Assignment 3

Juan Manuel Florez, Barbara Maweu - Group 5

Only the instance of Internal Duplication was presented as a smell in Assignment 2. We didn't use any other smells from that assignment because we deemed they would require substantial architectural changes to remove.

Note: As is a convention on many systems, the tests classes created are located in the same package as the tested classes but in the *test* source root.

Automatic Refactoring

Internal Duplication

We removed an instance of the Internal Duplication smell reported by inCode in the **SearchService** class. The smell was being caused by a fragment of code taking a **Patient** object, performing a check on one of its properties and then creating a **PatientItem** object from it. This fragment of code was found in the methods *retrievePatientsForSearch* and *retrievePatientsFromQueryString*.

- We highlighted the code fragment in the *retrievePatientsFromQueryString* method and used the “extract method” refactoring provided by the IDEA IDE. The IDE correctly identified the *patient* variable as a parameter to the method. However, it also identified a list as a parameter, but we deemed this didn’t belong inside the new method and unchecked it from the dialog.
- The code’s ultimate goal was creating an object, but in *retrievePatientsFromQueryString* the result was being directly added to a list (the one that was identified as a parameter). Because of this, it was necessary to manually add a return type and a return statement to the new method and move the list addition back to the original method.
- We also had to manually replace the duplicated code in the *retrievePatientsForSearch* method and make sure the result was assigned to the variable named *currPatient* to keep functionality intact.

We did not create a test case for the new method because the optimal visibility for it is private. It is instead tested indirectly by testing the two methods from which it is called.

Manual Refactoring

Feature Envy

We removed one case of Feature Envy from the method *createResearchFilterItem* from class **ResearchController**. This method exhibited this smell because it created a **ResearchFilterItem** object using data from a **FilterViewModel** object. Because of this, getters and setters from the two classes were heavily invoked. We solved this by creating a constructor in **ResearchFilterItem** that takes a **FilterViewModel** object as parameter. This way, the initialization of its fields is handled by its constructor instead of by an external class, eliminating the smell.

The refactoring was tested indirectly with a test case for the method *indexPost*, since it calls *createResearchFilterItem*, which is private. This test case actually helped identify an error caused by a small overlook during refactoring.

Schizophrenic Class

An instance of Schizophrenic Class smell was removed from the class **LocaleUnitConverter**. We did this by changing the scope of some methods from public to private. These methods were not being used outside this class according to both inCode and IDEA. In the same fashion as the other refactorings performed, we tested these private methods through some of the invoking public methods.

Comparison of Methods

Automatic refactorings are very useful for avoiding repetitive manual labor such as renaming a variable. However, when more complex refactorings such as method and class extractions are needed, it is usually necessary to manually modify certain parts of the code to complete the refactoring. Sometimes it would be possible to use a series of smaller refactorings to significantly reduce the amount of manual work necessary, but this requires the developer to know about each one of them and have a clear plan. In other words, with the current state of refactoring tools (at least those present in the IDEA IDE), it is necessary to have a good amount of refactoring experience in order to take full advantage of the automation, except for the more menial tasks.