Assignment 4 (CS - 515)

Team 356 - Aaron Pereira and Ryan Becwar

JEdit

Automated Refactoring

1) **Code Smell:** Long Method

   Refactoring step: Extract Method - Extract lines 1441 - 1449 of org.grt.sp.jedit.Buffer into removeMarkerElement(char, int, int)

   Smell description: The method addMarker(char, int) is excessively long, and inside its for loop contains logic which can easily be separated into its own method.

   Why is this code smelly?: The extracted does modify the i variable of the for loop, so i is returned from the method and updated in the loop.

   Automatic Refactoring Explanation: We used Eclipse's Extract Method functionality to extract the code, and set the name of the new method to removeMarkerElement(char, int, int). After verifying that the refactoring did not break the method, we did not have to make an further modifications.

   Test Cases: We had difficulty creating JUnit test cases for this method since we would have to initialize the Buffer class in the unit tests, which was not feasible as it interacted with the user interface and depended on user input. Instead we used the Eclipse Debugger. To test the method, we placed a breakpoint at line 1439, and verified that after each iteration of the for loop the marker elements were removed and the variable 'i' set in the same manner after refactoring as they were before.

   JDeodorant Output : Our changes resulted in the smell in JDeodorant not appearing anymore.

2) **Code smell**: God Class

   Refactoring step: Extract Class - Extract the Flags member, getters and setters and dependent functions and place them in a new class

   Smell description: The class Buffer.java is excessively long. Flag functionality can be extracted and placed in it's own java file and then used as a member of Buffer.java.

Why is this code smelly?: This class is too long and it makes understanding and maintaining this class difficult. Breaking this class down into smaller, inter-related classes would help in understanding and modifying this class in the future.

Automatic Refactoring Explanation: We used Eclipse's Extract class functionality to extract the create a new class with 'flags' as a class member. This also created appropriate getters and setters for this class. We then used the Move method functionality to move related utility methods to the appropriate newly created BufferFlags.java class.

Test Cases: We had difficulty creating JUnit test cases for this method since we would have to initialize the Buffer class in the unit tests, which was not feasible as it interacted with the user interface and depended on user input. Instead we used the Eclipse Debugger. To test the class, we placed 4 breakpoints, one for each new method added. The breakpoints test inputs and outputs to the parent methods of the newly added methods. The following are the inputs and outputs for these methods:

Method name	Input	Output	Comments
org.gjt.sp.jedit.bufferio.BufferLoadRequest._run()	none	none	On being changed, this method still functions exactly the same and does not throw any exceptions. It calls setNewFile()
org.gjt.sp.jedit.buffer.setAutoReload()	true(Boolean)	none	This method successfully sets the flag before and after we made changes to the code. This method also internally called 'isAutoreloadPropertyOverriden()' that returned the correctly set value before and after changes.
org.gjt.sp.jedit.buffer.getAutoReload()	Calling setAutoReload()	true(Boolean)	We manually called this method after calling setAutoReload() and observed if the set value was being returned. It was indeed working as expected before and after the changes

JDeodorant Output : Our changes resulted in the smell in JDeodorant not appearing anymore.

Manual Refactoring

1) **Code Smell**: Feature Envy

Refactoring step: Move Method - Move org.grt.sp.jedit.TextArea.SelectionManager.getSelectionStartAndEnd(int, int, Selection) to org.grt.sp.jedit.TextArea.TextArea

Smell description: The method getSelectionStartAndEnd() makes several calls to the TextArea object contained inside a SelectionManager. Based on the TextArea it returns the Start and End positions of the selection. This method makes multiple calls to TextArea's members and does not make a single call to any of its own members.

Why is this code smelly?: This code is smelly because it seems like the method refers to multiple members of TextArea and does not have a single reference to SelectionManager's members. It would seem that this method should belong to TextArea rather than SelectionManager.

Manual Refactoring Explanation: We manually removed this method from SelectionManager and placed it inside TextArea. We also had to change each reference inside the method where 'TextArea.member' was called to only 'member'. We also had to do a 'File Search' to find out where this method was called and then change each call to reflect the TextArea object and not a SelectionManager. This occurred in 2 files.

Rationale for changes made: As the method used many members of TextArea and no members of SelectionManager, it made sense to perform 'Move Method' and move it from SelectionManager to TextArea.

Test Cases: We faced problems writing JUnit test cases for this smell as it would require multiple objects to be instantiated to test SelectionManager successfully. So what we did instead was, we used the debugger to pause execution at the start of the method 'SelectionManager.insideSelection()' as this is the only place where getSelectionStartAndEnd() was called. We then varied the inputs changing values in the debugger and observed the return values of the method. The results were consistent before and after we made our refactoring changes. We used the same input file opened in the editor for all test cases and also the same selection in the editor.

1) Input(x, y) : (2, 16) | Output : true
2) input(x, y) : (0, 10) | Output : true
3) input(x, y) : (1, 20) | Output : true

The above results were exactly the same before and after we made Refactoring changes. We also ensured the new method was being called by placing a breakpoint inside the newly created method.

JDeodorant Output: Our changes resulted in the smell in JDeodorant not appearing anymore.

2) **Code Smell:** Long Method

Refactoring step: Extract Method - Extract lines 1549-1552 (the logic to remove a markers) of org.grt.sp.jedit.Buffer::removeAllMarkers() to the new method removeAllMarkersLogic().

Explanation: removeAllMarkers sets flags, performs the actual logic to remove markers, and triggers a buffer update. JDeodorant flags this as a long method.

Why is this code smelly?: At 15 lines, this is not a particularly long method. The logic selected for extraction can be easily removed, but it would not be used by other methods, and is likely best kept within removeAllMarkers().

Manual Refactoring Explanation: To perform the refactoring operation, we cut the lines which iterate over the list of markers and call the RemovePosition() method of each marker, then clears the markers list. We then created a new method removeAllMarkersLogic() and pasted these lines inside it. As the new method does not modify local variables, and is not dependent on the values of local variables, it needed no parameters and returns void.

Test Cases: We had difficulty creating JUnit test cases for this method since we would have to initialize the Buffer class in the unit tests, which was not feasible as it interacted with the user interface and depended on user input. Instead we used the Eclipse Debugger. To test the method, we placed a breakpoint at line 1549 before we preformed the refactoring operation. We stepped through the debugger, verified that the markers list was successfully modified, and recorded the contents of the list after the remove logic executed. The list was empty. We then performed the refactoring operation, and repeated the same debugging steps. The results were identical, and our test case passed.

JDeodorant Output: Our changes resulted in the smell in JDeodorant not appearing anymore.

PDFsam

Automated Refactoring

1) **Code smell**: God Class

Refactoring step: Extract Class - Extract the following members from org.pdfsam.ui.ContentPane (pdfsam-gui) :

- public VBox newsContainer
- public FadeTransition fadeIn
- public FadeTransition fadeOut
- public void onShowNewsPanel()
- public void onHideNewsPanel()

We placed these in an extracted class that we named ‘ContentPaneData’.

Smell description: The class ContentPane.java is excessively long. VBox and its transitions did not seem to have a logical connection in terms of structure to ContentPane. It would make sense to create a separate class for this data and include it as a member in ContentPane.java

Why is this code smelly?: This class is too long and it makes understanding and maintaining this class difficult. Breaking this class down into smaller, inter-related classes would help in understanding and modifying this class in the future.

Automatic Refactoring Explanation: We used Eclipse’s Extract class functionality to extract the create a new class with ‘flags’ as a class member. This also created appropriate getters and setters for this class. We then used the Move method functionality to move related utility methods to the appropriate newly created ContentPaneData.java class.

Test Cases: We initially did a project search for all the members moved from the original class to the newly extracted class. We observed that all calls to these members were made on creation of a ‘ContentPane’. Hence it would have seemed logical to test this class. However, a fully covered test case was already present for this class. The following was the package directory for this class.

pdfsam-gui/src/test/java/org/pdfsam/ui/ContentPaneTest.java

We ran this test before and after we made our changes and confirmed that this test passed both before and after we made our changes.

JDeodorant Output: Our changes resulted in the smell in JDeodorant not appearing anymore.

2) **Code Smell**: Long Method

Refactoring step: Extract Method - For this refactoring, we modified our own code for Assignment 2. We created an excessively long method to remove page intersections in the Merge Module. We split this method up into two more understandable and maintainable sections of code.

Smell description: The method addInput is a long method at 47 lines, and contains logic which can easily be extracted into two new methods. JDeodorant flagged lines multiple lines of this method as acceptable for method extraction.

Why is this code smelly?: This code is smelly because it is excessively long and can easily be split into multiple methods of a smaller size, which have little logical dependency on each other.

Automatic Refactoring Explanation: We called 'Extract Method' two times on different snippets of code. The long code was logically broken down into two parts. One that fetched unique pages (getUniquePages()) , and the other that got rid of intersections from these pages (removePageIntersections()).

Rationale for changes made: Performing this refactoring operation reduced the size of the initial method as well as made the code understandable in terms of what exactly the snippets of code are doing.

Test Cases: PDFsam includes tests for the MergeParametersBuilder class in pdfsam-merge:org.pdfsam.merge.MergeParametersBuilderTest. This test case instantiates a MergeParametersBuilder object with various input parameters. This includes passing different page ranges into the builder. This in turn, tests our addInputs() method.

JDeodorant Output: Our changes resulted in the smell in JDeodorant not appearing anymore. But the newly created method 'removePageIntersections()' pops up as a long method as it can further be broken down into smaller methods itself. The initial smell on 'addInputs' is not present anymore.

Manual Refactoring

1) **Code Smell**: Long Method

Refactoring step: Extract Method - Extract lines 264-269 of pdfsam-fx:org.pdfsam.ui.selection.single.SingleSelectionPane::initContextMenu() to a new method SingleSelectionPane::getInfoItem().

Smell description: The method initContextMenu is a long method at 31 lines, and contains logic which can easily be extracted into a new method. JDeodorant flagged lines 264-269 as the best candidates for extraction.

Why is this code smelly?: This code is smelly because it is excessively long and can easily be split into multiple methods of a smaller size, which have little logical dependency on each other.

Manual Refactoring Explanation: We created a new method getInfoItem() by extracting lines 264-269 of initContextMenu(). The only interaction these lines have with local variables is to set the variable MenuItem infoItem, so this method needs only to return a MenuItem variable and requires no arguments. We did not use the refactoring in Eclipse for this operation. The only additional changes necessary were to return the updated infoItem variable, and set it in initContextMenu() to the returned value.

Rationale for changes made: We performed this operation as the method could easily be extracted, and reduced the complexity of initContextMenu().

Test Cases: PDFsam includes tests for the SingleSelectionPane class in pdfsam-fx:org.pdfsam.ui.selection.single.SingleSelectionPaneTest. The initContextMenu() method is called is called within the constructor SingleSelectionPane(), which is in turn called by the start(Stage) method which sets up all unit tests. Therefore, the method is tested by each unit test in the Class, as it is crucial in setting up the SingleSelectionPaneTest. We ran these tests before refactoring, observed that they all passed, performed our refactoring, reran the tests, and observed that they still passed.

2) **Code Smell**: God Class

Refactoring step: We created a new class called SelectionTableData.java and moved the following members into it :

- public Consumer<SelectionChangedEvent> selectionChangedConsumer
- void initTopSectionContextMenu(SelectionTable selectionTable, ContextMenu contextMenu, boolean hasRanges)
- void initItemsSectionContextMenu(SelectionTable selectionTable, ContextMenu contextMenu, boolean canDuplicate, boolean canMove)
- void initBottomSectionContextMenu(SelectionTable selectionTable, ContextMenu contextMenu)

We then did file searches for any occurance of calls to these members. We found 3 matches, all calls to the methods that internally use the selectionChangedConsumer that we moved to the new class. All of these occur in the constructor of SelectionTable.

Smell description: The class getSelectionStartAndEnd() is excessively long. selectionChangedConsumer and its accompanying methods did not seem to have an logical connection in terms of structure to SelectionTable. It would make sense to create a separate class for this data and include it as a member in SelectionTable.java

Why is this code smelly?: This class is too long and it makes understanding and maintaining this class difficult. Breaking this class down into smaller, inter-related classes would help in understanding and modifying this class in the future.

Manual Refactoring Explanation: We explained our steps to perform manual refactoring in the **Refactoring step** above.

Rationale for changes made: As these members were not logically related directly to SelectionTable, it would seem better to break down the big 'SelectionTable' down into two smaller classes. Place all the functionality related to selectionChangedConsumer inside the new class and then change any calls to these methods to reflect the new method signature.

Test Cases: Our changes directly affected only the constructor of SelectionTable.java. It would make sense to test the construction of the SelectionTable object. Fortunately, there already existed a test case, that did exactly this. he following was the package directory for this class.

pdfsam-fx/src/test/java/org/pdfsam/ui/selection/multiple/SelectionTableTest.java

This test case tests the creation of a SelectionTable object as well as all operations on it. Our changes affected the constructor. So if we passed the creation of the object, it should be enough validation that our changes did not break anything.

This test case passed successfully before and after we made out changes.

JDeodorant Output: Our changes resulted in the smell in JDeodorant not appearing anymore.

Manual Refactoring vs Automatic Refactoring

- Automatic refactoring was easier most of the time.
- Sometimes, automatic refactoring resulted in build errors because it could not understand the java-reflection dependencies fully.
- Manual refactoring took more time
- Manual refactoring needed us to do searches to find what part of the code would be affected by the refactoring.
- Automatic refactoring led to cleaner code.

- Automatic refactoring took way lesser time than manual refactoring.