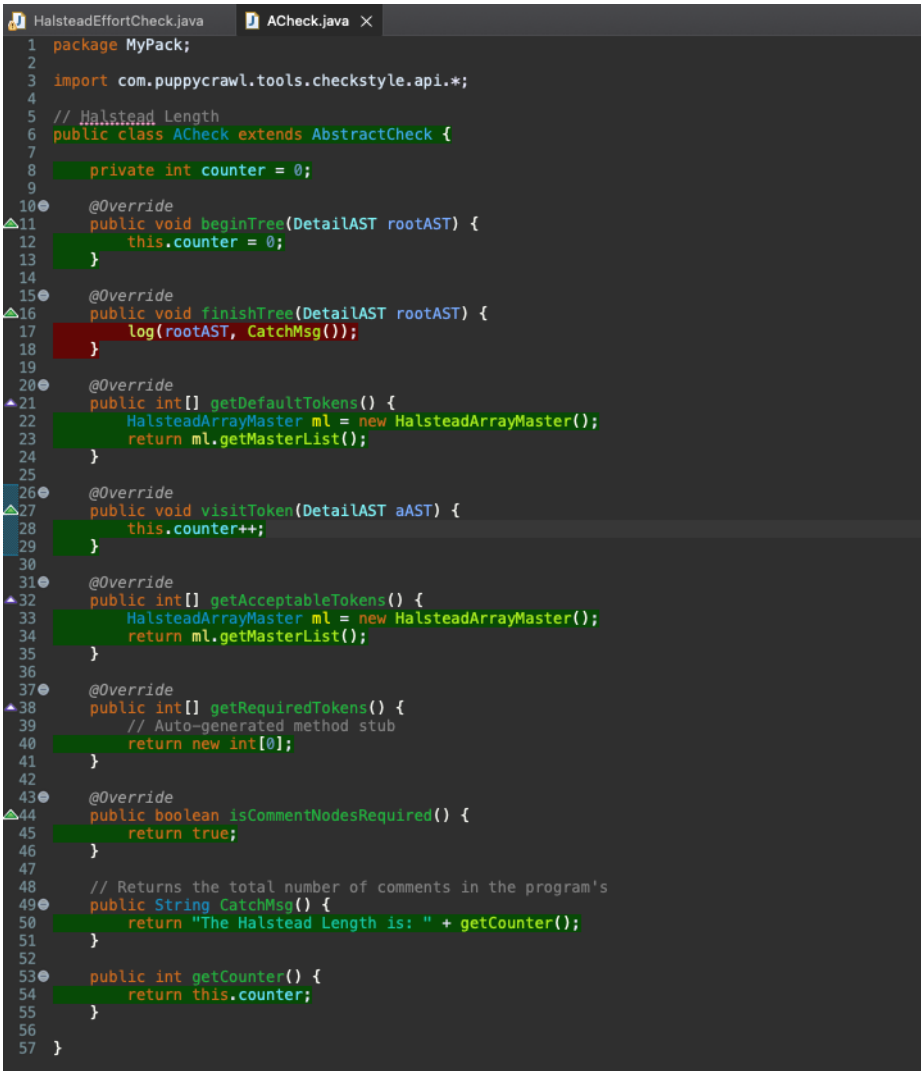
Deliverable 2 – Tony Barrera

Halstead Length Check (ACheck.java From Deliverable 1)

The Halstead Length Check has a 87.5% line coverage, missing two lines involving the log() method. This method proved to be tricky as even when trying to mock and spy, the line coverage never seemed to count it. Perhaps it was because I told it to “doNothing”, but I cannot be sure. Using the eclipse tools for both line count and branching analysis. I used spy and real objects to allow for the most amount of coverage for these tests. The only method not covered by my tests was finishTree(DetailAST) because it used a log(…) function that needed some params (DetailaAST) and I could not figure out how to spy or mock while getting coverage. I was able to spy and mock but the coverage would not count it. I imagine it is because we are not actually executing the code and it makes some sense. All Other methods that needed a DetailAST were provided with a default DetailAstImpl class object as sometimes we needed a to extract the type from the DetailAST, but some methods did not need it so I just provided them with null. This also goes for most of the check classes moving forward.



HalsteadVocabularyCheck
The Halstead Vocabulary Check has a slightly less percentage of lines covered at **86.7%,** we have to deal with the log() function just as before which will not be covered. However everything else is tested with a real object until we get to testing the catchMsg(). This does not absolutely require the use of a spy object but it can be seen that using it allows us to test for different strings that get returned.

HalsteadVolumeCheck:
The HalsteadVolumeCheck was very similar to the class from before with a coverage score of **90.9%** line coverage, all because of our log() method that gives us trouble. This was also the first time we can take advantage of the Branch coverage as it's the first time we use an if statement to prevent us from using log(0) which is not a good idea to do. (Which I caught because of the Unit tests). Branch coverage is: 100%
This is mostly using real objects and one spy object for the catchMessage with a different number. Still tried using a mock object for finishTree but it still does not count it.

HalsteadDifficultyCheck:
Another bug found through Unit Testing! Found a division by zero and squashed it! Also had to find ways to increase the complexity of the tests to increase the coverage percentage. After everything I was able to achieve **94.6%** line coverage. **100%** branch coverage was achieved with creating tests that would allow the flow go through every new possible branch.

HalsteadEffortCheck:
This check was a little more difficult to deal with because of the calculation of the Effort, not only are we getting the difficulty, we are also getting the volume. So we need to create tests that take all of those divisions into account. Thanks to the testing I was able to safeguard all those divisions from zero and clean up the code so that it is easier to read and more concise. In order to achieve **100% coverage** I used spy objects to simulate when Volume is NonZero and when difficulty is zero as those are branches I had created due to all the safeguarding I put into place. This gave me a **96% line** coverage.

B CHECKS:

NumberofComments (BCheck.java)
Compared to all the AChecks, these checks are less complicated and will prove to be quicker to get through in terms of safety guards needed to avoid any invalid outputs and inputs. For Number of comments, we get an **87.5% line** coverage. Still struggling to figure out how to get the spy or mock objects to count this as a line covered. For now, we move forward. No branch coverage needed.

CommentLineCountCheck
Similar to the above, we're only using the spy objects where it is possible, although it does not seem like it is absolutely needed. Just being used to place a value in the getCounter() method to see if the CatchMessage will come back as something we expect given those circumstances. Line coverage is **88.2%,** again dealing with that log() function. Tried multiple times to create a DeatilAstImpl() and setting the type and text to see if it would allow for it but no luck.

LoopStatementCountCheck
During this check I was able to find a small inconsistency in terms of what was being returned to be logged, so it was nice to see that the tests caught it. We are still using the spy object to change the return value of the counter to play around with the possible return values. Line coverage is **88.2%,** again dealing with that log function. Trying to use the spy object.

OperandCountCheck:
When going through the unit test process of this class I discovered that I wrote it in a very roundabout way and it was causing my tests to be complicated as a side-effect. That made me re-write my implementation of operand counter and make it make more sense. This follows a structure more closely to the classes above. Testing the line coverage gives us a coverage of **91.3%,** again getting no coverage for the log function. No branch coverage needed.

OperatorCountCheck:
This check is almost identical to the OperandCountCheck, the only things that change are the tokens and the return message that gets logged. I also ended up having to rewrite this in order to make it simpler. Spy object for the getcount() method that will allow us to modify the message displayed on checkstyle reports. Line coverage is **88.2%,** no need to branch coverage.

ExpressionCountCheck:
Similar to the last two class checks above, Expression count only changes the tokens accepted and the display message. We follow the same structure for the tests making sure all possible lines are covered and verify that there is no branching to do. This leaves us with a coverage of **85.7%,** again the only line not covered is the log(). No branching is required for this one.

TypeCastCountCheck:
For an attempt at extra, I decided to implement the typecast count check as it aligns to our previous BChecks. This check is only changing the tokens that are accepted and the message again. The spy object is being used to change the value of getCounter() which allows us to display a different message from what may have appeared. Even though we could use real tokens to visit, mock lets us do it quickly. Line coverage is **87.5%** with no branch coverage needed.