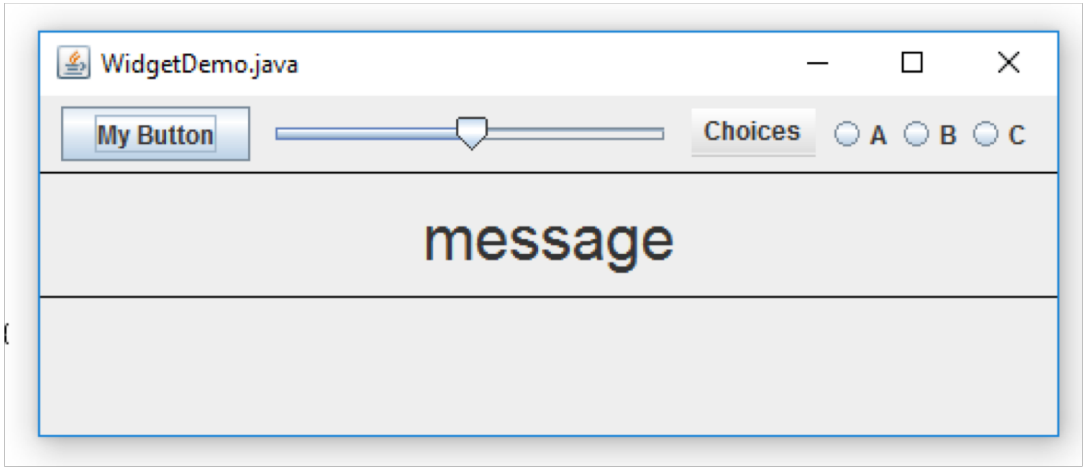
Widgets

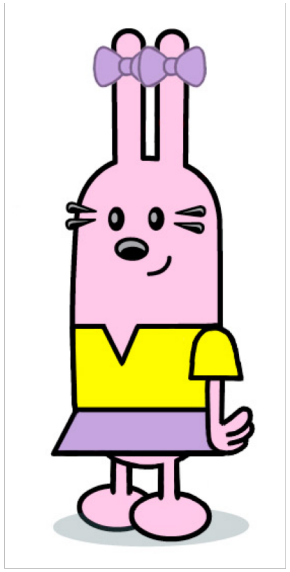
Widgets

Widget Toolkits

User Interface Widget

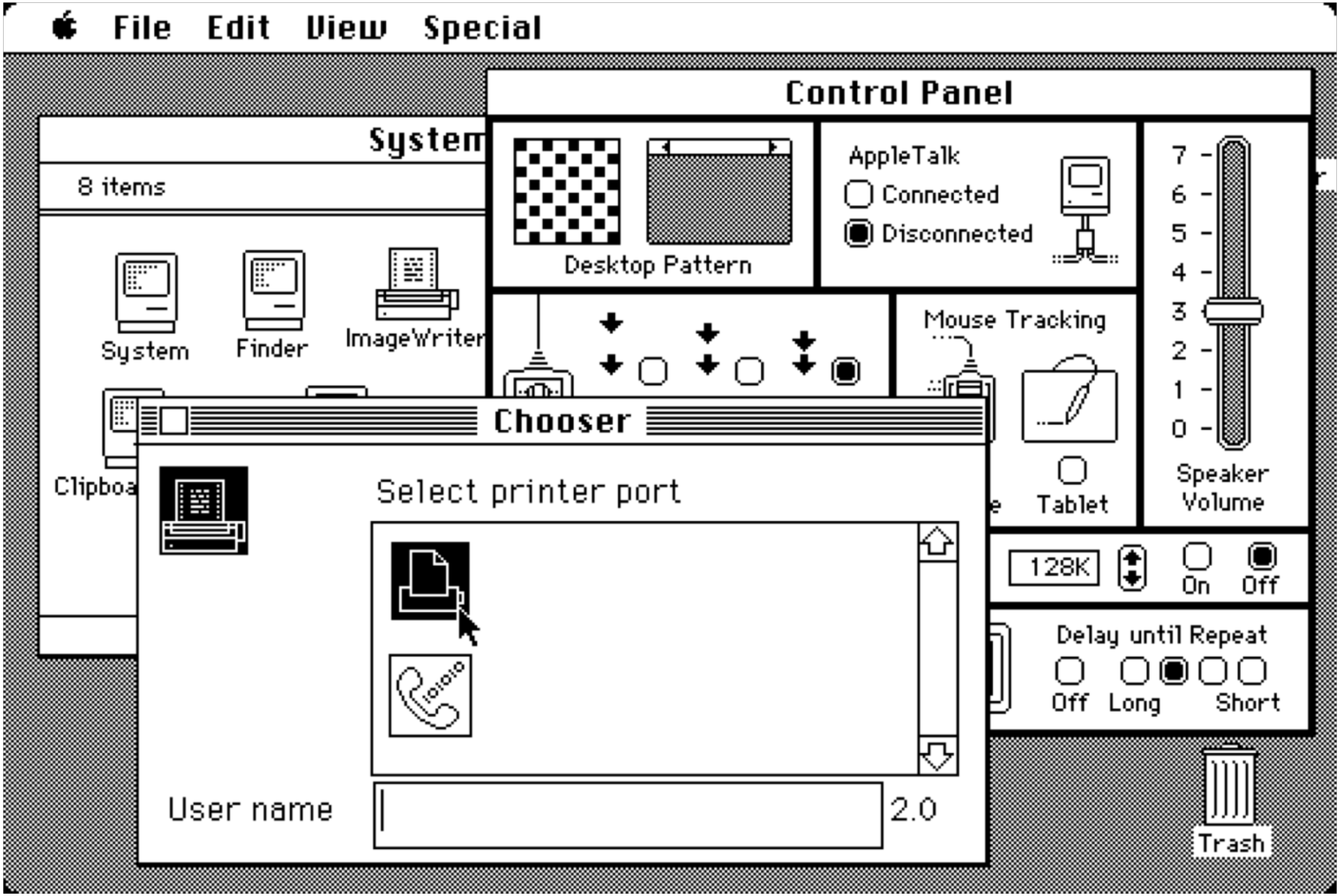
- **Widget** is a generic name for parts of an interface that have their own behavior: buttons, drop-down menus, spinners, file dialog boxes, progress bars, sliders, …
- widgets also called **components**, or **controls**
- They provide user feedback and capture user input
- They have a defined appearance
- They send and receive events





Widget from ***Wow Wow Wubbzy***

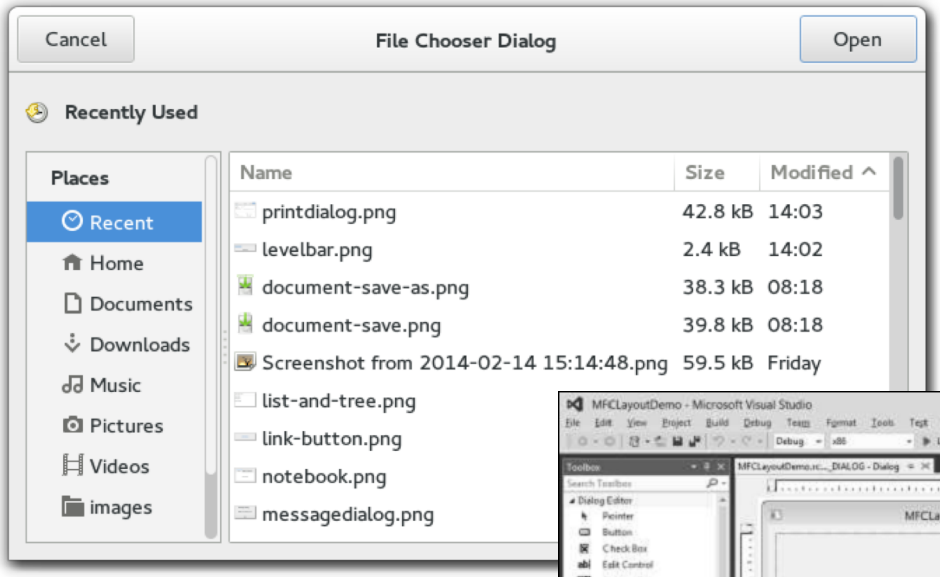
Early User Interface Widgets



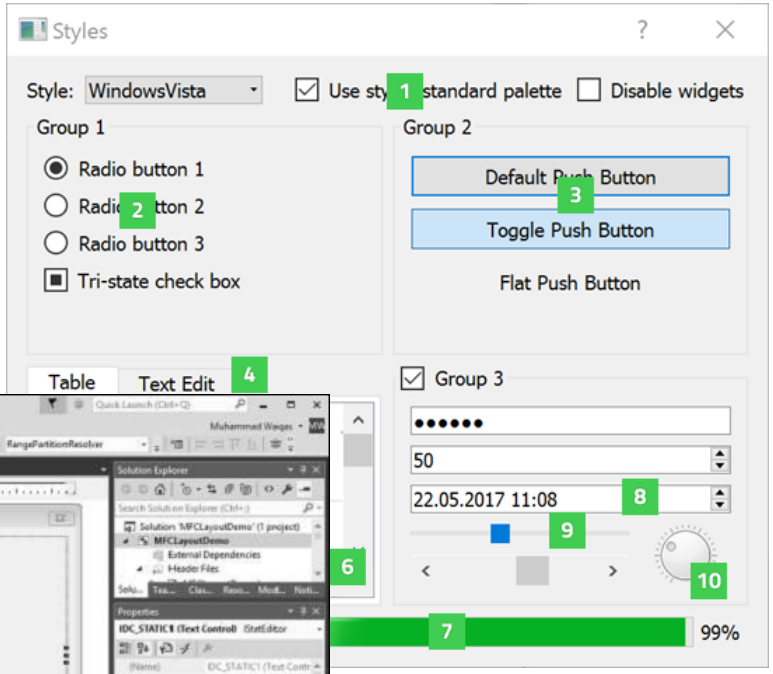
Macintosh System 5, circa 1987

Modern Widgets

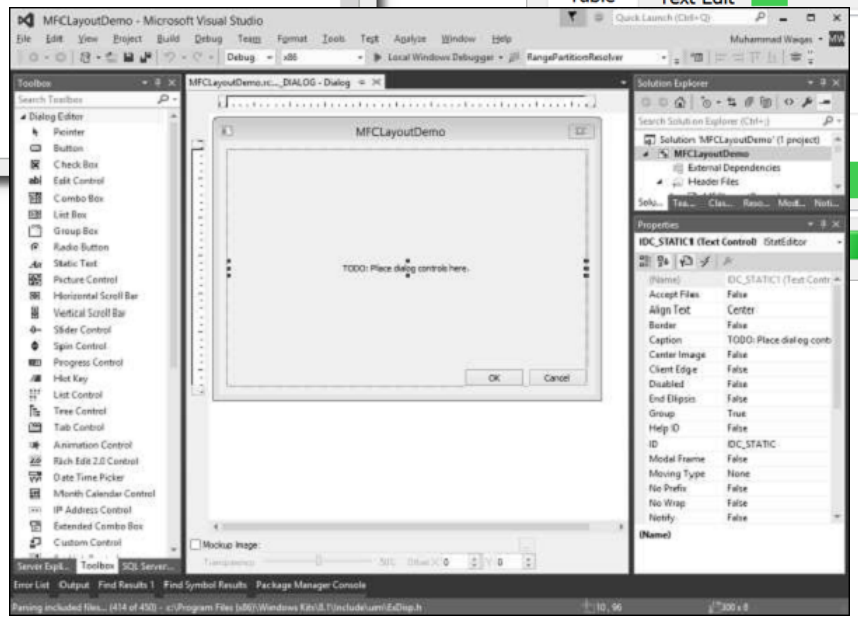
- Widget toolkits vary in presentation, but all include “standard” widgets.



Gtk+ (Linux, C++)



Qt (Windows, C++)



MFC (Windows, C++)

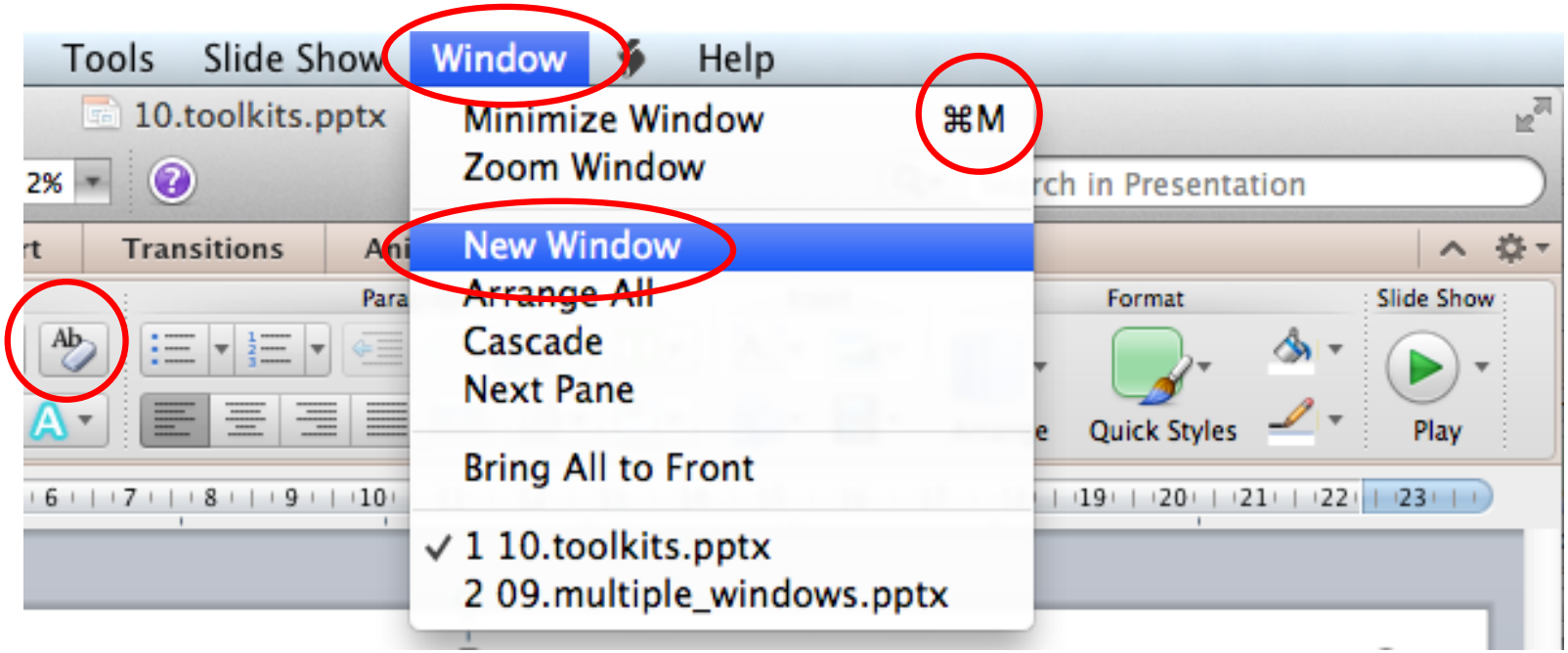
Logical Input Devices

We want to describe widgets based on their function (and not just appearance)

- A **logical device** is the essence of what a widget does, its *function*
- e.g. *logical button device* generates a “pushed” or “activated” event

A widget is a logical device with an appearance

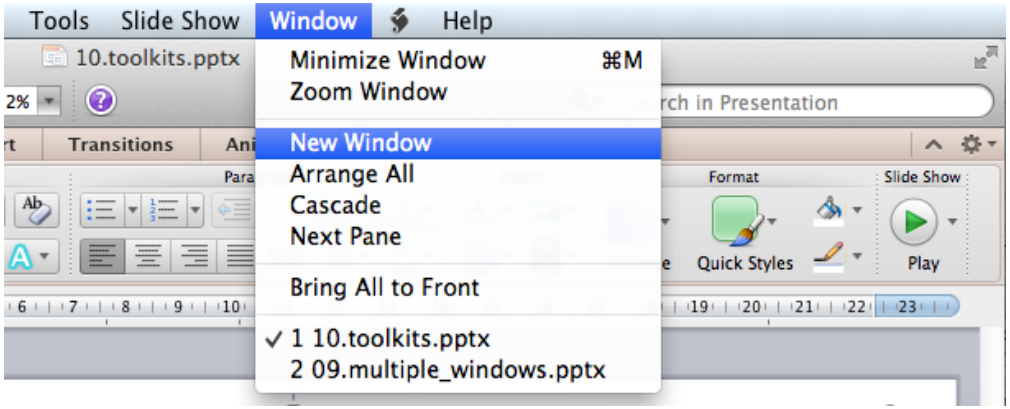
- e.g. widgets based on *logical button device*
  - appearances: push button, keyboard shortcut, menu item, …



Examples of Logical Input Devices

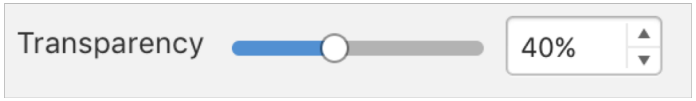
Logical Button Device

function: generate"activated" event
appearances: button, menu item, keyboard shortcut



Logical Number Device

function: adjust a number through a range of vaslues, generates "changed" event
appearances: slider, spinner, numeric textbox, …

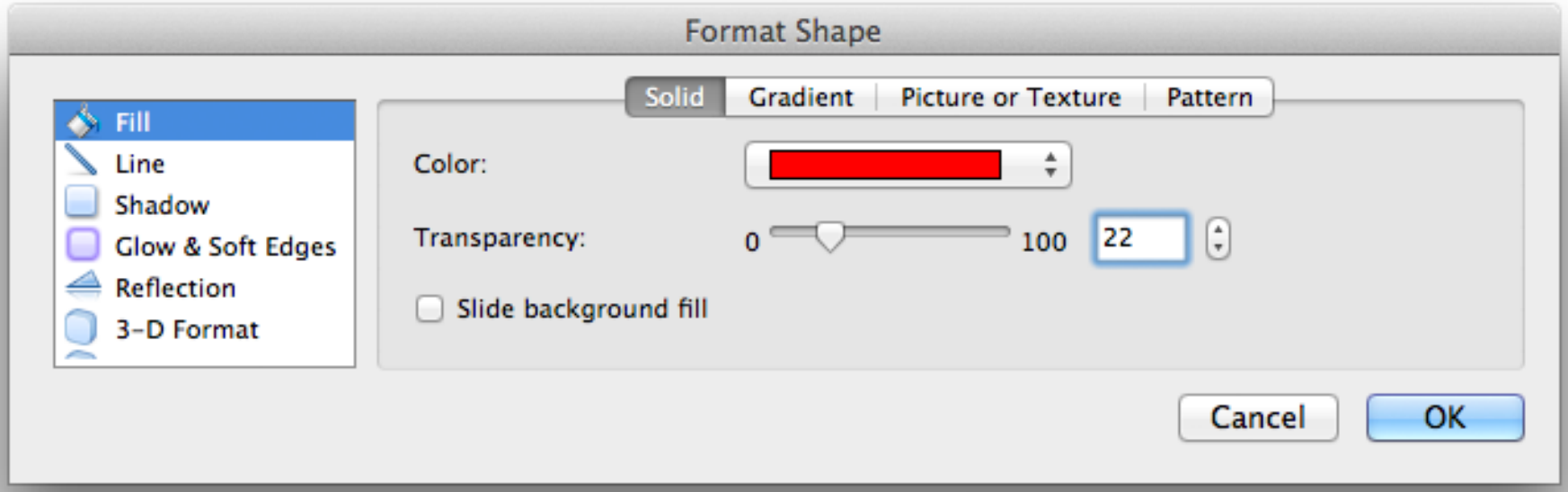


Logical Boolean Device

function: select/unselect, generates "changed" event
appearances: radio button, checkbox, toggle

Categorizing and Characterizing Widgets

- Logical input device. What functionality does it support?
- Can it contain other widgets? container vs. simple
- Other characteristics
  - Data that it represents
  - Events the widget generates
  - Properties to change behaviour and appearance



Widget Functionality

Logical input devices support the idea of categorizing widgets based on their behaviour (i.e. the type of input or interaction that they support).

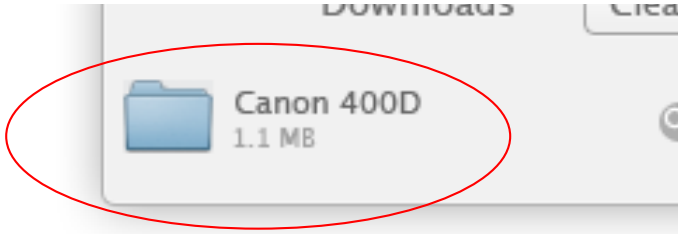
We also consider these dimensions:

- **State**: what data does the widget store? It often has to hold one or more values to model it's current state.
  - e.g. textbox holds a string, slider holds min/max/current value.

- **Events**: what events does the widget generate when activated?
  - e.g. buttons generate "pressed" event; slider generates "changed"

- **Properties**: flags that are used to settings or how the widget is presented; some of these are standard, while others are specific to a widget.
  - e.g. position (x,y); size (width, height); color; border_thickness

Simple Widgets



- Label (e.g. label, image, spacer, icon)
  - State: -
  - Events: -
  - Properties: label, size, visible, enabled

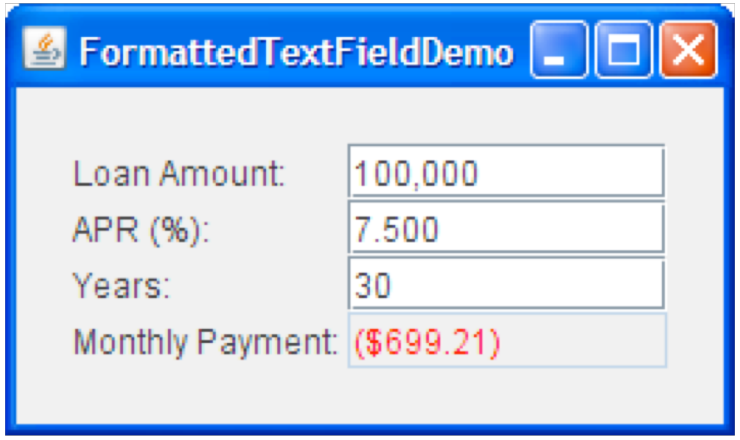
- Button (e.g. button, menu item)
  - State: -
  - Event: pushed
  - Properties: label, size , visible, enabled

- Boolean (e.g. radio button, checkbox, toggle button)
  - State: true/false model
  - Event: changed event
  - Properties: label, size, visible, enabled

“Radio Button”

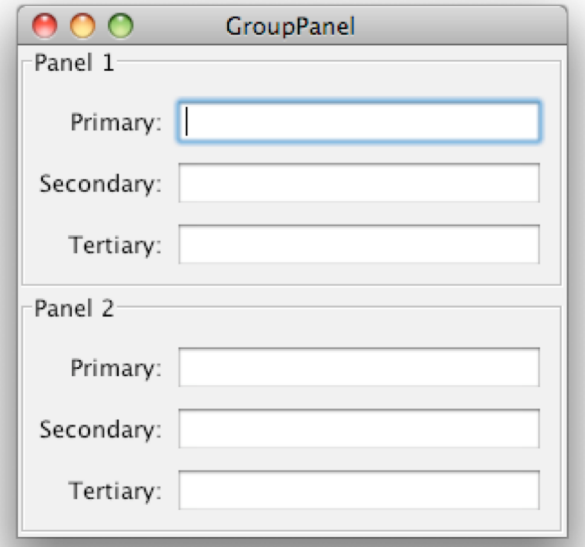
Simple Widgets

- Number (e.g. slider, progress bar, scrollbar)
  - State: real number
  - Event: changed
  - Properties: range, step

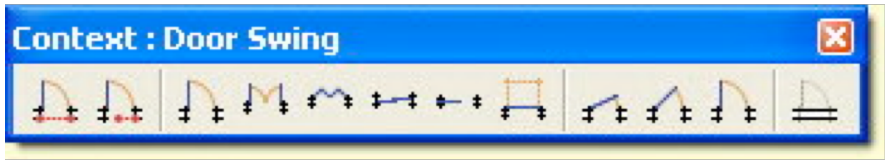


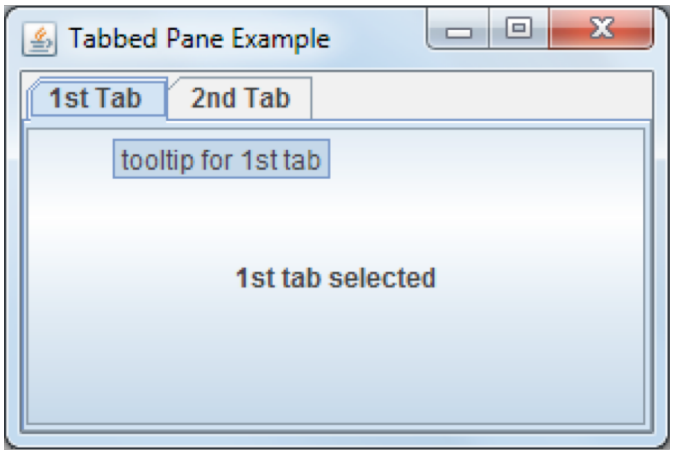
- Text (e.g. text field)
  - State: string
  - Events: changed, selection, insertion
  - Properties: formatters (numeric, phone number, …)

Container Widgets



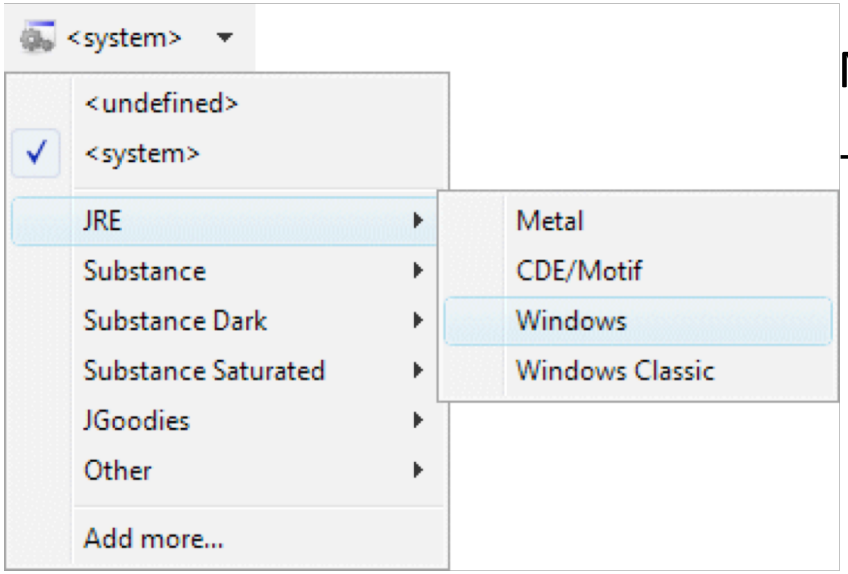
- Panel (e.g. Pane, Form, Toolbar)
  - arrangement of widgets





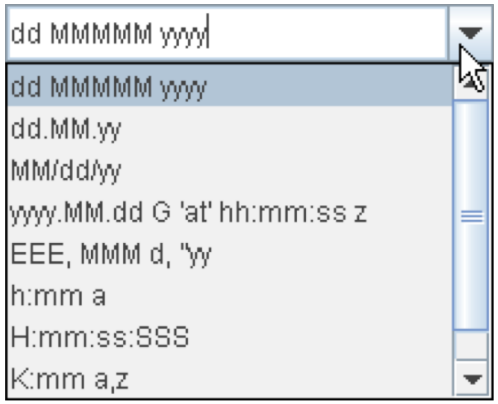
- Tab
  - choice between arrangements of widgets

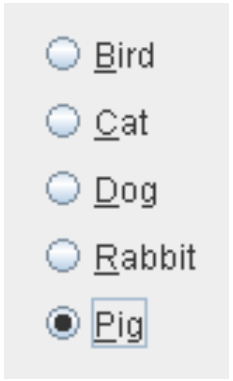
Container Widgets



Menu

– hierarchical list of (usually) buttons

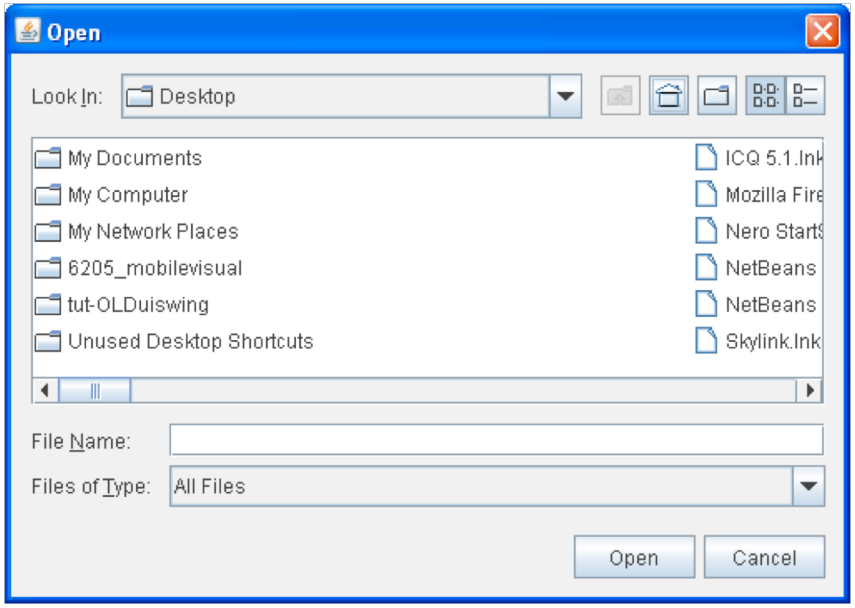




- Choice from a List
  - list of boolean widgets
  - e.g. drop-down, combo-box, radio button group, split button

Special Value Widgets

- colour/file/date/time pickers



Widget toolkits

Also called widget libraries or GUI toolkits or GUI APIs

- Defines a set of common GUI components for programmers
  - Examples: buttons, drop-down menus, sliders, progress bars, lists, scrollbars, tab panes, file selection dialogs, etc.

Low-Level Toolkits

- Integrated with the operating system
  - e.g. Cocoa with macOS, Windows API with Windows (graphics integrated with kernel until 2006)

High-Level Toolkits

- Higher-level of abstraction, not integrated with the operating system
  - e.g. Qt (C++), MFC (C++ Windows), Gtk+ (C, Python), wxWidgets (C++), Swing (Java), Cocoa (ObjC, Swift), Tk (C)

Widget Toolkit Design Goals

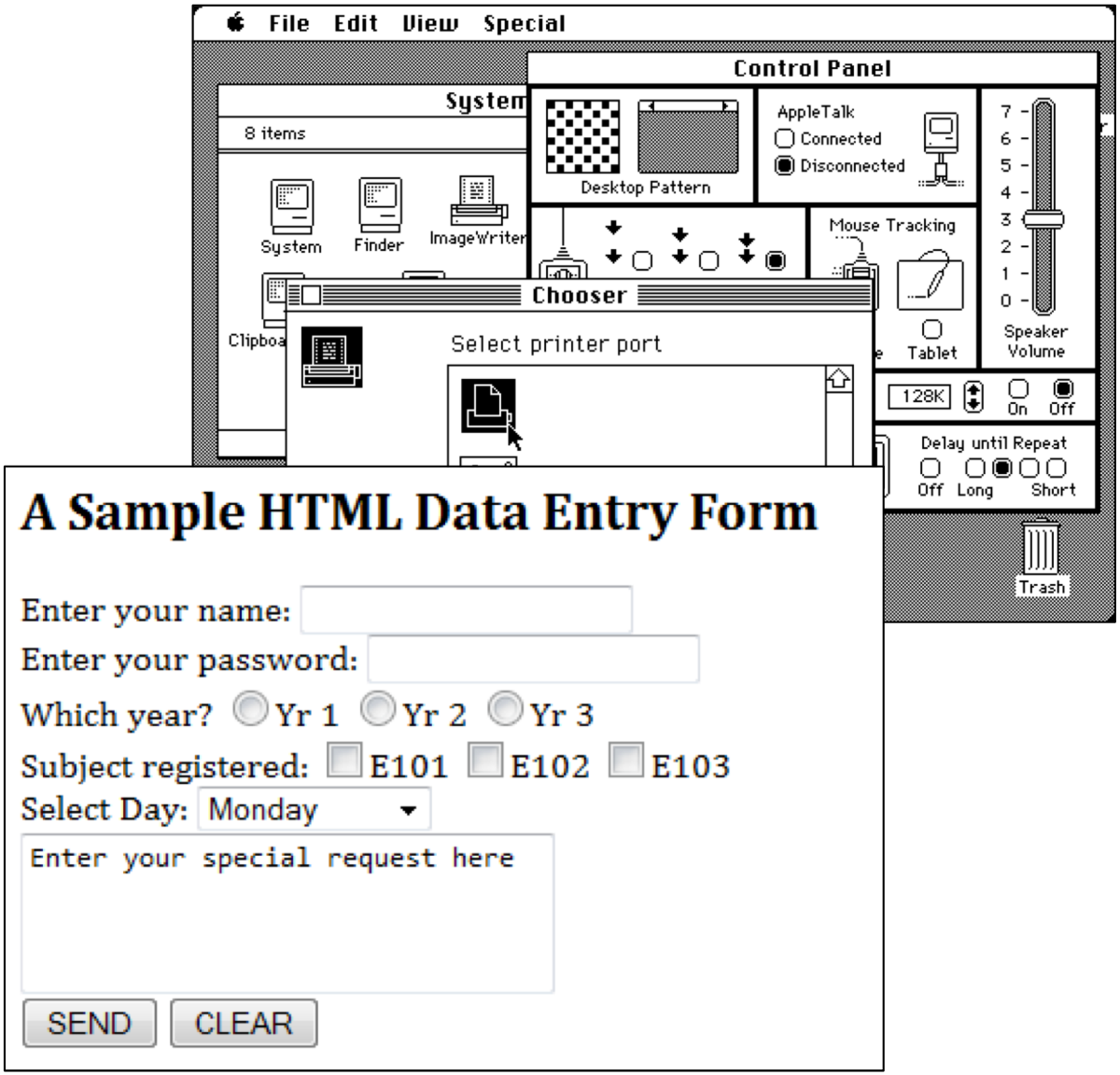
What widgets features are desirable in a widget toolkit?

1. **Completeness**
   - GUI designers have everything they need
2. **Consistency**
   - Behaviour is consistent across components
3. **Customizability**
   - Developer can reasonably extend functionality to meet particular needs of application

Meeting these requirements encourages *reuse.*

1/ Completeness

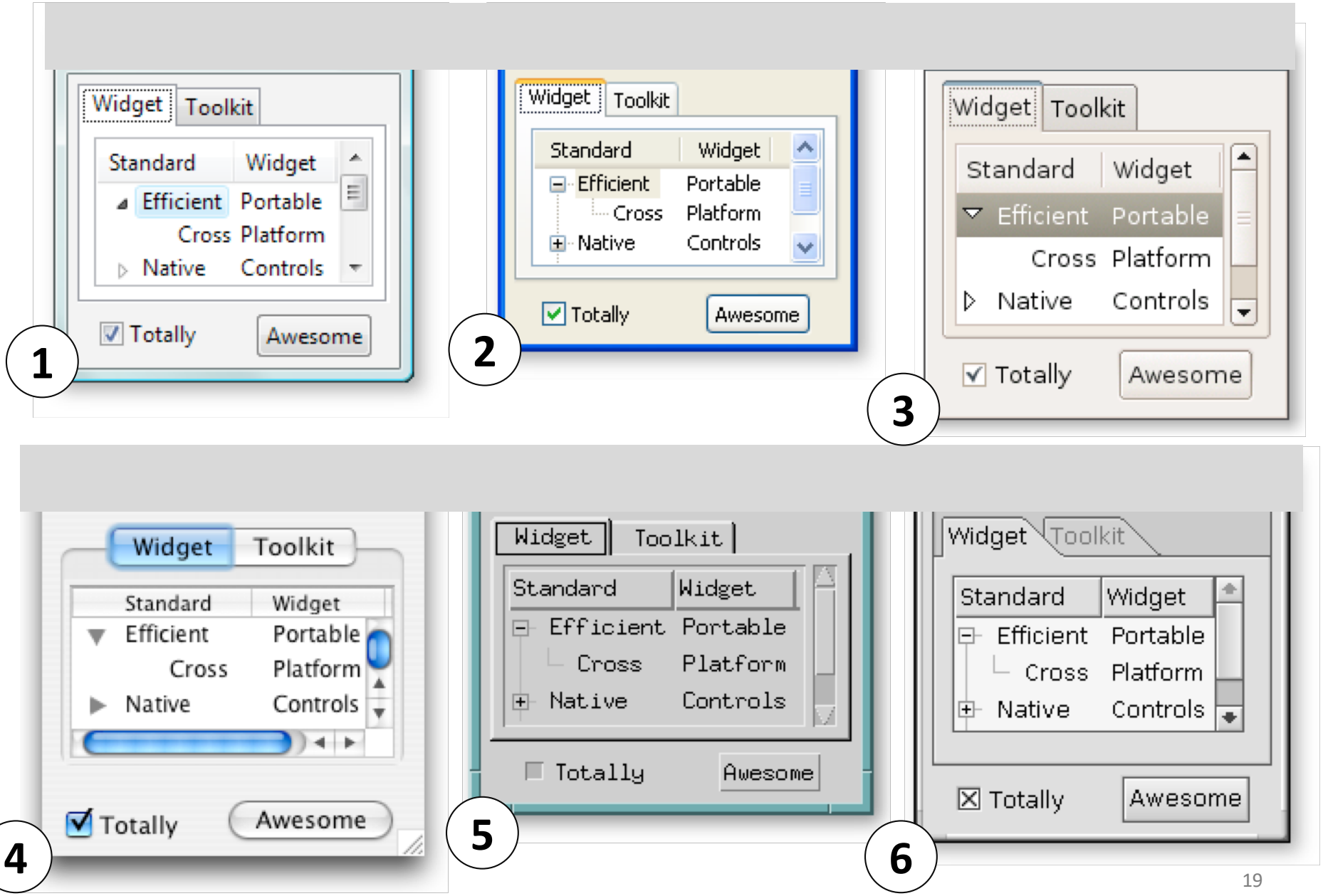
- All you really need are:
  - Button
  - Slider
  - Pulldown menu
  - Check box
  - Radio button
  - Text field
  - Spinner



Widget Choice

- Use a common look and feel
- Use widgets appropriately

2/ Consistency: Name that Look



3/ Customization

How do we customize widget behaviour and appearance?

Common strategies:

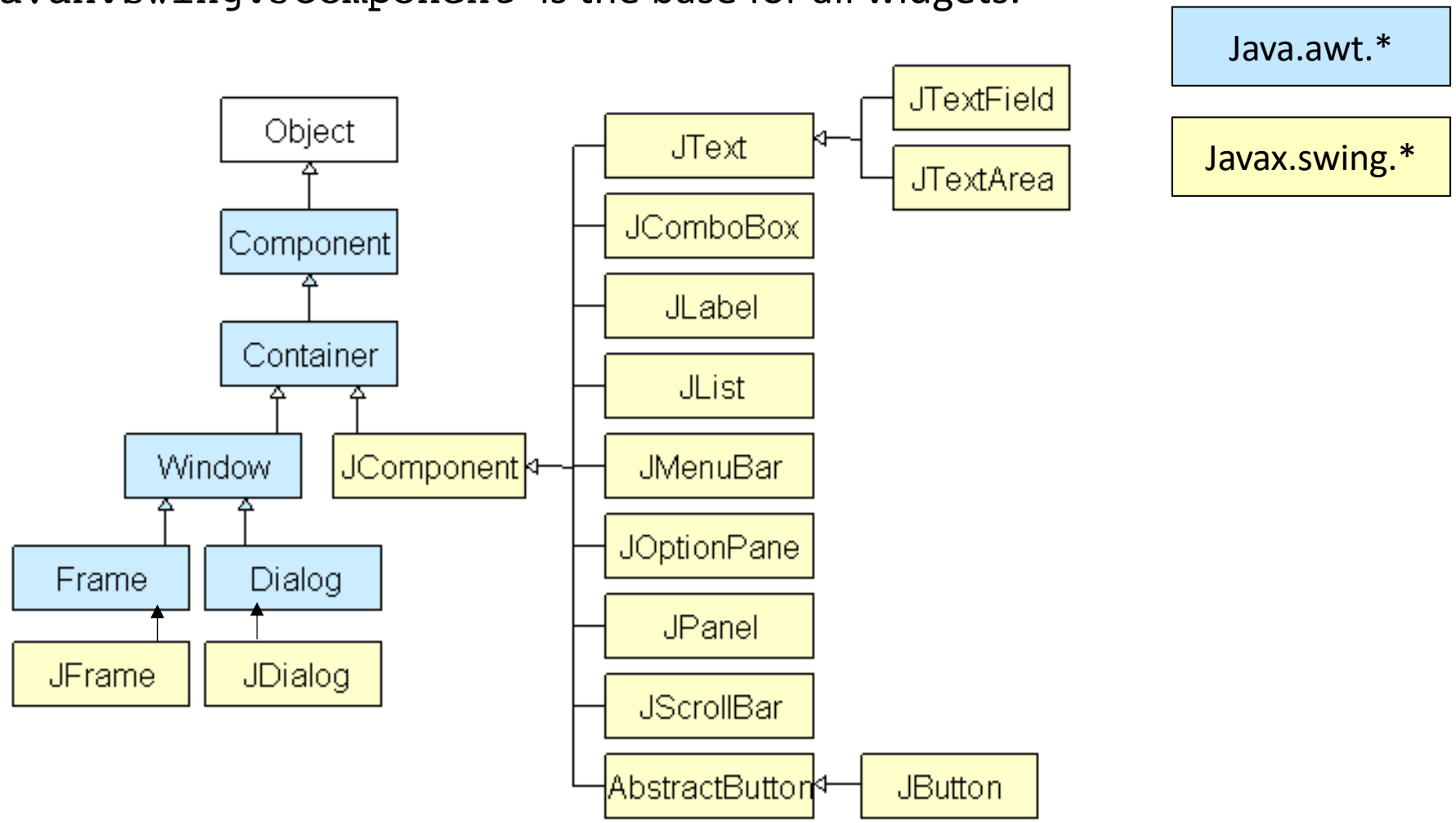
1. Expose properties for user to modify
   - e.g. change colour, font, orientation, text formatter, …
2. Factor out behaviour (i.e. plug-in behaviour)
   - Swing's UIManager for changing look and feel
   - LayoutManager allows custom layouts
3. Expose class hierarchy that promotes customization
   - JButton extends AbstractButton
     - Contains a ButtonModel to support state information
     - Contains Controller methods to fireActionPerformed
     - Contains an EventListenerList which contains a bunch of EventListener descendants (see declaration in tab)

Java GUI Toolkits

Toolkit	Description
AWT	• Low-level or “heavyweight”, used platform-specific widgets. • AWT applications were limited to common-functionality that existed on all platforms.
Swing	• High-level or “lightweight”, full widget implementation using *imperative* model. • Commonly used and deployed cross-platform.
Standard Window Toolkit / SWT	• ”Heavyweight” hybrid model: native, and tied to specific platform (Windows). • Used in Eclipse.
Java FX	• Intended for rich desktop + mobile apps. • *Declarative* programming model, designed to replace Swing.

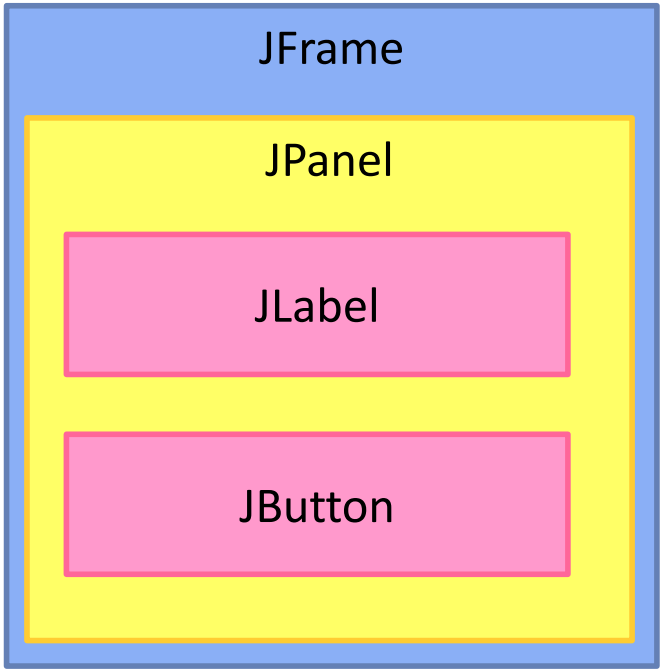
Swing Component Hierarchy

- `java.awt.Window` is the base for all containers.
- `javax.swing.Jcomponent` is the base for all widgets.



How to build a Swing UI

- Create a top-level application window, using a Swing container (`JFrame` or `JDialog`).
- Add Swing components to this window.
  - Typically, you create a smaller container (like a `JPanel`) and add components to the panel.
  - This makes dynamic layouts easier (more on that later!)



Using Swing Widgets

- Build a hierarchy by adding top-level containers, and nested components.

```
public static void main(String[] args) {
    // create a window
    JFrame frame = new JFrame("Window Demo");
    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

    // create a panel and add components
    JPanel panel = new JPanel();
    JButton button = new JButton("Ok");
    button.addMouseListener(new MouseAdapter() {
            public void mouseClicked(MouseEvent e) {
            class instead of MyMouseAdapter class
                System.exit(1);
    }});
    panel.add(button);

    // add panel to the window
    frame.add(panel);
```

BasicForm.java

WidgetDemo.java

- Add JLabel, set properties
- Add JButton, JSlider
- Create JMenuItems, add to JMenu in a JMenuBar
- Create JRadioButtons, put into ButtonGroup and JPanel
- Create events for all widgets to setText in label
- Set layout manager for JFrame