Eclipse technologies

The technologies behind Eclipse

Marco Savard, neosapiens

Technologies

- Data: JDT, CDT, EMF, UML2, DTP, SDO..
- Process: XSD, TPTP, JET, ..
- GUI: SWT, JFace, GEF, GMF, ..

SWT

- SWT is easier to use than Swing
- SWT gives UIs that look more native than Swing

SWT

- History
- Platforms
- Swing Comparison
- Class Hierarchy
- Layout
- Painting

SWT History

- IBM VisualAge/Smalltalk
- Lessons Learned from Smalltalk
  - In Smalltalk, everything was emulated
  - Common interactions, such as scrolling, typing, felt differently
  - Native OS features, such as DnD, language support, partly implemented
  - As new release of OS came out, their appearance changed, leaving Smalltalk apps looking dated.
- CommonWidgets API, ancestor of SWT

SWT Platforms

- Within Eclipse:
  - Runs everywhere Eclipse runs
- Outside Eclipse (standalone apps):
  - Windows 2000, XP
  - Linux (RedHat SuSE)
  - Solaris (X/Motif)
  - Mac (OS X 10.2)

Heavy and Lightweight components

- Heavyweight components: AWT
  - Lowest-Common Denominator
  - Few components!
- Lightweight components: Swing
  - Always draw components
  - Slow!
- Middleweight components: SWT
  - Use native GUI libraries through JNI
  - The pragmatic approach: use if available, draw otherwise

Advantages over Swing

- Faster (native widgets already optimized)
- Memory efficient (widgets disposal)
- Real Windows Look and Feel
- More widgets available
- More up-to-date
- Simpler API: less classes (ex: buttons)
- Untyped Listeners
- Composition versus Aggregation
- Eclipse Public Licensed (Swing is proprietary)

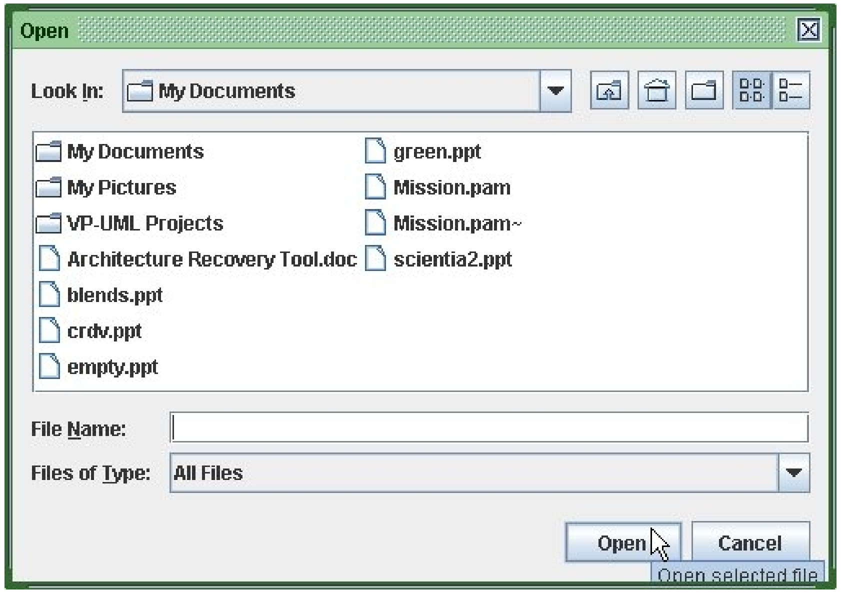
High Performance

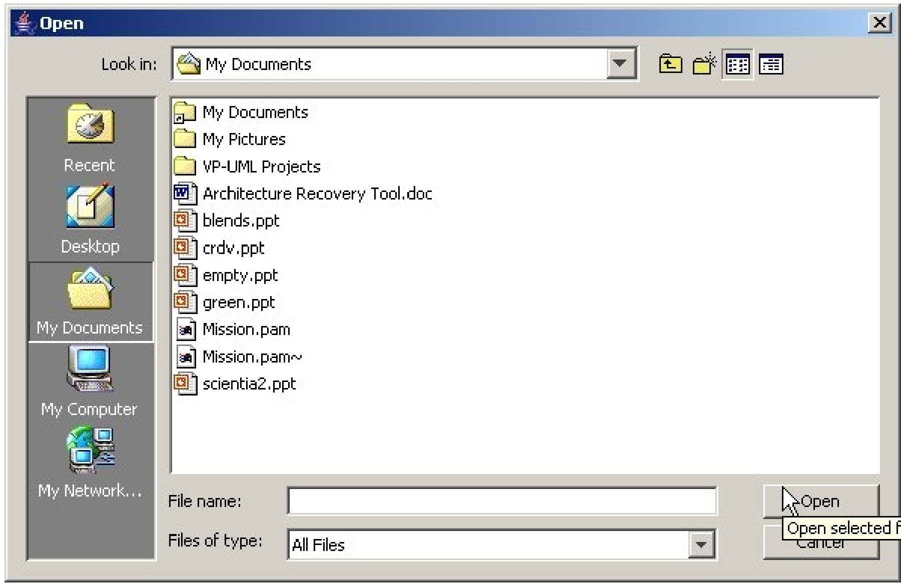
- SWT has been designed to be a "high performance" GUI toolkit; faster, more responsive and lighter on system resource usage than Swing.
- Benchmarks: 3-4 more internal calls in Swing; SWT calls are more efficient.

Native look and feel

- SWT, due to it's use of native widgets features a native "look and feel"
- The same cannot be said of Swing, which must be updated to mirror Operating System GUI changes (such as theme or other look and feel updates).

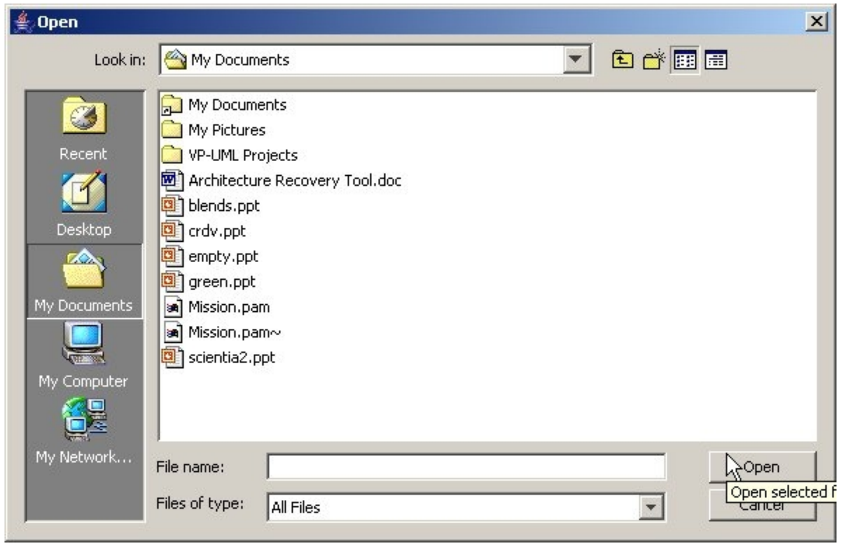
Swing Look-And-Feels

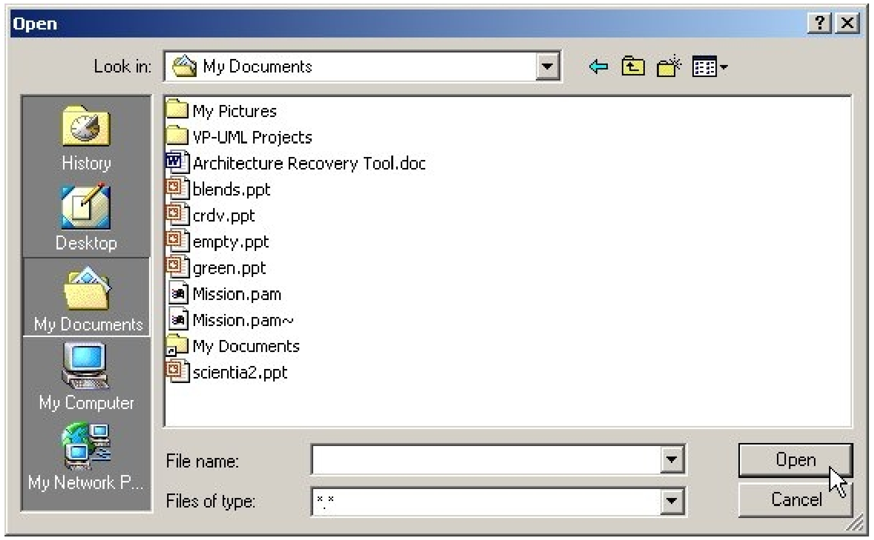




- Swing w/ Metal L&F
- Swing w/ Windows XP L&F

Swing versus SWT: find the differences

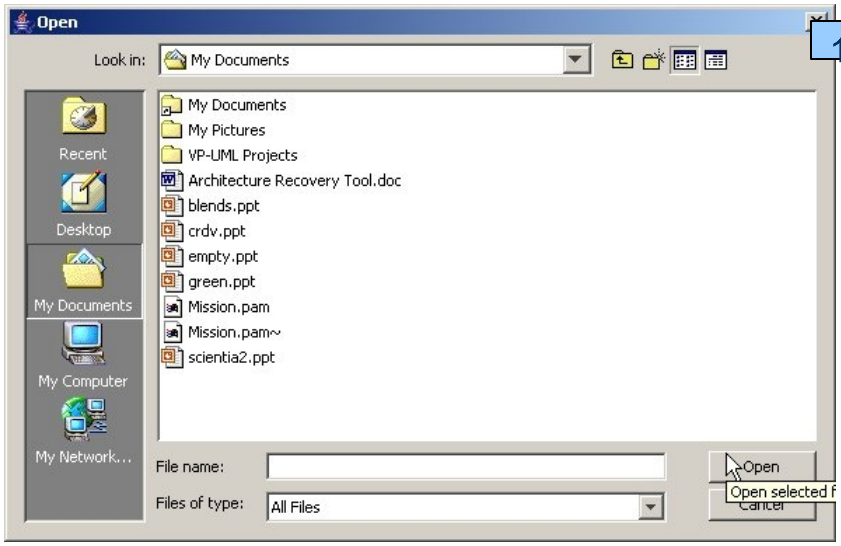


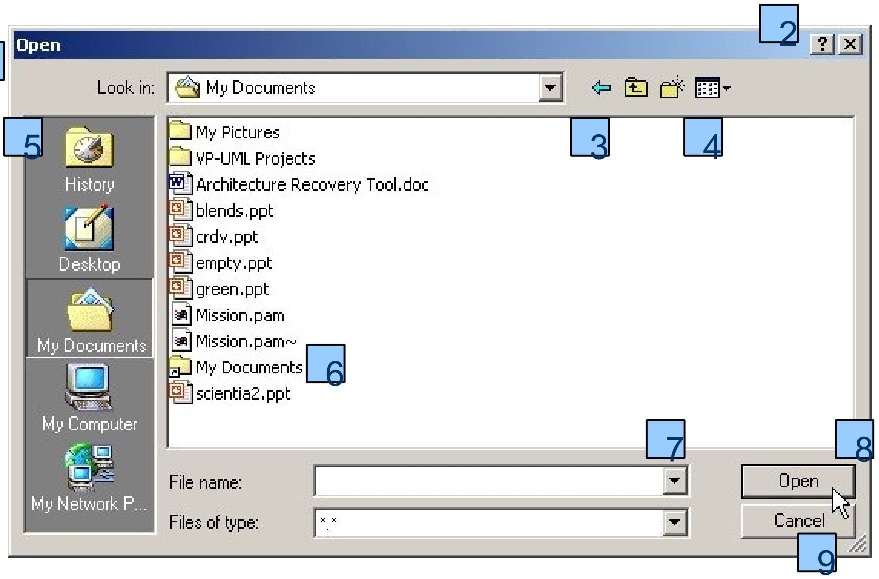


- Swing w/ Windows XP L&F
- Find the 9 differences!

- SWT by default

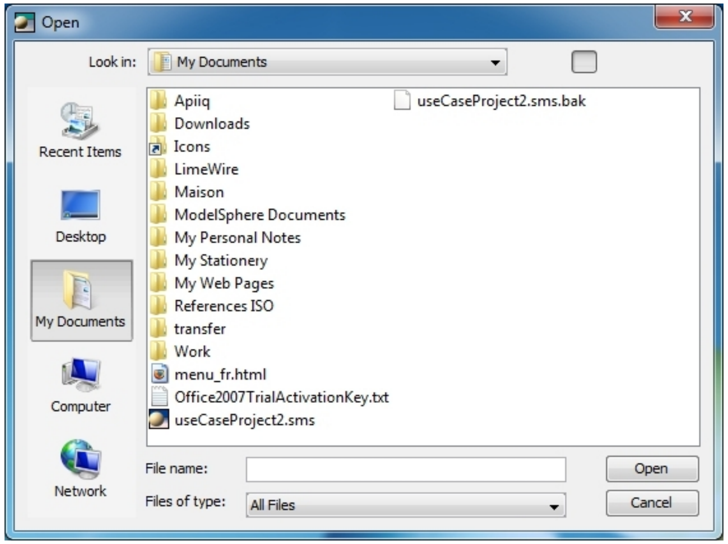
Swing versus SWT: find the differences





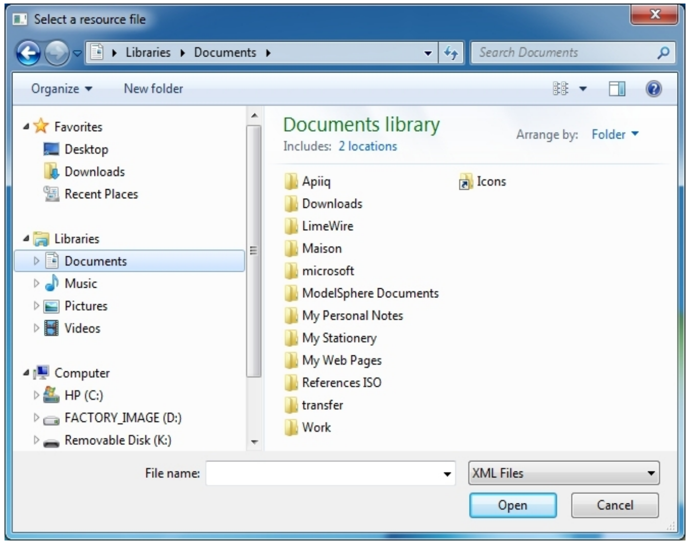
- Swing w/ Windows L&F
- SWT by default

Swing versus SWT: Windows 7


Open

Look in: My Documents

Files of type: All Files


Select a resource file

Libraries ▸ Documents ▸

Documents library
Includes: 2 locations

Arrange by: Folder

File name:

XML Files

- Swing w/ Windows 7 L&F
- Breadcrumbs, Search fields, views, work well on SWT

Clean Design

- SWT designed by Erich Gamma, of the Design Patterns Gang.
- Based on composite design pattern.
  - Create the composite first (Shell, Group)
  - The component’s constructor requires the composite as parameter (Button, TextArea)

Drawbacks compared to Swing

- Resource disposal: programming (a little bit) more difficult
- Native-widget limitations
  - tables always have scrollbars, Windows limitations. But Windows users have never seen a table without scrollbard.
- Less platforms supported
  - But the four platforms supported represent 99% of the market!
- Naming conv.: JList, JText, better than SWT

Class hierarchy

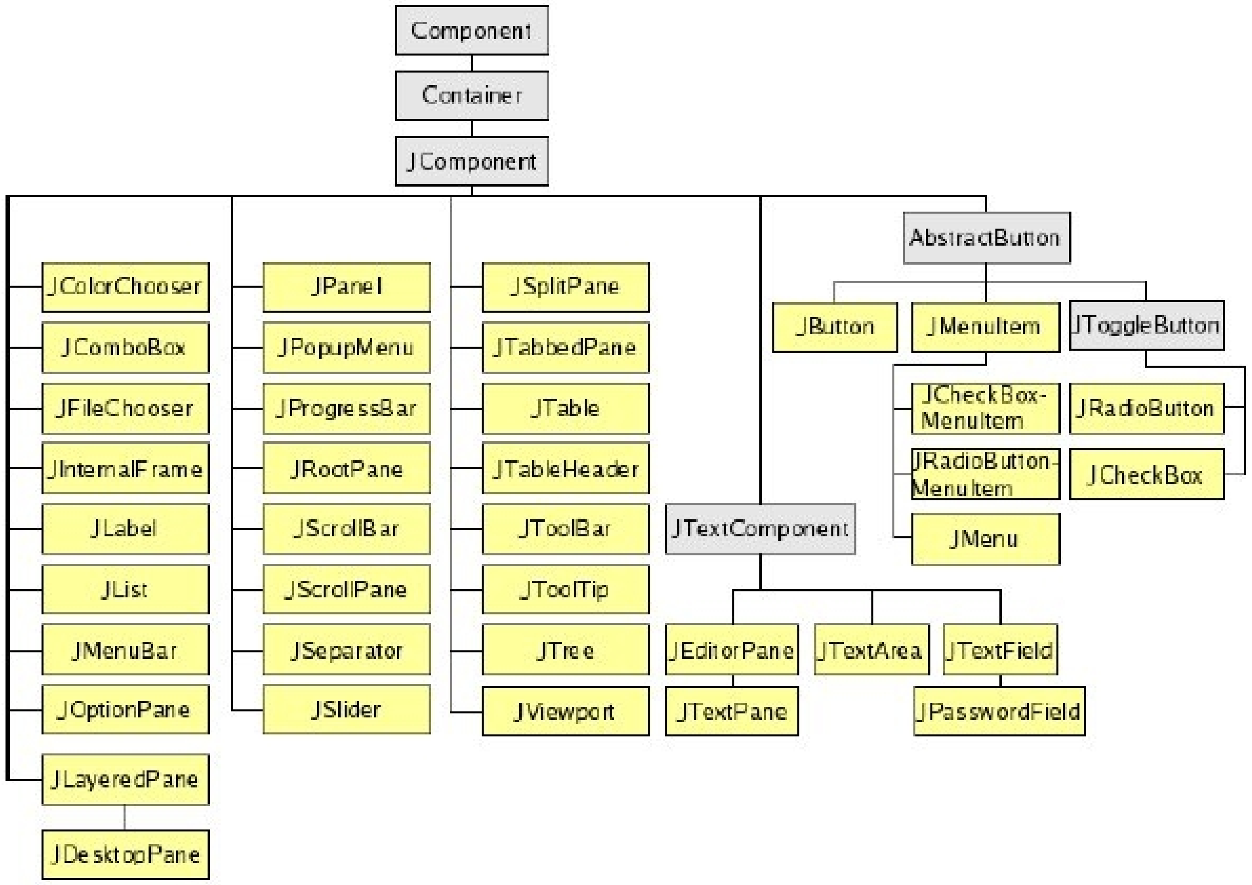
- Object
  - Widget
    - Control (heavyweight)
      - Button
      - Label
      - Composite
        - Canvas
        - Group
        - Tree
    - Item (lightweight)

- Object
  - Component (AWT)
    - Container (AWT)
      - JComponent (Swing)
        - JButton
        - JLabel
        - JPanel
        - JTree

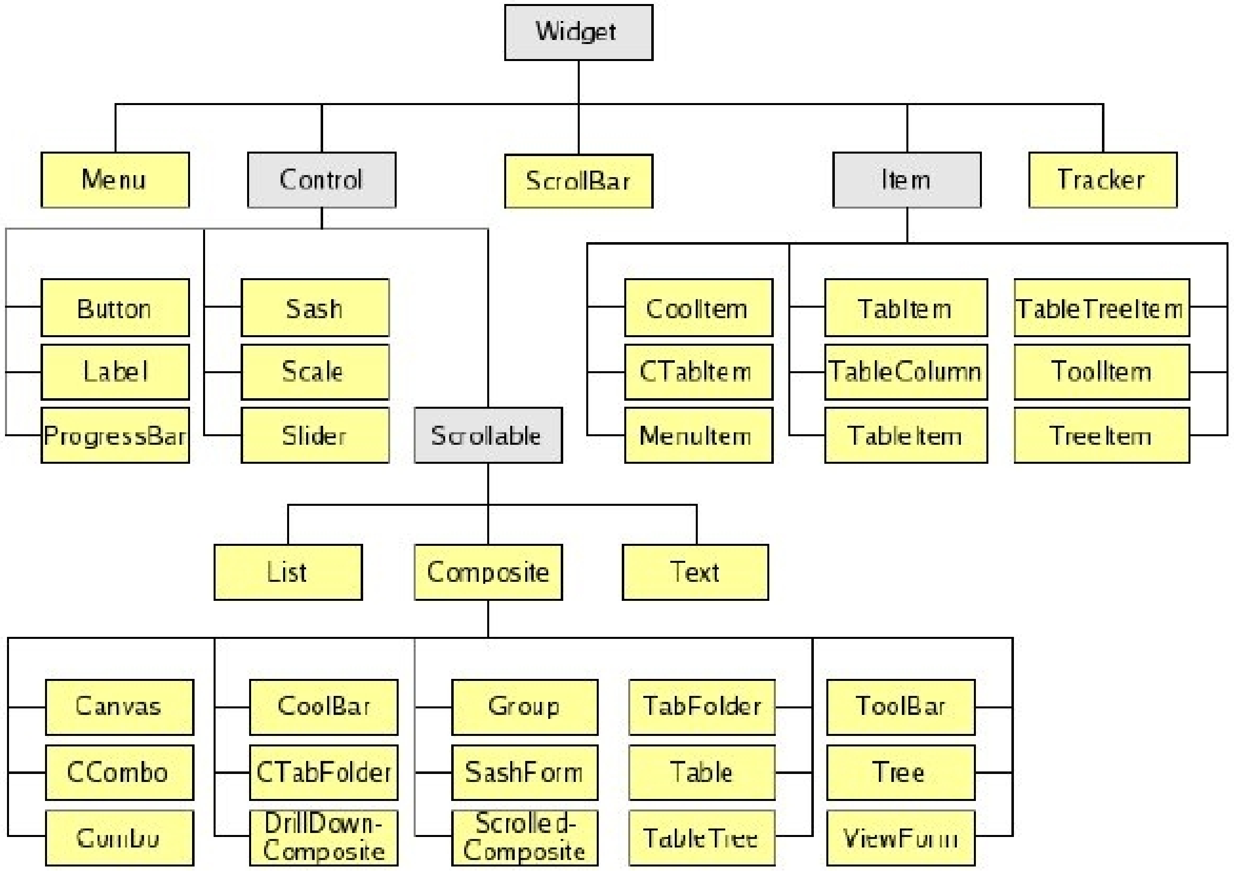
Class mapping
note: for one SWT class, often several Swing classes

SWT	Swing
Button	JButton, JCheckBox,..
ColorDialog	JColorChooser
Group	JPanel
Label	JLabel
MessageBox	JOptionPane
SashForm	JSplitPane
ScrolledComposite	JViewport
Text	JTextArea, JTextField

Swing Class Diagram



SWT Class Diagram

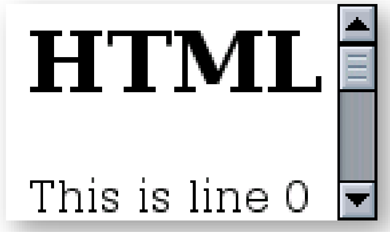


Class mapping (layout)

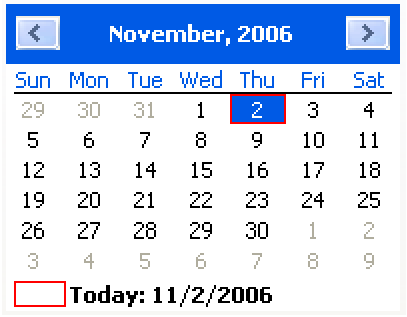
SWT	Swing
FillLayout	BorderLayout
RowLayout	FlowLayout
GridLayout	GridBagLayout
FormLayout	N/A
N/A	CardLayout

Classes w/o Swing counterparts (1)

Browser



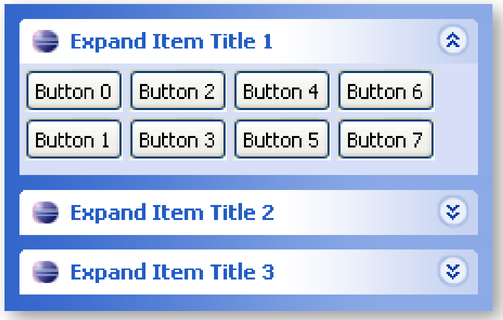
DateChooser



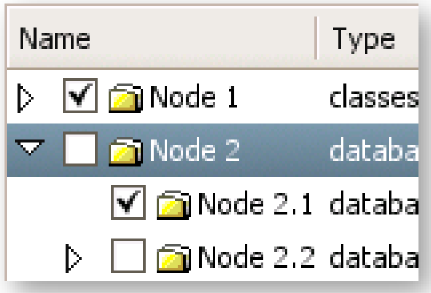
Sun	Mon	Tue	Wed	Thu	Fri	Sat
29	30	31	1	2	3	4
5	6	7	8	9	10	11
12	13	14	15	16	17	18
19	20	21	22	23	24	25
26	27	28	29	30	1	2
3	4	5	6	7	8	9

Today: 11/2/2006

Expand Bar



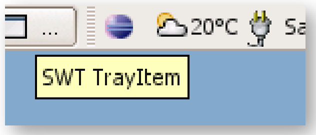
TreeTable



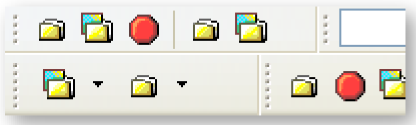
Name	Type
Node 1	classes
Node 2	databa
Node 2.1	databa
Node 2.2	databa

Classes w/o Swing counterparts (2)

Tray



CoolBar



Also: DirectoryDialog, FontDialog, Wizard (Jface) and many others..

Ref: http://www.eclipse.org/swt/widgets/

The facade class : SWT

- Defined constants
- More used then SwingConstants
- Example:
  - Button b = new Button(parent, SWT.PUSH);
  - b.setText(“OK”);
  - //PUSH could be changed by RADIO, TOGGLE,
- Remarks:
  - One class per concept
  - composite required for construction
  - No convenience constructor

Rules for disposing widgets

- 1-If you create it, you dispose it.
  - Tricks: add dispose() right after the constructor()
  - Keep dispose() in the same method where widget was created
- 2-Disposing a parent disposes the children.
  - In practice, dispose() is rare.

Painting

- Widgets (except Canvas) responsible for drawing themselves (Deferred Update Strategy).
- Painting called by the OS when a region is damaged (needs to be redraw)
  - SWT: PaintListener()
  - AWT: overrides paint() //callback method, don't call!
  - Swing: paintComponent() //ditto
- Redraw(): Tell widgets are damaged
  - Swing: repaint()
- Update(): force drawing (powerful, costy)
  - Swing: paintImmediately() /= update(), callback method

The Event Loop

- SWT: apartment threading strategy: calling a SWT object outside the UI thread throws a SWTException;
  - readAndDispatch(): read the next event
  - sleep(): let CPU time to other threads if no events
  - wake(): event loop wakes
- Long operations in dedicated thread
- Same model as Swing

Typed and Untyped Listeners

- Typed Listeners (like Swing):
  - widget.addMouseListener(new MouseAdapter() {

    };
- Untyped Listeners:
  - widget.addListener(SWT.Mouse, new Listener() {

    };
- Untyped listeners are generic, minimizes the number of listeners on the same widgets.

Composition Pattern (1/2)

- In SWT, components cannot be created w/o composite (composition)
- In Swing, components and composite are created separately, and then associated (aggregated).
- In SWT, no floating widgets, no shared widgets.
- In SWT, creation of objects always in the same order (helps consistency among developers) and no unnecessary .add() method.

Composition Pattern (2/2)

- The composite creates its components
- SWT:
  - Group group = new Group(parent);
  - Button b = new Button(group, SWT.PUSH);
- Swing:
  - JButton b = new JButton(“OK”); //create child 1st?
  - JPanel panel1 = new JPanel();
  - panel1.add(b);
  - panel2.add(b); // what happens ???

Quote from Bruce Eckel:

- "The proof is in the pudding. You rarely see an AWT application, even most Swing apps are ugly and OS strangers. You can get close but never close enough. For example when MS added theme support in Windows XP, SWT got those for free. There are more and more SWT built applications appearing. In general, why struggle to emulate pixel by pixel what Microsoft, Apple, and all the Linux developers are doing for you? Don't reinvent, use.”

Conclusion

- SWT has learned from the Swing’s mistakes
- SWT is simpler, faster, nicer, more efficient than Swing
- Middleweight widgets: the best of two worlds
- Will eventually replace Swing