Object-Oriented Programming CSE-703029

Faculty of Computer Science
Phenikaa University
Lecture 8: Applications, Applets & Swing

- Applications
- Java GUI

Applications (My Definition)

- An application is a stand-alone program that runs locally.
- Applications can use console I/O, or can have a GUI developed using the Java class library.
- Applications have no system resource restrictions.

Java GUI Application

Programming GUI :

- AWT
- Swing
- JavaFX

Java GUI Libraries

- Originally, Abstract Windowing Toolkit (**AWT**).

  Old, workable
- Recently, **Swing**.

  Nice! O-O approach

  Caution: A browser needs a current plug-in

  http://java.sun.com/j2se/1.4.2/download.html
- More Recently, **JavaFX**

  (be familiar with Web application).

  Need plug-in https://openjfx.io/openjfx-docs/

AWT Classes Layout

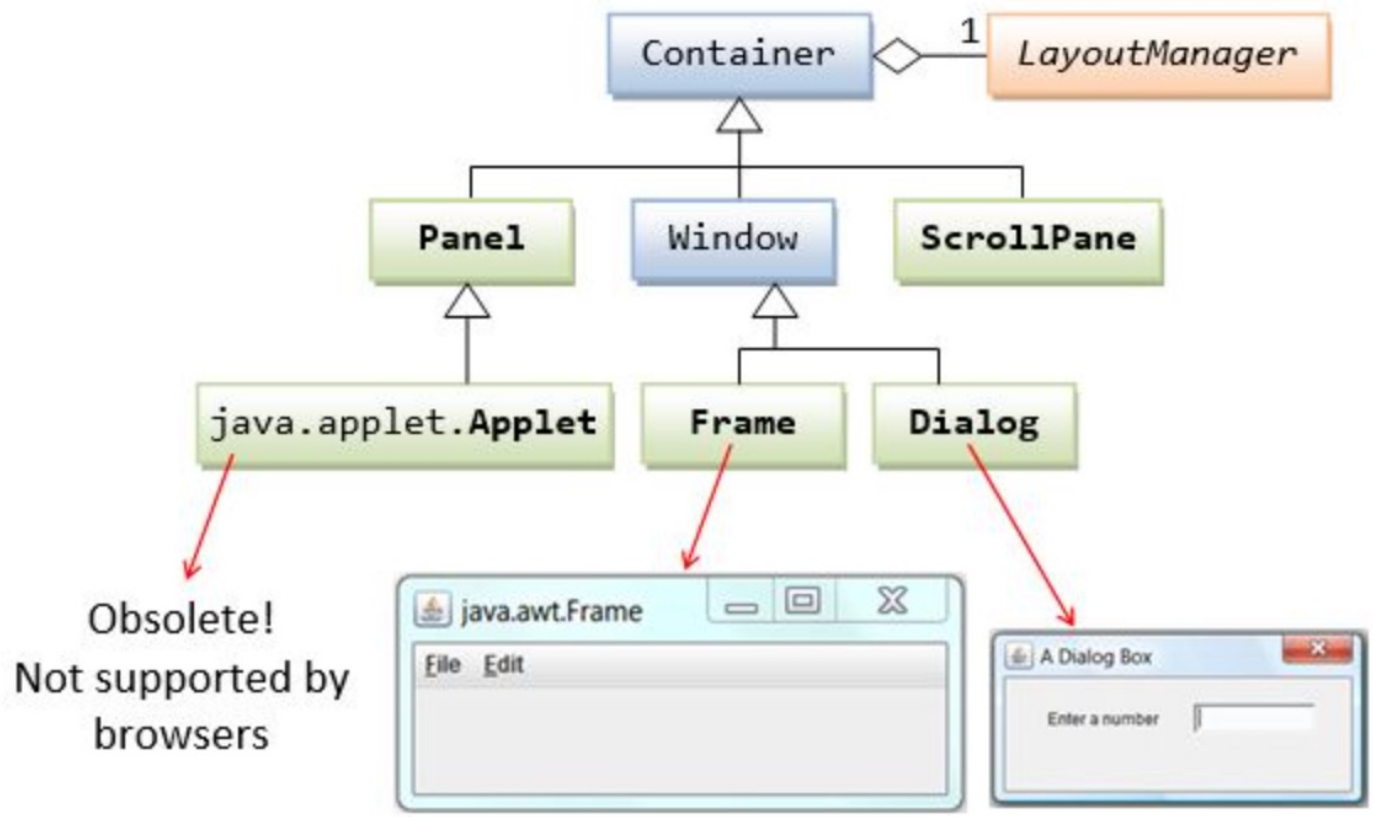


Image retrieved from
https://www3.ntu.edu.sg/home/ehchua/programming/java/j4a_gui.html

Swing Class Layout

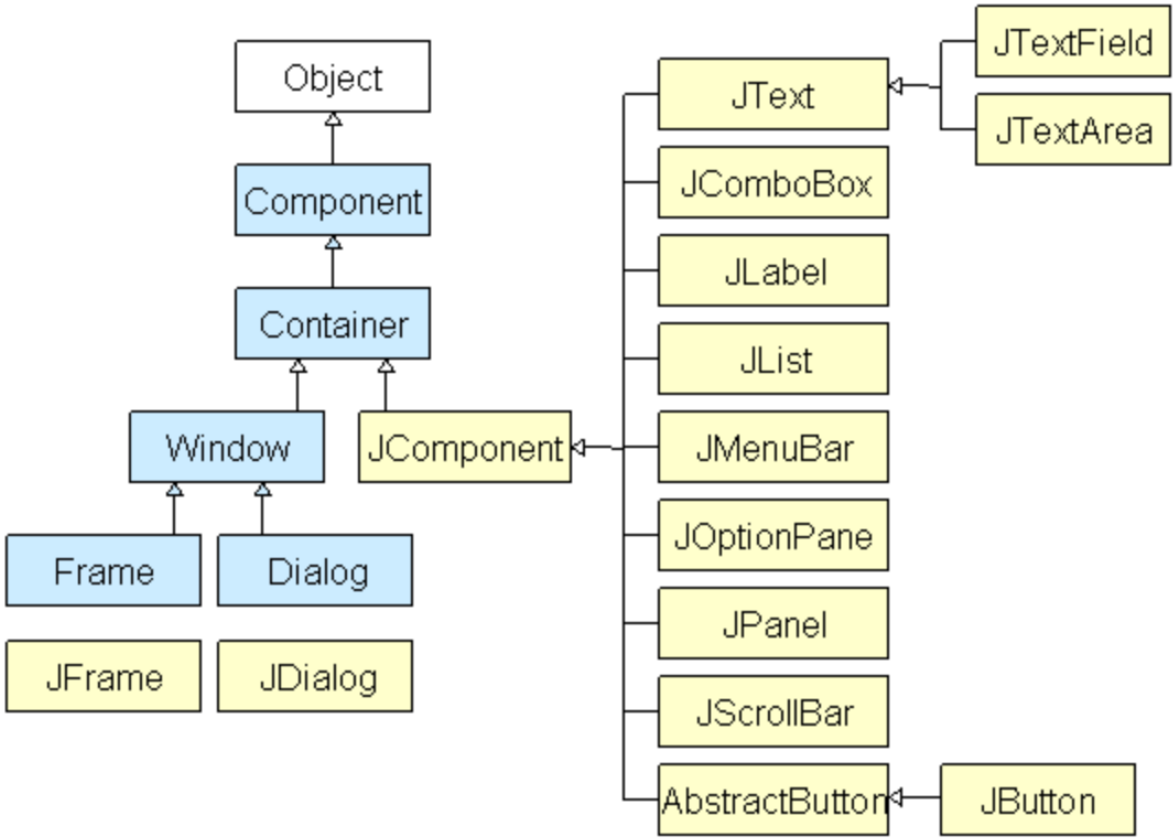


Image retrieved from
https://courses.cs.washington.edu/courses/cse331/11wi/sections/section8.html

JavaFX Class Layout

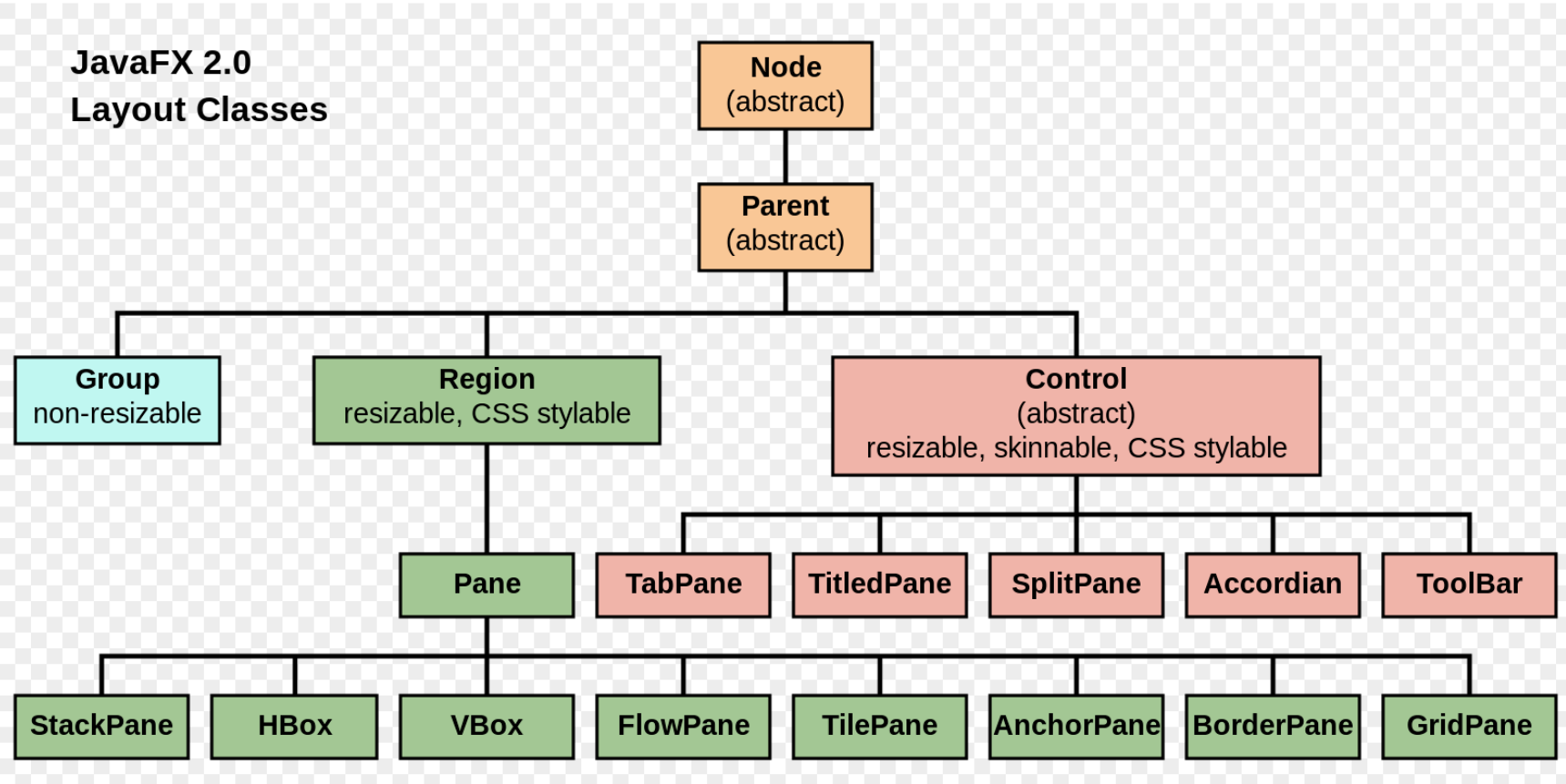
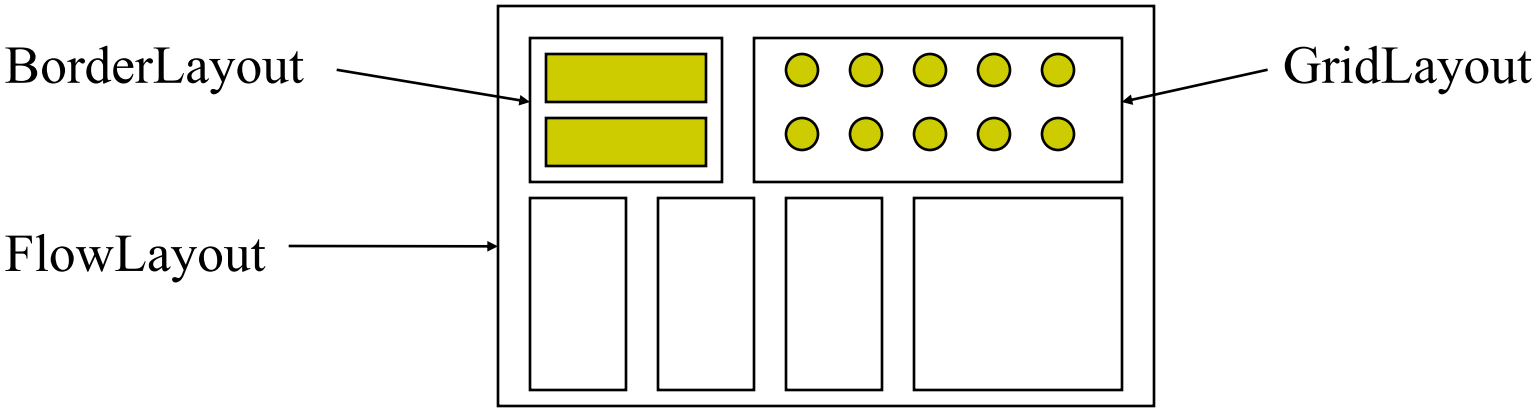


Image from https://commons.wikimedia.org/wiki/File:Javafx-layout-classes.svg

Layout Classes (cont.)

- A common strategy for GUI construction is to form the app window as a nested set of containers, each with an appropriate layout manager:



Java GUI : AWT

java.awt package : *core* AWT graphics classes

Component classes: Button, TextField, & Label.

Container classes : Frame & Panel.

+ Layout managers: FlowLayout, BorderLayout and GridLayout.

+Custom graphics classes: Graphics, Color & Font.

java.awt.event : event handling

+ Event classes: ActionEvent, MouseEvent, KeyEvent and WindowEvent,

+Event Listener Interfaces : ActionListener, MouseListener, MouseMotionListener, KeyListener & WindowListener,

+Event Listener Adapter classes : MouseAdapter, KeyAdapter & WindowAdapter.

Development Frameworks

- Some library classes are designed for use “as-is” (e.g., the collection classes).
- Other classes are intended to be reused through subclassing.
- A development framework is a set of classes that provide basic behavior for an application or applet.
- To use a framework, you write subclasses that specialize the framework classes’ behavior.

A Simple AWT

```
import java.awt.*;

public class AWTExp extends Frame {

AWTExp() {// initializing using constructor

Button b = new Button("Click "); // creating a button
b.setBounds(30,100,80,30); // setting button position on screen
add(b); // adding button into frame

setSize(300,300); // frame size 300 width and 300 height

setTitle("This is our basic AWT example"); // setting the title of Frame
setLayout(null); // no layout manager
setVisible(true); // now frame will be visible, by default it is not visible
}
 public static void main(String args[]) {
 AWTExp f = new AWTExp(); // creating instance of Frame class
    }
}
```

The Swing Event Model

- Swing components can “fire” different kinds of events.
- Each type of event is represented by a distinct class.
- When an event is fired, it is received by any listener that has “signed up” to respond to the event, by implementing the associated listener interface.
- The listener can be most anywhere (e.g., Button2 could listen for events fired by Button1).
- This “separates interface from implementation” (here interface means GUI interface!).

Java GUI : Swing

Java.awt

javax.swing

Swing Class Layout

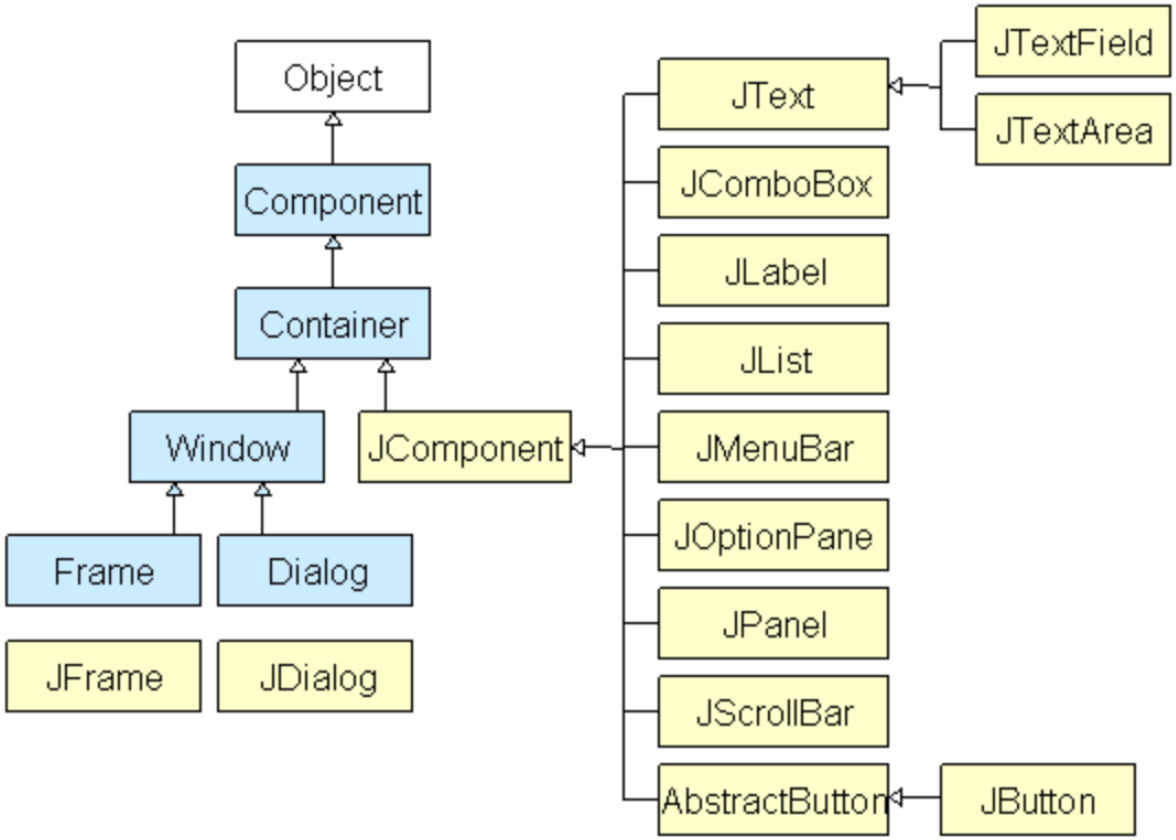


Image retrieved from
https://courses.cs.washington.edu/courses/cse331/11wi/sections/section8.html

A Simple Swing

```
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JPanel;
import javax.swing.JOptionPane;
```

A Simple Swing

```
public final class SwingExp {
    public static void main(String args[]){
    SwingExp app = new SwingExp();
    app.buildAndDisplayGui();
  }

   private void buildAndDisplayGui(){
    JFrame frame = new JFrame("Test Frame");
    buildContent(frame);
    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    frame.pack();
    frame.setVisible(true);
  }
```

A Simple Swing

```
private void buildContent(JFrame aFrame){
   JPanel panel = new JPanel();
   panel.add(new JLabel("Hello"));
   JButton ok = new JButton("OK");
   ok.addActionListener(new ShowDialog(aFrame));
   panel.add(ok);
   aFrame.getContentPane().add(panel);
}
```

A Simple Swing

```
private static final class ShowDialog implements ActionListener {
    /** Defining the dialog's owner JFrame is highly recommended. */
    ShowDialog(JFrame aFrame){
      fFrame = aFrame;
    }
    @Override public void actionPerformed(ActionEvent aEvent) {
      JOptionPane.showMessageDialog(fFrame, "This is a dialog");
    }
    private JFrame fFrame;
  }
}
```

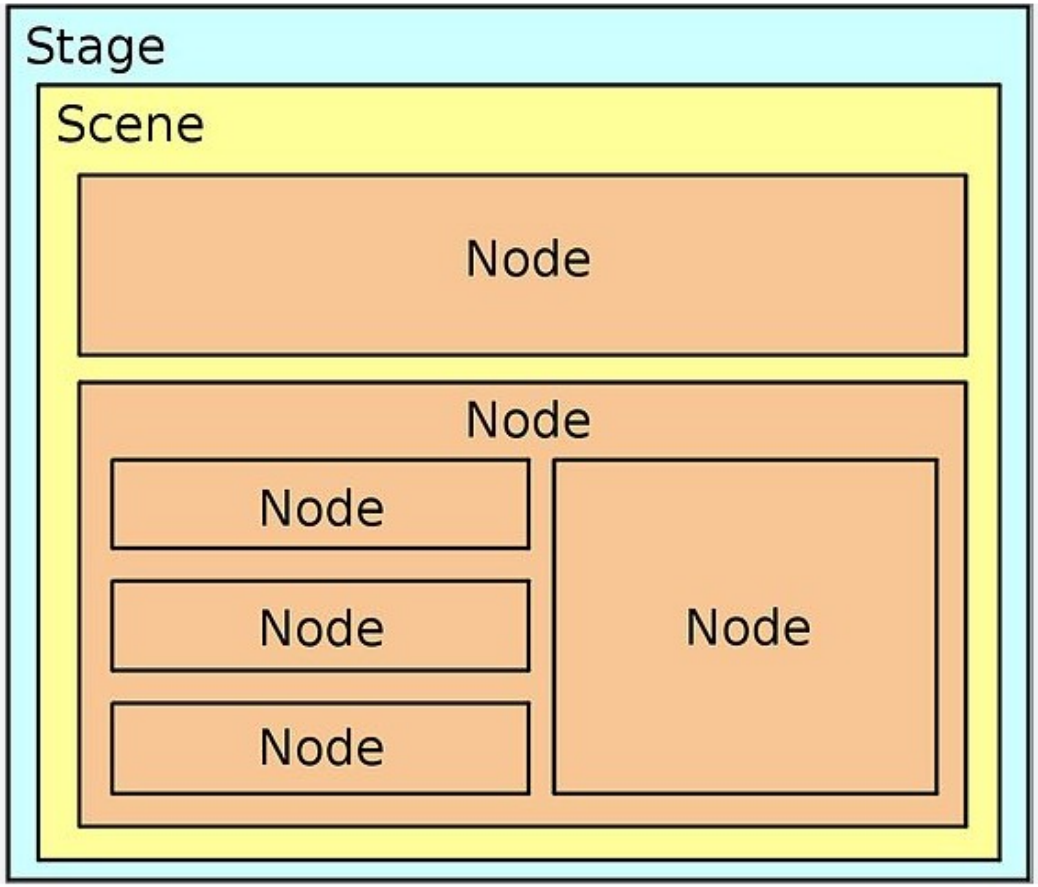
A Simple JavaFX

```
import javafx.application.Application;
import javafx.stage.Stage;
import javafx.scene.Scene;
import javafx.scene.layout.BorderPane;
```

A Simple JavaFX

```
public class JFXExp extends Application {
@Override
    public void start(Stage primaryStage) {
        try {
            BorderPane root = new BorderPane();
            Scene scene = new Scene(root,400,400);
            primaryStage.setScene(scene);
            primaryStage.show();
        } catch(Exception e) {
            e.printStackTrace();
        }
    }
```

Layout Classes



Life Cycle of JavaFX

- **init()** – is called after the application instance is created. At this point, the JavaFX API isn't ready yet, so we can't create graphical components here.
- **start(Stage stage)** – all the graphical components are created here. Also, the main thread for the graphical activities starts here.
- **stop()** – is called before the application shutdown;

Layout Classes (cont.)

- **Stage** is the main container and the entry point of the application
- **Scene** is a container for holding the UI elements, such as **GridPane, Label, TextField, Button.**
- **Action Event** and **Event Handler** to control and handle the component.

Buttons That Do Something

```
Button btn = new Button();

btn.setOnAction(new EventHandler<ActionEvent>() {
    @Override
     publicvoid handle(ActionEvent arg0) {

            System.out.println("Button clicked");
            }

} );
```

Various Kinds of **javafx.event.EventHandler**

- import javafx.event.ActionEvent;
- import javafx.event.EventHandler;