

CSE 3461 F10

Widgets

Outline

- What is a widget?
- Buttons
- Combo boxes
- Text components
- Message boxes

2

What is a Widget?

- A **self-contained screen object**
- Also called a **control**
- Examples of widgets:
 - buttons, check boxes, lists, scrollbars

3

Properties of Widgets

- A widget is usually **manipulable**
 - it has some *behaviour*, meaning that it responds to the user's actions on them
 - visually, by changing their appearance
 - auditorially, by producing a sound
- Widgets are used for both input and output
 - For input: user provides information to application, specifies commands to be performed
 - For output: application provides information to user (such as the application status)
- A widget may contain other widgets

4

Widgets are Important

- Widgets:
 - are the most obvious **visual contribution** of GUIs
 - are the **basic building blocks** of graphical user interfaces (GUIs)
 - provide **a means of “communication”** between users and software

5

Using Widgets

- can make use of **pre-defined widgets** or **create new ones**
 - a set of pre-defined widgets is included in the user interface toolkit (such as Swing)
 - sometimes referred to as the “set of canned controls”
- user interface toolkit also provides:
 - a mechanism to display widgets (layout management)
 - a means to design new widgets

6

Using Widgets

- Design questions
 - what widget (or set of widgets) is appropriate for this task?
 - what layout is appropriate for these widgets
 - given the available screen real estate
 - Is this widget intuitive to use?
 - does it make use of metaphors of physical controls from the real-world?

7

Outline

- What is a widget?
- **Buttons**
- Text components
- Combo boxes
- Message boxes



8

Imperative Controls

- typically, imperative control that says:
“take this action and take it immediately”
 - the recipient of the action might be already implied
 - e.g., the recipient of the “close” on the toolbar is always the toolbar’s window
 - the recipient might be modifiable
 - e.g., the recipient of the “bold” command in a word processing application has a default recipient, but it can be changed
- imperative control usually has a <verb> or a <verb> <noun> structure

9

Push Buttons

- The most common type of button
- Identified by visual features that suggest *pressability*
 - simulated 3D raised aspect
 - shadow on right and bottom; highlight on top and left
 - shows whether the button is raised or indented
 - buttons might also be “painted” on the screen
 - they don’t actually “move” when clicked
 - this provides poor visual information

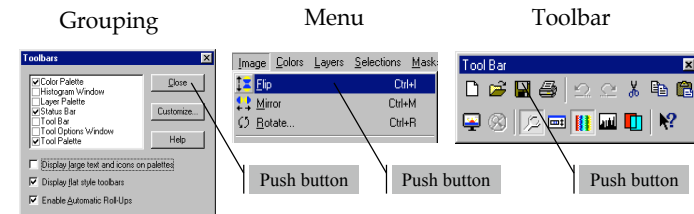
10

Push Buttons

- A button is pushed to invoke a command
 - with mouse:
command executes as soon as the user clicks (a mouse click is the sum of mouse pressed and released; as opposed to mouse press or mouse release) ActionListener
 - keyboard:
button needs focus, command executes with key press (as opposed to key release)
- Button needs to **indicate to the semantics of the command** that it invokes
 - it can do this with an icon, text, or both

11

Examples of Push Buttons



12

Three Devices for Organizing Push Buttons

- Groupings:
 - one or more buttons in proximity
 - meaning signaled by text or icon
- Menus:
 - a collection of buttons in a list
 - the list appears and disappears
 - dependent upon a button action
 - meaning typically signaled by text or icon
 - relative position in list carries meaning too

13

Three Devices for Organizing Push Buttons

- Toolbars:
 - a collection of buttons, typically organized horizontally
 - the collection is *persistent*
 - appropriate for commonly used functions (we'll contrast later with menus)
 - meaning typically signaled by an icon
 - text is *less opaque*, but this is a compromise to cope with reduced screen real estate

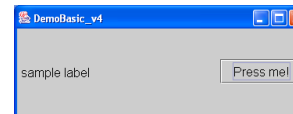
14

Butcons

- butcon = half button, half icon
- used in toolbars
- clues about its *pressability* **removed**
 - information that it is pressable becomes apparent only when pointed at
 - butcons are more difficult than buttons for newcomers
- In theory, butcons easy to use:
 - always visible (therefore easy to memorize)
 - requires less time, dexterity than drop-down menu
 - inextricably linked to toolbars
 - make use of icons, which can be difficult to decipher
 - ToolTips (rollovers) are a solution to this problem

15

Examples



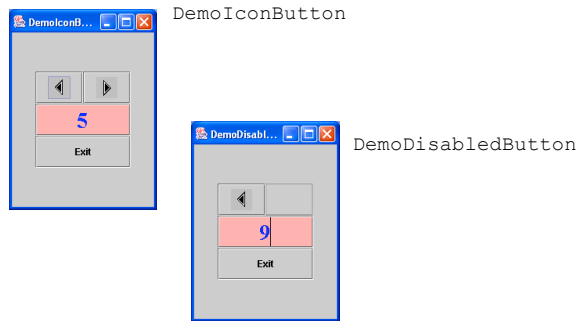
DemoBasicV4



DemoBasicButton

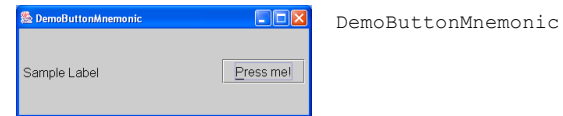
16

Examples



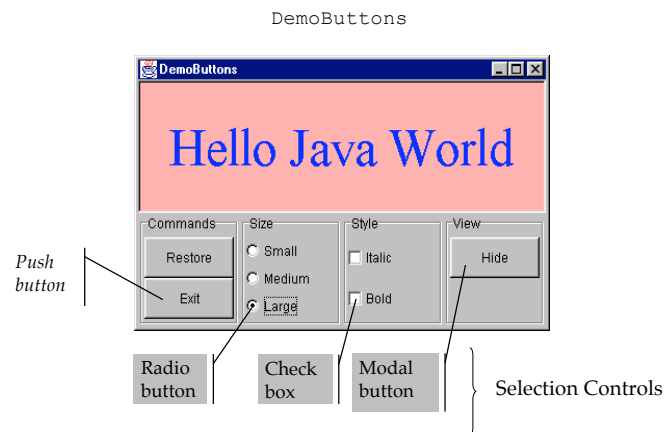
17

Examples



18

Selection Control



19

Radio Buttons

- Used to present a set of *mutually-exclusive* options (the *domain* of options)
- Come in groups of two or more
 - a single radio button is undefined
- Require a substantial amount of screen real estate
 - this use of space must be justified
 - E.g., it is necessary to show the user the full set of available choices

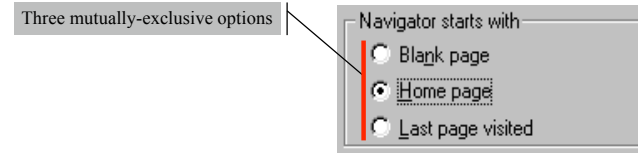
20

Radio Buttons

- The domain of options must be:
 - finite
 - small
 - mutually exclusive
- The name “radio button” is derived from the selector buttons used to select pre-set radio stations

21

Radio Button Example



22

Radio Button Example

In DemoButtons

```
smallButton = new JRadioButton("Small");
mediumButton = new JRadioButton("Medium");
largeButton = new JRadioButton("Large");
mediumButton.setSelected(true);
```

need to create a multiple-exclusion scope for this set of buttons:

```
ButtonGroup sizeGroup = new ButtonGroup();
sizeGroup.add(smallButton);
sizeGroup.add(mediumButton);
sizeGroup.add(largeButton);
```

- need to add each button to an intermediate container (*not* the group)
- need to register an `ActionListener` on each button

23

Check boxes

- One of the first visual controls invented
- Used to select or deselect an option
- The check mark provides feedback
- Primarily text-based
 - the graphic supports the text, not the other way around

24

push button → butcons checkbox → ??

- The push button evolved into the butcon
 - its text was replaced with an icon
 - it was migrated on the toolbar
- What is the equivalent for a check box?
 - if we press a button and it stays recessed (“pushed in”)
 - the button *latches*
 - this is the same as a check box
 - a *latching butcon* is equivalent to a checkbox
 - allows us to dispense with text

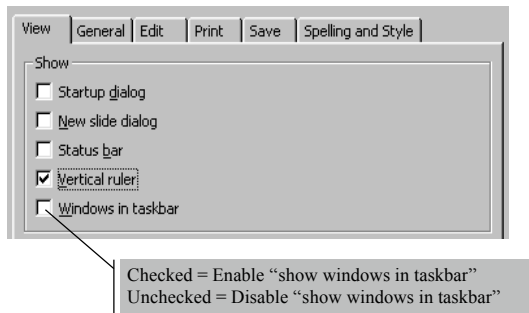
25

Check boxes

- Checkboxes are appropriate when:
 - the feature or characteristic has precisely two states (e.g., lightbulb = on | off)
 - States are appropriately described as being “enabled” / “disabled” or “on” / “off”
- Checkboxes are inappropriate when:
 - the feature or characteristic has more than two states (e.g., season = spring | summer | autumn | winter)
 - enabled/on and disabled/off are inappropriate descriptions of the states (e.g., gender = male | female)

26

Check box Example



27

Check box Example

- In DemoButtons

```
italicCheckBox = new JCheckBox("Italic");  
boldCheckBox = new JCheckBox("Bold");
```

- need to add each check box to an intermediate container; don't need ButtonGroup, like radio buttons
- need to register an ItemListener on each button
- need to implement the method `itemStateChanged`, invoke the method `isSelected()` on the check box instance

28

Modal Buttons

- Also called flip-flop button
- Used to select from multiple options
 - A hybrid of a push button and a radio button
- Looks like a push button
 - When you push it, the selected option changes
 - Should the user interpret the button text as
 - a description of the currently-selected option
 - a description of the option that will be selected if the button is pushed
 - *Need to spell it out clearly*

29

Modal Button Example

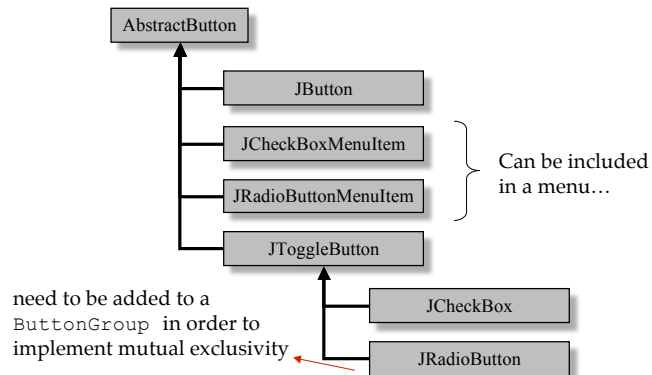
- In `DemoButtons`

```
showHideButton = new JToggleButton("Hide");
```

- The text on the button is the button's *action command*
- register an `ActionListener` on the button
- all instances of `ActionEvent` that are generated by the toggle button have an associated *action command*
- use the method `getActionCommand()` to determine what it is


30

Java's Button Classes



31

Outline

- What is a widget?
- Buttons
- Combo boxes 
- Text components
- Message boxes

32

Selection Control, Part II

Consider the interface to `DemoButtons`
Suppose we want to present a **variety** of font sizes,
and a **variety** of font types?



33

The Solution is to Use Combo Boxes

`DemoComboBox`



34

Combo Boxes

- An alternative to radio buttons
 - Appropriate when we have a *large number* of mutually-exclusive options
- Advantage over radio buttons
 - More choices can be displayed in less screen space
- Disadvantage over radio buttons
 - Choices are not displayed until combo box is selected

35

Combo Box Example

- In `DemoComboBox`

```
final String[] SZ
    = { "10", "14", "18", "22", "26", "32", "38", "48" };

sizeCombo = new JComboBox(SZ);
fontCombo = new JComboBox();
```

- for `fontCombo`, need to add the items to the list

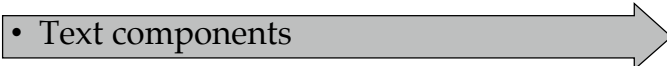
```
for (int i = 0; i < fontList.length; ++i)
    fontCombo.addItem(fontList[i].getName());
```

- register an `ActionListener` on each combo box
- selections from the list will generate an `ActionEvent`
- use the method `getSelectedItem()` to determine which list element was selected

36

Outline

- What is a widget?
- Buttons
- Combo boxes
- Text components
- Message boxes



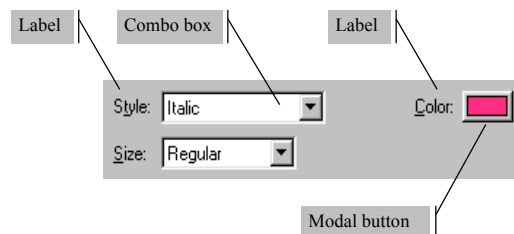
37

Types of Text Components

- Output components
 - cannot be edited
 - Labels, Labeled borders (JLabel, TitledBorder)
 - Tool tips, Message Boxes
- Input/output components
 - can be edited
 - Text fields, Text areas, Editable combo boxes
 - Dialog boxes

38

Labels: Example



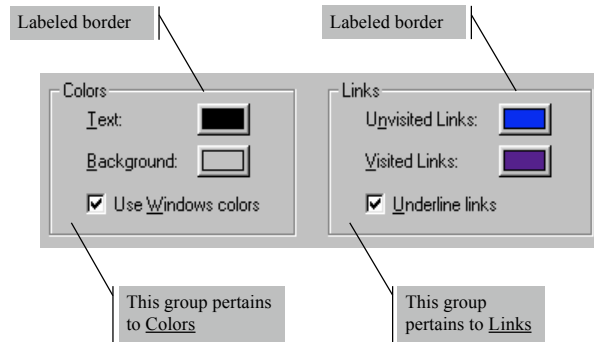
39

Labels

- Create using JLabel
- Do not react to input events, cannot get keyboard focus
- Used to display information
 - in particular, placed adjacent to a component that has a keyboard alternative but can't display it
- Position is determined by Layout Manager
- Advantage:
 - The information it provides can be useful; aid user's performance of task
- Disadvantage:
 - Uses screen real estate
 - Poor wording may be worse than none at all

40

Labeled Borders: Example



41

Labeled Borders

- The `setBorder` method is defined for all instances of `JComponent`
 - Used to create visual clue about groupings
 - A label for the grouping is optional
- The parameter is an instance of a `Border`
- `Border` is an interface
 - `AbstractBorder` is an abstract class that implements it
 - `TitleBorder` extends `AbstractBorder`

E.g., in `DemoButtons`:

```
JPanel sizeGroupPanel = new JPanel();
sizeGroupPanel.setBorder(
    new TitledBorder(new EtchedBorder(), "Size"));
```

42

Quick Note About Groupings

- Grouping can reduce cognitive load
 - E.g., consider 12 components:
 - With borders around each of 3 groupings, each with 4 components, the user identifies the group first, then the item within the group (two-step process)
 - Without grouping, user must locate item from among 12 items (this is more difficult!)
- Grouping can have disadvantages:
 - Uses screen real estate
 - Organization of components requires knowledge of task domain
 - Poor grouping may be worse than none at all

43

Text Fields and Text Areas

- Different types:
 - Text Field: single line
 - Text Area: multiple lines
 - The abstract class `JTextComponent` has the subclasses `JTextField`, `JTextArea`
- Challenges for design:
 - How to validate the text that has been input?
 - How to navigate within and between text elements?

44

Editable Combo Boxes

- Similar to a combo box, except that user may also enter text directly
- Same challenges as text fields and areas
 - validation, navigation
- Editable and non-editable combo boxes are both instantiated from `JComboBox`
 - use the methods: `setEditable(true)`, `setEditable(false)`

45

Navigation

- All components have a *focus state*
 - The possible focus states are *in focus* or *out of focus*
 - For a key press to affect a component, the component must have focus
 - Visual clues are given to show which component has focus
 - I-beam cursor appears, special highlighting
- Every time the focus changes, a `FocusEvent` is generated
 - a component loses focus, another gains focus,

46

Navigation

- A component generally gains the focus by the user:
 - clicking it
 - tabbing to it, or
 - otherwise interacting with a component.
- A component can also be given the focus programmatically
 - e.g., a component can request the focus when its containing frame or dialog is made visible
- The *focus traversal policy* determines the order in which a group of components are navigated

47

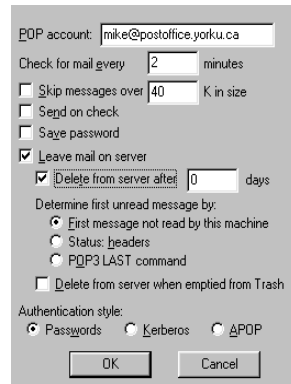
DemoLookAndFeel.java



48

Analysis Exercise

How might the organization of these widgets be improved?



49

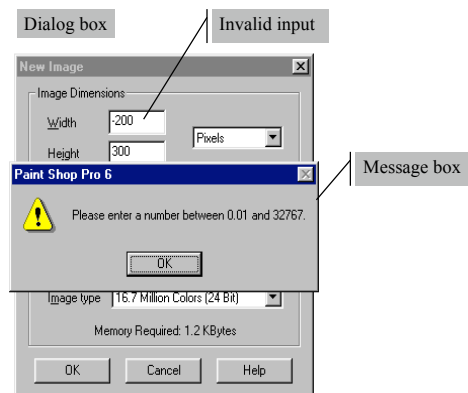
Outline

- What is a widget?
- Buttons
- Combo boxes
- Text components
- Message boxes



50

Message Box: Example



51

Message Boxes

- A message box (aka dialog box) is a popup window
- Primary purpose is to govern the interaction
 - presents a text message to the user
 - seeks input for confirmation (and to close the box)
- Functions to:
 - Notify the user of a problem (e.g., invalid choice)
 - Notify the user of potentially destructive outcome (e.g., overwrite a file)
 - Provide information

52

Message Boxes (2)

- Advantage
 - Comprehensive messages are possible (unlike tool tips)
- Disadvantage
 - Slows interaction (because underlying thread is halted until confirmation is received)

53

Behaviour of Message Boxes

- Message boxes demand immediate attention
 - can't close the message box (user is required to make a choice or to provide confirmation)
 - user is not able to make use of other widgets

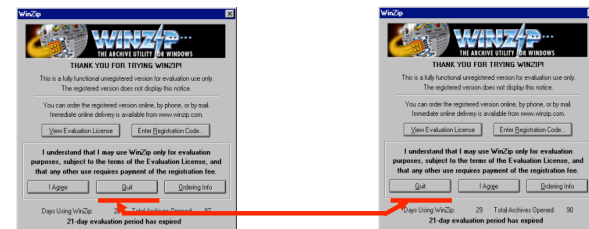
54

Input-Handling Techniques

1. For actions with serious consequences, require an explicit button click or key press before proceeding (pressing Enter does not result in a default action)
2. For invalid input, provide feedback (e.g., alarm tone or visual feedback)
3. Take advantage of user consistency (e.g., change the position of buttons from one invocation to the next)

55

Example



Button positions change from one invocation to next

Hitting ENTER produces...



56

Example 4.6

DemoMessageBox.java

