Exercise_03

Ramona Walker, Dominik Johann Arnold, Mark Woolley, Otto Buck

November 2022

Notes regarding testing

- **JUnit Version**:
  Since we ran into problems with running the jupyter JUnit Version higher than 5.8+, we decided to use JUnit 5.8.2.

- **Branch coverage and assert statements**:
  We realized that the branch coverage in our JUnit setup is negatively affected by assert statements in methods as used by Design by Contract. Even though the assert statement is covered by tests, it is always marked as "partially covered". See figure 2 for an example from class Tableau, where branch coverage is low (66%), although all lines are covered by tests.

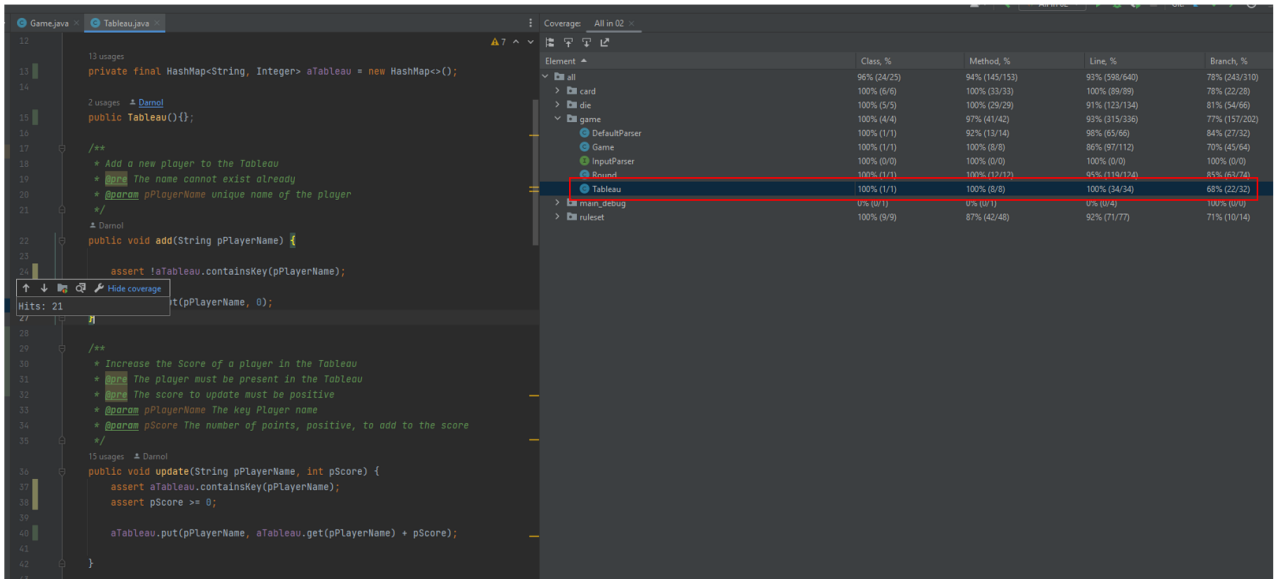

Figure 1: Example branch coverage in class Tableau

We tried to cover the assert statements by adding explicit tests checking that an assertion is indeed thrown. This helps the branch coverage a little bit, but does not lead to fully covered flags for the lines with assert. See figure **??** for an example from the tests of the Bonus Ruleset. We didn't include those tests extensively, because it is very time consuming, clutters our test code and, most importantly, is not advised by the book. In our understanding, assert statements do not have to be covered by tests, because you expect them to fail if the input is invalid.

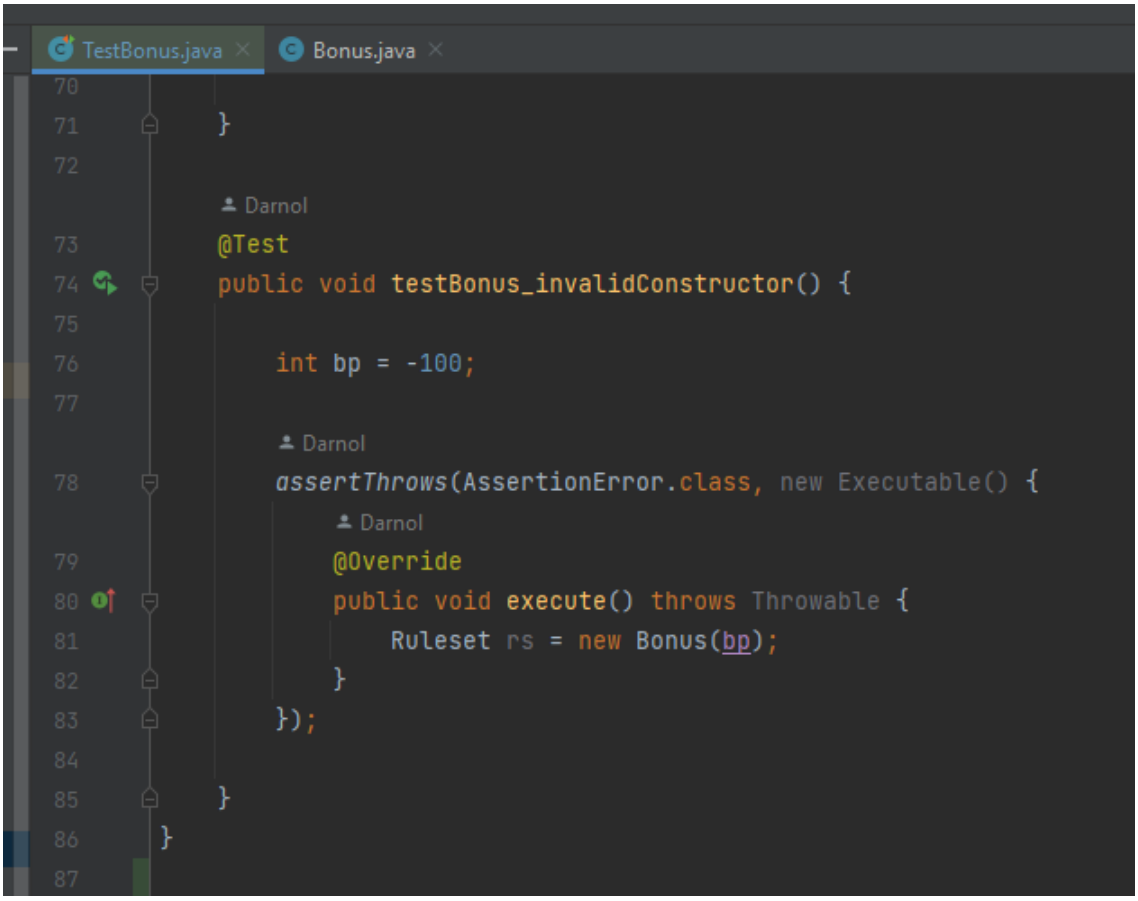

Figure 2: Example test which tests for AssertionError

Overall, we still reached the 70% branch coverage, but the branch coverage could be much higher.

- **Testing with user inputs**:
  We decided to test functions that require user input by using an *ByteArrayInputStream* which supplies the *InputParser* class with predefined answers of the player. Combined with the debug mode, it was possible to test specific constellations of the game.

- **Testing random variables**:
  One problem we couldn't solve to our full satisfaction was the testing of the rolling of the remaining dice. We wanted to write a test that makes sure that the method *DiceSet.rollRemaining* correctly re-rolls the dice. Since we used a Flyweight pattern for the DiceSet, there is only one DiceSet instance and our debug DiceSet does always roll ONE, hence it's not suitable to test whether a re-rolling produces a different outcome.
  Our solution is a DiceSet where we re-roll all six dice two times and compare the combinations of the two sets. While it is possible that two consecutive rolls with six dice produce the exact same set of combinations, it's rather rare. Never the less, the test *testRollRemaining* in the file *TestDiceSet* can possibly fail. If that were to happen, one needs to rerun the test and given the tiny possibility that this happens twice in a row, the test should pass. Should the test fail twice in a row, one can assume that the test does indeed fail because the implementation of the method is flawed.