JAVA PROGRAMMING BASICS

Module 3: Java Standard Edition

Training program

1. Java I/O Streams
2. Java Serialization
3. Java Database Connectivity
4. Java GUI Programming
5. The basics of Java class loaders
6. Reflections
7. Annotations
8. The proxy classes
9. Java Software Development
10. Garbage Collection

Module contents

Java GUI Programming

- An Introduction to Swing
- Swing - Controls Basics
- Event Handling
- Layout Managers
- Components

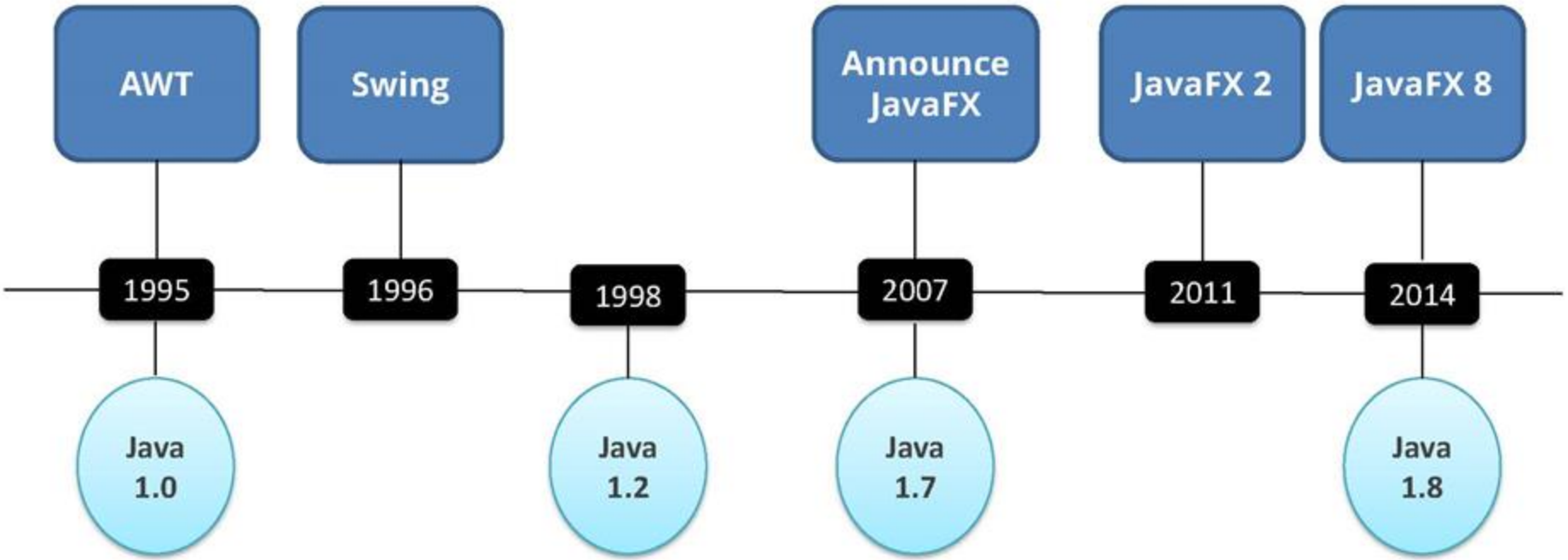
Module contents

Java GUI Programming

- An Introduction to Swing
- Swing - Controls Basics
- Event Handling
- Layout Managers
- Components

An Introduction to Swing

since Java 11 -> https://openjfx.io/



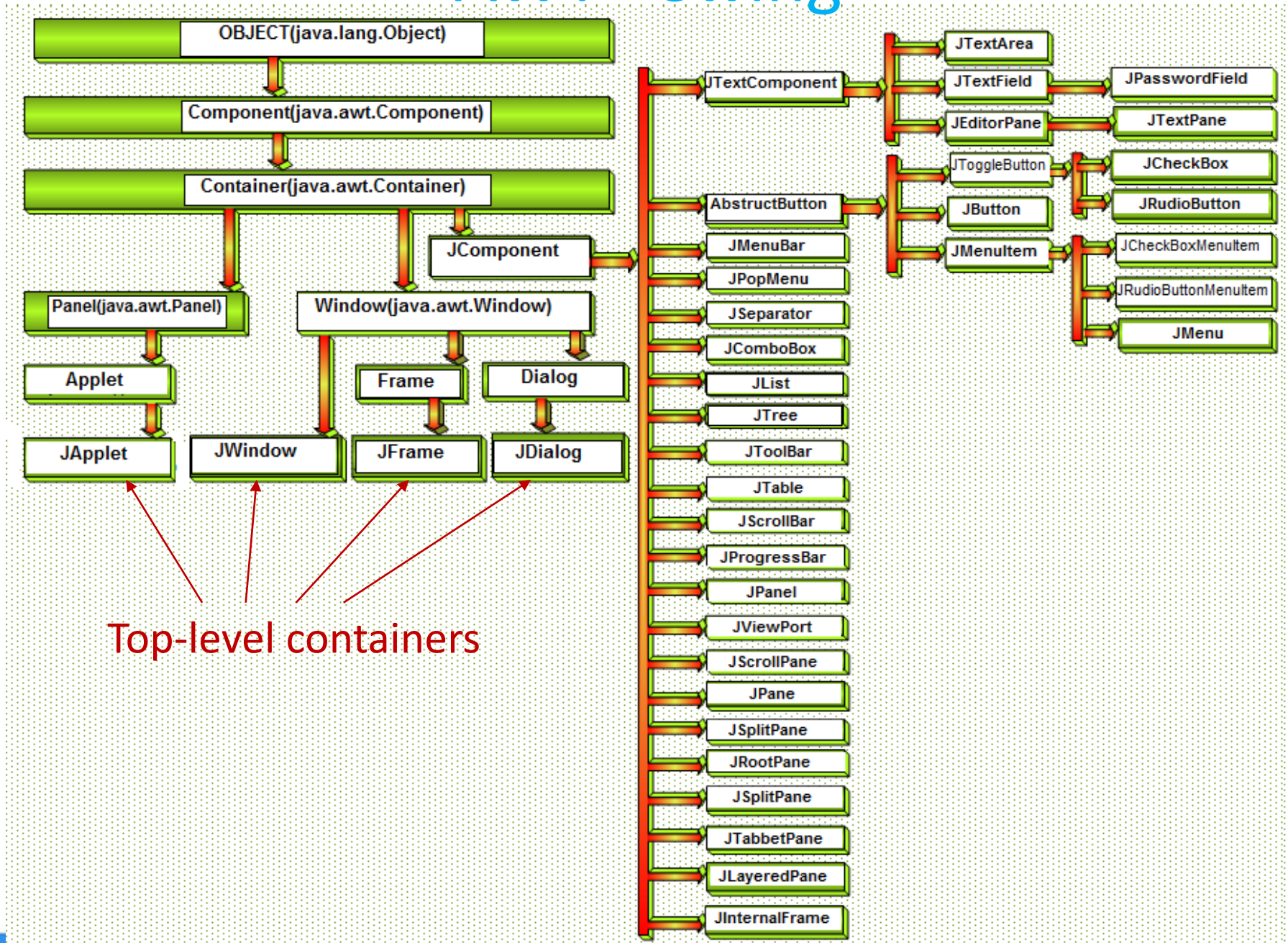
until Java 11

An Introduction to Swing

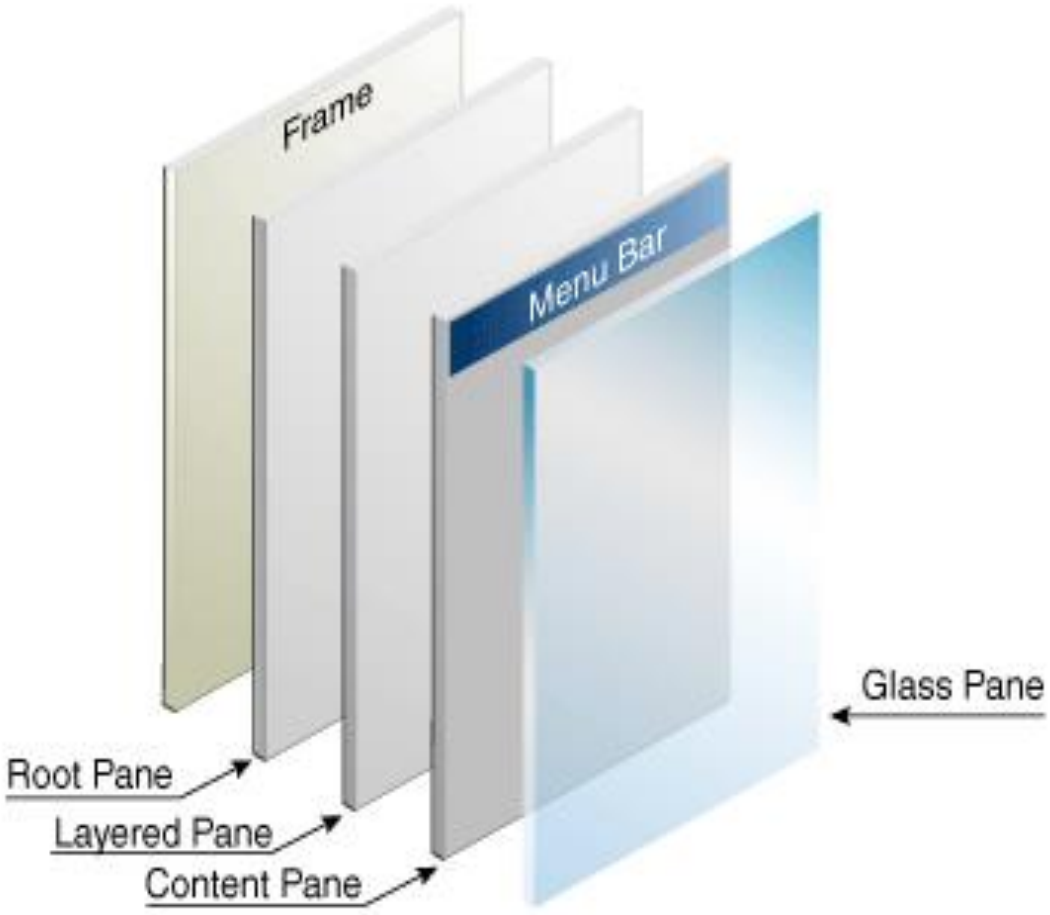
- Swing API is set of extensible GUI Components to ease developer's life to create JAVA GUI Applications.
- It is build upon top of AWT API and acts as replacement of AWT API as it has almost every control corresponding to AWT controls.

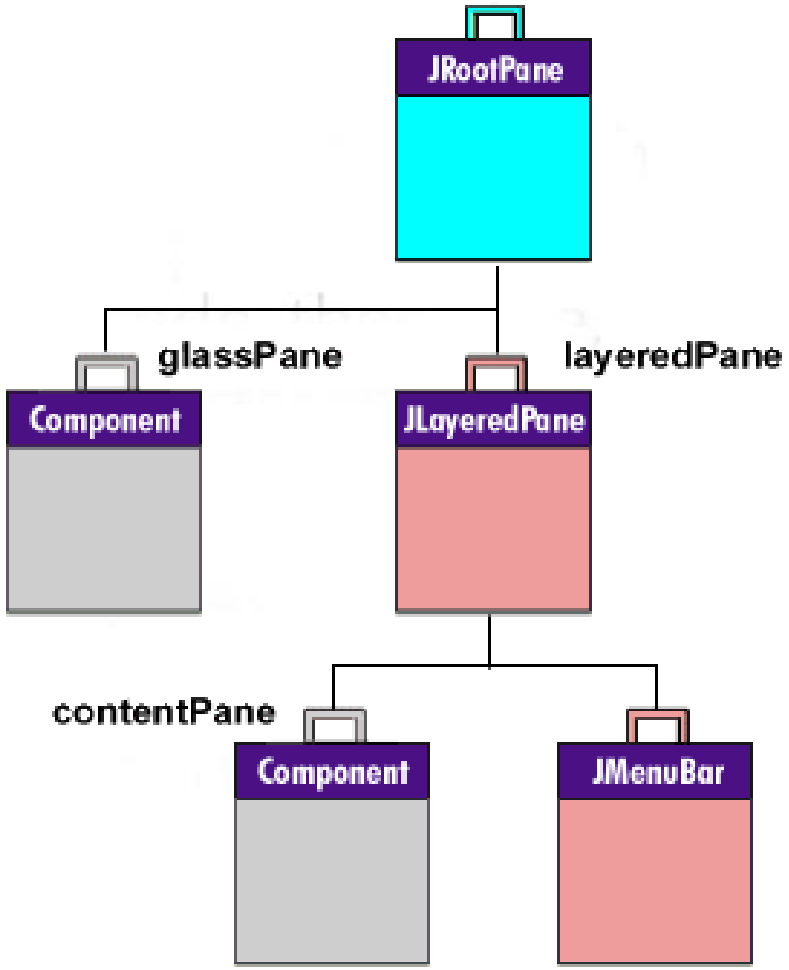
Vs

AWT - Swing


OBJECT(java.lang.Object)
Component(java.awt.Component)
Container(java.awt.Container)
JComponent
Panel(java.awt.Panel)
Window(java.awt.Window)
Applet
Frame
Dialog
JApplet
JWindow
JFrame
JDialog
Top-level containers
JTextComponent
JTextArea
JTextField
JPasswordField
JEditorPane
JTextPane
AbstructButton
JToggleButton
JCheckBox
JRudioButton
JButton
JMenuItem
JCheckBoxMenuItem
JRudioButtonMenuItem
JMenu
JMenuBar
JPopMenu
JSeparator
JComboBox
JList
JTree
JToolBar
JTable
JScrollBar
JProgressBar
JPanel
JViewPort
JScrollPane
JPane
JSplitPane
JRootPane
JSplitPane
JTabbetPane
JLayeredPane
JInternalFrame

Top-level containers structure





See basic\SwingDemo

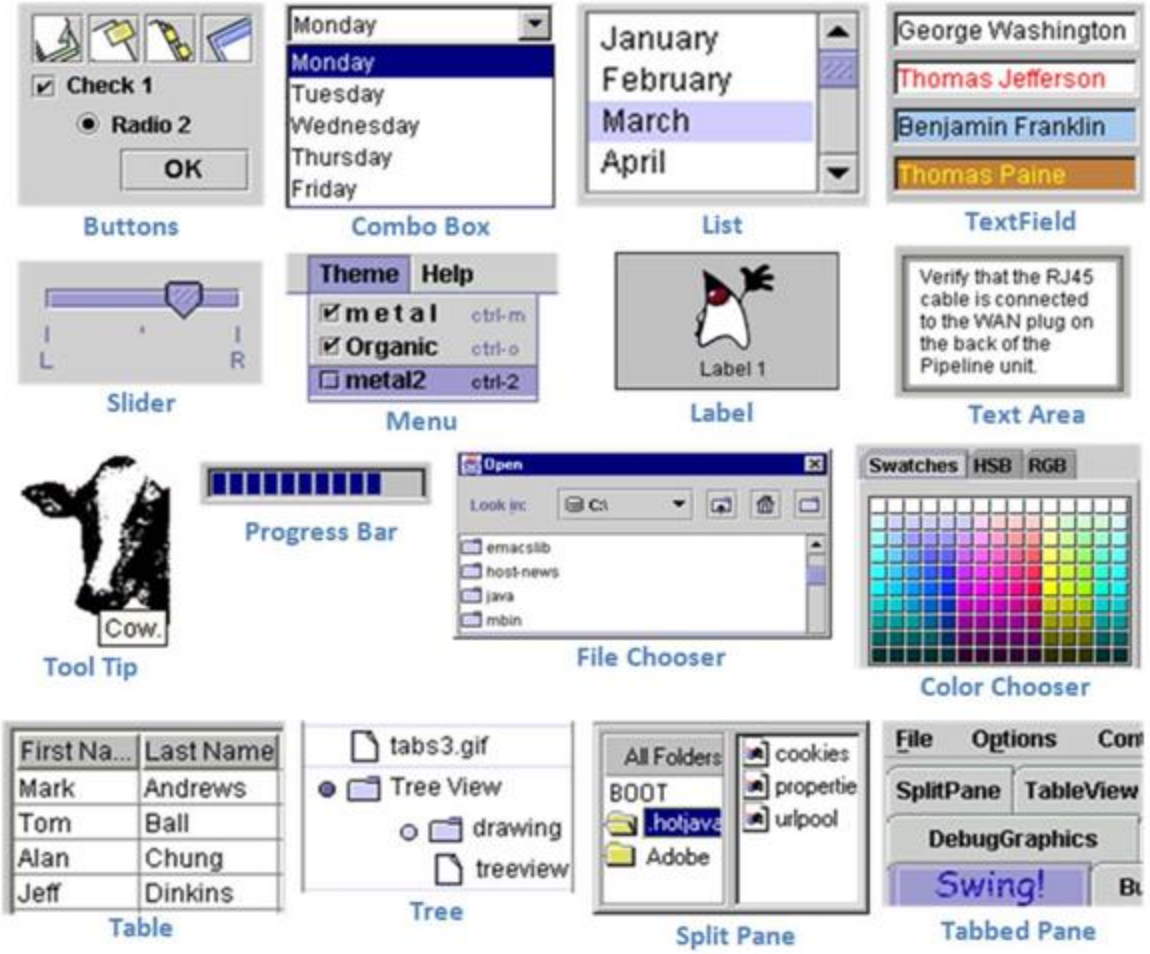
Module contents

Java GUI Programming

- An Introduction to Swing
- Swing - Controls Basics
- Event Handling
- Layout Managers
- Components

Swing – Controls

- Swing provides a rich set of advanced controls

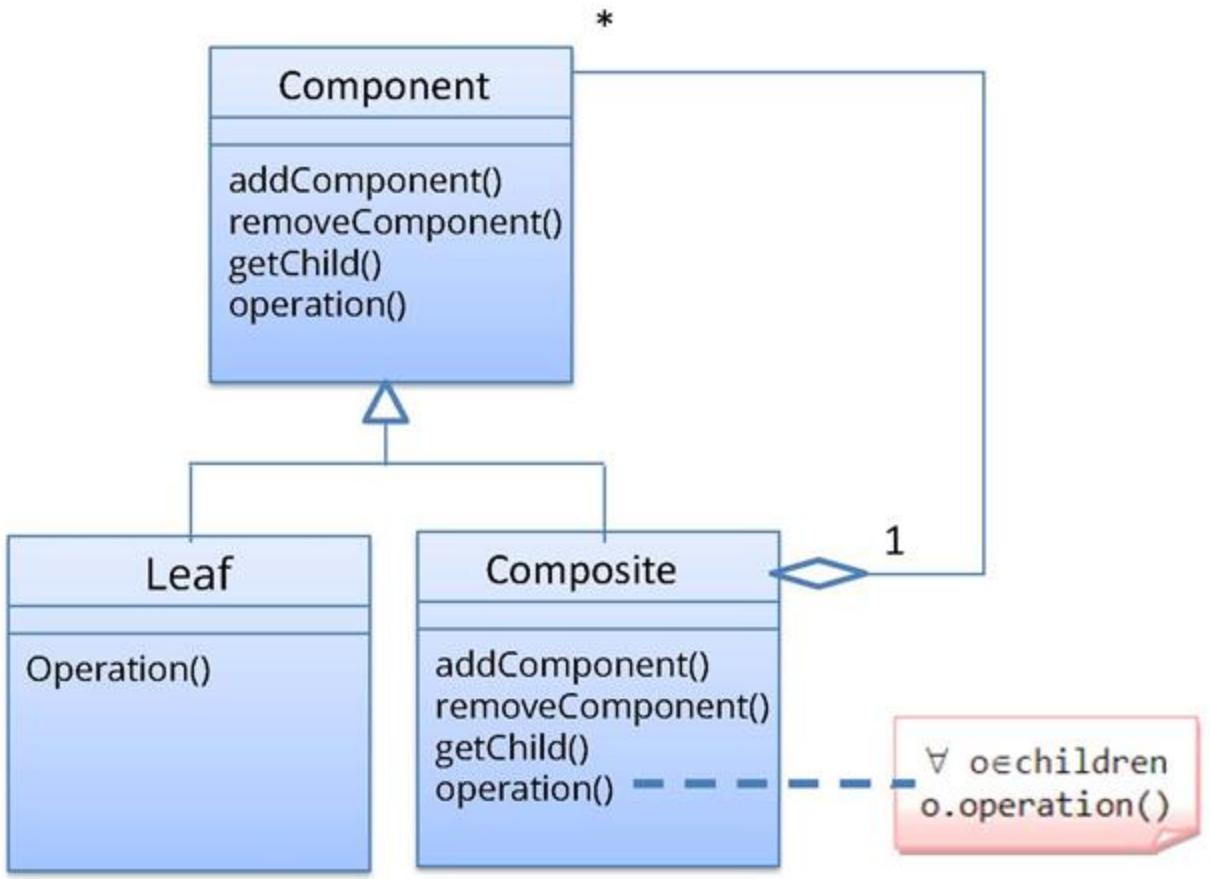


Swing - Composite pattern

- **Composite** is a structural design pattern that lets you compose objects into tree structures and then work with these structures as if they were individual objects.
- Swing uses this pattern to group Component objects with Container and represent both Container and Component instances as leaves of the same tree.

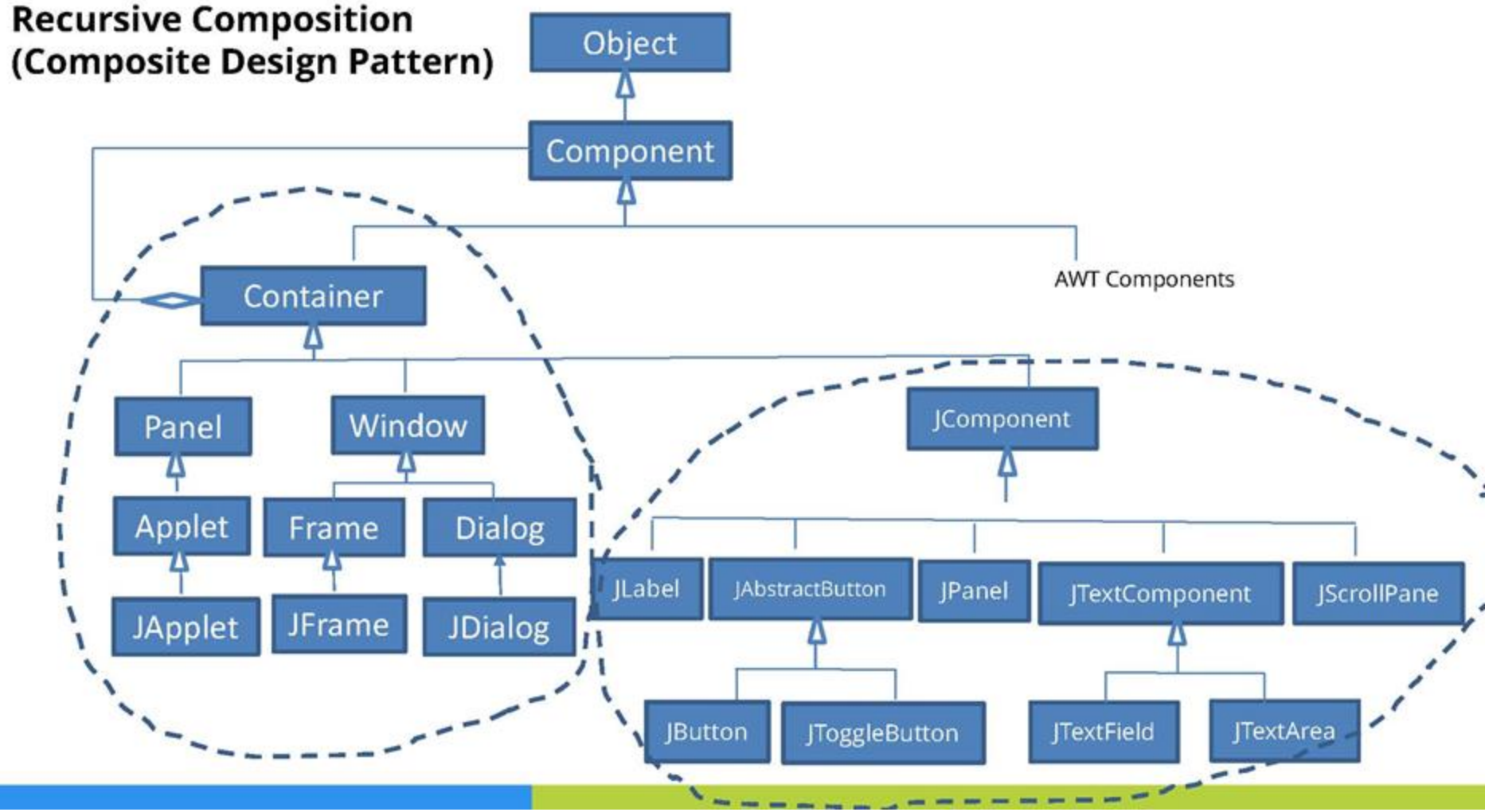
Swing – Controls

- *Composite pattern*



See compositepattern

Swing – Controls



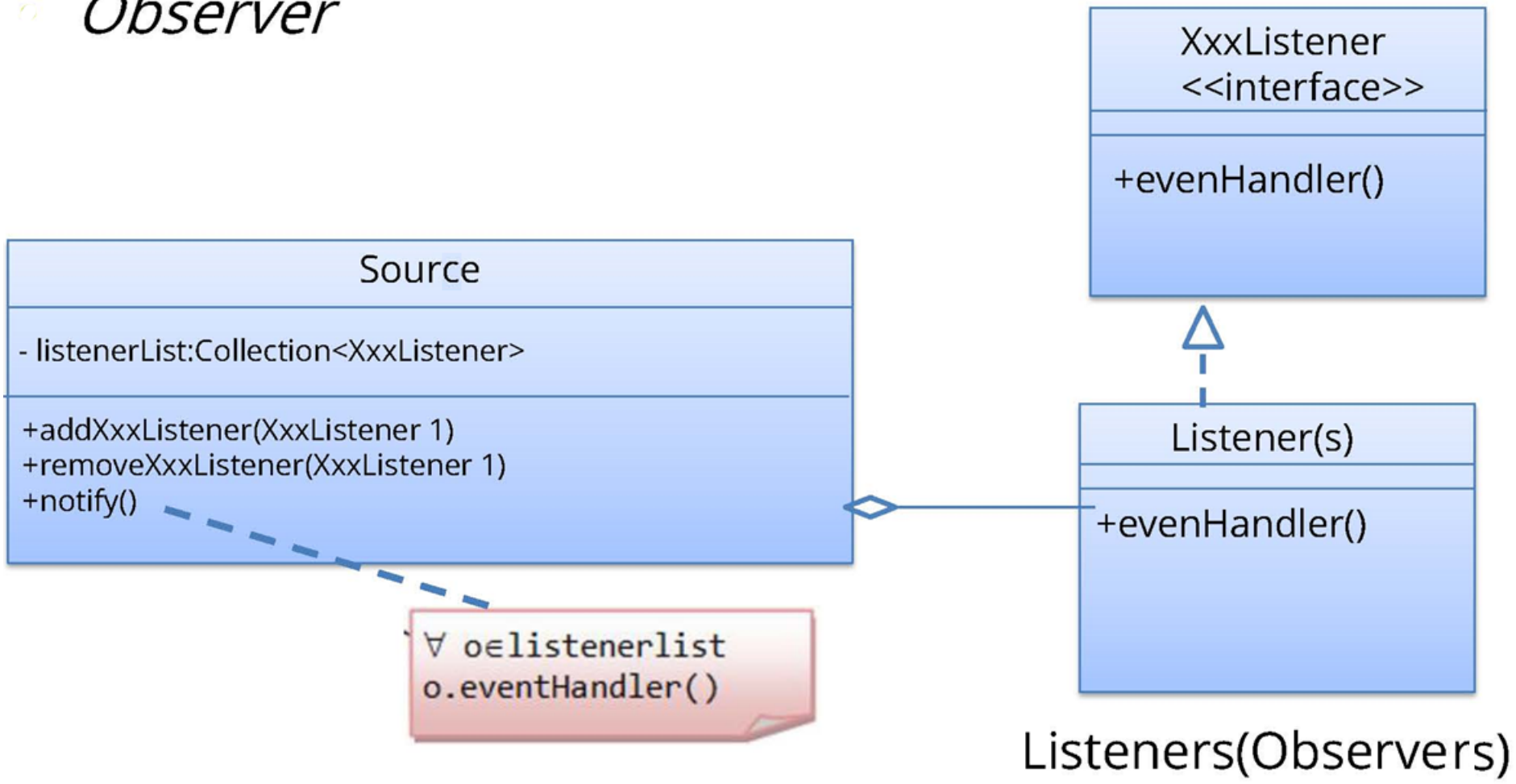
Module contents

Java GUI Programming

- An Introduction to Swing
- Swing - Controls Basics
- Event Handling
- Layout Managers
- Components

Listener (observer) pattern

- *Observer*



See listenerpattern

Event Handling

```
1. JButton btn = new JButton("Click me!");
2. btn.addActionListener(new ActionListener() {
3.     @Override
4.     public void actionPerformed(ActionEvent e) {
5.         System.out.println("Button Click!");
6.     }
7. });
```

See basic\TwoButtonsFrameDemo

See basic\TwoButtonsFrameListenerDemo

Events & Listeners

Event class	Description	Listener interface
java.awt.event		
ActionEvent	A semantic event which indicates that a component-defined action occurred.	ActionListener
AdjustmentEvent	The adjustment event emitted by Adjustable objects like Scrollbar and ScrollPane.	AdjustmentListener
FocusEvent	A low-level event which indicates that a Component has gained or lost the input focus.	FocusListener
ItemEvent	A semantic event which indicates that an item was selected or deselected.	ItemListener
KeyEvent	An event which indicates that a keystroke occurred in a component.	KeyListener
MouseEvent	An event which indicates that a mouse action occurred in a component.	MouseListener и MouseMotionListener

Events & Listeners

Event class	Description	Listener interface
	java.awt.event	
MouseWheelEvent	An event which indicates that the mouse wheel was rotated in a component.	MouseWheelListener
WindowEvent	A low-level event that indicates that a window has changed its status.	WindowListener
	javax.swing.event	
AncestorEvent	An event reported to a child component that originated from an ancestor in the component hierarchy.	AncestorListener
CaretEvent	It is used to notify interested parties that the text caret has changed in the event source.	CaretListener
ChangeEvent	It is used to notify interested parties that state has changed in the event source.	ChangeListener
HyperlinkEvent	It is used to notify interested parties that something has happened with respect to a hypertext link.	HyperlinkListener
ListDataEvent	Defines an event that encapsulates changes to a list.	ListDataListener

Events & Listeners

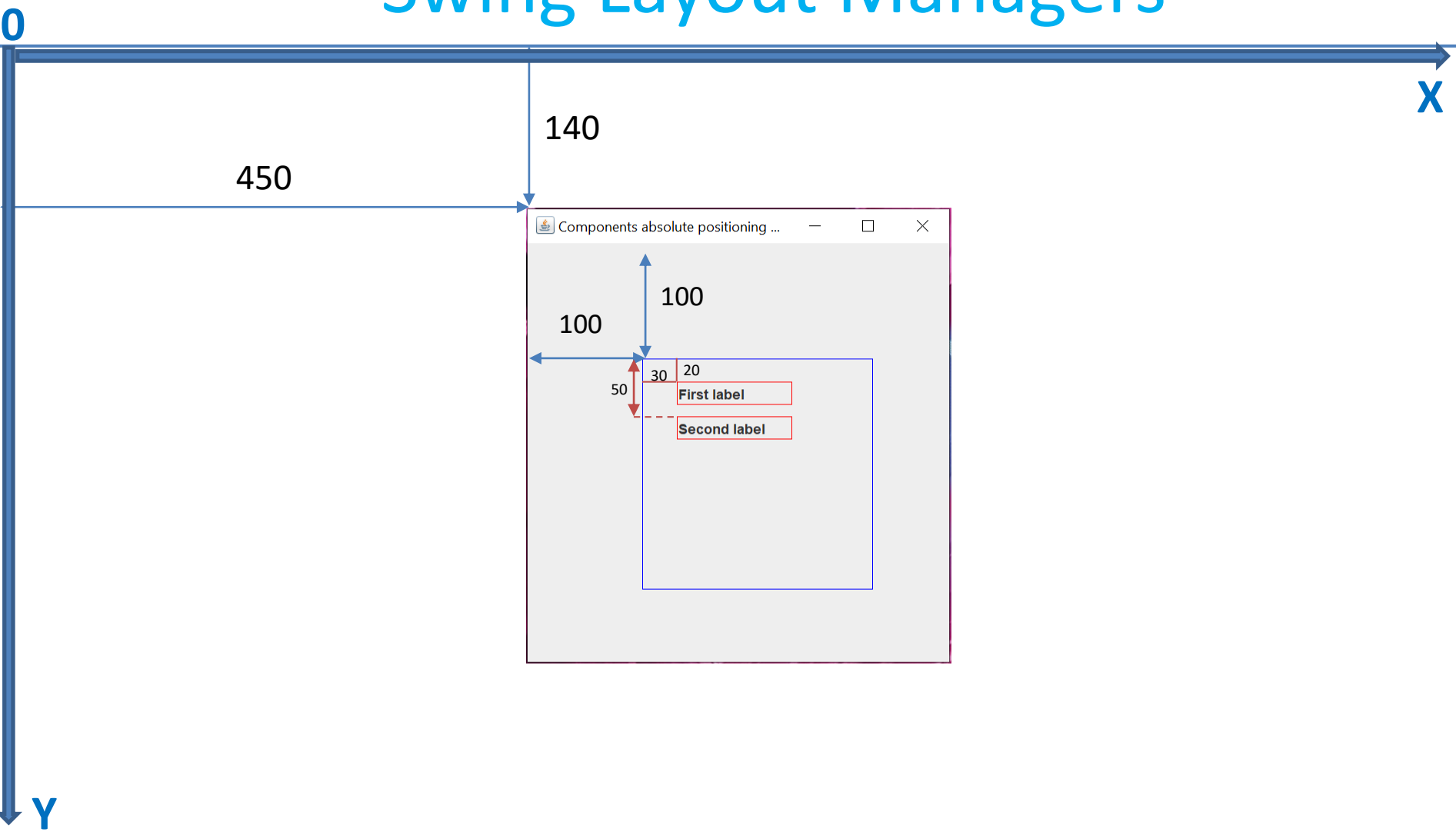
Event class	Description	Listener interface
javax.swing.event		
ListSelectionEvent	An event that characterizes a change in selection.	ListSelectionListener
MenuEvent	It is used to notify interested parties that the menu which is the event source has been posted, selected, or canceled.	MenuListener
TableModelEvent	It is used to notify listeners that a table model has changed.	TableModelListener
TreeExpansionEvent	An event used to identify a single path in a tree.	TreeExpansionListener
TreeModelEvent	Encapsulates information describing changes to a tree model, and used to notify tree model listeners of the change.	TreeModelListener
TreeSelectionEvent	An event that characterizes a change in the current selection.	TreeSelectionListener

Module contents

Java GUI Programming

- An Introduction to Swing
- Swing - Controls Basics
- Event Handling
- Layout Managers
- Components

Swing Layout Managers



See basic\ComponentsPositioningDemo

Swing Layout Managers

```
1. public static void main(String[] args){
2.     JFrame jfrm = new  JFrame("My Program");
3.     jfrm.setSize (300, 200);
4.     jfrm.setLocation (100, 200);
5.     jfrm.setVisible (true);
6.     JButton jbtn = new JButton("КНОПКА");
7.     jbtn.setSize(120,50);
8.     jfrm.add(jbtn);
9. }
```

Swing Layout Managers

- JFrame jfrm = **new** JFrame("**My Program**");
- JPanel jpan = **new** JPanel();
- jpan.setLayout (**new** BorderLayout());
- jpan.add(**new** JButton("**North**"), BorderLayout.***NORTH***);
- jpan.add(**new** JButton("**South**"), BorderLayout.***SOUTH***);
- jpan.add(**new** JButton("**West**"), BorderLayout.***WEST***);
- jpan.add(**new** JButton("**East**"), BorderLayout.***EAST***);
- jpan.add(**new** JButton("**Center**"), BorderLayout.***CENTER***);
- jfrm.add(jpan);
- jfrm.setVisible(**true**);

See layouts\BorderLayoutDemo

Swing Layout Managers

```
JFrame jfrm = new  JFrame("My Program");
JPanel jpan = new JPanel();
jpan.setLayout ( new FlowLayout() );
for (int i=0; i<5; i++){
    jpan.add(new JButton("Button "+ i));
}
jfrm.add(jpan);
jfrm.setVisible(true);
```

See layouts\FlowLayoutDemo

Swing Layout Managers

```
JFrame jfrm = new  JFrame("My Program");
JPanel jpan = new JPanel();
jpan.setLayout (
        new BoxLayout(jpan, BoxLayout.Y_AXIS));
for (int i=0; i<5; i++){
    jpan.add(new JButton("Button "+ i));
}
jfrm.add(jpan);
jfrm.setVisible(true);
```

See layouts\BoxLayoutDemo

Swing Layout Managers

```
JFrame jfrm = new  JFrame("My Program");
JPanel jpan = new JPanel();
jpan.setLayout ( new GridLayout(4, 3, 10, 10));
for (int i=0; i<12; i++) {
    jpan.add(new JButton("Button "+ i));
}
jfrm.add(jpan);
jfrm.setVisible(true);
```

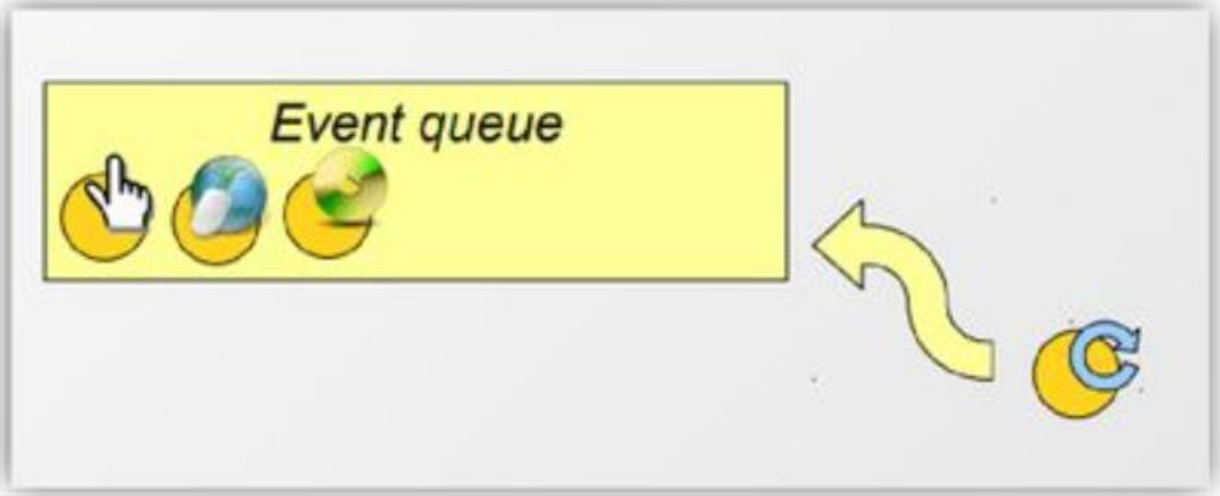
See layouts\GridLayoutDemo

Swing Layout Managers

- JFrame jfrm = **new** JFrame(**"My Program"**);
- JPanel jpan = **new** JPanel();
- JButton btn = **new** JButton(**"Size 180x50"**);
- btn.setPreferredSize(**new** Dimension(180,50));
- jpan.add(btn);
- jfrm.add(jpan);
- jfrm.setVisible(**true**);

See layouts\CompositeLayoutDemo

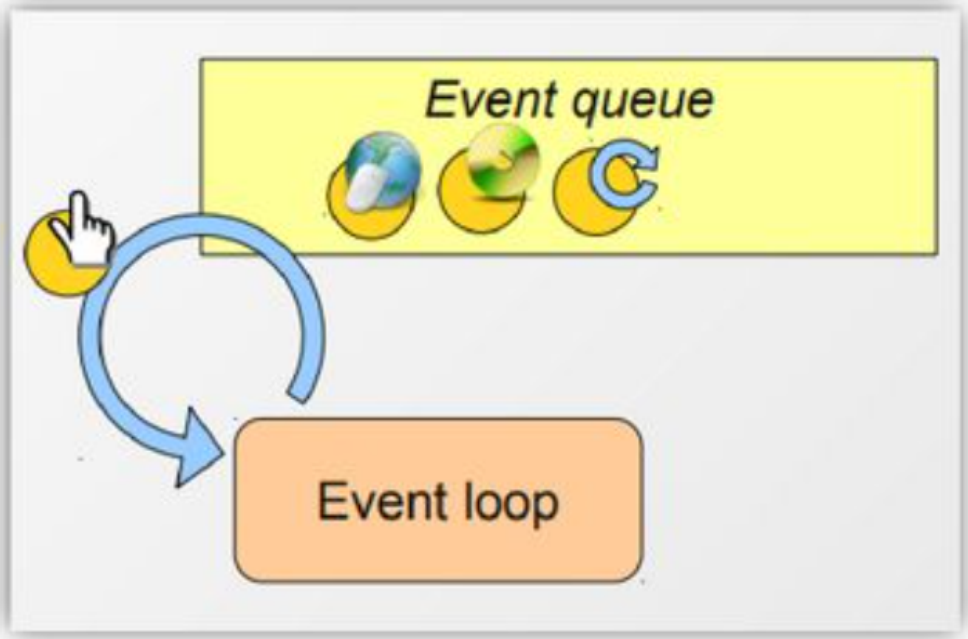
Responsive GUI



Event: Something of interest has happened. For example a MouseEvent (Click, Move etc.)

Event Queue: Each event is added to the Event Queue to be processed. These will be processed in First In First Out order like a normal queue operates.

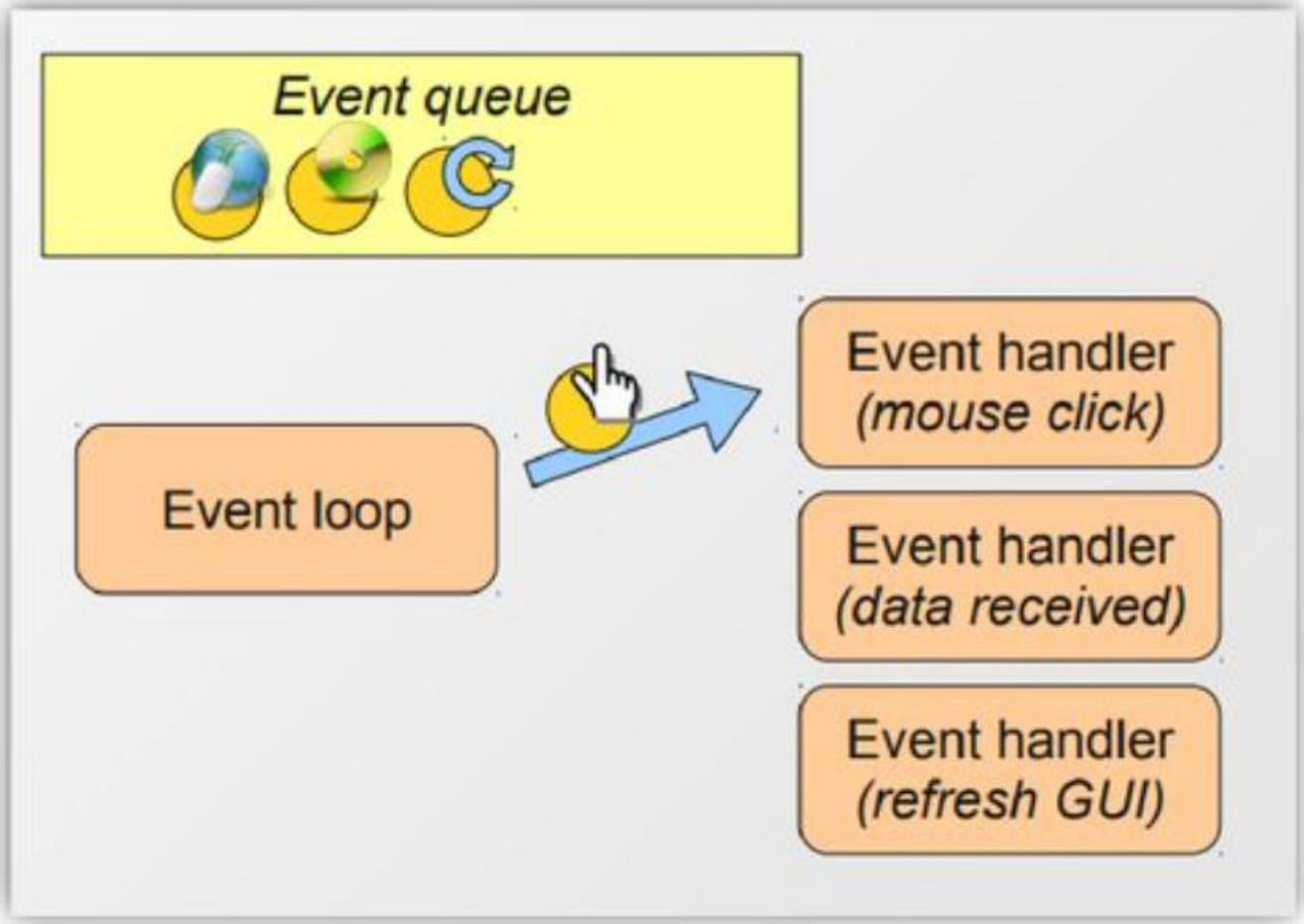
Responsive GUI



Event Loop: The events that are passed into the event queue are processed by a continuously running Event Loop. The event at the front of the queue will be removed and executed by the Event Loop.

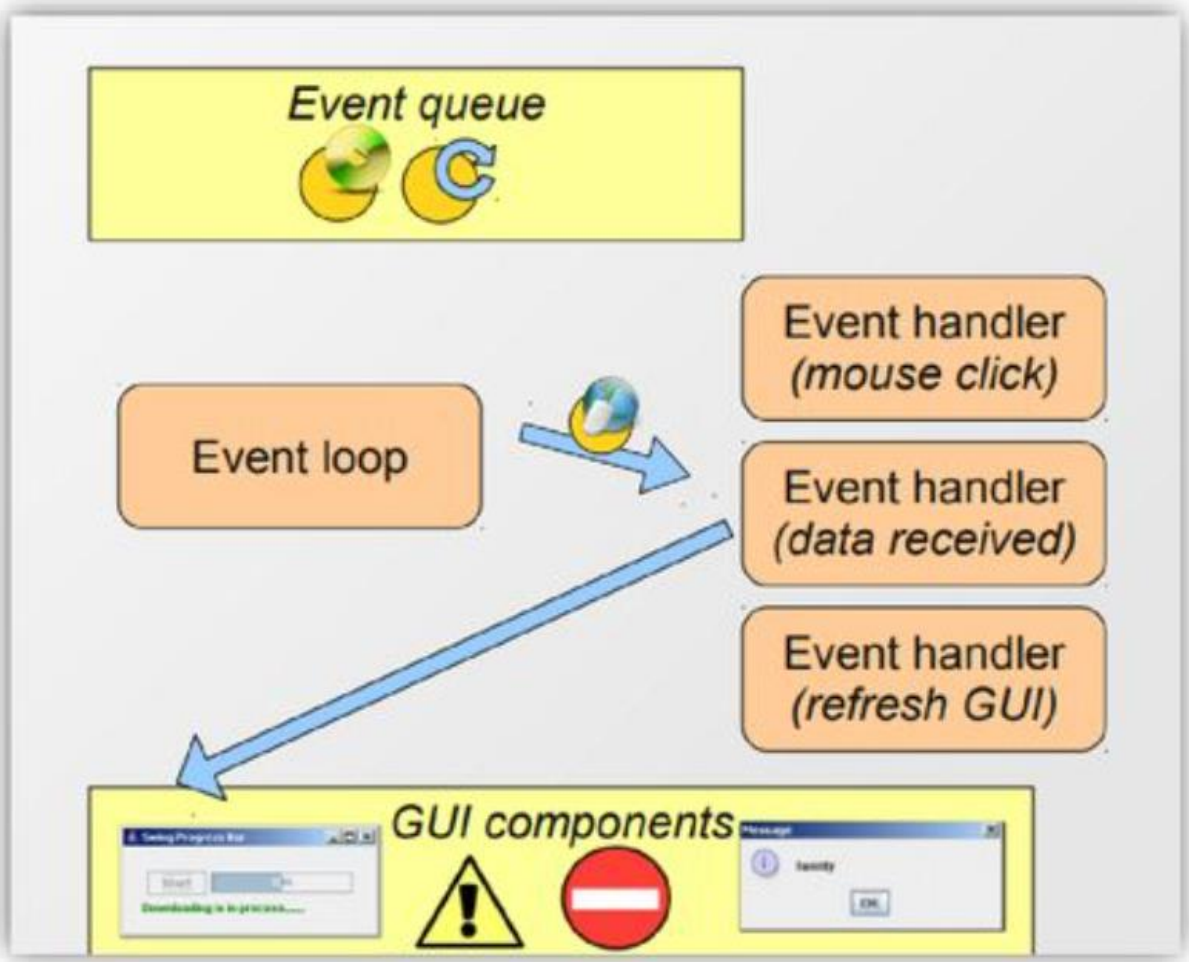
This execution of logic is performed by the Event triggering its relevant Event Handler

Responsive GUI



Event Handler: The different Event types will execute certain tasks depending on how their Event Handler is implemented.

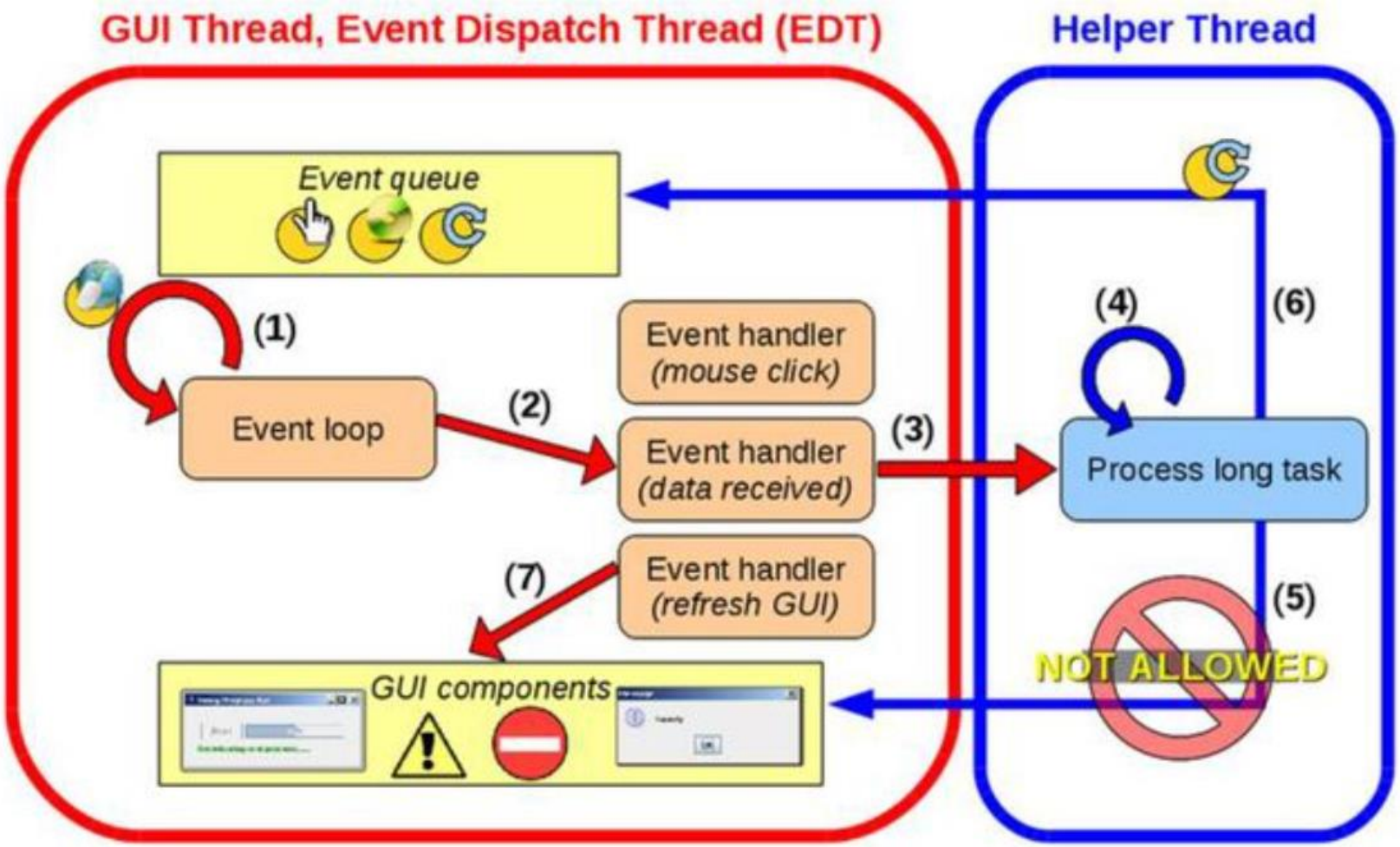
Responsive GUI



What's wrong with this design structure in the above figure?

Using a single thread - see multithreading\LongRunningEvent

Responsive GUI



Using a helper thread -
see multithreading\LongRunningEventThreading

Responsive GUI

- **SwingWorker** is an abstract class used to perform lengthy GUI interaction tasks in a background thread.
- Java language has **three threads**, namely listed below as follows:
  1. **Current Thread** (Initial Thread): this is the thread on which the initial application logic executes.
  2. **Event Dispatch Thread**: all event handling code executes on this thread.
  3. **Worker Threads**: also known as background threads where all time-consuming background tasks are executed.
- SwingWorker allows users to schedule the execution of background tasks on Worker Thread.
- It is necessary to run the Swing GUI in the Event Dispatch Thread (EDT) because the Swing components are not thread-safe and all Swing based code should be accessed/modified/interacted with from the context of the EDT.

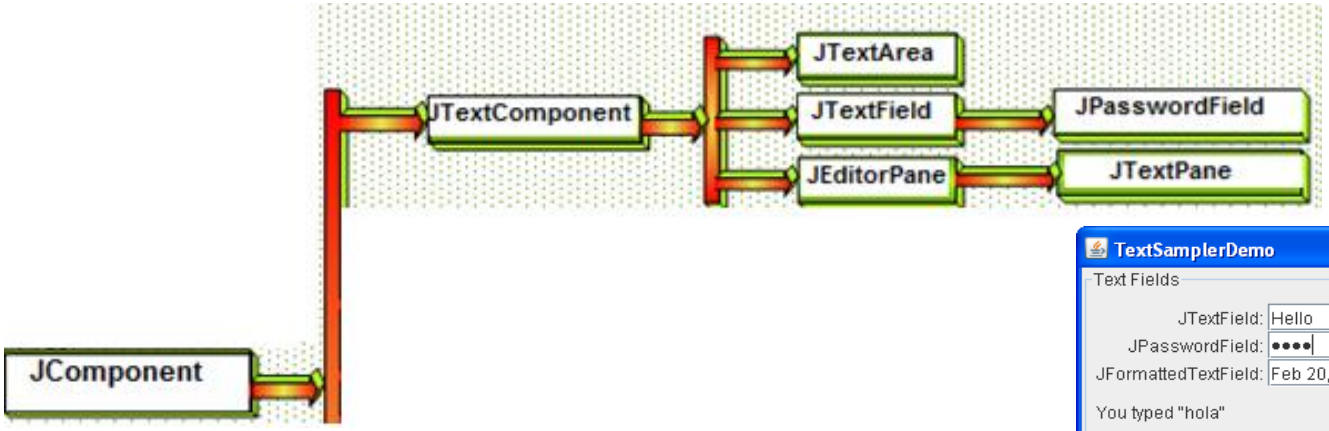
see multithreading\LongRunningEventBySwingWorker

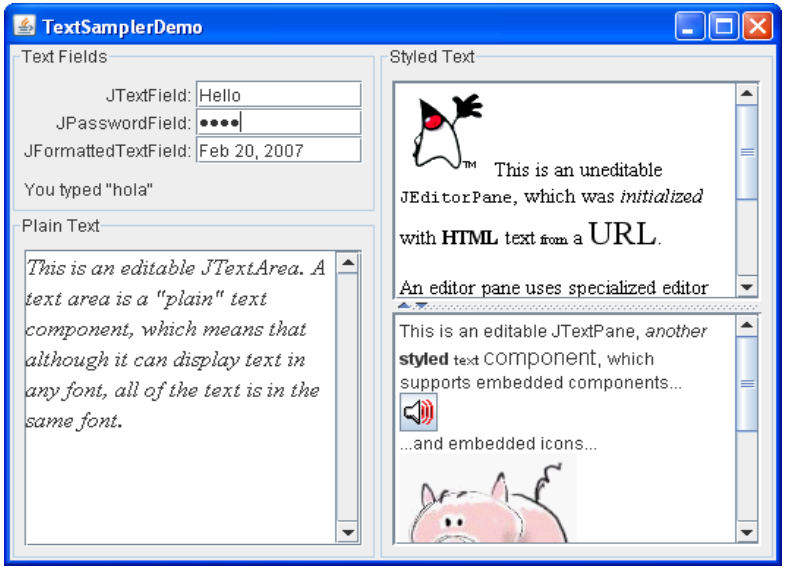
Module contents

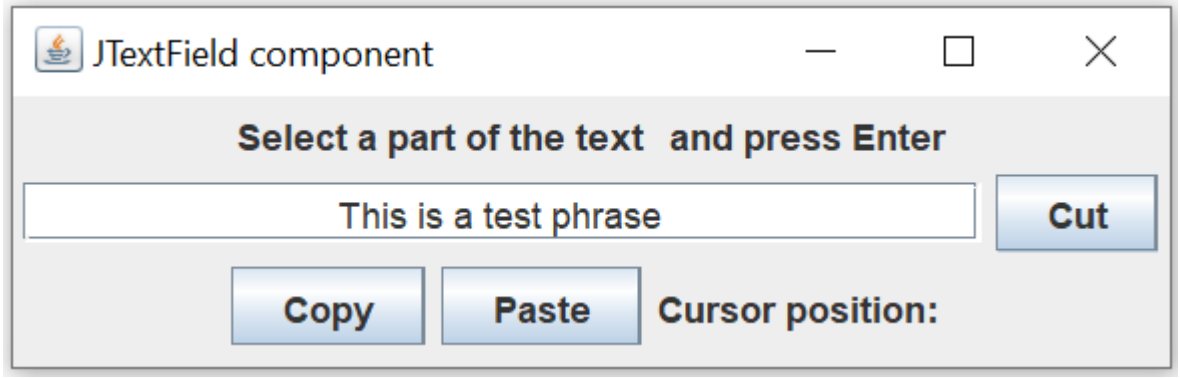
Java GUI Programming

- An Introduction to Swing
- Swing - Controls Basics
- Event Handling
- Layout Managers
- Components

JTextField

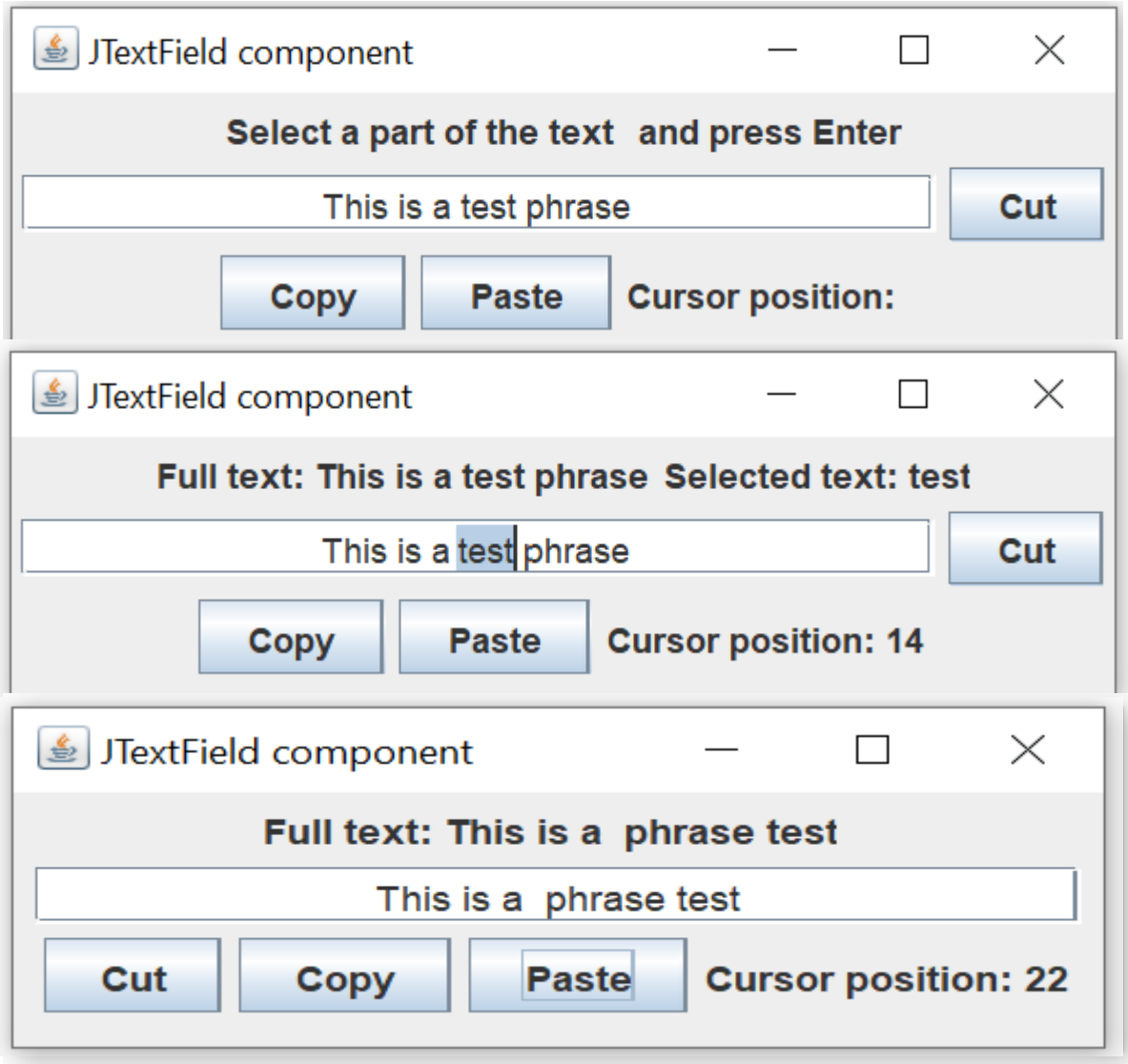






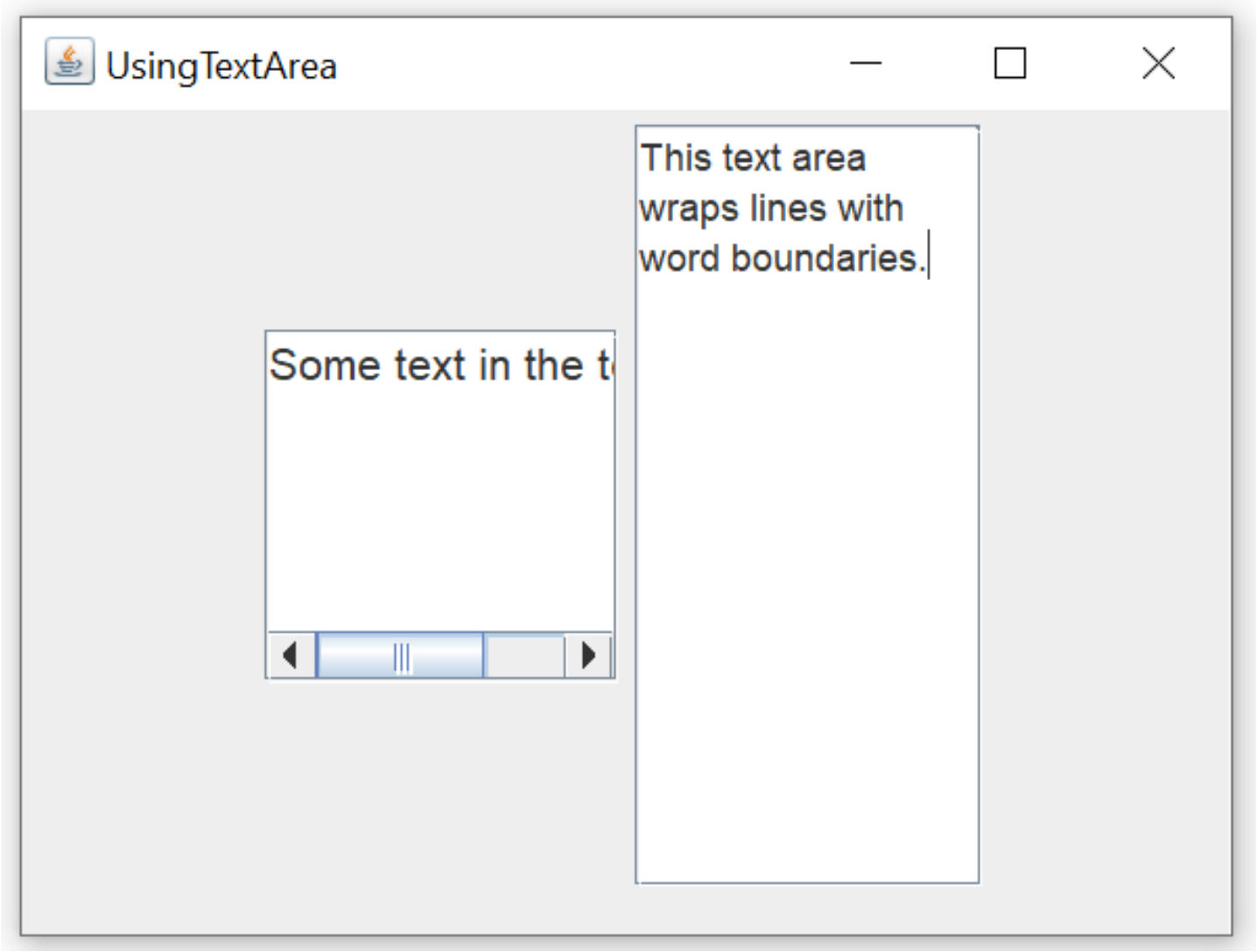
See components\TextFieldDemo

JTextField



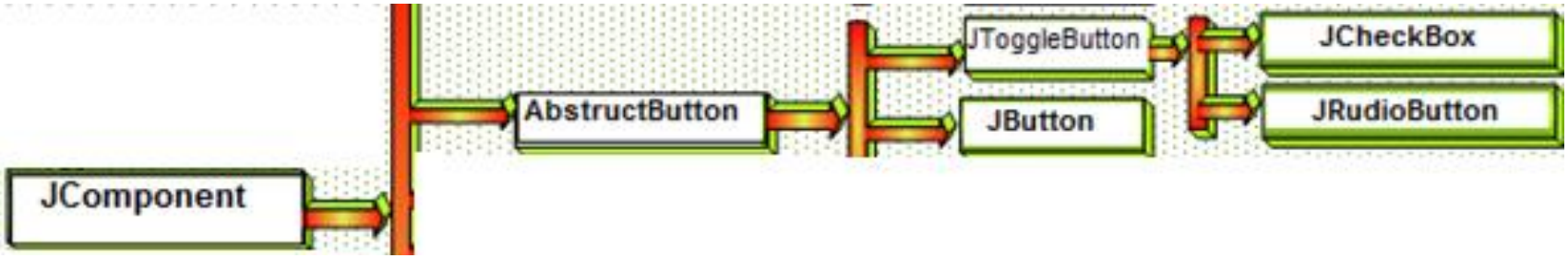
See components\TextFieldDemo

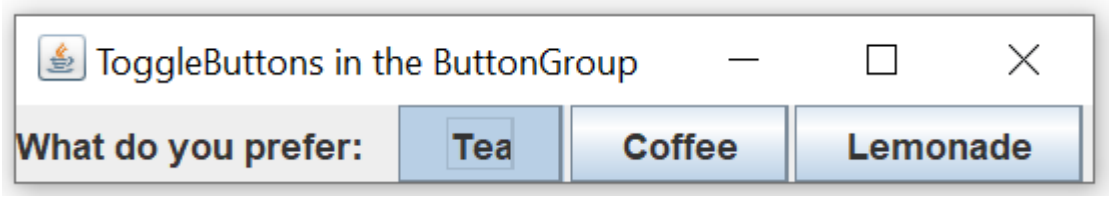
JTextArea



See components\TextAreaDemo

JToggleButton



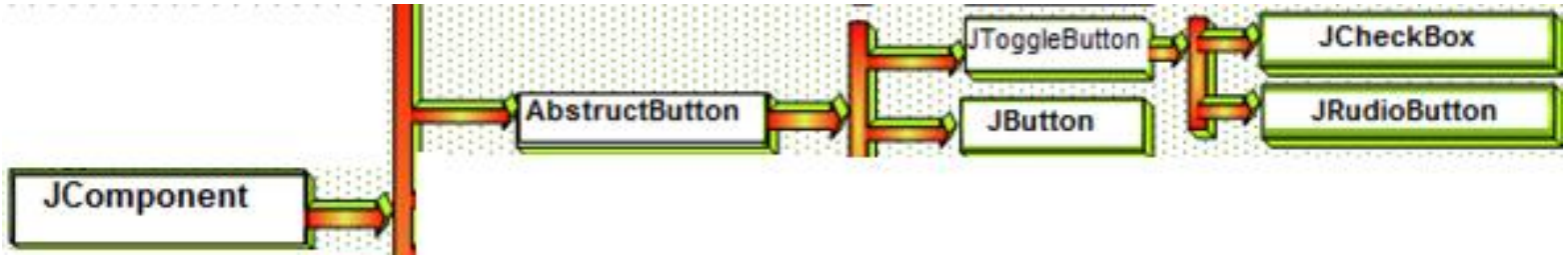


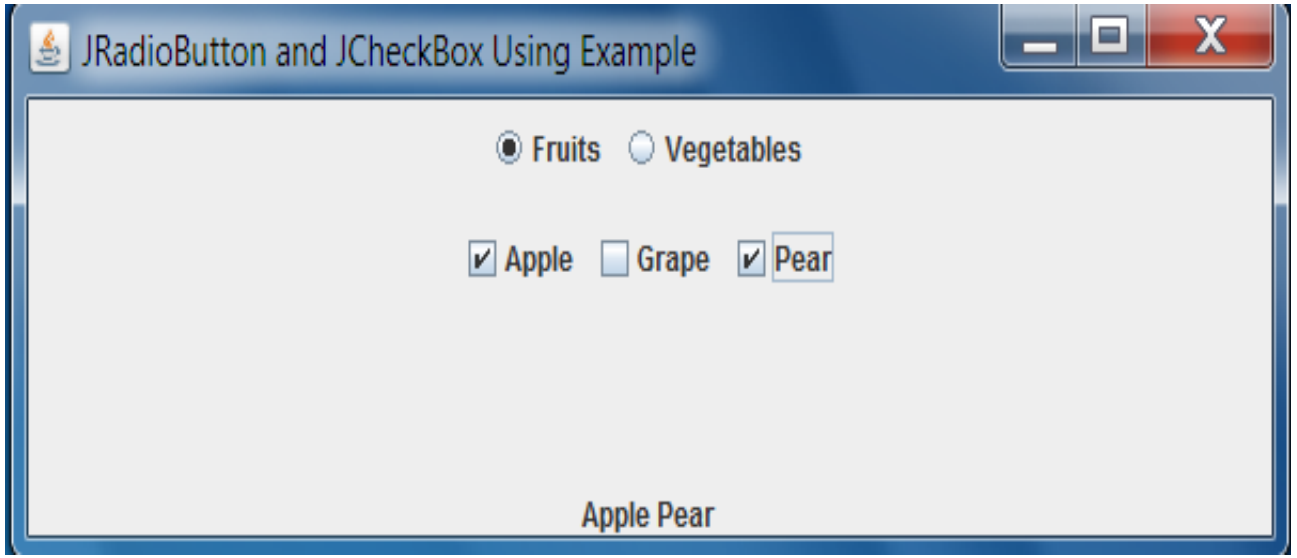
Output:
You like Tea

- JToggleButton is the button with 2 states.
- Several JToggleButtons are usually combined into a javax.swing.ButtonGroup.

See components\ToggleButtonDemo

JCheckBox and JRadioButton

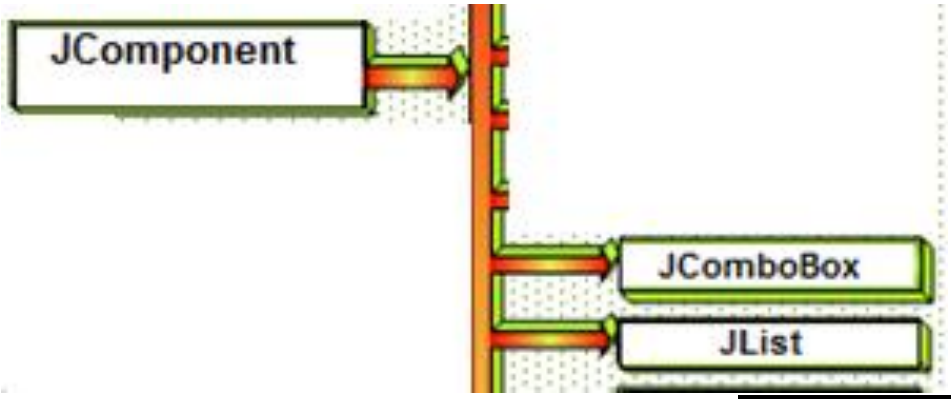


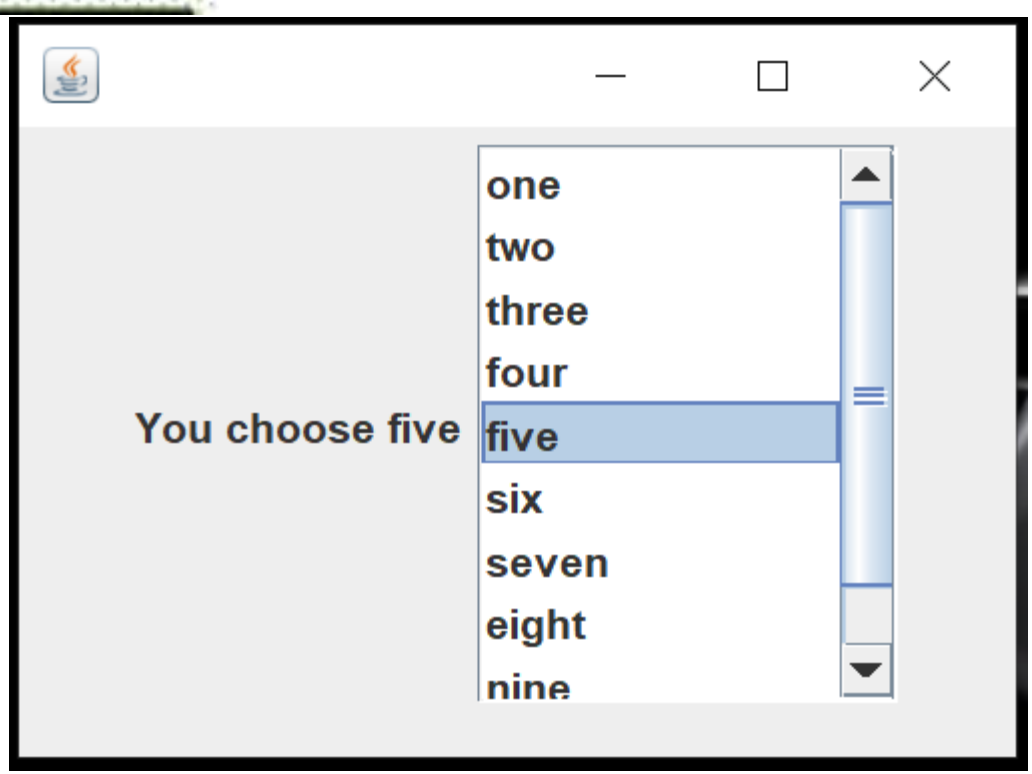


- Several JCheckBoxes or JRadioButtons are usually combined into a javax.swing.ButtonGroup

See components\CheckBoxRadioButtonDemo

JList





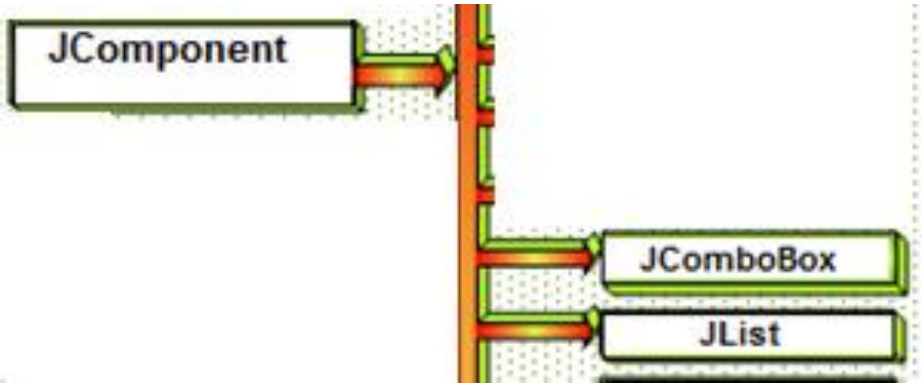
See components\ListDemo

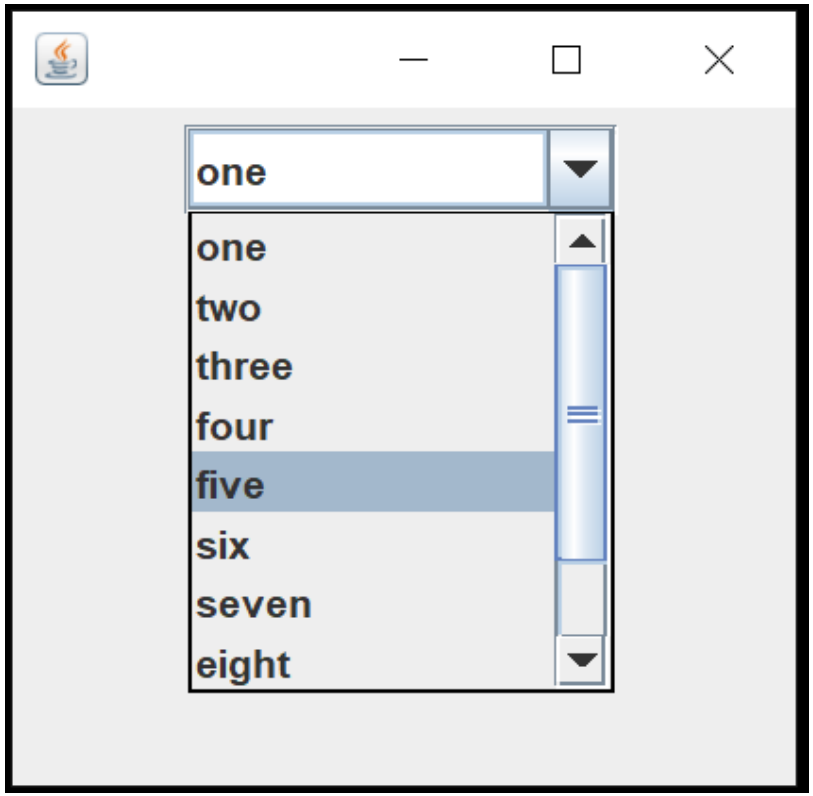
JList

- The **javax.swing.ListModel<E>** interface defines the methods to get the value of each list item and the length of the list.
- Logically the model is a vector, indices vary from 0 to ListDataModel.getSize() - 1.
- Any change to the contents or length of the data model must be reported to all of the **ListDataListeners**.
- The DefaultListModel<E> class implements ListModel<E> interface, has
  private Vector<E> delegate
  field and provides Vector's methods for getting, adding, changing, removing and searching list items.

See components\ListModelDemo

JComboBox





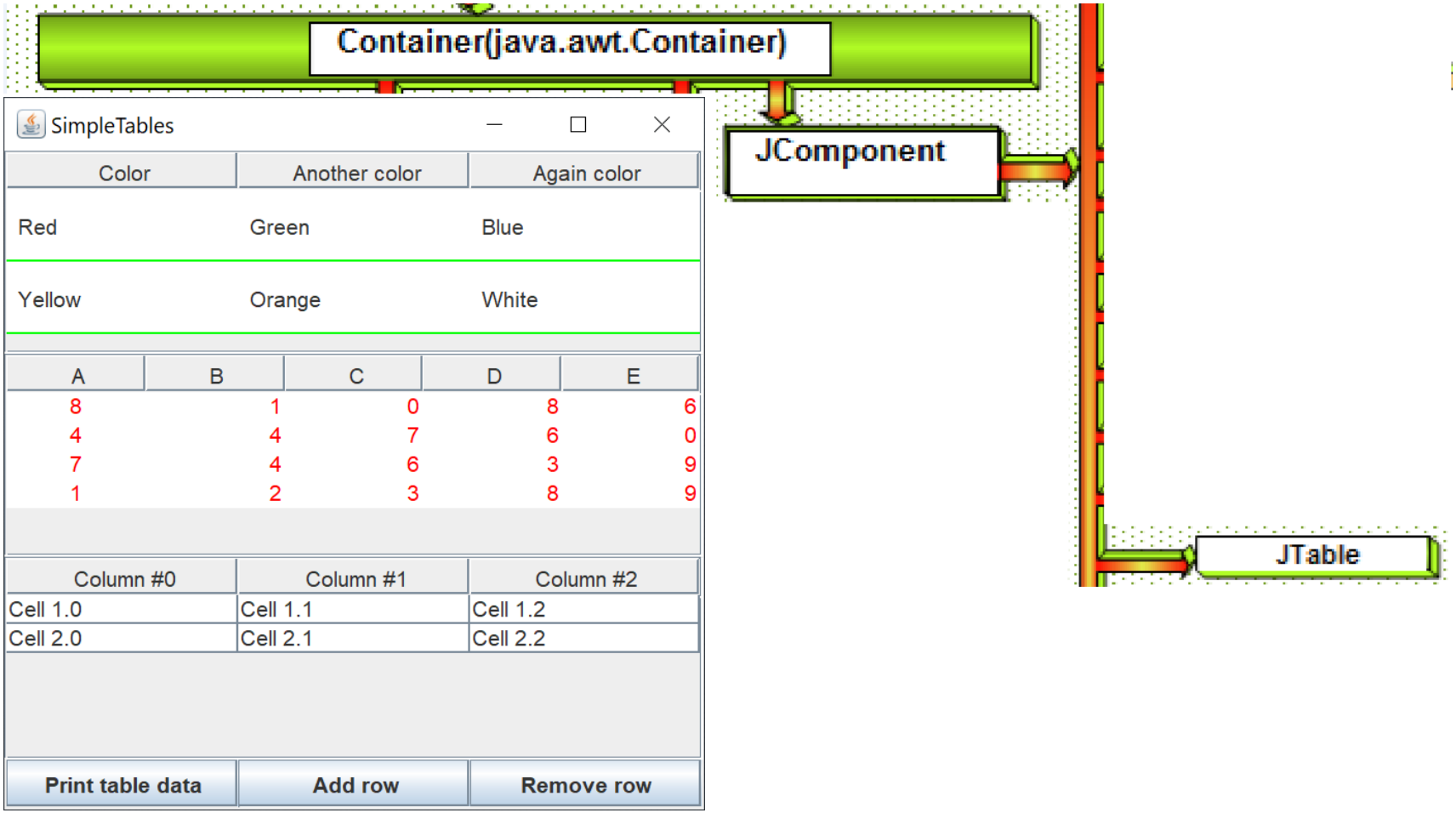
- There is Editable mode for ComboBox that allows the user to type into the field or to edit selected item.
- The DefaultComboBoxModel<E> class implements ListModel<E> interface and provides methods for getting, adding, changing, removing and searching list items.

See components\ComboBoxDemo

ImageIcon

- Many Swing components, such as labels, buttons and tabbed panes, can be decorated with an ***icon*** — a fixed-sized picture.
- An ***icon*** is an object that implements **javax.swing.Icon** interface.
- Swing provides implementation of the **Icon** interface: **javax.swing.ImageIcon**, which paints an icon from a GIF, JPEG, or PNG image.
- **ImageIcon** has constructors that receive as parameters byte array or image file name or image URL or java.awt.Image instance.
- **ImageIcon** class does not extend Component class, so we have to wrap it with some Component, such as labels, buttons, and tabbed panes with sizes equal or more than image size. See components\ImageIconDemo

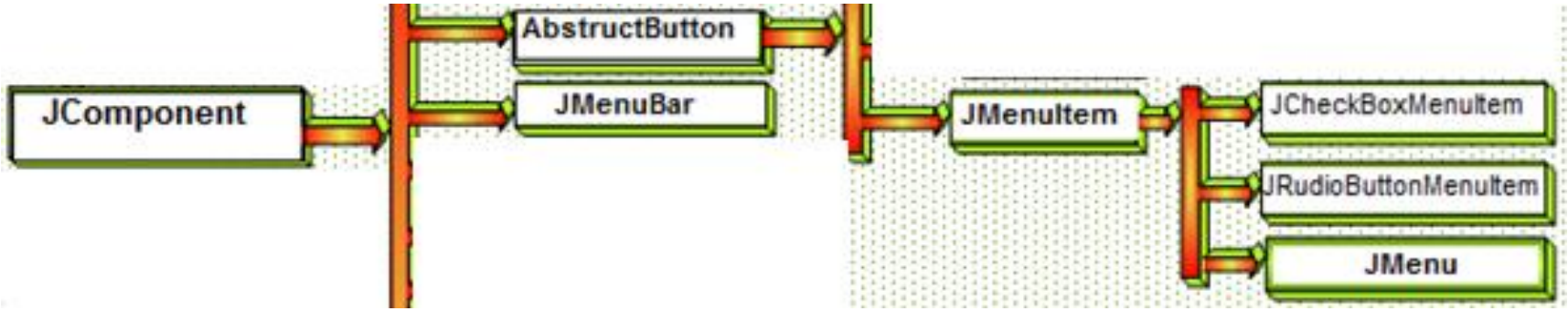
JTable

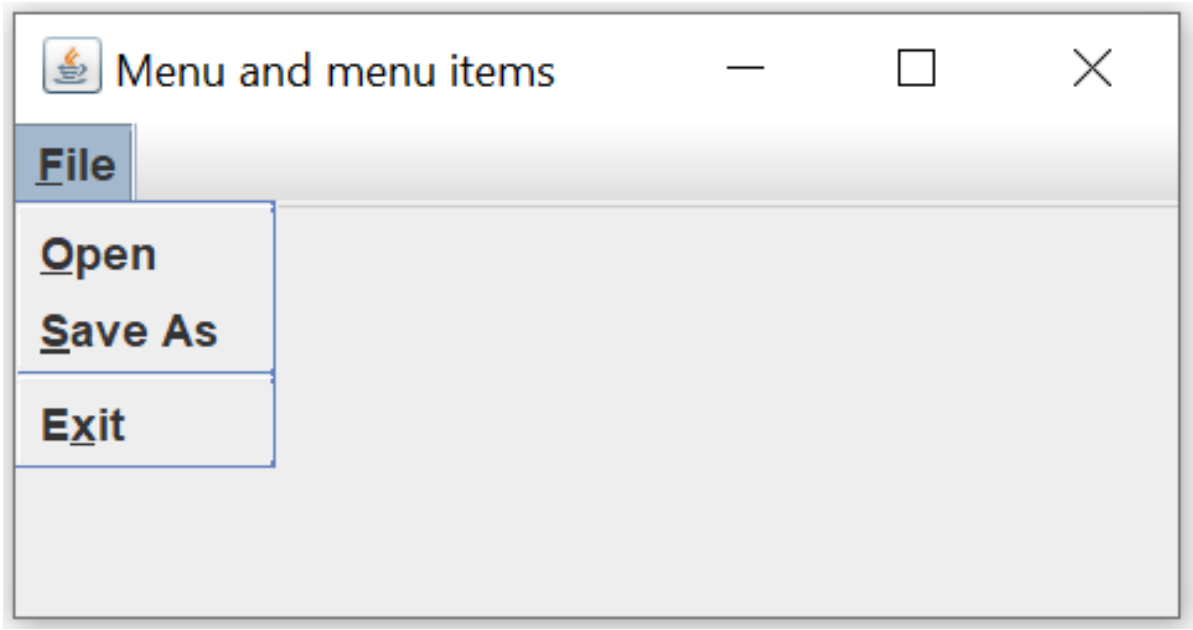


- The DefaultTableModel<E> class implements TableModel<E> interface and provides column and row add and remove and cell values set and get methods.

See components\TableDemo

JMenuBar, JMenu & JMenuItem

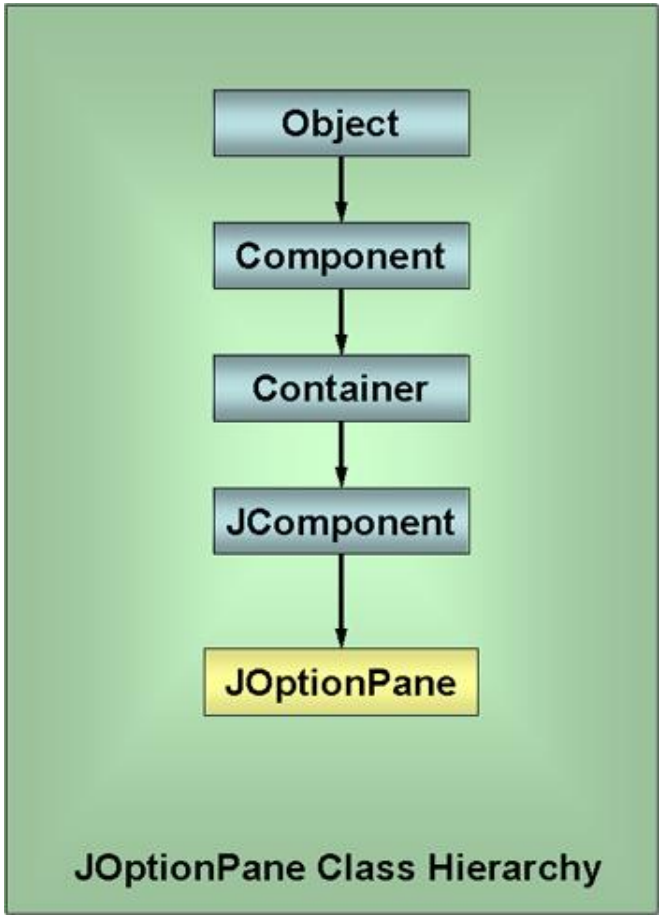




- We will use ***FileChooser*** component for this example.

See components\MenuDemo

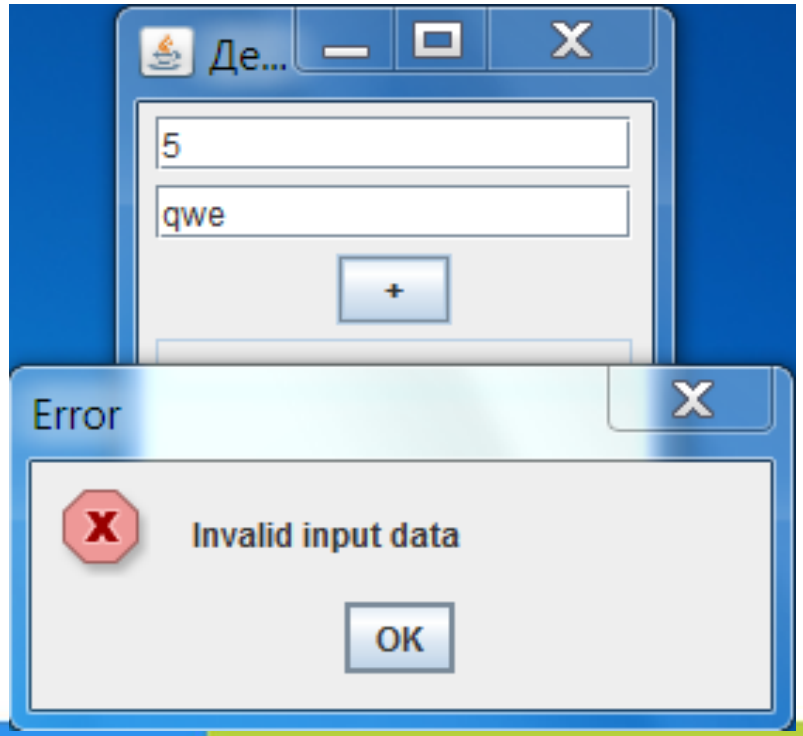
JOptionPane



- The **JOptionPane** class is used to provide standard dialog boxes such as message dialog box, confirm dialog box and input dialog box.

JOptionPane

```
public void actionPerformed(ActionEvent e) {
      try {
         double num1 = Double.parseDouble(tfNum1.getText());
         double num2 = Double.parseDouble(tfNum2.getText());
         tfRes.setText(String.valueOf(num1 + num2));
      } catch (NumberFormatException ex) {
         JOptionPane.showMessageDialog(this, "Invalid input data",
                        "Error", JOptionPane.ERROR_MESSAGE);
      }
   }
```



JOptionPane

- `void` **`showMessageDialog`**`(Component parentComponent, Object message, String title,` **`int messageType`**`,` **`Icon icon`**`)`
- `int` **`showConfirmDialog`**`(Component parentComponent, Object message, String title,` **`int optionType`**`,` **`int messageType`**`,` **`Icon icon`**`)`
- `Object` **`showInputDialog`**`(Component parentComponent, Object message, String title,` **`int messageType`**`,` **`Icon icon`**`, Object[] selectionValues, Object initialSelectionValue)`
- `int` **`showOptionDialog`**`(Component parentComponent, Object message, String title,` **`int optionType`**`,` **`int messageType`**`,` **`Icon icon`**`, Object[] options, Object initialValue)`

optionType:

JOptionPane.DEFAULT_OPTION
JOptionPane.YES_NO_OPTION
JOptionPane.YES_NO_CANCEL_OPTION
JOptionPane.OK_CANCEL_OPTION

messageType:


- JOptionPane.QUESTION_MESSAGE


- JOptionPane.INFORMATION_MESSAGE

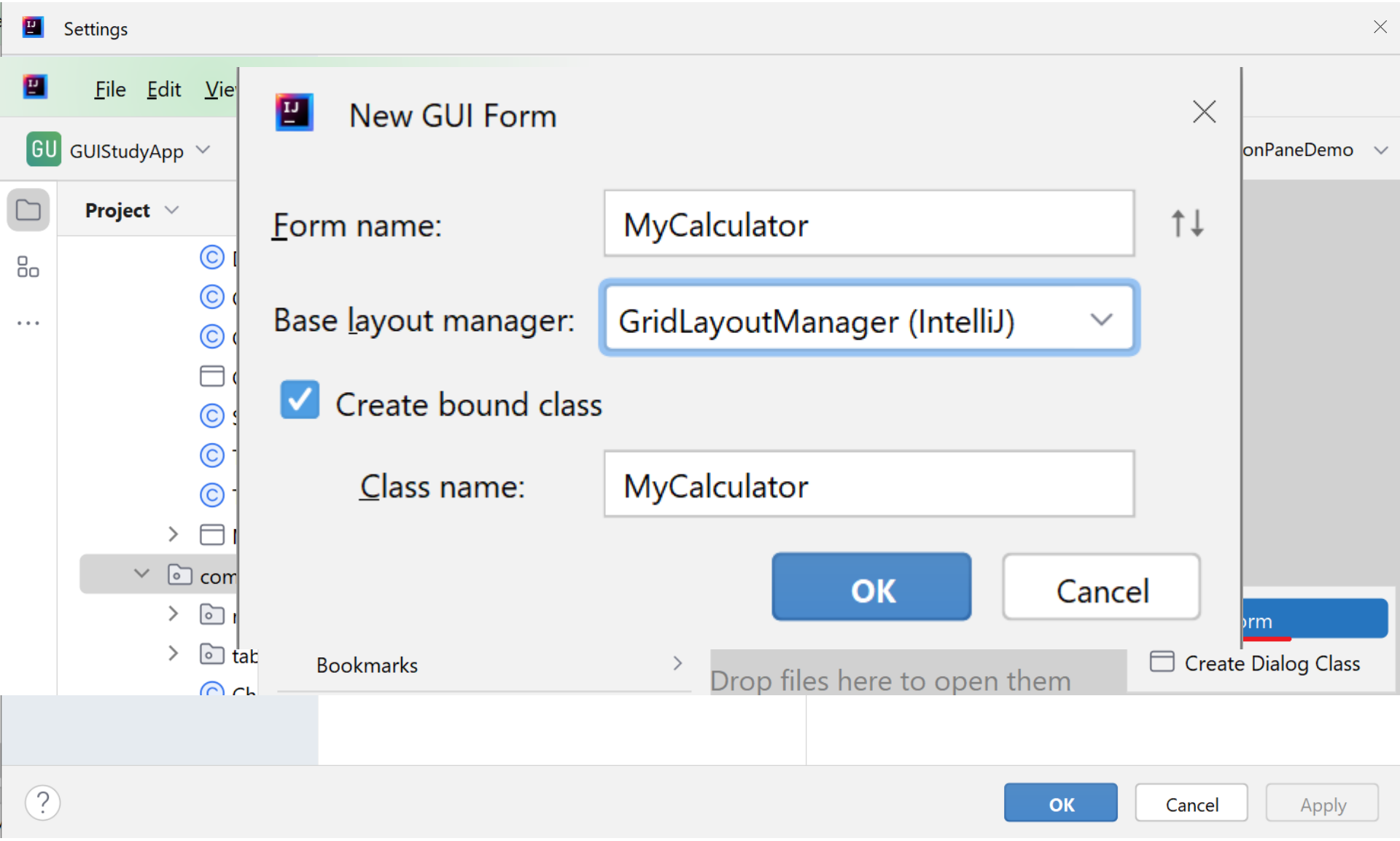

- JOptionPane.WARNING_MESSAGE


- JOptionPane.ERROR_MESSAGE

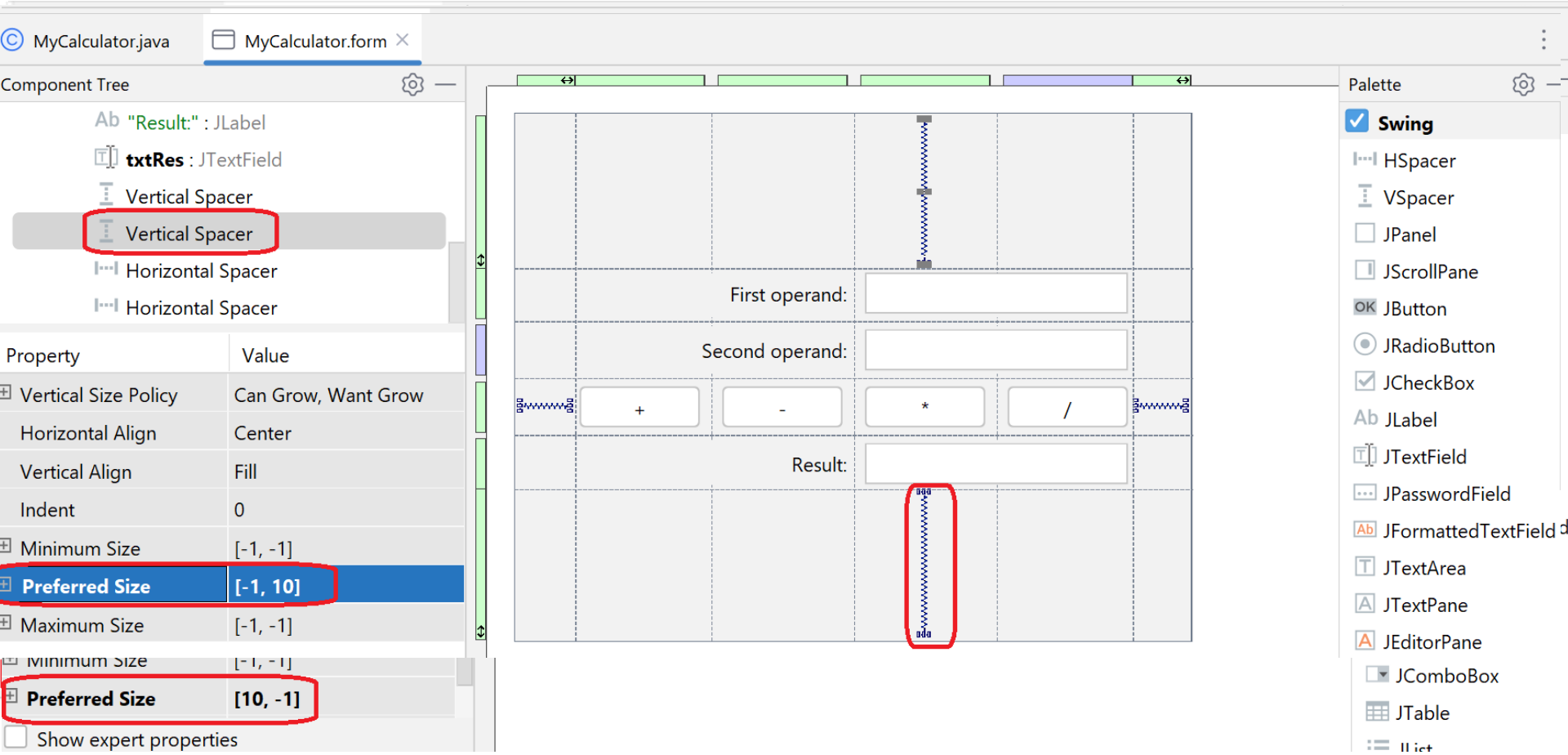
no icon - JOptionPane.PLAIN_MESSAGE

See components\OptionPaneDemo

UI Designer


Settings
File Edit
GUIStudyApp
Project
New GUI Form
Form name:
MyCalculator
Base layout manager:
GridLayoutManager (IntelliJ)
Create bound class
Class name:
MyCalculator
OK
Cancel
Bookmarks
Drop files here to open them
Create Dialog Class
OK
Cancel
Apply

UI Designer


MyCalculator.java
MyCalculator.form
Component Tree
"Result:" : JLabel
txtRes : JTextField
Vertical Spacer
Vertical Spacer
Horizontal Spacer
Horizontal Spacer
Property
Value
Vertical Size Policy
Can Grow, Want Grow
Horizontal Align
Center
Vertical Align
Fill
Indent
0
Minimum Size
[-1, -1]
Preferred Size
[-1, 10]
Maximum Size
[-1, -1]
Preferred Size
[10, -1]
Show expert properties
First operand:
Second operand:
+
-
*
/
Result:
Palette
Swing
HSpacer
VSpacer
JPanel
JScrollPane
JButton
JRadioButton
JCheckBox
JLabel
JTextField
JPasswordField
JFormattedTextField
JTextArea
JTextPane
JEditorPane
JComboBox
JTable

UI Designer

MyCalculator.java | MyCalculator.form

```
6   private JPanel panel1;
    1 usage
7   private JTextField txtOp1;
    1 usage
8   private JTextField txtOp2;
    1 usage
9   private JButton btnAdd;
    1 usage
10  private JButton btnSub;
    1 usage
11  private JButton btnMul;
    1 usage
12  private JButton btnDiv;
    1 usage
13  private JTextField txtRes;
14
15  public static void main(String[] args) {
16      JFrame frame = new JFrame( title: "MyCalculator");
17      frame.setContentPane(new MyCalculator().panel1);
18      frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
19      frame.pack();
20      frame.setVisible(true);
21  }
22  }
```

Create GUI Form MyCalculator in Swing UI Designer and see MyCalculator.form

UI Designer