Lecture 26 – Swing Introduction

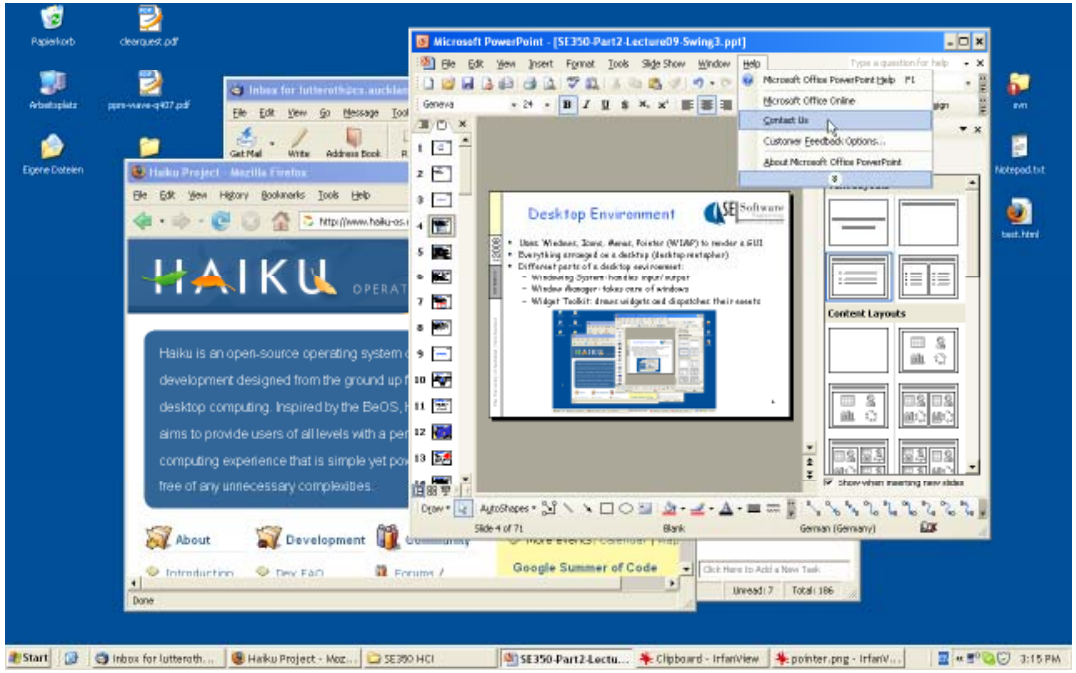
Lecturer: Gerald Weber

Outline

- Modern Windowing/GUI environment
- Welcome to Java Swing
- Handling events

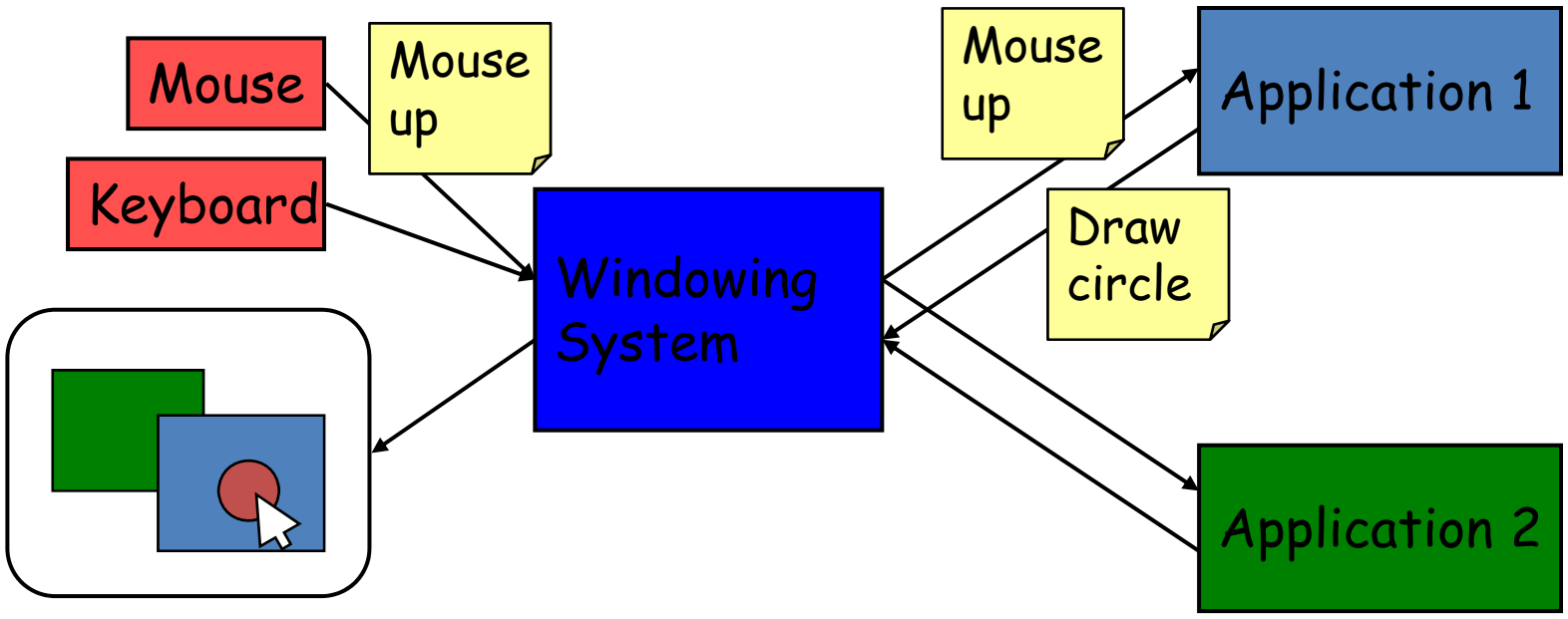
Desktop Environment

- Uses **W**indows, **I**cons, **M**enus, **P**ointer (WIMP) to render a GUI
- Everything arranged on a desktop (desktop metaphor)
- Different parts of a desktop environment (may be merged):
  - Windowing System: handles input/output
  - Widget Toolkit: draws widgets and dispatches their events
  - Window Manager: takes care of windows



Windowing System

- Manages input and output devices:
  graphics cards, screens, mice, keyboards
- Redirects data from input devices to applications
- Receives and processes drawing commands from applications
- May be able to talk to remote applications: send input events and receive drawing commands over the network



GUI Interaction Events

Primitive Pointer Events

- Mouse Moved
- Mouse Down
- Mouse Up

Primitive Keyboard Events

- Key Down
- Key Up

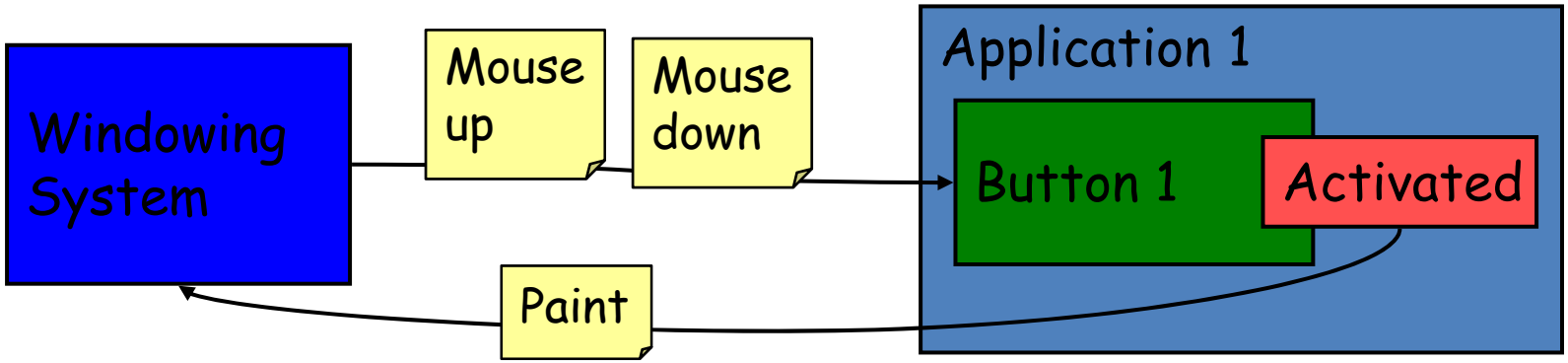
Complex Pointer Events

- Click = mouse down, mouse up
- Double Click = two clicks within a certain time
- Enter = mouse moves into a region
- Leave = mouse moves out of a region
- Hover = mouse stays in region for a period of time
- Drag and Drop = mouse down, mouse moved, mouse up

5

Input Handling in Widgets

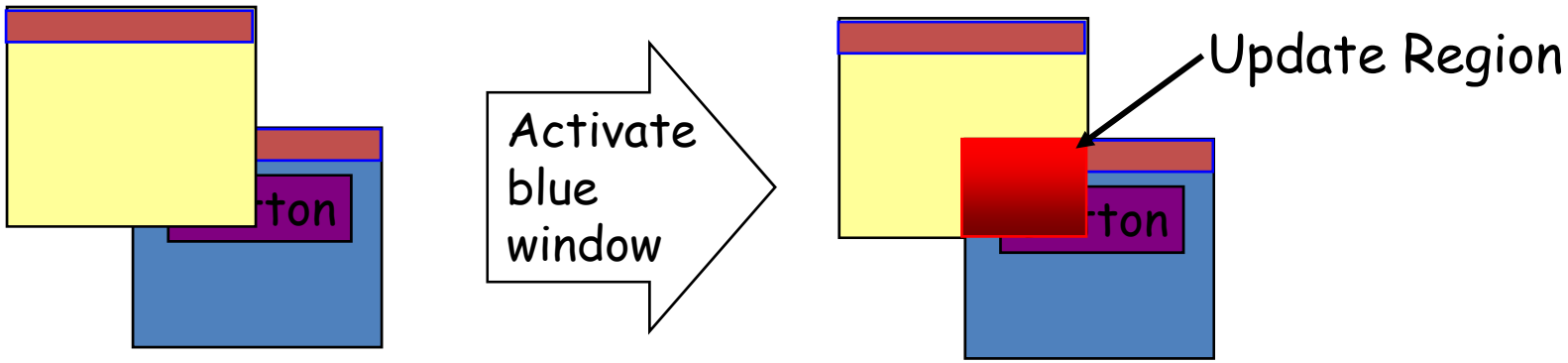
- Widgets are objects that occupy screen space.
- Input events are dispatched to the right widgets by windowing system and/or toolkit
- Keyboard events are sent to widget with **input focus** in active window
- Widgets have handlers for input events;
  they can translate simple input events into more complex, specific ones (e.g. “activated”)
- Developers can set event handlers for widgets, which invoke application logic



Rendering of Widgets

Widgets have a visual representation

- Widgets define “paint” event handler: draws the widget by sending commands to the windowing system
- Widget gets “paint” (or “update”) events from the windowing system (possibly through toolkit)
  - Often not complete redrawing, but “update region”
  - Redrawing the update region is achieved with clipping
- Application can send “invalidate” events to the windowing system if redrawing necessary (potentially triggers paint events)



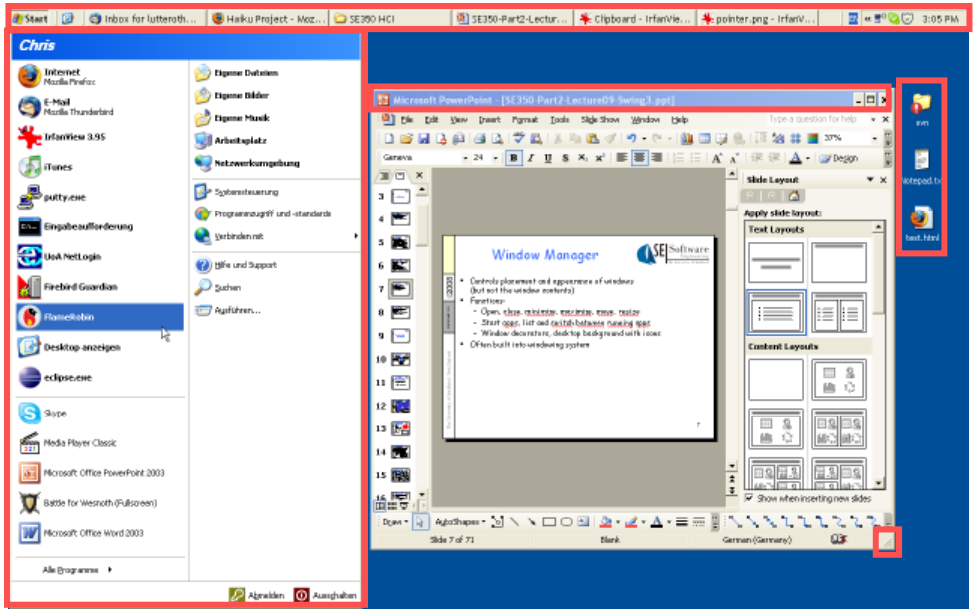
The GUI Event Loop

1. GUI application is started
2. Widgets are set up
3. Event loop is started
4. Wait for events from the windowing system (event queue)
5. Dispatch each event to the right widget
   - Input event: call appropriate event handler ($\rightarrow$ call to application logic)
   - Paint event: call paint method
6. Go back to 4.

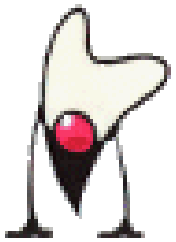
$\rightarrow$ Event-Driven Programming

Window Manager

- Controls placement and appearance of windows (but not the window contents)
  - Open, close, minimize, maximize, move, resize
  - Start apps, list and switch between running apps
  - Window decorators, desktop background with icons
- Often built into windowing system
- Implemented using a widget toolkit



Introduction to Java Swing



AWT vs. Swing

Abstract Windowing Toolkit (AWT)

- Original Java GUI toolkit
- Wrapper API for native GUI components
- Lowest-common denominator for all Java host environments

Swing

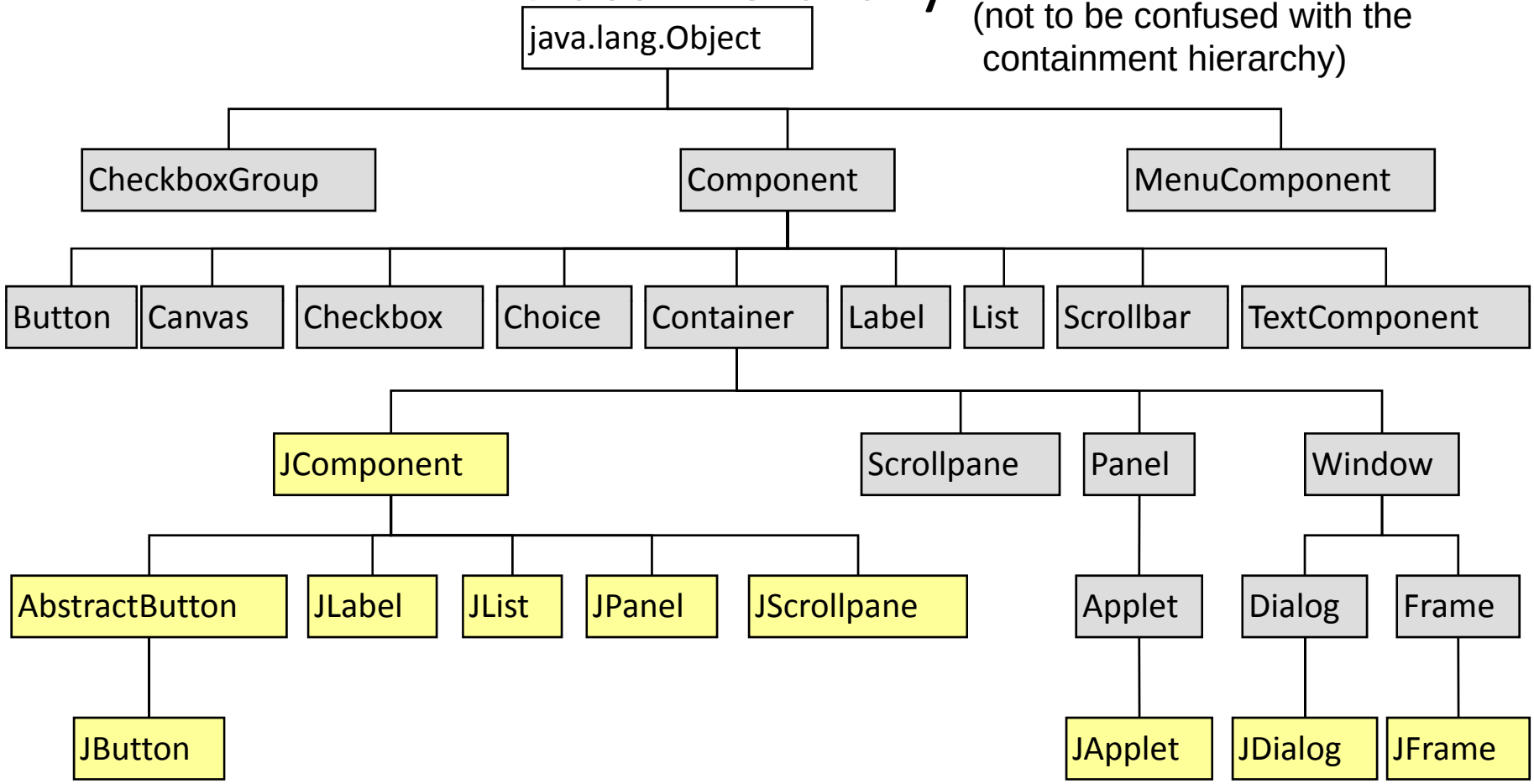
- Implemented entirely in Java on top of AWT
- Richer set of GUI components
- Pluggable look-and-feel support

Swing Design Principles

- Each GUI is built as **containment hierarchy** of widgets
  (i.e. there is a parent-child nesting relation between them)
- Event objects and event listeners
  - **Event object**: is created when event occurs (e.g. click), contains additional info (e.g. mouse coordinates)
  - **Event listener**: object implementing an interface with an event handler method that gets an event object as argument
- Separation of Model and View:
  - **Model**: the data that is presented by a widget
  - **View**: the actual presentation on the screen

Partial AWT and Swing Class Hierarchy

(not to be confused with the containment hierarchy)



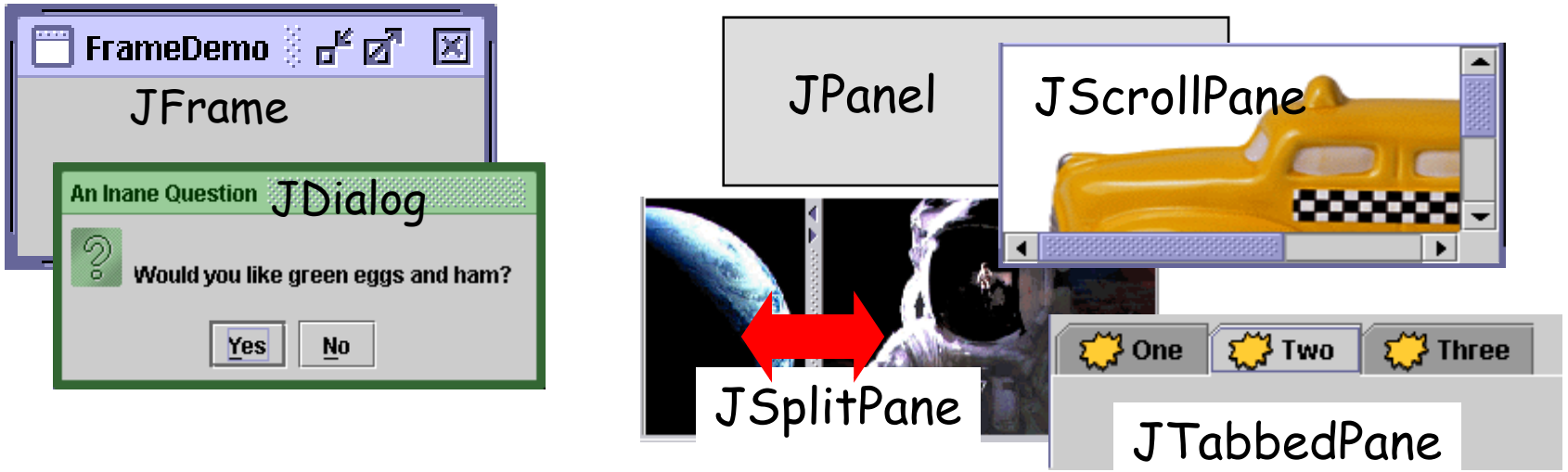
java.awt.*

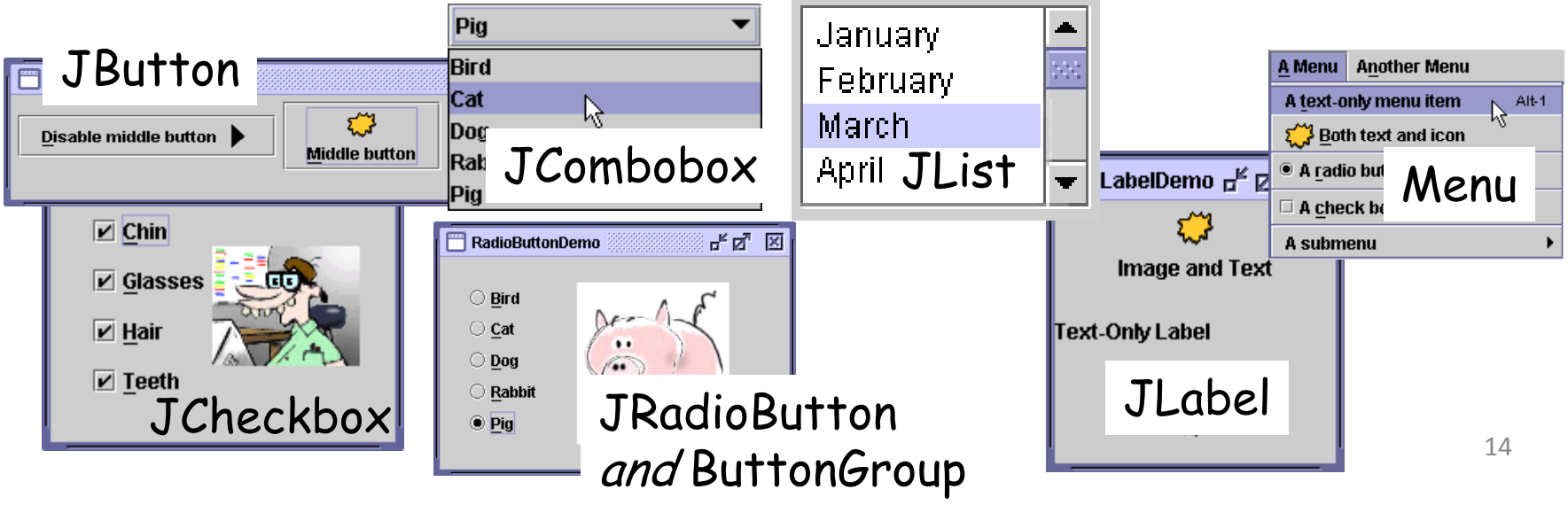
javax.swing.*

Swing Widgets (more on this next lecture)

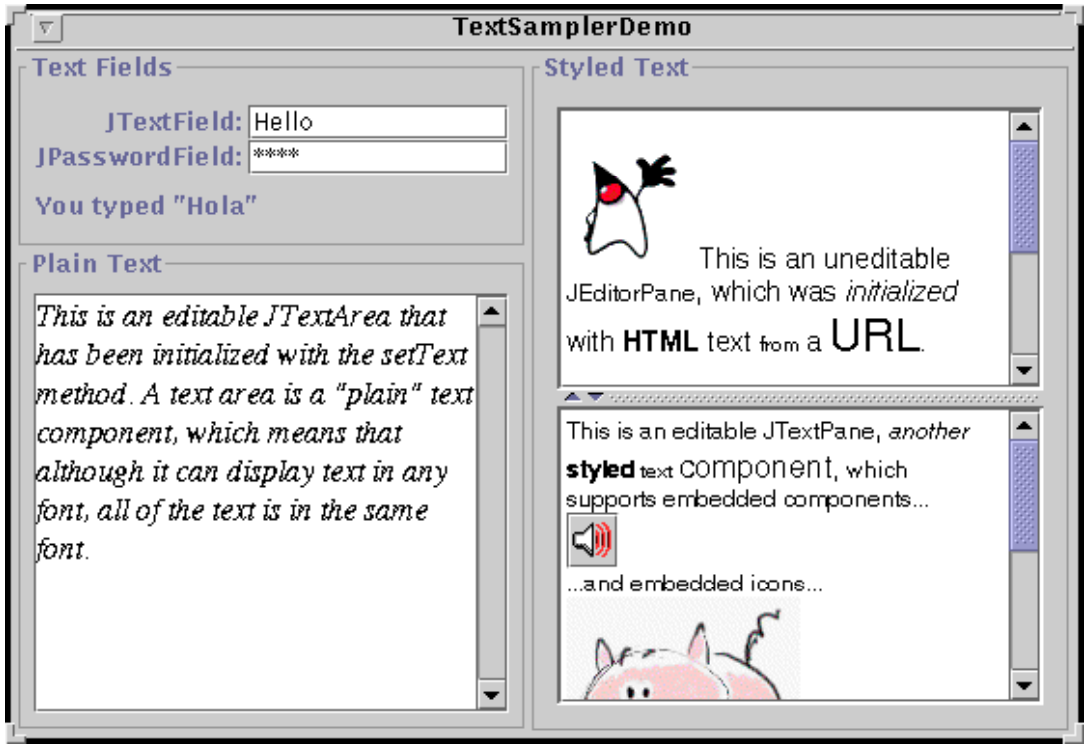
Top-Level Containers

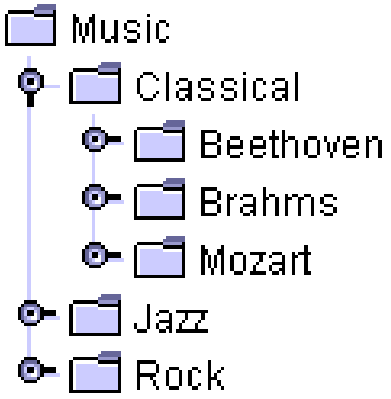
General-Purpose Containers





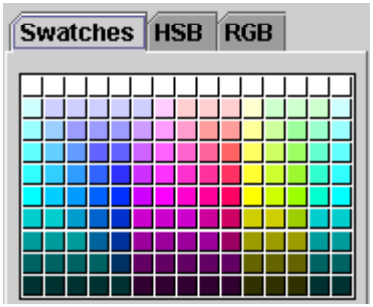
More Swing Widgets





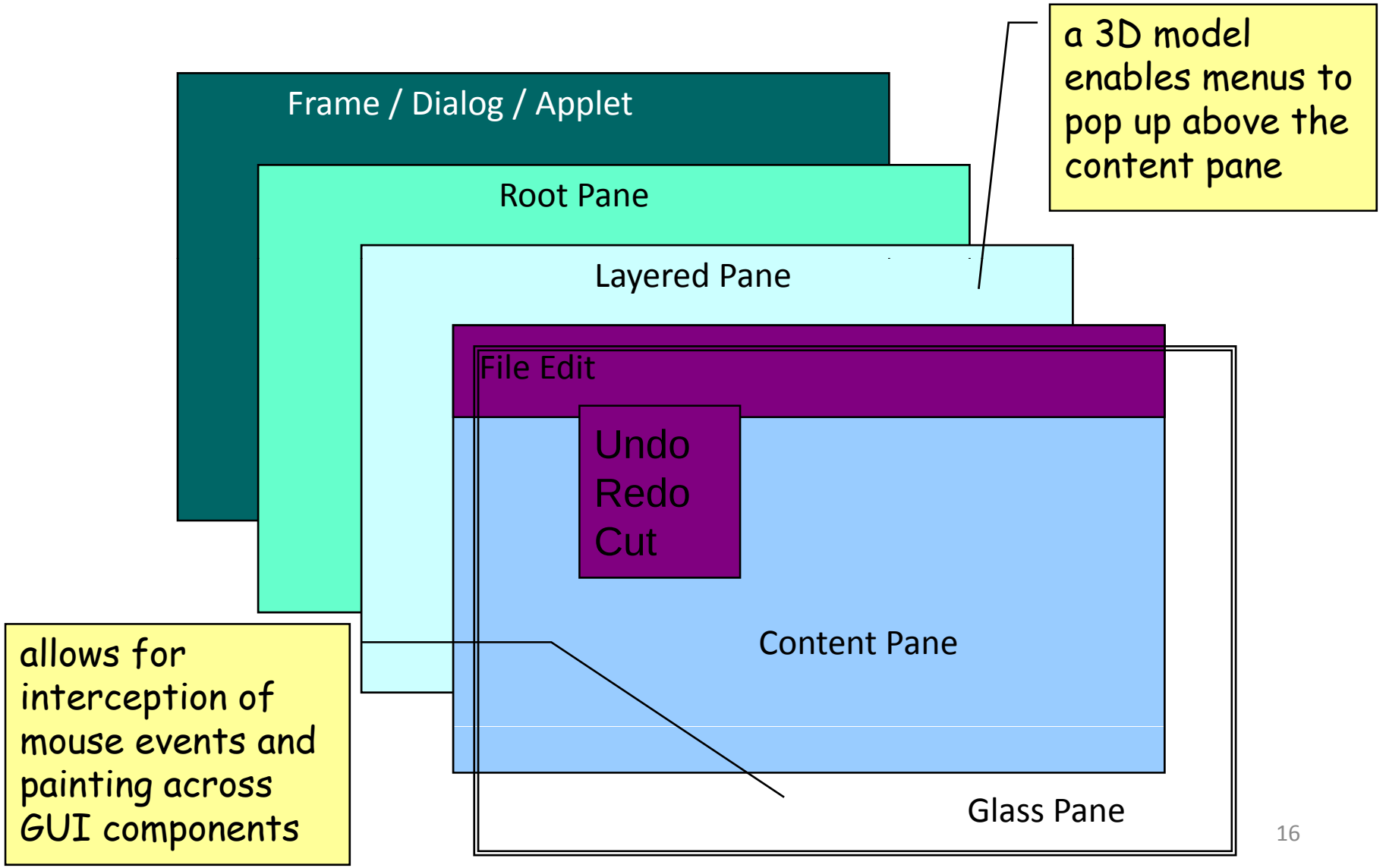
First Name	Last Name	Favorite Food
Jeff	Dinkins	
Ewan	Dinkins	
Amy	Fowler	
Hania	Gajewska	
David	Geary	

JTable

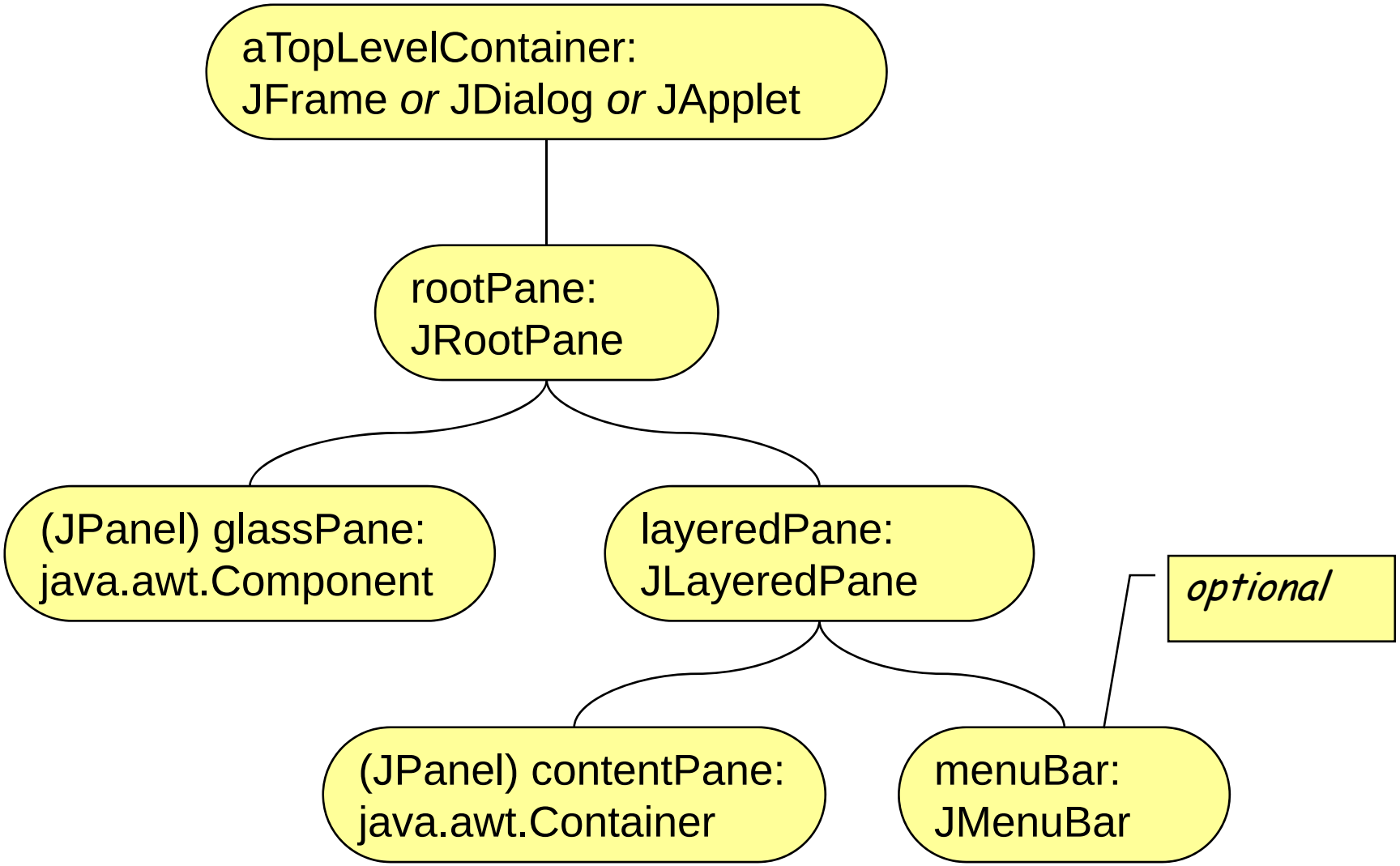


JColorChooser

The Initial Swing GUI Containment Hierarchy

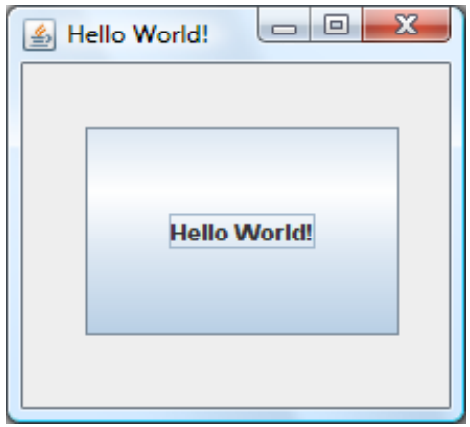


The Initial Swing GUI Containment Hierarchy



Swing Hello World

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

public class HelloWorld {
   public static void main(String[] args) {
        JFrame frame = new JFrame("Hello World!");
        frame.setSize(220, 200);
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

        Container contentPane = frame.getContentPane();
        contentPane.setLayout(null);

        JButton button = new JButton("Hello World!");
        button.setLocation(30, 30);
        button.setSize(150, 100);
        contentPane.add(button);

        frame.setVisible(true);
   }
}
```



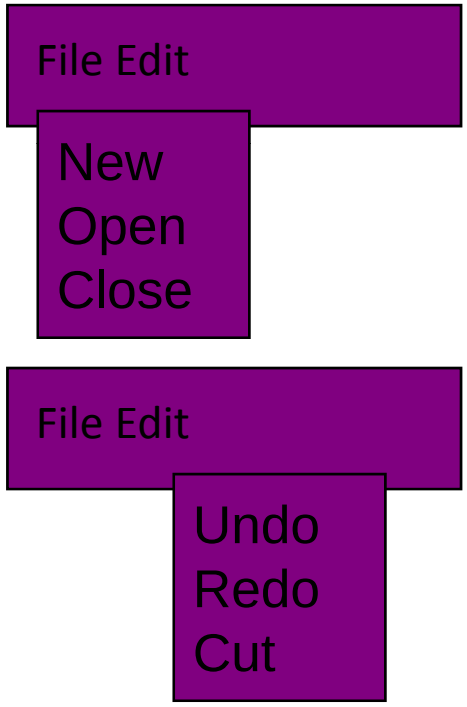
Swing Hello World with Events

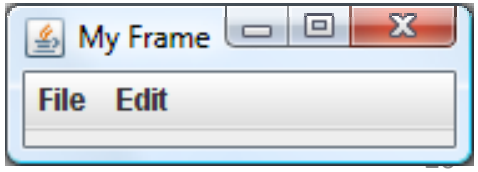
```
...
public class HelloWorld {
   public static void main(String[] args) {
      ...
      JButton button = new JButton("Hello World!");
      button.addActionListener(new MyActionListener());
      ...
   }
}
```

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

public class MyActionListener implements ActionListener {
   public void actionPerformed(ActionEvent e) {
      Toolkit.getDefaultToolkit().beep();
   }
}
```

Containment Hierarchy of a Menu

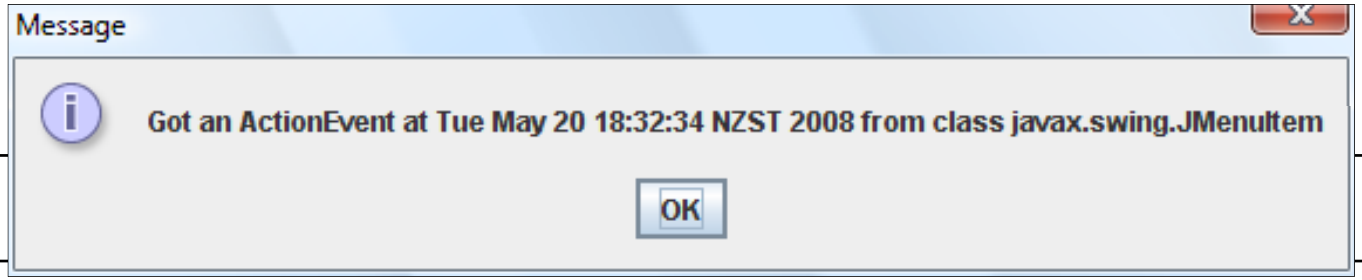
```
...
public class MenuExample {
  public static void main(String[] args) {
    JFrame frame = new JFrame("My Frame");
    frame.setDefaultCloseOperation(
      JFrame.EXIT_ON_CLOSE);

    JMenu fileMenu = new JMenu("File");
    fileMenu.add(new JMenuItem("New"));
    fileMenu.add(new JMenuItem("Open"));
    fileMenu.add(new JMenuItem("Close"));

    JMenu editMenu = new JMenu("Edit");
    editMenu.add(new JMenuItem("Undo"));
    editMenu.add(new JMenuItem("Redo"));
    editMenu.add(new JMenuItem("Cut"));

    JMenuBar menubar = new JMenuBar();
    menubar.add(fileMenu);
    menubar.add(editMenu);

    frame.setJMenuBar(menubar);
    frame.setVisible(true);
} }
```





Handling Menu Events

```
...
public class MenuExample {
  static JFrame frame;
  public static void main(String[] args) {
    ...
    JMenuItem item = new JMenuItem("Close");
    item.addActionListener(new MenuActionListener());
    fileMenu.add(item);
    ...
} }
```




```
...
public class MenuActionListener implements ActionListener {
   public void actionPerformed(ActionEvent e) {
      JOptionPane.showMessageDialog(MenuExample.frame,
          "Got an ActionEvent at " + new Date(e.getWhen())
            + " from " + e.getSource().getClass());
}  }
```

Defining Event Listeners with Anonynous Classes

```
...
public class MenuExample {
  public static void main(String[] args) {
    final JFrame frame = new JFrame("My Frame");
    ...
    JMenuItem item = new JMenuItem("Close");
    item.addActionListener(new ActionListener() {
        public void actionPerformed(ActionEvent e) {
          int n = JOptionPane.showOptionDialog(frame,...
        }
    });
    ...
} }
```

- Use **`new Classname() {…}`** or **`new Interfacename(){…}`** to create a single object of an anonymous subclass of the given class/interface
- Anonymous classes can access **`final`** variables of their context (i.e. **`final`** variables of the method or class they are created in)

Different Kinds of Swing Events

Low-level events

- **`MouseEvent`**: Component got mouse-down, mouse-move, etc.
- **`KeyEvent`**: Component got key-press, key-release, etc.
- **`ComponentEvent`**: Component resized, moved, etc.
- **`ContainerEvent`**: Container's contents changed because a component was added or removed
- **`FocusEvent`**: Component got focus or lost focus
- **`WindowEvent`**: Window opened, closed, etc.

High-level semantic events

- **`ActionEvent`**: Main action of control invoked (e.g. JButton click)
- **`AdjustmentEvent`**: Value was adjusted (e.g. JScrollBar moved)
- **`ItemEvent`**: Item was selected or deselected (e.g. in JList)
- **`TextEvent`**: Text in component has changed (e.g in JTextField)

Events, Listeners, Adapters and Handler Methods

Event	Listener / Adapter	Handler Methods
`ActionEvent`	`ActionListener`	`actionPerformed`
`AdjustmentEvent`	`AdjustmentListener`	`adjustmentValueChanged`

`MouseEvent`	`MouseListener`
	`MouseAdapter`
		`mouseClicked`
		`mouseEntered`
		`mouseExited`
		`mousePressed`
		`mouseReleased`

`KeyEvent`	`KeyListener`
	`KeyAdapter`
		`keyPressed`
		`keyReleased`
		`keyTyped`

`ComponentEvent`	`ComponentListener`
	`ComponentAdapter`
		`componentShown`
		`componentHidden`
		`componentMoved`
		`componentResized`

Adapter classes with empty methods for Listener interfaces with >1 methods

Summary

- Desktop environments consist of:
  - Windowing System: handles input/output
  - Widget Toolkit:
    draws widgets and dispatches their events
  - Window Manager: takes care of windows
- Swing is a widget toolkit for Java
  - GUI as containment hierarchy of widgets
  - Event objects and event listeners

References:

http://java.sun.com/docs/books/tutorial/uiswing/

http://www.javabeginner.com/java-swing-tutorial.htm