# a4

## Team

Nick Lawrence and Satya Kalyan

## jEdit

1. Automatic Refactorings

    a. Feature Envy in *installer.TarEntry*

        i. Used the *Move Method* automatic refactoring technique on *installer.TarEntry.nameTarHeader()* and *installer.TarEntry.parseTarHeader*() to move them to the *installer.TarHeader* class

        ii. JDeodorant discovered Feature Envy with these methods, and it makes much more logical sense for them to be in the *installer.TarHeader* class rather than in *installer.TarEntry*.

        iii. Eclipse was able to handle these operations all on its own, and no manual refactorings had to be done.

    b. God Class *org.gjt.sp.jedit.gui.SplashScreen*

        i. The instance variables *labelFont*, *versionColor1*, and *versionColor2* were moved into a separate class, *VersionLabel*, along with the method *paintString().* This was done automatically with Eclipse's extract class refactoring tool.

        ii. JDeodorant identified this class as a God Class that handled too many concepts on its own. Moving these variables to a separate class will increase the cohesion of *SplashScreen* but add a small amount of coupling.

        iii. All changes were done automatically with eclipse.

2. Manual Refactorings

    a. Feature Envy in *org.gjt.sp.jedit.gui.DockingLayoutManager*

        i. Manually moved

        *org.gjt.sp.jedit.gui.DockingLayoutManager.getCurrentEditMode()* to *org.gjt.sp.jedit.View*

        ii. JDeodorant discovered Feature Envy with this method, and it makes much more logical sense for it to be in the View class.

iii. All code changes were done by hand. The method was copied over and all places in the code that it was used were changed from *getCurrentEditMode(view)* to *view.getCurrentEditMode()*. This method was only used in *DockingLayoutManager* so the extent of the changes were limited.

b. Type Checking on *org.gjt.sp.jedit.textarea.Selection*

i. Many minor changes had to be made to add polymorphism to the *SearchAndReplace* operation

1. A new abstract function, *replaceInSelection()* was added to Selection.

2. The implementation of the function was added for *Selection.Rect* and *Selection.Range*

3. *org.gjt.sp.jedit.search.SearchAndReplace* was modified to use *Selection.replaceInSelection()* instead of type checking.

ii. JDeodorant identified these *instanceof* type checks to be smelly, because polymorphism can be used to make the code more clear and to make a better use of Object Oriented design. It will make the code more cohesive overall.

iii. All code changes were done manually.

## pdfSam

1. Automatic Refactorings

a. God Class *org.pdfsam.pdf.PdfDocumentDescriptor*

i. *AtomicInteger* references was moved to a separate class along with its associated methods, *hasReferences()*, *release(), releaseAll(),* and *retain().* This was all done with Eclipse's automatic tools.

ii. JDeodorant identified this class as a God Class and these methods as things that don't belong in the *PdfDocumentDescriptor* class itself. This use of the *AtomicInteger* could potentially be used easier in other classes as well if it was separated from *PdfDocumentDescriptor*.

iii. None of the code changes had to be performed manually, everything was done automatically by Eclipse.

b. Long Method

*org.pdfsam.ui.selection.multiple.SelectionTable.showPasswordFieldPopup*

i. This method was split into two separate methods, the new method being *displayPasswordFieldPopup*, which will handle the actual display of the popup if the right conditions are met.

ii. JDeodorant identified this method as performing multiple separate tasks, which should be handled by multiple methods. The task of determining whether to display the password field popup was assigned to *showPasswordFieldPopup()* while the task of actually displaying the popup was given to *displayPasswordFieldPopup().*

iii. None of the code changes had to be performed manually. Everything was done automatically by Eclipse.

2. Manual Refactorings

a. God Class *org.pdfsam.ui.ContentPane*

i. This class is considered smelly because there are many sets of local variables that relate to each other, but not to the rest of the class. Specifically, three local variables *newsContainer, fadeIn,* and *fadeOut*, can be seamlessly moved to another class since they are only used in conjunction with each other, and only in a few methods. This will keep the class more cohesive, but add a small amount of coupling.

ii. Instance variables *VBox newsContainer*, *FadeTransition fadeIn*, and *FadeTransition fadeOut*, along with associated methods were moved into a separate class called *ContentPaneContainer*. A constructor had to be created that performed the initialization of these components that was originally done in the constructor for *ContentPane*. Getters and setters for all of the local variables were also created.

iii. JDeodorant identified this class as a God Class and these variables as things that should be extracted into a separate class. This was done to make the class more cohesive.

iv. All changes were done manually.

1) Copied local variables into a new class file

2) Created getters and setters for the variables.

3) Created new local variable *contentPaneContainer* in *ContentPane* of type *ContentPaneContainer*.

4) Replaced uses of the old local variabes with the getters/setters of *ContentPaneContainer*.

5) Methods *onShowNewsPanel* and *onHideNewsPanel* were moved to *ContentPaneContainer*.

6) The constructor for *ContentPaneContainer* was created to do the initialization that was originally done in *ContentPane* for these variables.

b. Long Methods in *org.pdfsam.ui.dialog.OverwriteDialogController*

i. This is considered a code smell because two methods in *OverwriteDialogController* had six lines of duplicated code. It is never good to have duplicated code because a change to one will likely require a change to the other. By creating a new helper method to replace these six lines in both methods, it will make the code more easy to maintain in the future.

ii. The functions *onDirectory* and *onFile* had some duplicated code that could be separated out into a new method. The new method *handleOverwrite()* was created to be placed in both these methods.

iii. This was done to eliminate the duplicated code that JDeodorant identified in this class.

iv. All code changes were done manually, by writing a new method based off the old duplicated code, then replacing the duplicated code with two method calls to the *handleOverwrite()* method we created.

## Differences between Automatic and Manual Refactoring Techniques

The automatic refactoring performed by eclipse can be incredibly helpful and can save a significant amount of time in your refactoring. Refactoring manually can be incredibly time consuming especially if the changes are widespread across the code-base. However, the automatic refactoring system is not perfect. There are a few operations at which it performs exceptionally well, such as extract method and move method. Other, more complex operations can lead to unforeseen consequences or errors. If the changes are widespread and Eclipse makes an error in the refactoring, the amount of time it takes to track down all the new errors or bugs could be longer than the time it would have taken to just perform manual refactoring. We discovered for simple operations, such as Move Method and Extract Method, automatic refactoring is the way to go. However, for more complex operations, either a combination of automatic refactoring and manual refactoring should be used or just manual refactoring. When performing tasks like renaming variables or classes, automatic refactoring should always be used, as Eclipse will never mess that up and the time taken is significantly less than changing names by hand in every location they are used. It is just important to understand what the automatic tools are capable of handling.