Harold Bowers (hrbowers)

SER316 – Mehlhase

Assignment 7

Task 1

SIZE:

1.) Total LOC is 2187
2.) The largest single code file is EventsManager.java with a total LOC of 329
3.) Used noteChanged, a method that iterates through note listeners and saves current note

COHESION:

1.) Defintions of LCOM2 :

    m – number of procedures (methods) in a class
    a – number of variables (attributes) in a class
    mA – number of methods that access a variable (attribute)
    sum(mA) – sum of mA over attributes of a class

    LCOM2 can be calculated as:

    LCOM2 = 1 – sum(mA)/(m*a)

    Where LCOM2 equals the percentage of methods that do not access a specific attribute averaged over all attributes in the class. If the number of attributes or methods is zero, LCOM2 is undefined and displayed as zero.

2.) The class with the highest cohesion is a 10-way tie of classes with a score of zero. One of them is ProjectImpl.java, and I believe this is because it has methods that mostly operate on the same attributes of the class.

COMPLEXITY:

1.) The mean cyclomatic complexity for the main package is 1.746.
2.) The class with the highest McCabe score is Start.java with a score of 3.5.
3.) In EventsManager.java I commented out the first if statement located in public static Collection getEventsForDate(CalendarDate date), specifically line 114. I decided to make this change because this specific statement did not seem necessary to the program's functionality, and since complexity measures the possible paths of a class, I knew removing an if statement would reduce the number of paths. I was able to reduce the score (for EventsManager.java specifically) from 2.5 to 2.469. This lowered the overall package score from 1.746 to 1.743.

PACKAGE-LEVEL COUPLING:

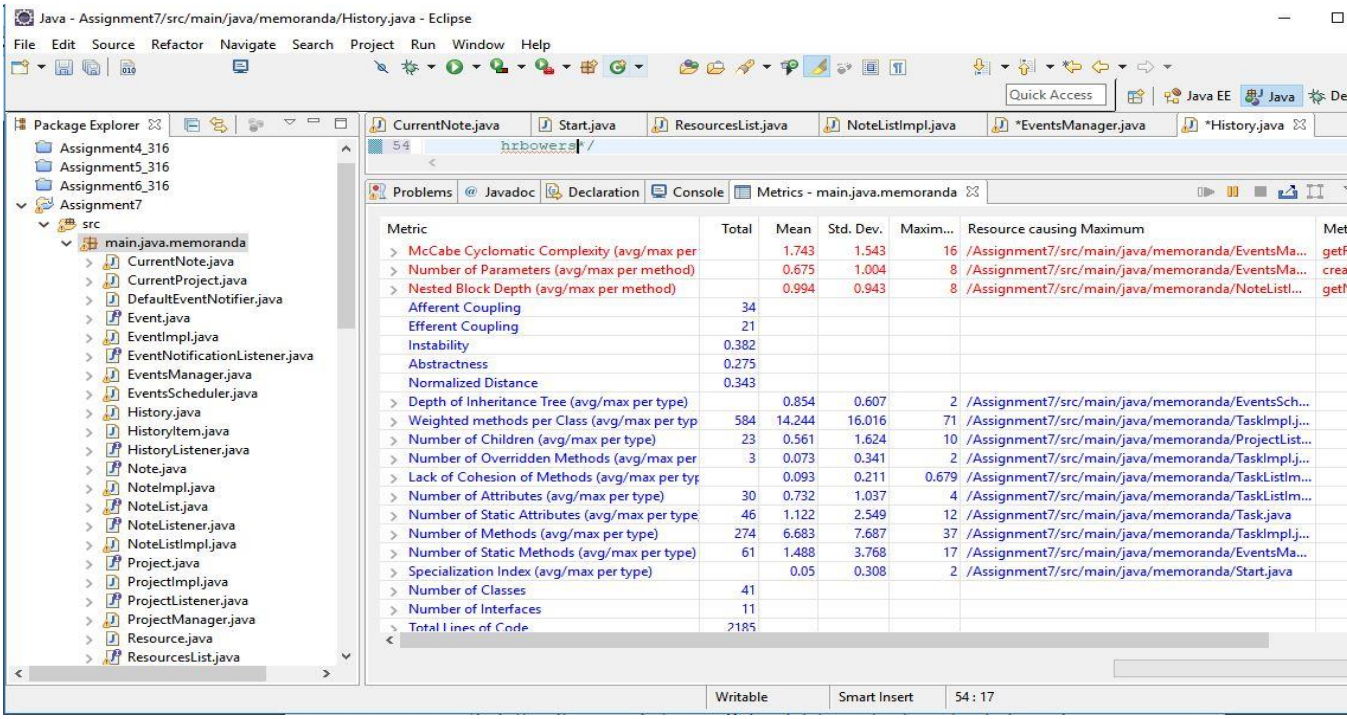
1.) Coupling is the degree of interdependence between software modules. **Afferent coupling** is the number of classes in other packages that depend upon classes within the package is an indicator of the package's responsibility. Afferent couplings signal inward. **Efferent coupling** is the number of classes in other packages that the classes in a package depend upon is an indicator of the package's dependence on externalities. Efferent couplings signal outward. The key difference between the two is that afferent measures responsibility to other classes while efferent measures dependence on other classes.
2.) main.java.memoranda.util has the highest afferent coupling with a score of 57.
3.) main.java.memoranda.ui has the highest efferent coupling with a score of 49.

WORST QUALITY:

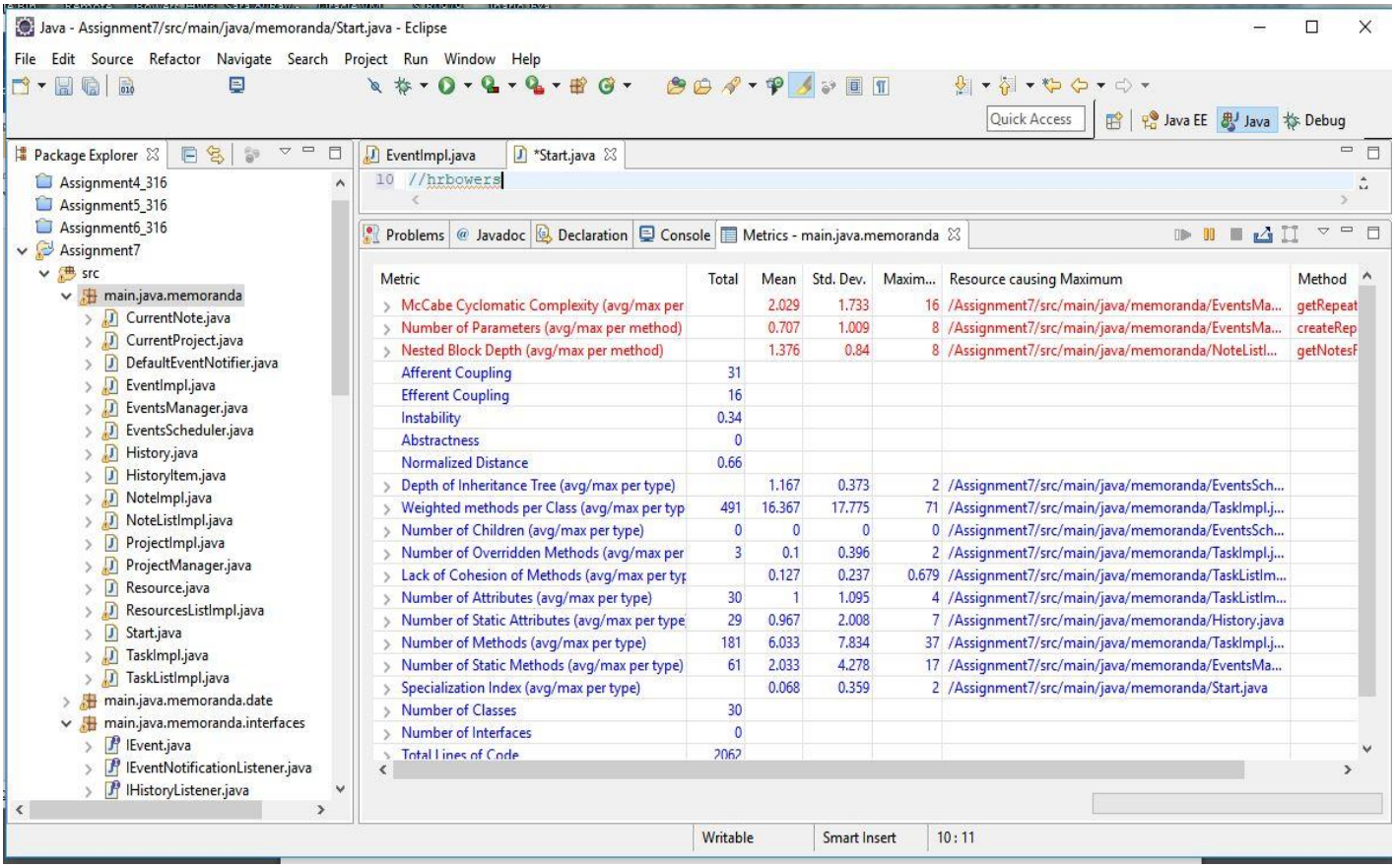
The class with the worst quality in my opinion is TaskListImpl.java. To arrive at this conclusion, I searched to find a class that ranked high in both cyclomatic complexity and lack of cohesion. TaskListImpl.java, which had the highest lack of cohesion score (0.679) as well as being above a score of 2 in complexity (2.273). While other methods would score worse in complexity, TaskListImpl.java was the only one that also scored poorly in cohesion.

Task 2

1.) Rerun of metrics plugin



7.)Rerun of metrics plugin after moving interfaces to new package and refactoring



8.) Abstractness dropped from 0.275 to 0 and Instability dropped from 0.382 to 0.34. Efferent coupling dropped from 21 to 16. Lines of code dropped from 2185 to 2062. These drops occurred because of the movement of interfaces to the new package, while stats like complexity and cohesion actually got worse because the interfaces helped to lower the score.

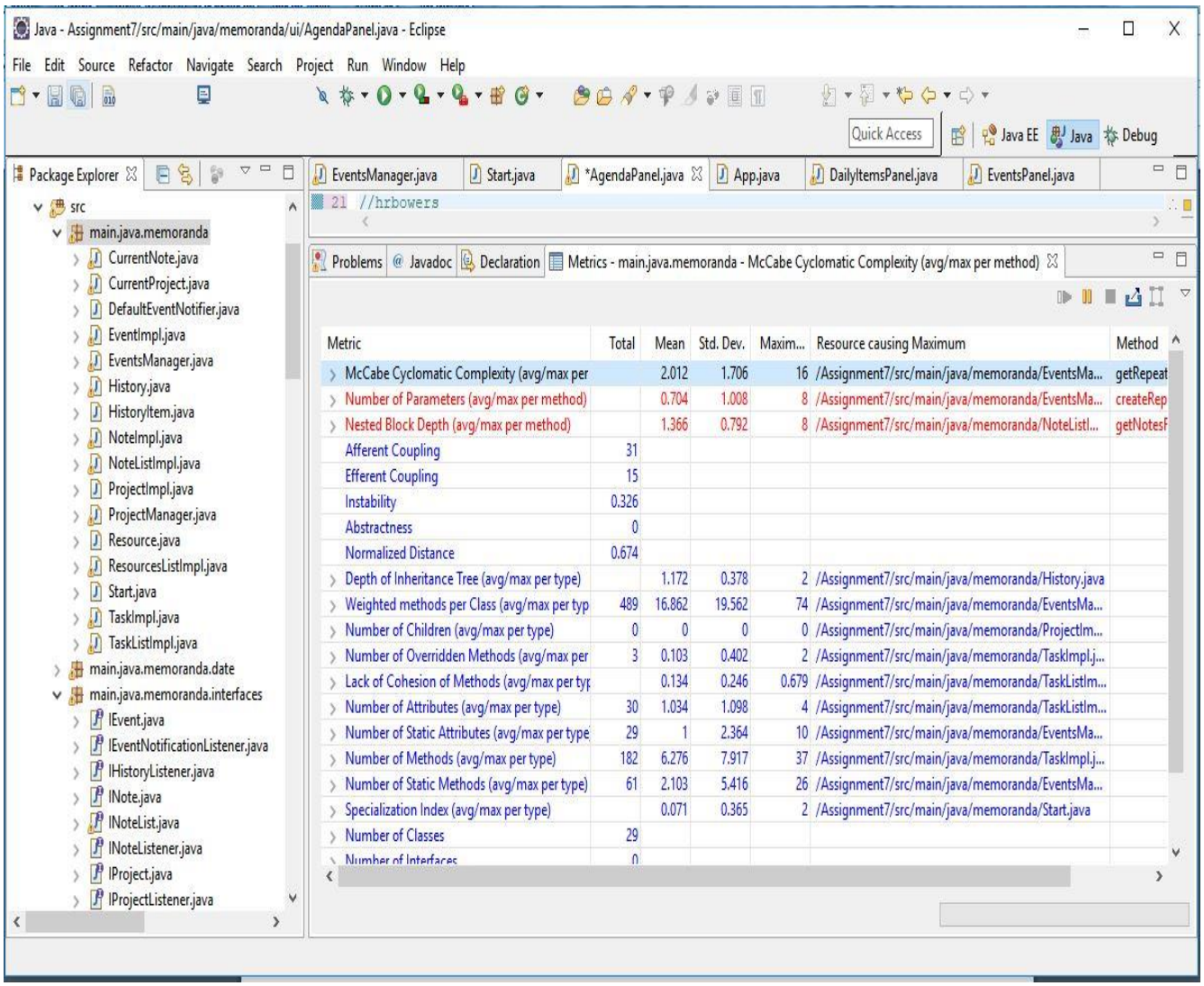
Task 3

1.) For smell within a class I found duplicate code in NoteListImpl.java, specifically the nested for loops used in public Collection getAllNotes() (line 77) and public Collection getMarkedNotes() (line 99) used to iterate through dates. To fix this, I created a private method inside the same class called private Vector v() (line 54) which handles the for loop iteration and is then used by both of the methods.

2.) Smells between classes was a bit more difficult as I wasn't sure if changes would enhance anything, but I decided that class envy between EventsScheduler.java and EventsManager.java was a valid smell. To alter this, I moved all code from EventsScheduler into EventsManager and deleted the EventsScheduler class. References to EventsScheduler were changed to reference

EventsManager in AgendaGenerator.java of package main.java.memoranda.util. In package main.java.memoranda.ui changes occurred in AgendaPanel.java, App.java, DailyItemsPanel.java, and EventsPanel.java. The end result is removal of the EventsScheduler class.

3.) After (code smell fixes)



4.) Numerous metrics show a change for the better after fixing the code smells. Cyclomatic complexity has dropped from 2.029 to 2.012 because of the duplicate code fix, moving all the if and for loop statements into a single method shared by others. Efferent coupling dropped from 15 to 16 as a class was removed due to class envy, reducing dependency. Instability dropped from 0.34 to 0.326, and obviously the number of class has dropped from 30 to 29. While all these were positive changes, I did notice that LCOM actually rose from 0.127 to 0.134, which I believe is due to the EventsManager class now taking on all the variables of the EventsScheduler class. All in all, I believe the smell fixes were beneficial.