**Task 1: Evaluate Metrics**

**Size**

1. What is the Total Lines of Code (LOC) in the project?

22,539

2. What is the largest single code file in the project and its Total LOC?

EventsManager.java 329 LOC

3. Inspect CurrentNote.java - what method did the Metrics tool use to determine Total LOC? Describe the method.

It used Method 1. This method counts 1 for every statement and 0 otherwise (blank lines, comments, etc.)

**Cohesion**

1. The tool calculates “Lack of Cohesion of Methods” (LCOM) using the HendersonSellers method, or what is commonly referred to as LCOM2 (there are LCOM1 through LCOM4 and different methods to calculate them). What is the definition of LCOM2 and how is it calculated? (there are different methods used to calculate LCOM2).

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

LCOM2 = 1 - sum(mA)/(m\*a)

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

2. Which class has the highest cohesion, and do you have an idea why?

EventsScheduler.java because its only concern is to schedule events. All of its code is contained within itself and it doesn’t create any UI.

**Complexity**

1. What is the cyclomatic complexity in the main package?

5

2. What class has, on average, the worst McCabe Cyclomatic Complexity (CC) and what is it?

EventsManager.java 2.5

3. Go back to your code and reduce the Cyclomatic Complexity. You can choose any class, but the Cyclomatic Complexity needs to be reduced at least by a small amount somewhere. Explain what you changed and why, and why it reduced the complexity and how much you were able to reduce the complexity.

I reduced TaskListImpl.hasSubTasks method from 3 to 2. I removed the “if” conditional because we can simply return the statement. It will return true or false without a conditional.

**Package-Level Coupling**

1. What do Afferent and Efferent coupling mean? Look these terms up on Wikipedia and summarize the distinction.

Afferent Couplings: 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. Summary: Classes from other packages depend on classes within a package.

Efferent Couplings: 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. Summary: Classes within a package depend on classes from other packages.

2. What package has the worse Afferent Coupling measure and what is the value?

main.java.memoranda.util - 57

3. What package has the worse Efferent Coupling measure and what is the value?

main.java.memoranda.ui - 49

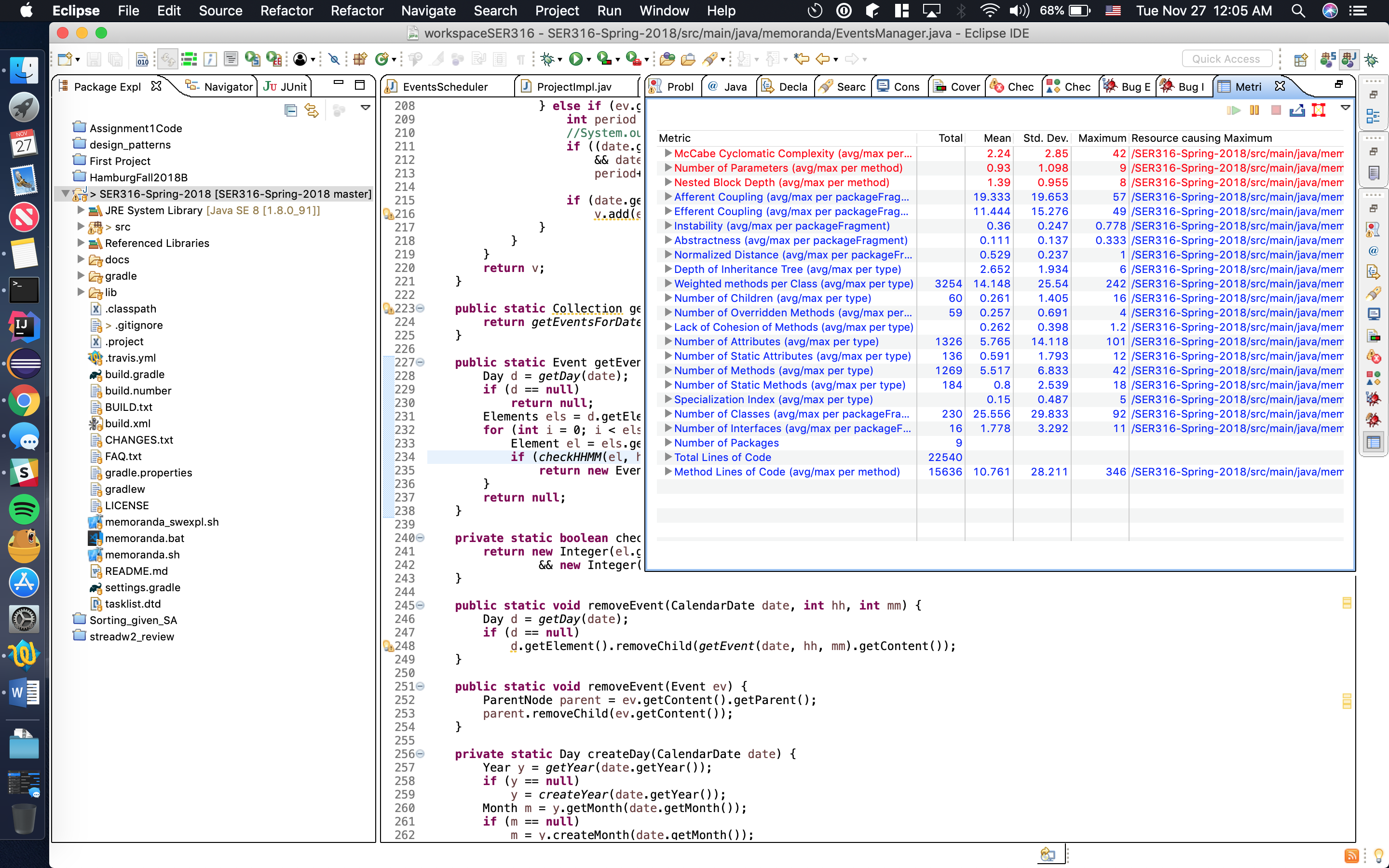
**Worst Quality**

Which class has the worst quality and why? Use the metrics to support your answer. You can use any metric or different metrics together, but you have to give valid arguments.

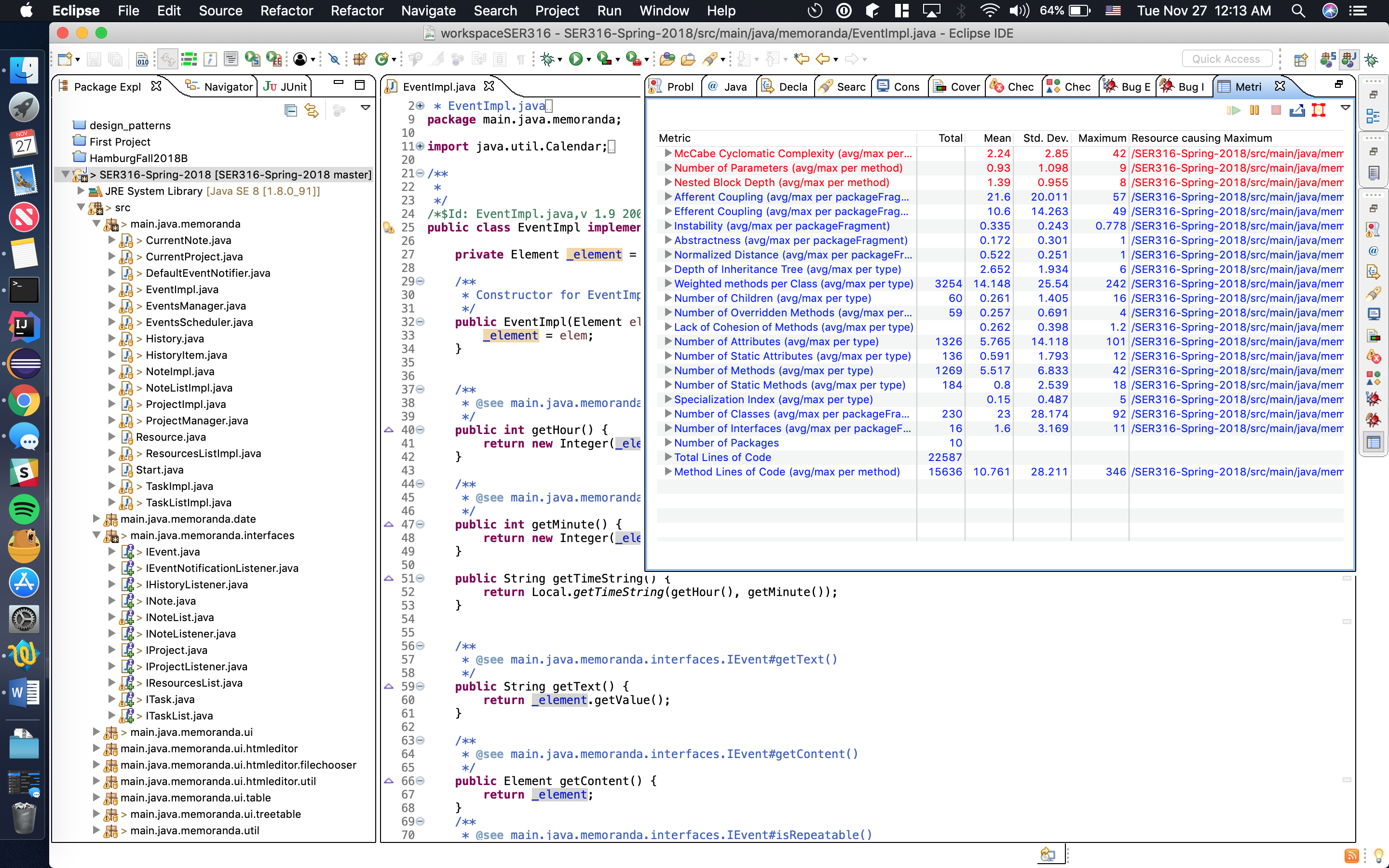
NoteListImpl.java has the worst quality because it’s second in cyclomatic complexity, first in nested block depth and it’s fourth in lack of cohesion of methods. Seems like a lot of refactoring could be done to this class.

**Task 2: Eclipse Refactorings**

**Before:**



**After:**

****

**Changes:**

Afferent Coupling average went up. Efferent Coupling average went down. Abstractness average went up. I believe all three of these are good because other packages will obviously depend on our interfaces, but our interfaces won’t depend on other packages.

**Task 3: Find Code Smells and Refactor**

**Smell Within a Class:** EventsManager.createRepeatableEvent method

**Category:** Long Parameter List

**Reason:** Method has 8 parameters. Too many parameters to pass into a function.

**Fix:** Reduce parameters to 1 by accepting an Object with those 8 fields.

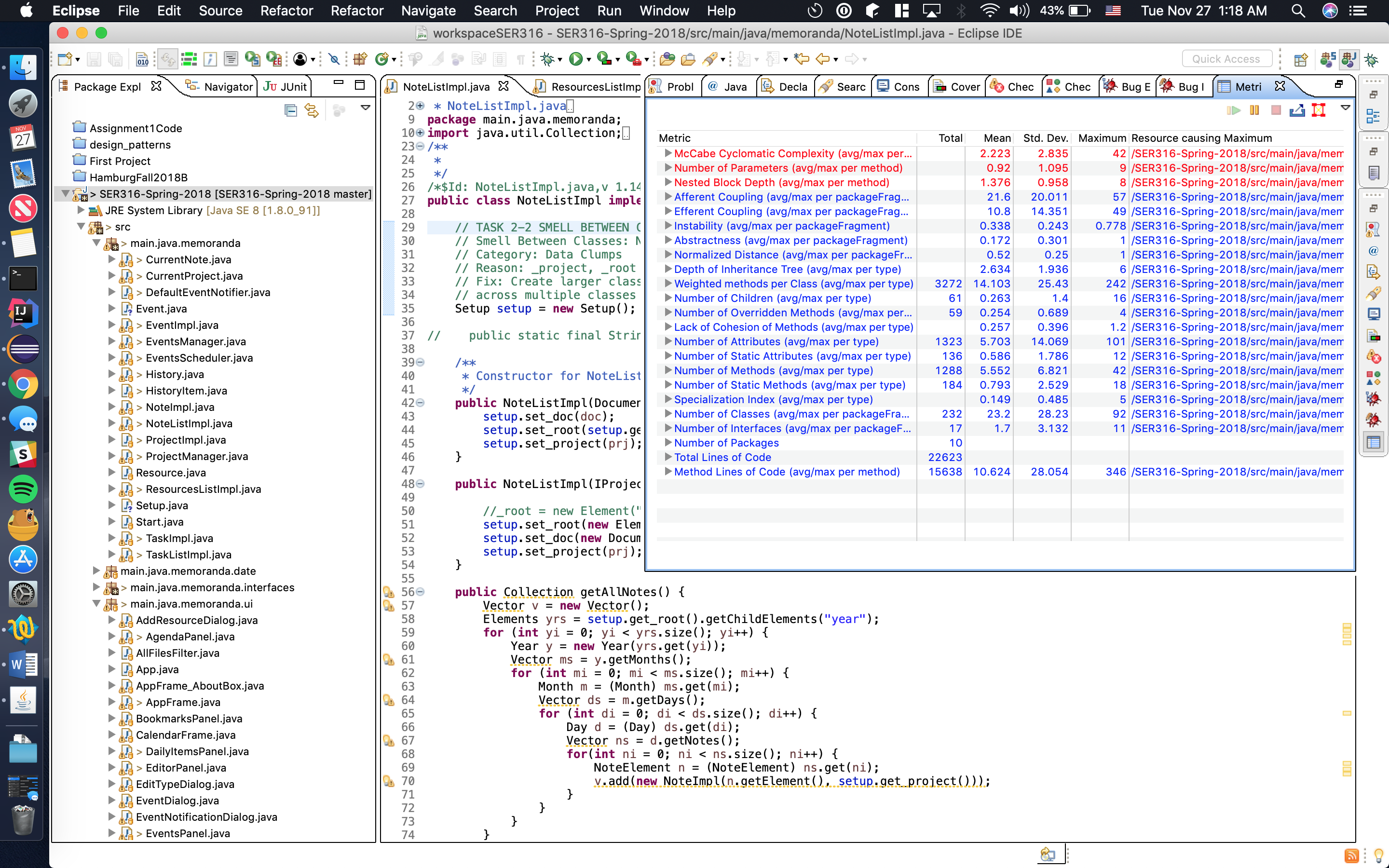
**Smell Between Classes:** NotesListImpl.java, ResourcesListImpl.java, TaskListImpl.java

**Category:** Data Clumps

**Reason:** \_project, \_root & \_document variables were being set up in multiple classes.

**Fix:** Create larger class (Setup) to hold these three variables and use this class across multiple classes to get/set the 3 variables.

**After:**



Cyclomatic complexity and Lack of Cohesion of Methods dropped. Any time these metrics drop, I would consider it a successful change. Number of parameters dropped because of my “smell within a class” fix. Lack of cohesion dropped because of my “smell between classes” fix.