CODE SMELLS

Collapsible "if" statements should be merged

Illustrating code snippet:

if (cli.isWriteMetadatatoPdf() || cli.isWriteXMPtoPdf() || cli.isEmbeddBibfileInPdf()) {  
 if (!loaded.isEmpty()) {  
 writeMetadatatoPdf(loaded,  
 cli.getWriteMetadatatoPdf(),  
 preferencesService.getGeneralPreferences().getDefaultEncoding(),  
 preferencesService.getXmpPreferences(),  
 preferencesService.getFilePreferences(),  
 preferencesService.getGeneralPreferences().getDefaultBibDatabaseMode(),  
 Globals.*entryTypesManager*,  
 preferencesService.getFieldWriterPreferences(),  
 cli.isWriteXMPtoPdf() || cli.isWriteMetadatatoPdf(),  
 cli.isEmbeddBibfileInPdf() || cli.isWriteMetadatatoPdf());  
 }  
}

The exact location on the codebase:

package org.jabref.cli;

public class ArgumentProcessor{

An explanation of the rationale for identifying this code smell:

Merging collapsible if statements increases the code’s readability. You can join all if statements in one sentence

A refactoring proposal:

if ((cli.isWriteMetadatatoPdf() || cli.isWriteXMPtoPdf() || cli.isEmbeddBibfileInPdf()) && !loaded.isEmpty()) {  
   
 writeMetadatatoPdf(loaded,  
 cli.getWriteMetadatatoPdf(),  
 preferencesService.getGeneralPreferences().getDefaultEncoding(),  
 preferencesService.getXmpPreferences(),  
 preferencesService.getFilePreferences(),  
 preferencesService.getGeneralPreferences().getDefaultBibDatabaseMode(),  
 Globals.*entryTypesManager*,  
 preferencesService.getFieldWriterPreferences(),  
 cli.isWriteXMPtoPdf() || cli.isWriteMetadatatoPdf(),  
 cli.isEmbeddBibfileInPdf() || cli.isWriteMetadatatoPdf());  
   
}

Synchronized classes Vector, Hashtable, Stack and StringBuffer should not be used

Illustrating code snippet:

Vector<String> citeKeys = new Vector<>();

The exact location on the codebase:

package org.jabref.cli;

public class ArgumentProcessor{

An explanation of the rationale for identifying this code smell:

Early classes of the Java API, such as Vector, Hashtable and StringBuffer, were synchronized to make them thread-safe. Unfortunately, synchronization has a big negative impact on performance, even when using these collections from a single thread.

It is better to use their new unsynchronized replacements:

ArrayList or LinkedList instead of Vector

A refactoring proposal:

ArrayList<String> citeKeys = new ArrayList<>();

String literals should not be duplicated

Illustrating code snippet:

public boolean isPreferencesExport() {  
 return cl.hasOption("prexp");  
}  
  
public String getPreferencesExport() {  
 return cl.getOptionValue("prexp", "jabref\_prefs.xml");  
}

options.addOption(Option  
 .*builder*("x")  
 .longOpt("prexp")  
 .desc(String.*format*("%s: '%s'", Localization.*lang*("Export preferences to a file"), "-x prefs.xml"))  
 .hasArg()  
 .argName("[FILE]")  
 .build());

The exact location on the codebase:

package org.jabref.cli;  
  
  
public class JabRefCLI {

An explanation of the rationale for identifying this code smell:

Duplicated string literals make the process of refactoring error-prone, since you must be sure to update all occurrences.

On the other hand, constants can be referenced from many places, but only need to be updated in a single place.

A refactoring proposal:

private static final String *PREXP* = "prexp";

public boolean isPreferencesExport() {  
 return cl.hasOption(*PREXP*);  
}  
  
public String getPreferencesExport() {  
 return cl.getOptionValue(*PREXP*, "jabref\_prefs.xml");  
}

options.addOption(Option  
 .*builder*("x")  
 .longOpt(*PREXP*)  
 .desc(String.*format*("%s: '%s'", Localization.*lang*("Export preferences to a file"), "-x prefs.xml"))  
 .hasArg()  
 .argName("[FILE]")  
 .build());

Local variables should not shadow class fields

Illustrating code snippet:

private Optional<Color> color;

public Node getGraphicNode() {  
 Ikon icon = icons.get(0);  
 FontIcon fontIcon = FontIcon.*of*(icon);  
 fontIcon.getStyleClass().add("glyph-icon");  
  
// Override the default color from the css files  
 color.ifPresent(color -> fontIcon.setStyle(fontIcon.getStyle() +  
 String.*format*("-fx-fill: %s;", ColorUtil.*toRGBCode*(color)) +  
 String.*format*("-fx-icon-color: %s;", ColorUtil.*toRGBCode*(color))));  
  
 return fontIcon;  
 }

The exact location on the codebase:

package org.jabref.gui.icon;  
  
public class InternalMaterialDesignIcon implements JabRefIcon

An explanation of the rationale for identifying this code smell:

Overriding or shadowing a variable declared in an outer scope can strongly impact the readability, and therefore the maintainability, of a piece of code.

It makes it confusing to know whether the field or the variable is and should be accessed.

In this case “color” hide the field declared at the first line

A refactoring proposal:

color.ifPresent(incolor -> fontIcon.setStyle(fontIcon.getStyle() +  
 String.*format*("-fx-fill: %s;", ColorUtil.*toRGBCode*(incolor)) +  
 String.*format*("-fx-icon-color: %s;", ColorUtil.*toRGBCode*(incolor))));

Strings and Boxed types should be compared using "equals()"

Illustrating code snippet:

this.liveMode.addListener((observable, oldValue, newValue) -> {  
 // Switch to currently selected entry if mode is changed to live  
 if ((oldValue != newValue) && newValue) {  
 setCurrentEntries(this.stateManager.getSelectedEntries());  
 }  
});

The exact location on the codebase:

package org.jabref.gui.documentviewer;  
  
public class DocumentViewerViewModel extends AbstractViewModel

An explanation of the rationale for identifying this code smell:

It’s almost always a mistake to compare two instances of java.lang.String or boxed types like java.lang.Integer using reference equality == or !=, because it is not comparing actual value but locations in memory.

A refactoring proposal:

this.liveMode.addListener((observable, oldValue, newValue) -> {  
 // Switch to currently selected entry if mode is changed to live  
 if ((oldValue.equals(newValue)) && newValue) {  
 setCurrentEntries(this.stateManager.getSelectedEntries());  
 }  
});

Speculative generality

Illustrating code snippet:

private final DateFormat dateFormat = new SimpleDateFormat("yyyyMMddHHmmss");  
private final Date date = new Date();  
private final DialogService dialogService;  
private final ClipBoardManager clipBoardManager;  
private final BuildInfo buildInfo;  
private final ListProperty<LogEventViewModel> allMessagesData;

The exact location on the codebase:

package org.jabref.gui.errorconsole;  
  
public class ErrorConsoleViewModel extends AbstractViewModel

An explanation of the rationale for identifying this code smell:

Occurs when you make code that is not needed at the time, but that may be useful someday. This practice introduces generality that may not actually help the code, but “over-engineers” it.

If a private field is declared but not used in the program, it can be considered dead code and should therefore be removed. In this case the unused fields “dateFormat” and date.

A refactoring proposal:

private final DialogService dialogService;  
private final ClipBoardManager clipBoardManager;  
private final BuildInfo buildInfo;  
private final ListProperty<LogEventViewModel> allMessagesData;