Widgets

Components

Widget toolkits

Logical input devices

MVC at widget-level

Recall: Widget

Widget is a generic name for parts of an interface that have their own behavior.

- e.g.: buttons, progress bars, sliders, drop-down menus, spinners, file dialog boxes, …

Widgets also called “components”, or “controls”

- Control their own appearance
- Receive and interpret their own events (event handling mechanisms that we’ve already discussed)

Often put into libraries (toolkits) for reuse

Recall: Widget Toolkits

Widget toolkits / libraries (also called GUI toolkits)

- Software bundled with a window manager, operating system, development language, hardware platform

The toolkit defines a set of GUI components for programmers

- – Examples: buttons, drop-down menus, sliders, progress bars, lists, scrollbars, tab panes, file selection dialogs, etc.

Programmers access these GUI components via an application programming interface (API)

Heavyweight Widgets

- OS provides widgets and hierarchical “windowing” system
- Widget toolkit wraps OS widgets for programming language
- Base Window System (BWS) can dispatch events to a specific widget
- Examples: nested X Windows, Java’s AWT, OSX Cocoa, standard HTML form widgets, Windows MFC

Advantages

- Events generated by user are passed directly to components by BWS/OS
- Preserves OS look and feel

Disadvantages

- OS-specific programming
- Multi-platform toolkits tend to be defined as the “lowest-common set” of components

Implementation: Heavyweight vs. Lightweight

Lightweight Widgets

- OS provides a top level window; widget toolkit draws its own widgets in the window.
- Toolkit is responsible for mapping events to their corresponding widgets
- Examples: Java Swing, JQuery UI, Windows WPF

Advantages

- Can guarantee identical look-and-feel across platforms.
- Can guarantee consistent widget set on all platforms.
- Can implement very light/optimized widgets.

Disadvantages

- Concerns that they appear “non-native”.
- Concerns about performance with extra layer of abstraction.

Widget Toolkit Design Goals

1. Complete
   - “Complete” set of widgets and functionality
   - Goal: GUI designers have everything they need

2. Consistent
   - User: Look and Feel is consistent across components
   - Developer: Consistent usage paradigms

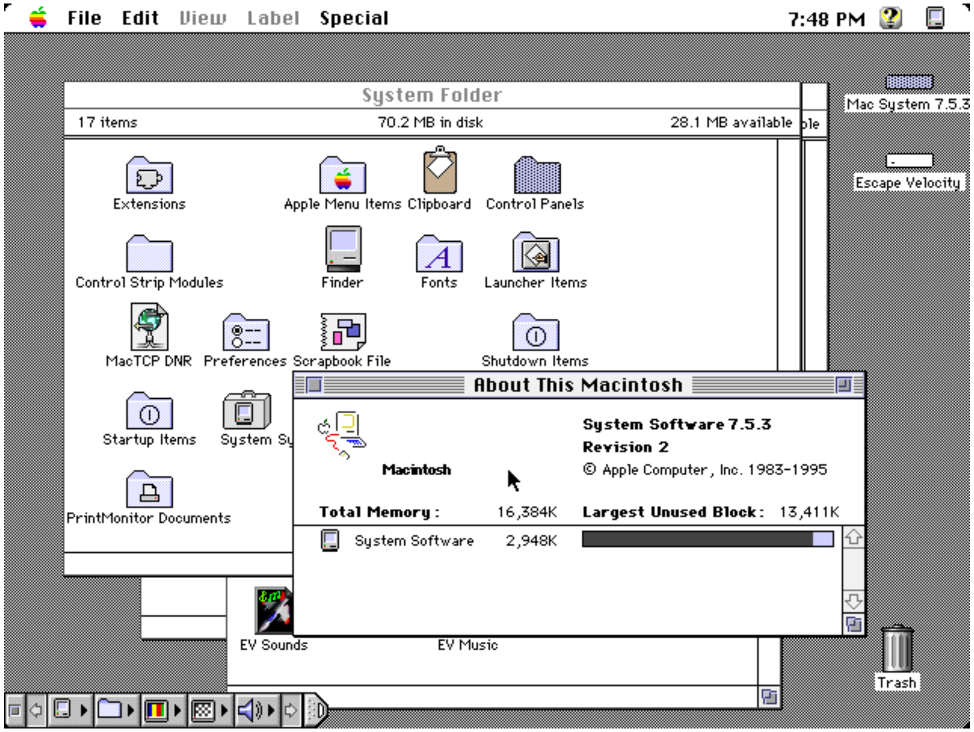
3. Customizable
   - Developer can reasonably extend functionality to meet particular needs of application

Meeting these requirements encourages reuse

Completeness

The “Macintosh 7” (Dix, Finlay, Abowd, et al. 1998)

1. Button
2. Slider
3. Pull-down menu
4. Check box
5. Radio button
6. Text entry fields
7. File open / save



System Software 7.5.3
Revision 2
© Apple Computer, Inc. 1983-1995

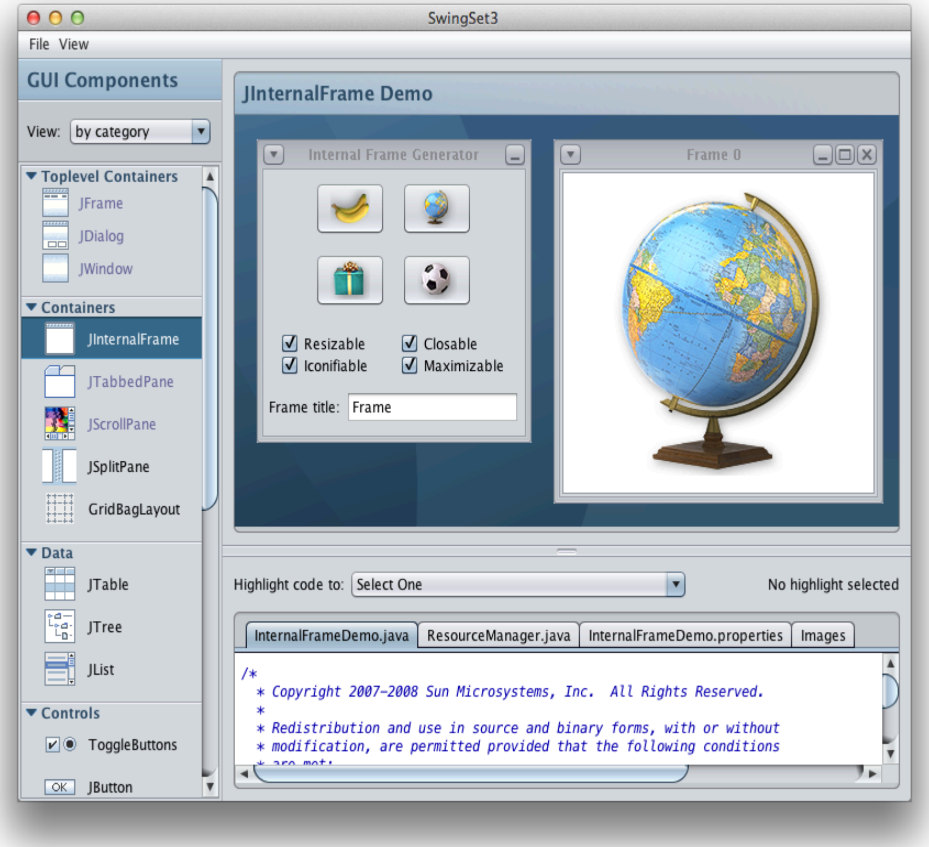
Total Memory : 16,384K
Largest Unused Block : 13,411K
System Software 2,948K

Java Swing has many more widgets …

Completeness

SwingSet Demo

- Shows lots of different widgets with lots of variations
- Can easily view source code



☑ Resizable ☑ Closable
☑ Iconifiable ☑ Maximizable

Frame title: Frame

```
/*
 * Copyright 2007-2008 Sun Microsystems, Inc.  All Rights Reserved.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
```

- To run:
  - Download jar from course web site

```
java –cp SwingSet2.jar SwingSet2
```

Consistency

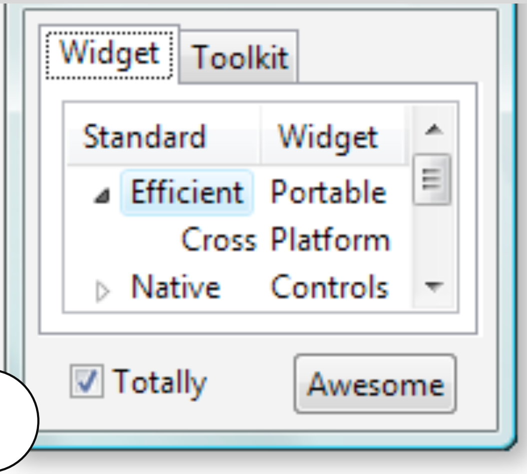
Facilitate learning by:

- Common look and feel
- Using Widgets appropriately

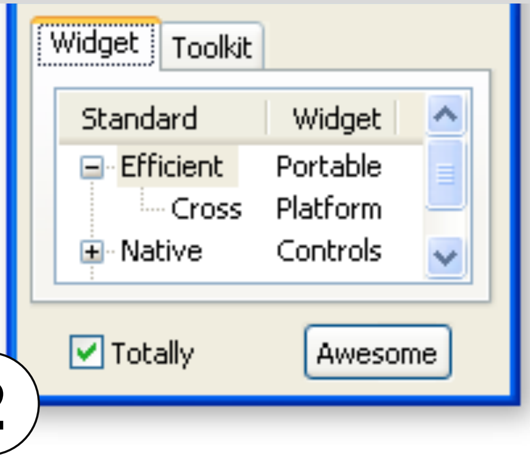
- Look: consistent visual appearance
- Feel: consistent and expected behaviour

Consistency helps users anticipate how the interface will react, and promotes easier *discoverability* of features.

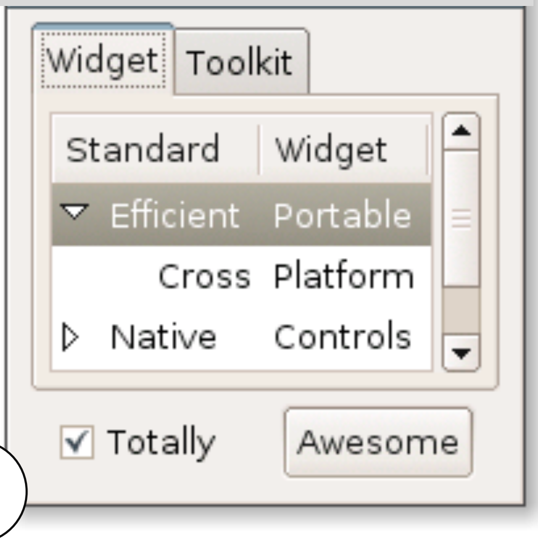
Consistency: Name that Look



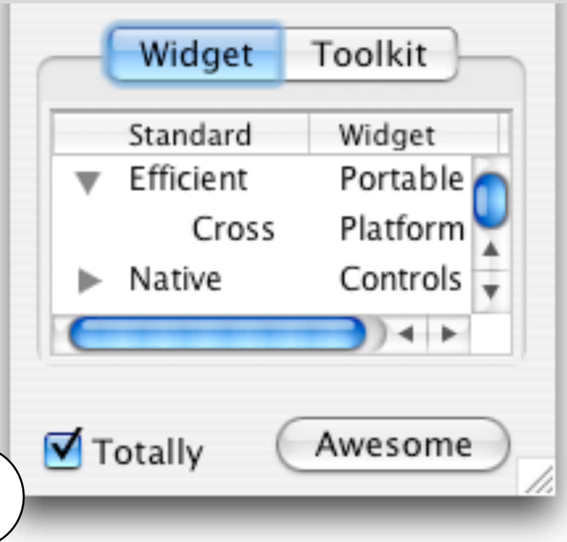
1



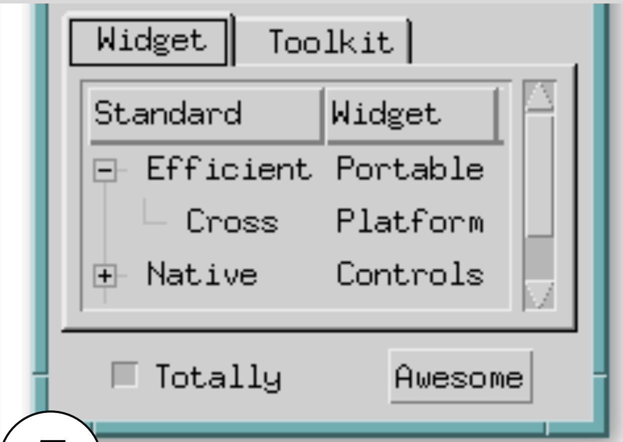
2



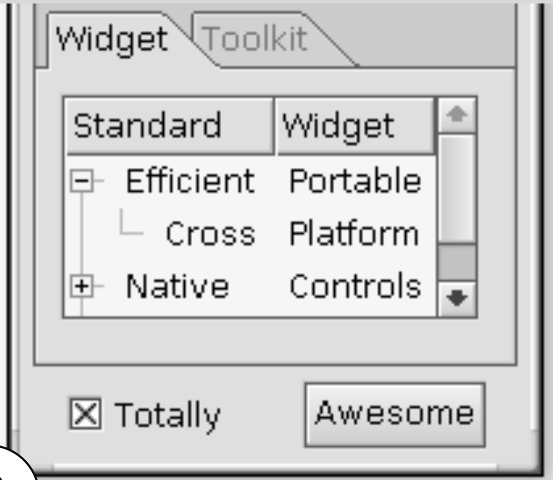
3



4



5



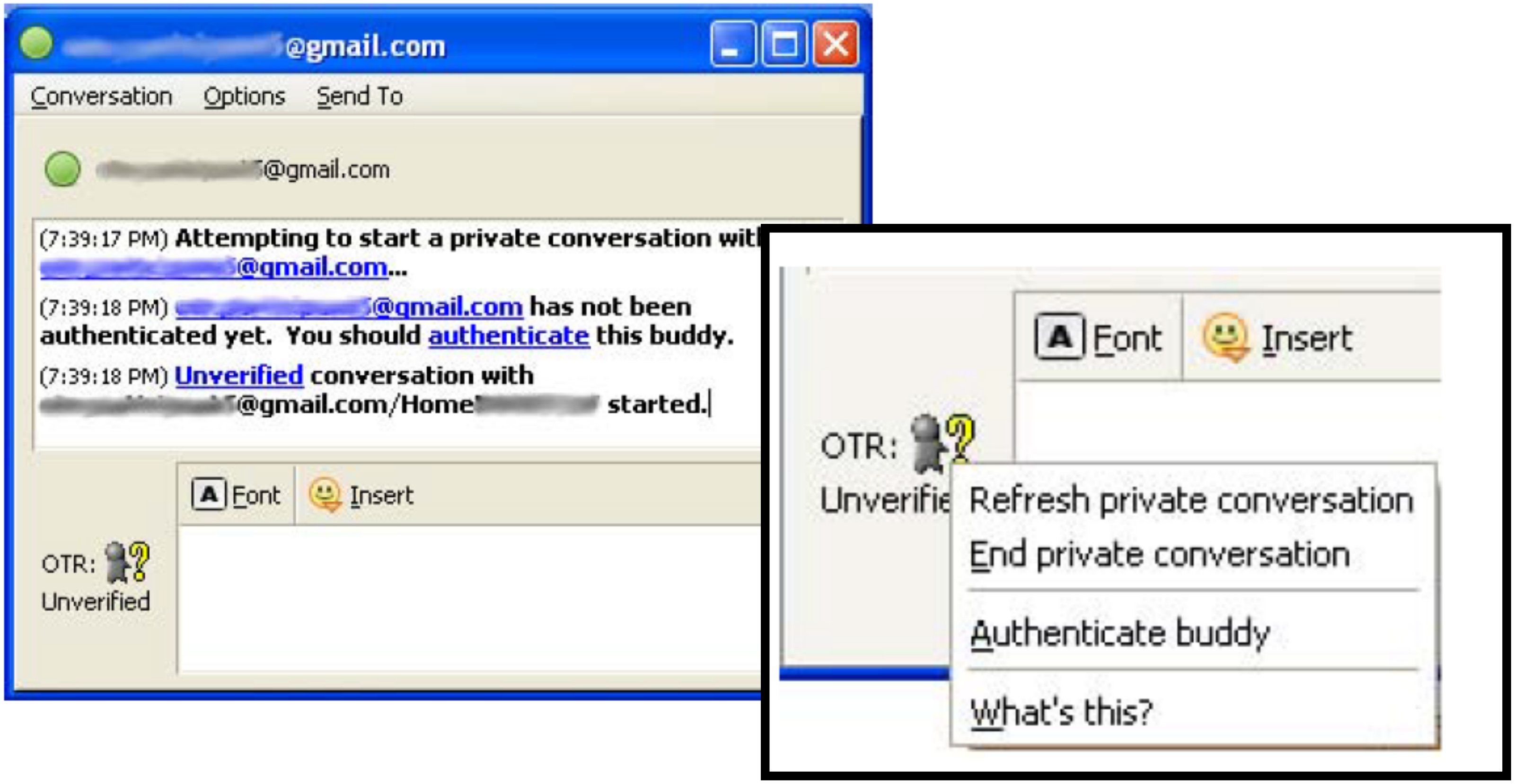
6

Consistency: Using Widgets Appropriately

People expect widgets to behave in certain ways

Off The Record (OTR) messaging study by Stedman et al.

- Question: How do you authenticate this buddy?
- Answer: Right-click on the label at bottom left!



(7:39:17 PM) Attempting to start a private conversation wit [redacted]@gmail.com...

(7:39:18 PM) [redacted]@gmail.com has not been authenticated yet. You should authenticate this buddy.

(7:39:18 PM) Unverified conversation with [redacted]@gmail.com/Home[redacted] started.

http://www.cypherpunks.ca/~iang/pubs/otr_userstudy.pdf

Customizable

How do we customize widget behaviour and appearance?

Two common strategies:

1. Properties
   - e.g. change colour, font, orientation, text formatter, …
2. Factor out behaviour (Pluggable behaviour)
   - Responding to an action: ActionListener
   - Swing’s UIManager for changing look and feel
   - JTable example…

More on this in a few slides…

Widgets as Input Devices

Logical input devices

How widgets use MVC

Physical Input Devices

Lots of different mechanisms for capturing user intent

- mechanical (e.g., switch, potentiometer)
- motion (e.g., accelerometer, gyroscope)
- contact (e.g., capacitive touch, pressure sensor)
- signal processing (e.g., computer vision, audio)

Logical Input Device

We can also view input devices as *logical* input devices.

Logical input devices are defined by their function (*not* what they looks like!)

Each device transmits a particular kind of input primitives:

- locator: inputs a (X,Y) position
- pick: identifies a displayed object
- choice: selects from a set of alternatives
- valuator: inputs a value
- string: inputs a string of characters
- stroke: inputs a sequence of (X,Y) positions

There may be multiple physical devices (e.g., mouse, joystick, tablet) that map to the same logical input device.

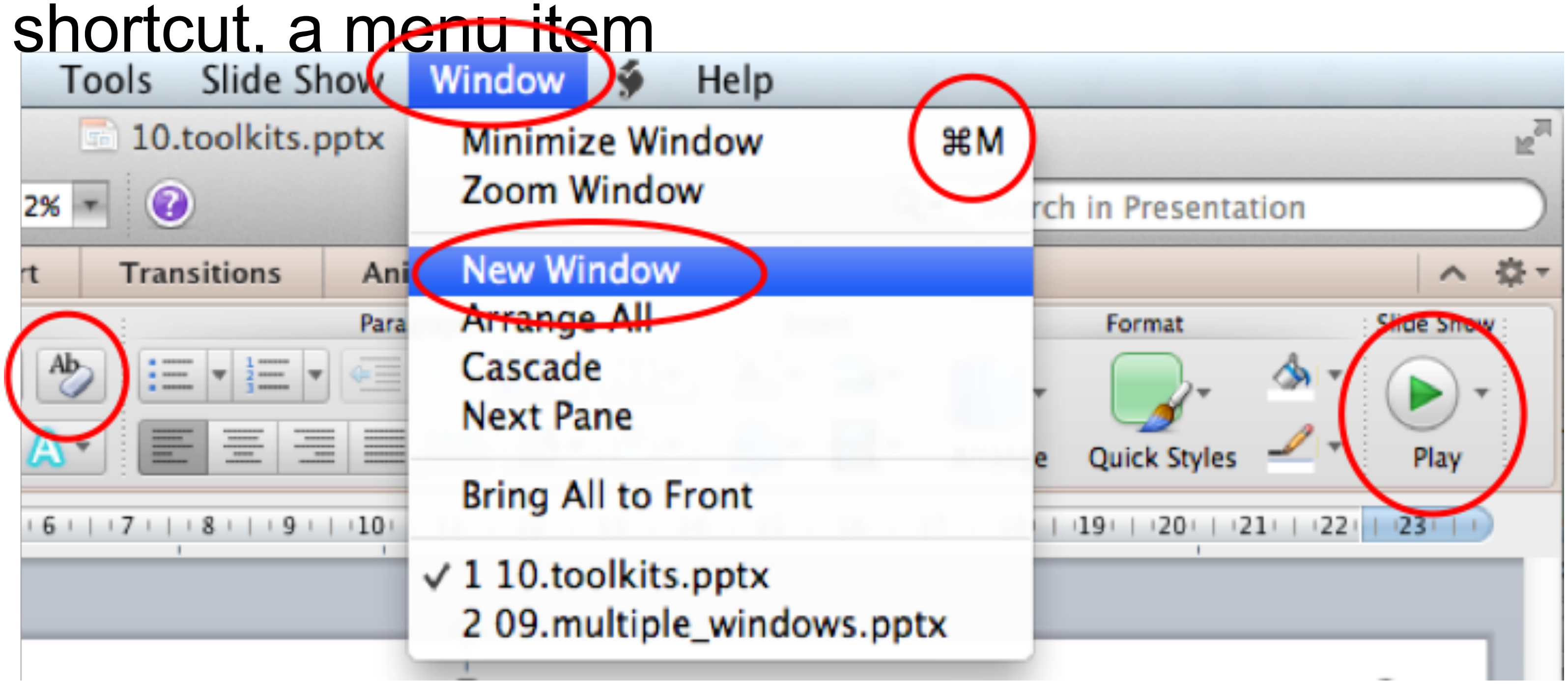
Logical Input Device

A widget can be considered a realization of a particular logical input device.

- Each logical input device can be represented by one or more widgets.

e.g. Logical Button Device

- Model: none
- Events: generates a "pushed" event
- Appearance: can look like a push button, a keyboard shortcut, a menu item

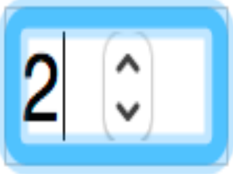


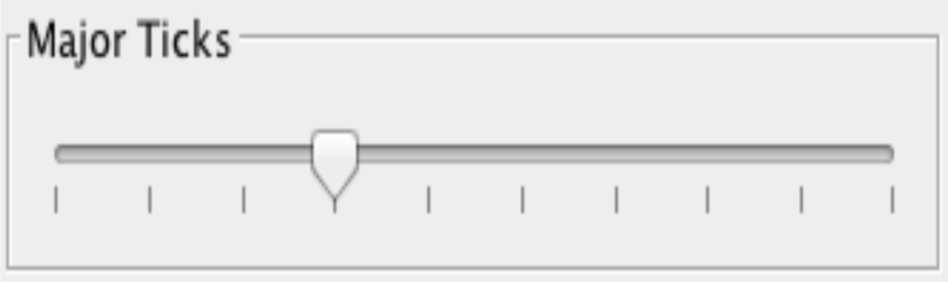
Logical Input Device

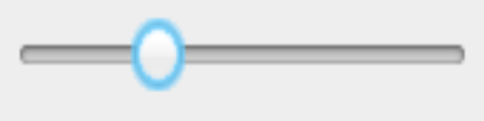
A widget can be considered a logical input device with appearance.

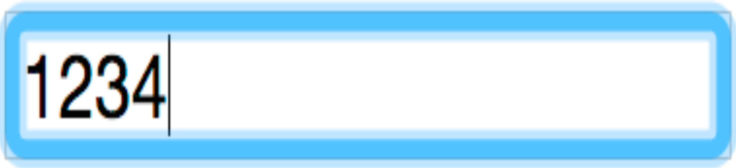
e.g. Logical Number Device

- Model: a number
- Events: “changed”
- Appearance: slider, spinner, textbox (with validation)





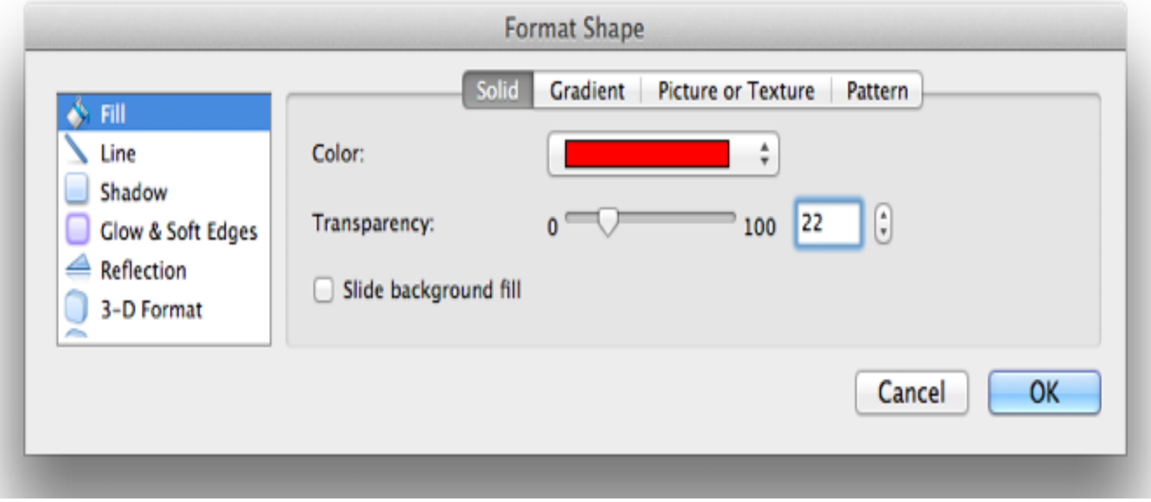




Categorizing and Characterizing Widgets

We can consider logical input devices and widgets in terms of these characteristics.

- Model the widget manipulates (number, text, choice…)
  - implementation (simple, abstract)
- Events the widget generates (action, change,…)
- Properties to change behaviour and appearance (colour, size, icon, allowable values, …)
  - Contains other widgets vs. stand-alone



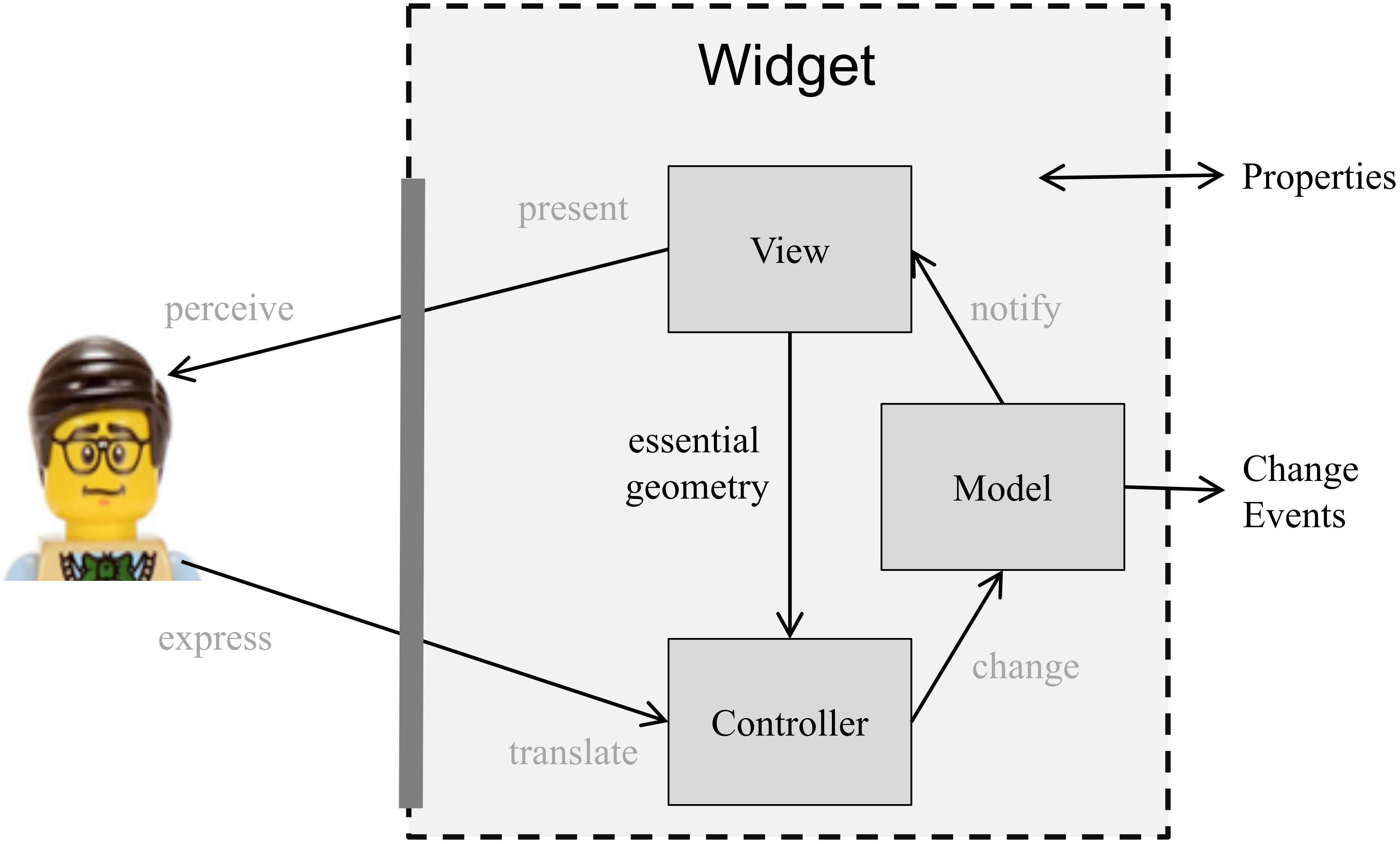
Color: 🟥

Transparency: 22

□ Slide background fill

MVC Widget Architecture

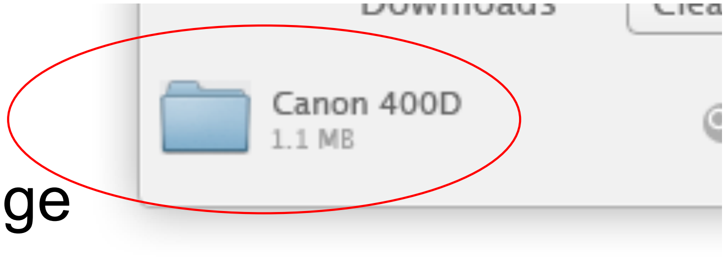
Note: We've now introduced MVC at two distinct levels: the widget and the entire application.



Simple Widgets 1

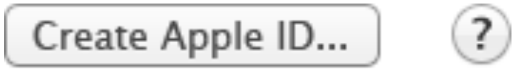
Labels and Images

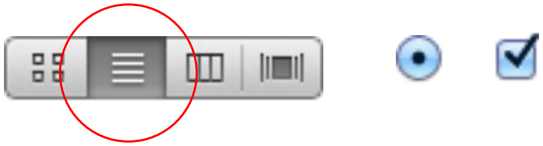
- Model: none
- Events: usually none
- Properties: text (font, size,…), image
- e.g. label, icon, spacer,



Button

- Model: none
- Events: push
- Properties: label, size, color, …
- e.g. button

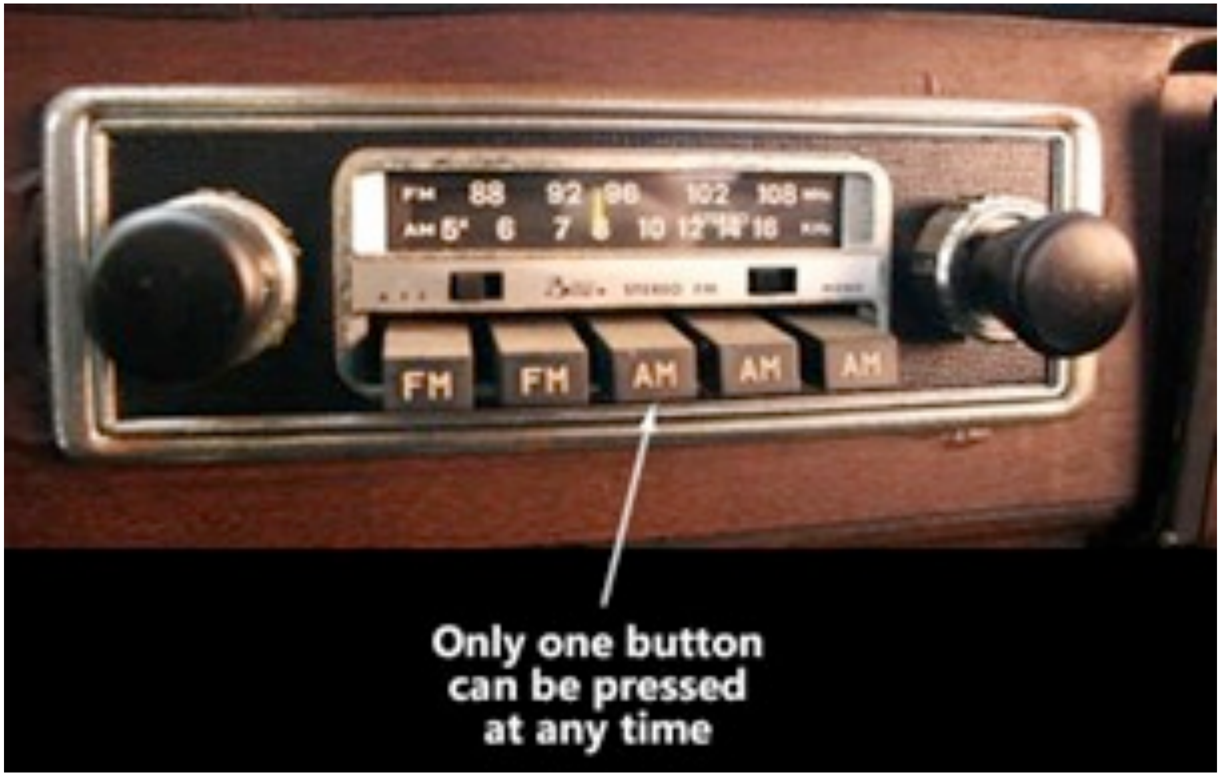




Boolean

- Model: true/false
- Events: changed event,
- Properties: size, color, style
- e.g. radio button, checkbox, toggle button

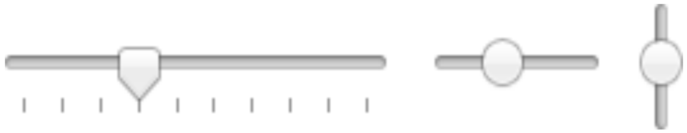
“Radio Button”



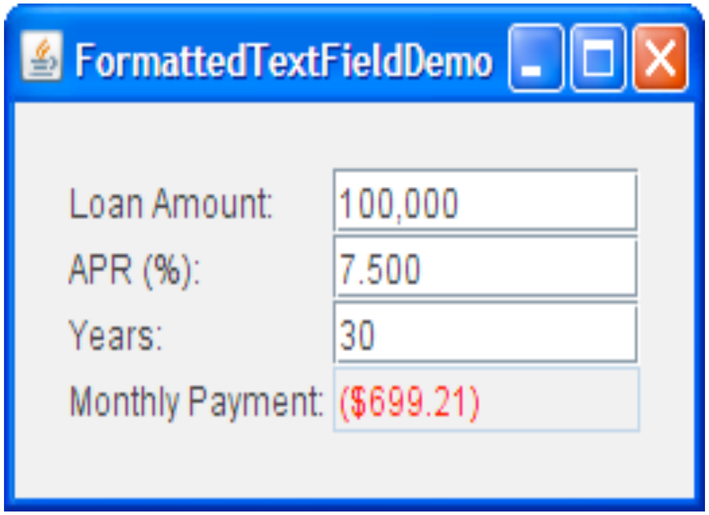
○ ○ ○ ◉ ○

Simple Widgets 2

- Number
  - Model: bounded real number
  - Events: changed event,
  - Properties: style, format
  - e.g. slider, progress bar, scrollbar



- Text
  - Model: string
  - Events: changed, selection, insertion
  - Properties: optional formatters (numeric, phone number, …)
  - e.g. text fields, text areas,



FormattedTextFieldDemo

Loan Amount: 100,000

APR (%): 7.500

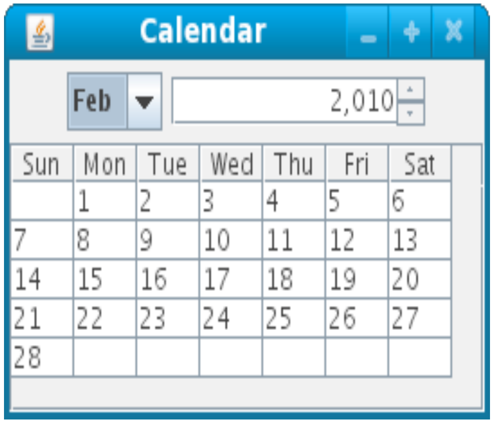
Years: 30

Monthly Payment: ($699.21)

Special Value Widgets

Examples:

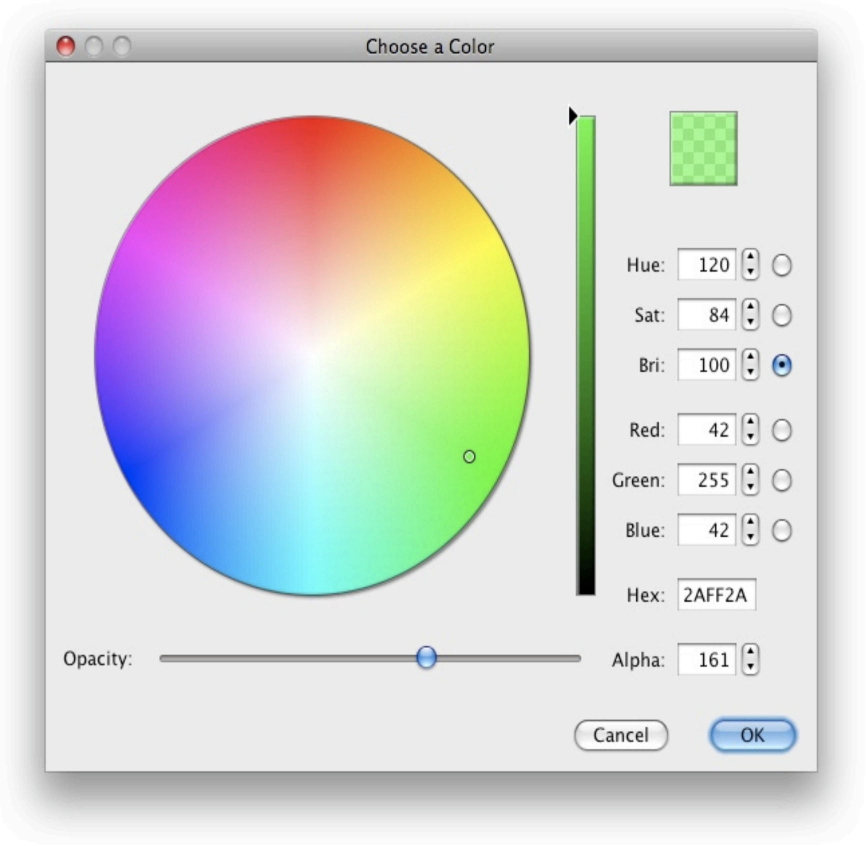
colour / file / date / time pickers



Calendar

Feb ▾ 2,010

Sun	Mon	Tue	Wed	Thu	Fri	Sat
	1	2	3	4	5	6
7	8	9	10	11	12	13
14	15	16	17	18	19	20
21	22	23	24	25	26	27
28						



Choose a Color

Hue: 120 ○

Sat: 84 ○

Bri: 100 ●

Red: 42 ○

Green: 255 ○

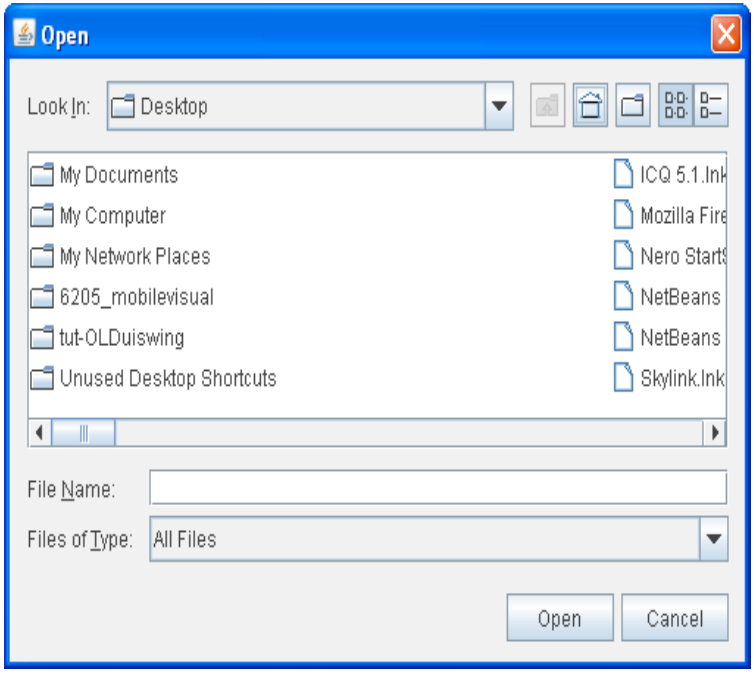
Blue: 42 ○

Hex: 2AFF2A

Opacity:

Alpha: 161

Cancel OK



Open

Look In: Desktop

My Documents

My Computer

My Network Places

6205_mobilevisual

tut-OLDuiswing

Unused Desktop Shortcuts

ICQ 5.1.lnk

Mozilla Fire

Nero Starts

NetBeans

NetBeans

Skylink.lnk

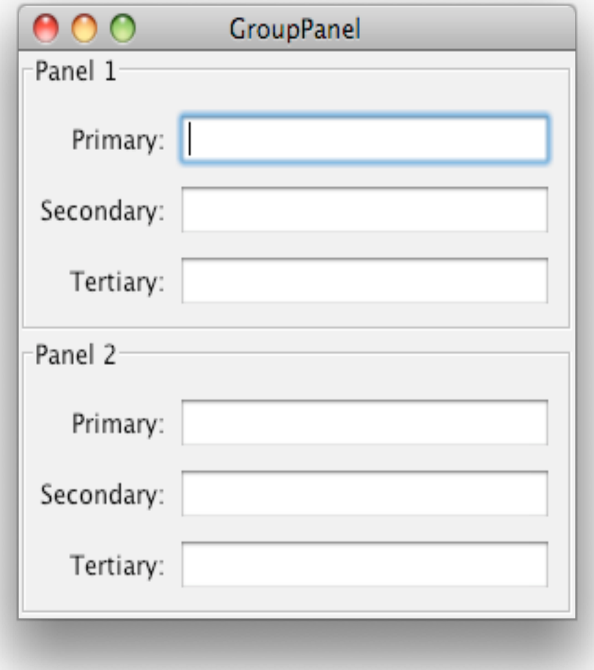
File Name:

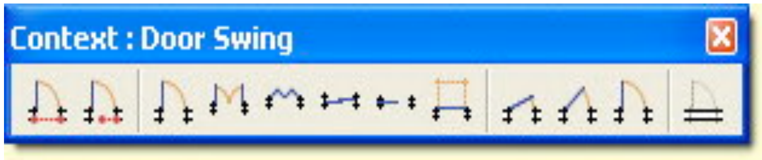
Files of Type: All Files

Open Cancel

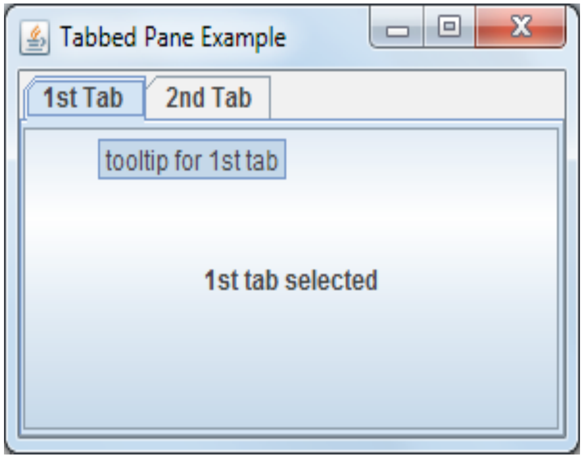
Container Widgets 1

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



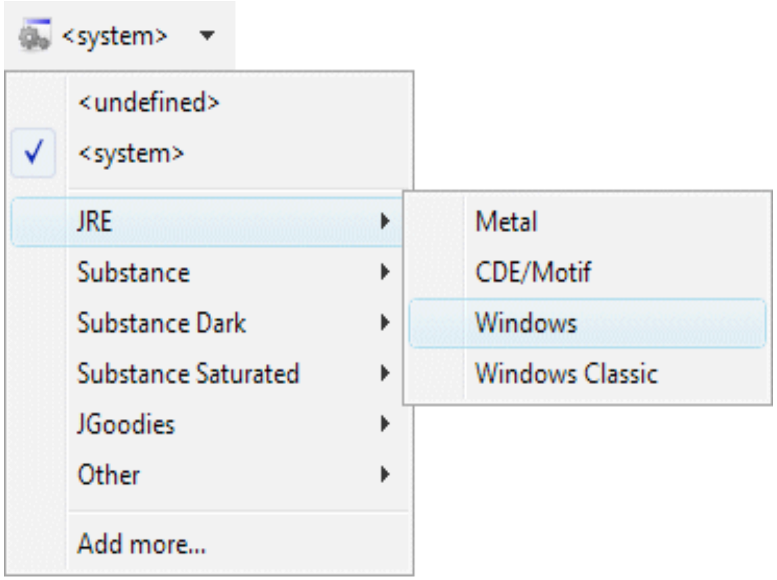


- Tab
  - choice between arrangements of widgets

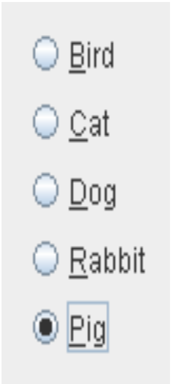


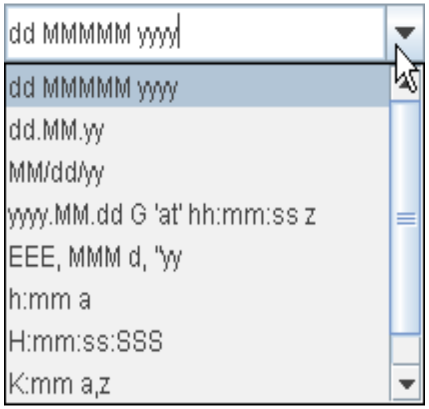
Container Widgets 2

- 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

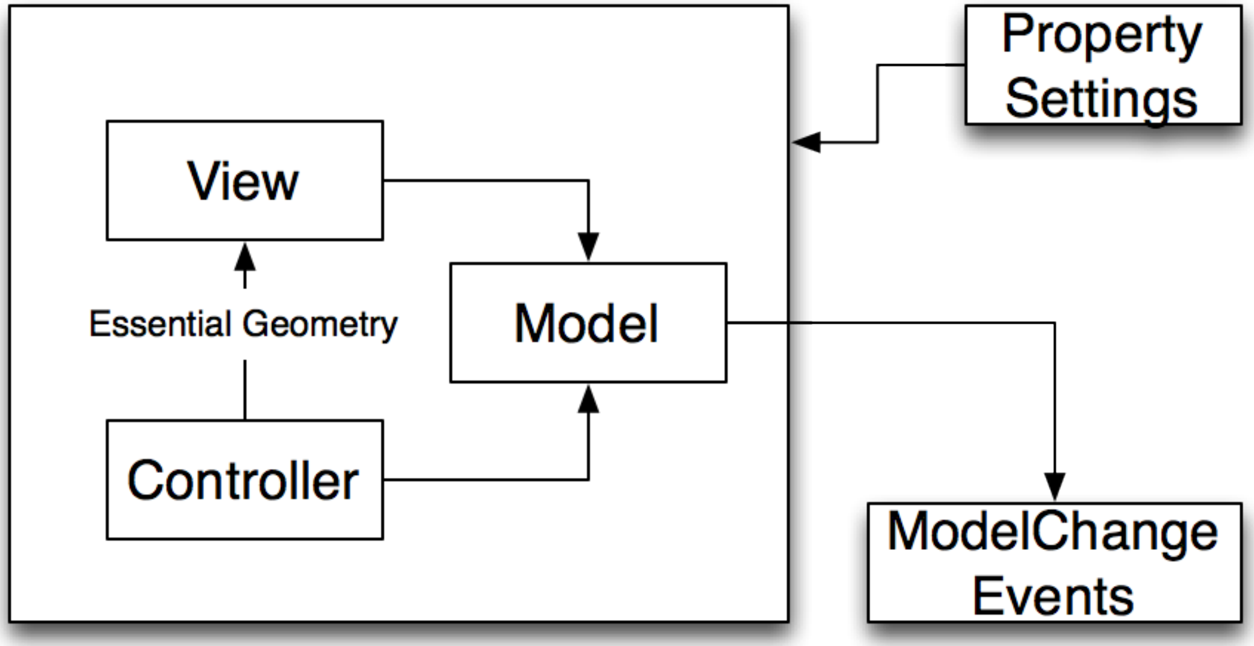




Customization: Simple Widgets

- Modern widget toolkits use MVC throughout
  - Simple widgets usually contain a default model within themselves
  - Examples: buttons, checkboxes, scrollbars, ...

Widget architecture

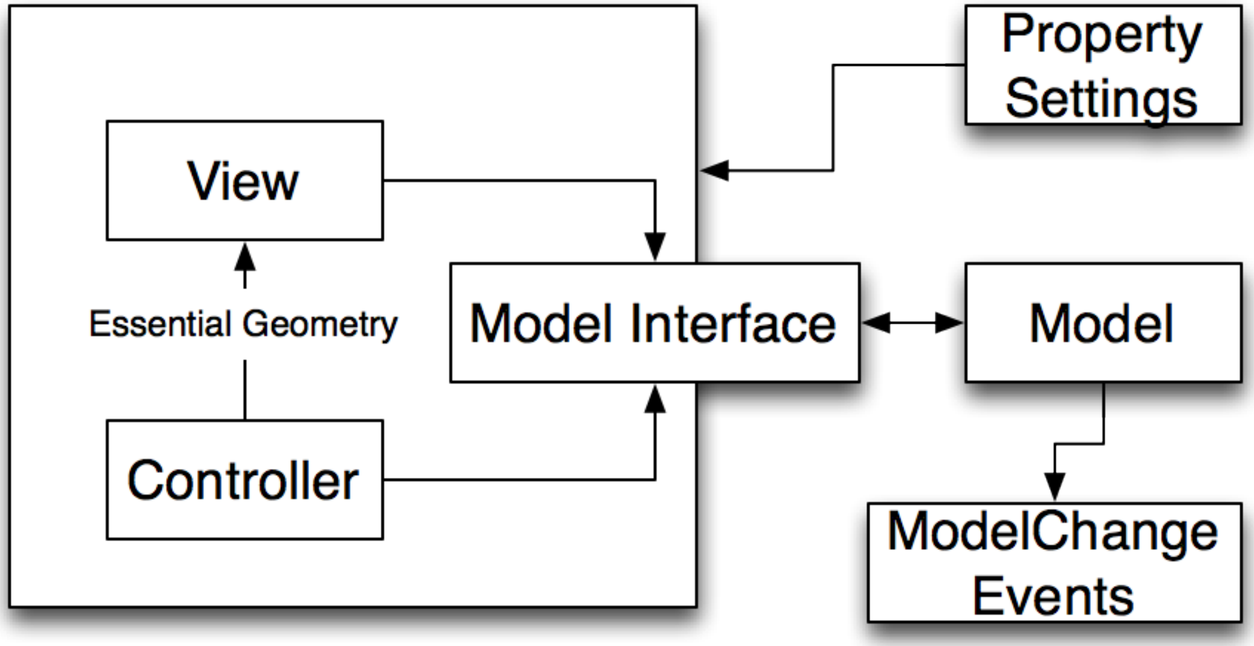


Example JButton

- In some ways, Java pushes MVC too far
- Consider `JButton` class (*see Java documentation*)
  - `JButton` extends `AbstractButton`
  - Check out `AbstractButton`
    - Contains a `ButtonModel` to support state information, listener information
    - Contains controller methods to `fireActionPerformed`
    - Contains an `EventListenerList` which contains a bunch of `EventListener` descendants (see declaration in tab)

Customization: Abstract Model Widgets

- More complex widgets expect the application to implement a model interface or extend an abstract class
- Examples: `JTable` and `JTree`



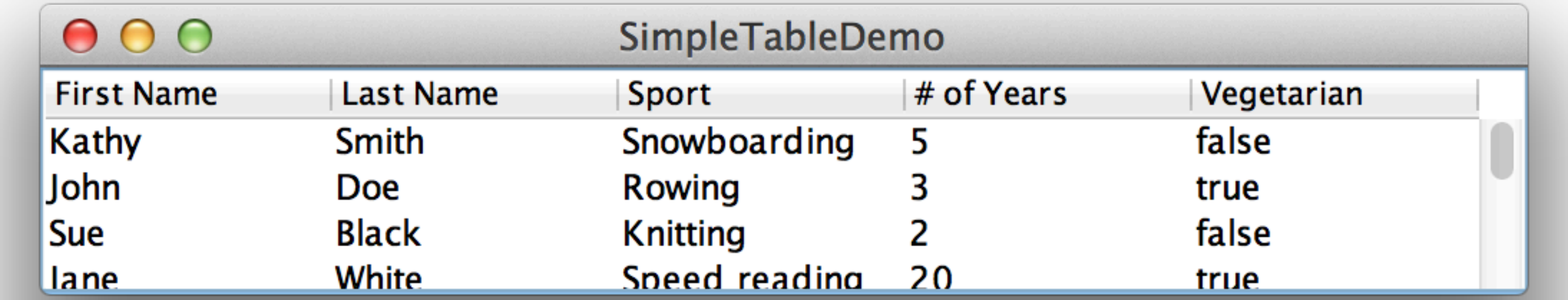
SimpleTable Demo Code

- Use default table model created by constructor:

```
JTable table = new JTable (data, columnNames);
```

- Add a scroll pane with this pattern:

```
JScrollPane scrollPane = new JScrollPane(table);
table.setFillsViewportHeight(true);
```



SimpleTableDemo

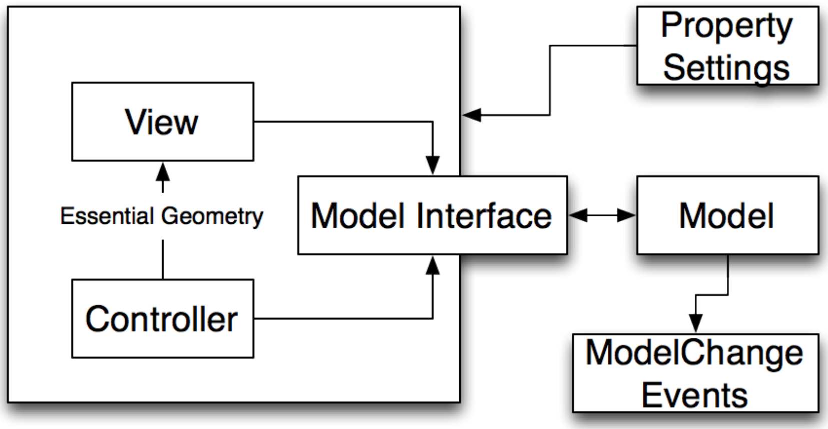
First Name	Last Name	Sport	# of Years	Vegetarian
Kathy	Smith	Snowboarding	5	false
John	Doe	Rowing	3	true
Sue	Black	Knitting	2	false
Jane	White	Speed reading	20	true

The sample code is not a clean enough design to emulate for CS349!

TableModel Interface

```
public interface TableModel {
    int getColumnCount();
    String getColumnName(int columnIndex);
    Class<?> getColumnClass(int columnIndex);
    int getRowCount();
    Object getValueAt(int rowIndex, int columnIndex);
    void setValueAt(Object aValue, int rowIndex,
          int columnIndex);
    boolean isCellEditable(int rowIndex, int columnIndex);
    void addTableModelListener(TableModelListener l);
    void removeTableModelListener(TableModelListener l);
}
```

- AbstractTableModel provides default implementations for most of these …



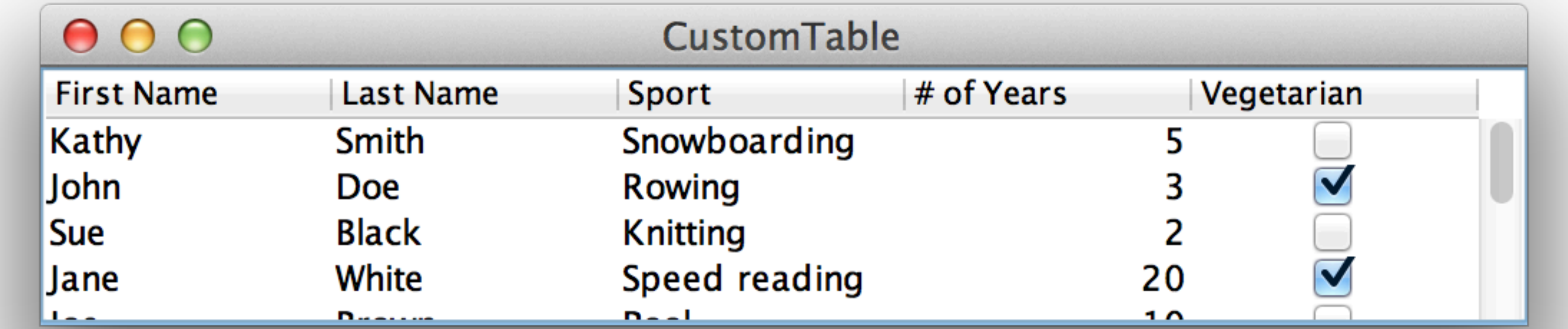
Custom JTable using AbstractTableModel

- To customize a JTable, you need to implement three methods of AbstractTableModel
  - public int getColumnCount();
  - public int getRowCount() ;
  - public Object getValueAt(int row, int col);

- Creates table of readonly columns with generic names
- To change this default behaviour, override:
  - public String getColumnName(int col);
  - public Class getColumnClass(int c);
  - public boolean isCellEditable(int row, int col);
  - public void setValueAt(Object value, int row, int col);

CustomTable Code Demo

- Inner table model class extended from AbstractTableModel
  - only *some* columns are editable
  - display Boolean type as checkbox
  - sets column names



CustomTable

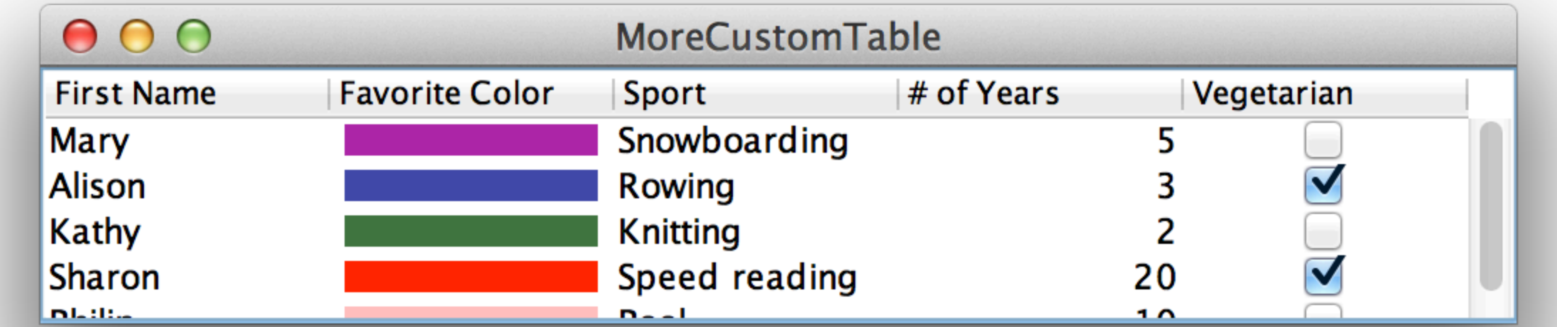
First Name	Last Name	Sport	# of Years	Vegetarian
Kathy	Smith	Snowboarding	5	□
John	Doe	Rowing	3	☑
Sue	Black	Knitting	2	□
Jane	White	Speed reading	20	☑
Joe	Brown	Pool	10	□

Firing Events in an AbstractTableModel

- Provides helper functions to fire events:
  - void fireTableCellUpdated(int row, int column);
  - void fireTableChanged(TableModelEvent e);
  - void fireTableDataChanged();
  - void fireTableRowsDeleted(int firstRow, int lastRow);
  - void fireTableRowsInserted(int firstRow, int lastRow);
  - void fireTableRowsUpdated(int firstRow, int lastRow);
  - void fireTableStructureChanged();

MoreCustomTable Code Demo

- Even more customization with
  - Custom TableCellRenderer
  - Custom TableCellEditor
- Can change default cell renderer/editor by class or column
- Also sets tool tip for cell
- Uses JColorChooser dialog



MoreCustomTable

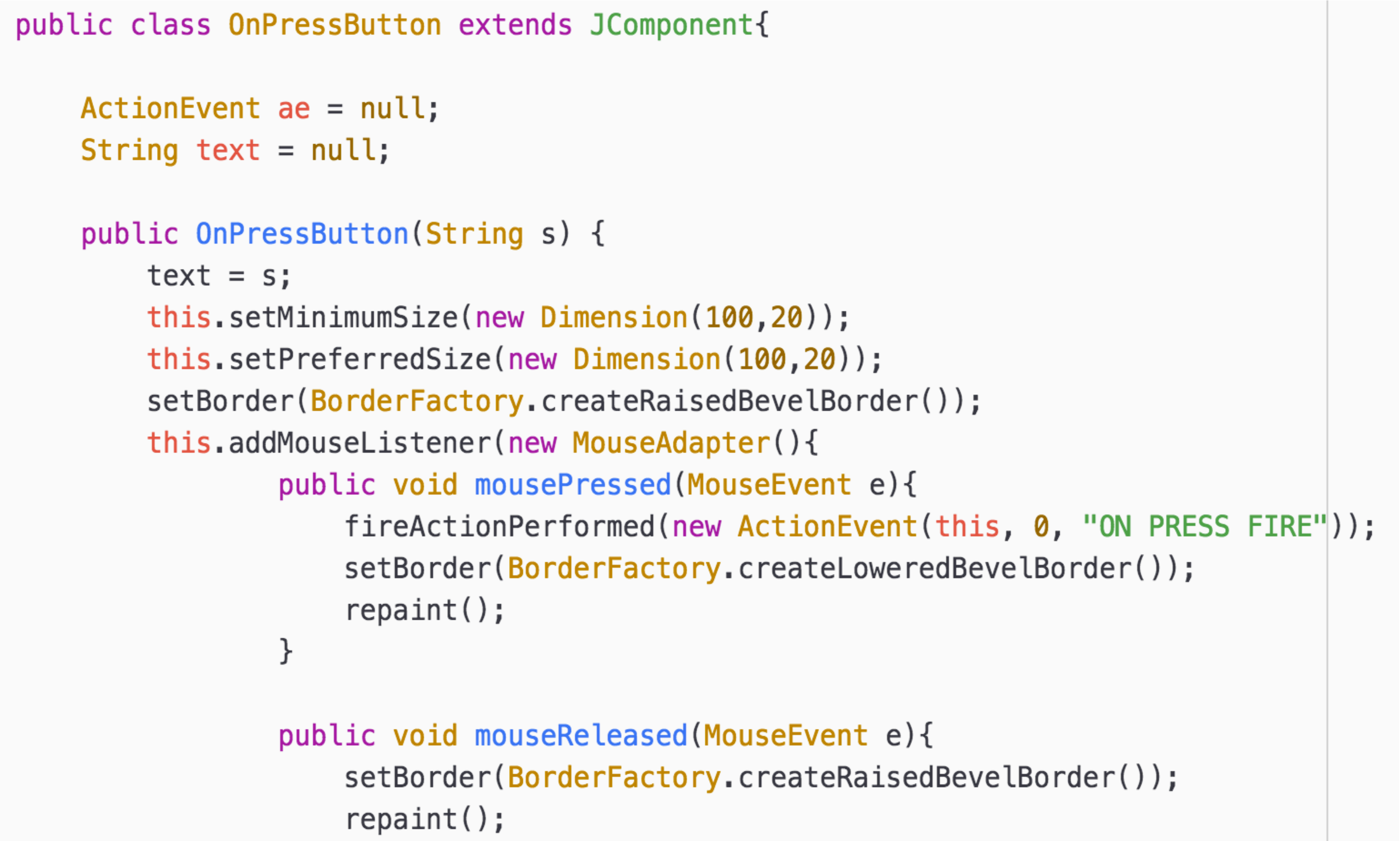
First Name	Favorite Color	Sport	# of Years	Vegetarian
Mary	🟪	Snowboarding	5	□
Alison	🟦	Rowing	3	☑
Kathy	🟩	Knitting	2	□
Sharon	🟥	Speed reading	20	☑
Philip		Pool	10	□

Customization with TableColumnModel

- The TableColumnModel has methods like:
  - void addColumn(TableColumn aColumn)
  - TableColumn getColumn(int columnIndex)
  - int getColumnCount()
  - int[ ] getSelectedColumns()
  - void moveColumn(int columnIndex, int newIndex)
  - void setPreferredWidth(int preferredWidth)
  - void setMinWidth(int minWidth)
  - void setResizable(boolean isResizable)
  - void setHeaderRenderer(TableCellRenderer headerRenderer)

- More details about JTable customization here:
  - http://docs.oracle.com/javase/tutorial/uiswing/components/table.html

Custom Control: OnPressButton

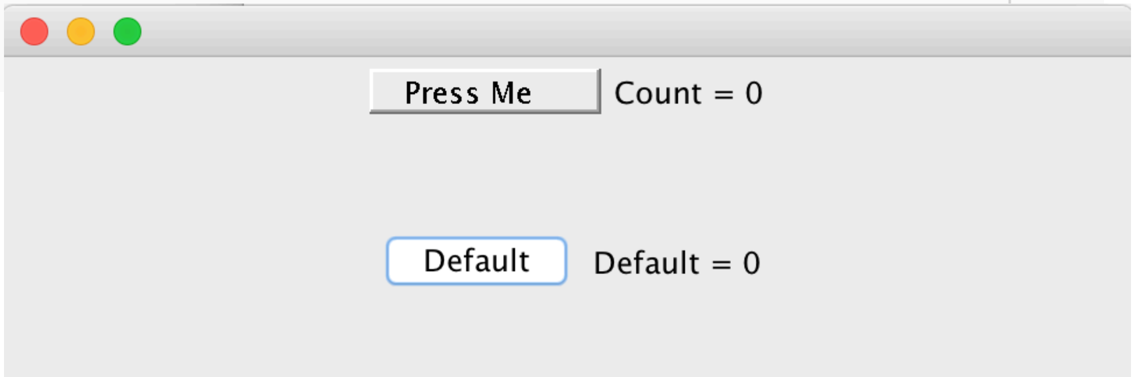


```
public class OnPressButton extends JComponent{

    ActionEvent ae = null;
    String text = null;

    public OnPressButton(String s) {
        text = s;
        this.setMinimumSize(new Dimension(100,20));
        this.setPreferredSize(new Dimension(100,20));
        setBorder(BorderFactory.createRaisedBevelBorder());
        this.addMouseListener(new MouseAdapter(){
                public void mousePressed(MouseEvent e){
                    fireActionPerformed(new ActionEvent(this, 0, "ON PRESS FIRE"));
                    setBorder(BorderFactory.createLoweredBevelBorder());
                    repaint();
                }

                public void mouseReleased(MouseEvent e){
                    setBorder(BorderFactory.createRaisedBevelBorder());
                    repaint();
                }
```



Summary

- Widgets are a fundamental building block of modern GUIs.
- Widget toolkits or libraries need to be complete, consistent, and customizable.
- MVC provides benefits at the widget-level as well!
  - Rich widget toolkits promote code reuse and simplicity
  - Separation of concerns enables programmers to more easily use a stock set of widgets to manipulate their unique application data.
    - Example: JTable
    - Because the model is separated out, it can be used to manipulate many kinds of data stored in many different ways.
    - More time and attention can be given to JTable itself to make it more robust and versatile.
- You aren’t constrained by the available widgets. Make your own if you need new functionality!