Brad Jakobitz
Nate Olson
CS581a3 (spring ‘18)
Assignment 4

jEdit

1. Analysis (Feature Envy, God class, Type Checking, Feature Envy)
    a. Feature Envy - PaintHighlight
        - Moved the getOffsets and paintHighlight methods from the org.gjt.sp.jedit.textarea.Highlight class TextArea class. Both of these methods were moved to the TextArea class and the Highlight class called these moved methods.
        - Test case
            1. Open a new jEdit file
            2. Type this block of text into the new file
                a. {
                b. {
                c. this is some text inside of brackets
                d. this is another line
                e. }
                f. }
            3. Put the cursor on the right of the first bracket. The last bracket should have a box drawn around it.
            4. Put your cursor to the right of the last bracket. The first bracket should have a box around it.
            5. Repeat steps 3 and 4 with the inner bracket pair.
    b. God class - Operations
        - A new org.gjt.sp.jedit.pluginmgr.RosterOperations class was created to handle the functionality of the Operations list that was in the Roster class. Methods that add, remove, and check the operations list are now in the RosterOperations class. This new class also starts threads for the Operations to be done. This took some of the excessive functionality from the org.gjt.sp.jedit.pluginmgr.Roster class.
        - Test case
            1. Open the plugin manager → Plugins → Plugin Manager
            2. Click the install tab
            3. Choose the android plugin from the list
            4. Click install
            5. This should install the plugin and show the new plugin in the manage tab.
    c. Type Checking - Rotation (Manual)
        - Created an abstract org.gjt.sp.jedit.gui.RotationType class to hold the static rotation codes. It has an abstract method that gets the rotation

code. It has an abstract method that does the rotation work. It has a factory method that creates a concrete rotation type based on the rotation code. Three rotation concrete classes were created to implement the abstract classed based on which direction the icon was supposed to rotate. The switch statement that switched on the rotation code was removed and replaced with an abstract rotate method that used polymorphism to rotate the icon the correct direction. Then in the PanelWindowContainer, the int rotation had to be changed to the new RotationType object.

- Test case
    1. Open the file browser dockable window. Utilities → File System Browser
    2. Click the down arrow in the top left of the File System Browser window.
    3. Choose dock at left
    4. The File Browser button should rotate counter clockwise and appear at the left of the jEdit window.
    5. Click the down button again and choose dock at right
    6. The File Browser button should rotate clockwise and move to the right of the jEdit window.

d. Feature Envy - org.gjt.sp.jedit.Markers (Manual)
    - Moved the addMarker method from the EditPanel class to the org.gjt.sp.jedit.textarea.TextArea class. A Buffer parameter was added because the TextArea did not have the correct buffer.
    - Test case
        1. Type some text on the first line
        2. Right click the line and choose Add/Remove Marker
        3. The section to the left of the text with the line number should be highlighted green.
        4. A marker should also be added to the Markers menu item with the lines text as the name.

2. Rational
    a. PaintHighlight
        - This method was moved because all the variables used were either passed in as arguments or belonged to the Text Area class
    b. Operations
        - By moving the operations and all the functions pertaining to the list the responsibilities of the Roster class lessened.
    c. Rotation
        - The code is easier to read because the switch statement is replaced by polymorphism. This also made the program more maintainable because if a rotation type is added then it is just an addition to the RotationType class and a new concrete class that inherits from RotationType.

    d. Markers
        - Most of the methods that pertain to markers such as add, remove, next have more references to TextArea than any other class so all these methods should be moved to the TextArea class but I started with addMarker because it was the simplest.
3. Manual code changes
    a. I assume that the only manual changes required in this section are the refactoring changes pertaining to the automatic refactoring so I will skip the manual refactorings.
    b. PaintHighlight also required me to move the getOffsets method that is called. Other than that it was just calling the internal TextArea methods with “this” instead of textArea.method().
    c. The Operation refactor only required me to import the new RosterOperation class to Roster. The automation did most of the work.
4. Differences
    a. The automation was easier in most cases because a bulk of the changes were moving methods. The God class automated refactoring was difficult because there is a bug where if you want to change the name of the new class it does not always create the class.
    b. Another difficulty I found with the automation was it was not good at predicting the consequences of a change. In one case I had to move additional methods. In another case I had opt out of the suggested refactoring because it was trying to refactor an abstract class which does not work well with polymorphism.
    c. The manual refactoring took more time because it was more typing but it was easier to make the changes because I made each change and understood the reasons and consequences.
    d. I think if I used this tool again I would only use it as a suggestion but then do the refactoring myself so there were no surprises as to how it thought the refactoring should be done.

PDFSam

There are two instances of Long Method described below because out of the 5 code smells (Duplicated Code, God Class, Feature Envy, Long Method and Type Checking) there were no reports of duplicated code and only two instances of type checking. I personally did not agree with the refactoring steps that would have been required in either case for the type checking. The two cases are described below with my rationale on why not to refactor them.

- In pdfsam-fx/src/main/java/org/pdfsam/ui/selection/multiple/SelectionChangedEvent::canMove(MoveType) : boolean there is a switch statement that determines whether or not a move is possible based on the current selection. This is only a single switch statement in the entire class and it only has four possible cases: BOTTOM, DOWN, TOP, UP. Because this selection is dealing with a list of things (meaning left and right moves are not

possible), refactoring this would be inefficient as the possibility that this would need to be expanded in future cases is minimal.

- In pdfsam-fx/src/main/java/org/pdfsam/ui/io/RememberLatestFileChooserWrapper::showDialog : File there is a switch statement that determines which type of dialog to show the user. There are current two types allowed to show a save dialog or to show an open dialog. Following the logic from above, I do not believe this should be refactored because there are not many other operational dialogs to show for a file. In most programs you either open a file or save a file - there are not many other choices. I do not believe the effort required to refactor this type checking is warranted.

1. Smell description and explanation
    a. Long Method (Auto)
        - This smell is reported by JDeodorant and is not included in our assignment 3. In pdfsam-fx/src/main/java/org/pdfsam/ui/io/BrowsableFileField::enforceValidation(boolean, boolean) : boolean, there are 4 lines of code that are all used to set a Validator. In total this method is 6 lines long and 4 of these lines have to do with getting a Validator. JDeodorant recommends splitting these four lines into a method of its own.
        - I agree with this code smell by JDeodorant. Although this method is only 6 lines long, the fact that two-thirds of the lines in the method are setting a Validator means this method of 'enforceValidation' is doing more work than it should. If these four lines were split out into its own method, then the enforceValidation method could be just two or three lines, depending on programming style, and the new method 'getValidator' would explain exactly what it is used for.
    b. Feature Envy (Auto)
        - This smell is reported by JDeodorant and is not included in our assignment 3. In pdfsam-gui/src/main/java/org/pdfsam/WindowStatusController, there is a method 'hasAvailableScreen' that takes a StageStatus as an argument and then checks whether the screen is available based on the coordinates of the pdfsam-core/src/main/java/org/pdfsam/ui/StageStatus. This method uses none of the member variables within the WindowStatusController and instead calls many methods from the passed in StageStatus method.
        - I agree with this code smell by JDeodorant. This method would be much better in the StageStatus class where it has local access to all of the methods that are being called. In fact, the only thing required for this is StageStatus so this method should be moved there.
    c. Long Method (Manual)
        - This smell is reported by JDeodorant and included in our assignment 3. In pdfsam-fx/src/main/java/org/pdfsam/ui/selection/single/SingleSelectionPane::sho

wPasswordFieldPopup, it suggests that the action of actually getting and displaying the Window be moved to its own method.
    - This is reported as smelly because the code for getting the Window and displaying it to the user could be used in other code and does not strictly relate to showing the password field popup only.

d. God Class (Manual)
    - pdfsam-gui/src/main/java/org/pdfsam/ui/dashboard/modules/DashboardTile class is being reported by JDeodorant and included in our assignment 3. It is being reported as a God Class because there is a static initialization of a variable that is required to override the variables interface. After the static initialization is done then it requires the class to provide getters and setters for the static initialization. JDeodorant is recommending that this snippet of code be extracted to its own class and the current class just calls the getters of the class it is extracted to.
    - This is reported as smelly because it believes the responsibility of the static member variable does not apply to its current class and instead should be moved to a different class.

2. Description and analysis of code changes, and testing

a. Long Method (Auto)
    - This method is called anytime a field needs to be validated. It appears this happens most often to validate that an input exists and that an output path is not empty. It is subclassed by pdfsam-fx/src/main/java/org/pdfsam/ui/io/BrowsablePdfInputField and pdfsam-fx/src/main/java/org/pdfsam/ui/io/BrowsablePdfOutputField. These are almost certainly tested with the automated tests, but in addition to that, I will set a breakpoint in both of the subclasses and give it invalid input and output fields to ensure it handles them correctly. For initial testing, I have validated that enforceValidation is called with selectedFileMustExist=true and allowEmptyString=false for an input file. This appears to behave as expected, when the string is empty it complains or if the path I give it is not to an existing file it complains. For output it is called with selectedFileMustExist=false and allowEmptyString=false. It also behaves as expected. I cannot figure out a way through the GUI to get it to call this method in the other two possible combinations of these input variables.
    - Refactoring
        1. I highlighted the four lines JDeodorant recommends to be refactored, right-clicked in Eclipse and selected Refactor -> Extract Method…
        2. I gave the new method the name of ‘getValidator’ which takes the same two inputs as the previous method and returns a Validator to be used in the previous method.
        3. After doing this, I clicked OK and let Eclipse do the work.

- After performing refactoring with Eclipse, I performed the manual changes outlined in the next section to clean up the code and then performed the same manual steps for testing as described above. I saw the exact same behavior in the code as noted above and the results were also exactly the same as noted above. For good measure, I also executed all of the regressions provided with PDFsam and they ran as expected (with the single error I have always seen):

```
[ERROR] Failures:
[ERROR]   PrefixFieldTest.contextMenuReplacesText:67 expected:<[][BASENAME]> but was:<[PDFsam_][BASENAME]>
[INFO]
[ERROR] Tests run: 290, Failures: 1, Errors: 0, Skipped: 6
```

b. Feature Envy (Auto)

- pdfsam-gui/src/main/java/org/pdfsam/WindowStatusController::hasAvailableScreen method is called whenever the PDFsam app is started to initialize the window. Because of when this method is called, it will be covered by the automated test cases of PDFsam.
- Refactoring
    1. I highlighted the 'hasAvailableScreen(StageStatus)' method, right-clicked in Eclipse and selected Refactor -> Move…
    2. This gave me two options of where to move to 'StageStatus' or 'StageService'. Since JDeodorant recommended moving this to StageStatus, this is what I selected and clicked 'OK".
    3. There were a couple wizard windows I had to click through. These windows were basically asking me the name of the new method and a warning that the new method would be public (since the old method was private). I accepted the defaults. The only one I was thinking about checking was the option to leave the original method as a delegator. However, since this method is private this should not be necessary.
- After performing refactoring with Eclipse, I ran the regressions tests and they ran as expected (with the single error I have always seen):

```
[ERROR] Failures:
[ERROR]   PrefixFieldTest.contextMenuReplacesText:67 expected:<[][BASENAME]> but was:<[PDFsam_][BASENAME]>
[INFO]
[ERROR] Tests run: 290, Failures: 1, Errors: 0, Skipped: 6
[INFO]
```

c. Long Method (Manual)

- The only time it is called in the code is if the PDF file loaded is encrypted. In this case PDFSam displays a lock icon next to the input file. If this lock icon is clicked, then the passwordFieldPopup method is called to display a place for users to type their password. Instead, I am manually testing this by loading an encrypted PDF file, clicking this icon and typing the password and ensuring it can be loaded (which ensures it was decrypted). It functions correctly before modification. All regressions pass as usual before modification as well.
- Refactoring

1. I located the code recommended to be extracted using JDeodorant
2. I created an empty method stub called 'showPasswordPopupWindow' to move the code reported by JDeodorant to.
3. I cut the code highlighted by JDeodorant and pasted it into the new method. When doing this, I realized that my method needed to take a 'Scene' variable as input so I added this to the method signature and then called this new method from where I cut the original code from.

- I ran the exact same test manually as described above (loaded encrypted file, typed password, ensured it could be loaded) to make sure this change did not affect functionality. It functioned as expected after this change.
- After this refactoring the original code smell for long method in SingleSelectionPane::showPasswordFieldPopup is gone. However there is a new "Long Method" smell for our newly generated SingleSelectionPane::showPasswordPopupWindow method.
    1. I refactored this code smell using the same methods as described above. I first added a new method stub for 'showPasswordPopupWindowAtCoordinates'. I cut the code recommended by JDeodorant and pasted it in the new method stub. Doing so highlighted errors that required a 'Scene' and a 'Window' variable to be passed into the new method, so I added them. I then called this new method from the old method passing the appropriate 'Scene' and 'Window' variables.
    2. I re-ran the tests, it passed and there is no longer a code smell associated with any of these methods.

d. God Class (Manual)

- This static initialized variable is called each time a pdfsam-gui/src/main/java/org/pdfsam/ui/dashboard/modules/DashboardTile is loaded on the main, home screen of PDFsam. Because of how often this function is called; the automated tests cover the use-case of this variable and will be sufficient for testing.
- Refactoring
    1. The static initialization is done in the 'DashboardTile' class. The first step I took was to locate where this class was within the code and add a new empty class called 'DasboardTileArmed' in the same package. This is where I will move the 'armed' variable that JDeodorant is highlighting.
    2. I cut the code recommended by JDeodorant from the DashboardTile class and pasted it in the pdfsam-gui/src/main/java/org/pdfsam/ui/dashboard/modules/DashboardTil

eArmed class. This created a couple errors in both the DashboardTile and DashboardTileArmed class.

a. One error was that the code cut was referencing a private static final variable called ARMED_PSEUDOCLASS_STATE from the DashboardTile class. I simply moved this variable to the new class also as it was no longer being referenced in the original class.
b. The next error was also in the new DashboardTileArmed class and that was due to the fact the static initialization of the ReadOnlyBooleanWrapper was calling a super class method of DashboardTile. Since we moved this initialization, and our new class does not have the same inheritance, we no longer have a pointer to this method. To solve this, I created a constructor that took in a DashboardTile object, initialized a member variable and then used this member variable to call the method.
c. The last error was in the original DashboardTile class and it is because it still has a reference to this 'armed' variable that has been moved. I created a new member variable of DashboardTileArmed, called its constructor from the DashboardTile constructor passing the DashboardTile object to initialize it. After I did this, there was still a call to 'bind' method of the old member variable 'armed' which was of type ReadOnlyBooleanProperty. Our new DashboardTileArmed class does not have this method, so I created a pass through method in DashboardTileArmed that simply calls the bind method of the ReadOnlyBooleanWrapper.

3. After performing all of these refactorings, I re-ran the JDeodorant against the refactored to ensure the code smell was taken care of, and it was.
4. I ran the automated test suite that came with PDFsam and realized that my refactoring had caused an error in the pdfsam-gui/src/test/java/org/pdfsam/ui/dashboard/modules/ModulesDashboardTileTest. It was calling the 'isArmed' method of DashboardTile which has now been moved to the 'DashboardTileArmed' class. In order to fix this, I created a pass through method in the DashboardTile class to call the isArmed of the new class. After this fix, the test results were exactly the same as they have been all along (I have always seen this single failure):

```
[ERROR] Failures:
[ERROR]   PrefixFieldTest.contextMenuReplacesText:67 expected:<[][BASENAME]> but was:<[PDFsam_][BASENAME]>
[INFO]
[ERROR] Tests run: 290, Failures: 1, Errors: 0, Skipped: 6
```

3. Rationale of refactoring operations

a. Long Method (Auto)
    - This long method was hardly long, it was only 6 lines. However after inspecting the ‘enforceValidation’ method reported by JDeodorant it was obvious that two-thirds of the lines in this method were getting a Validator and actually had nothing to do with enforcing validation. Due to this, I agreed with JDeodorant that these four lines of code should be moved to a separate method called ‘getValidator’ this way the code is more self-explanatory when reading the code. The enforceValidation method now does exactly what it says by setting a Validator on a text field. It now gets this Validator from a method called getValidator which is highly self-describing.

b. Feature Envy (Auto)
    - This feature envy was obvious as soon as JDeodorant highlighted the issue. The problem was this ‘hasAvailableScreen’ method in the ‘WindowStatusController’ class did not use any member variables or call any methods from WindowStatusController. It instead called 4 methods of the StageStatus class which was being passed to this method as an argument. Moving this method to StageStatus where all of those methods are local made logical sense. Also, since it called no methods or variables of WindowStatusController class the new method doesn’t even need to accept any arguments.

c. Long Method (Manual)
    - Although the original method JDeodorant reported was, in my opinion, not “long” - I still believe breaking it down further was the right choice. This refactoring made each method more descriptive in exactly what it is supposed to do. The other advantage this refactoring operation provides is if a different popup (other than the passwordField) needs to be added in the future we already have method stubs to show popup windows and a popup window at given coordinates. Although these methods are, right now, for the password popup only; we are now in-line to refactor them further to allow more generic method calls by passing more variables to these methods.

d. God Class (Manual)
    - Just as the Long Method changes, I did not think this class was overly long or too much of a God Class to begin with. It was only about 100 lines of code. However, the moved code was all related to a single member variable and accounted for about a third of the class size. Moving this single member variable significantly reduced the size of the original class. It also provided a mechanism that would allow others to use this same ReadOnlyBooleanWrapper property so the other classes no longer have to provide a static initializer that overrides a certain interface. If more analysis of all of the code was done, it would be possible we could refactor many more things to use the new class.

4. Manual code changes (in addition to Eclipse tools)

a. Long Method (Auto)
    - There were no manual code changes strictly required because after Eclipse did its auto refactoring everything was compiling and running correctly. However, with the way Eclipse refactored it called the new ‘getValidator’ method on the first line and assigned it to a local variable. On the next line it used the local variable to set the Validator to a text field. I decided instead to have the call to ‘getValidator’ inside of the ‘setValidator’ call on the next line. This way we do not have to assign a local variable to it and, I believe, it makes the code a little cleaner.
b. Feature Envy (Auto)
    - No manual code changes were required in addition to what Eclipse did. There were no build errors and all tests passed after Eclipse performed its refactoring magic.
c. Long Method (Manual)
    - Entire refactoring done manually.
d. God Class (Manual)
    i. Entire refactoring done manually.