

CS 349

Widget Toolkits

Byron Weber Becker
Winter 2012



CS 349 Winter 2012 1

Overview

- Widget toolkits overview
- Design goals
- Event-driven program
- Common widget types
- Lightweight vs heavyweight widgets



CS 349 Winter 2012

Widget toolkits

- Also called widget libraries or GUI toolkits
- Software bundled with a window manager, operating system, or application platform
- Defines a set of GUI components that can be used by programmers
- Programmers access these GUI components via an application programming interface (API)
- Users use these components to control application



CS 349 Winter 2012 3

Event-driven programming

- Widget toolkits typically support event-driven programming model
- Reactive systems
 - User action -> program response
 - Most of the time the program sits around doing nothing
- Widget toolkit supports a mechanism for mapping user action on widget to appropriate application code to handle that action

Design Goals

- GUI toolkit should be:
 - Complete
 - Designer has everything they need
 - Consistent
 - Behaviour is consistent across components
 - Customizable
 - Developer can reasonably extend functionality to meet particular needs of application
- Goals can be (partially) achieved via using a common GUI toolkit.
- Meeting these requirements encourages *reuse*.

Completeness

- Component types
 - Canvases
 - For drawing
 - Informational
 - Non-interactive presentation (e.g. label, tooltip, progress bar, ...)
 - Selectors/Changers
 - For choosing among items (e.g. list, menu, checkbox, button, ...)
 - For modifying a value (e.g. sliders, scrollbars, ...)
 - Containers
 - For containing/grouping other components (e.g. frames, internal frames, panels, ...)
 - Text input
 - For text entry (e.g. single line input, text area, rich text editor, ...)

Completeness

- The “Macintosh 7” (Dix, Finlay, Abowd, Beale, 1998)
 - Button
 - Slider
 - Pulldown menu
 - Check box
 - Radio button
 - Text entry / edit fields
 - File open / save

To see the rich spectrum of controls available in Java Swing, check out `SwingSet2` in `demo/jfc` folder of jdk installation (`/usr/jdk.../demo/jfc/SwingSet2`). Or download from java.sun.com/products/jfc/jws/SwingSet2.jnlp



Consistency

- Facilitate learning by:
 - Employing common visual and/or auditory presentation
 - Sharing look and feel
 - What is meant by “feel”?



Consistency

- Look and Feel
 - Look: The visual appearance of objects on the display
 - Feel: The behaviour of the program in response to user actions
- Widget toolkits support look and feel
 - Would you recognize a Java application?
 - Given only display (excluding window frames)



Customizable

- Component should provide suitable hooks for extension and customization for particular applications
- Common strategies
 - Flags/variables that can be directly set
 - Examples:
 - Component colour
 - Component text
 - Component size
 - Factor out behaviour that can change
 - Examples:
 - Responding to an action: ActionListener
 - Swing's UIManager look and feel
 - JTable as a rich example...



Customizable: JTable

- JTable factors out much of its functionality
 - The actual data (TableModel) (part of MVC pattern)
 - Selection of items (ListSelectionModel)
 - Rendering of cells (TableCellRenderer)
 - Editing of cells (TableCellEditor)
- Developer has lots of flexibility in the data that can be represented, how it can be selected, its method of presentation, and its method of editing



Implementation Choices

- Heavyweight Widgets
 - Provided by the OS
 - Examples: nested XWindows, Java's AWT
- Lightweight Widgets
 - OS provides a canvas; toolkit draws widgets on it and interprets events delivered to the canvas
 - Example: Java Swing



Heavyweight Widgets

- Operating system typically includes built-in widget toolkit
 - OS widgets are called heavyweight widgets
- Benefits
 - Events generated by user action is passed by OS directly to components
 - Preserves OS look and feel
- Disadvantages
 - EITHER OS specific programming, OR Widgets are a greatest common subset of those available on different platforms



Lightweight widget toolkits

- Application built with widget toolkit gets one heavyweight component
 - An application window
- OS delivers all user generated events to that window
- Widget toolkit draws its own widgets and is responsible for mapping events to their corresponding widgets



Java

- Java has two primary widget toolkits
 - AWT
 - Swing
- AWT is a heavyweight toolkit
 - Components in AWT are OS components, mapped onto the Java language
 - Objects like Button, Canvas, Choice, Frame, Label, List, MenuBar, Panel, PopupMenu, Scrollbar, TextArea, Window.
 - OS is aware of these components and can map events onto them



AWT Toolkit

- AWT is minimal toolkit
 - No Spinner, no combo box, no progressbar
- Idea is to try to identify components that are supported on most platforms and map the Java language to these
- Means that programmers need to re-create unsupported widgets, or find a library that creates these unsupported widgets
- AWT itself is “least-common denominator”



Swing Toolkit

- Originally a separate download
- Toolkit is a collection of widgets implemented in Java
- Basic mapping is:
 - Java gets a Canvas object inside a window from the OS
 - Canvas object also has graphics object
 - Java code is used with the graphics object to draw swing components onto Canvas.
- Mixing Swing and AWT in the same application gave problems with visibility
 - Two different rendering pipelines with the OS pipeline being unaware of the Swing pipeline.
 - Result: Swing was all-or-nothing



Eclipse's SWT

- Standard Widget Toolkit
- OS components written in Java but rendered using native OS routines
 - SWT and Swing are different tools that were built with different goals in mind. The purpose of SWT is to provide a common API for accessing native widgets across a spectrum of platforms. The primary design goals are high performance, native look and feel, and deep platform integration. Swing, on the other hand, is designed to allow for a highly customizable look and feel that is common across all platforms



SWT Pros & Cons

- Swing
- Components written in Java and rendered in Java
 - Consistency across platforms
 - More extensive
- Platform feel issues
 - On Mac OS X, default buttons actually have an animated pulsing glow to focus the user's attention on them
- SWT
- Java wrapper for native libraries, with special purpose components added in Java
 - Deeper integration with OS to preserve look and feel
 - Simpler without extraneous functionality
- Portability issues
 - Good integration with Win32, but problems on other platforms
 - Very difficult to port

Swing Demo

- Creating a new project
- “Hello, World!”
- JFrame; exit-on-close
- Add a button
- Set layout manager
- Make the button do something
- Add a table