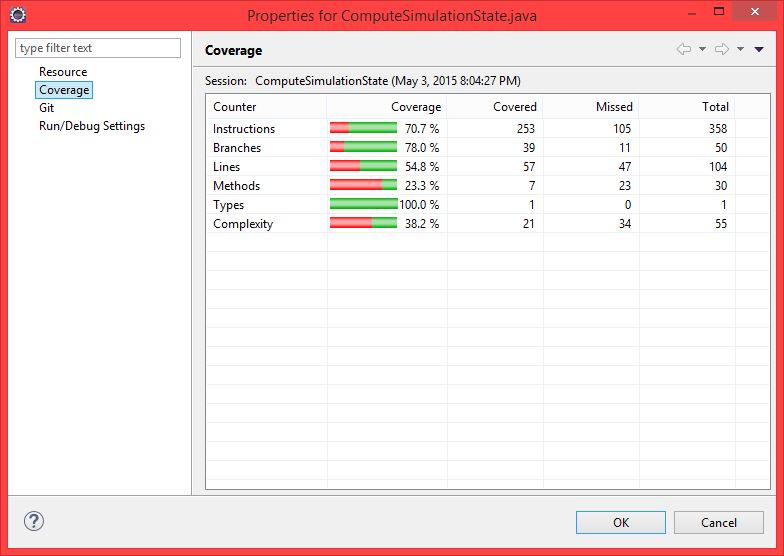
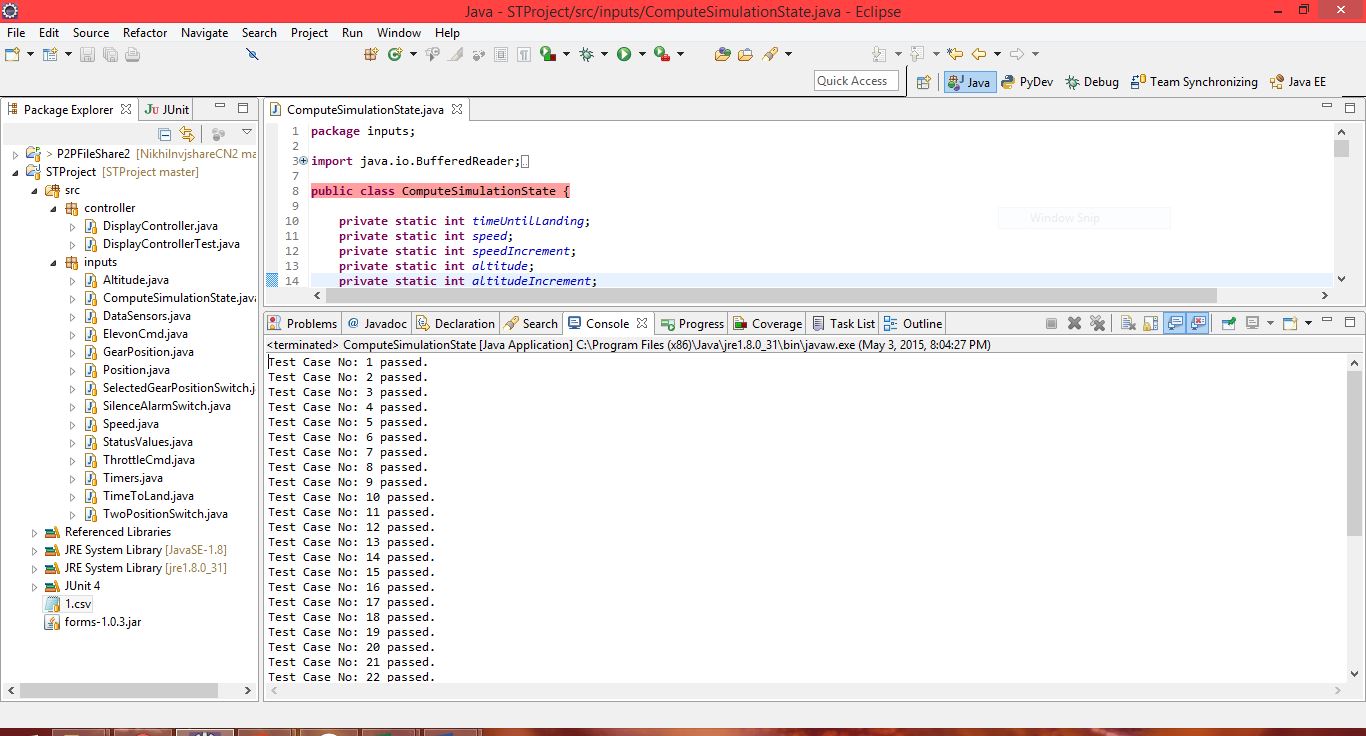
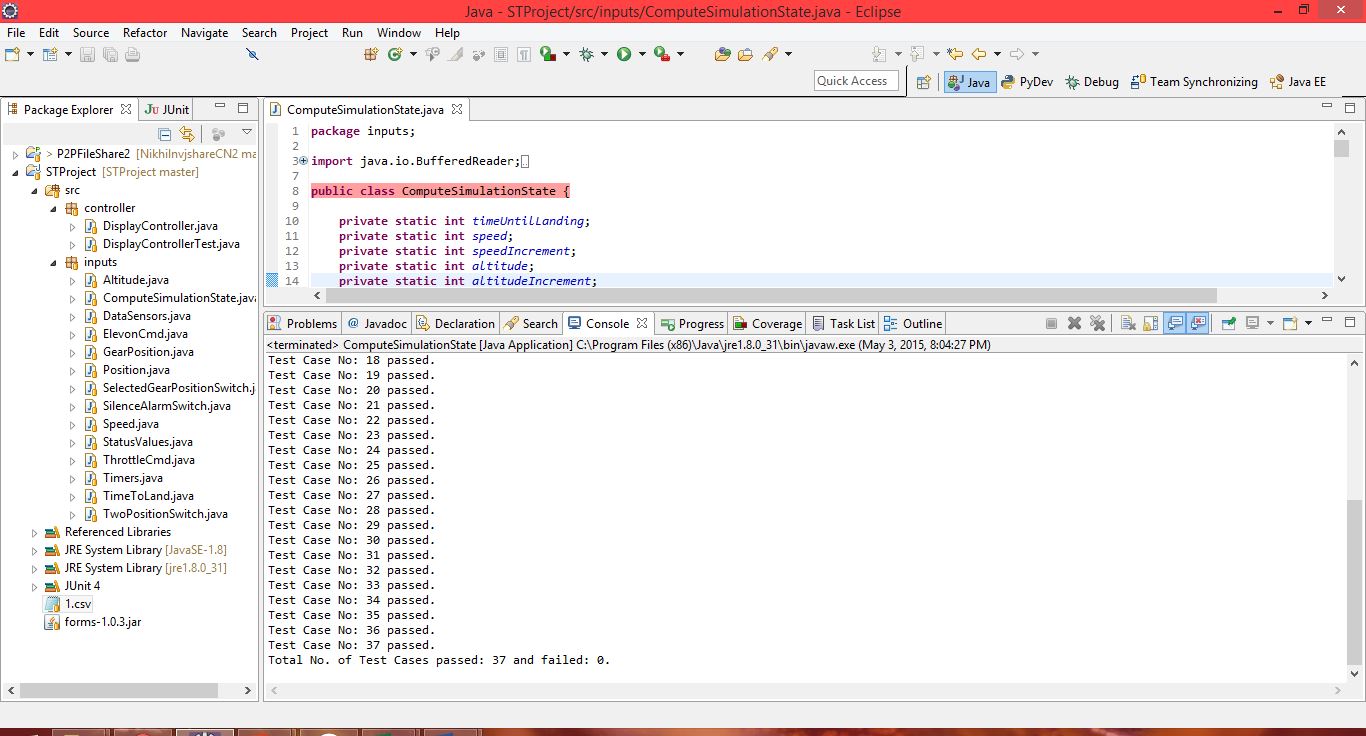
Test report

1. **Compute Simulation State method**
2. Source code of ComputeSimulationState.java with code coverage

|  |
| --- |
| C:\Users\Munna\Desktop\Photos\ComputeSimulationCode1.JPG |
| C:\Users\Munna\Desktop\Photos\ComputeSimulationCode2.JPG |
| C:\Users\Munna\Desktop\Photos\ComputeSimulationCode3.JPG |
| C:\Users\Munna\Desktop\Photos\ComputeSimulationCode4.JPG |
| C:\Users\Munna\Desktop\Photos\ComputeSimulationCode5.JPG |
| C:\Users\Munna\Desktop\Photos\ComputeSimulationCode6.JPG |
| C:\Users\Munna\Desktop\Photos\ComputeSimulationCode7.JPG |

1. Coverage report of ComputeSimulationState.java
2. Test result output on the console as we run the ComputeSimulationState method



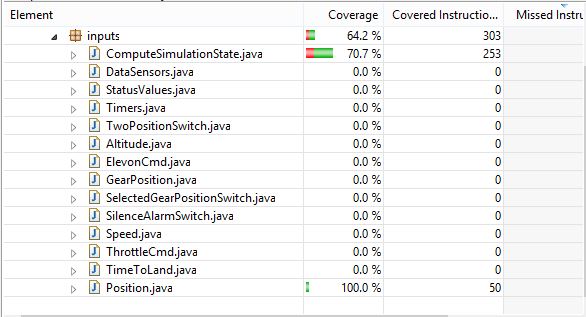


1. Test summary:
2. We have tested the *computeSimulationState* method by reading the inputs from the ‘TestCases.csv’ file and comparing the outputs with the expected outputs provided in the csv file.

* Test case Inputs: Speed, Gear Down, Altitude, Time.
* Test Case Outputs: Gear Not Down alarm, Gear Air Speed alarm, Air Brake warning, Gear Override warning, Gear-up command, Silence Alarm.



1. In the above file, we have 37 test cases, which were run by the *computeSimulationState* method of the *ComputeSimulationState* class which resulted in 100% test success i.e. all the 37 test cases passed successfully and 0 failed. The same can be seen in console output provided above.
2. As shown in the coverage properties above, we have achieved overall 70.7% code coverage. The shortfall of 29.3% is observed because in addition to the *computeSimulationState* method, the ComputeSimulationState class also has some code which is used only by the DisplayController class to get and set input variables for computing alarms and warnings. Also the code that hasn’t been covered in this test-case run is the one handling the button click events (+ , - , Up, Down) of the DisplayController JFrame UI. Also as all the test cases passed successfully, the exception handling statements were not covered in the test run.



1. The test cases helped us in uncovering some issues like uninitialized variables, missing boundary values and some minor logical errors. After resolving them, all the test cases passed successfully.
2. Here, we can observe that we are only testing specific alarms and warnings but we haven’t checked the conditions such as no alarms/warnings etc i.e. we are only testing the positive test cases.

1. **DisplayController Class JUnit Test Run:**

To test and display the 8 scenarios, we comment the main method of the ComputeSimulationState class, that is used to test the above-mentioned 37 test cases, and the main method of the DisplayController class, that is used to run real-time simulation of the aircraft, as these are not required while running the Junit test class DisplayControllerTest.java. If left uncommented, they would decrease the code coverage as those statements would never be executed during the test run.

1. Source code of DisplayControllerTest.java with code coverage

|  |
| --- |
| C:\Users\Munna\Desktop\Snaps\test1.JPG |
| C:\Users\Munna\Desktop\Snaps\test2.JPG |
| C:\Users\Munna\Desktop\Snaps\test3.JPG |

Source code of DisplayController.java with code coverage,

|  |
| --- |
| C:\Users\Munna\Desktop\Snaps\display1.JPG |
| C:\Users\Munna\Desktop\Snaps\display2.JPG |
| C:\Users\Munna\Desktop\Snaps\display3.JPG |
| C:\Users\Munna\Desktop\Snaps\display4.JPG |
| C:\Users\Munna\Desktop\Snaps\display5.JPG |
| C:\Users\Munna\Desktop\Snaps\display6.JPG |
| C:\Users\Munna\Desktop\Snaps\display7.JPG |
| C:\Users\Munna\Desktop\Snaps\display8.JPG |

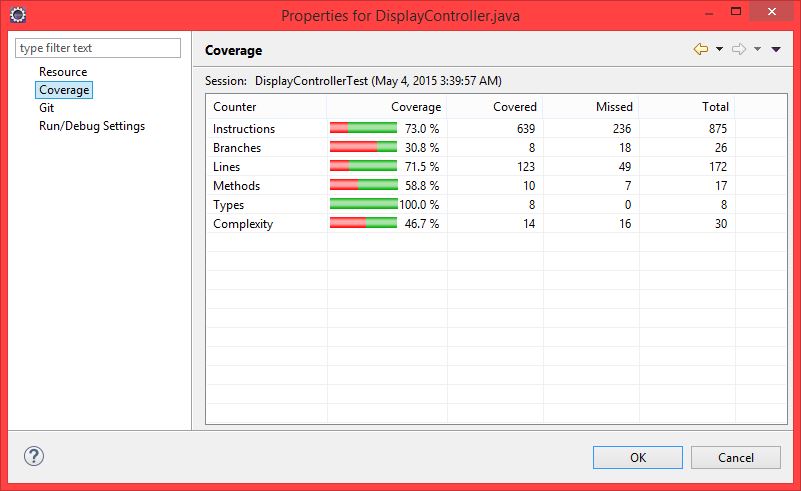
Source code of ComputeSimulationState.java with code coverage,

|  |
| --- |
| C:\Users\Munna\Desktop\Snaps\ComputeCode1.JPG |
| C:\Users\Munna\Desktop\Snaps\ComputeCode2.JPG |
| C:\Users\Munna\Desktop\Snaps\ComputeCode3.JPG |
| C:\Users\Munna\Desktop\Snaps\ComputeCode4.1.JPG |
| C:\Users\Munna\Desktop\Snaps\ComputeCode4.JPG |
| C:\Users\Munna\Desktop\Snaps\ComputeCode5.JPG |

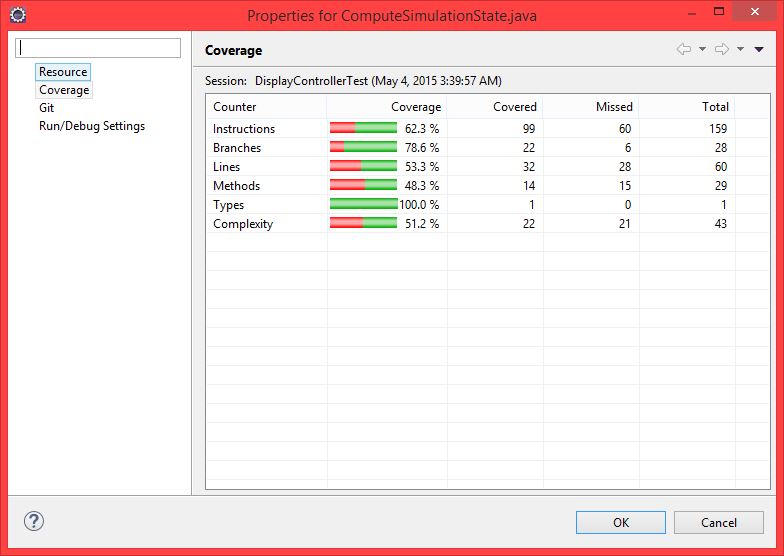
1. Coverage report of DisplayControllerTest.java



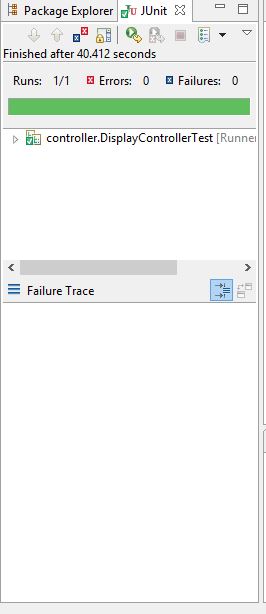
Coverage report of DisplayController.java,



Coverage report of ComputSimulationState.java,



1. JUnit test result successful notification of DisplayControllerTest.java.



1. Test Summary
2. Here, we are testing 8 Display Scenarios comparing Display Scenarios implemented in PART 2 of project to display the scenarios generated by code.

Test case Inputs: Speed, Gear Down, Altitude, Time

Test Case Outputs: Gear Not Down alarm, Gear Air Speed alarm, Air Brake warning, Gear Override warning, Gear-up command.

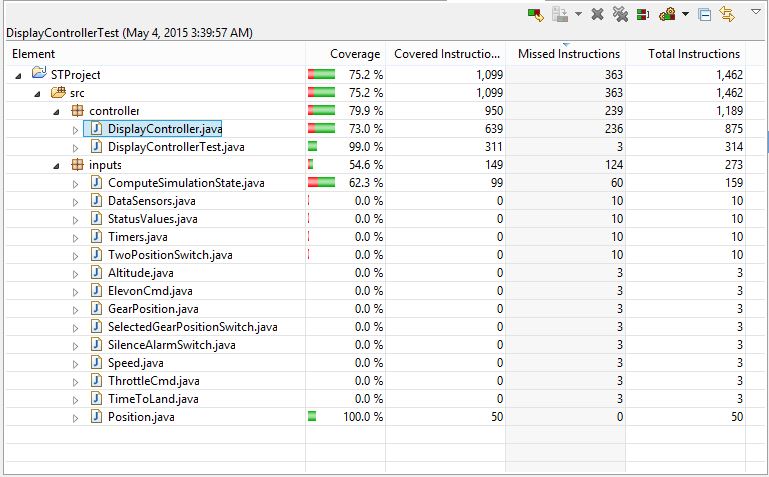
Display Scenarios Test Oracle

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Display Scenario | Speed | Gear Down | Altitude | Time | Gear Not Down alarm | Gear Air Speed alarm | Air Brake warning | Gear Override warning | Gear-up command |
| 1 | 249 | N | 1500 | 59 | X |  |  |  |  |
| 2 | 401 | N | 1500 | 59 | X |  | X |  |  |
| 3 | 401 | Y | 1500 | 59 |  | X | X | X | X |
| 4 | 401 | Y | 1500 | 121 |  | X |  | X | X |
| 5 | 301 | Y | 1500 | 59 |  | X | X |  |  |
| 6 | 301 | Y | 1500 | 121 |  | X |  |  |  |
| 7 | 250 | Y | 1500 | 59 |  |  | X |  |  |
| 8 | 415 | N | 1025 | 130 |  |  |  |  |  |

1. **Test Results**

|  |  |
| --- | --- |
| PART 2 Outputs | Program Output |
| C:\Users\Munna\Desktop\Snaps\dsx1.JPG | C:\Users\Munna\Desktop\Snaps\dsnew1.JPG |
| C:\Users\Munna\Desktop\Snaps\dsx2.JPG | C:\Users\Munna\Desktop\Snaps\dsnew2.JPG |
| C:\Users\Munna\Desktop\Snaps\dsx3.JPG | C:\Users\Munna\Desktop\Snaps\dsnew3.JPG |
| C:\Users\Munna\Desktop\Snaps\dsx4.JPG | C:\Users\Munna\Desktop\Snaps\dsnew4.JPG |
| C:\Users\Munna\Desktop\Snaps\dsx5.JPG | C:\Users\Munna\Desktop\Snaps\dsnew5.JPG |
|  |  |
| C:\Users\Munna\Desktop\Snaps\dsx6.JPG | C:\Users\Munna\Desktop\Snaps\dsnew6.JPG |
|  |  |
| C:\Users\Munna\Desktop\Snaps\dsx7.JPG | C:\Users\Munna\Desktop\Snaps\dsnew7.JPG |
|  |  |
| C:\Users\Munna\Desktop\Snaps\dsx8.JPG | C:\Users\Munna\Desktop\Snaps\dsnew8.JPG |

1. In our code, we are running all 8 display scenarios with a time lag (Thread.sleep) of 10sec each in a single run. We have all display scenarios matched with Test oracle. 100%Pass and 0% failed.
2. We have 73% overall coverage, with 27% shortfall, because in DisplayController class there is code related to UI button click actions that aren’t performed while simulating the 8 display scenarios. Hence that part of the code won’t be covered. The same applies for the code in the ComputeSimulationState class as mentioned earlier above and in part-1.



1. In Scenarios 3 and 4 the ‘Landing Gear Override warning’ is turned on which results in automatically raising the landing gear position. The scenario is implemented in the simulation, but this change in the gear position from down to up isn’t shown in display scenarios above as it is a snapshot of the scenario when the landing gear override warning is displayed. To view the same, we can run the simulation and modify the values in the UI to simulate the condition.
2. We have added an additional functionality of controlling the altitude of the plane. The pilot can increase/decrease the altitude by pressing the ‘+’ and ‘-‘ buttons beside the Altitude label in the UI. This is an additional feature, hence we haven’t included the same in the snapshot of the slides as that doesn’t have any impact on the outcome of the display scenarios.
3. Other items of discussion:
4. We have implemented full working landing simulation of aircraft which displays the altitude, speed, time until landing as well as different alarms/warnings, and performs actions, if any are needed. It also checks for valid landing conditions i.e. TimeUntilLanding = 0 and speed =0 and altitude = 0 and displays ‘Landed’ else displays ‘Failed’ on the UI.
5. We had some initial difficulties with understanding the overall structure and flow while beginning the development. Also it involved an initial learning curve for developing UI using JFrames.
6. N/A
7. The project would be useful for the future classes too as it has helped gain practical experience w.r.t. Junit and Jacoco that are widely used in industry. A suggestion for the next project would be building a simulator for a driverless car. That could have various challenges and testing requirements.