very object-oriented
  - not very powerful (just simple dialog boxes)

JOptionPane

javax.swing.JOptionPane

This class contains many simple methods for creating and using dialog boxes.

- There are both static methods and instance methods for dialogs.
- **4 Common Dialog Types:**
  - Message Dialog - display message only
  - Confirm Dialog - choose Yes, No, Cancel
  - Option Dialog - click button to make choice
  - Input Dialog - type a String, then click OK.

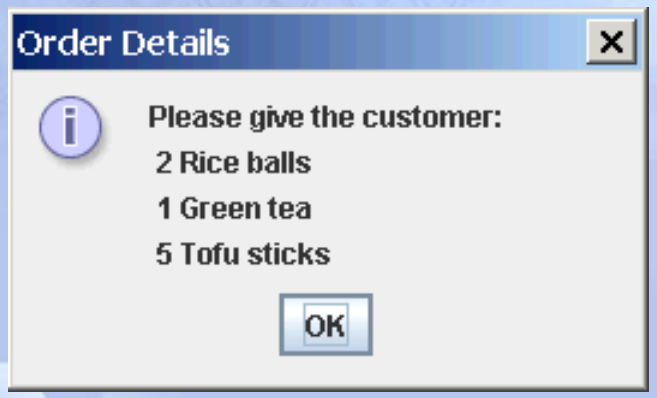
JOptionPane (Message Dialog)

- showMessageDialog analogous to System.out.println for displaying a simple message

```
import javax.swing.*;

public class MessageDialogExample {

    public static void main(String[] args) {

        JOptionPane.showMessageDialog(null,"How's the weather?");
        JOptionPane.showMessageDialog(null, "Second message");
    }
}
```

JOptionPane (Multiline Message Dialog)

```
String message = "Please give the customer:\n"
                 + "2 Rice balls\n"
                 + "1 Green tea\n"
                 + "5 Tofu sticks");

JOptionPane.showMessageDialog( null, message, "Order Details",
                 JOptionPane.INFORMATION_MESSAGE );
```



Use singular / plural

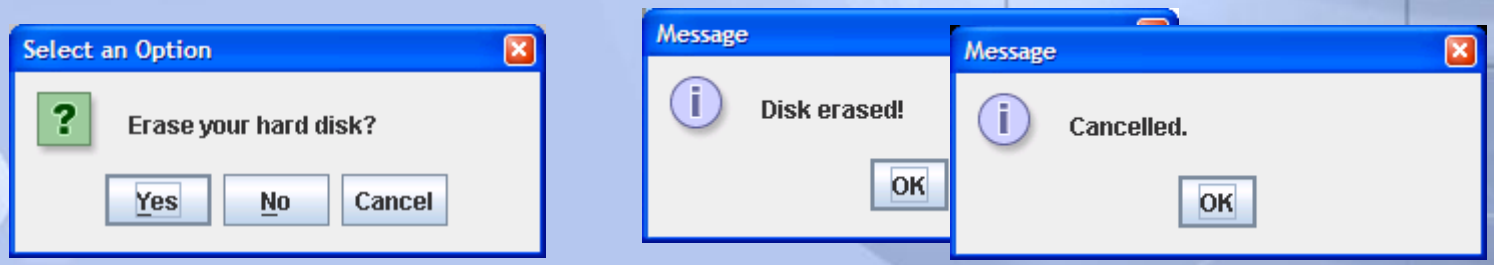
Avoid plurality errors like:

You have 1 new messages

JOptionPane (Comfirm Dialog)

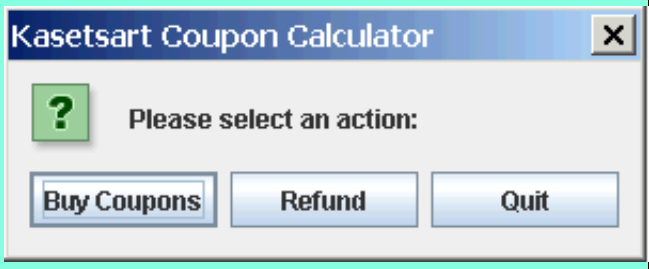
- showConfirmDialog analogous to a System.out.print that prints a question, then reading an input value from the user (can only be one of the provided choices)

```
import javax.swing.*;

public class ConfirmDialogExample {
    public static void main(String[] args) {
        int choice = JOptionPane.showConfirmDialog(null, "Erase your hard disk?");
        if (choice == JOptionPane.YES_OPTION) {
            JOptionPane.showMessageDialog(null, "Disk erased!");
        } else {
            JOptionPane.showMessageDialog(null, "Cancelled.");
        }
    }
}
```



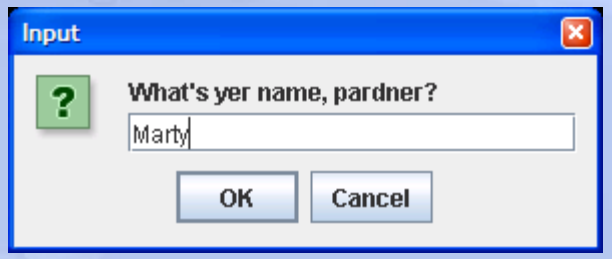
JOptionPane (Option Dialog)

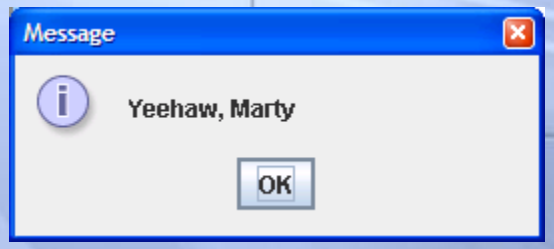
```
String [] choices = { "Buy Coupons",  "Refund", "Quit"};
int reply = JOptionPane.showOptionDialog( null,
      "Please select an action:",          // message
      "Kasetsart Coupon Calculator",  // title string
      JOptionPane.YES_NO_OPTION,          // useless
      JOptionPane.QUESTION_MESSAGE, // msg type
      null,                          // no icon
      choices,               // array of choices
      choices[0]             // default choice
            );
switch( reply ) {
      case 0:  couponDialog( ) ; break;
      case 1:  refundDialog( ) ; break;
      default:  confirmQuit( ) ;
}
```



JOptionPane (Input Dialog)

```
import javax.swing.*;

public class InputDialogExample {
    public static void main(String[] args) {
        String name = JOptionPane.showInputDialog(null,"What's yer
    name, pardner?");
        JOptionPane.showMessageDialog(null, "Yeehaw, " + name);
    }
}
```





JTabbedPane

JTabbedPane

A container that can hold many "tab" subcontainers, each with components in it

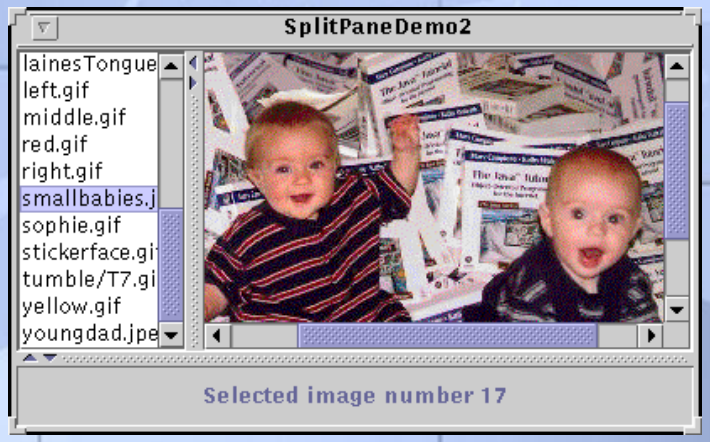
- public JTabbedPane()
- public JTabbedPane(int tabAlignment)
  - Constructs a new tabbed pane. Defaults to having the tabs on top; can be set to JTabbedPane.BOTTOM, LEFT, RIGHT, etc.
- public void addTab(String title, Component comp)
- public void addTab(String title, Icon icon,Component comp)
- public void addTab(String title, Icon icon,Component comp, String tooltip)
  - Adds the given component as a tab in this tabbed pane. Can optionally use an icon and/or tool tip.

JTabbedPane methods

- public void insertTab(String title, Icon icon, Component comp, String tooltip, int index)
- public void remove(Component comp)
- public void remove(int index)
- public void removeAll()
- public void setSelectedComponent(Component c)
- public void setSelectedIndex(int index)

JSplitPane

JSplitPane



A container that can hold two components, divided by a movable separator

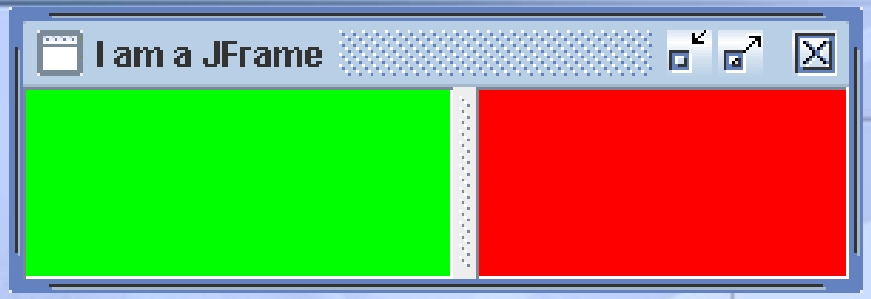
- public JSplitPane()
- public JSplitPane(int orientation)
  - Constructs a new tabbed pane. Defaults to having a horizontal split; can be set to JSplitPane.HORIZONTAL_SPLIT, VERTICAL_SPLIT, etc.
- public void setBottomComponent(Component comp)
- public void setLeftComponent(Component comp)
- public void setRightComponent(Component comp)
- public void setTopComponent(Component comp)
  - Sets the given component to occupy the desired region of the split pane.

Class JSplitPane

Syntax:

```
JSplitPane splitPane;
JPanel panel1, panel2;

splitPane = new JSplitPane ( JSplitPane.HORIZONTAL_SPLIT );
splitPane.setLeftComponent ( panel1 );
splitPane.setRightComponent ( panel2 );

getContentPane().add ( splitPane, BorderLayout.CENTER );
```

JSplitPane with JPanels




```
//Create a split pane
JSplitPane myPane = new JSplitPane();
myPane.setOpaque(true);
myPane.setDividerLocation(150);
    // make two panels
    JPanel right = new JPanel();
    right.setBackground(new Color(255,0,0));
    JPanel left = new JPanel();
    left.setBackground(new Color(0,255,0));
// set as left and right in split
myPane.setRightComponent(right);
myPane.setLeftComponent(left);
```

JSlider

JSlider

- The **JSlider** class represents a slider in which the user can move a nob to a desired position.
- The position of the nob on the slider determines the selected value.
- When a nob is moved, a **JSlider** object generates a change event (this event occurs when there is a change in the event source).
- The event listener for this object must implement the **ChangeListener** interface.

JSlider - Example

```
import java.awt.*;
import javax.swing.*;
import javax.swing.event.ChangeEvent;
import javax.swing.event.ChangeListener;
public class SliderTest extends JFrame
{
   JSlider slider1, slider2;
   public SliderTest()
   {
      super("Using JSlider");
      Container content = getContentPane();
      content.setBackground(Color.white);
      slider1 = new JSlider();
      slider1.setBorder(BorderFactory.createTitledBorder("JSlider without Tick
Marks"));
```

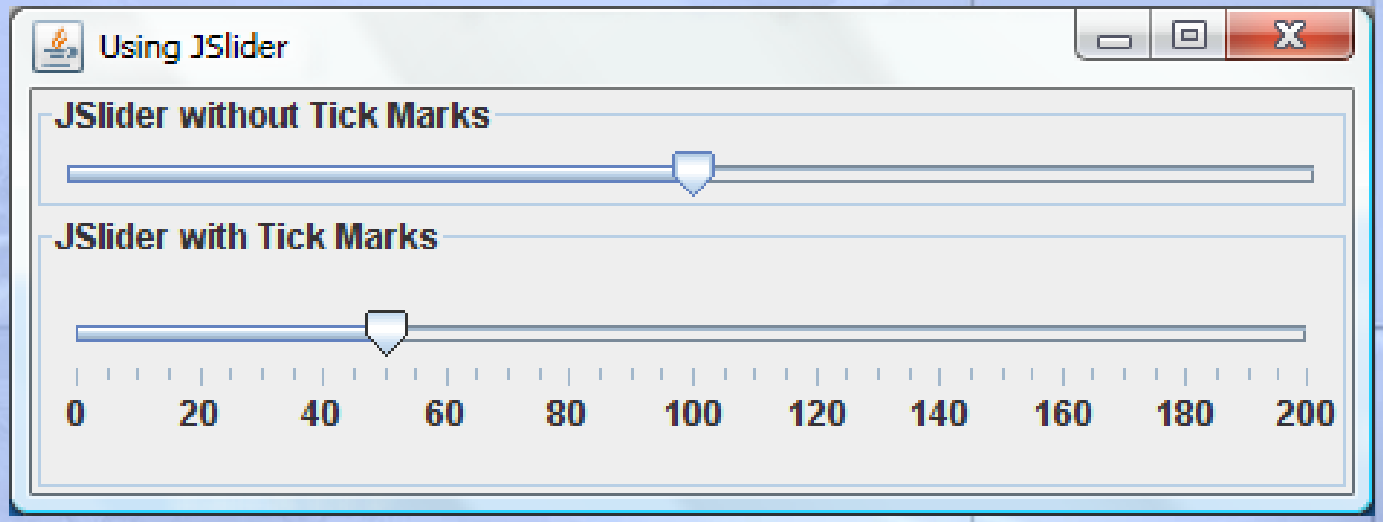
JSlider - Example

```
content.add(slider1, BorderLayout.NORTH);
slider2 = new JSlider();
slider2.setBorder(BorderFactory.createTitledBorder("JSlider with Tick Marks"));
slider2.setMaximum(200);
slider2.setMajorTickSpacing(20);
slider2.setMinorTickSpacing(5);
slider2.setPaintLabels(true);
slider2.addChangeListener(new ChangeListener()
{
  public void stateChanged(ChangeEvent arg0) {
  System.out.println(slider2.getValue());
  }
});
```

JSlider - Example

```
  slider2.setPaintTicks(true);
  content.add(slider2, BorderLayout.CENTER);

  this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
  pack();
  setVisible(true);
 }

    public static void main(String[] args) {
        new SliderTest();
    }

}
```

Output of Example



JProgressBar

JProgressBar - Example

```
import javax.swing.*;
public class progress extends JFrame
{
    JProgressBar jp=new JProgressBar();
    public progress() throws Exception
    {
        jp.setMaximum(100);
        this.getContentPane().add(jp);
        this.pack();
        this.setVisible(true);
```

```
        for(int i=1; i<=100; i++)
        {
            Thread.sleep(100);
            jp.setValue(i);
        }
        }

public static void main(String[] args)
        throws Exception
        {
        new progress();
        }
}
```

JProgressBar – Example with Thread

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

public class ProgressBarTest
{
  public static void main(String[] args)
  {
    ProgressBarFrame frame = new ProgressBarFrame();
    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    frame.setVisible(true);
  }
}
```

JProgressBar – Example with Thread

```
class ProgressBarFrame extends JFrame{
  public ProgressBarFrame()   {
    setTitle("ProgressBarTest");
    setSize(WIDTH, HEIGHT);
    Container contentPane = getContentPane();
    textArea = new JTextArea();
    JPanel panel = new JPanel();
    startButton = new JButton("Start");
    progressBar = new JProgressBar();
    progressBar.setStringPainted(true);
    panel.add(startButton);        panel.add(progressBar);
    contentPane.add(new JScrollPane(textArea), BorderLayout.CENTER);
    contentPane.add(panel, BorderLayout.SOUTH);
```

JProgressBar – Example with Thread

```
startButton.addActionListener(new
        ActionListener()
        {
          public void actionPerformed(ActionEvent event)
          {
            progressBar.setMaximum(1000);
            activity = new SimulatedActivity(1000);
            activity.start();
            activityMonitor.start();
            startButton.setEnabled(false);
          }
        });
```

JProgressBar – Example with Thread

```
activityMonitor = new Timer(500, new ActionListener()
        {
          public void actionPerformed(ActionEvent event) {
            int current = activity.getCurrent();
            textArea.append(current + "\n");
            progressBar.setValue(current);
            if (current == activity.getTarget())
            {
              activityMonitor.stop();
              startButton.setEnabled(true);
            }
          }
        });
  }//end ProgressBarFrame() constructor
```

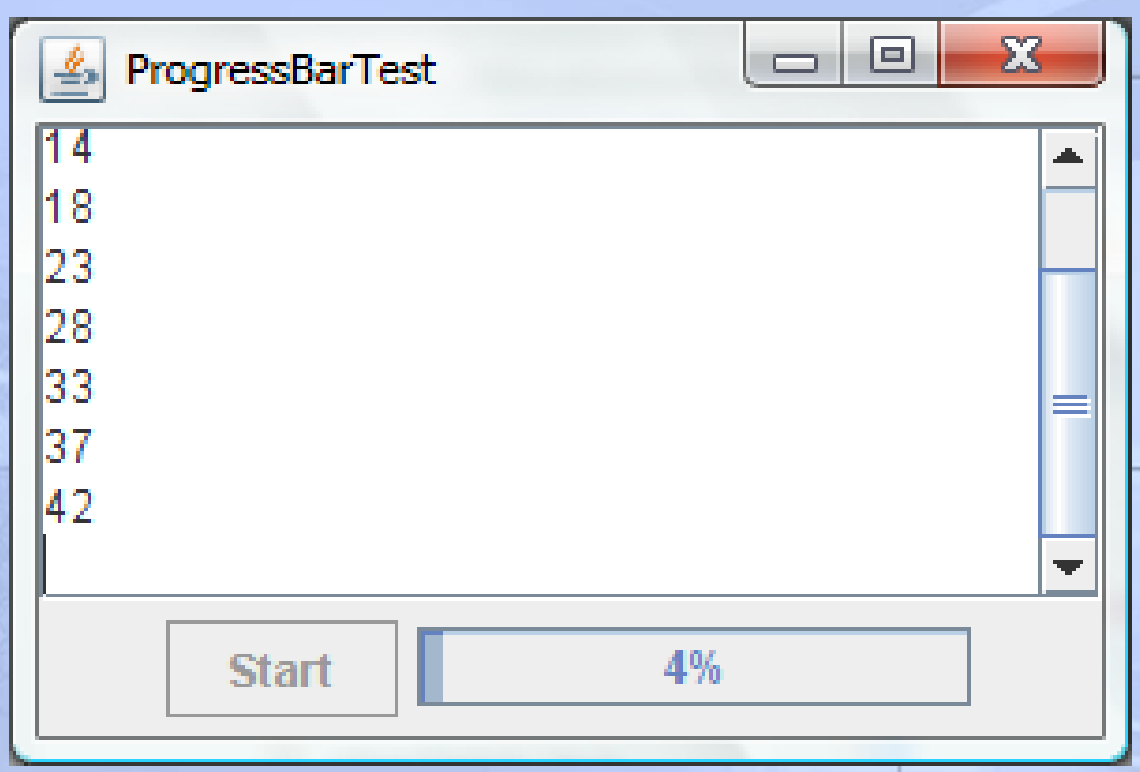
JProgressBar – Example with Thread

```
  private Timer activityMonitor;
  private JButton startButton;
  private JProgressBar progressBar;
  private JTextArea textArea;
  private SimulatedActivity activity;
  public static final int WIDTH = 300;
  public static final int HEIGHT = 200;
}
```

JProgressBar – Example with Thread

```
class SimulatedActivity extends Thread{
  public SimulatedActivity(int t)   {
    current = 0;
    target = t;   }
  public int getTarget()   {        return target;   }
  public int getCurrent()   {        return current;   }
  public void run()   {
    try      {         while (current < target && !interrupted())
                        {
                                sleep(100);
                                current++;
                         }
                 } catch(InterruptedException e) {        }
  }
  private int current;
  private int target;
}
```

Output



JToolBar

JToolBar

A movable container to hold common buttons, commands

, etc

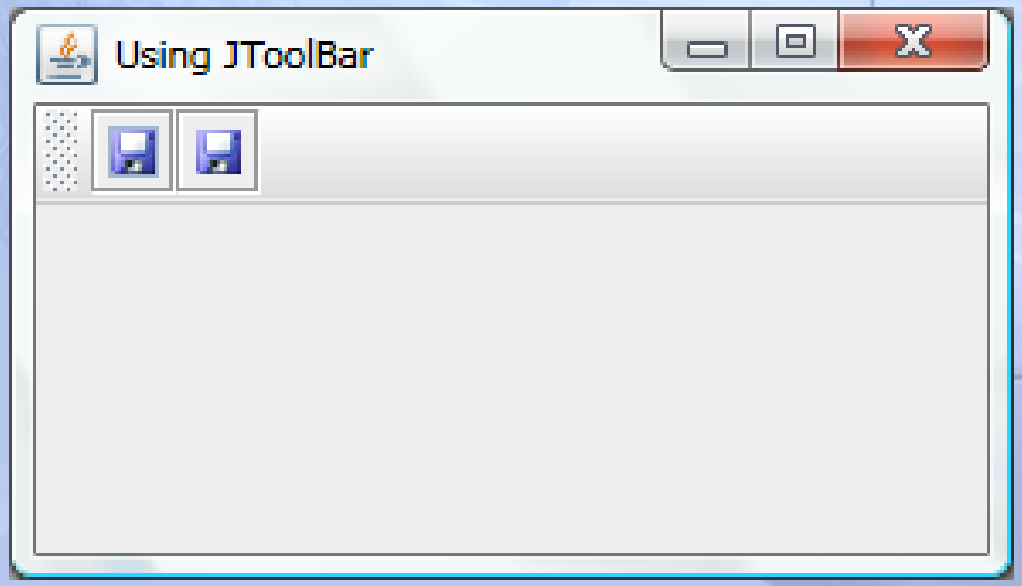
- public JToolBar()
- public JToolBar(int orientation)
- public JToolBar(String title)
- public JToolBar(String title, int orientation)
  - Constructs a new tool bar, optionally with a title and orientation; can be JToolBar.HORIZONTAL or VERTICAL, defaults to horizontal
- public void add(Component comp)
  - Adds the given component to this tool bar's horizontal/vertical flowing layout.

JToolBar: Usage

- construct toolbar
- add items (usually buttons) to toolbar
- add toolbar to edge of BorderLayout of content pane (usually NORTH)
  - don't put anything in other edges (N/S/E/W)!
- **JToolbar** allows toolbars to be added to GUI
- Can modify appearance, "dock" toolbar on top, side or bottom, can be a *floating* toolbar.
- A toolbar is a container, so it can contain other GUI components.
- *Orientation* specifies how child components are arranged.
- **Default is horizontal.**

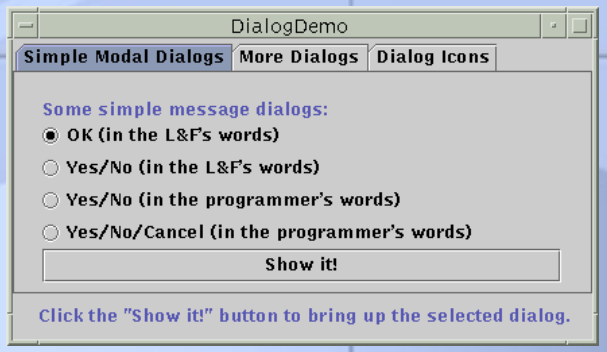
Example

```
JToolBar toolBar=new JToolBar();
toolBar.add(new JButton(new ImageIcon("./save.gif")));
toolBar.add(new JButton(new ImageIcon("./open.gif")));
//toolBar.setOrientation(JToolBar.VERTICAL);
contentPane.add(toolBar,BorderLayout.NORTH););
```



JDialog

A window that is a child of the overall JFrame; used for popup windows, option/config boxes, etc.



- **public JDialog(Dialog parent, boolean modal)**

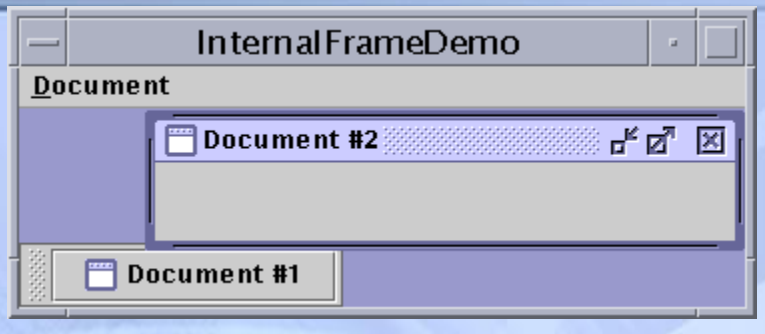
- **public JDialog(Frame parent, String title, boolean modal)**
  Constructs a new dialog with the given parent and title. If the modal flag is set, this dialog is a child of the parent and the parent will be locked until the dialog is closed.

- **public void show()**
  Shows this dialog on screen. If the dialog is modal, calling show() will lock the parent frame/dialog.

- JDialog has most all JFrame methods: getContentPane(), setJMenuBar(JMenuBar), setResizable(boolean), setTitle(String), ...

JDesktopPane & JInternalFrame

Desktop and internal frames



- A useful container for opening many items is the desktop pane (JDesktopPane) which displays internal frames (JInternalFrame).
- The internal frames act much like regular frames, but inside the desktop pane
  - You need to set their size and location.
  - You need to add them to a desktop pane and set them as visible to appear .
  - They have a content pane and a menu bar.
- However, internal frames are more easy to manage.

Multiple-Document Interfaces

- Multiple Document Interfaces (MDI) allow users to view multiple documents in a single application.
- Each document appears in a separate window.
- **JDesktopPane** and **JInternalFrame** are used to build MDI applications.
- **JDesktopPane** is the application's "desktop."
- **JInternalFrame** is similar to **JFrame**, with a title bar and buttons to iconify etc.
- A useful container for opening many items is the desktop pane (JDesktopPane) which displays internal frames (JInternalFrame)
- The internal frames act much like regular frames, but inside the desktop pane
  - need to set their size and location
  - need to add them to a desktop pane and set them as visible to appear

JDesktopPane & JInternalFrame

- Provide support for creating MDI applications
  - allow to embed frames inside parent frames
  - child frame can move within its parent and be minimized, maximized, or restored

- Class **JDesktopPane**
  - manages JInternalFrame child windows
  - getAllFrames() returns an array of all internal frames

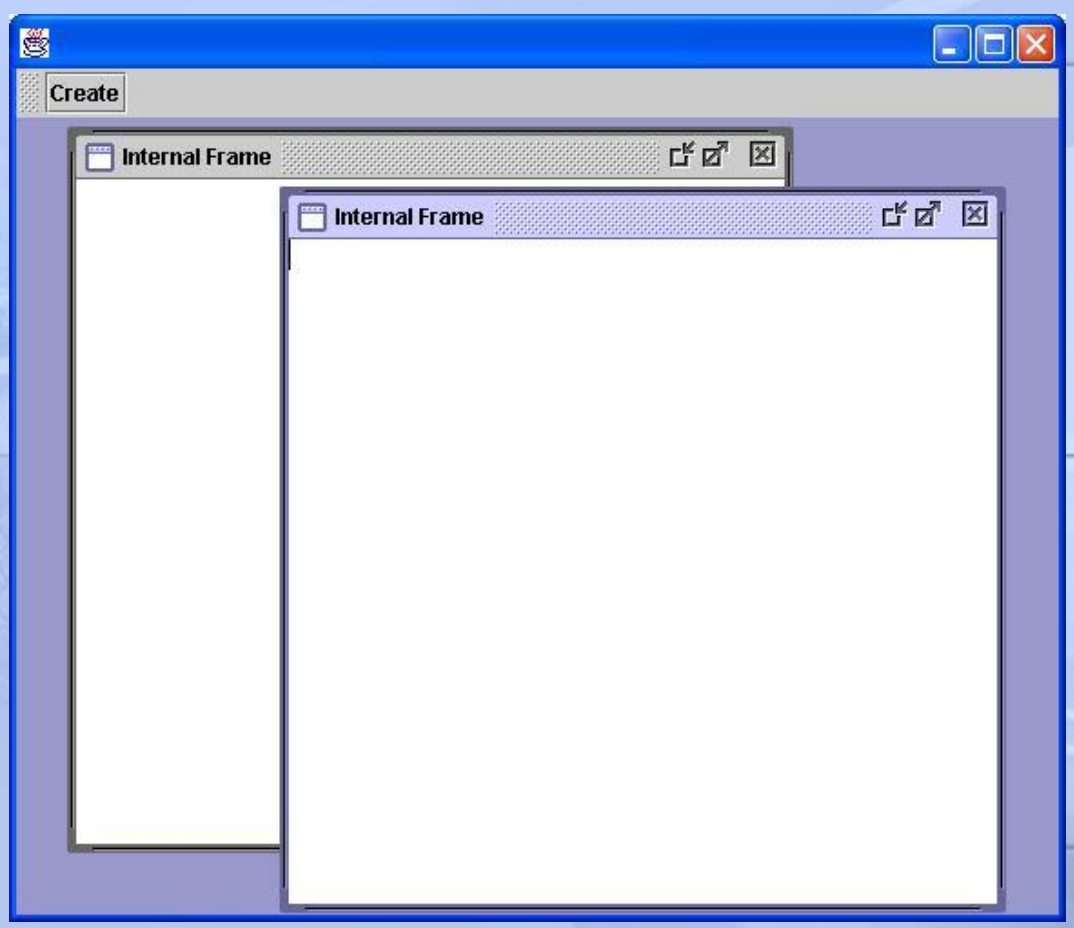
- Class **JInternalFrame**
  - has a content pane for GUI components to be attached

Class JInternalFrame

- JInternalFrame ();
- JInternalFrame (String title);
- isVisible(); setVisible (bool);
- setResizable (bool), setClosable (bool), setMaximizable (bool),
- setIconifiable (bool)
- moveToFront(), moveToBack()
- isSelected (), setSelected (bool)
- reshape (int x, int y, int width, int height)

- Other methods
  - setMenuBar(JMenuBar menubar)
  - addInternalFrameListener(InternalFrameListener listener)

JInternalFrame



JInternalFrame Example

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
public class InternalExample extends JFrame
                                        implements ActionListener {
  private JDesktopPane deskPane;
  public InternalExample() {
    setSize(500,500);
    // create a tool bar
    JToolBar toolbar = new JToolBar();
    JButton newButton = new JButton("Create");
    newButton.addActionListener(this);
    toolbar.add(newButton);
```

```
    // create a desktop pane
    deskPane = new JDesktopPane();
    getContentPane().add("Center",deskPane);
    getContentPane().add("North",toolbar);
  }

 public void actionPerformed(ActionEvent e) {
    TestInternalFrame frame = new TestInternalFrame();
    frame.setVisible(true);
    deskPane.add(frame);
  }

  public static void main(String[] args) {
    InternalExample internalExample = new InternalExample();
    internalExample.setVisible(true);
  }
} // end Internal Example
```

```
package internalexample;
import java.awt.*;
import javax.swing.*;
public class TestInternalFrame extends JInternalFrame {
  public TestInternalFrame() {
    super("Internal Frame");
    // set the frame properties
    setClosable(true);
    setMaximizable(true);
    setIconifiable(true);
    setResizable(true);
 // set the size and add test area
    setSize(400,400);
    getContentPane().add("Center",new JTextArea());
  }
} // end TestInternalFrame
```

Thank You!