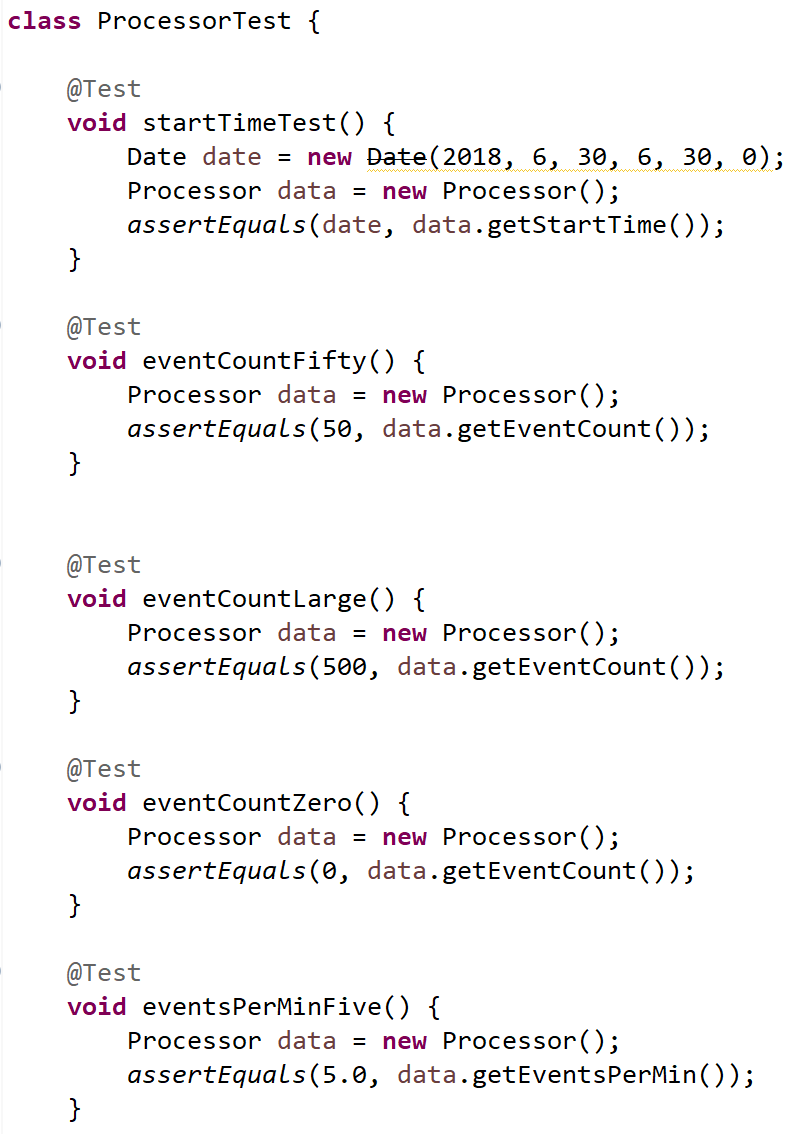
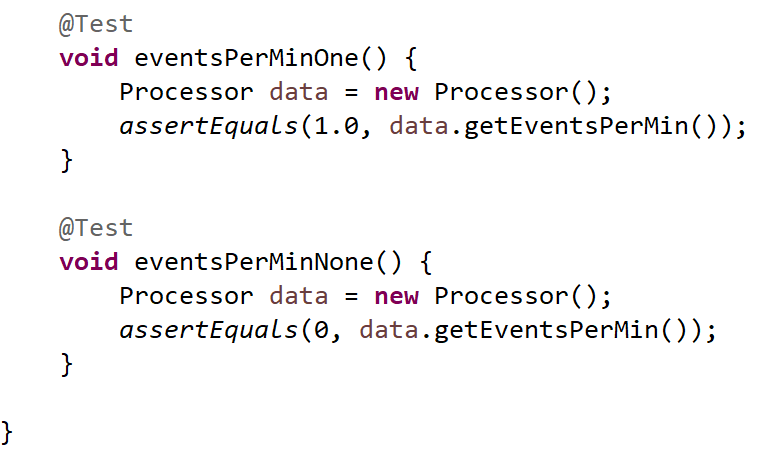
**3-1. Initial Tests**

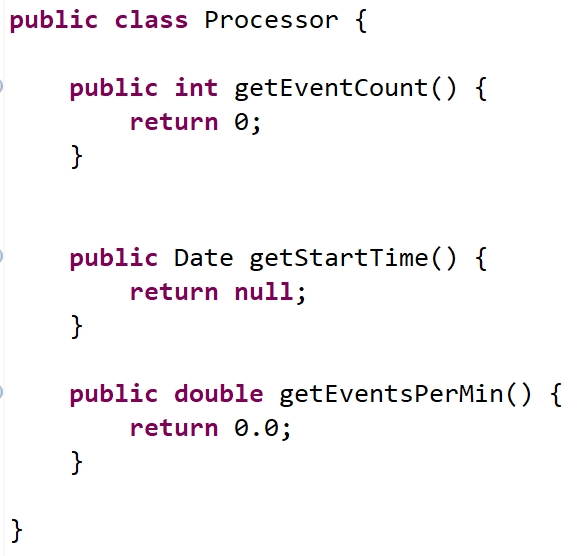
For the requirement of “Calculating the number of events per minute over some time interval”, we will focus on testing the Processor class. The Processor will be notified when a muon event occurs and will store the total number of events over a data collection period. It will also store when data collection starts and ends so that it can calculate events per minute.

Here, we see seven initial test cases that focus on checking if the correct number of events have been stored (using the getEventCount() method) and on calculating the number of events per minute. None of the test code is complete save for the expected values.

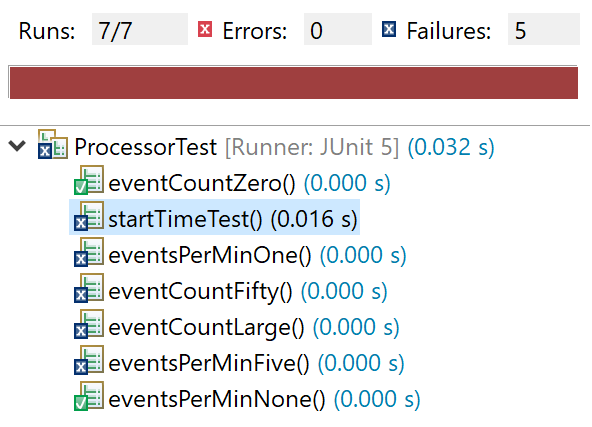




To compile the test code, a basic skeleton of the Processor class was created.

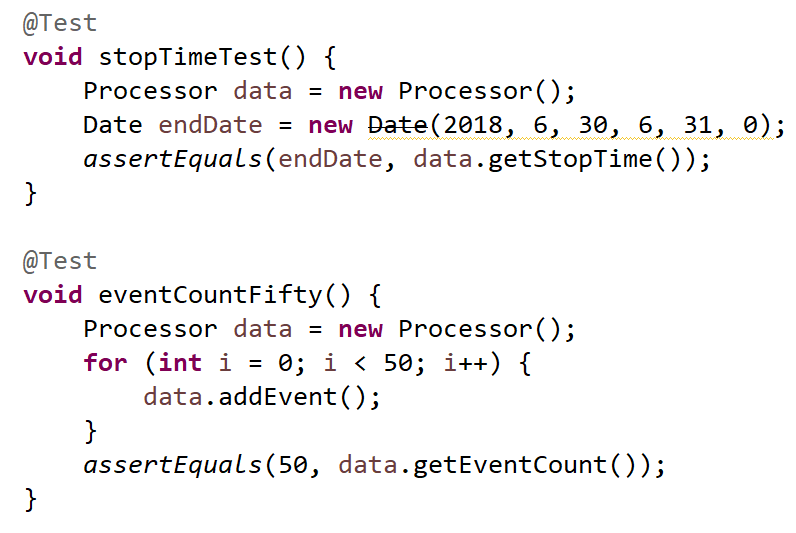


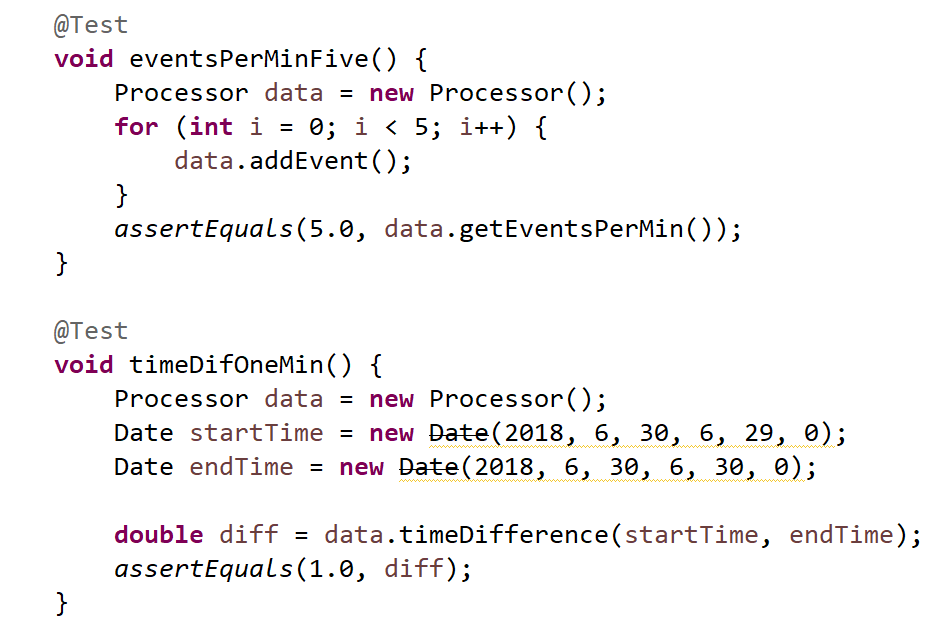
When the tests were run, all but two of the tests failed; the only tests that passed already expected “0” as a result.



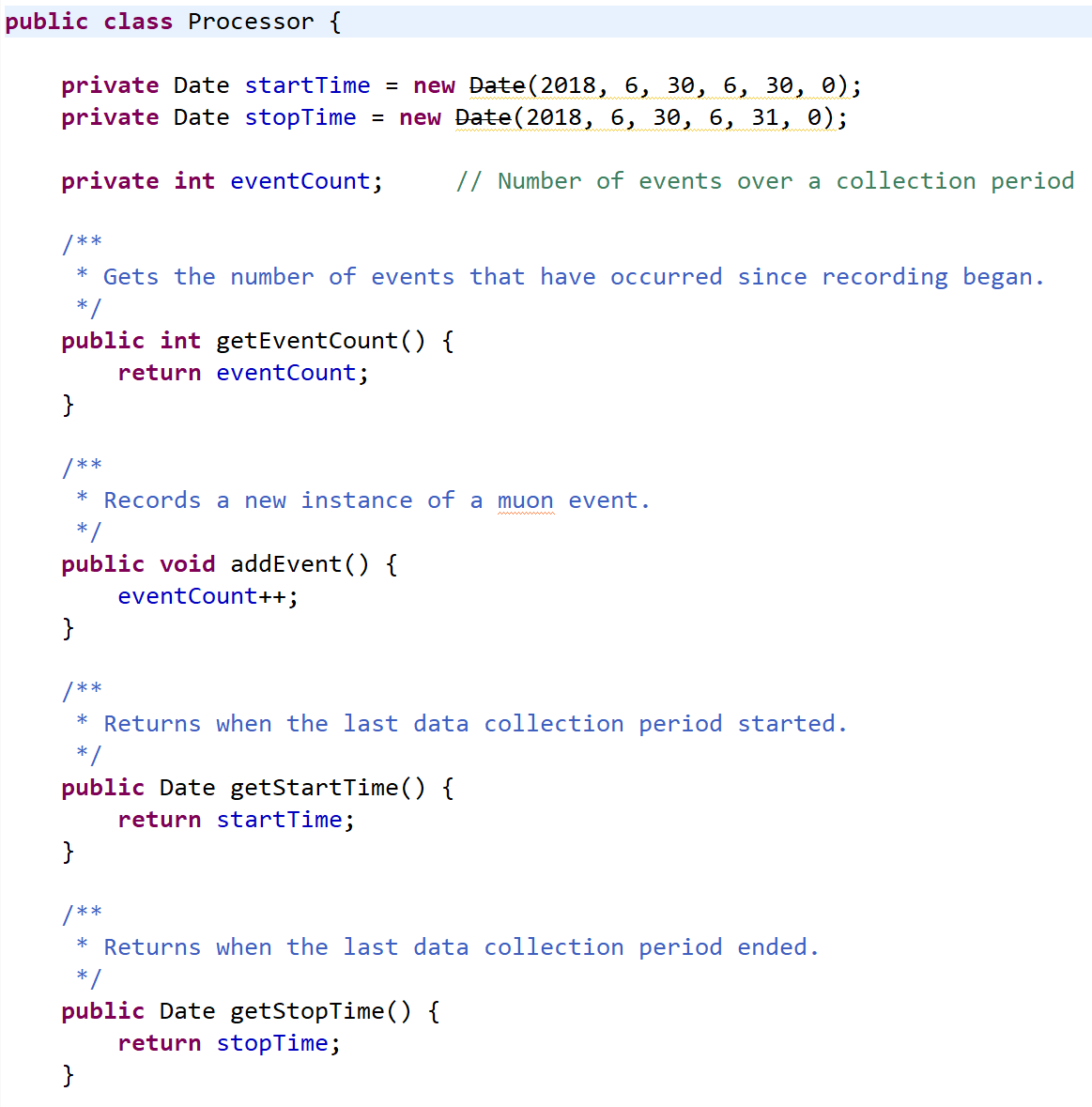
**3.2** **– Tests with Partial Implementation**

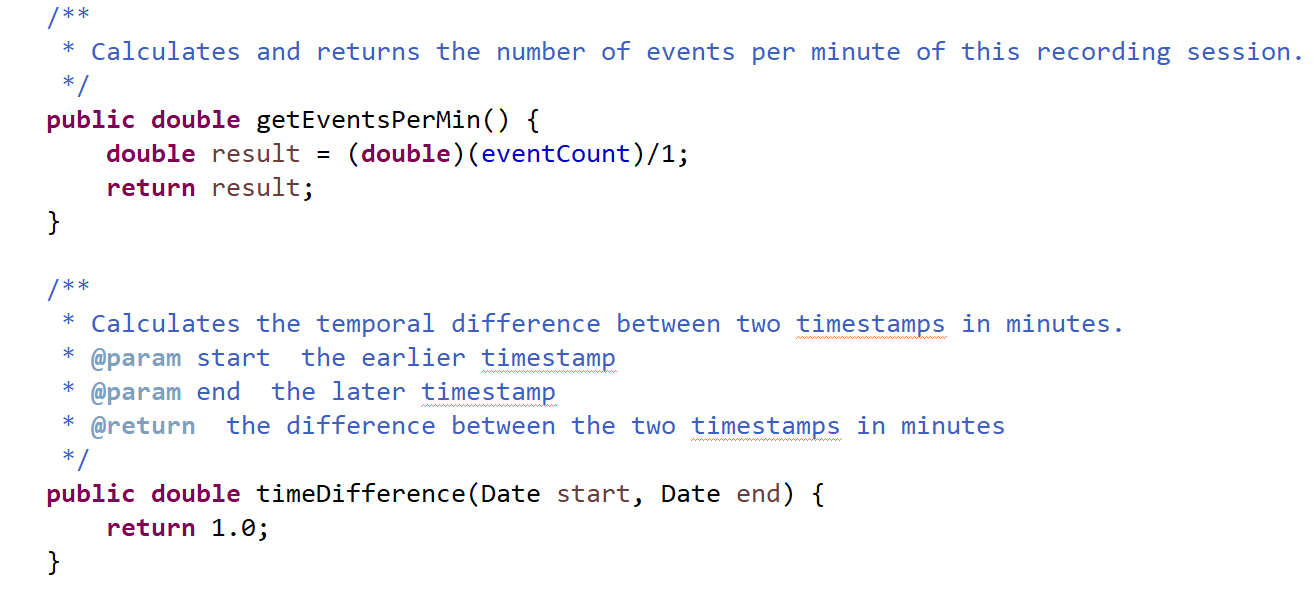
While working on the implementation, we added more test cases for methods that we thought would be necessary later (getStopTime and timeDifference). We began writing more code for the tests and the Processor class so the initial tests would pass. Some examples of the new test code are shown below:



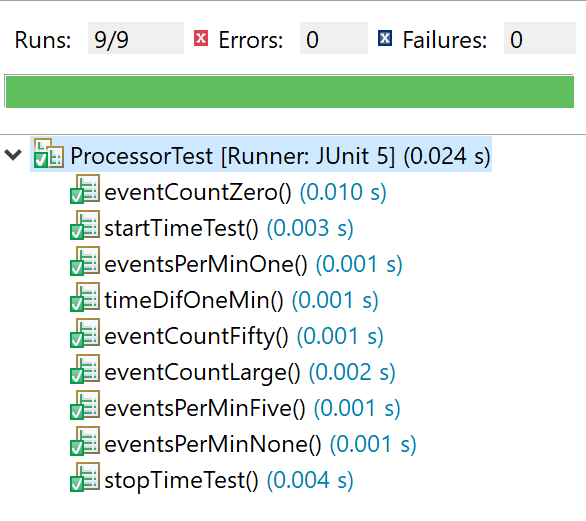


Now, the Processor class stores some basic data that is needed for the tests. However, the values for startTime and stopTime are hardcoded. The Processor class also has partial functionality in its methods, but they mainly rely on hardcoded values as well.



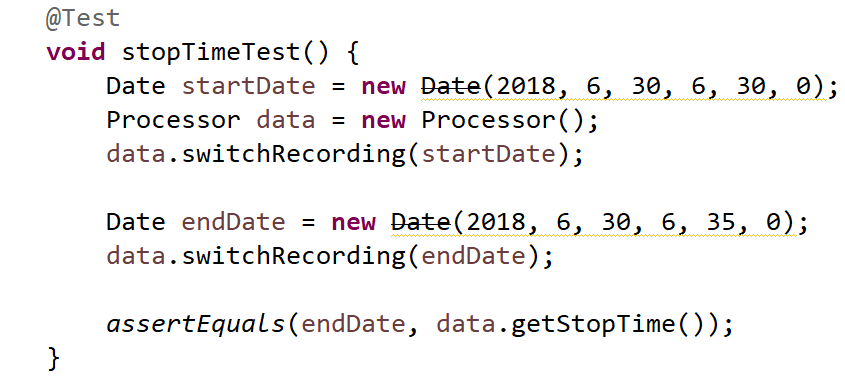


This minimal implementation was sufficient to pass the tests.



**3-3. Tests with Full Implementation**

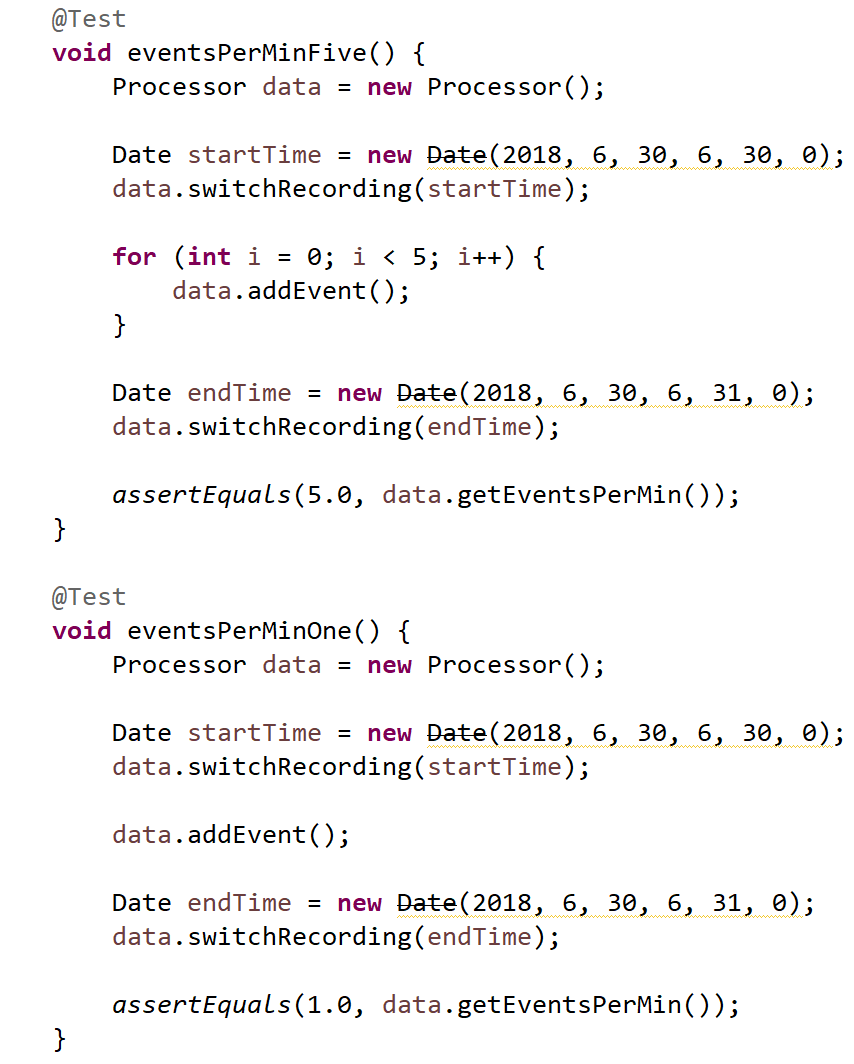
After several cycles of testing and improving the code, we created a fully functional Processor class. Since the Processor is reliant on true event data from the external muon detector, the test cases now attempt to simulate the order that the Processor would receive data. For example:



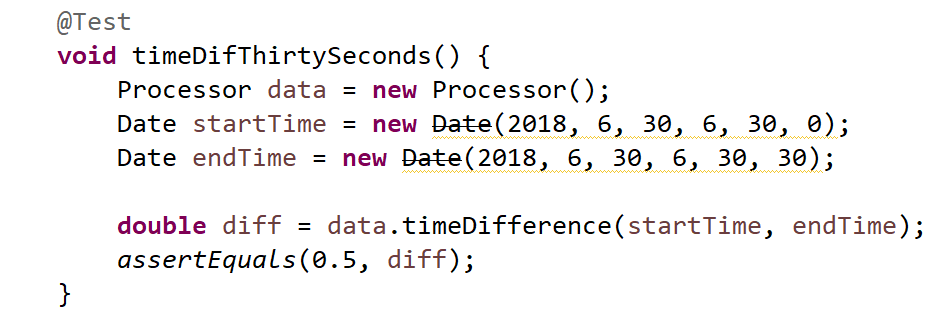
The user of the app is first expected to begin a new data collection session. This is represented by the first call to the new switchRecording method, using the startDate as a parameter. The startDate is used as the time data collection begins in this test case.

Eventually, the user will choose to stop recording, represented by calling switchRecording again. This time the endDate is used as the time when recording is complete. In the end, this test checks that the correct timestamp was stored.

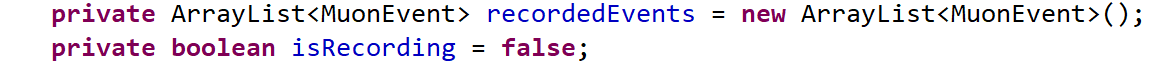
Other tests follow a similar pattern, but also have events added during the “recording time”.



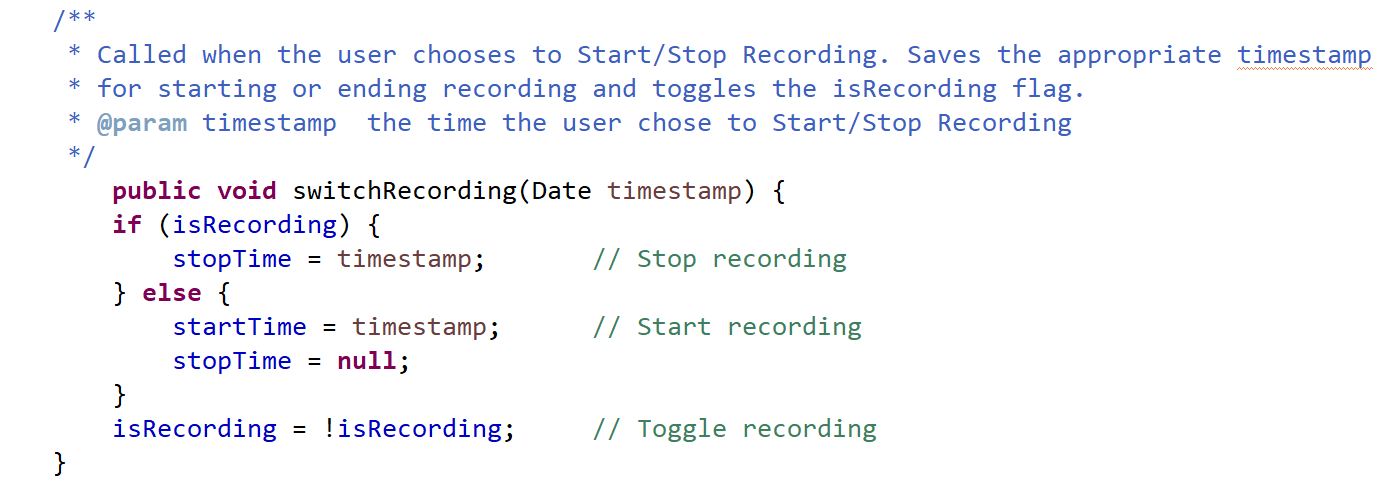
Additionally, a new test was created to check if time differences less than 1 minute could be calculated.

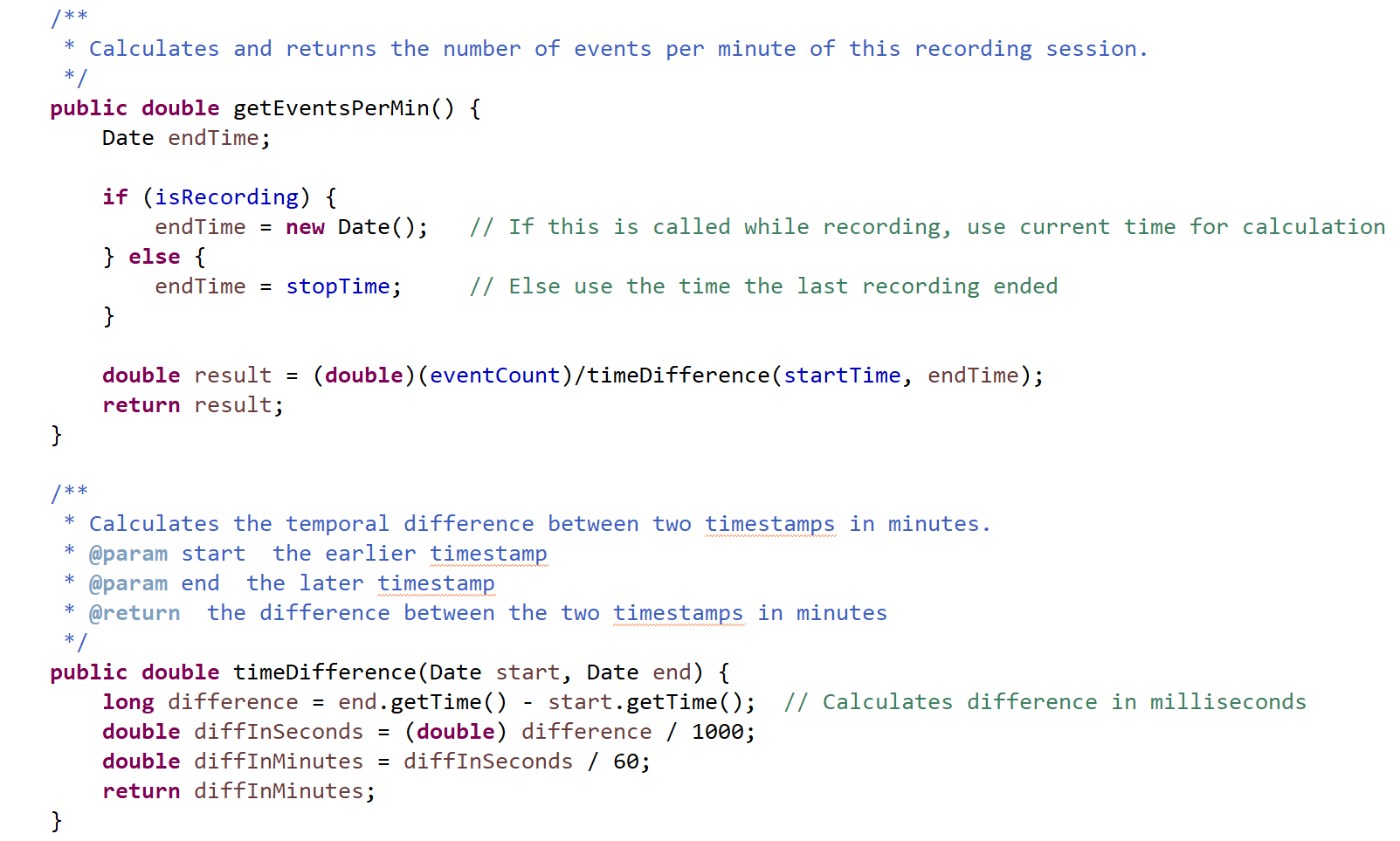


The Processor class now has a flag to check if data is currently being recorded.



The switchRecording methods toggles the flag and stores the timestamps for startTime and endTime. This is again meant to simulate the user’s ability to start and stop recording data.

  
Lastly, the two calculation-based methods are fully implemented rather than using hard-coded values.



The test cases pass once again!

