Object Oriented Programming with Java

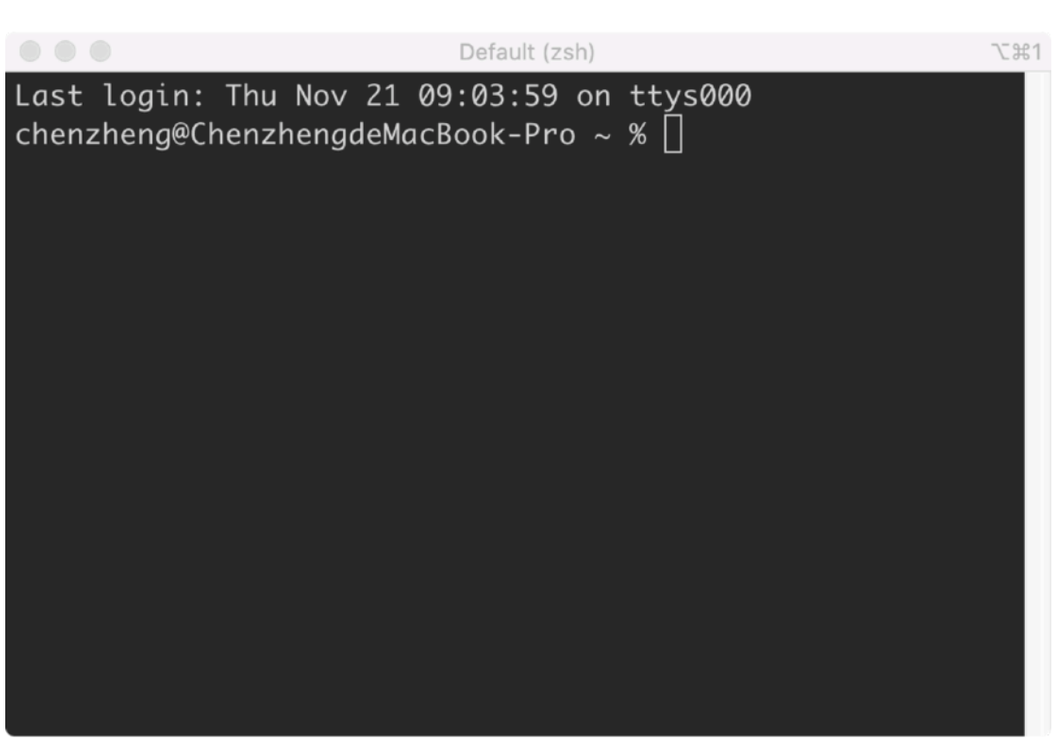
Zheng Chen

GUI programming

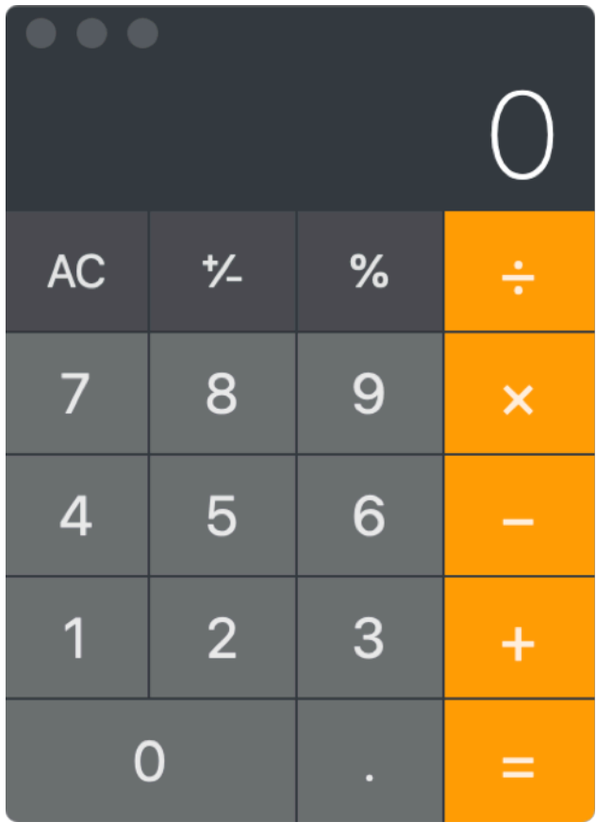
1. AWT, Swing and JavaFX
2. Components and Containers
3. Layout
4. Event Handling

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.



Console



GUI

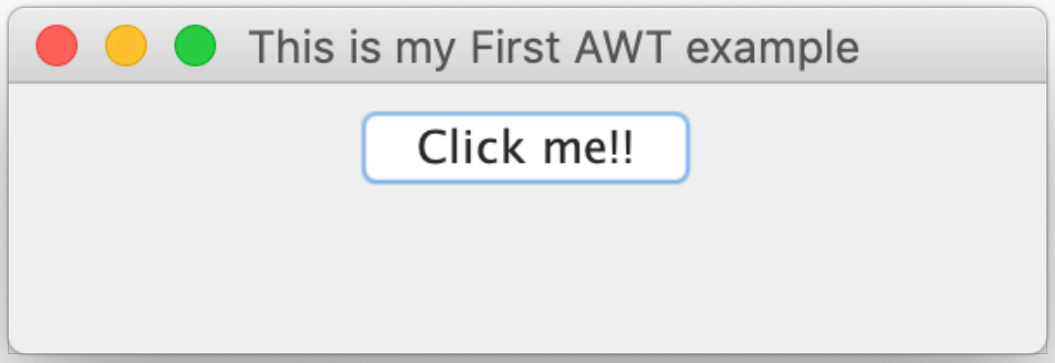
AWT vs. Swing

- 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.

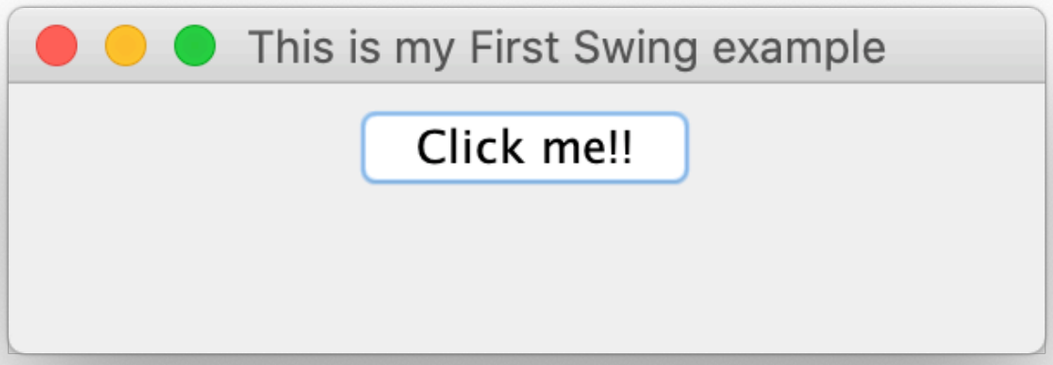
AWT Example

```
import java.awt.*;

public class SimpleExample 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);
    }
}
```

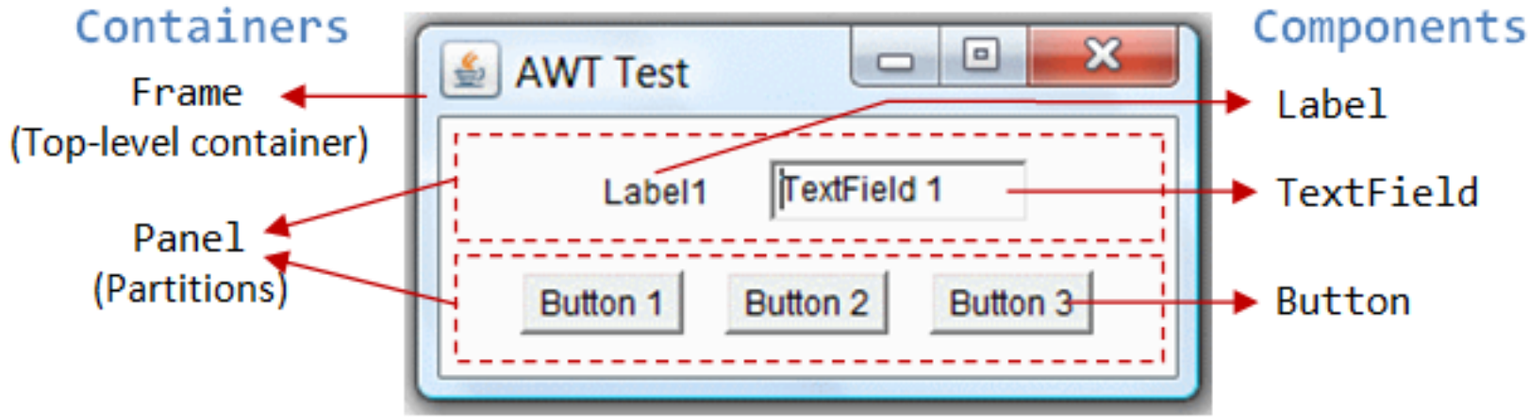


Swing Example

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

public class SimpleExample extends Frame {

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

        f.add(b);
        f.setSize(300, 100);
        f.setTitle("This is my First 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.

Containers

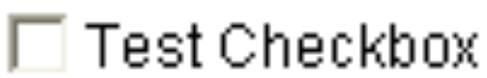
- **Window** — The window is the container that have no borders and menu bars. You must use frame, dialog or another window for creating a window.

- **Panel** — The Panel is the container that doesn't contain title bar and menu bars. It can have other components like button, textfield etc.

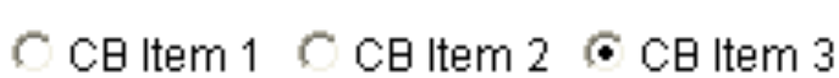
- **Frame** — The Frame is the container that contain title bar and can have menu bars. It can have other components like button, textfield etc.

Components

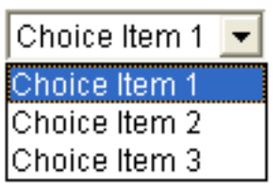
Button



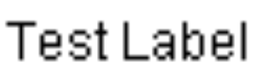
Checkbox



CheckboxGroup



Choice

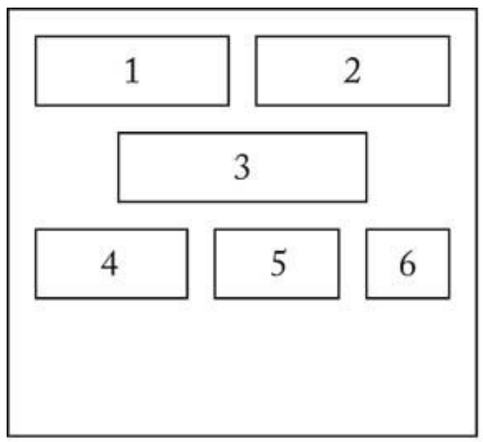


Label

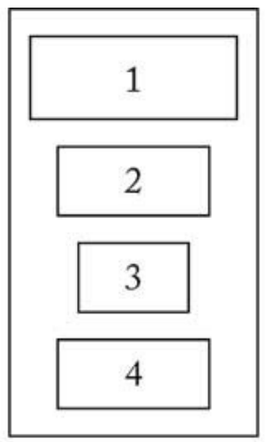
TextField

Layout Manager

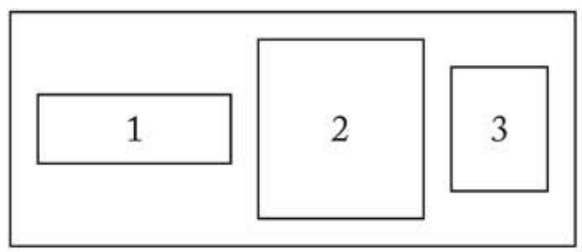
- A layout manager controls the size and position (layout) of components inside a Container object.
- For example, a window is a container that contains components such as buttons and labels. The layout manager in effect for the window determines how the components are sized and positioned inside the window.



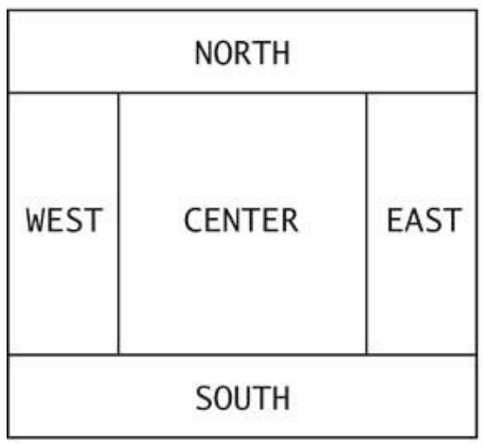
FlowLayout



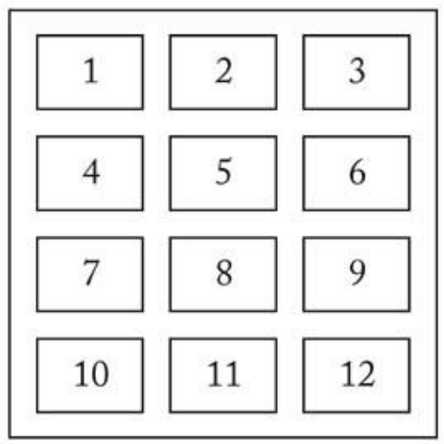
BoxLayout (vertical)



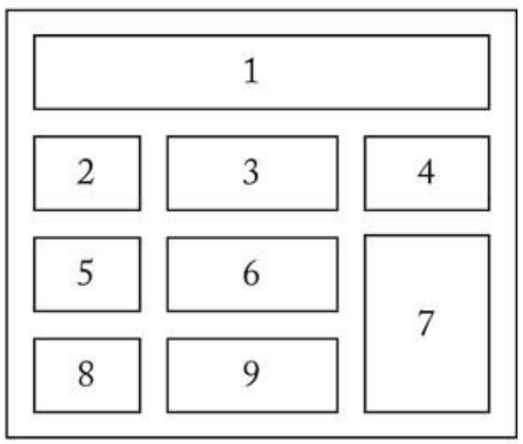
BoxLayout (horizontal)



BorderLayout



GridLayout



GridBagLayout

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);
   }
}
```

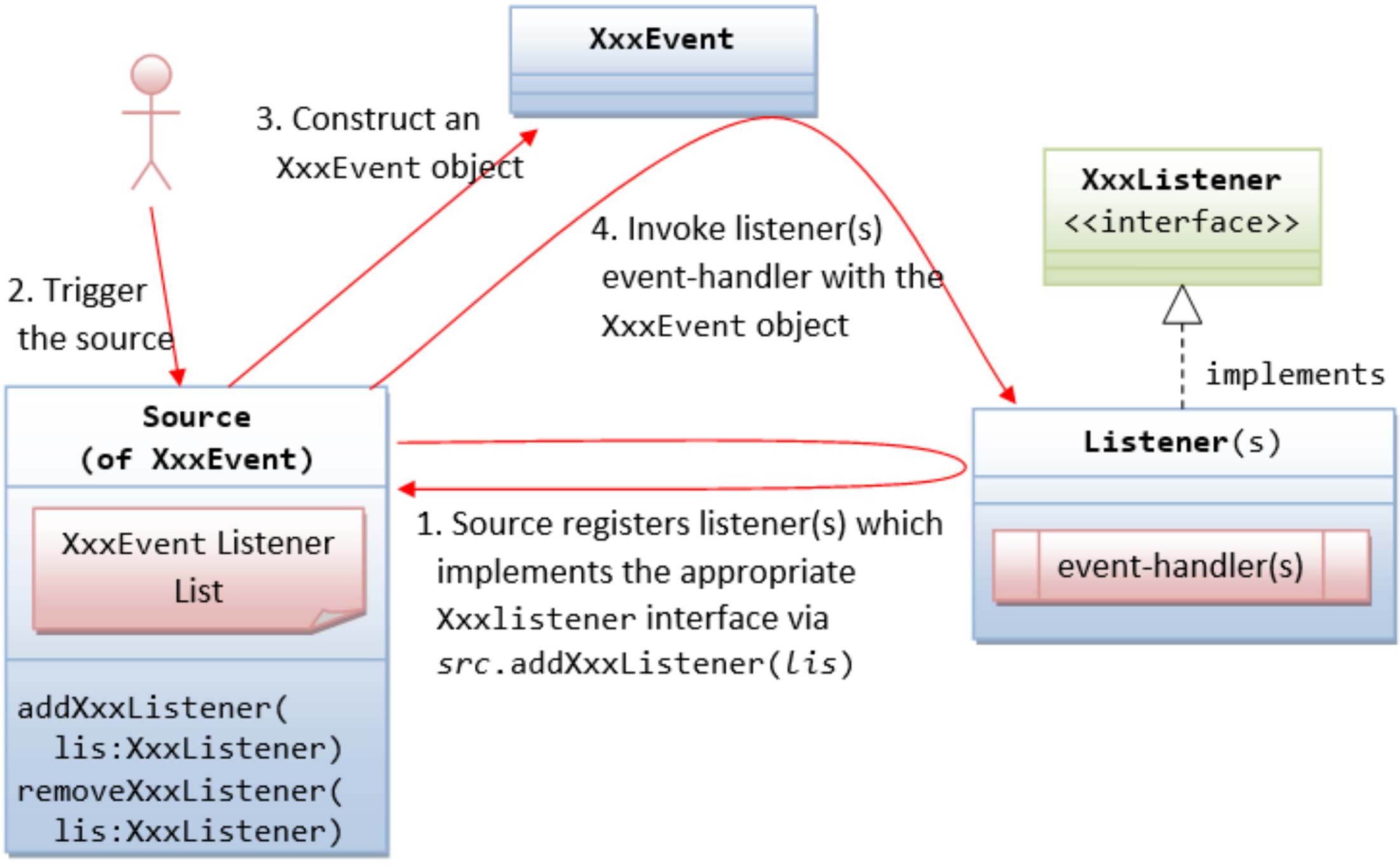
But, how to react?

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.

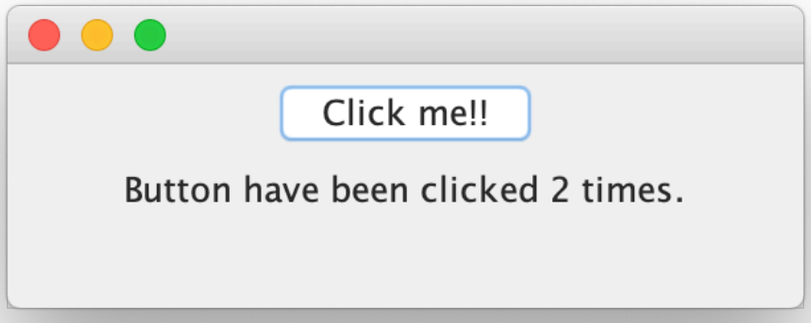
Source, Event and Listener



Event example

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

public class SimpleExample{

    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);
        b.addActionListener(listener);
    }
}
```

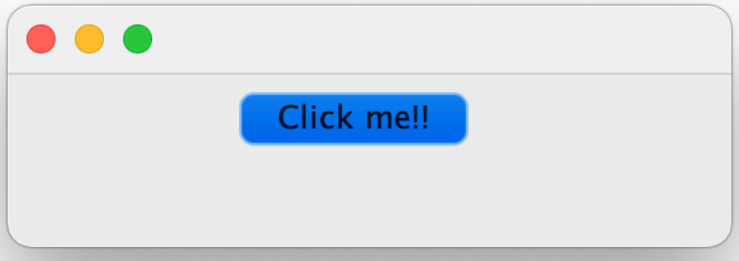
```
class MyActionListener implements ActionListener {
    Label l;
    int i;
    MyActionListener(Label l) {
        this.l = l;
    }
    public void actionPerformed(ActionEvent e) {
        l.setText("Button have been clicked " + ++i + " times.");
    }
}
```



What if the task is really heavy?

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

public class HeavyTask{

    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);
        b.addActionListener(listener);
    }
}
```

```
class MyActionListener implements ActionListener {
    Label l;
    int i;
    MyActionListener(Label l) {
            try {
                Thread.sleep(5000);
            } catch (InterruptedException e1) {
            }
                l.setText("Button have been clicked " + ++i + " times.");
    }
    public void actionPerformed(ActionEvent e) {
        l.setText("Button have been clicked " + ++i + " times.");
     }
}
```



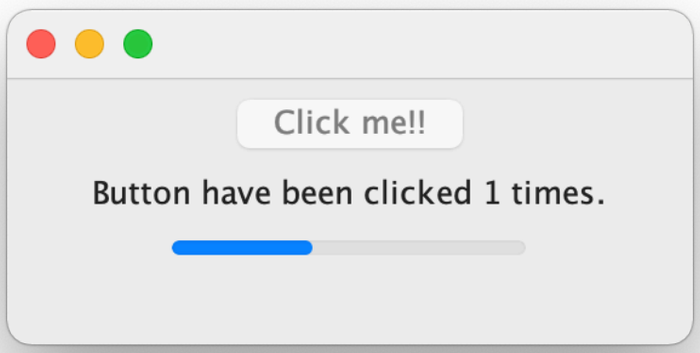
GUI with thread

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

public class GuiThread{

    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);
        b.addActionListener(listener);
    }
}
```

```
class MyActionListener implements ActionListener {
    Label l;
    int i;
    MyActionListener(Label l) {
                new Thread() {
                        public void run() {
                                try {
                                        Thread.sleep(5000);
                                } catch (InterruptedException e1) {
                                }
                                l.setText("Button have been clicked "
                                + ++i + " times.");
                        }
                }.start();
    }
    public void actionPerformed(ActionEvent e) {
        l.setText("Button have been clicked " + ++i + " times.");
    }
}
```

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!!");
        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(350, 250);
                f.setLayout(new FlowLayout());
                f.setVisible(true);

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




```
class MyActionListener implements ActionListener {
        Label l;
        JProgressBar p;
        Button b;
        int i;

        MyActionListener(Label l, JProgressBar p, Button b) {
                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);
                                        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);
                                } catch (InterruptedException e1) {
                                }
                        }
                }.start();
        }
}
```

JavaFX and SceneBuilder

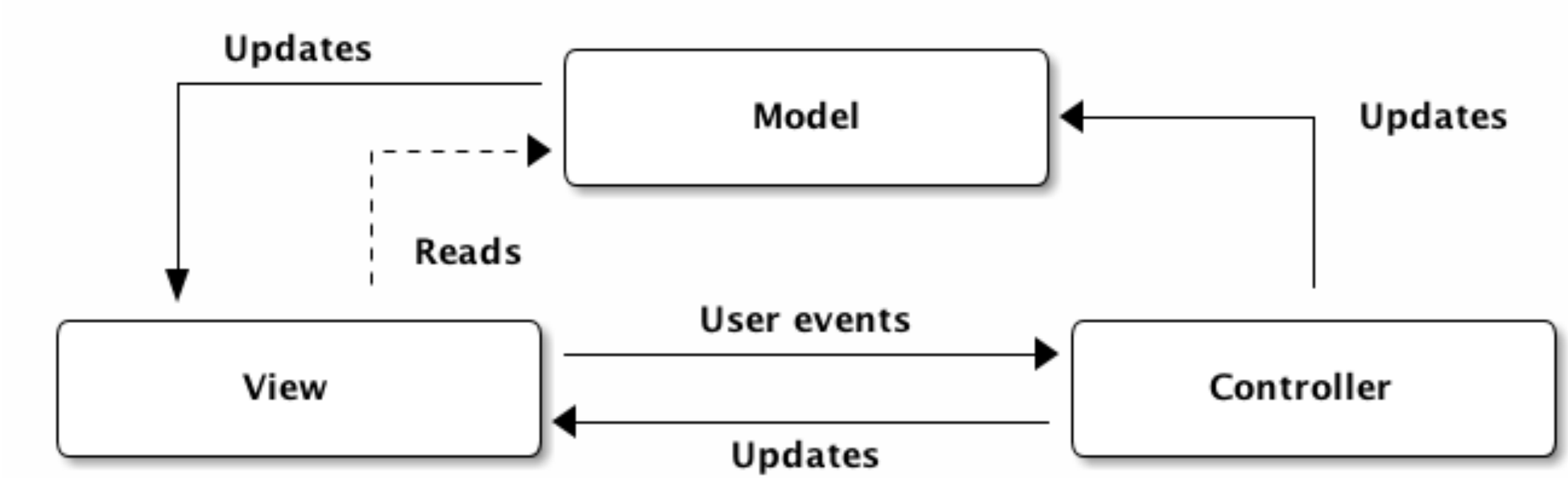
- JavaFX is the next generation client application platform for desktop, mobile and embedded systems built on Java.

- 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).



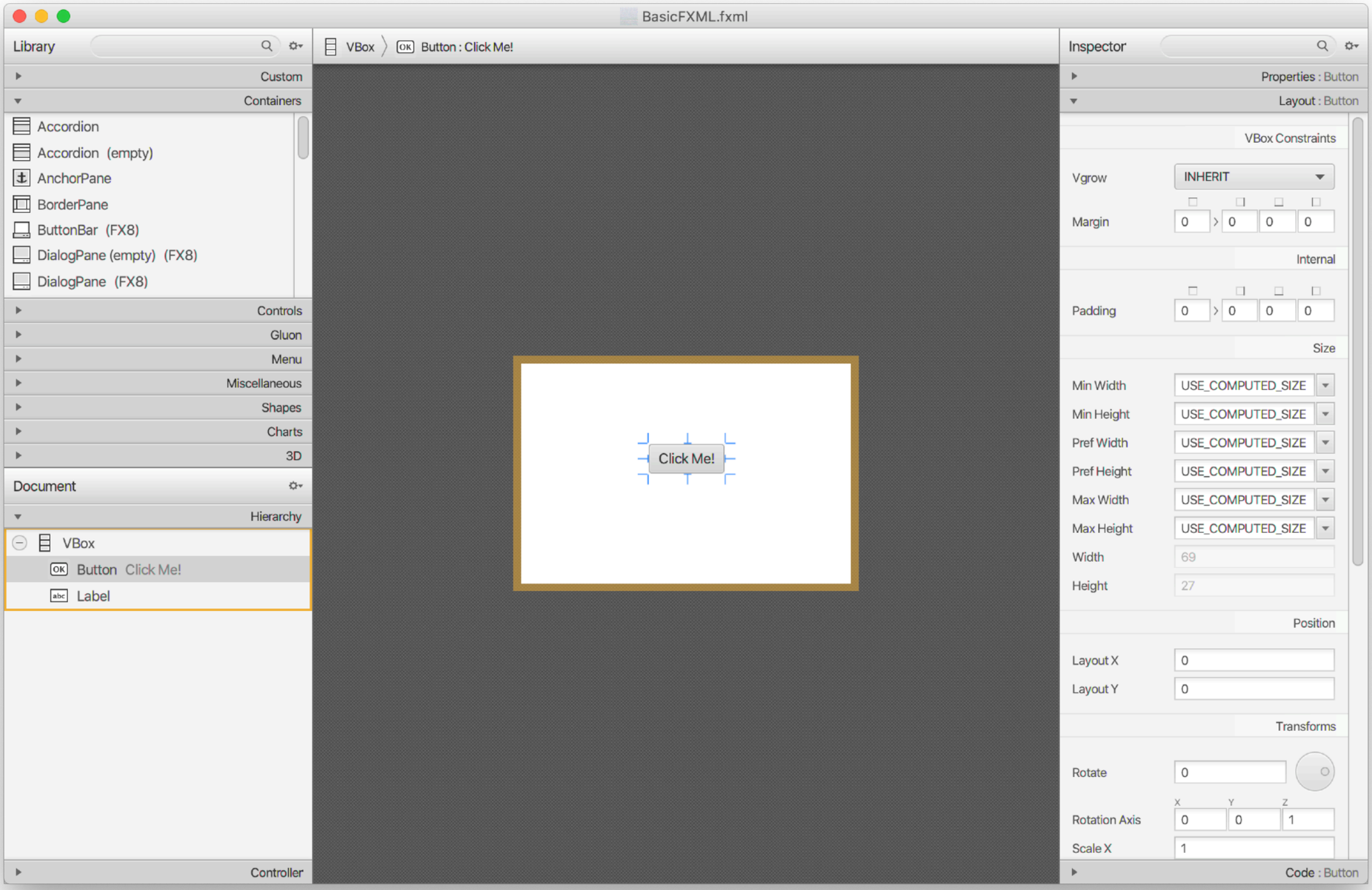
JavaFX Example — Model

```
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>
```

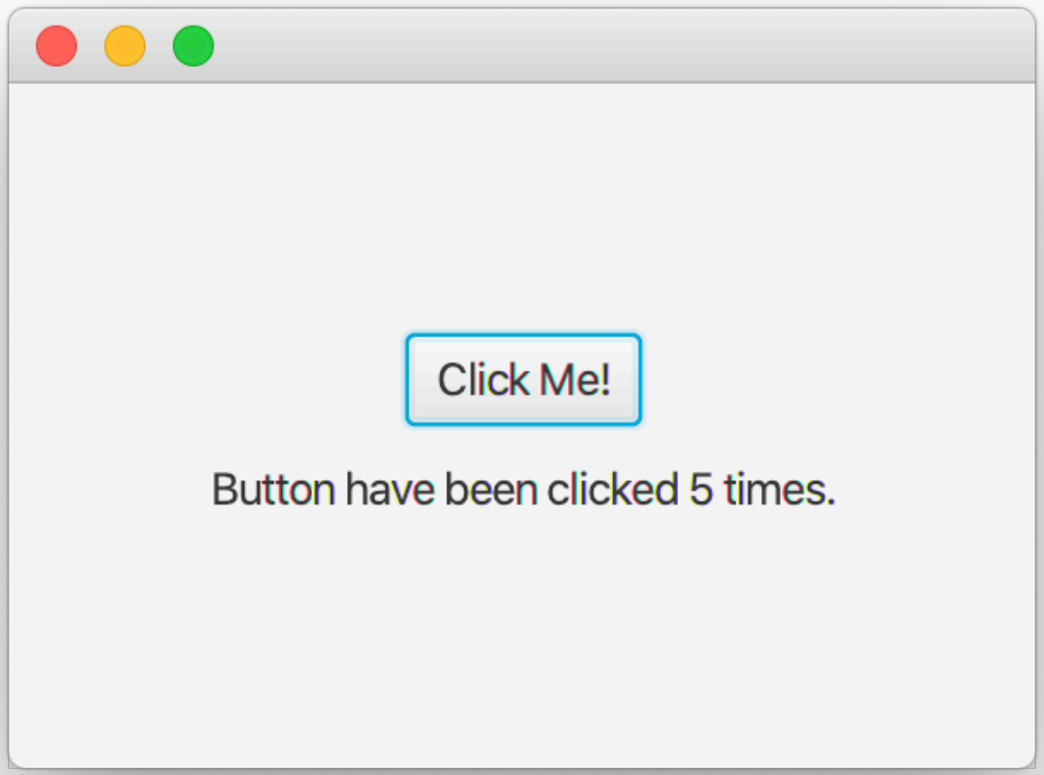
View designed with Scene Builder



JavaFX Example — Controller

```
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



Homework

Implement a GUI calculater.