Object Oriented Programming with Java

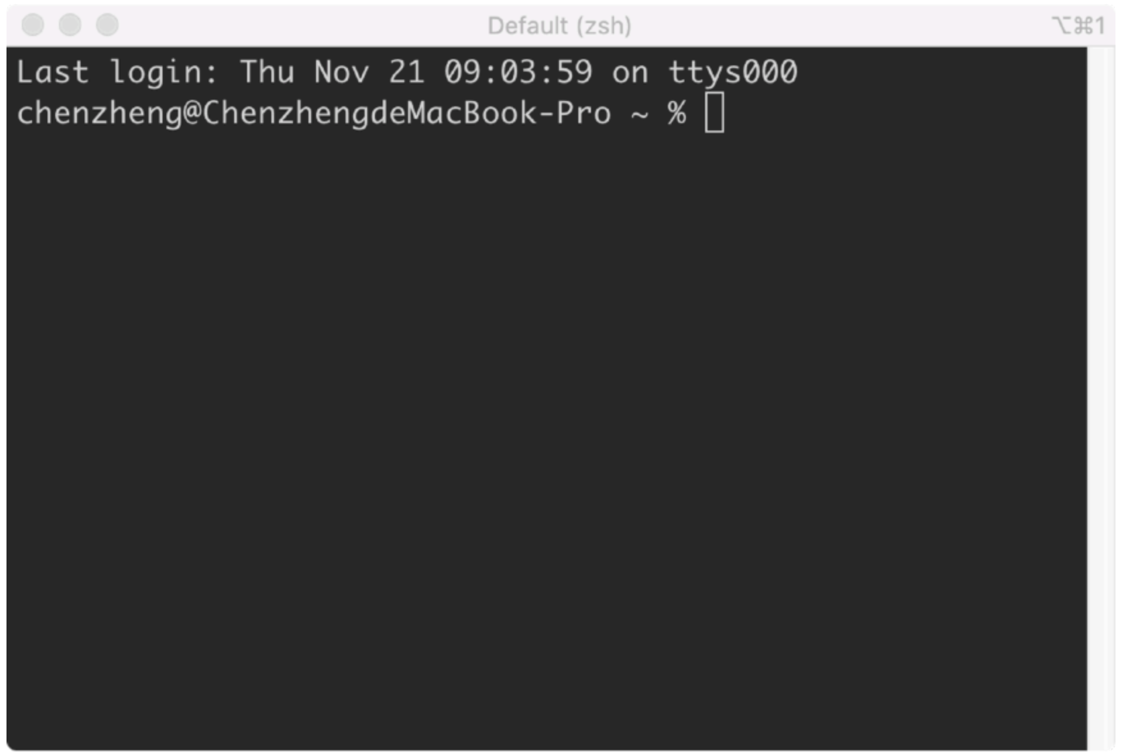
Zheng Chen

GUI programming

1. AWT, Swing and JavaFX
2. Components and Containers
3. Layout
4. Event Handling

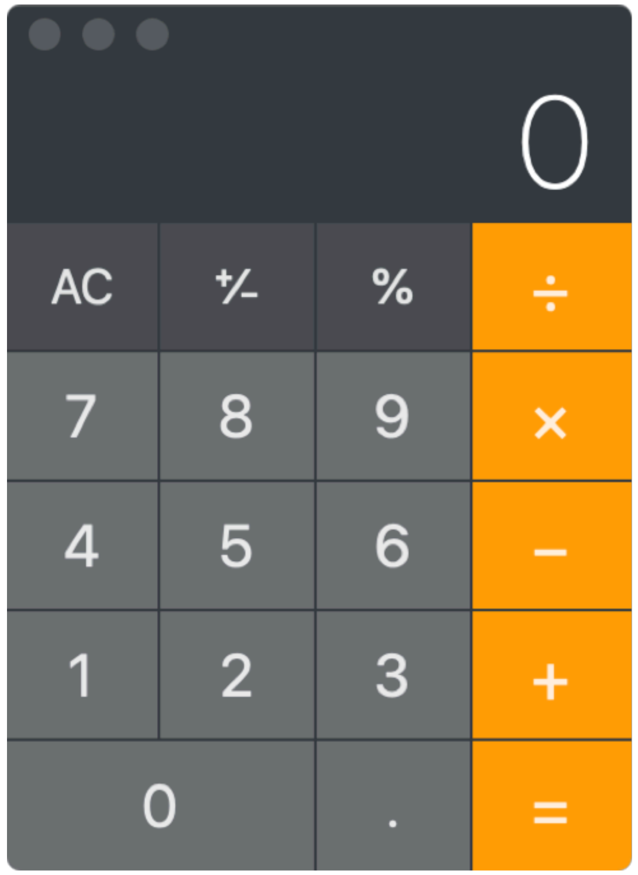
GUI

- The **graphical user interface** is a form of user interface that allows users to interact with electronic devices through graphical icons and audio indicator such as primary notation, instead of text-based user interfaces, typed command labels or text navigation.



```
Last login: Thu Nov 21 09:03:59 on ttys000
chenzheng@ChenzhengdeMacBook-Pro ~ %
```

Console



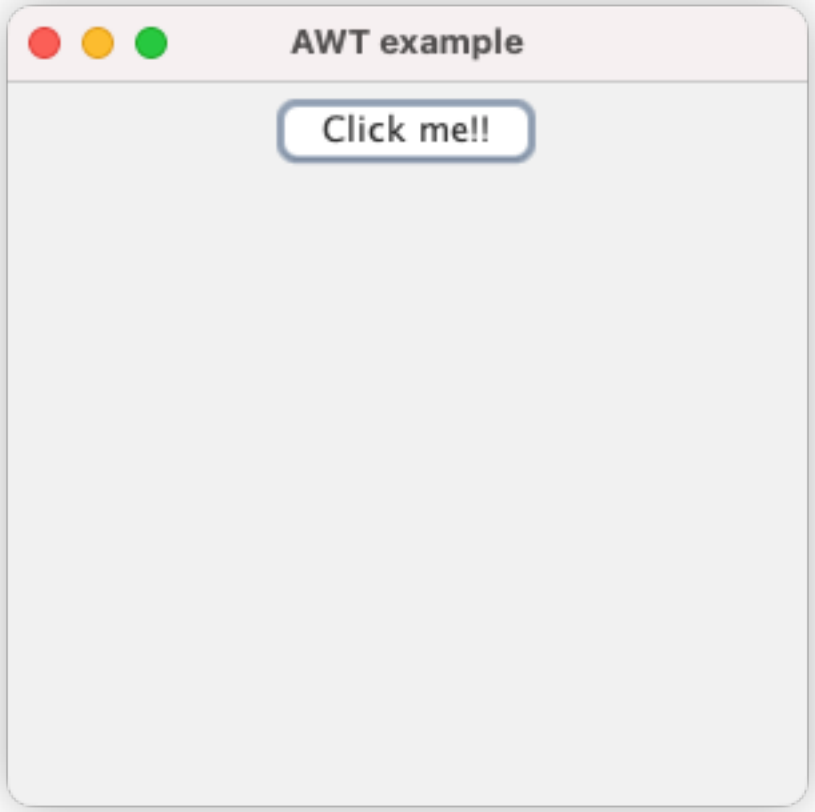
GUI

AWT vs. Swing

- Java AWT (Abstract Window Toolkit) is an API to develop GUI or window-based applications in java.
    - Java AWT components are **platform-dependent** i.e. components are displayed according to the view of operating system. AWT is **heavyweight** i.e. its components are using the resources of OS.
- Java Swing is built on the top of AWT API and entirely written in java.
    - Unlike AWT, Java Swing provides **platform-independent** and **lightweight** components.

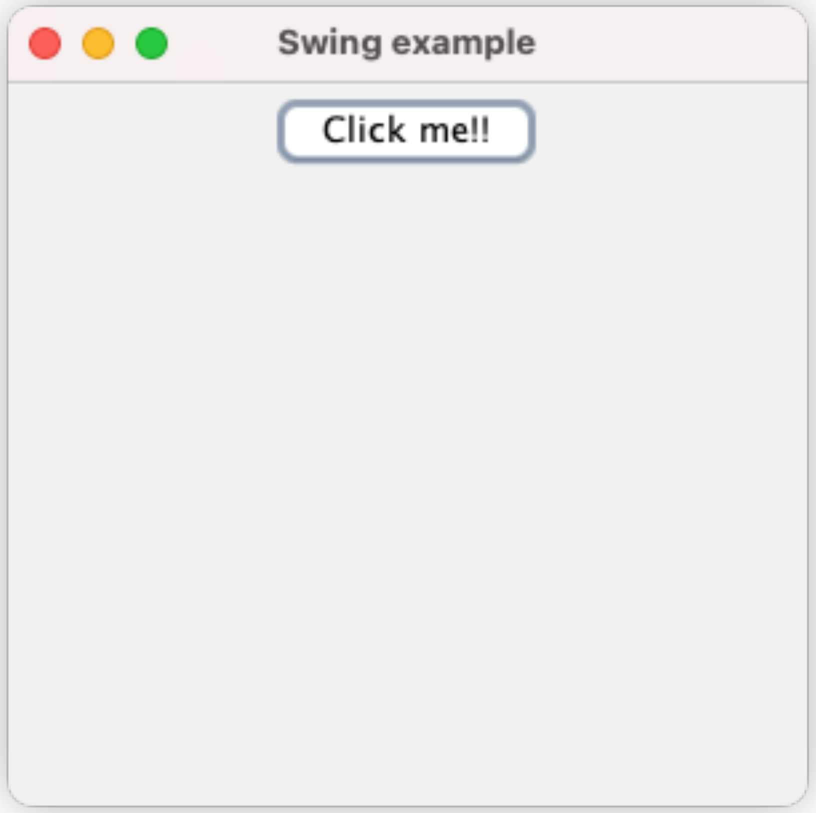
AWT Example

```
import java.awt.*;

public class SimpleExample {

  public static void main(String args[]) {
    Frame f = new Frame();
    Button b = new Button("Click me!!");

    f.add(b);
    f.setSize(300, 300);
    f.setTitle("AWT example");
    f.setLayout(new FlowLayout());
    f.setVisible(true);
  }
}
```



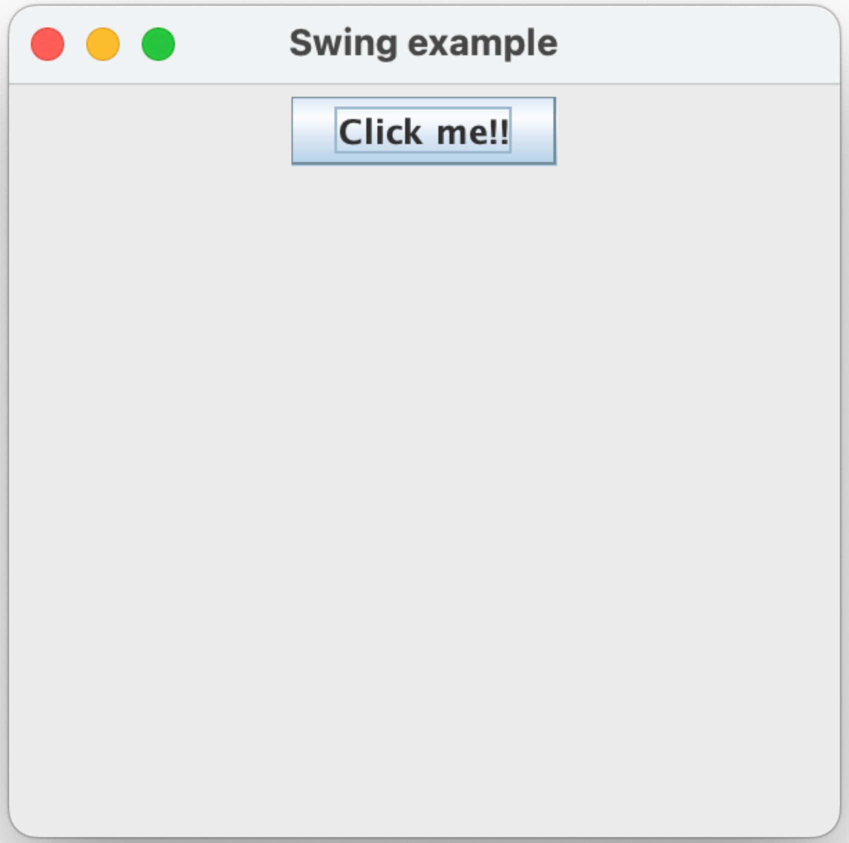
Swing Example

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

public class SimpleExample {

  public static void main(String args[]) {
    JFrame f = new JFrame();
    JButton b = new JButton("Click me!!");

    f.add(b);
    f.setSize(300, 300);
    f.setTitle("Swing example");
    f.setLayout(new FlowLayout());
    f.setVisible(true);
  }
}
```

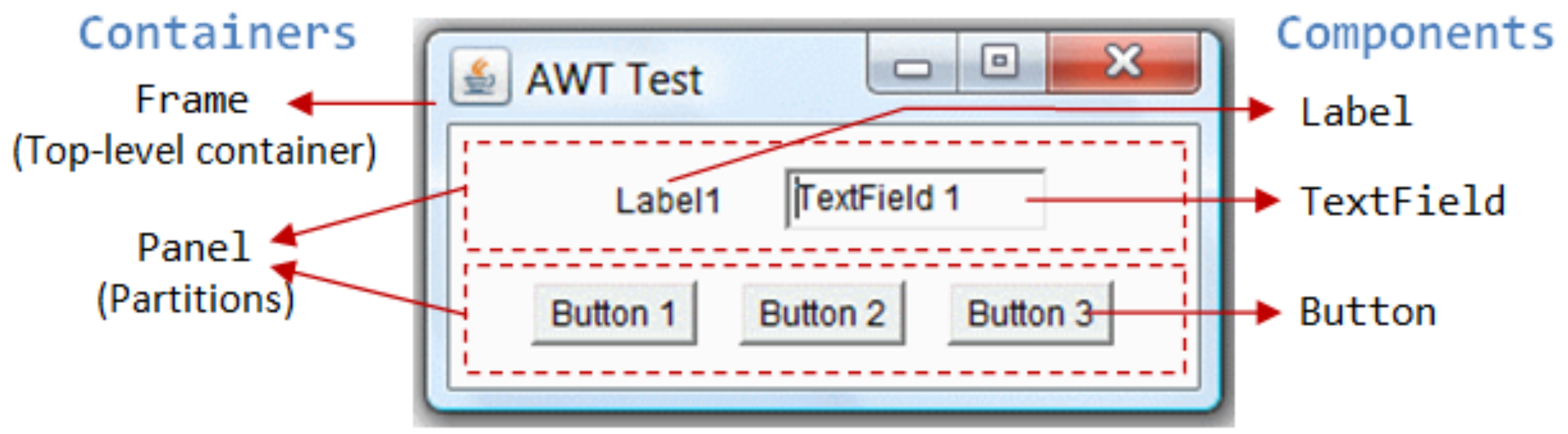


Swing Example

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

public class SimpleExample {

  public static void main(String args[]) {
    String lookAndFeel = "javax.swing.plaf.metal.MetalLookAndFeel";
    UIManager.setLookAndFeel(lookAndFeel);

    JFrame f = new JFrame();
    JButton b = new JButton("Click me!!");

    f.add(b);
    f.setSize(300, 300);
    f.setTitle("Swing example");
    f.setLayout(new FlowLayout());
    f.setVisible(true);
  }
}
```

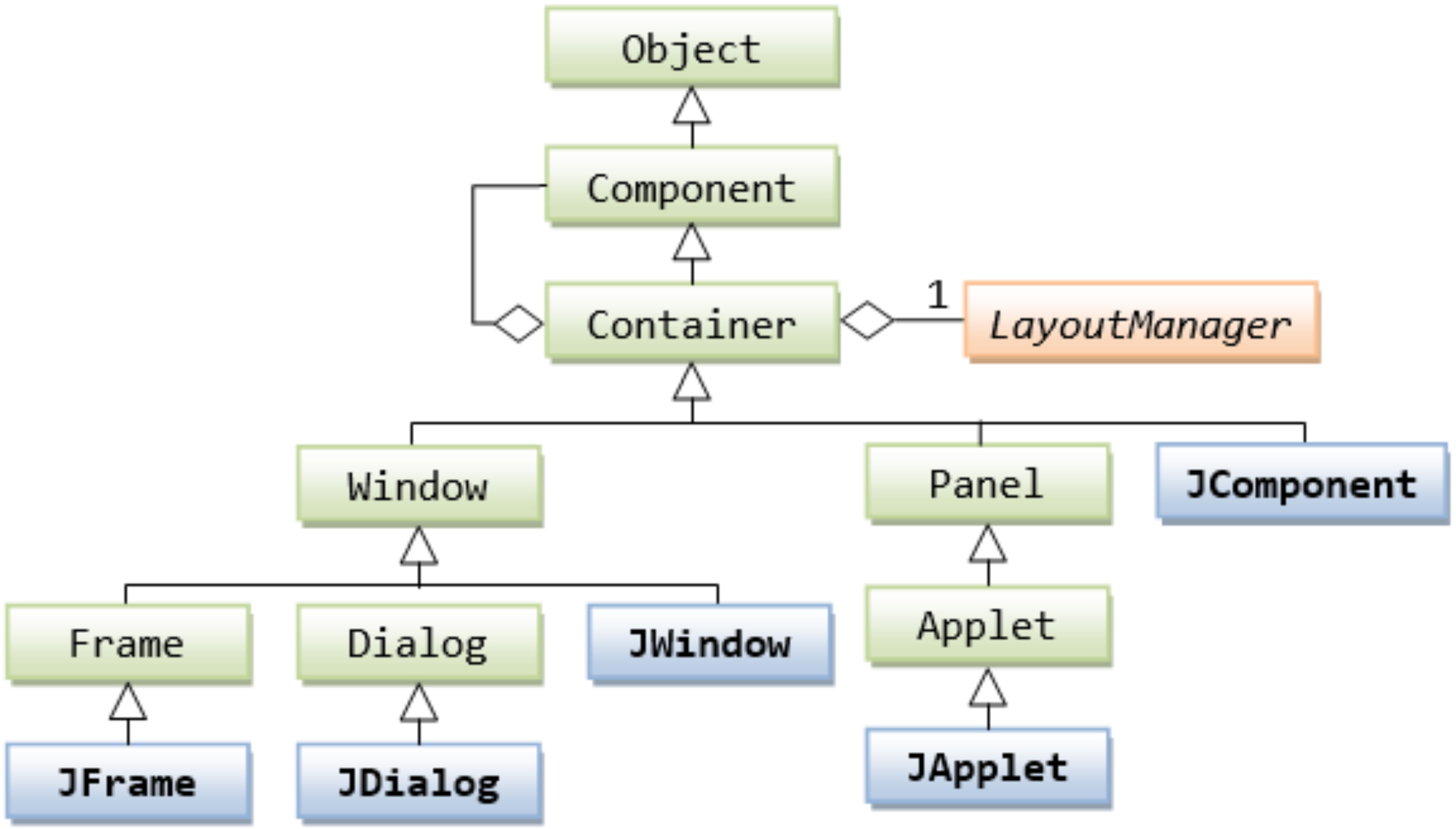


AWT in Detail



- **Components** are elementary GUI entities, such as Button, Label, and TextField.
- **Containers**, such as Frame and Panel, are used to hold components in a specific layout (such as FlowLayout or GridLayout). A container can also hold sub-containers.
- **Layout manager** are used to arrange components within a container.

Java AWT and Swing Hierarchy



Containers

- **Window** — The window is the container that have no borders and menu bars. You must use frame, dialog or another window for creating a window.
- **Panel** — The Panel is the container that doesn't contain title bar and menu bars. It can have other components like button, textfield etc.
- **Frame** — The Frame is the container that contain title bar and can have menu bars. It can have other components like button, textfield etc.

Common useds Methods

- java.awt.Container
  - Component add(Component comp) Appends the specified component to the end of this container.
  - Component[] getComponents() Gets all the components in this container.
  - void remove(int index) Removes the component, specified by index, from this container.
  - void removeAll() Removes all the components from this container.
  - void setLayout(LayoutManager mgr) Sets the layout manager for this container.
  - void doLayout() Causes this container to lay out its components.
  - void paint(Graphics g) Paints the container.
  - void update(Graphics g) Updates the container.
  - void validate() Validates this container and all of its subcomponents.

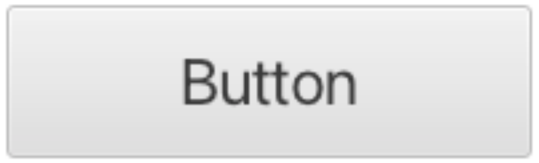
Common useds Methods

- java.awt.Window extends Container
  - void pack() Causes this Window to be sized to fit the preferred size and layouts of its subcomponents.
  - void setAlwaysOnTop(boolean alwaysOnTop) Sets whether this window should always be above other windows.
  - void setBackground(Color bgColor) Sets the background color of this window.
  - void setLocation(int x, int y) Moves this component to a new location.
  - void setSize(int width, int height) Resizes this component so that it has width width and height height.
  - void setVisible(boolean b) Shows or hides this Window depending on the value of parameter b.
  - void toBack() If this Window is visible, sends this Window to the back and may cause it to lose focus or activation if it is the focused or active Window.
  - void toFront() If this Window is visible, brings this Window to the front and may make it the focused Window.

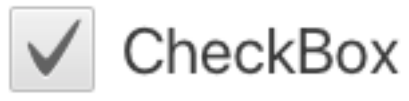
Common useds Methods

- java.awt.Frame extends Window
    - void setMenuBar(MenuBar mb) Sets the menu bar for this frame to the specified menu bar.
    - void setTitle(String title) Sets the title for this frame to the specified string.
    - void setResizable(boolean resizable) Sets whether this frame is resizable by the user.

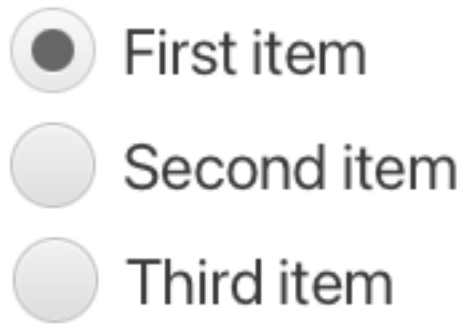
Components



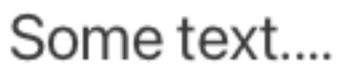
Button



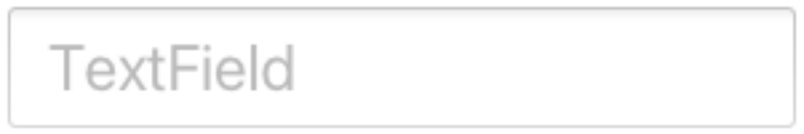
Checkbox



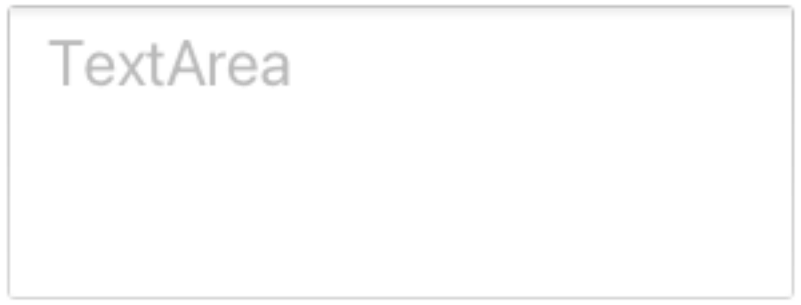
CheckboxGroup



Label



TextField



TextArea

Common useds Methods

- java.awt.Button
  - String getLabel() Gets the label of this button.
  - void setLabel(String label) Sets the button's label to be the specified string.
- java.awt.Checkbox
  - String getLabel() Gets the label of this check box.
  - void setLabel(String label) Sets this check box's label to be the string argument.
  - boolean getState() Determines whether this check box is in the "on" or "off" state.
  - void setState(boolean state) Sets the state of this check box to the specified state.
  - void setCheckboxGroup(CheckboxGroup g) Sets this check box's group to the specified check box group.
  - Object[] getSelectedObjects() Returns an array (length 1) containing the checkbox label or null if the checkbox is not selected.

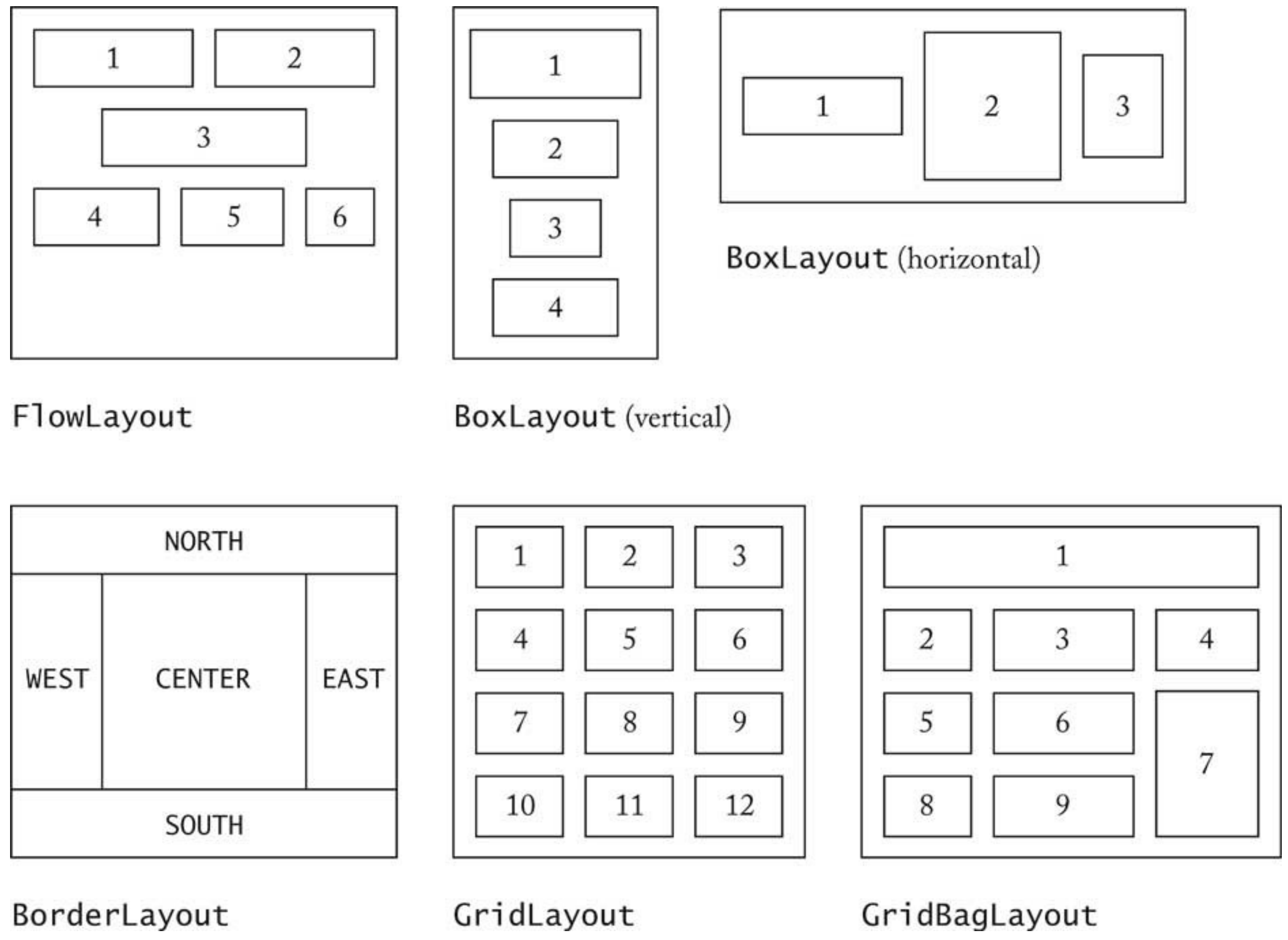
Common useds Methods

- java.awt.Label
  - String getText() Gets the text of this label.
  - void setText(String text) Sets the text for this label to the specified text.
- java.awt.TextComponent
  - String getText() Returns the text that is presented by this text component.
  - void setText(String t) Sets the text that is presented by this text component to be the specified text.
  - String getSelectedText() Returns the selected text from the text that is presented by this text component.
  - void setEditable(boolean b) Sets the flag that determines whether or not this text component is editable.
  - boolean isEditable() Indicates whether or not this text component is editable.

Layout Manager

- A layout manager controls the size and position (layout) of components inside a Container object.

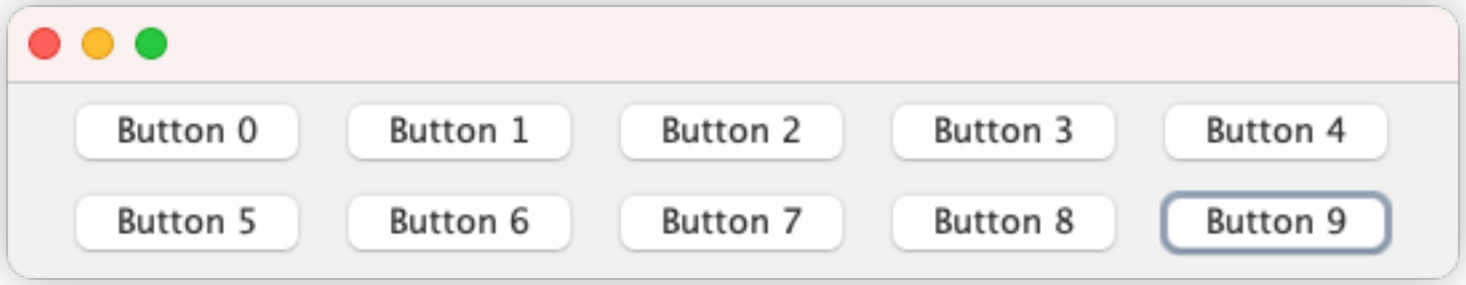
- For example, a window is a container that contains components such as buttons and labels. The layout manager in effect for the window determines how the components are sized and positioned inside the window.

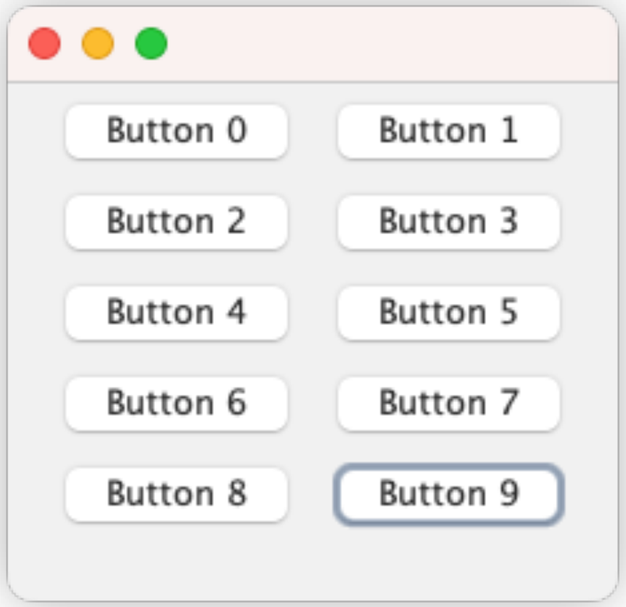


FlowLayout

- A flow layout arranges components in a directional flow, much like lines of text in a paragraph.

```
import java.awt.*;
public class FlowLayoutExample{
    public static void main(String args[]) {
        Frame f = new Frame();
        for (int i=0;i<10;i++) f.add(new Button( "Button "+i));
        f.setLayout(new FlowLayout());
        f.pack();
        f.setVisible(true);
    }
}
```





Steps of GUI Programming

1. Create a container.

2. Adding Components.

3. Arrange them with Layout Manager

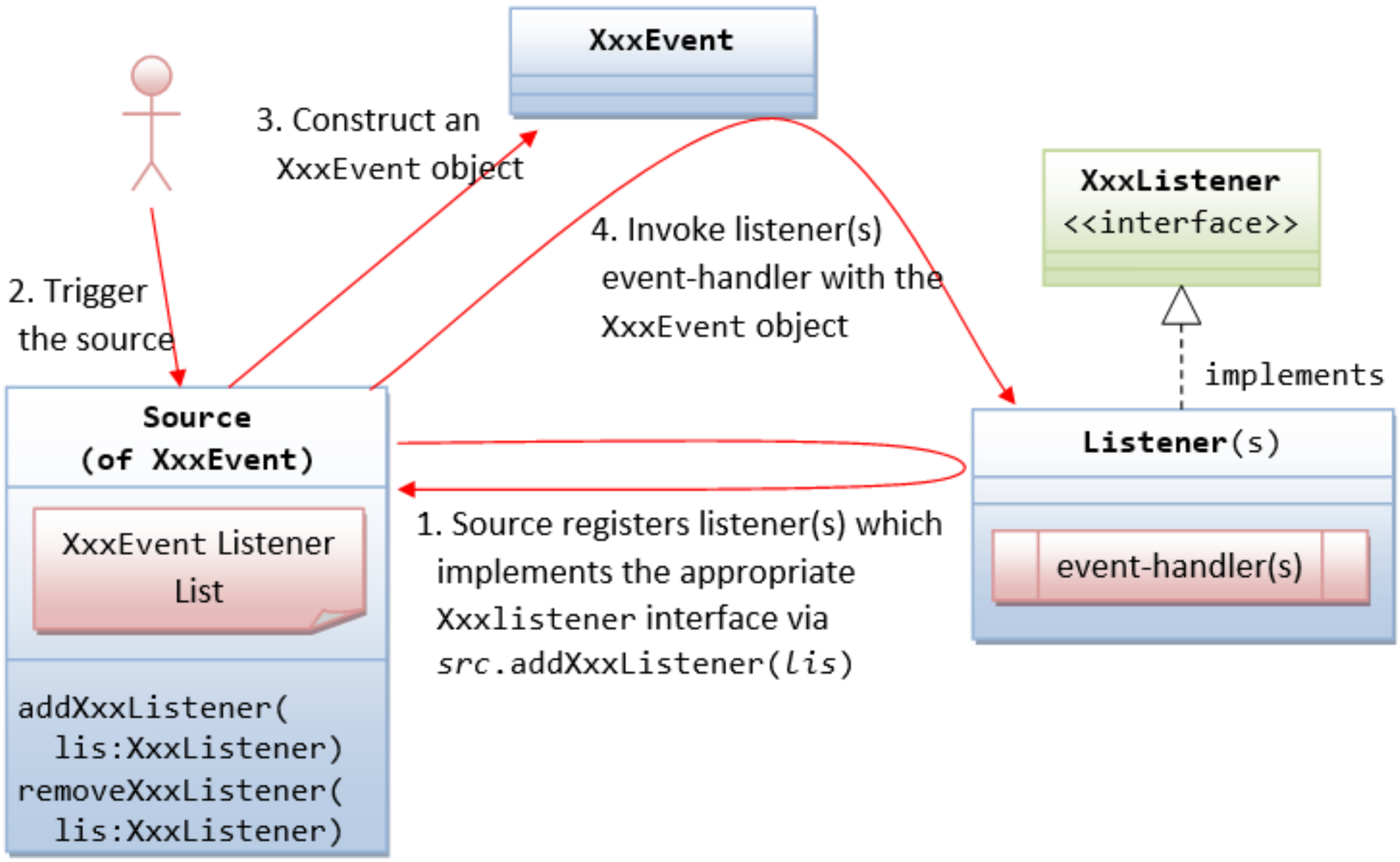
```
import java.awt.*;

public class SimpleExample extends Frame {

    public static void main(String args[]) {
        Frame f = new Frame();
        Button b = new Button("Click me!!");

        f.add(b);
        f.setSize(300, 100);
        f.setTitle("This is my First AWT example");
        f.setLayout(new FlowLayout());
        f.setVisible(true);
    }
}
```

But, how to interact?

Event-Handling

- Java adopts the so-called "Event-Driven" (or "Event-Delegation") programming model for event-handling.
    - The **source object** (such as Button and Textfield) interacts with the user.
    - Upon triggered, the source object creates an **event object** to capture the action (e.g., mouse-click x and y, texts entered, etc).
    - This event object will be messaged to all the registered **listener object**(s), and an appropriate **event-handler method** of the listener(s) is called-back to provide the response.

Source, Event and Listener


XxxEvent
3. Construct an XxxEvent object
2. Trigger the source
4. Invoke listener(s) event-handler with the XxxEvent object
XxxListener
<<interface>>
implements
Source
(of XxxEvent)
XxxEvent Listener List
addXxxListener(
lis:XxxListener)
removeXxxListener(
lis:XxxListener)
1. Source registers listener(s) which implements the appropriate Xxxlistener interface via *src*.addXxxListener(*lis*)
Listener(s)
event-handler(s)

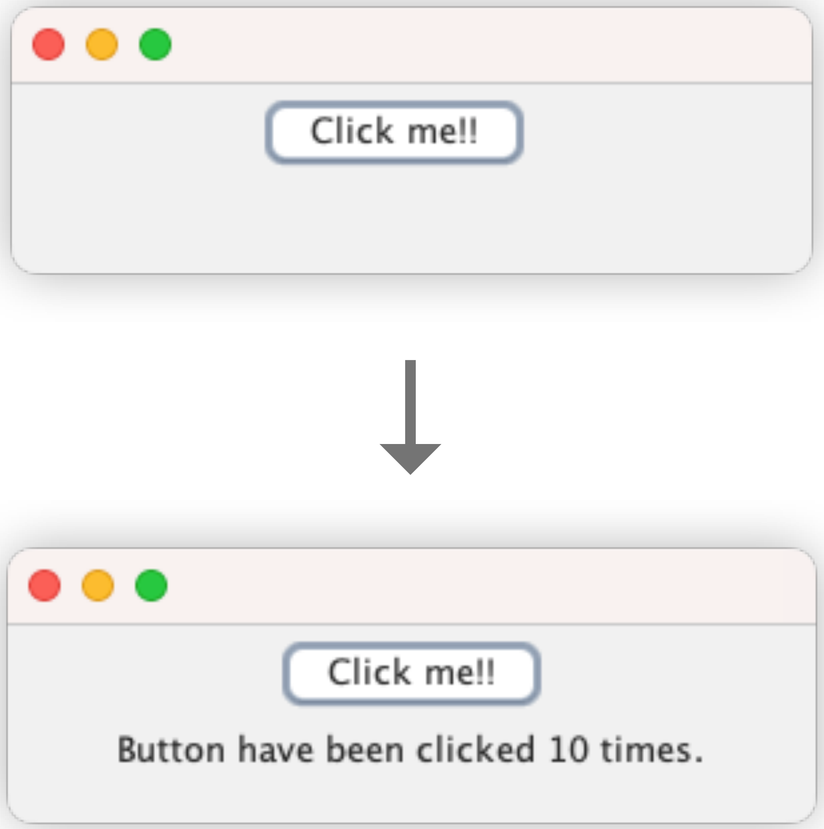
Event Classes	**Listener Interfaces**
ActionEvent	ActionListener
MouseEvent	MouseListener and MouseMotionListener
MouseWheelEvent	MouseWheelListener
KeyEvent	KeyListener
ItemEvent	ItemListener
TextEvent	TextListener
AdjustmentEvent	AdjustmentListener
WindowEvent	WindowListener
ComponentEvent	ComponentListener
ContainerEvent	ContainerListener
FocusEvent	FocusListener

ActionEvent and ActionListener

- class java.awt.event.ActionEvent
  - A semantic event which indicates that a component-defined action occurred. This high-level event is generated by a component (such as a Button) when the component-specific action occurs (such as being pressed). The event is passed to every ActionListener object that registered to receive such events using the component's addActionListener method.
- interface ActionListener
  - The listener interface for receiving action events. The class that is interested in processing an action event implements this interface, and the object created with that class is registered with a component, using the component's addActionListener method. When the action event occurs, that object's actionPerformed method is invoked.
  - void actionPerformed(ActionEvent e) Invoked when an action occurs.
- class java.awt.Button
  - void addActionListener(ActionListener l) Adds the specified action listener to receive action events from this button.

Event example

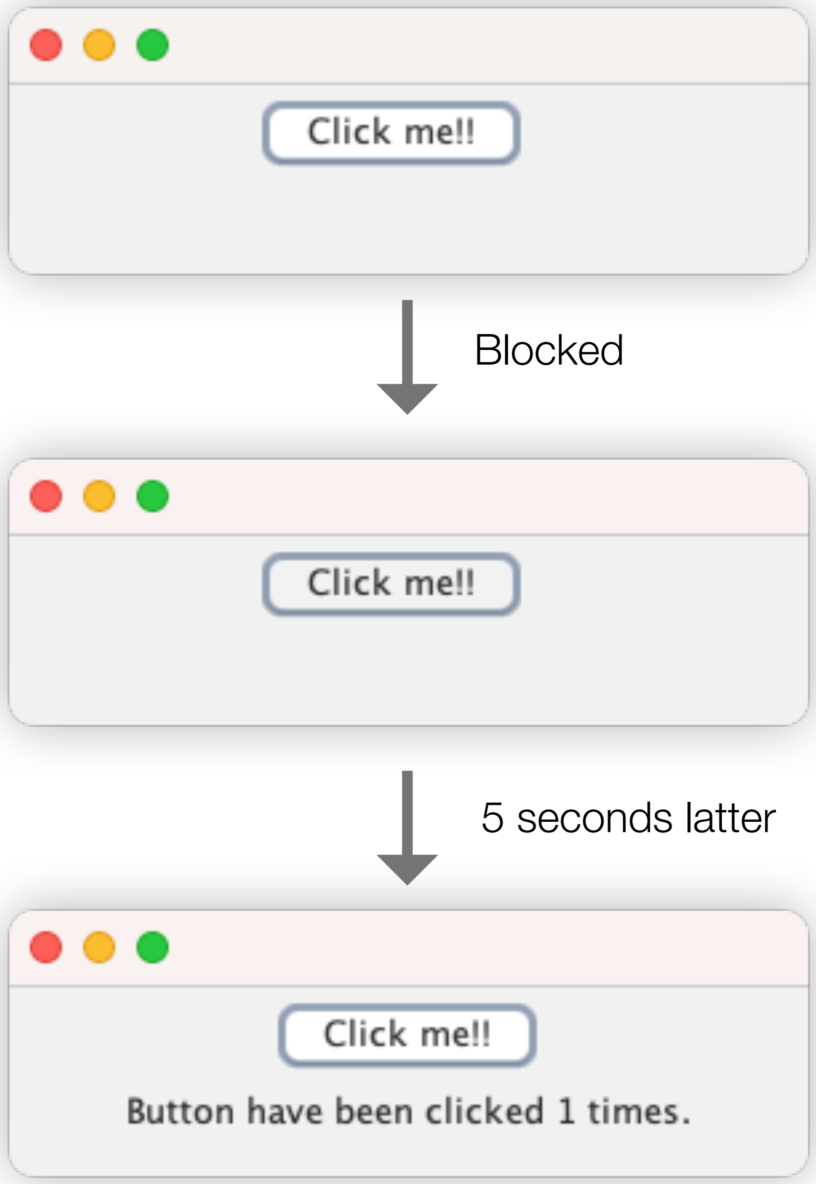
```
import java.awt.*;
import java.awt.event.*;

public class ActionEventExample {
  public static void main(String args[]) {
    Frame f = new Frame();
    Button b = new Button("Click me!!");
    Label l = new Label();
    f.add(b);
    f.add(l);
    f.setSize(300, 100);
    f.setLayout(new FlowLayout());
    f.setVisible(true);
    ActionListener listener = new MyActionListener(l,f);
    b.addActionListener(listener);
  }
}

class MyActionListener implements ActionListener {
  Label l;
  Frame f;
  int i;
  MyActionListener(Label l, Frame f) {
    this.l = l;
    this.f = f;
  }
 public void actionPerformed(ActionEvent e) {
    l.setText("Button have been clicked " + ++i + " times.");
    f.validate();
  }
}
```



```
Button have been clicked 10 times.
```

What if the task is really heavy?

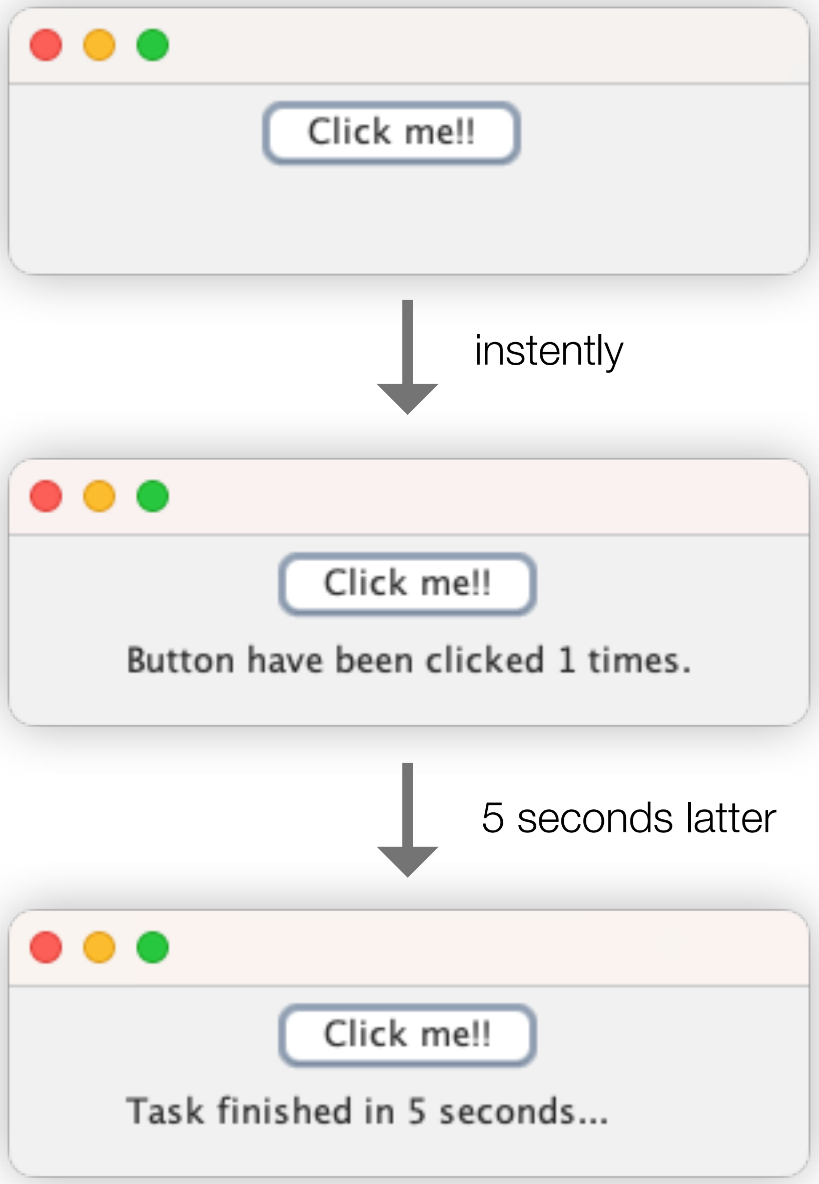
```
class MyActionListener implements ActionListener {

  Label l;
  Frame f;
  int i;

  MyActionListener(Label l, Frame f) {
    this.l = l;
    this.f = f;
  }


 public void actionPerformed(ActionEvent e) {

    try {
      // Pretend to be working on a complicated task
      // UI thread will be blocked untill the task is done
      Thread.sleep(5000);
    } catch (InterruptedException e1) {
      // do nothing
    }

    l.setText("Button have been clicked " + ++i + " times.");
    f.validate();
  }
}
```



Button have been clicked 1 times.

GUI with thread

```
class MyActionListener implements ActionListener {

  Label l;
  Frame f;
  int i;

  MyActionListener(Label l, Frame f) {
    this.l = l;
    this.f = f;
  }


  public void actionPerformed(ActionEvent e) {
    new Thread() {
      public void run() {
        try {
          // Working in another thread
          // UI thread will not be blocked
          Thread.sleep(5000);
        } catch (InterruptedException e1) {
          // do nothing
        }
        l.setText("Task finished in 5 seconds...");
      }
    }.start();
    l.setText("Button have been clicked " + ++i + " times.");
    f.validate();
  }
}
```

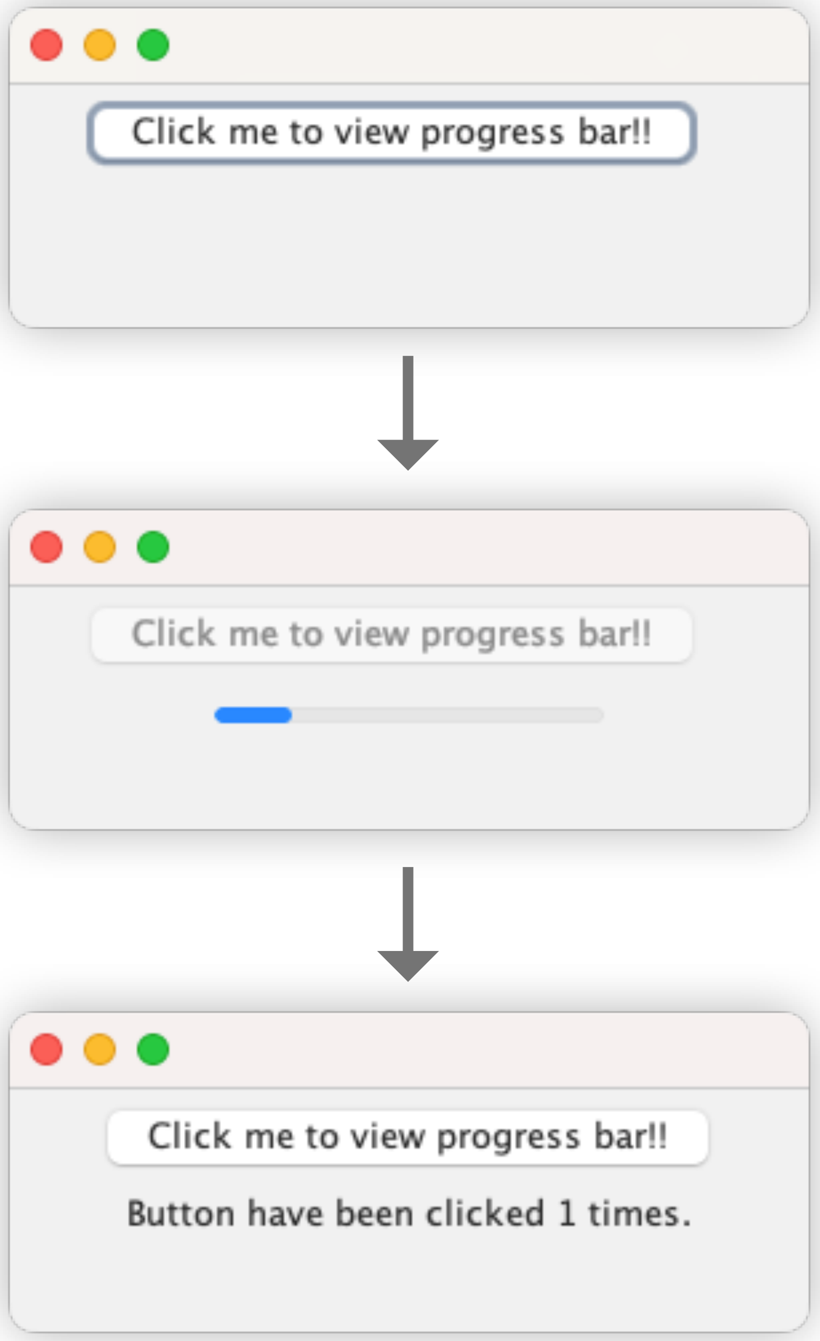


Button have been clicked 1 times.

Task finished in 5 seconds...

GUI with visual indicator

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

public class GUIIndicator {

  static Frame f = new Frame();
  static Button b = new Button("Click me to view progress bar!!");
  static Label l = new Label();
  static JProgressBar p = new JProgressBar();

  public static void main(String args[]) {
    f.add(b);
    f.add(l);
    f.add(p);
    p.setVisible(false);

    f.pack();
    f.setSize(300, 150);
    f.setLayout(new FlowLayout());
    f.setVisible(true);

    ActionListener listener = new MyActionListener(f, l, p, b);
    b.addActionListener(listener);
  }
}

class MyActionListener implements ActionListener {

  Frame f;
  Label l;
  JProgressBar p;
  Button b;
  int i;

  MyActionListener(Frame f, Label l, JProgressBar p, Button b) {
    this.f = f;
    this.l = l;
    this.b = b;
    this.p = p;
  }

  public void actionPerformed(ActionEvent e) {
    new Thread() {
      public void run() {
        try {
          b.setEnabled(false);
          p.setValue(0);
          p.setVisible(true);
          f.validate();
          for (int i = 0; i < 5; i++) {
            Thread.sleep(1000);
            p.setValue(p.getValue() + 20);
          }
          Thread.sleep(200);
          p.setVisible(false);
          l.setText("Button have been clicked " + ++i + " times.");
          b.setEnabled(true);
          f.validate();
        } catch (InterruptedException e1) {}
      }
    }
      .start();
  }
}
```


Click me to view progress bar!!
Click me to view progress bar!!
Click me to view progress bar!!
Button have been clicked 1 times.

JavaFX and SceneBuilder

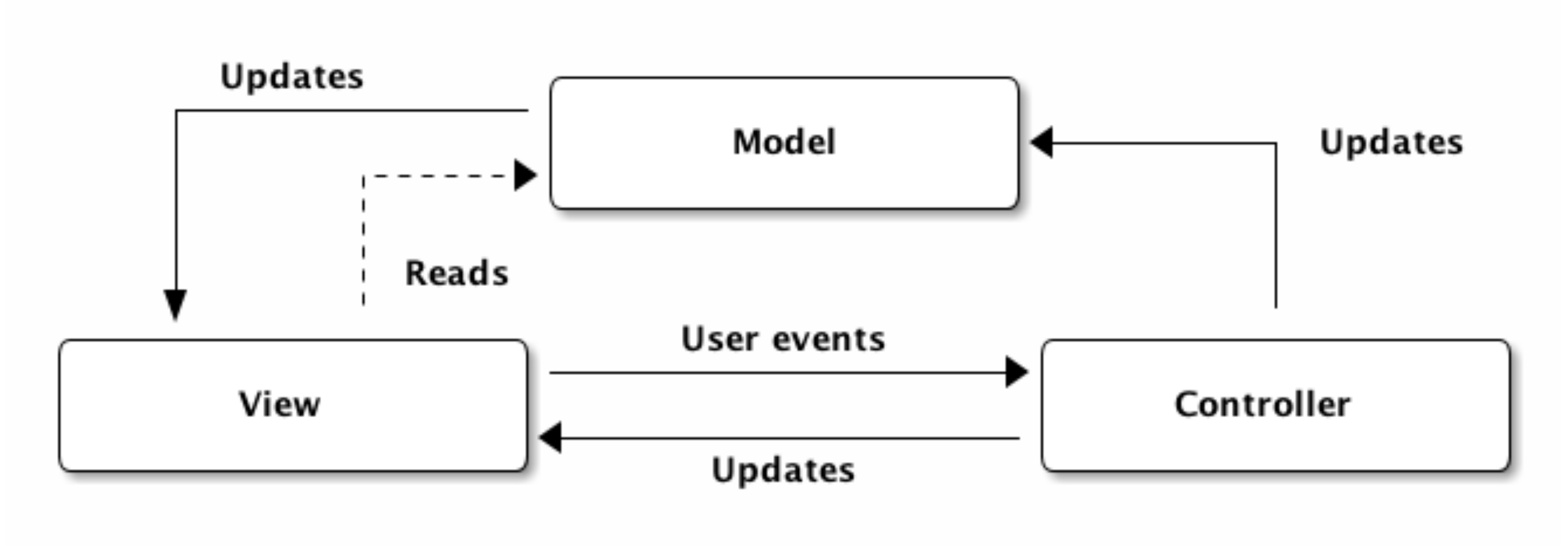
- JavaFX is the next generation client application platform for desktop, mobile and embedded systems built on Java.

- With JavaFX, you can easily construct an application with the Model-View-Controller (MVC) pattern.

- JavaFX come with an integrated Scene Builder, which can perform Drag & Drop user interface design.

MVC: Model-View-Controller

- The MVC pattern arose as a solution to keep 3 concerns separate from each other: visuals (View), data (Model), and logic (Controller).



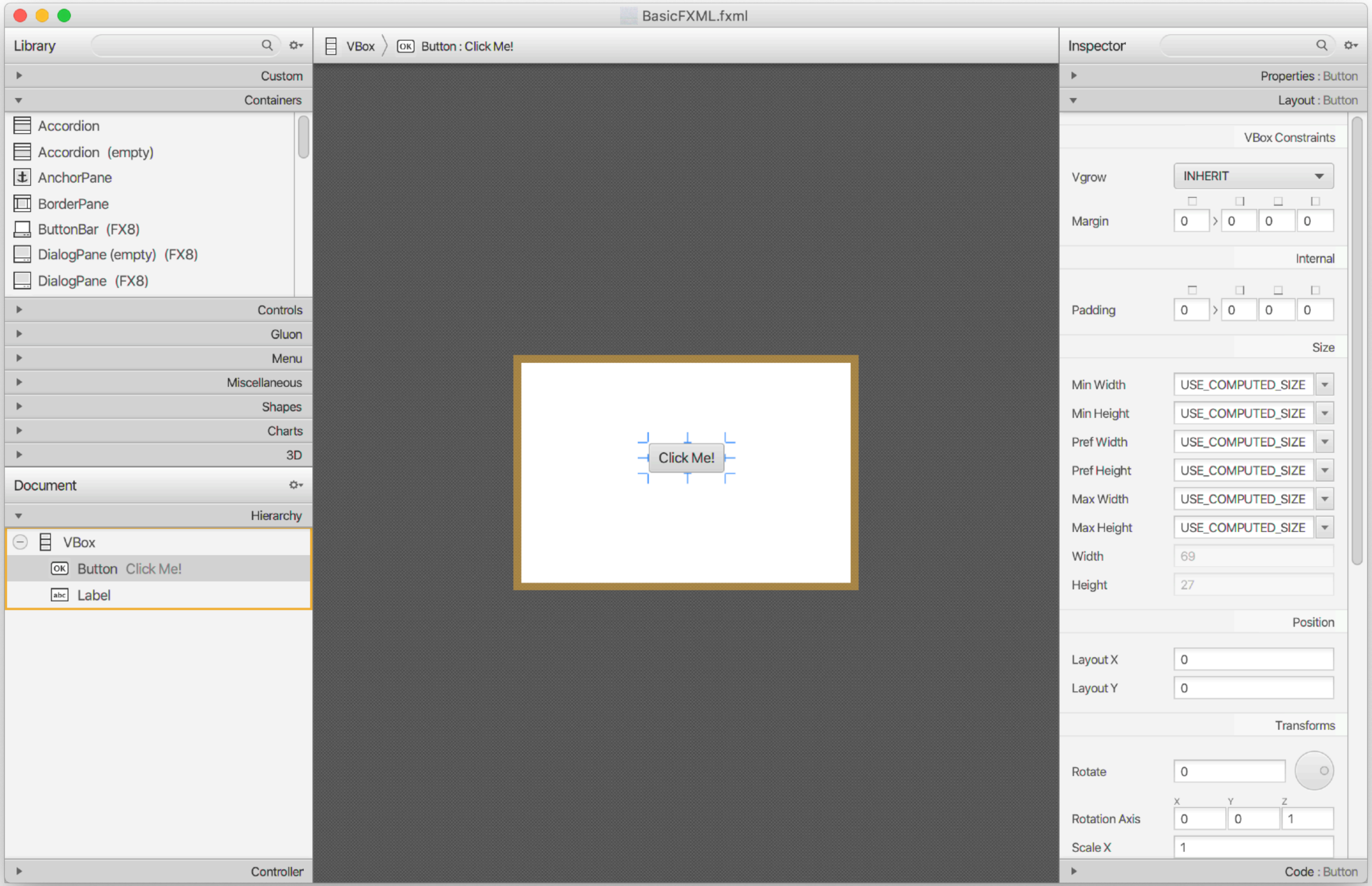
JavaFX Example — Model

```
import javafx.application.Application;
import javafx.fxml.FXMLLoader;
import javafx.scene.Parent;
import javafx.scene.Scene;
import javafx.stage.Stage;

public class BasicApplication extends Application {

    @Override
    public void start(Stage stage) throws Exception {
        Parent root = FXMLLoader.load(getClass().getResource("BasicFXML.fxml"));
        Scene scene = new Scene(root);
        stage.setScene(scene);
        stage.show();
    }

    public static void main(String[] args) {
        launch(args);
    }
}
```

JavaFX Example — View

```
<?xml version="1.0" encoding="UTF-8"?>

<?import javafx.scene.control.Button?>
<?import javafx.scene.control.Label?>
<?import javafx.scene.layout.VBox?>


<VBox alignment="CENTER" prefHeight="200.0" prefWidth="300.0"
      spacing="10.0" xmlns="http://javafx.com/javafx/8.0.141"
      xmlns:fx="http://javafx.com/fxml/1"
      fx:controller="BasicFXMLController">
   <children>
        <Button fx:id="button" onAction="#handleButtonAction" text="Click Me!" />
        <Label fx:id="label" minHeight="16" minWidth="69" />
   </children>
</VBox>
```

View designed with Scene Builder



VBox Constraints

Vgrow: INHERIT
Margin: 0 0 0 0

Internal

Padding: 0 0 0 0

Size

Min Width	USE_COMPUTED_SIZE
Min Height	USE_COMPUTED_SIZE
Pref Width	USE_COMPUTED_SIZE
Pref Height	USE_COMPUTED_SIZE
Max Width	USE_COMPUTED_SIZE
Max Height	USE_COMPUTED_SIZE
Width	69
Height	27

Position

Layout X: 0
Layout Y: 0

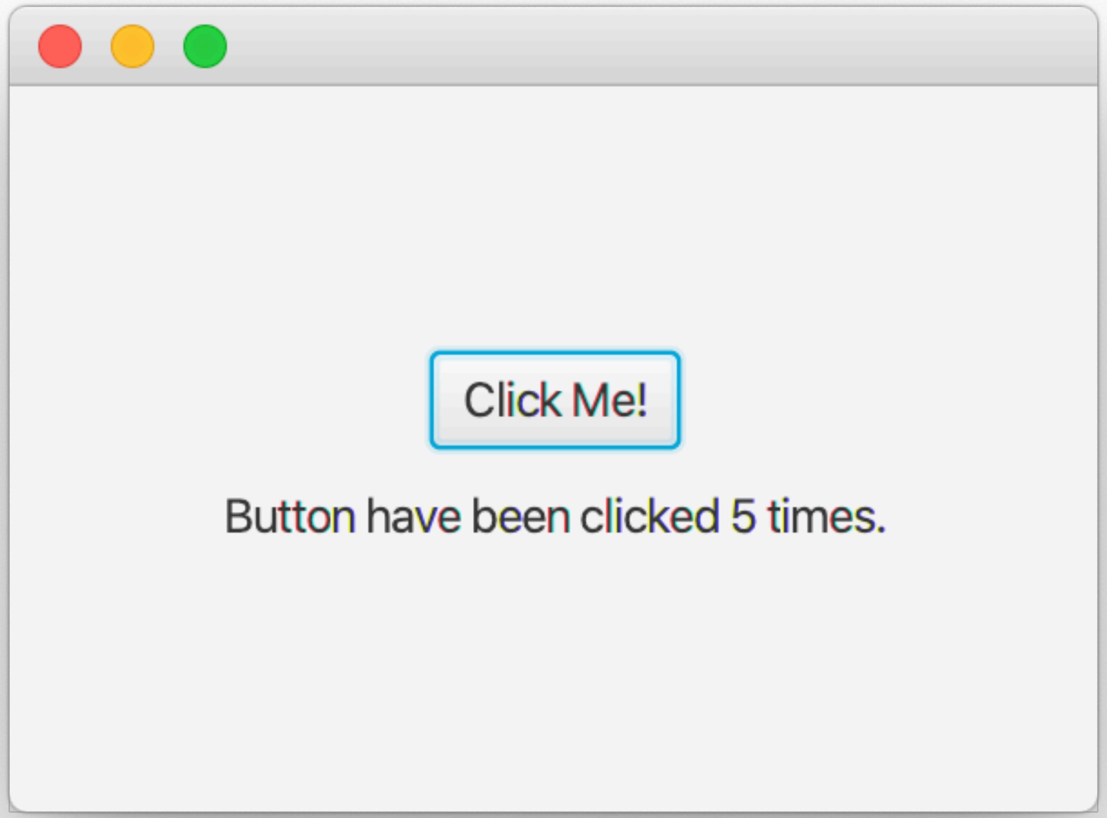
Transforms

Rotate: 0
Rotation Axis: X 0, Y 0, Z 1
Scale X: 1

JavaFX Example — Controller

```java
import javafx.event.ActionEvent;
import javafx.fxml.FXML;
import javafx.scene.control.Label;

public class BasicFXMLController {

    int i;
    @FXML
    private Label label;

    public void initialize() {
   }

    @FXML
    private void handleButtonAction(ActionEvent event) {
        label.setText("Button have been clicked " + ++i + " times.");
    }
}
```

JavaFX Example — Run



Button have been clicked 5 times.

Homework

None…