Introduction to the JAVA UI classes
Advanced HCI
IAT351

Week 3 Lecture 1

17.09.2012

Lyn Bartram

lyn@sfu.ca

SFU

SCHOOL OF INTERACTIVE ARTS + TECHNOLOGY

About JFC and Swing

- JFC – Java™ Foundation Classes
- Encompass a group of features for constructing graphical user interfaces (GUI).
- Implemented without any *native* code.
  - "Swing" is codename for lightweight JFC.
  - AWT (heavyweight), Swing (lightweight)

Heavyweight and *lightweight* components

Swing

AWT

- Platform independence
- Performance
- Side effects!
- Some extra work down the road to control timing of events

About JFC and Swing

- Swing features:
  - The Swing Components
    - Dialog, Tabbed pane, Buttons, File Chooser, ...
  - Pluggable Look and Feel
  - Accessibility API
    - Screen readers, Braille displays, ...
  - Java 2D™ API
  - Drag and Drop Between Java applications and native applications.

How to Learn Swing

- Don't even try.
- Learn general framework principles and design styles.

- Then use the API reference, and Swing Tutorials to discover detailed usage of each component.
- Java v7 API Reference available at:
  - http://docs.oracle.com/javase/7/docs/api/
  - LOTS more resources on parent site

javax.security.cert
javax.security.sasl
javax.sound.midi
javax.sound.midi.spi
javax.sound.sampled
javax.sound.sampled.spi
javax.sql
javax.sql.rowset
javax.sql.rowset.serial
javax.sql.rowset.spi
javax.swing
javax.swing.border
javax.swing.colorchooser
javax.swing.event
javax.swing.filechooser
javax.swing.plaf
javax.swing.plaf.basic
javax.swing.plaf.metal

DefaultListModel
DefaultListSelectionModel
DefaultRowSorter
DefaultRowSorter.ModelWrapper
DefaultSingleSelectionModel
FocusManager
GrayFilter
GroupLayout
ImageIcon
InputMap
InputVerifier
InternalFrameFocusTraversalPolicy
JApplet
JButton
JCheckBox
JCheckBoxMenuItem
JColorChooser
JComboBox
JComponent
JDesktopPane
JDialog
JEditorPane
JFileChooser
JFormattedTextField
JFormattedTextField.AbstractFormatter
JFormattedTextField.AbstractFormatterFactory
JFrame
JInternalFrame
JInternalFrame.JDesktopIcon
JLabel
JLayer
JLayeredPane
JList
JList.DropLocation
JMenu

javax.swing

Class JButton

java.lang.Object
  java.awt.Component
    java.awt.Container
      javax.swing.JComponent
        javax.swing.AbstractButton
          javax.swing.JButton

All Implemented Interfaces:

ImageObserver, ItemSelectable, MenuContainer, Serializable, Accessible, SwingConstants

Direct Known Subclasses:

BasicArrowButton, MetalComboBoxButton

```
public class JButton
extends AbstractButton
implements Accessible
```

An implementation of a "push" button.

Buttons can be configured, and to some degree controlled, by `Action`s. Using an `Action` with a button has many benefits beyond directly configuring a button. Refer to Swing Components Supporting `Action` for more details, and you can find more information in How to Use Actio section in *The Java Tutorial*.

See How to Use Buttons, Check Boxes, and Radio Buttons in *The Java Tutorial* for information and examples of using buttons.

Warning: Swing is not thread safe. For more information see Swing's Threading Policy.

Warning: Serialized objects of this class will not be compatible with future Swing releases. The current serialization support is appropriate for term storage or RMI between applications running the same version of Swing. As of 1.4, support for long term storage of all JavaBeans™ has added to the `java.beans` package. Please see `XMLEncoder`.

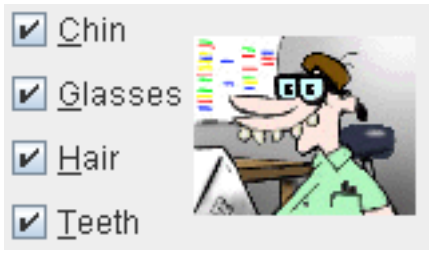
Nested Class Summary

Nested Classes

Modifier and Type	Class and Description

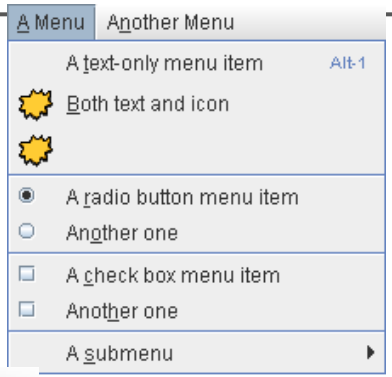
Swing Elements

- many standard GUI **`components`** such as buttons, lists, menus, and text areas, which you combine to create your program's GUI.
- Swing provides **`containers`** such as windows and tool bars.
  - top level: frames, dialogs
  - intermediate level: panel, scroll pane, tabbed pane, ...
  - other Swing components: buttons, labels, ...
- A visual index of Swing components

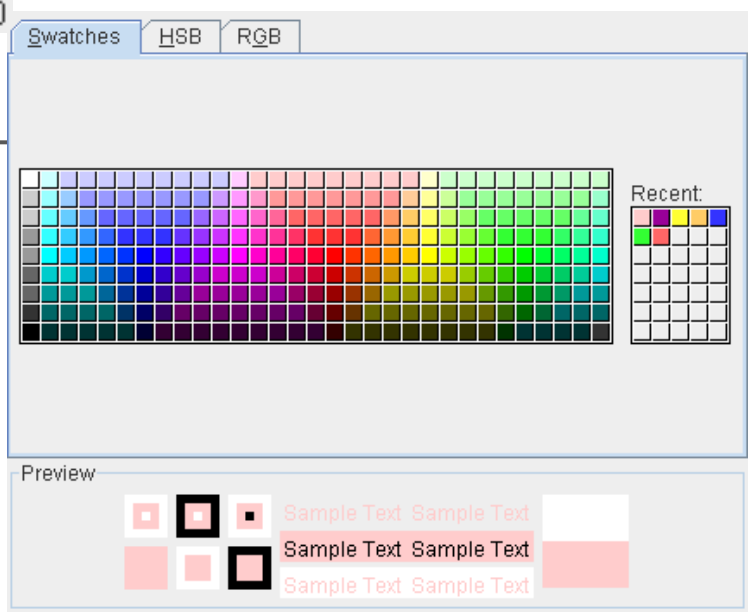


JSlider

JButton

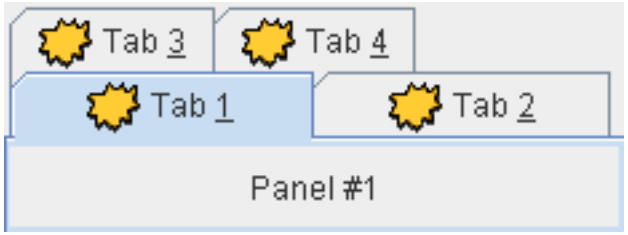


JMenu



JColorChooser

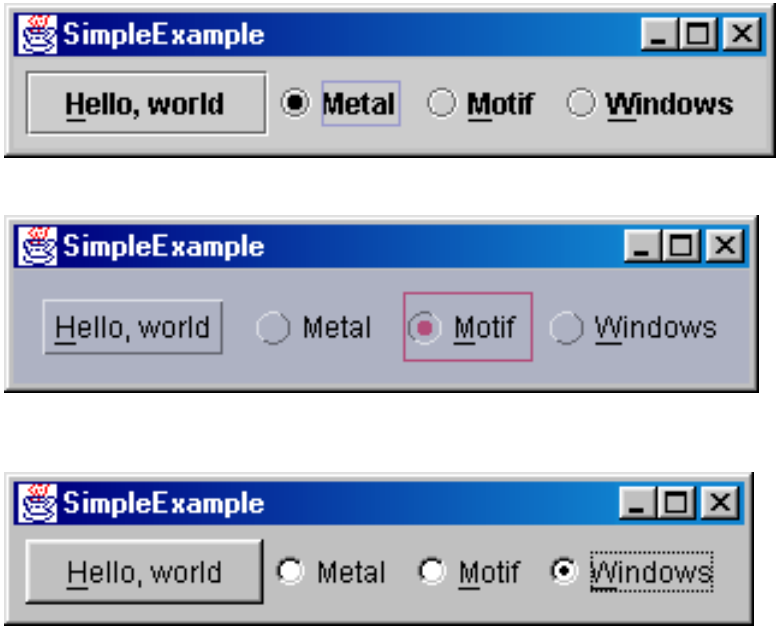
JOptionPane



JTabbedPane

Pluggable Look and Feel

- same program , different look and feel

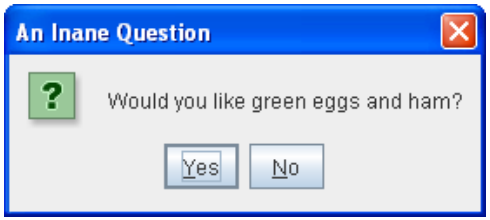


Containers

- Descendents of `the java.awt.Container` class
- can contain other components.
- Use a layout manager to position and size the components contained in them.
- **`Components`** are added to a **`Container`** using one of the various forms of its `add` method
  - layout managers
  - `panel.add(component);`
- All `Containers` are `Components`
- All `Components` are **not** `COntainers`

Top Level Containers

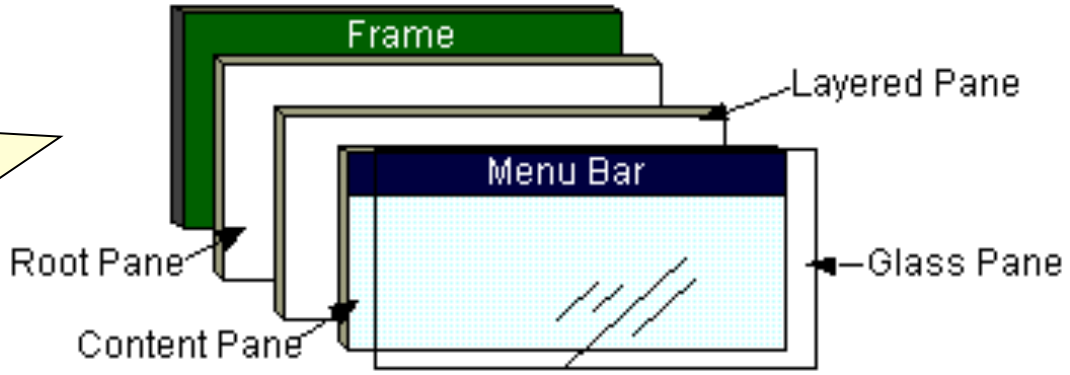
- Every program that presents a Swing GUI contains **at least one** top-level container.
- provides the support to perform painting and event-handling.
- three top-level containers:
  - **`JFrame`** (Main window)
  - **`JDialog`** (Secondary window)
  - **`JApplet`** (An applet display area within a browser window)



Top Level Containers

- To appear on screen, every GUI component must be part of a **containment hierarchy**, with a top-level container as its root.
- Each top-level container has a **content pane** that contains visible components in that top-level container's GUI.

Don't add a component directly to a top-level container.



`JFrame`

- A frame implemented as an instance of the `JFrame` class is a window that has decorations such as a border, a title and buttons for closing and iconifying the window.
  - The decorations on a frame are platform dependent.
- Applications with a GUI typically use at least one frame.

Example 1


```
import javax.swing.*;

public class MainUI {

  public static void main(String[] args) {
    JFrame frame = new JFrame("HelloWorldSwing");
    final JLabel label = new JLabel("Hello World");
    frame.getContentPane().add(label);
    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    frame.pack();
          frame.setVisible(true);
  }
```

Can't add to the JFrame directly; add to its default contentPane()

Need to tell the program to close after the window (JFrame) is closed

pack() causes a window to be sized to fit the preferred size and layouts of its sub-components

Example 2

In this example a custom frame is created

```
import javax.swing.*;

public class MainUI extends JFrame {
  public MainUI() {
    super("HelloWorldSwing");
    final JLabel label = new JLabel("Hello World");
    getContentPane().add(label);
    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    pack();
    setVisible(true);
  }
  public static void main(String[] args) {
    MainUI frame = new MainUI();
  }
}
```

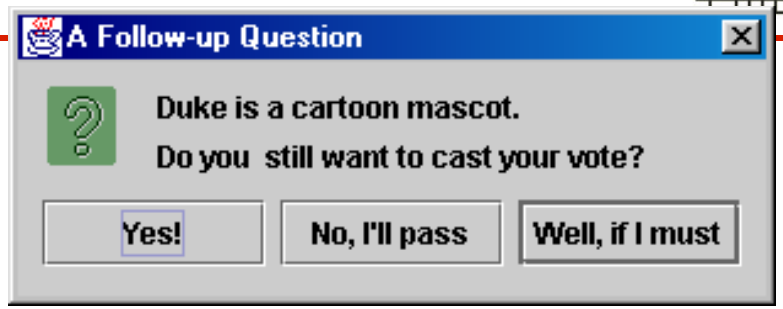
JDialog

- Every dialog is dependent on a frame
  - Destroying a frame destroys all its dependent dialogs.
  - When the frame is iconified, its dependent dialogs disappear from the screen.
  - When the frame is deiconified, its dependent dialogs return to the screen.
- A dialog can be **modal**. When a modal dialog is visible it blocks user input to all other windows in the program.

`JDialog` (cont)

- To create custom dialogs, use the **`JDialog`** class directly (as in the previous examples).
- Swing provides several standard dialogs
  - `JProgressBar, JFileChooser, JColorChooser, ...`
- The **`JOptionPane`** class can be used to create simple modal dialogs
  - icons, title, text and buttons can be customized.

Example 3




```
Object[] options = {"Yes!", "No, I'll pass",
                    "Well, if I must"};
int n = JOptionPane.showOptionDialog(
        frame, "Duke is a cartoon mascot. \n" +
        "Do you still want to cast your vote?",
        "A Follow-up Question",
        JOptionPane.YES_NO_CANCEL_OPTION,
        JOptionPane.QUESTION_MESSAGE,
        null,
        options,
        options[2]);
```

JComponent

- **`JComponent`** is the base class for all Swing components except top-level containers.
  - `JLabel, JButton, JList, JPanel, JTable, ...`
- To use a component that inherits from **`JComponent`**, it must be placed in a containment hierarchy whose base is a top-level container.

JComponent

- The **JComponent** class provides the following (partial list):
  - Pluggable Look & Feel
  - Keystroke handling
  - Tooltip support
  - An infrastructure for painting
  - Support for borders.
- All descendents of **JComponent** are also Containers
  - A **JButton** can hold text, icons etc
  - But not top-level containers

Borders

- Every `JComponent` can have one or more borders.
- The class `JBorderFactory` may be used to create standard borders

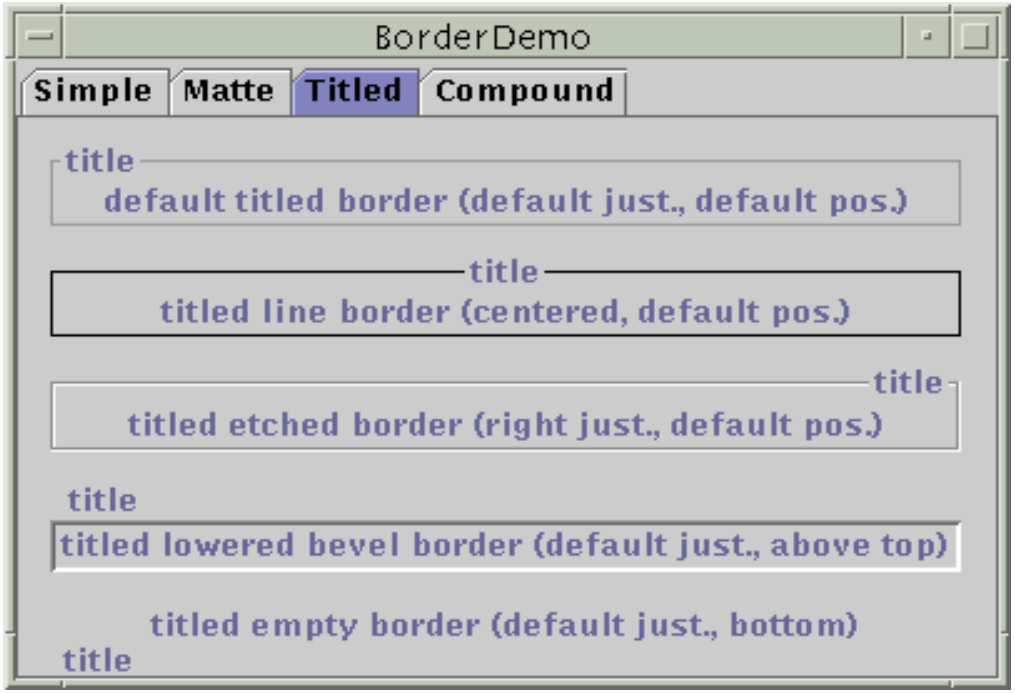
```
pane.setBorder(BorderFactory.
                    createLineBorder(Color.black));
```

- Using a compound border, you can combine any two borders, which can themselves be compound borders

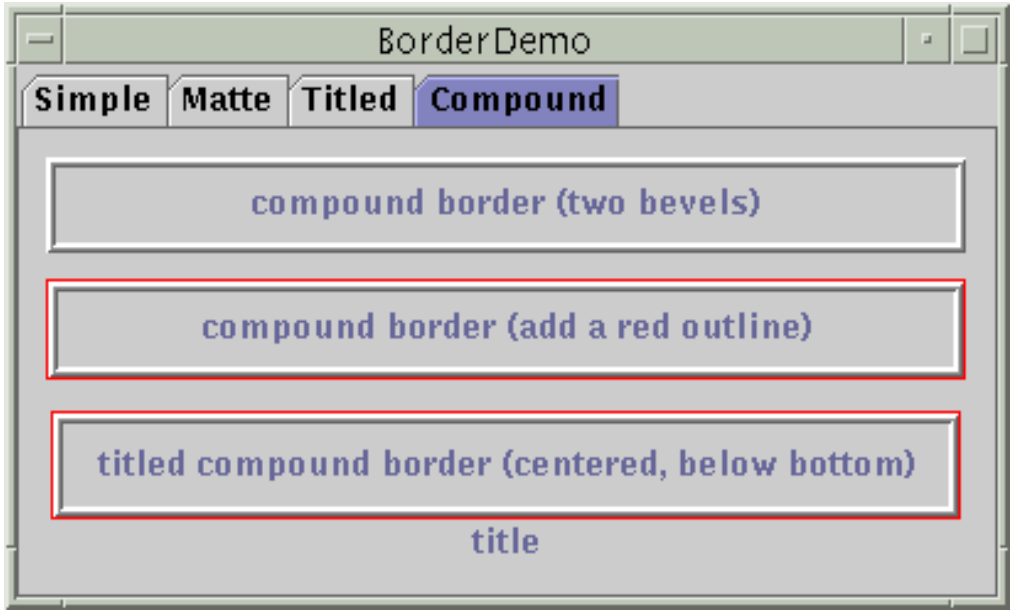
```
BorderFactory.createCompoundBorder(border1,
border2);
```

Simple Borders

Titled Borders



Compound Border



Intermediate Level Containers

- Also known as **panels** or **panes**
- Simplify the positioning of other components.
  - **`JPanel`**
  - **`JPanels`** are the default `contentPane` type
  - Use a **`JPanel`** in `setContentPane()`
- Play a visible and interactive role in a program's GUI
  - **`JScrollPane`**
  - **`JTabbedPane`**
- A panel's default layout manager is `FlowLayout`.
  - Other layout managers can easily be set

```
panel.setLayout(new BorderLayout());
```

Intermediate Level Containers (cont)

- By default, panels don't paint anything except for their background.
- By default, panels are opaque.
  - An opaque panel can be set as a top-level container's content pane.
  - transparent (non-opaque) panels draw no background.

Layout Management

- The process of determining the size and position of components.
- Layout management can be done using **absolute positioning**
  - Size and position of every component within the container must be specified.
  - Does not adjust well when the top-level container is resized.
  - Does not adjust well to differences between users and systems, such as font size.

Layout Management (cont)

- Layout management is often performed using **layout mangers**
  - Components can provide size and position *hints* to layout managers, but layout managers have the final say on the size and position of those components.

Layout Management (cont)

- Layout hints
  - Minimum, preferred and maximum size
  - X axis alignment, Y axis alignment
- Customizing layout hints
  - Invoking setter methods: `setMinimumSize`, `setAlignmentX`, ...
  - Subclassing and overriding the getter methods: `getMinimumSize`, `getAlignmentX`, ...

Layout Management (cont)

- The Java platform supplies five commonly used layout managers:
  - BorderLayout
  - BoxLayout
  - FlowLayout
  - GridLayout
  - GridBagLayout

Layout Management (cont)

- When using the *add* method to put a component in a container, the container's layout manager must be taken into account.
  - Relative position (BorderLayout)

```
panel.add(component, BorderLayout.CENTER);
```

  - Order of addition (BoxLayout, GridLayout, ...)

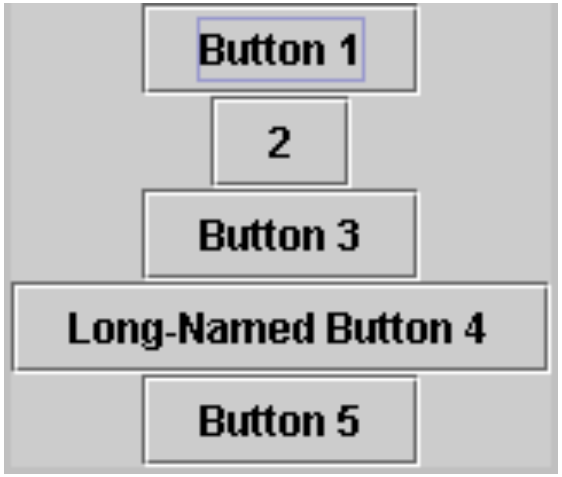
```
panel.add(component);
```

BorderLayout

- Has five areas available to hold components
  - north, south, east, west and center
- All extra space is placed in the center area
  - Only the center area is affected when the container is resized.
- Default layout manager of content panes.

BoxLayout

- Places components in a single row (left to right) or column (top to bottom).
- Respects component's maximum size and alignment hints.



FlowLayout

- Places components from left to right, starting new rows if necessary.
- Default LayoutManager of Jpanel

- Note: to avoid confusion with contentPane layout
  - Jpanel **mypanel;**
  - **mypanel.setLayout(new BorderLayout());**
  - setContentPane(**mypanel**);

GridLayout

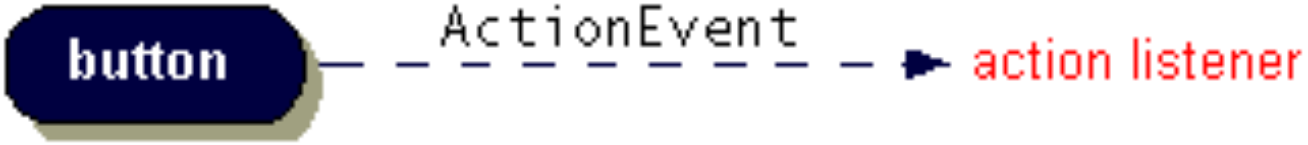
- Places components in a requested number of rows and columns.
- Components are placed left-to-right and top-to-bottom.
- Forces all components to be the same size
  - as wide as the widest component's preferred width
  - as high as the highest component's preferred height

Layout Management (cont)

- The following factors influence the amount of space between visible components in a container:
  - Layout manager
    - automatically, user specified, none
  - Invisible components
    - often used with BoxLayout
  - Empty borders
    - works best with components that have no default border such as panels and labels.

Events Handling

- Every time a user types a character or pushes a mouse button, an ***event*** occurs.
- Any object can be notified of an event by registering as an ***event listener*** on the appropriate ***event source***.
- Multiple listeners can register to be notified of events of a particular type from a particular source.



Types of Event Listeners

Act that results in event	Listener type
User clicks a button, presses Return while typing in a text field, or chooses a menu item	ActionListener
User closes a frame (main window)	WindowListener
User presses a mouse button while the cursor is over a component	MouseListener
User moves the mouse over a component	MouseMotionListener
Component becomes visible	ComponentListener
Component gets the keyboard focus	FocusListener
Table or list selection changes	ListSelectionListener

Implementing an Event Handler

- Implement a listener interface or extend a class that implements a listener interface.
- Register an instance of the event handler class as a listener upon one or more components.
- Implement the methods in the listener interface to handle the event.

Example 4


```
button.addActionListener(new ActionListener() {
  public void actionPerformed(ActionEvent e) {
    numClicks++;
    label.setText(labelPrefix + numClicks);
  }});
```

Getting started (with external libraries!)

- Swing is subdivided into `packages`:
  - **`javax.swing`, `javax.accessibility`, `javax.swing.border`** …
- At the start of your code - always
  - **`import javax.swing.*;`**
- Most Swing programs also need
  - **`import java.awt.*;`**
  - **`import java.awt.event.*;`**