Drew Smith
CEN4021
March 16th, 2016
Assignment 4 - Refactoring

Smell 1:
Type: Coupling
Class Name: FilenameUtils
Function Name: doNormalize(String filename, char separator, Boolean keepSeparator): String
Explanation: The method is excessively large and complex, as it contains a high cyclomatic complexity and nesting level. The method carries a Coupling Quality Deficit of 5. The method contains a series of for loops that iterates through the data to simply perform normalization and 'if' statements to check which arguments are being used (filename, separator, keepSeperator). To fix the smell, I first broke up the long argument list into fields, where I could then create a case statement instead of using a series of 'if' statements. To refactor the 'for' loops, I applied Extract Variable in the loop collection, which gave me a starting point for the pipeline operations that IntelliJ recommended. When there were conditional checks, I moved them into the pipeline with filter operations and map operations. After all was completed, I could then remove the 'for' loops and just return the pipelined result.

Smell 2:
Type: Inheritance
Class Name: FilenameUtilsTestCase
Function Name: setup() : void
Explanation: The Quality Deficit Index of this method is 8. The method is mostly affected by Sibling Duplication, which means that the method has duplication with a method from the inheritance hierarchy. The method uses the base class extensively and in the leaf class, the method is being used by other derived classes and also derives overriding. This smell is use of poor inheritance methods. The function derives the parent function FileFilterTestCase, which contains nearly the same functionality as FilenameUtilsTestCase. To fix the smell, I could have used three different approaches including either renaming the method, moving the field from one class to the other, or consolidating the two similar methods into a superclass. Since the two duplicated methods are implemented in two separate classes and consist of duplication in two sibling subclasses, I decided to refactor the code into a method and put it into a common superclass. Therefore, I was able to call the single method, setup(), for both separate objects. I also had to manually add a few conditional statements to check which object was being referenced to and each of their accessor methods for the conditional statements. Last, I had to change of the supporting methods

Smell 3:
Type: Cohesion
Class Name: FileCleaningTrackerTestCase
Function Name: teardown() : void
Explanation: The method has a Quality Deficit of 2.4 as it heavily uses attributes from one or more external classes, directly or via accessor operations. Furthermore, in accessing external data, the method is intensively using data from at least one external capsule. The function is affected by Feature Envy and is weakly connected to its containing class in terms of data usage. All the function does is to reset a class but also contains a block of code that supports the resetting of instances that may be removed. IntelliJ recommended the block of code that resets instances could be removed when the deprecated function, FileCleaner, is removed. By doing this, I would be supporting the reuse of the FileCleaningTracker instances, which sounded like a promising solution at first but ran into much trouble doing so. So I had the remaining refactoring options of moving the method into the class where the function is heavily used, extracting the method, or extracting the class. Since there were several envious methods and the functionality did not quite belong in each of the others objects entirely, I decided to extract the class. Therefore, I created a new class called NewClean and moved the relevant fields and methods from the old class to the newly created class to have all the relevant functionality done in a single class. Last, I had to add the import declarations to include the newly created class into the classes where the methods were being used.

Drew Smith
CEN4021
March 16th, 2016
Assignment 4 - Refactoring