Lecture 8: Design Guidelines

The GUI development process

• We’ve now seen some tools that enable us to build our interface.
• So let’s get back to the GUI process...
  - How do we go from idea to design?
  - How do we specify and evaluate this design?
  - How do we go from design to implementation?

Focus on design

• The assumed process to this point:
  - (1) we have an idea for an interface
  - (2) we understand our intended users
    • that is, we’ve got personas for our system
    • let’s ignore requirements gathering, etc. for now
• We need to specify an initial design, or prototype, for our interface.
• We will examine 3 constraints to guide us:
  - (high-level) theories
  - (mid-level) principles
  - (low-level) guidelines

Theories

• Two types of theories useful for our purpose
  - explanatory: given a phenomenon, try to understand its causes, effects, etc.
  - predictive: given a task/interface, try to predict what will/can happen in the course of execution
• Same idea as theories in other domains!
  - physics: how does a ball fly through the air?
  - chemistry: what happens when we mix X & Y?

2 Na + 2 H_2O → 2 NaOH + H_2
Theories

- Let's propose a theory of mouse movement.
- How do people select a menu? Assume...
  - mouse starts somewhere on the screen
  - user clicks down, selects option, releases
- What might we try to explain?
  - time to execute
  - event sequence
  - errors

### Tasks

<table>
<thead>
<tr>
<th>Task</th>
<th>Help</th>
</tr>
</thead>
<tbody>
<tr>
<td>Navigator</td>
<td>1</td>
</tr>
<tr>
<td>Mail/Groups</td>
<td>3</td>
</tr>
<tr>
<td>Instant Messenger</td>
<td>5</td>
</tr>
<tr>
<td>Composer</td>
<td>4</td>
</tr>
<tr>
<td>Address Book</td>
<td>5</td>
</tr>
<tr>
<td>Privacy &amp; Security</td>
<td>1</td>
</tr>
<tr>
<td>Tools</td>
<td>1</td>
</tr>
<tr>
<td>Welcome to Netscape 6.2!</td>
<td>1</td>
</tr>
</tbody>
</table>

Theories

- Taxonomies
  - puts order on complex set of phenomena
  - we usually think of animals -->
  - remember Byrne et al.'s “taskonomy”?

Theories

- Fitts' Law
  - how long does it take to move a mouse A→B?

$$T = c_1 + c_2 \log\left(\frac{2d}{w}\right)$$

if B has width w and is at a distance d...

Theories

- GOMS modeling
  - Goals, Operators, Methods, Selection Rules
  - “keystroke-level” model of human performance
    - one version of GOMS, anyway (KLM)
  - primary operators
    - K: keystroke  280 ms
    - P: point with mouse to a target  1100 ms
    - B: press or release mouse button  100 ms
    - BB: click mouse button  200 ms
    - H: “home” hands to kb/mouse  400 ms
    - M: “mental” act of thinking  1200 ms
Theories

- **GOMS modeling**
  - phone example

<table>
<thead>
<tr>
<th>Description</th>
<th>#</th>
<th>Type</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>“turn on phone”</td>
<td>1</td>
<td>M</td>
<td>1.20</td>
</tr>
<tr>
<td>press ‘Phone’</td>
<td>1</td>
<td>M</td>
<td>.28</td>
</tr>
<tr>
<td>“dial number”</td>
<td>1</td>
<td>M</td>
<td>1.20</td>
</tr>
<tr>
<td>type digits</td>
<td>10</td>
<td>K</td>
<td>2.80</td>
</tr>
<tr>
<td>“send number”</td>
<td>1</td>
<td>M</td>
<td>1.20</td>
</tr>
<tr>
<td>press ‘Send’</td>
<td>1</td>
<td>K</td>
<td>.28</td>
</tr>
<tr>
<td><strong>Total Time</strong></td>
<td></td>
<td></td>
<td>6.96</td>
</tr>
</tbody>
</table>

Principles

- Theories provide general descriptions of human behavior, performance, etc.
  - very valuable for understanding
  - so far, have had more limited practical value
- Principles embody particular aspects of theories applied in context
  - still somewhat general, but more prescriptive
    - they tell you what to do
    - though they may not tell you specific steps to doing it
- Let’s look at some recognized principles of design (as found by practitioners)...
Principle #1

• Novice / first-time users
  - novice are truly novice; first-timers have skills, but just don’t know the interface
  - rules of thumb...
    • restrict vocabulary to simple, necessary terms
    • keep (sub-)tasks to minimal number of actions
    • provide informative feedback, simple error messages, and on-line step-by-step help

Principle #1

• Intermittent / intermediate users
  - have the basic knowledge, but don’t see the interface often enough to become expert
  - rules of thumb...
    • “take off the training wheels”
    • consistent structure of views & menus
    • consistent terminology
    • visuals emphasize recognition over recall
    • “protection from danger”
    • reference materials (on-line or printed)

Principle #1

• Frequent / expert users
  - “Power” users thoroughly familiar with both task and interface
  - rules of thumb...
    • rapid response times all around
    • macros (built-in or recordable) for frequent sequences of commands
    • shortcuts: keystrokes, abbreviations, etc.

Principle #2

• P#2: Use the Eight Golden Rules of Interface Design
  • (1) Strive for consistency (again).
    • check all aspects of the interface for consistency
    • terminology in prompts, menus, help
    • text/image color, layout, caps, fonts
    • there are exceptions... but limited
  • (2) Enable frequent users to use shortcuts.
    • just said this, but worth repeating
Principle #2

• (3) Offer informative feedback.
  - every action (within reason) should get feedback
  - frequent/minor actions → subtle feedback
    • e.g., dragging; how about clicks/keystrokes?
  - infrequent/major actions → obvious feedback
    • though limit use of most obvious! (e.g., dialogs)

• (4) Design interaction to yield closure.
  - after performing subtask (sequence of actions),
    user should feel closure
  - (“take one dip and end it!”)

• (5) Offer error prevention and handling.
  - users should not be able to make a serious error
    • OS’s do this (memory protection: no “poke 1,0”)
    • UI’s are no different!
  - if system detects error, recovery should be quick
    • e.g., don’t re-type command -- fix faulty part

• (6) Permit easy reversal of actions.
  - one word: UNDO
  - increases comfort level, decreases frustration
  - encourages exploration of the unfamiliar

Principle #2

• (7) Support internal locus of control.
  - make the user feel like they’re in control
  - surprises are good for horror movies, not UI’s
  - “avoid acausality” (huh?)
    • if you know the cause & effect, you’re happy

• (8) Reduce short-term memory load.
  - humans remember 7±2 things
    (normal humans, not Drexel students :) --
    they’re more likely around 10 or 12)
  - the more to recall, the more expertise is needed
    for efficient use
  - let users find functions, not recall them

Guidelines

• Guidelines are more specific, prescriptive
  methods for designing interfaces
  - line between principles and guidelines is fuzzy
  - actually, some use the terms interchangeably

• You be the judge...
  - Let’s look at some sample guidelines...
    - and maybe some of this will sound familiar...
Guidelines for Display Organization

• (1) Consistency of data display
• (2) Efficient information assimilation
  - use common formatting (e.g., left vs. right justification) for text, numbers, etc.
• (3) Minimal memory load on user
• (4) Compatibility of display with data entry
  - clear ordering, clear links
• (5) Flexibility for user control of display
  - allow users to change columns, sorting, etc.

Guidelines for Data Entry

• (1) Consistency of entry transactions
• (2) Minimal user input actions
• (3) Minimal memory load on user
• (4) Compatibility of display with data entry
• (5) Flexibility for user control of entry
  - but beware (!): when does flexibility interfere with consistency guideline?

Putting it all together

• We have theories, principles, & guidelines.
  - actually, theories → principles → guidelines
• But is there something specific to follow?
• Yes: Java Look-and-Feel Design Guidelines
  - there are others...
    • Apple’s Human Interface Guidelines
      - the original, the classic!
    • Microsoft’s Windows Human Interface Guidelines
  - all are very useful because they’re detailed, comprehensive, standard, and publicly available
  - Java maps best to onto our course...
  - but let’s look at Apple’s for a minute...

Apple Guidelines

• “Following the guidelines is to your advantage because…”
  - users will learn your application faster if the interface looks and behaves like applications they’re already familiar with
  - users will accomplish their tasks quickly, because well-designed applications don’t get in user’s way
  - users with special needs will find your product more accessible
  - your application will have the same modern, elegant appearance as other Mac OS X applications
Apple Guidelines

• (quote continued)
  - your application will be easier to document, because an intuitive interface and standard behaviors don’t require as much explanation
  - customer support calls will be reduced
  - your application will be easier to localize, because Apple has worked through many localization issues in the Aqua design process
  - media reviews of your product will be more positive; reviewers easily target software that doesn’t look or behave the way “true” Macintosh applications do

Apple Design Principles

• (1) Metaphors from the real world
  - use concrete metaphors and make them plain, so users have expectations to apply to computer environments
  - whenever appropriate, use audio and visual effects that support the metaphor

• (2) Direct manipulation
  - users want to feel in charge of the computer’s activities

• (3) See-and-point
  - (instead of remember-and-type)
  - users rely on recognition, not recall; they shouldn’t have to remember anything the computer already knows
  - the general form of user actions is noun-then-verb, or “Hey, you --- do this”

Apple Design Principles

• (4) Consistency
  - effective applications are both consistent within themselves and consistent within one another

• (5) WYSIWYG
  - (What you see is what you get)
  - there should be no secrets from the user, no abstract commands that only promise future results
  - there should be no significant difference between what the user sees on the screen and what eventually gets printed

• (6) User Control
  - user, not computer, initiates and controls all actions

Apple Design Principles

• (7) Feedback and dialog
  - keep the user informed
  - provide immediate feedback
  - user activities should be simple at any moment, though they may be complex taken together

• (8) Forgiveness
  - users make mistakes; forgive them
  - the user’s actions are generally reversible; let users know about any that aren’t
Apple Design Principles

• (9) Perceived stability
  - users feel comfortable in an environment that remains understandable and familiar rather than changing randomly

• (10) Aesthetic integrity
  - visually confusing or unattractive displays detract from the effectiveness of human-computer interaction
  - different “things” look different on the screen
  - users should be able to control the superficial appearance of their computer workplaces -- to display their own style and individuality
  - messes are acceptable only if users make them -- applications aren’t allowed this freedom

Apple Guidelines

• Ok, so far these principles sound like many of the others we’ve seen.

• What’s the difference? Specific guidelines that go with the general principles...
  - somewhat specific: “Try for a more centered approach to dialog layout as opposed to the strongly left-biased approach…”
  - more specific: “Maintain a 20-pixel space between the left and right edge of the window and any controls”
  - more specific: “Set 16 pixels of vertical space between groups of controls”

Apple Guidelines

• The Apple Guidelines include a variety of information, some important, some not
  - not important: properties of Apple look-and-feel
  - not important: Apple-specific components
  - important: ideas behind nice, clean layout

Java Guidelines

• The Java Guidelines work best for us -- general and specific all relevant to Swing -- namely, to their “Metal” look-and-feel

• What do the guidelines guide?
  - “flush 3D” style, “drag” texture, color...
**Our approach**

- What we’ll do...
  - we’ll look at various guidelines, general (from various sources) and specific (from the Java Guidelines)
  - when we have a specific rule, we’ll underline it
    - “Maintain a 12-pixel space between the left and right edge of the window and any controls”
  - as the term progresses, we’ll continually expand our set of rules
  - it’s these rules that you’ll follow in developing your larger-scale GUI projects

**Windows**

- Windows form the basis of our GUI
- The 4 basic types...

**Primary windows**

- Primary windows are handled mostly by the particular platform you’re on
  - OS handles title/outline, moving, resizing, etc.
  - Swing draws the content (assuming Metal feel)
  - e.g., on MS Windows...

**Secondary windows**

- Typically, dialog & alert boxes
  - again, OS handles outside, Swing handles inside
  - Key: dependent on primary window
    - if primary closes, secondary closes
Utility windows

• Typically, used to display a collection of tools, colors, or patterns

• Unlike secondary windows, are not dependent on primary window
  - thus, doesn’t close when primary closes

Plain windows

• Typically, “splash screen” windows
  - that is, the window “announcing” the application as seen on launch

Creating windows

• What Swing classes do we use for each?
  - primary windows: `window = new JFrame();`
  - secondary windows: `window = new JDialog();`
  - utility windows: `window = new JWindow();`
  - plain windows: `window = new JFrame();`

• Luckily, all these behave very similarly
  - just create them & lay them out as usual
  - in a bit we’ll talk more about putting them all together to work smoothly
  - for now let’s focus on a single window

Window visual design

• Say we’re designing a new window.
  - we know components should go in the window
  - but where do we put these components?

• We’ll rely on Java’s Human Interface Guidelines to specify how to do this
  - as discussed before, the guidelines aren’t perfect
  - but they do give us a cohesive, integrated picture
    • all specifications go together into a unified theme
    • think style sheets in HTML...
Layout process

• (1) Determine the functional requirements for your dialog box and the type and importance of the dialog box components
  - example: “Find” dialog box
  - what information do we need?
    • find text
    • options: match case? whole word?
  - what components do we need?
    • text box for find text
    • check/radio buttons (etc.) for options
    • command buttons: “Find”, “Close”

Layout process

• (2) Create a window with 12-pixel margins between the border of the dialog box and its components
  - Include 12 pixels between the top and left borders of a dialog box and its components.

(We interrupt this program...)

• If you read the Java Human Interface Guidelines, you’ll see things like:
  - “Include 11 pixels between the bottom and right borders of a dialog box and its command buttons. (To the eye, the 11-pixel spacing appears to be 12 pixels because the white borders on the lower and right edges of the button components are not visually significant.)”

• We won’t worry about this 1 pixel. Either one is fine.
Layout process

• (4) Create a design grid, using the number and width of the rest of the components to decide the number of columns and the column width for the grid.
  - design grid = rows & columns to structure design
    • we’re not talking a strict grid
    • we’re not talking GridLayout’s
    • essentially, we’re talking guides to help with horizontal and vertical alignment

• (4) Continued...
  - design grid example for “Find” box
    - guidelines suggest that grid derives from command buttons and spacing
    - we won’t abide by this — but having a grid roughly this size is a good idea
      • that is, pick a grid and stick to it

Layout process

• (5) Add the components to the dialog box in their logical sequence.
  - When designing a dialog box, place the most important options, or those you expect users to complete first, prior to others (in reading order).

• (6) Align related components using the design grid.
  - Align related components vertically using a chosen design grid column, and horizontally using a chosen design grid row.

Component specifics

• Labels
  - Place labels before the component to which they refer (generally, above and to the left.)
  - Align labels with the tops of their components.
  - Insert 12 pixels between the trailing edge of a label and any associated components.
  - Left-align all vertically aligned labels.
    • Insert 12 pixels after the edge of the longest label.
Component specifics

- Command buttons
  - Center all button text, with a 12-pixel left & right boundary between button label and button edge.
  - Make all grouped command buttons equal width, which doesn’t change when window is resized.
  - Space buttons in a group 6 pixels apart.
  - Make the default button the left-most button.
  - Insert 17 pixels between these & rest of window.

- Radio & check-box buttons
  - Align the leading edge with that of other components.
  - Space grouped buttons 6 pixels apart.
  - Space separate groups 12 pixels apart.

- Toolbar buttons
  - Space individual toolbar buttons 2 pixels apart.
  - Space groups of toolbar buttons 11 pixels apart.
  - Include 3 pixels of space above and below all toolbar buttons.

- Interface text
  - Most components use “headline” capitalization.
    - [capitalize every word except articles, conjunctions, and short prepositions]
    - Checkbox text (Automatic Save Every Five Minutes)
    - Combo box text (Centimeters)
    - Command button text (Don’t Save)
    - Icon names (Trash Can)
    - Labels for buttons or controls (New Contribution To:)
    - Menu items (Save As…)
    - Menu titles (View)
    - Radio button text (Start at Top)
    - Titles of windows and dialogs (Color Chooser)
    - Tool tips (Cut Selection)
Component specifics

- Interface text (continued)
  - Alerts, errors, and long labels use “sentence” capitalization.
    - [capitalize only the first word in the sentence]
    - Alert box text
      ("The document you are closing has unsaved changes.")
    - Error or help messages
      ("The printer is out of paper.")
    - Labels that indicate changes in status
      ("Operation is 75% complete.")

Implementation of guidelines

- Not surprisingly, Swing is set up to implement the specified guidelines
- Some tips to help you...
  - hard spacing around buttons, labels, etc.
    pane.add(Box.createRigidArea(new Dimension(0,5)));
  - spacing around main window with border
    Border spacer = BorderFactory.createEmptyBorder(top, left, bottom, right);
    component.setBorder(spacer);

- Implementation of guidelines (continued)
  - layout managers are your friends for consistent alignment and spacing
    - e.g., BorderLayout ‘South’ panels for command buttons
    - e.g., GridLayout to get equal-sized command buttons

Dialogs & Alerts

- Dialog boxes
  - secondary window in which user performs task that is supplemental to the primary window
  - typically laid out by programmer

- Alert boxes
  - secondary window that provides brief, temporary interaction with user
  - typically laid out by Swing (built-in)
Types of dialogs

- Modal: blocks all other user input
- Modeless: can still interact with main frame
- Official word:

  Use modeless dialog boxes whenever possible.

  Use modal dialog boxes when interaction with the application cannot proceed while dialog box is displayed.

- In practice, this isn’t easy

Dialog box design

- Basic design
  - In dialog boxes, include mnemonics for all user interface elements except default and Cancel.
  - When opening a dialog box, provide initial keyboard focus to the logical first component.
  - Consider the effect of internationalization on your design.
    - Use layout managers!

Dialog box design

- Tab traversal order
  - Specify a logical tab traversal order for the user interface elements in a dialog box.
  - The default traversal order is the sequence in which you add the components to the dialog box.
    - do it this way, it’s easiest
    - but there’s also another, more excruciating way

Dialog command buttons

- General rules
  - Place command buttons that apply to the dialog box as a whole in the command button row at the bottom of the dialog box.
  - Align buttons in the command button row along the lower-right edge of the dialog box.
    - note: the alignment of command buttons in alert boxes supplied by Swing is different from the standard alignment in dialog boxes (why?!!)
  - Place command buttons that apply to one or a few components next to their associated components.
    - e.g., “Browse” button at the trailing edge of the text field it fills in
Dialog command buttons

• “OK” and “Cancel” buttons
  - best for *single-use* dialogs, modal or modeless
  - in most cases, “OK” is the default button
  - When users click “OK”, save the settings (or carry out the commands) and close the dialog.
    - Whenever possible, provide the button with a command name that describes the action (such as Print or Find) instead of OK.
    - When users click “Cancel”, close the window and restore the settings to the state they were in when the dialog box was opened.

Dialog command buttons

• “Apply” and “Close” buttons
  - best for *multiple-use*, modeless dialogs
  - in most cases, “Apply” is the default button
  - Use “Apply” to carry out the changes users specify in a dialog box without closing the window.
    - Whenever possible, use a specific action name (such as “Find”) instead of “Apply”.
  - Include a “Close” button in a dialog box with an “Apply” (or other action) button. Close dismisses the dialog box without applying changes.
    - If the user has made changes in a dialog box and clicks Close before clicking the Apply button, display a Warning alert box to apply changes, discard changes, or cancel the close operation.

Dialog command buttons

• “Close” button
  - used to dismiss informative secondary windows
  - Never use an “OK” button in a window that has a “Close” button.
  - When users click the “Close” button, dismiss the dialog box and do not make additional changes to the system.

Dialog command buttons

• “Help” button
  - used in any dialog box that needs it
    - “how do I do X?”, “what does X mean?”
  - When users click the “Help” button, open an additional window that displays help information.
    - Avoid removing or obscuring information in the window where users clicked “Help”.
  - Place “Help” at the trailing (right) edge of a group of command buttons.
Dialog command buttons

- Default button
  - represents the most likely user action
  - If a dialog box has a default button, make it the first (left-most) command button in the group.
  - Commands that cause the user to lose data should never be the default button, even if they are the most common option.

```java
frame.getRootPane().setDefaultButton(okButton);
```

Common dialog boxes

- Find dialog
  - enables users to search for a specified text string
  - often modeless, multiple-use
- Login dialog
  - enables users to identify themselves and enter a password
  - typically modal
- Progress dialog
  - provides feedback for long operations

Common dialog boxes

- Preferences dialog
  - enables users to view and modify the characteristics of an application
  - complex windows can be organized & simplified using tabbed panes

Common dialog boxes

- Color Chooser dialog
  - provides one or more content panes from which users can select colors and view them in context
  - chooser can be implemented as panel(s), not separate window (so can embed into dialogs)
Alert boxes

- Alerts convey a message or warning to users, and also provide an easy way for you to create a secondary window
- Swing has four basic types of built-in alerts
- Create alerts as JOptionPane’s
  - this will give you the built-in goodies

Alert boxes

- Basic design
  - Begin your message with a brief heading in bold. Start the body of the message on a separate line.
  - use <b>...</b> for heading, <br> for line break
  - If appropriate, provide a Help button in an alert box that opens an additional window with more information.
  - Different alignment rule for buttons!!! (Sigh.)

Alert boxes

- Info alert boxes
  - present general information to users
  - Provide a Close button to dismiss an Info alert.

Alert boxes

- Warning alert boxes
  - warn users about the possible consequences of an action and ask users for a response
  - Keep the message in a Warning alert box brief, and use terms that are familiar to users.
  - Include at least two buttons in a Warning alert box: one button to perform the action and the other to cancel the action.
  - If the most common action could result in data loss, do not provide a default.
Alert boxes

• Error alert boxes
  - report system and application errors to users
  - In the message of an Error alert box, explain what happened, the cause of the problem, and what the user can do about it. Keep the message brief and use terms that are familiar to users.
  - If possible, provide buttons or other controls to resolve the error noted in the Error alert box.
    • Label the buttons according to the action they perform.
    • If users cannot resolve the error, provide a Close button.
  - If appropriate, include an error number in the title bar of an Error alert box.
  - If appropriate, provide a Help button in an Error alert box to open a separate window that gives background information about the error.

• Question alert boxes
  - request information from users
  - sometimes, can be used for error correction
  - you can add your own components here (unlike other alert boxes)

Creating dialogs & alerts

• When creating custom modal dialogs, use ...

```java
// Creates a modal or non-modal dialog without a title and with the owner Frame.
JDialog (Frame owner, boolean modal)

// Creates a modal or non-modal dialog with the specified title and the owner Frame.
JDialog (Frame owner, String title, boolean modal)

// Creates a non-modal dialog without a title with the specified Frame as its owner.
JDialog (Frame owner)

// Creates a non-modal dialog with the specified title and with the owner frame.
JDialog (Frame owner, String title)
```

• Other constructors give non-modal dialogs...

```java
// Creates a modal or non-modal dialog without a title and with the owner Frame.
JDialog (Frame owner, boolean modal)

// Creates a modal or non-modal dialog with the specified title and the owner Frame.
JDialog (Frame owner, String title, boolean modal)

// Creates a non-modal dialog without a title with the specified Frame as its owner.
JDialog (Frame owner)

// Creates a non-modal dialog with the specified title and with the owner frame.
JDialog (Frame owner, String title)
```

Most alert boxes use JOptionPane

```java
JOptionPane.showMessageDialog (frame, "Eggs aren’t supposed to be green.");
```

```java
int n = JOptionPane.showConfirmDialog (frame, "Would you like green eggs and ham?", "An Inane Question", JOptionPane.YES_NO_OPTION);
```
Creating dialogs & alerts

• JColorChooser, standard and custom

```java
Color newColor = JColorChooser.showDialog(ColorChooserDemo2.this, 
   "Choose Background Color", 
   banner.getBackground());
```

Creating dialogs & alerts

• JFileChooser, standard and custom

```java
final JFileChooser fc = new JFileChooser();
int returnVal = fc.showOpenDialog (aComponent);
if (returnVal == JFileChooser.APPROVE_OPTION) ... // handle file 
else ... // file chooser cancelled
```

Selection components

• Selectable lists
  - 1-column arrangement

• Selectable tables
  - multi-column arrangement (2-d grid)

• Selectable trees
  - hierarchical arrangement

Lists

• Storage for a list of items
  - typically text, or textual representations of complex objects (e.g., 9:00pm)
  - could also be icons

• Selection
  - single selection
  - single interval
  - multiple interval

January
February
March
April
Lists

• General guidelines
  - Put your selectable list in a scroll pane.
    • of course, especially if can’t see all items...
      but best to do this as a general rule

    JScrollPane scrollPane = new JScrollPane(list);

  - Display a whole number of lines in the list.
  - Activate the dialog box’s default command when
    the user double-clicks a list item.
    • not provided automatically; you need to program this.

Tables

• Basic parts
  - rows & columns, headers, scrollbar
  - editable vs. non-editable cells

<table>
<thead>
<tr>
<th>First Name</th>
<th>Last Name</th>
<th>Employee ID</th>
<th>Position</th>
</tr>
</thead>
<tbody>
<tr>
<td>Jacob</td>
<td>Lehman</td>
<td>533</td>
<td>Finance</td>
</tr>
<tr>
<td>Sophia</td>
<td>Arminn</td>
<td>372</td>
<td>Kawasaki</td>
</tr>
<tr>
<td>Samuel</td>
<td>Stewart</td>
<td>452</td>
<td>Butler</td>
</tr>
<tr>
<td>Eva</td>
<td>Kidney</td>
<td>1273</td>
<td>Microtech</td>
</tr>
<tr>
<td>Mary</td>
<td>Doe</td>
<td>911</td>
<td>FinDog</td>
</tr>
<tr>
<td>Rose</td>
<td>Anderson</td>
<td>28</td>
<td>FinDog</td>
</tr>
<tr>
<td>Tina</td>
<td>Brooks</td>
<td>193</td>
<td>Microtech</td>
</tr>
</tbody>
</table>

• Cell specifics
  - Swing’s (like any system’s) is very specific about
    the look of cells in different situations...

  - unselected cell, editable
  - unselected cell, non-editable
  - selected cell, editable with keyboard focus
  - selected cell, non-editable with keyboard focus
  - any other selected cell

• Resizing columns
  - users can resize columns themselves
  - several methods are available (and whichever
    you set is used) — for example...

  - resize next
  - resize subsequent
  - resize last
Tables

- **Reordering columns**
  - a nice feature transparent to the programmer

<table>
<thead>
<tr>
<th>First Name</th>
<th>Employee ID</th>
<th>Last Name</th>
<th>Age</th>
</tr>
</thead>
<tbody>
<tr>
<td>Jakob</td>
<td>532</td>
<td>Linke</td>
<td>35</td>
</tr>
<tr>
<td>Peter</td>
<td>27</td>
<td>Winter</td>
<td>38</td>
</tr>
<tr>
<td>Sophia</td>
<td>377</td>
<td>Ammann</td>
<td>40</td>
</tr>
<tr>
<td>Samuel</td>
<td>482</td>
<td>Stewar</td>
<td>42</td>
</tr>
<tr>
<td>Eva</td>
<td>1273</td>
<td>Keenan</td>
<td>39</td>
</tr>
<tr>
<td>Mary</td>
<td>911</td>
<td>Doe</td>
<td>41</td>
</tr>
<tr>
<td>Rosie</td>
<td>28</td>
<td>Anwarahh</td>
<td>40</td>
</tr>
<tr>
<td>Mira</td>
<td>192</td>
<td>Brooks</td>
<td>42</td>
</tr>
</tbody>
</table>

Trees

- **Basic structure:**

<table>
<thead>
<tr>
<th>Top-level container</th>
<th>Left</th>
<th>Right</th>
</tr>
</thead>
<tbody>
<tr>
<td>Collapsed container</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Expanded container</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Leaf</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Tables

- **General guidelines**
  - **Put your selectable table in a scroll pane.**
    - the code is the same as for lists
  - When resizing a table vertically, make sure that it always displays a whole number of rows.
  - When sorting table rows...
    - Make the column header text bold to indicate the table column that currently determines the sort order.
      (This is not automatic!)
    - If your application has a menu bar, provide row sorting as a set of menu items as well (e.g., “Sort by Sender”).

Trees

- **General guidelines**
  - **Put your selectable tree in a scroll pane.**
    - sound familiar? again, same code
  - For most trees, display the second level of the hierarchy as your highest level.
    - you can set tree.rootVisible to false
  - Display turners for all containers in the tree component, including the containers at the highest level.
    - you can set tree.showsRootHandles to true
Labels

- **Label basics**
  - can be read-only text, graphics, or both
  - serve two functions in an application
    - to identify components and enable navigation to components that don’t have their own text strings
    - to communicate status and other information

- **Labels that identify controls**
  - Keep label text brief, and use terminology that is familiar to users.
  - Use headline capitalization in the label text and place a colon at the end of the text.

- **Labels that communicate status**
  - Use sentence capitalization in the text of a label that communicates status. (Use a period only when text is a complete sentence.)

- **Available vs. unavailable labels**
  - Make a label unavailable when the component it describes is unavailable.

Labels

- **Menu bars**
  - Use menu titles that make it easy for users to determine which menu contains the items of interest to them.
    - e.g., “Format” typically contains commands that enable users to change the formatting of docs
  - Use a single word for each menu title.
  - Be sure to include mnemonics for every menu title in your menu bar.
  - Do not display menu bars in secondary windows.
  - For applets that run in the user’s current browser window, do not display your own menu bar in the applet.

- **Menu operation**
  - (1) To post a menu (that is, to display it and have it stay open until the next click), users click the menu title. Users can then move the pointer over other menu titles to view other menus.
  - (2) To pull down a menu, users press the mouse button when the pointer is over the menu title. The menu title is highlighted, and the menu drops down. When users choose a command and release the mouse button, the menu closes.

  Of course, Swing already handles this for you
• Submenus
  - sometimes, can help shorten or organize menus

  - Because many people, especially novice users, children, and older people find submenus difficult to use, minimize the use of submenus.
    - When possible, avoid using submenus at all.
    - If you want to present a large or complex set of choices, display them in a dialog box.

• Menu items
  - Make your menu items brief — typically verb phrases (e.g., “Align Left”) or nouns (e.g., “Font”).
  - Never give a menu item the same name as its menu title.
    - e.g., an “Edit” menu should not contain an “Edit” menu item
  - Use headline capitalization for menu items.
  - Include mnemonics for all menu items.

• Menu items (cont.)
  - Offer keyboard shortcuts for frequently used menu items.
  - Use the same keyboard shortcut if a menu item appears in multiple menus.
    - e.g., if a Cut item appears in a contextual menu as well as in a drop-down Edit menu, use Ctrl-X for both
  - Use the same mnemonic if a menu item appears in multiple menus.
    - e.g., if a Copy item appears in a contextual menu as well as in a drop-down Edit menu, use the ‘C’ mnemonic for both

• Menu items (cont.)
  - If a menu item does not fully specify a command and users need a dialog box to finish the specification, use an ellipsis (“…”) after the menu item.
    - e.g., after choosing “Save As…”, users are presented with a file chooser to specify a file name and location
  - Do not use an ellipsis mark simply to indicate that a secondary or utility window will appear.
    - e.g., choosing “Preferences” displays a dialog box — however, because that display is the entire effect of the command, “Preferences” is not followed by an ellipsis
• Separators
  - Use separators to group similar menu items in a way that helps users find items and better understand their range of choices.
  - Users can never choose a separator.
  - While separators serve important functions on menus, avoid using them elsewhere in your application.
    • Instead, use blank space or an occasional titled border to delineate areas in dialog boxes or other components.

• Menu item graphics
  - Provide menu item graphics when there are corresponding toolbar button graphics in your application.
    • The graphics help users associate the toolbar button with the corresponding menu command.

• Checkbox menu items
  - Use checkbox menu items with restraint.
    • If users must set >2 related attributes, place the checkboxes in a dialog box (or provide a utility window or toolbar buttons for the attributes).
  - Use checkbox menu items instead of toggle menu items (e.g., “Italics On” and “Italics Off”) to indicate choices you can turn on or off.
    • Toggle menu items confuse users — it is unclear if the commands are telling users the current state of the selected object, or the state they can change the object to by choosing the menu item.

• Radio button menu items
  - typically used for multiple values, except simple toggle on/off
    • To indicate that the radio button items are part of a set, group them and use separators to separate them from other menu items.
Common menus

- **File Menu**
  - Place commands that apply to the document or the main object (or the application as a whole) in the File menu.
  - If your application manipulates “non-file” objects, give the File menu a different name.
    - e.g., “Project”, “Mailbox”
  - When the Close item dismisses the active window, close any dependent windows at the same time.
  - Provide an Exit item, which closes all associated windows and terminates the application.
    - Be sure to use Exit, not Quit! (by convention)

Common menus

- **Edit Menu**
  - Place commands that modify the contents of documents or other data in the Edit menu.

Common menus

- **Format Menu**
  - Place commands that change the format of objects, usually text, in the Format menu.

Common menus

- **View Menu**
  - Place commands that change the view of the data in the View menu.
  - Ensure that commands in the View menu alter only the view/presentation of the underlying data without changing the data directly.
Common menus

• Help Menu
  - Place access to online information about the features of the application in the Help menu.
  - Include “About <app-name>” as the last item and place a separator before the item.
    • “About <app-name>” should display a window with the product name, version number, company logo, product logo, legal notices, and names of contributors.

• Menu Bar
  - If your application needs the commonly used menus, place the menu titles in this order:
    • File, Edit, Format, View, Help
  - If needed, insert other menus between the View and Help menus (and sometimes between Edit and View, as appropriate).

Contextual (or pop-up) menus

• Tips
  - Ensure that all features presented in contextual menus are also available in more visible and accessible places, such as drop-down menus.
  - Users might not know contextual menus are available, especially if you do not use contextual menus consistently throughout your application.
  - Display keyboard shortcuts and mnemonics in contextual menus that are consistent with their usage in any corresponding drop-down menus.

• Tips (cont.)
  - If no object is selected when a contextual menu is displayed, select the object under the pointer and display the contextual menu appropriate to that object.
    • e.g., if the object under the pointer is text, display the contextual menu with editing commands.
  - If the pointer is over an existing selection at the time the user opens the contextual menu, display the menu that is associated with that selection.
  - If the user opens a contextual menu when the pointer is over an area that cannot be selected (e.g., container background), remove any existing selection and display the contextual menu for the container.
Toolbars

• General tips
  - Include commonly used menu items as buttons (or other components) in your toolbar.
  - Even if your window has a toolbar, make all toolbar commands accessible from menus.
  - Be sure to provide tool tips for all toolbar buttons.
  - Consider providing text on toolbar buttons as a user option.
    - This makes the meaning of the button clear to new users, and enables low-vision users to use large fonts.

• Toolbar buttons
  - Use button graphics that are either 16 x 16 or 24 x 24 pixels (but not both in the same toolbar), depending on the space available in your application.
  - Provide optional text-only toolbar buttons to enable viewing by low-vision users.
  - If you use text on the toolbar buttons, provide a user setting to display only the graphics.
    - Using graphics only, you can conserve space and display more commands and settings in the toolbar.
  - Sometimes, to create functional groupings of toolbar buttons, provide a separate toolbar for each.
    - however, often not needed and/or not practical

• Toolbar tips
  - Ensure that the keyboard shortcuts for toolbar buttons match the keyboard shortcuts for the corresponding menu items.
  - Attach tool tips to all toolbar components that do not include text identifiers.

  - interesting: tips generally use an onset of 250 ms and stay on for 15 seconds
  - If your application does not have menus, attach tool tips to the toolbar buttons in order to display keyboard shortcuts.
Code structure

• What's the best way to code up a window?
  - like design, there's no best way really
  - but we'll give you a good way
• Starting point: remember the model-view-controller architecture?
  - Swing's components all have an implicit/explicit model separate from the view & controller
  - let's say we want a window to represent some chunk of information
  - we want to separate the model of this information from its view & controller

Code structure

Given our chunk of information...

- model = information
- view & controller = component
- let's have a Java class for each part:

Model

class MyInfo
  - includes all model information

View+Controller

class MyInfoWindow
  extends JDialog
  includes all window's components

Code structure

Example: Calendar "Day Window"

class Day
  - Date date
  - int dayOfWeek
  - Event events[]

class DayWindow
  extends JFrame
  - JTextField
  - JComboBox
  - JTable+TableModel

For window classes, we'll follow this style...

public class MyInfoWindow extends JFrame {
  << instance variables for our window frame >>
  class <<MyListener>> implements <<ListenerInterface>> { … }
  class <<MyListener>> implements <<ListenerInterface>> { … }
  …
  MyInfoWindow (MyInfo) {
    super (<<name>>);
    << create all components and add to frame >>
    << add frame listeners >>
  }
  // for primary windows, include main();...
  public static void main (String s[]) {
    JFrame frame = new MyInfoWindow (MyInfo);
    << finish and display frame >>
  }
}
Handling multiple windows

- When do we use multiple windows?
- One idea that’s been proposed makes sense: Treat windows like rooms...
  - each “room” should have a logical purpose
    - e.g., kitchen, bedroom, basement, etc.
    - e.g., document window, find window, etc.
  - keep things in one room as much as possible
  - but don’t cram rooms to the point of becoming cluttered and incoherent

Handling multiple windows

• One way to think about our “rooms”
  - recall our discussion of handling single windows
  - can do the same for multiple windows

Model

class DayWindow
extends JFrame

class Day
Model+View+Controller

class EventWindow
extends JDialog

class Event
(etc.)

Handling multiple windows

• Passing information, modal dialog -> frame
  - in DayWindow, we’ll create the EventWindow
  - when the user opens an event...

button.addActionListener (new ActionListener () {
  public void actionPerformed (ActionEvent e) {
    event = new Event (); // or edit selected event, etc.
    EventWindow dialog = new EventWindow (frame, event);
    dialog.pack ();
    dialog.setVisible (true);
    // dialog interaction happens here while dialog is visible;
    // when user hits “OK”, following code continues
    if (dialog.event != null) // that is, event actually created
      day.addEvent (event);
  }
});

Handling multiple windows

• Passing information, modal dialog -> frame
  - in EventWindow, we’ll manage Event info
  - when the user hits “OK”...

okButton.addActionListener (new ActionListener () {
  public void actionPerformed (ActionEvent e) {
    // if not created already, create new event
    event = new Event ();
    // transfer info from components to data structure
    event.title = titleTextField.getText ();
    // etc.
    // now make window invisible, thus passing control back to parent
    self.setVisible (false);
  }
});