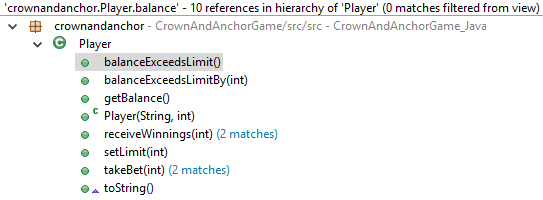
Debugging Log

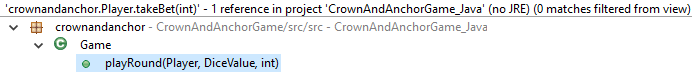
# Bug 1

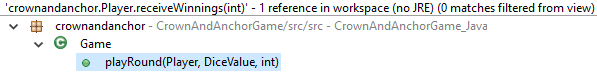
The data that is being updated incorrectly is the player’s balance. So after confirming in Main.java that it is stored in the player class, I read the methods of Player.java.

‘balance’ is a class variable integer of the player class, and its references appear here:



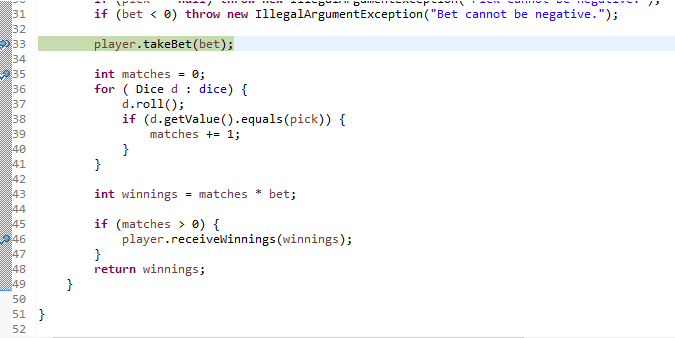
The three times that balance is adjusted are: in the constructor (naturally), in takeBet() and receiveWinnings(). Thanks to intuitive method names I can infer that takeBet() is called before the player’s die pick is compared with the results, and receiveWinnings() is called when the results fall in the player’s favour.





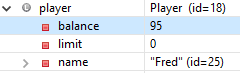
Fortunately both are called only in the one and same method: Game.playRound().

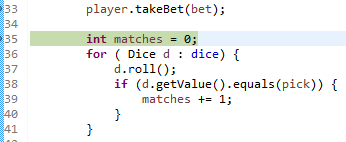
So in all bar one scenario, Player.balance is updated properly. The only time that balance is updated in the normal flow is in Game.playRound(). The bug is most likely to be found here.



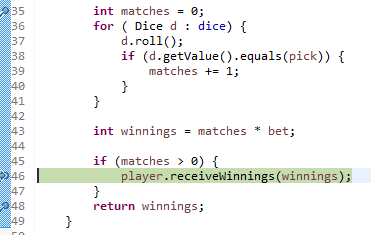


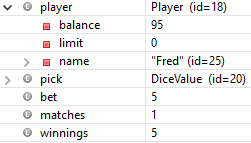
Above is the state of the program at the first call to Game.playRound(). player.balance is set to 100 as it was initialised.



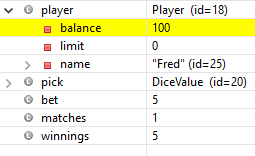


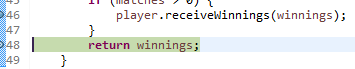
The next breakpoint is just after the first takeBet() call. bet is 5, so takeBet() has subtract 5 from 100. *The player puts down 5 units on their side of choice.*





The player has found 1 match, and so their winnings ought to give them 2:1 odds, according to the rules of Crown and Anchor.





In the next update, 5 has been added to the player’s balance, returning it to 100. We have just seen Bug 1 in action.

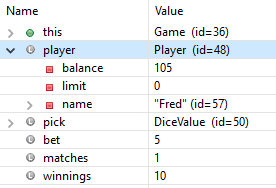
This bug is due to a misunderstanding that a net increase of 5 units is the same as adding 5 to the balance after the bet has been placed. The issue is resolved by increasing how much is won by the player on a success.

int winnings = matches \* bet;

becomes

int winnings = (matches + 1) \* bet;

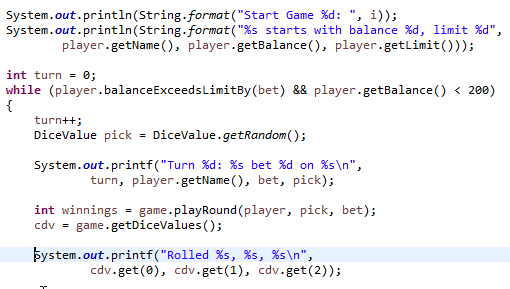
This is safe because the condition for progressing to player.receiveWinnings() is that matches > 0. This was functional before and has not been changed.



This is the new output of a 1-match win. The initial state was 100, 5 was subtracted as the bet was placed, and then 10 was added after the results were checked.

# Bug 2

The game ending as described in bug 2 occurs at the end of a while loop in the Main class. It is easy to look at the conditions of this class and determine the error.



This is the beginning of the while loop in question. It is immediately apparent that the problem lies not on the right side of the AND operator, as the bug specifies the game ending when the player’s balance is 5 (5 is less than 200, so the right side of the AND operation is true in that case).

In the circumstances listed in Bug 2, limit is 0 and (it is implied that) bet is 5. With that in mind, we examine Player.balanceExceedsLimitBy():

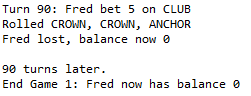


The game ends when this method returns false. The intended return value when limit = 0 and bet (or amount) = balance = 5 is true.

Balance (5) – amount (5) = 0

0 > 0 = false

The solution to this bug is to change the Boolean operator in Player.balanceExceedsLimitBy() to be >=.



Here are the intended results of the bug fix: the player now loses at 0 and not 5.

*There is a similar bug in the Player class, in the balanceExceedsLimit() method. This method is unused, but I fixed it nonetheless.*