CS 515 Spring 2020
Jared Crouse
Yashwanth Virupaksha

Assignment 4: Code Smells & Refactoring

3 Detecting and analyzing code smells

jEdit code smell 1:

The first code smell that we identified is in the jEdit.buffer package, JEditBuffer.java file, and the class is **JEditBuffer**. The code smell that we identified is a **bloater** having a **long class**.

What is a long class? It's a class having multiple responsibilities.

Why is this class flagged as smelly? The PMD tool classified this class as smelly giving the reason that it has "ExcessiveClassLength"

Do you agree that the detected smell is correct? Yes, we agree that this class is smelly because it holds a lot of responsibilities in terms of Constants, Constructors, and methods.

Here's more of the supporting information on why this class is smelly:

The number of lines in this class is 3051. There are many methods and class variables that can make it a good example of a long class.

jEdit code smell 2:

The second code smell that we identified is in the jEdit.options package, StatusBarOptionPane.java file, StatusBarOptionPane class, and the method is _init(). The code smell that we identified is a **bloater** having a **long method**.

What is a long method? It's a method having too many lines of code.

Why is this class flagged as smelly? The PMD tool classified this class as smelly giving the reason that it has "ExcessiveMethodLength"

Do you agree that the detected smell is correct? Yes, we agree that this class is smelly because the number of lines exceeds 10 (rule of thumb).

Here's more of the supporting information on why this class is smelly:

This method has 156 lines which make it a good example of a large method.

jEdit code smell 3:

The third code smell that we identified is a **Dispensable** having a **lazy class.** The code smell is present in the jEdit.bufferset package, BufferSetAdapter.java file, and the name of the class is BufferSetAdapter. We were able to detect this lazy class using the PMD tool. The PMD tool has flagged this class with multiple tags. These tags are MissingOverride, AtLeastOneConstructor, and UncommentedEmptyMethodBody. We were suspicious looking at the tags so we took a closer look at the class and found that there are only empty methods in this class. Apart from looking at the class we also did a workspace wide search to look if there's any class that's extending this class. The result of this search was that we couldn't find any class that extends this class. Therefore, we identify it as a lazy class not providing support for the other classes and no method implementations in itself.

What is a lazy class? It's a class that does not do enough.

Why is this class flagged as smelly? This class is flagged as smelly because it has empty methods, there's no class that inherits this class, and finally, the methods are missing overrides which makes sense because the project has no classes that extend this class.

Do you agree that the detected smell is correct? Yes, we agree that the class is a lazy class with no support for the other classes and consists of empty methods.

PDFsam code smell 1:

The first code smell we found for PDFsam consists of a few **lazy classes** which are **dispensables**. These include **MaxLogRowsChangedEvent**, **ErrorLoggedEvent** and **LogAreaVisibilityChangedEvent**. They are all located in pdfsam-gui/src/main/java/org/pdfsam/ui/log/ and they all contain nothing in the class at all. WE flagged them as smelly because they do not do anything on their own. Being that they are empty they only exist so that the logger can create objects of these types to log and trigger events. That may be useful but the classes by themselves are completely useless. We agree that the code is smelly for the reason that the classes do not do anything even though it is apparent that they are being used by the logger.

What is a lazy class? It is a class that does not do enough.

Why is this class flagged as smelly? The classes are flagged as smelly because they don't do anything at all. They are created and the objects are used in the logger but when taken alone they are completely useless.

Do you agree that the detected smell is correct? Yes, we agree that when each class is taken alone the code smell is correct. These classes are completely empty.

PDFsam code smell 2:

The next code smell we were able to identify in PDFsam is in **pdfsam-rotate/src/main/java/org/pdfsam/rotate/RotateSelectionPane.java** and it is a **message chain** which is a **coupler** code smell. The offence begins on line 60 and the message chain includes calls to getItems, filter, stream and others.

What is a message chain? This is a series of calls of the type a.b().c().d() where method calls are chained together one after the other.

Why is this method flagged as smelly? We flagged this as smelly because it meets the criteria for being a message chain. It can be refactored into a series of calls that can be more easily understood by a developer maintaining the code.

Do you agree that the detected smell is correct? Yes, we agree that the code smell is correct because it is a clear example of a message chain according to the definition.

PDFsam code smell 3:

The last code smell we found for this assignment is related to a **primitive obsession** which is a **bloater** code smell. It is located in the class **pdfsam-core/src/main/java/org/pdfsam/context/DefaultUserContext.java** and involves the use of several static final strings. There are six of these static final strings declared which are used in the methods of the class. We think it would be better if they were enums instead of strings. This way they are guaranteed not to change the same way a final string is not and are easily maintainable.

What is primitive obsession? This is when primitives or constants are used instead of small objects or when string constants are used as field names.

Why is this class flagged as smelly? This class was flagged because of the use of static final strings which could be enums instead. It is not a particularly “strong” smell but we agree that it may be better practice to use an enum.

Do you agree that the detected smell is correct? Yes, we mostly agree that this is indeed a correct code smell. Although it is not exactly wrong to use static final strings it may be a better practice to use enums instead.

4 Removing code smells via refactoring

jEdit code smell #2 removal:
For us to remove the code smell #2 we had to refactor code at the location org.gjt.sp.jedit.options/StatusBarOptionPaneTest.java. The _init() function in the StatusBarOptionPane class contains a lot of logic which we figured can be divided among simple methods. This method consists of logic w.r.t setting the layout north, building option panel, caret position, building widgets panel, and creating buttons. So we decided to create five methods for each of the individual logical categories. Thus reducing the _init() function and creating multiple other functions called setNorth(), buildOptionsPanel(), caretPosition(), buildWidgetsPane(), createButtons(). After refactoring, we ran PMD to check if the code smell still exists and we found that the code smell was eliminated successfully. We also designed a test case to check if the function is working properly. While creating the JUnit test class we felt that it was not necessary to create this function because of the logic present in the _init() function. The _init() function deals with properties related to UI elements which makes it impossible to test for the properties. Therefore, we use the function to test for its execution.

jEdit code smell #3 removal:
This code smell is removed by refactoring code at the location org.gjt.sp.jedit.bufferset/BufferSetAdapter.java. This code smell is simple to remove, it can be eliminated by deleting the file. The reason we are removing this file is that from what we observed the class BufferSetAdapter is not being used by any other class and this class has empty methods. We determined that the BufferSetAdapter is not being used by any class using the search function available through Eclipse. From the search results, we found that this class is not being used anywhere else and the functions are not called. Therefore, we found that this class of no importance with respect to the project and therefore we felt that this class is unnecessary and can be deleted. We have no test functions to test any of the functions in this class because we are removing the entire class and the file is deleted and the project builds and runs without this class.

PDFsam code smell #2 removal:
The first code smell that we are removing for PDFsam is the message chain that resides in pdfsam-rotate/src/main/java/org/pdfsam/rotate/RotateSelectionPane.java. There is currently a test case that exists in the package that ensures that the rotate functionality works correctly. After inspection we determined that the test case did not need to change and we left it as is. We also verified that the code smell was gone with PMD. In order to fix the code so we could remove the smell we converted the message chain which was originally on line 60 into a for-each loop that iterates over each item in the table row cell delimited by a comma. Within the for-each loop we have the if statement from the original code that checks to make sure the page selection entered is not zero. Within this if statement the page range and descriptor are passed to the builder so the appropriate pages can be rotated. Our rationale to change the original

message chain into a for loop that contained an if statement was to make the code easier to understand and therefore easier to maintain. It works the same way but is no longer a message chain which was our goal in removing the code smell. Since we used PMD to identify this code smell and it did not suggest or automatically perform any refactoring operations we did all of the refactoring discussed here manually.

PDFsam code smell #3 removal:

First we checked that test cases existed for this class. They do exist and ensure the proper functionality of the class. In order to remove this code smell we removed the static final strings and added an enumerated data type. The enumerated data type has all the static final variables names as its type names and the string assignments as the values respectively. We also create a constructor and a getter to retrieve the value w.r.t the corresponding type name. Moving the data types from final static to enums also affected the use of these variables. Therefore code refactoring has been made so that all of the code compiles and runs properly. We also had to change the test cases so that the enum type names and values are accessible.