Suraj Eswaran
CS515 SOFTWARE MAINTAINANCE AND EVOLUTION
ASSIGNMENT 4

Code Smells & Refactoring-JEdit

Refactoring is the process of changing a software code in such a way that it does not affect the behavior of the code and improves its internal structure. A good design always comes first followed by the coding. Over the time code will be modified and the integrity of the system structure according to that design would gradually fades. Thus, the code sinks from engineering to hacking.

While refactoring the code , we can take a bad design and rework it into a well-designed code. The resulting interaction leads to a program with a design that stays good as development continues.

There are several tools to automatically detect code smells. The tool that I have used for this assignment is JDeodorant. JDeodorant is an Eclipse plug-in that identifies design problems in software, known as bad smells, and resolves them by applying appropriate refactoring.

JDeodorant employs a variety of novel methods and techniques in order to identify code smells and suggest the appropriate refraction that resolve them. For the moment, the tool identifies five kinds of bad smells, namely Feature Envy, Type Checking, Long Method, God Class and Duplicated Code.

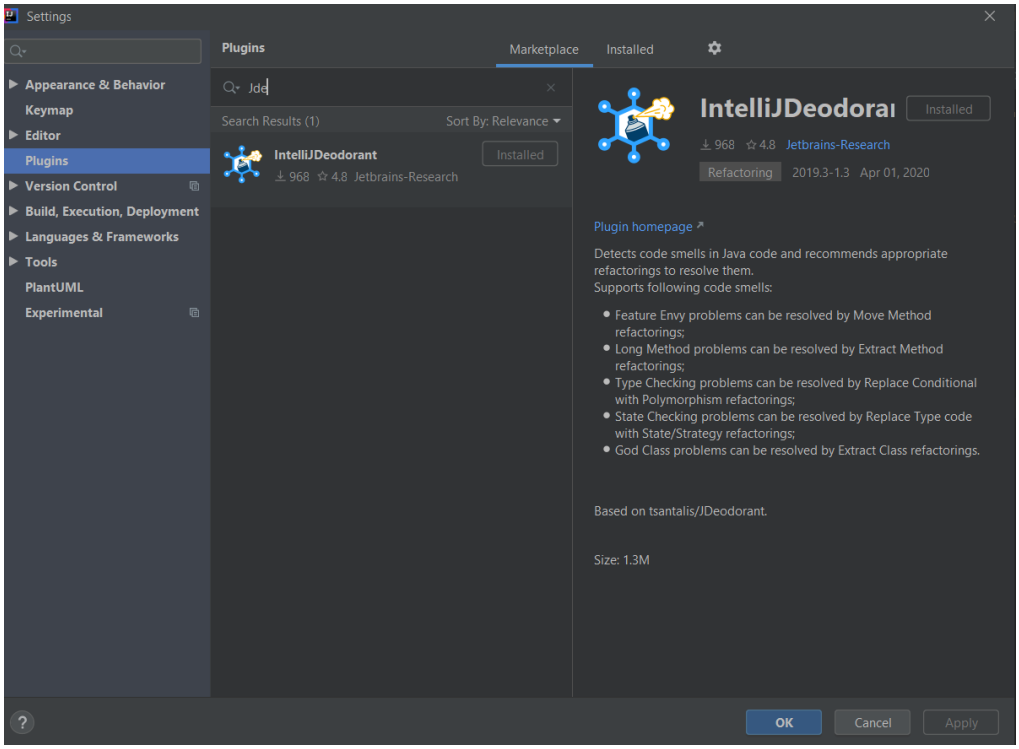

Figure 1: JDeodorant Plugin in Intellij

Intellij JDeodorant is a plugin which is based on Jdeodorant Eclipse plugin that detects code smells in Java code and recommends appropriate refactorings to resolve them. All of the suggested refactorings can be carried out automatically from within the plugin.

This tool supports several code smells. They are:

1. *Feature Envy: It performs a "Move Method"* when a method uses attributes/methods of another class more than those of the enclosing class. The tool can detect such methods and suggest moving them to a more related class.
2. *Type Checking follows a "Replace Conditional with Polymorphism" refactoring where it refers to cases when a set of conditional statements determine the outcome of the program by comparing the value of a variable representing the current state of an object with a set of named constants.*
3. *Long Method occurs when a method is too long and can be divided into several. For such methods, the tool identifies blocks of code that are responsible for calculating a variable and suggests extracting it into a separate method, i.e. perform an Extract Method refactoring.*
4. *God Class is done on large and complex class that contains too many components. The tool identifies sets of attributes and methods in a class that could be moved into a separate class to simplify the understanding of the code, i.e. an Extract Class refactoring can be performed.*

To run the tool:

- *After the plugin is installed , IntelliJDeodorant tool will appear below in IntelliJ IDEA.*

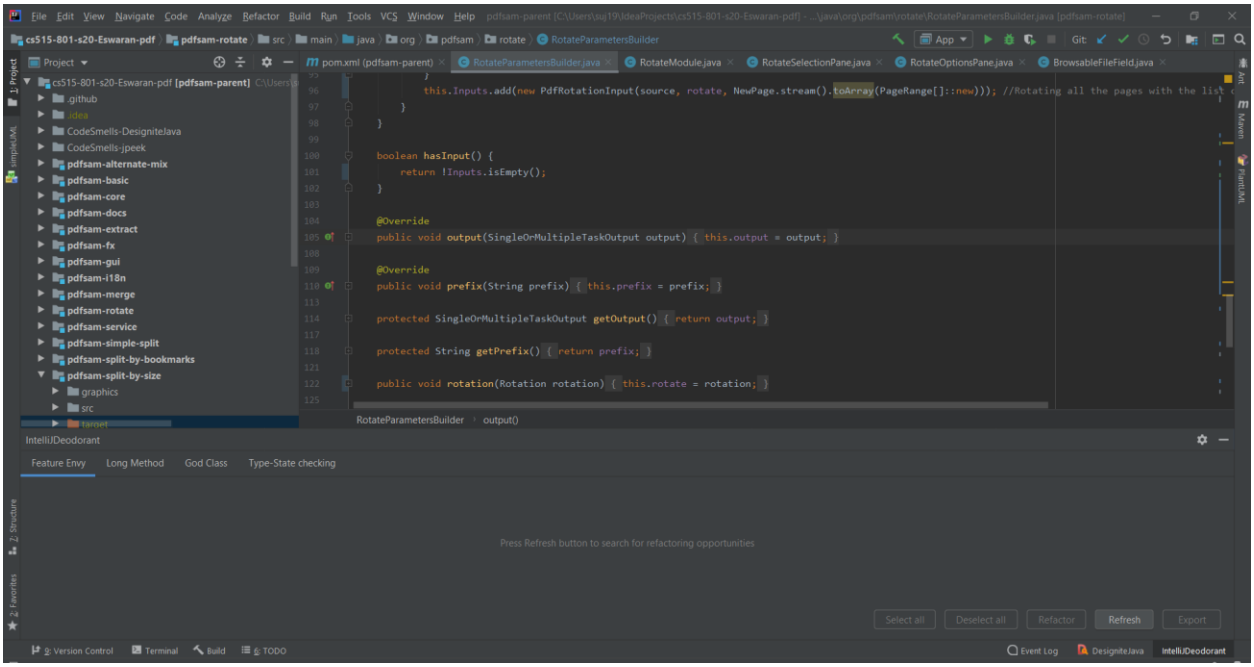

Figure 2: Preview of IntelliJ JDeodorant

- *Press Refresh button to search for refactoring opportunities.*
- *Each tab of this window contains a Refresh button that allows to search for the necessary code smell in the entire project and the table with the results of the search.*
- *For refactoring, select a suggestion in the table and click the Refactor button which is beside Refresh button.*

I. AUTOMATED REFACTORING

a. *Class :gjt/sp/jedit/options/GutterOptionPane.java*
Smell Type: God Class, Method: Extract Class

Rationale: The main for this code smell refactoring is to extract classes into other class.

<u>Before Refactoring</u>

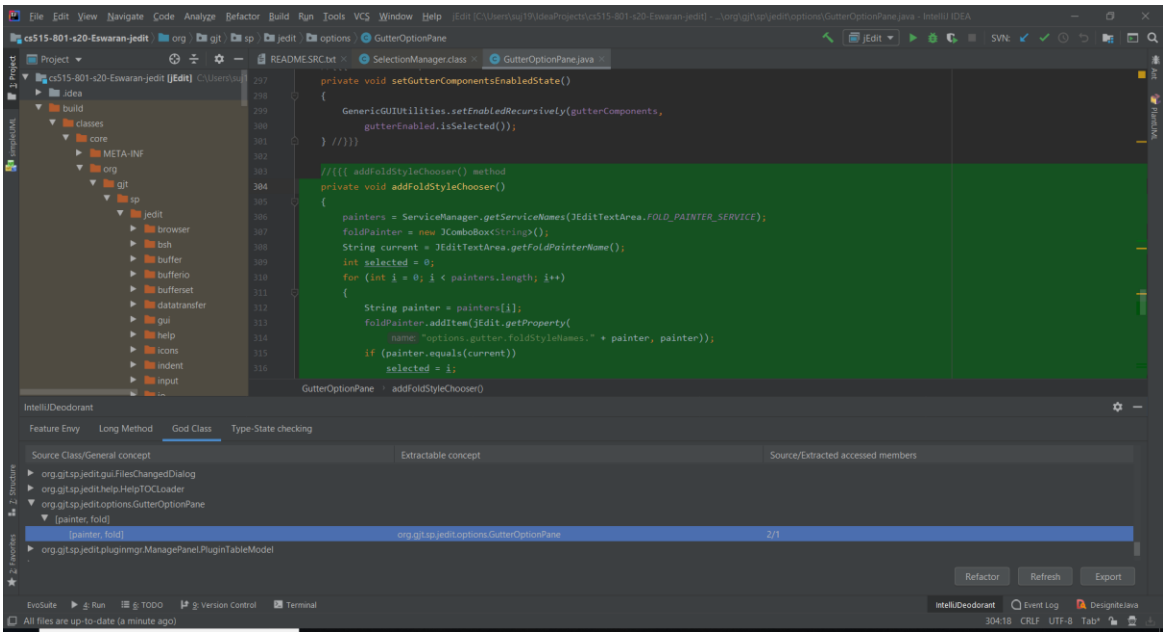

Figure 3: Detection of Code smell

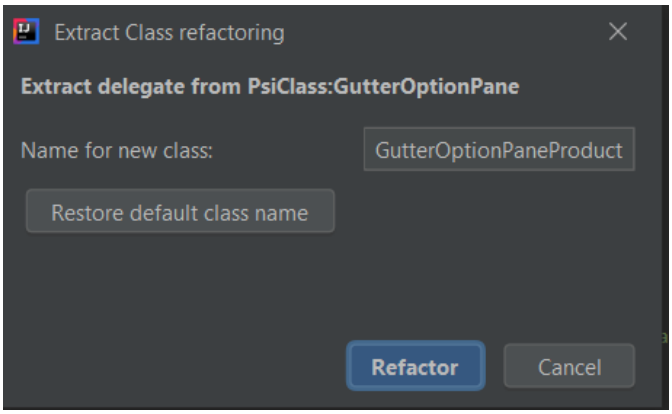

Figure 4: Extract Class Refactoring

After Refactoring

For removing the smell from God class GutterOptionPane, A class GutterOptionPaneProduct was created and fields are extracted from GutterOptionPane.

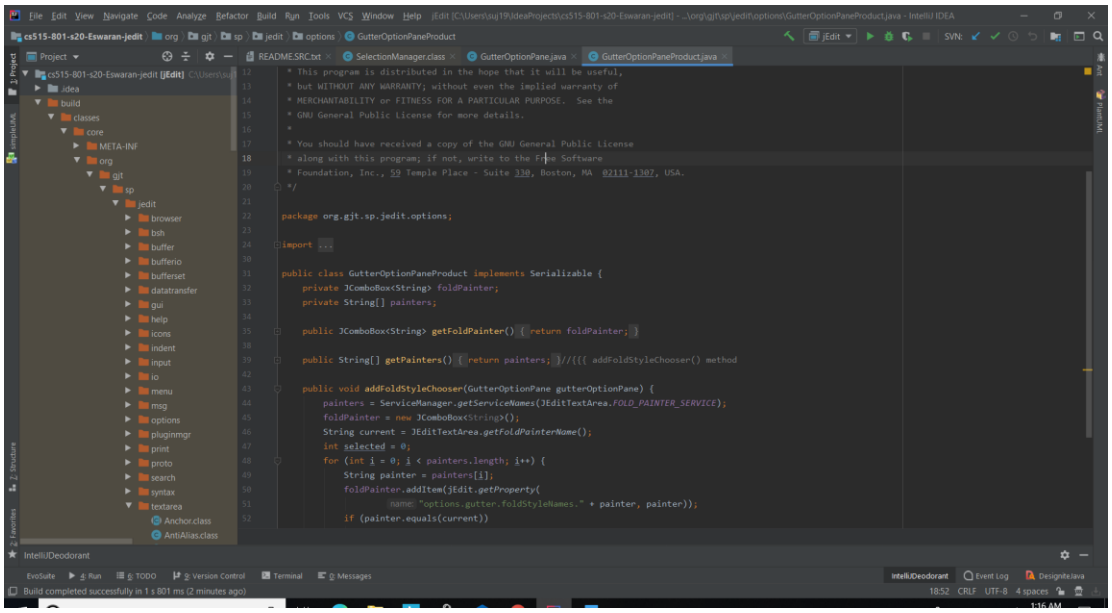

Figure 5: Extraction of fields in GutterOptionPaneProduct.java

There were not any changes in the code, just extraction was done. After refactoring is done, the smell is not detected, thus the smell is removed with the help of JDeodorant.

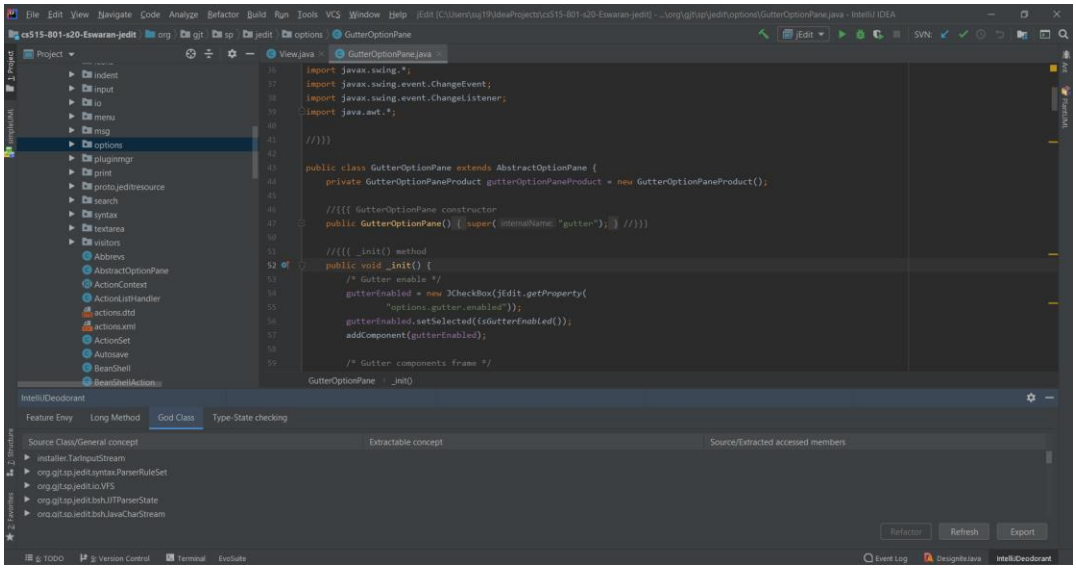

Figure 6: Detection of no smell from GutterOptionPane.java

Testing is done with the help of Junit testing. All the test were passed before and after refactoring.

b. *Class: gjt/sp/jedit/gui/MarkerViewer.java*
Smell Type: Feature Envy, Method: Move Method

Rationale: The main for this code smell refactoring is to move the methods method of one class used by another class .

<u>Before Refactoring</u>

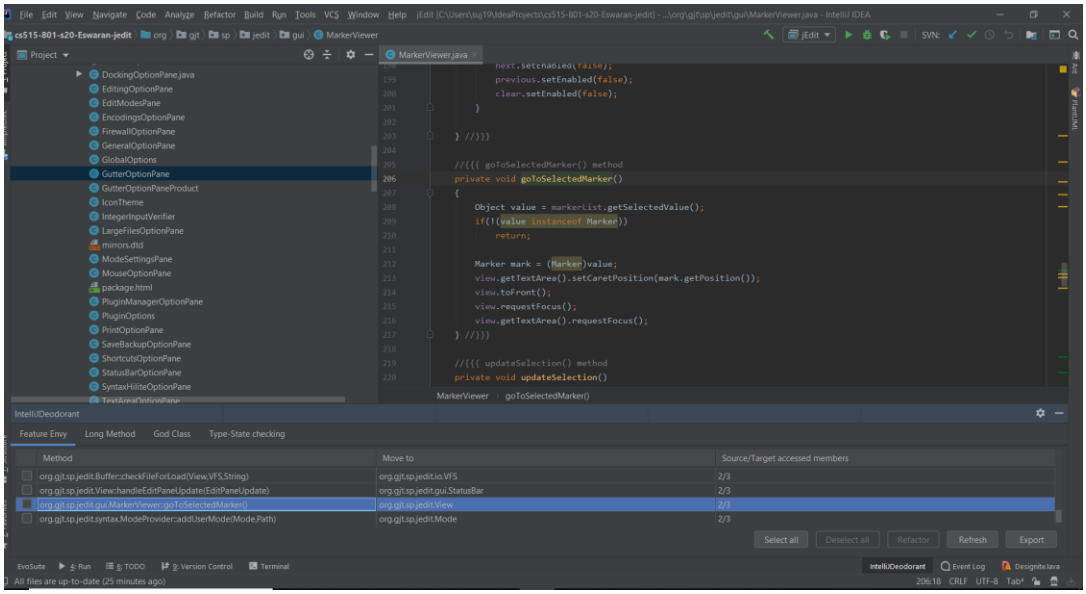

Figure 8: Detection of Code smell from MakerViewer.java

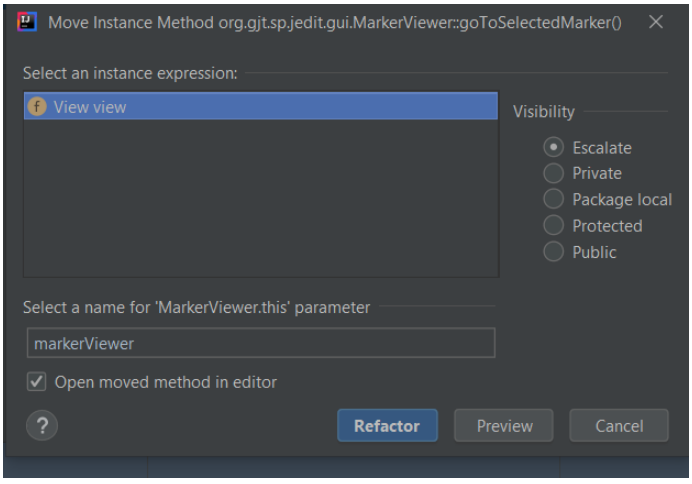

Figure 9: Move Method

<u>After Refactoring</u>

For removing the smell from Feature Envy MarkerViewer.java, the features are moved to View.java.

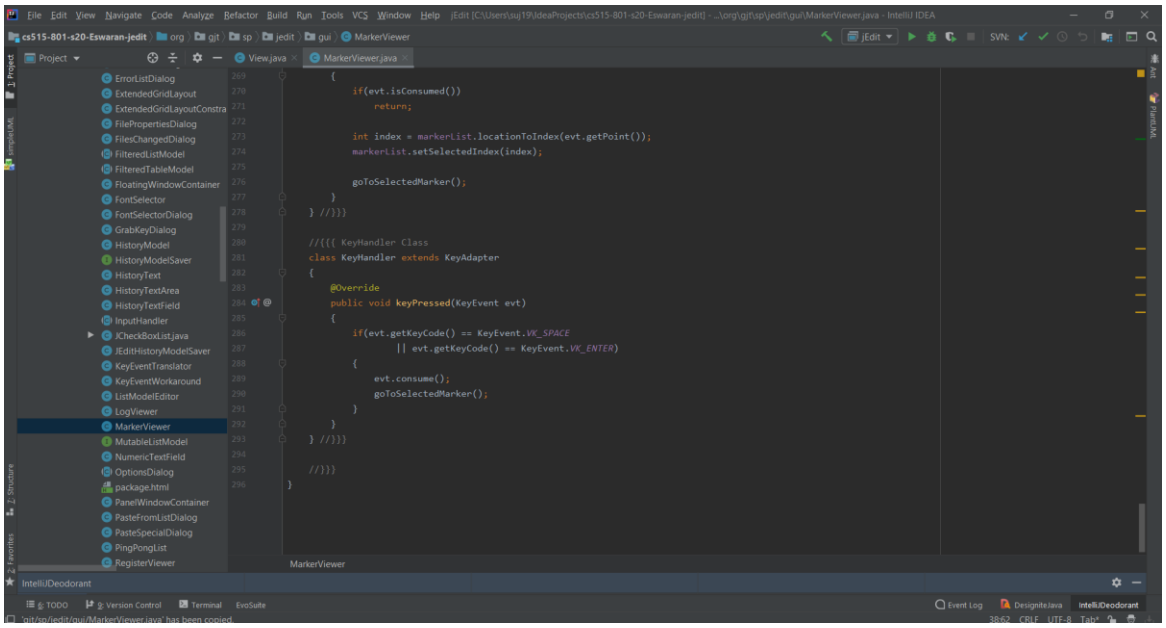

Figure 10: Preview of View.java and MarkerViewer.java

As there were not any code change instead of just moving the field to another class. After refactoring is done, the smell is not detected, thus the smell is removed with the help of JDeodorant.

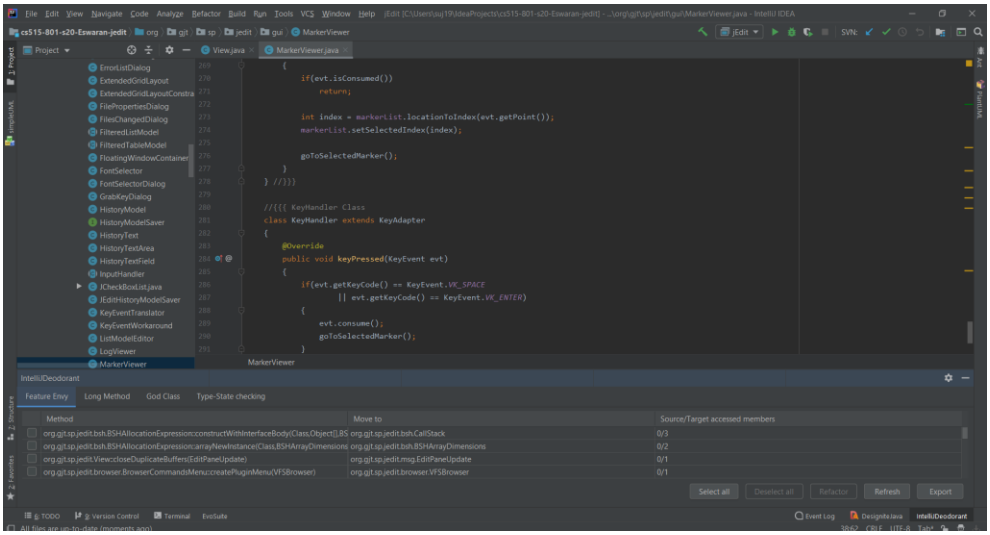

Figure 11: Detection of no Code Smells in MarkerViewer.java

Since there are not any test cases in Jedit, so it was necessary to use Randoop[1] tool for generating test cases. The command for Randoop is:

```
$ java -classpath %RANDOOP_JAR%;build/classes/core randoop.main.Main gentests --
testclass=<classname> --output-limit=100 --junit-outp

ut-dir=./<destination>
```

Testing is done with the help of Junit testing. All the test were passed before and after refactoring.

1. *The program ran successfully before and after refactoring.*
2. *The main focus was to check the fontProperty value. So, the test involves whether the fontProperty value is equal to 12 and "PLAIN". The test case passed before and after refactoring.*

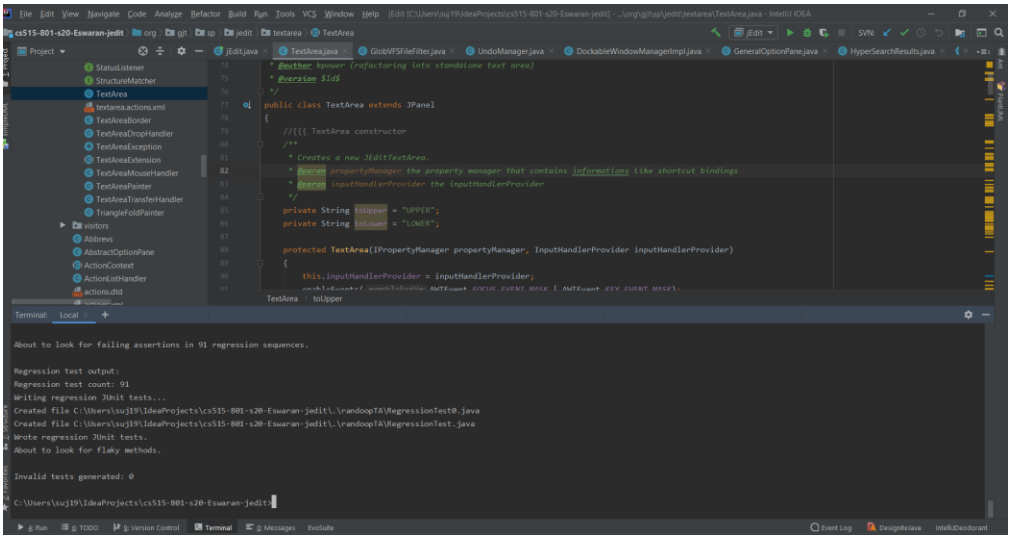

Figure 12: Test Cases generated with the help of Randoop

Thus, the changes in the code did not affect the functionality.

c. *Class: gjt/sp/jedit/textArea/TextArea.java*
Smell Type: Feature Envy, Method: Move Method
Destination Class: gjt/sp/jedit/buffer/JeditBuffer
Rationale: The main for this code smell refactoring is to move the methods method of one class used by another class .

<u>Before Refactoring</u>

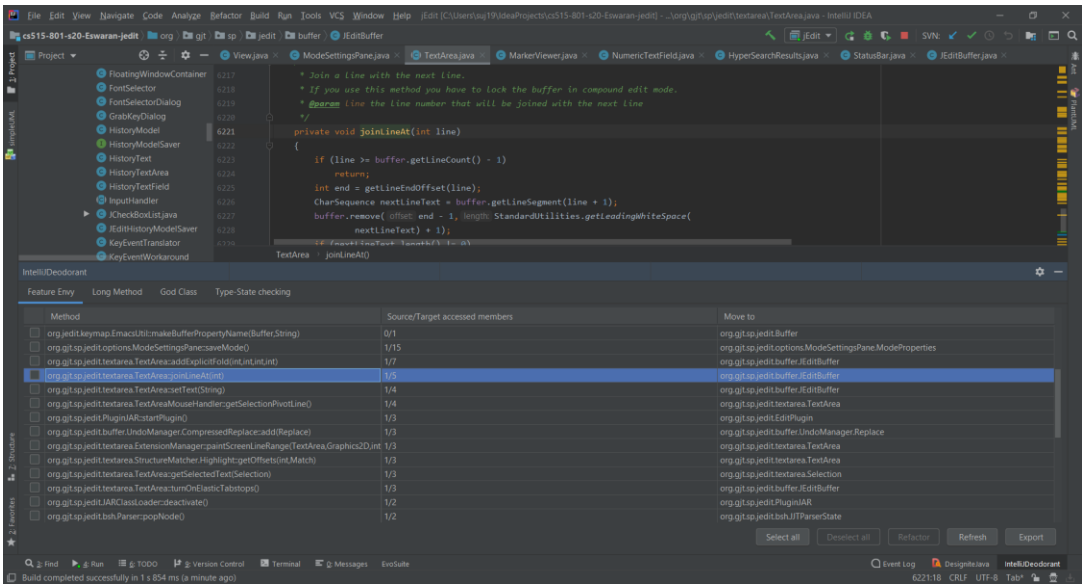

Figure 13: Detection of Code smell from TextArea.java

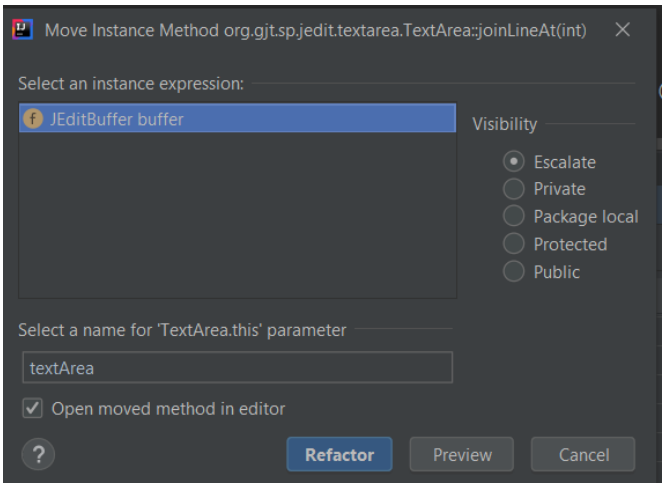

Figure 14: Move Method

After Refactoring

For removing the smell from Feature Envy TextArea.java, the features are moved to JeditBuffer.java.

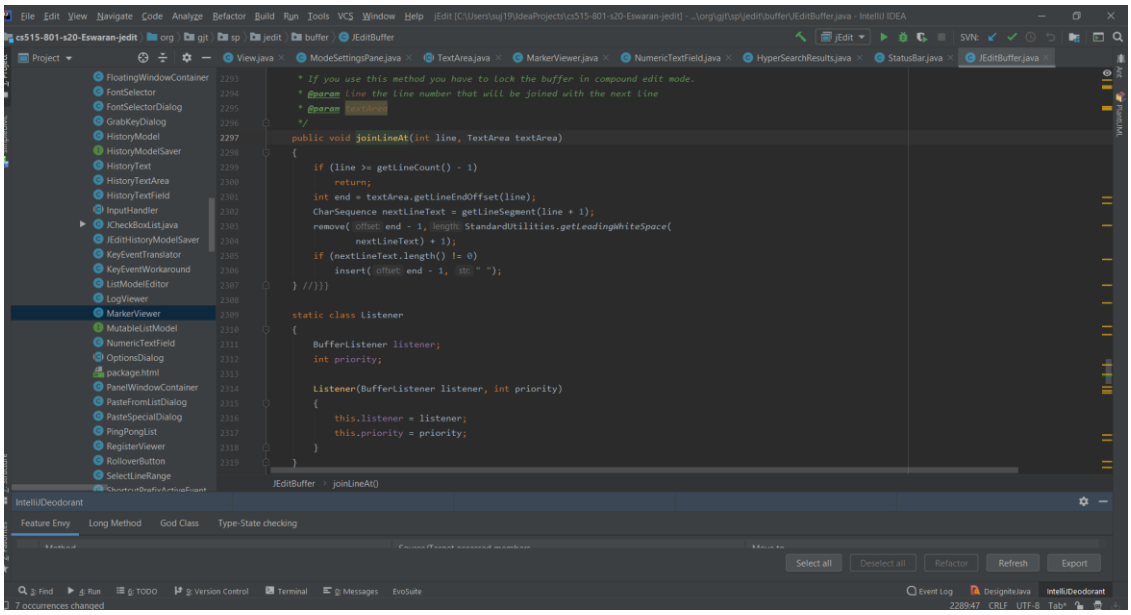

Figure 15: Preview of TextArea.java and JeditBuffer.java

As there were not any code change instead of just moving the field to another class. After refactoring is done, the smell is not detected, thus the smell is removed with the help of JDeodorant.

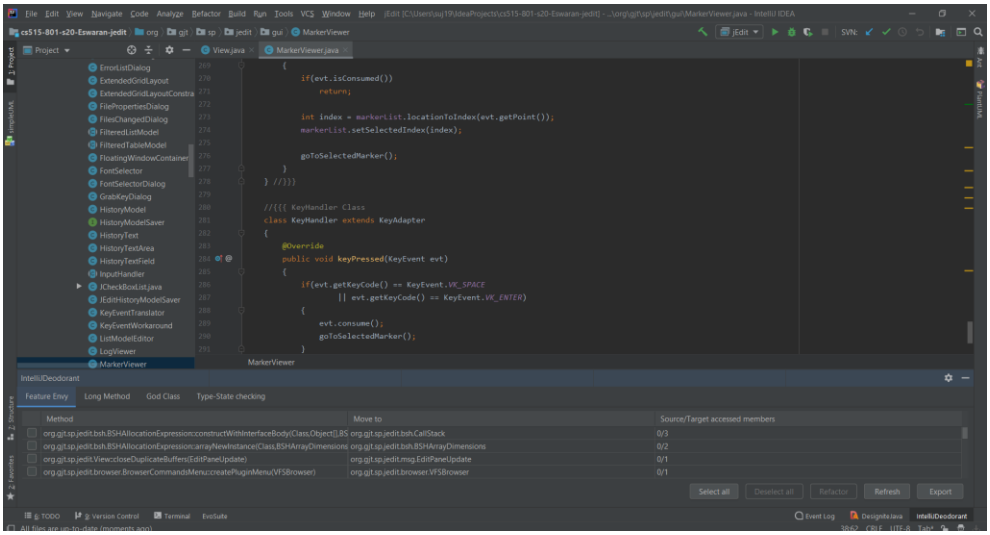

Figure 16: Detection of no Code Smells in JeditBuffer.java

Since there are not any test cases in Jedit, so it was necessary to use Randoop tool for generating test cases. The command for Randoop is:

```
$ java -classpath %RANDOOP_JAR%;build/classes/core randoop.main.Main gentests --
testclass=<classname> --output-limit=100 --junit-outp

ut-dir=./<destination>
```

Testing is done with the help of Junit testing. All the test were passed before and after refactoring.

1. *The program ran successfully before and after refactoring.*
2. *Checked whether JoinLineAt() value is at 10 or not, thus it has worked in before and after refactoring.*

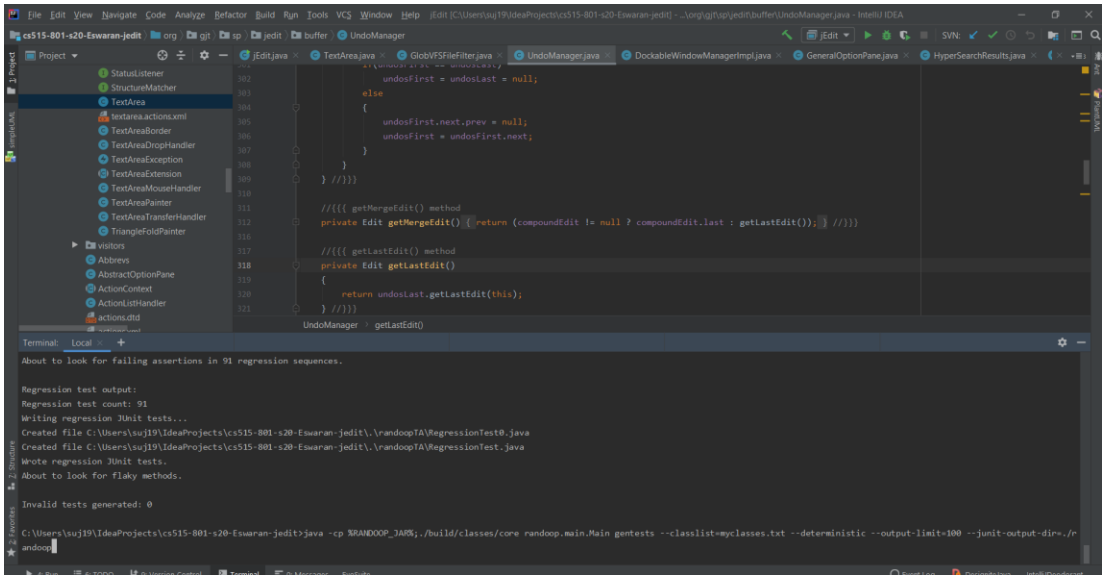

Figure 17: Test Cases generated with the help of Randoop

Thus, the changes in the code did not affect the functionality.

d. *Class : gjt/sp/jedit/search/HyperSearchResults.java*
Smell Type: God Class, Method: Extract Class
Rationale: The main for this code smell refactoring is to extract classes into other class.

<u>Before Refactoring</u>

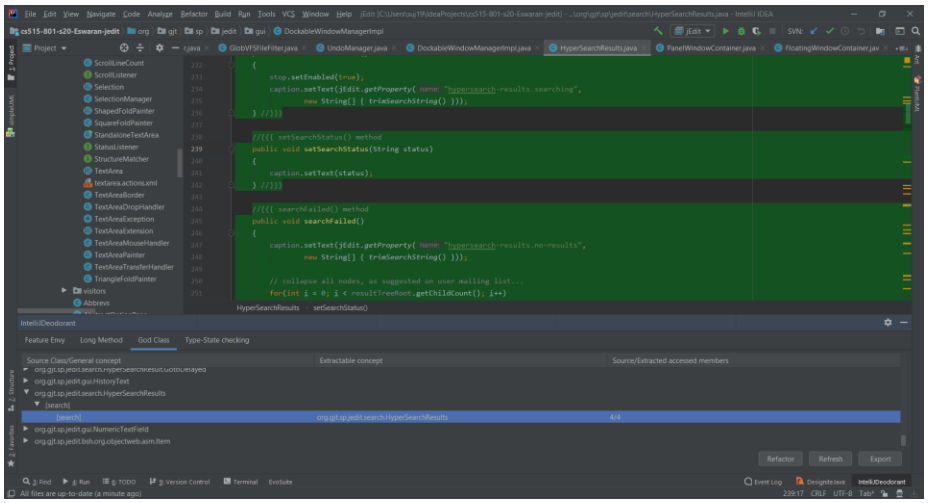

Figure 18: Detection of Code smell in HyperSearchResults.java

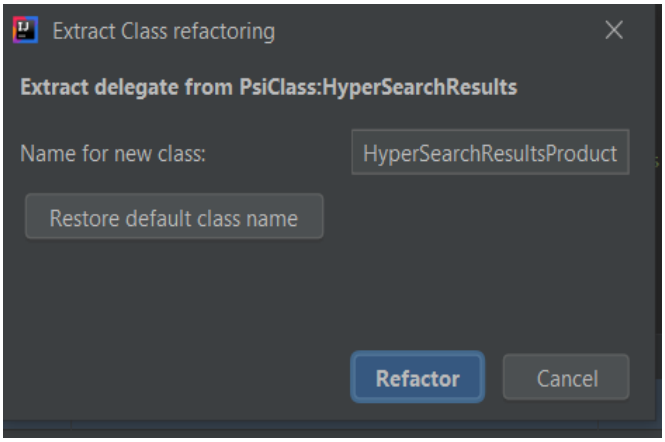

Figure 19: Extract Class Refactoring

<u>After Refactoring</u>

For removing the smell from God class HyperSearchResults, A class HyperSearchResultsProduct was created and fields are extracted from HyperSearchResults.

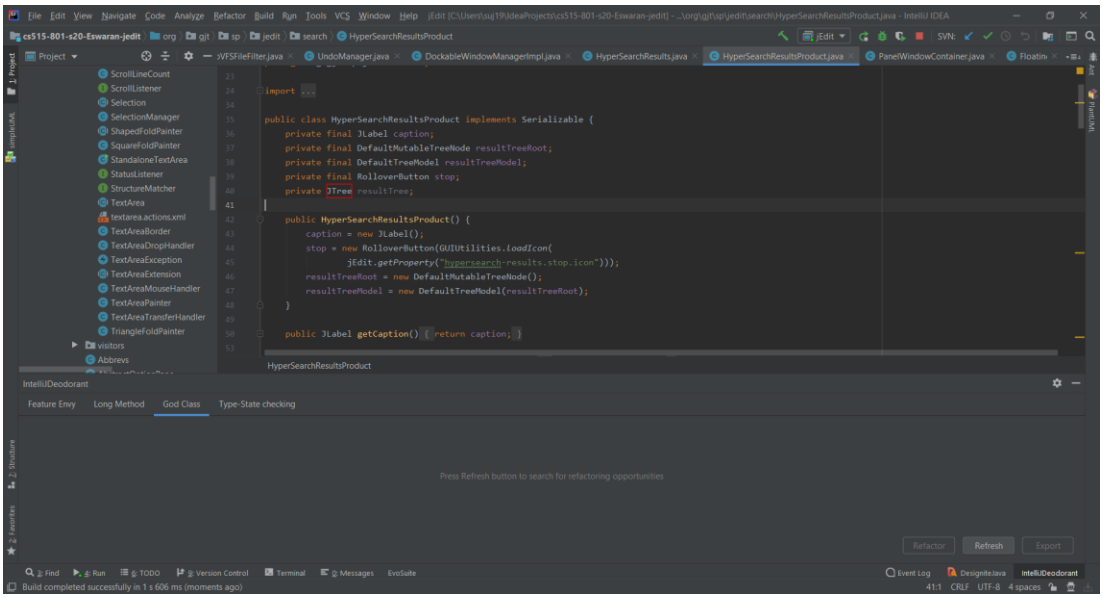

Figure 20: Extraction of fields in HyperSearchResultsProduct.java

There were not any changes in the code, just extraction was done. After refactoring is done, the smell is not detected, thus the smell is removed with the help of JDeodorant.

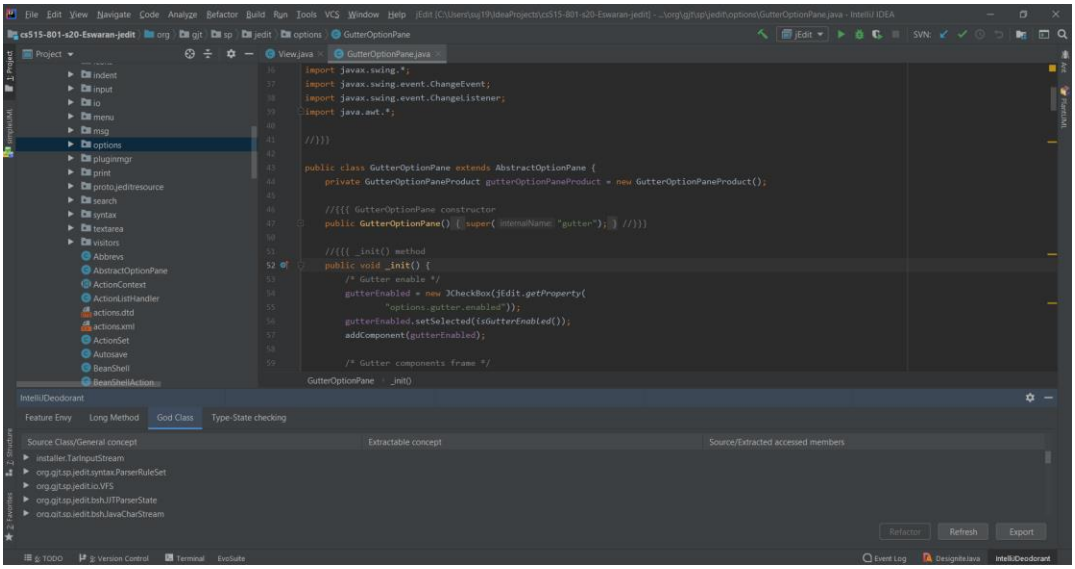

Figure 21: Detection of no smell from HyperSearchResults.java

Testing is done with the help of Junit testing. All the test were passed before and after refactoring.

Since there are not any test cases in Jedit, so it was necessary to use Randoop tool for generating test cases. The command for Randoop is:

```
$ java -classpath %RANDOOP_JAR%;build/classes/core randoop.main.Main gentests --
testclass=<classname> --output-limit=100 --junit-outp

ut-dir=./<destination>
```

Testing is done with the help of Junit testing. All the test were passed before and after refactoring.

1. *The program ran successfully before and after refactoring.*
2. *Check the trimString value returns a value of 10 since the test case involves a ten-letter word. So, the test involves whether the trimStringvalue value is equal to 10 or not. The test case passed before and after refactoring.*

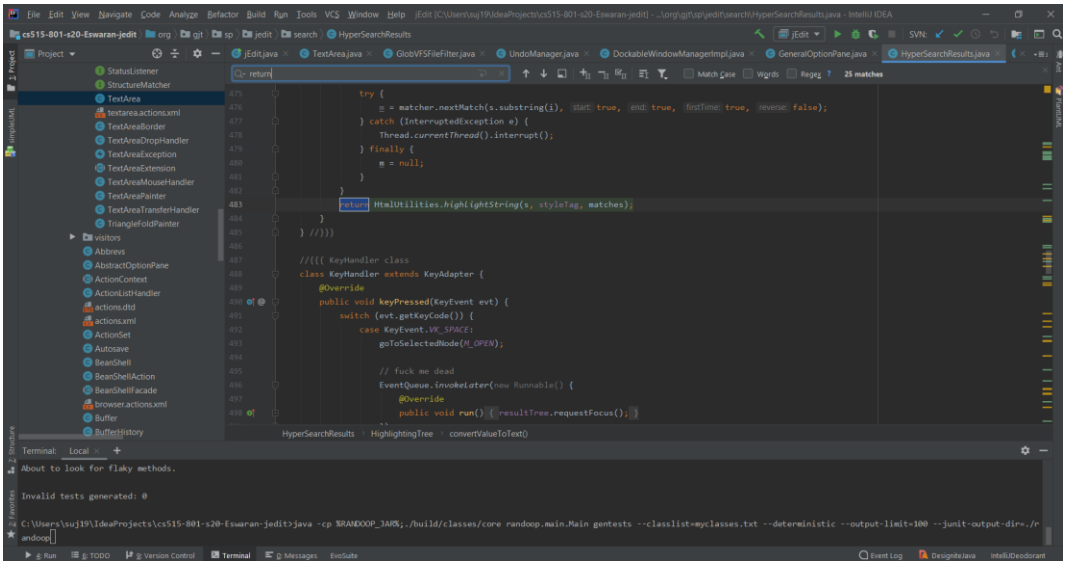

Figure 22: Test Cases generated with the help of Randoop

Thus, the changes in the code did not affect the functionality.

II. MANUAL REFACTORING

a. *Class: org/gjt/sp/jedit/textarea/TextArea.java*
Rationale: The main idea for this manual refracting is to find some bad smells like duplication within the projects.

Before Refactoring

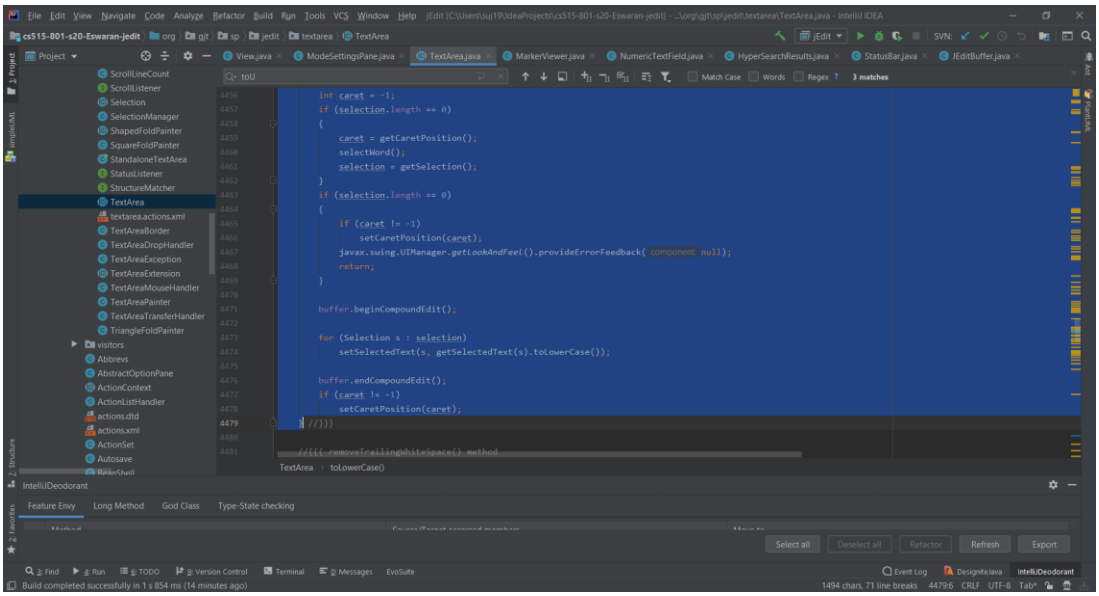

Figure 18: Detection of Code Smells in TextAre.java

After Refactoring

There were few code changes of combining both the functions by including if-else statement in it .After refactoring is done, the smell is not detected, thus the smell is removed manually.

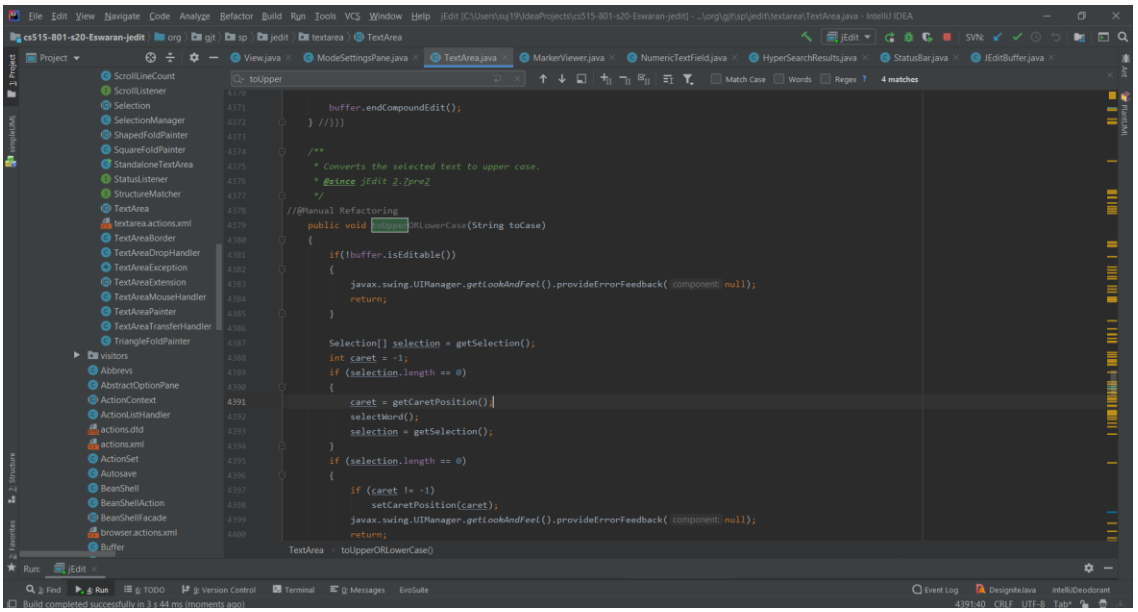

Figure 19: Removal of Code Smells in TextAre.java

Testing is done with the help of Junit testing. All the test were passed before and after refactoring.

Since there are not any test cases in Jedit, so it was necessary to use Randoop tool for generating test cases. The command for Randoop is:

```
$ java -classpath %RANDOOP_JAR%;build/classes/core randoop.main.Main gentests --
testclass=<classname>--output-limit=100 --junit-outp

ut-dir=./<destination>
```

Testing is done with the help of Junit testing. All the test were passed before and after refactoring.

1. *The program ran successfully before and after refactoring.*
2. *Checked whether isCaretBlinkEnabled () showing caretBlinks value or not, thus it has worked in before and after refactoring.*

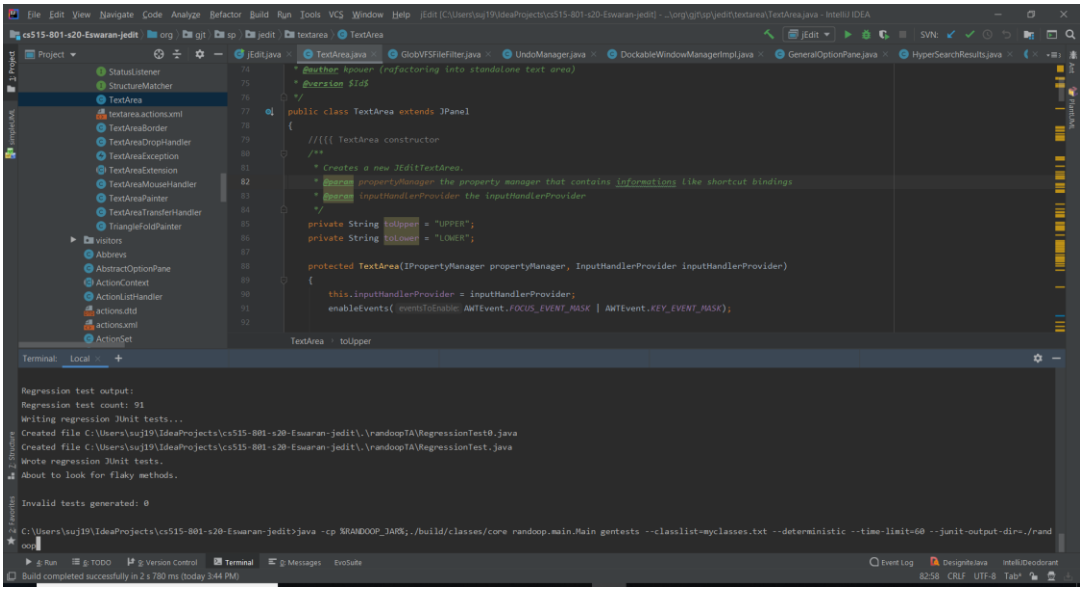

Figure 23: Test Cases generated with the help of Randoop

Thus, the changes in the code did not affect the functionality.

Type of refactoring	*Smelly Classes*	Briefly describe the smell by considering the class, methods, attributes, etc. involved in the smell	Explain why the class/method is flagged as smelly (be specific).	Do you agree that the detected smell is an actual smell? Justify your answer.
Automatic	*GutterOptionPane*	*Here, functionalities of class GutterOptionPanel doing all the function operation,so it should be separated into other different classes based on similar type of responsibilities .*	*Since addFoldStyleChooser () seem to do all the functionality of GutterOptionPanel, so this method would be considered as a code smell. Solution for this God class problem is to extract the class to a new default class named GutterOptionPaneProduct after refactoring.*	*According to me, this code smell can be an actual smell since it is a difficult task for the , programmers to create a new class for the feature. It can be solved by placing a new feature in an existing class which can be helpful in design aspect as well. The changed code did*

				not affect the functionality.
	MarkerViewer	*Here, the class uses a method from another class for its functionality, which is also a code smell.Hence, it would be a good motive to move methods to View class.*	*Fields like MarkerList is been used in both the class so one class uses that value from another class which does not have a proper design.Hence, it is detected as a code smell.*	*Yes, I agree to the fact that this detected smell is an actual smell since usage of fields from one class to another can often violate the code, thus it has to be considered as a code smell. The changed code did not affect the functionality*
	TextArea	*As same as MarkerViewer, the class uses a method from another class for its functionality, which is also a code smell. Hence, it would*	*joinLinaAt() has been used in both the class so one class uses that value from another class which does not have a proper design. Hence, it is detected as a code smell.*	*Same as MarkerViewer, this is also an actual smell since usage of fields from one class to another*

		be a good motive to move methods to JeditBuffer class.		*can often violate the code, thus it has to be considered as a code smell. The changed code did not affect the functionality*
	HyperSearchResults	*Functionalities of class HyperSearchResults doing all the function operation, thus it can be separated into other different classes based on similar type of responsibilities.*	*In the case of HyperSearchResults, Status seem to be used a lot, so this method would be considered as a code smell. Solution for this God class problem is to extract the class to a new default class named HyperSearchResults Product after refactoring.*	*Even this code smell can be an actual smell since it is a difficult task for the, programmers to create a new class for the feature rather than it can be solved by placing a new feature in an existing class which can be helpful in design*

				aspect as well. The changed code did not affect the functionality.
Manual	*TextArea*	*This kind of code smell seems to have duplication since both functions toUpperCase() and toLowerCase() seems to provide same code with different method. It would avoid unnecessary smells by adding if else statement in it.*	*Both toUpperCase() and tolowerCase() seem to provide the same code structure with different functionality, This is the reason why it is flagged as a smelly.*	*Even though it was a manual refactoring, I think this detected smell is an actual smell since it creates a smelly code which would spoil the Design of Existing Code . The changed code did not affect the functionality.*