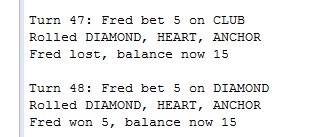
Bug Fixing Journal

# Bug 1 -

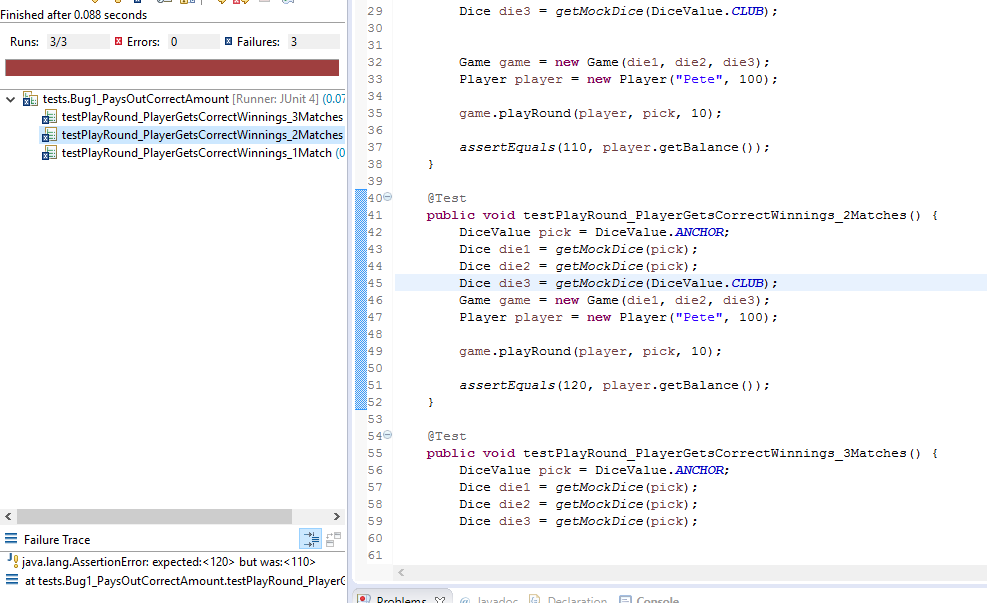
## User Test

Click Run and observe that when Fred wins, his balance does not increase. For example:



At this point I briefly inspected the code to work out how it could be mocked/tested. The bug will most likely in the Game or Player classes. To test reliably, I need to create a test scenario in which the player always wins. The logical way to do this would be to rig the dice rolls.

## Test Output



As expected due to the bug report, the tests all failed.

## Debug log

Note that before I start, it is quite obvious from the results of the tests that the game is not giving back the bet amount when the player wins. I’m going to pretend it’s not that simple.

I start by looking at the code Player class to see how the balance can be effected. The first thing I want to ensure is that I’m not simply being shown the wrong balance.

**Hypothesis:** getBalance is not showing the balance correctly.

**Test (Junit):** Ensure a player created with $100 returns 100 from getBalance.

**Result:** pass

**Conclusion:** getBalance works as intended.

Other than the constructor which was proven to work in the first test, 2 methods can modify the balance: takeBet and receiveWinnings. Since both of these functions are used in the playRound method, I will start by ensuring that these 2 functions work properly.

**Hypothesis:** takeBet is taking too much away from the balance.

**Test (Junit):** Ensure that if the Player starts with $100 and a $10 bet is taken, they end up with $90.

**Result:** pass

**Conclusion:** takeBet works as intended.

**Hypothesis:** receiveWinnings is not adding to the balance properly.

**Test (Junit):** Ensure that if the Player starts with $100 and a $10 of winnings are received, they end up with $110.

**Result: pass**

**Conclusion:** receiveWinnings works as intended.

This leads me to believe that the player class is functioning properly, so the problem must be the input it is given. The first time it is used on line 33 ( player.takeBet(bet) ), the input comes straight from the method argument, so there’s not a lot of room for error.

The second time, player.receiveWinnings(winnings) is passed (or ‘dependent on’) the ‘winnings’ local variable which is defined with int winnings = matches \* bet;. This would result on the player receiving his bet back when he wins, which is what the bug is doing.

**Hypothesis:** winnings variable is not being calculated correctly.

**Test:** Set a breakpoint on line 33 for 10 winnings at line 46 (player.receiveWinnings(winnings)) with 2 matches and a bet of 10, and ensure winnings == 30. If winnings = 20, the hypothesis is correct.

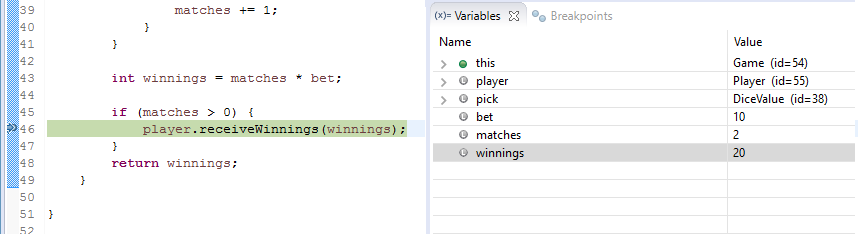
**Result: FAIL. Winnings = 20.**

**Conclusion:** winnings is not being calculated correctly.

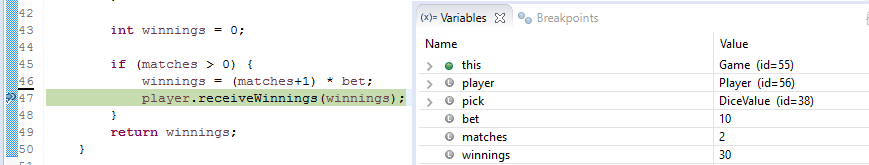
Winnings should be 0 if there are no matches, or (matches+1) \* bet if there is any. To ensure I don’t mess up the logic, at this point I’ll add an automated test for when there are no matches.

## Before and After Screenshots

The following is what the code and values look like before the bug is fixed:

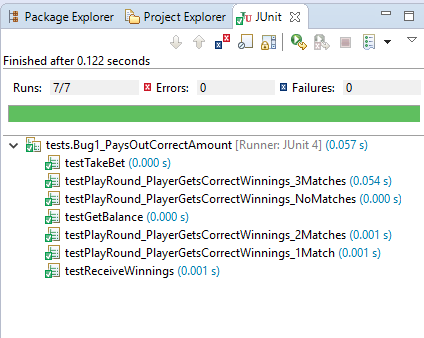


And after the bug is fixed:



## Test Output from Fixed Bug

All tests ran successfully with the bug fix included:



## Output of Fixed Bug

Output for a winning bet is given below:

