heuristic.evaluation.of

Java's Swing Toolkit

Jack.Li
└ Advanced.User.Interface.Software
  └ September.7.2004

I. Rating Scale

This study augments the UAR ratings to allow positive aspects to be rated as well. The first five are the original ratings; the next four are positive mirrors of the first four:

-4:	usability catastrophe—imperative to fix before
-3:	major usability problem–important to fix
-2:	minor usability problem–low priority
-1:	cosmetic problem only–need not be fixed
0:	not a real usability problem or merit
+1:	cosmetic merit
+2:	minor usability merit
+3:	major usability merit
+4:	usability excellence

II. Metrics

This evaluation report references the Nielson heuristics as follows (number in parentheses is the number of cited positive/negative heuristics in that category):

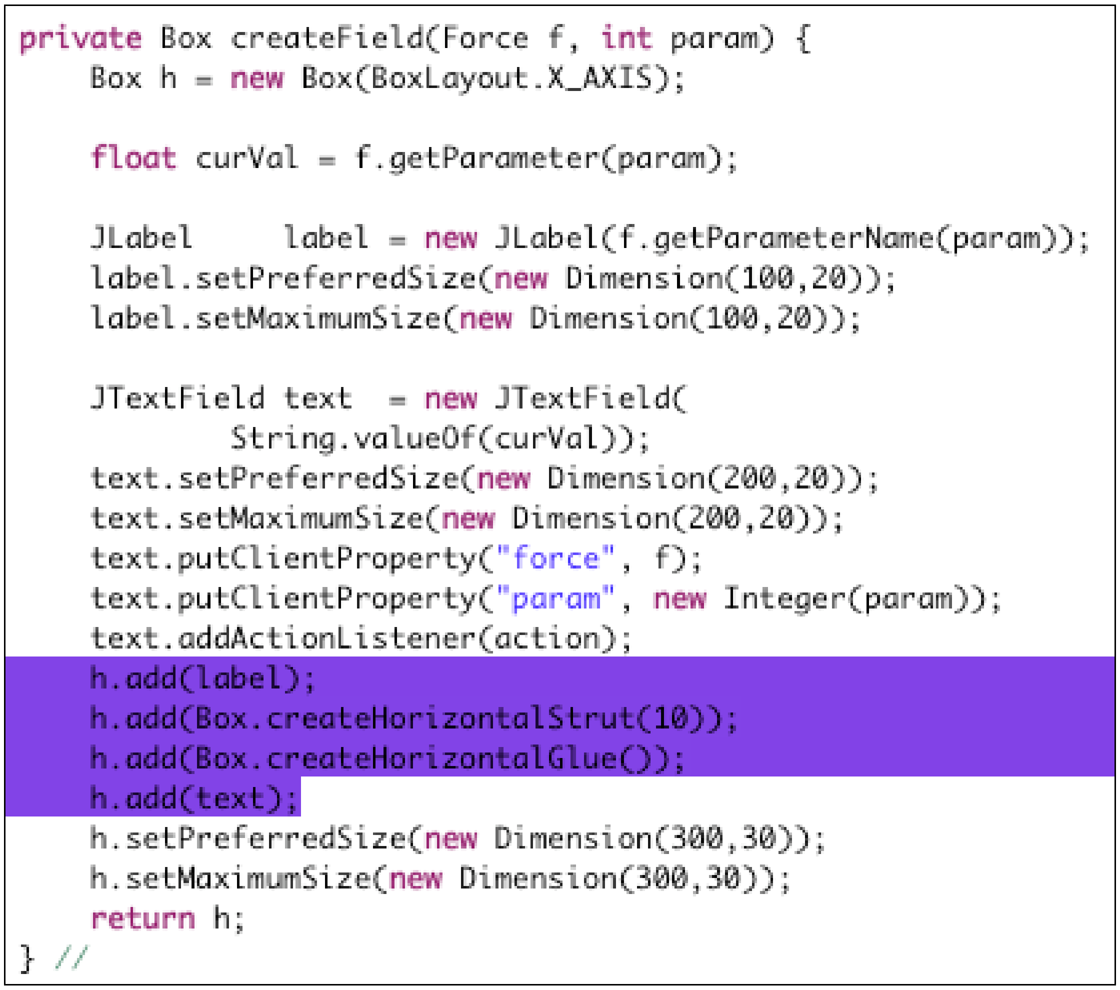
- **Visibility** of system status (1 bad)
- **Match** of system & real world (1 good)
- **User control** and freedom (1 bad)
- **Consistency** and standards (1 bad)
- **Error prevention** (3 bad)
- Recognition_rather than recall (not used)
- **Flexibility** and efficiency of use (1 bad)
- **Aesthetic** and minimalist design (1 good)
- **Error recovery** (1 good)
- **Documentation** and help (1 good)

III. Heuristic Evaluation of Swing

A. Explicit interactor tree add (rating: **-2** for Visibility)

Summary: In order for the created GUI to appear, all parts of the interactor tree must be well constructed and attached to the main content pane. Expert users learn to look for the missing branch in their code, but beginners get baffled when nothing appears on screen. These errors appear frequently because the bulk GUI creation involves customizing interactors to have the right attributes such as color, size, and semantic model (*see Exhibit i*). While adding to the right parent pane deals layout, the layout manager is often a separate entity specified in one of the ancestor panes that isn't directly related to the interactor in question.

Fix: The creation of an interactor should spawn its appearance at runtime; those not added to a specific container in the tree could appear in a window with other such orphaned interactors. This way, the programming user knows the error did not come from some property that wasn't set correctly or that was in conflict with something else.



```
private Box createField(Force f, int param) {
    Box h = new Box(BoxLayout.X_AXIS);

    float curVal = f.getParameter(param);

    JLabel     label = new JLabel(f.getParameterName(param));
    label.setPreferredSize(new Dimension(100,20));
    label.setMaximumSize(new Dimension(100,20));

    JTextField text   = new JTextField(
            String.valueOf(curVal));
    text.setPreferredSize(new Dimension(200,20));
    text.setMaximumSize(new Dimension(200,20));
    text.putClientProperty("force", f);
    text.putClientProperty("param", new Integer(param));
    text.addActionListener(action);
    h.add(label);
    h.add(Box.createHorizontalStrut(10));
    h.add(Box.createHorizontalGlue());
    h.add(text);
    h.setPreferredSize(new Dimension(300,30));
    h.setMaximumSize(new Dimension(300,30));
    return h;
} //
```

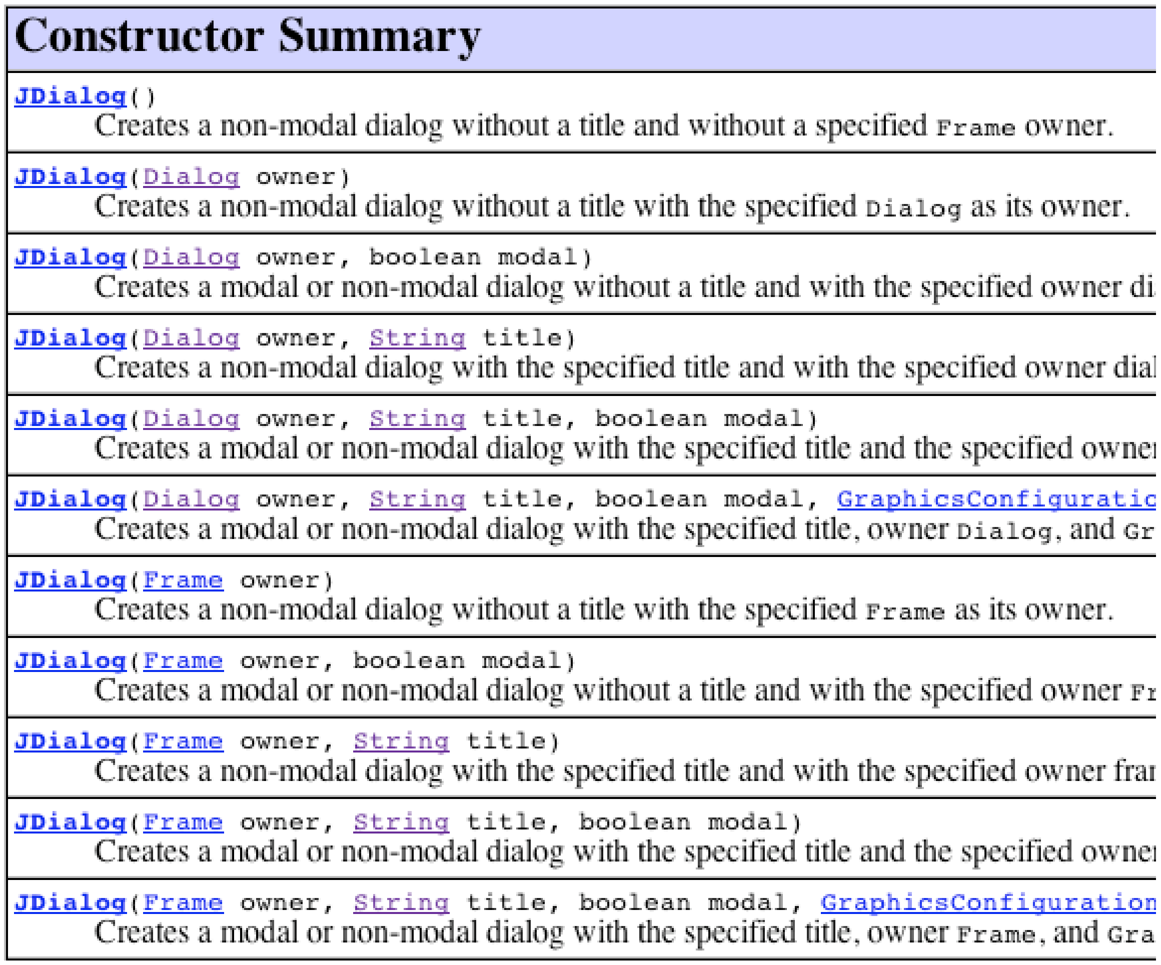
Exhibit i—Any one of these four lines dropped will cause the associated interactor to not show up at all, an easy miss since these lines (adding the label and the text field) fall more than a couple lines away from their creation and attribute specifications.

B. Informative runtime errors (rating: **+3** for Error recovery)
Summary: The current version of Swing addresses some of the incompatibilities with the buggy underlying AWT through quick runtime exits and fixing directions. For example, an application that calls `frame.setLayout,` the AWT way of setting layouts, exits at runtime with the following error println:

```
Exception in thread "main" java.lang.Error: Do
not use javax.swing.JFrame.setLayout() use javax.
swing.JFrame.getContentPane().setLayout() instead
```

C. Bad constructor abstractions (rating: **-2** for Flexibility)
Summary: The Swing API keeps improving with abstractions such as the `setDefaultCloseOperation` method for the `JFrame` whereas before one had to add a `WindowListener` that did a clean exit upon window closing. However, several

annoyances persist. The `JDialog`, for example, could cut out four constructors if Dialogs and Frames were properly abstracted:



Constructor Summary
`JDialog()` Creates a non-modal dialog without a title and without a specified `Frame` owner.
`JDialog(Dialog owner)` Creates a non-modal dialog without a title with the specified `Dialog` as its owner.
`JDialog(Dialog owner, boolean modal)` Creates a modal or non-modal dialog without a title and with the specified owner di
`JDialog(Dialog owner, String title)` Creates a non-modal dialog with the specified title and with the specified owner dial
`JDialog(Dialog owner, String title, boolean modal)` Creates a modal or non-modal dialog with the specified title and the specified owner
`JDialog(Dialog owner, String title, boolean modal, GraphicsConfiguratic` Creates a modal or non-modal dialog with the specified title, owner `Dialog`, and `Gr`
`JDialog(Frame owner)` Creates a non-modal dialog without a title with the specified `Frame` as its owner.
`JDialog(Frame owner, boolean modal)` Creates a modal or non-modal dialog without a title and with the specified owner `Fr`
`JDialog(Frame owner, String title)` Creates a non-modal dialog with the specified title and with the specified owner fran
`JDialog(Frame owner, String title, boolean modal)` Creates a modal or non-modal dialog with the specified title and the specified owner
`JDialog(Frame owner, String title, boolean modal, GraphicsConfiguration` Creates a modal or non-modal dialog with the specified title, owner `Frame`, and `Gra`

As a result, the programming user needs to create unnecessary duplicate code if the code wanted to deal with owner windows in general.

Fix: For this particular case, the `JDialog` constructors should be taking a `Window` as the constructor, or another made-up abstraction if the `Window` is too high, especially since dialogs and frames are used so often.

D. Dependence on call order (rating: **-4** for Error prevention)
Summary: Methods that depend on other certain actions to have been made need to be well documented since the programming user has no means of finding out other than the documentation. For instance, the bounds of an interactor are only updated after a panel is made visible. The last three lines of code causes the button to appear nondeterministically:

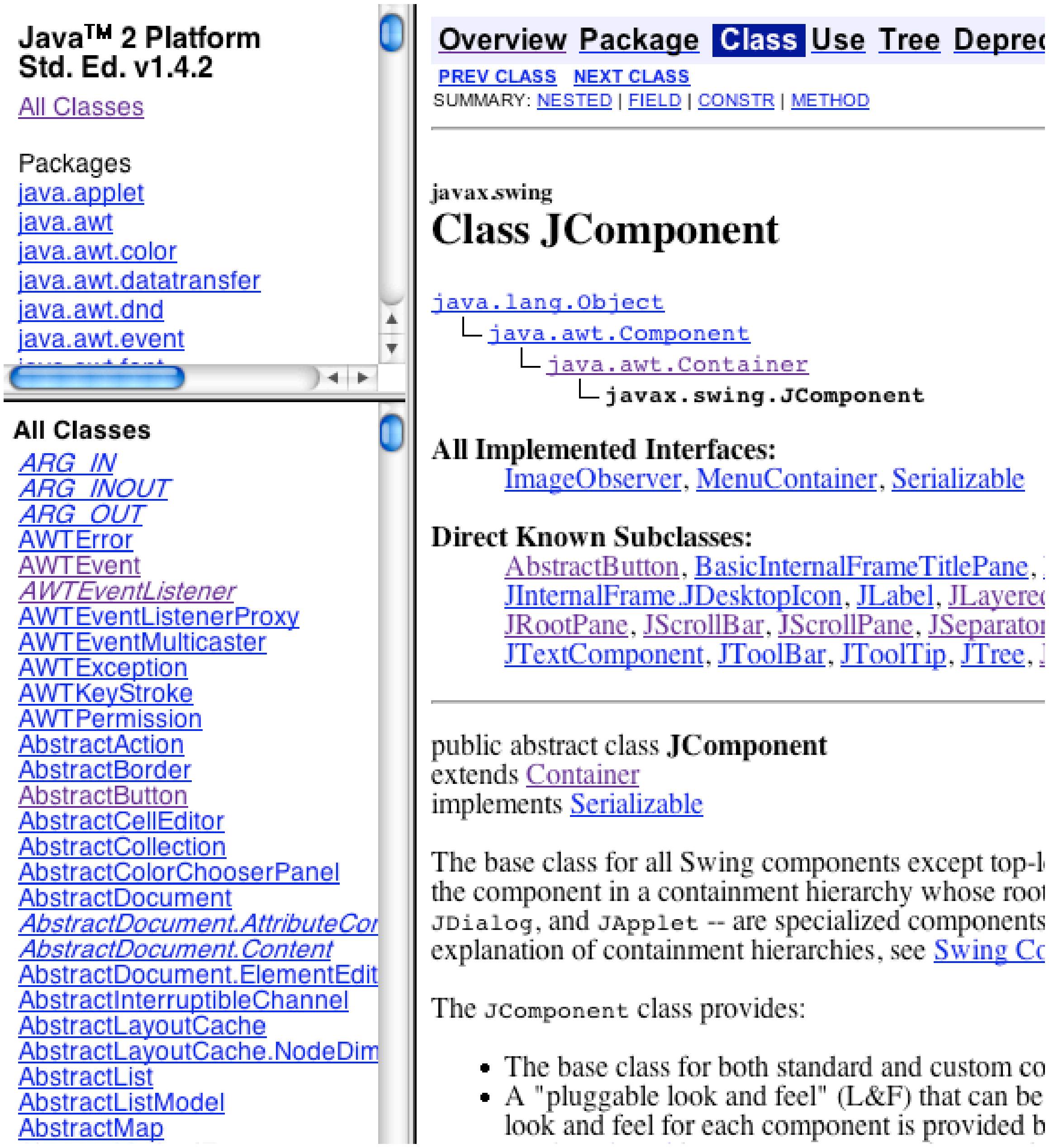
```
        final JFrame frame = new JFrame();
        frame.setSize(MAX_WIDTH, MAX_HEIGHT);
```

```
content = frame.getContentPane();
content.setLayout(null);
frame.setDefaultCloseOperation(JFrame.
  EXIT_ON_CLOSE);
final JButton button = new JButton();
button.setBounds(10, 10, 10, 10);
frame.setVisible(true);
frame.getContentPane().add(button);
```

If the `setBounds` method is put at the end of this code block, the button shows up all the time.

Fix: Better documentation could prevent such erroneous call flows since nowhere in the Javadoc does it mention such a requirement, Another fix would be to change the method name itself to reflect its dependence: for instance `getBounds` could be changed to `getVisibleBounds`.

E. The Javadoc as a reference (rating: **+3** for Documentation)
Summary: The Javadoc has a pretty good interface, especially for expert users. The left hand panels allow packages and classes to be filtered. The main pane displaying all the classes lists the hierarchical structure of each class at the front:



Java™ 2 Platform
Std. Ed. v1.4.2

All Classes

Packages
java.applet
java.awt
java.awt.color
java.awt.datatransfer
java.awt.dnd
java.awt.event

All Classes
ARG_IN
ARG_INOUT
ARG_OUT
AWTError
AWTEvent
AWTEventListener
AWTEventListenerProxy
AWTEventMulticaster
AWTException
AWTKeyStroke
AWTPermission
AbstractAction
AbstractBorder
AbstractButton
AbstractCellEditor
AbstractCollection
AbstractColorChooserPanel
AbstractDocument
AbstractDocument.AttributeCo
AbstractDocument.Content
AbstractDocument.ElementEdit
AbstractInterruptibleChannel
AbstractLayoutCache
AbstractLayoutCache.NodeDim
AbstractList
AbstractListModel
AbstractMap

Overview Package **Class** Use Tree Depre
PREV CLASS NEXT CLASS
SUMMARY: NESTED | FIELD | CONSTR | METHOD

javax.swing
Class JComponent

```
java.lang.Object
  └java.awt.Component
      └java.awt.Container
          └javax.swing.JComponent
```

All Implemented Interfaces:
ImageObserver, MenuContainer, Serializable

Direct Known Subclasses:
AbstractButton, BasicInternalFrameTitlePane,
JInternalFrame.JDesktopIcon, JLabel, JLayere
JRootPane, JScrollBar, JScrollPane, JSeparato
JTextComponent, JToolBar, JToolTip, JTree,

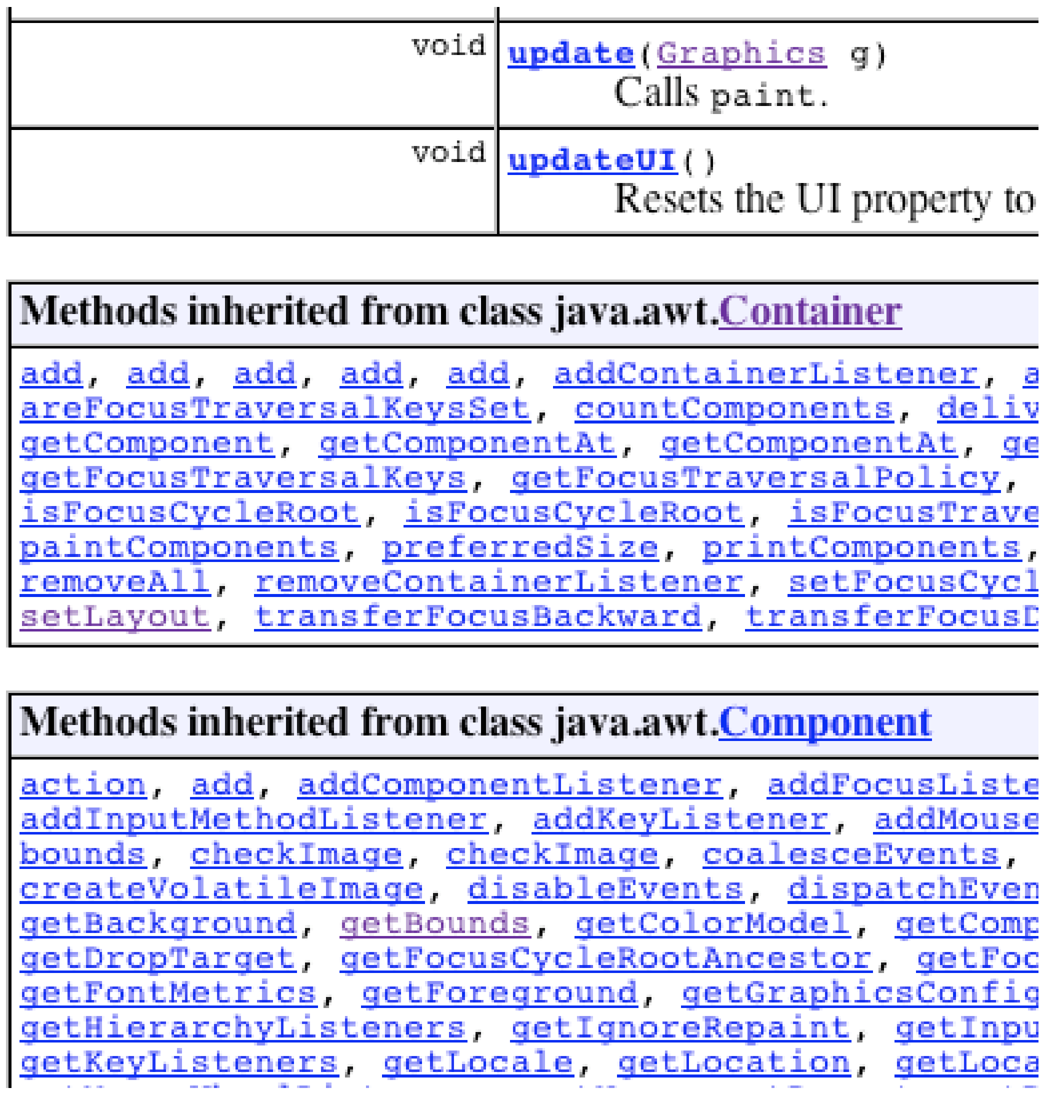
public abstract class **JComponent**
extends Container
implements Serializable

The base class for all Swing components except top-l
the component in a containment hierarchy whose roo
`JDialog`, and `JApplet` -- are specialized components
explanation of containment hierarchies, see Swing Co

The `JComponent` class provides:

- The base class for both standard and custom co
- A "pluggable look and feel" (L&F) that can be look and feel for each component is provided b

Unfortunately, sometimes inherited fields and methods aren't as apparent because they get bunched to the bottom; however, given the number of methods, it seems this less detailed choice of display is best for consistency across all Swing classes (and other Java as well):



void	update(Graphics g) Calls paint.
void	updateUI() Resets the UI property to

Methods inherited from class java.awt.Container

add, add, add, add, add, addContainerListener, a
areFocusTraversalKeysSet, countComponents, deliv
getComponent, getComponentAt, getComponentAt, ge
getFocusTraversalKeys, getFocusTraversalPolicy,
isFocusCycleRoot, isFocusCycleRoot, isFocusTrave
paintComponents, preferredSize, printComponents,
removeAll, removeContainerListener, setFocusCycl
setLayout, transferFocusBackward, transferFocusD

Methods inherited from class java.awt.Component

action, add, addComponentListener, addFocusListe
addInputMethodListener, addKeyListener, addMouse
bounds, checkImage, checkImage, coalesceEvents,
createVolatileImage, disableEvents, dispatchEven
getBackground, getBounds, getColorModel, getComp
getDropTarget, getFocusCycleRootAncestor, getFoc
getFontMetrics, getForeground, getGraphicsConfig
getHierarchyListeners, getIgnoreRepaint, getInpu
getKeyListeners, getLocale, getLocation, getLoca

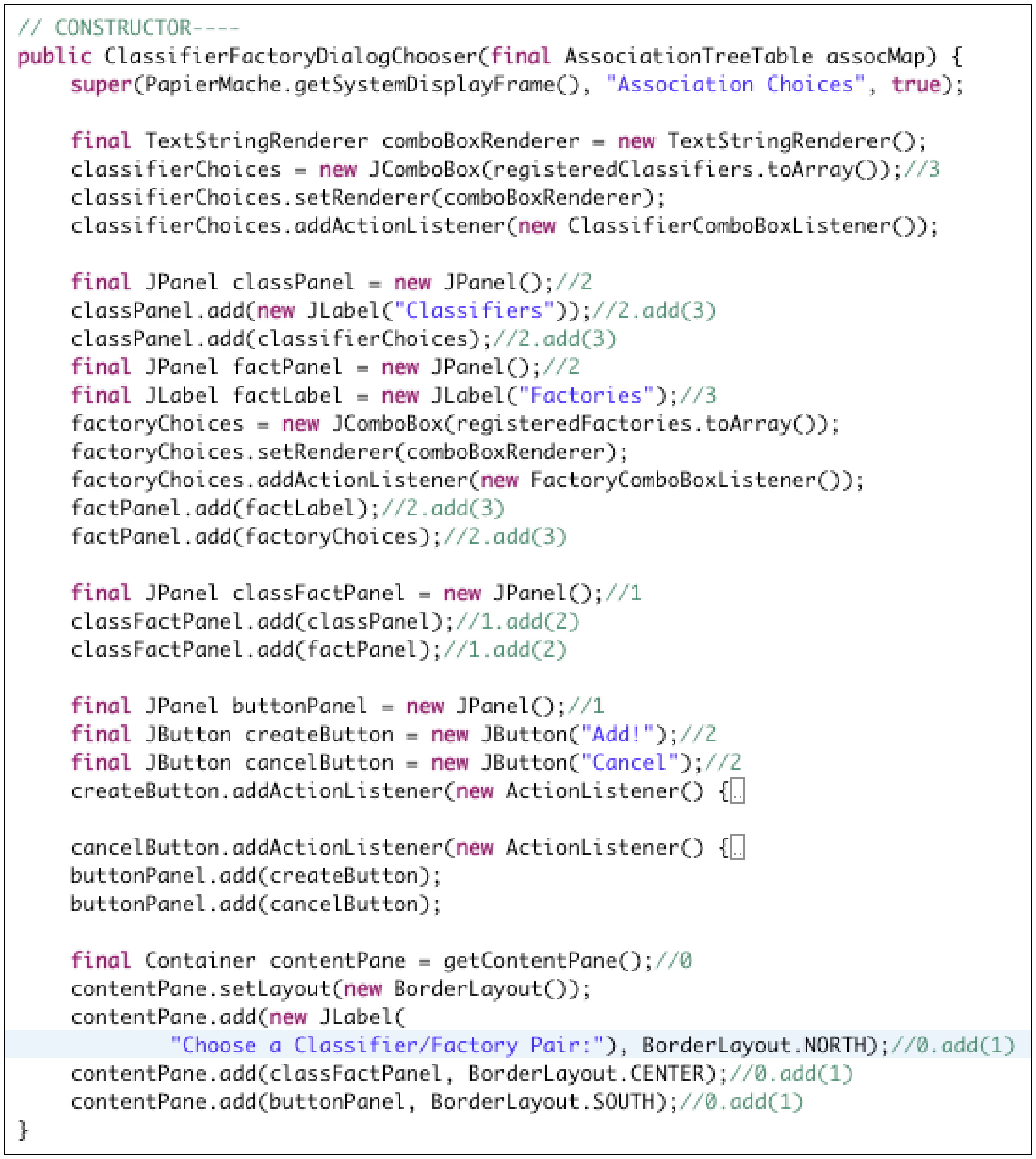
F. Method call collisions (rating: **-2** for Error prevention)
Summary: Methods that don't have obvious overriding effects (for instance, a `setBounds` method called after another `setBounds` call from the same object) ought to notify users of their similarity. For instance, the JFrame's `pack` method overrides its `setSize` method. The documentation mentions that `pack` sets the window to its preferred size, but the notion of having different types of sizes requires a level of knowledge that's not readily intuitive.

Fix: Methods could be better renamed to describe their functionality. `pack` could be renamed to `packToPreferredSize`, for instance. Supporting documentation should be updated to list related methods (`pack` should reference the size methods).

G. Language counterparts (rating: **+4** for Match)
Summary: Interactor creation follows fairly straightforward with buttons mapping to JButtons, checkboxes JCheckBoxes, comboboxes JComboBoxes. Higher-level containers

require slightly more reading about their usage especially for JRootPanes and JLayerPanes, which allow a more familiar user to create more complex GUIs.

H. A line for every attribute (rating: **+3** for Aesthetics)
Summary: Programming GUIs at the toolkit level inevitably leads to having a line for each attribute created. Although this explicit specification allows a code reviewer to understand the code more easily, the complexity of the interface correlates directly with the legibility of the code, especially when vast amounts of attributes cloud any hierarchical structure that may have been ascertained with simpler interfaces. Take the given GUI constructor:



```java
// CONSTRUCTOR----
public ClassifierFactoryDialogChooser(final AssociationTreeTable assocMap) {
    super(PapierMache.getSystemDisplayFrame(), "Association Choices", true);

    final TextStringRenderer comboBoxRenderer = new TextStringRenderer();
    classifierChoices = new JComboBox(registeredClassifiers.toArray());//3
    classifierChoices.setRenderer(comboBoxRenderer);
    classifierChoices.addActionListener(new ClassifierComboBoxListener());

    final JPanel classPanel = new JPanel();//2
    classPanel.add(new JLabel("Classifiers"));//2.add(3)
    classPanel.add(classifierChoices);//2.add(3)
    final JPanel factPanel = new JPanel();//2
    final JLabel factLabel = new JLabel("Factories");//3
    factoryChoices = new JComboBox(registeredFactories.toArray());
    factoryChoices.setRenderer(comboBoxRenderer);
    factoryChoices.addActionListener(new FactoryComboBoxListener());
    factPanel.add(factLabel);//2.add(3)
    factPanel.add(factoryChoices);//2.add(3)

    final JPanel classFactPanel = new JPanel();//1
    classFactPanel.add(classPanel);//1.add(2)
    classFactPanel.add(factPanel);//1.add(2)

    final JPanel buttonPanel = new JPanel();//1
    final JButton createButton = new JButton("Add!");//2
    final JButton cancelButton = new JButton("Cancel");//2
    createButton.addActionListener(new ActionListener() {…

    cancelButton.addActionListener(new ActionListener() {…
    buttonPanel.add(createButton);
    buttonPanel.add(cancelButton);

    final Container contentPane = getContentPane();//0
    contentPane.setLayout(new BorderLayout());
    contentPane.add(new JLabel(
            "Choose a Classifier/Factory Pair:"), BorderLayout.NORTH);//0.add(1)
    contentPane.add(classFactPanel, BorderLayout.CENTER);//0.add(1)
    contentPane.add(buttonPanel, BorderLayout.SOUTH);//0.add(1)
}
```

Each interactor has at most two attributes specified and already this six-widget GUI (counting only the leaf interactors of two buttons, two labels and two combo boxes) takes up an entire screen. The interactor tree has a depth of three, which may be harder to ascertain without the end line comments.

I. Threading handled awkwardly (rating: **-2** for Error prevention)
Summary: Swing uses its library function `invokeLater` to handle all issues that come up with asynchronous dependent updates. This method works waiting for a few synchronizations between the

Fix: The operating systems community acknowledges thread programming as a useful but error-prone style. Better thread management at the systems level may eventually bring about better management at the toolkit level beyond a single method for threads of any kind.

J. Panes have no internal meaning (rating: **-1** for Consistency)
Summary: Panes are prevalent in Swing so much that an abstraction for them seems appropriate at some level of the toolkit. The JFrame has a contentPane that's actually a container; JLayeredPanes, JRootPanes, JGlassPanes are all classes that exist, but the reason that these components are panes is not clear, certainly not from looking at their internal representation. There's also a JPanel that appears related, but has no relation hierarchically.

Fix: Calling all these different components "panes" should be rethought so that the toolkit's API is consistent with its internal structures. The solution may include getting rid of the word "pane," or creating a new JPane abstraction.

K. Layouts hard to customize (rating: **-3** for User control)
Summary: Layouts by nature are a visual design problem. Swing provides a library of ten or so layouts for a regular pane, but the layouts are often too inflexible or too low-level. `FlowLayouts` lay components out one after another so you have no control with spacing between individual components for instance. `GridBagLayouts` allow you direct control of where components should be placed directly in the window, but all semantic notions of hierarchical grouping get lost.

Fix: The problem of maintaining both hierarchical groupings and allowances for visual design requires current interface builders 1) to allow for the former and 2) to create an interactive visualization that supports 1). Since Swing deals with toolkit level GUI creation, it should provide better layouts to allow for more flexible manipulations.