Bug #1 – Player not receiving correct winnings

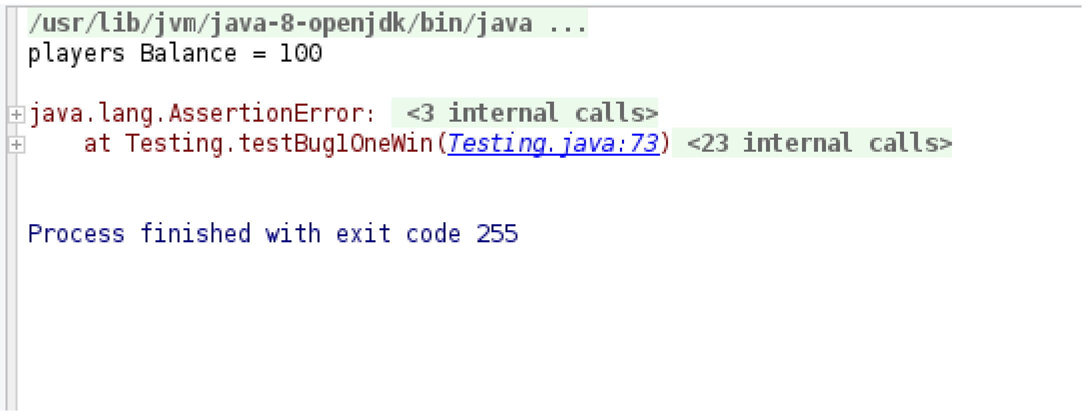
Possible Cause:

Likely within how the game detects winnings and adds them to the player

Testing

Created some mock die to ensure player always won, tried with 2 wins and 1 win with a verification that the players balance at the end of the game was the correct amount more(using an initial balance of 100 and bets of $10).

Test with one Win

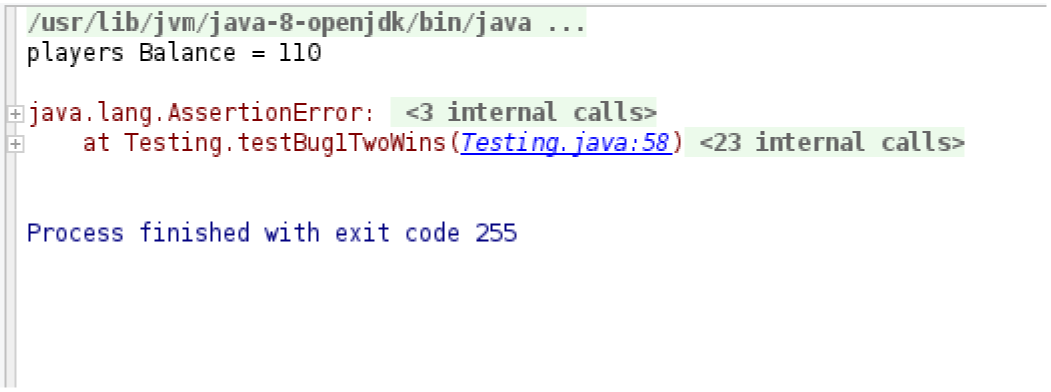


```
/usr/lib/jvm/java-8-openjdk/bin/java ...
players Balance = 100

java.lang.AssertionError:  <3 internal calls>
    at Testing.testBug1OneWin(Testing.java:73) <23 internal calls>


Process finished with exit code 255
```

Test with 2 Wins:

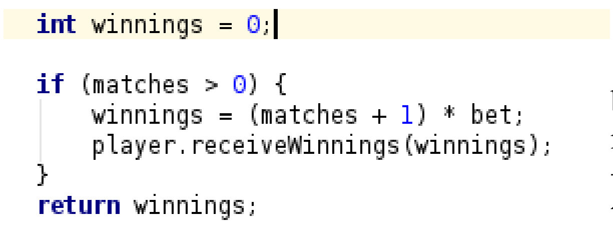


```
/usr/lib/jvm/java-8-openjdk/bin/java ...
players Balance = 110

java.lang.AssertionError:  <3 internal calls>
    at Testing.testBug1TwoWins(Testing.java:58) <23 internal calls>


Process finished with exit code 255
```

In both cases the test showed that the players balance was 10 less than what it should be, this means the game is only paying out the winnings and not returning the initial bet along with the payout. This goes for all possible winnings and not just one win.



```
int winnings = 0;

if (matches > 0) {
    winnings = (matches + 1) * bet;
    player.receiveWinnings(winnings);
}
return winnings;
```

The Fix:

First of all, winnings was initialised at zero to prevent breaking the return function and then if the number of matches was > 0 the winning became number of matches + 1 to add the initial bet to the winnings. Upon rerunning tests they passed with the fix.

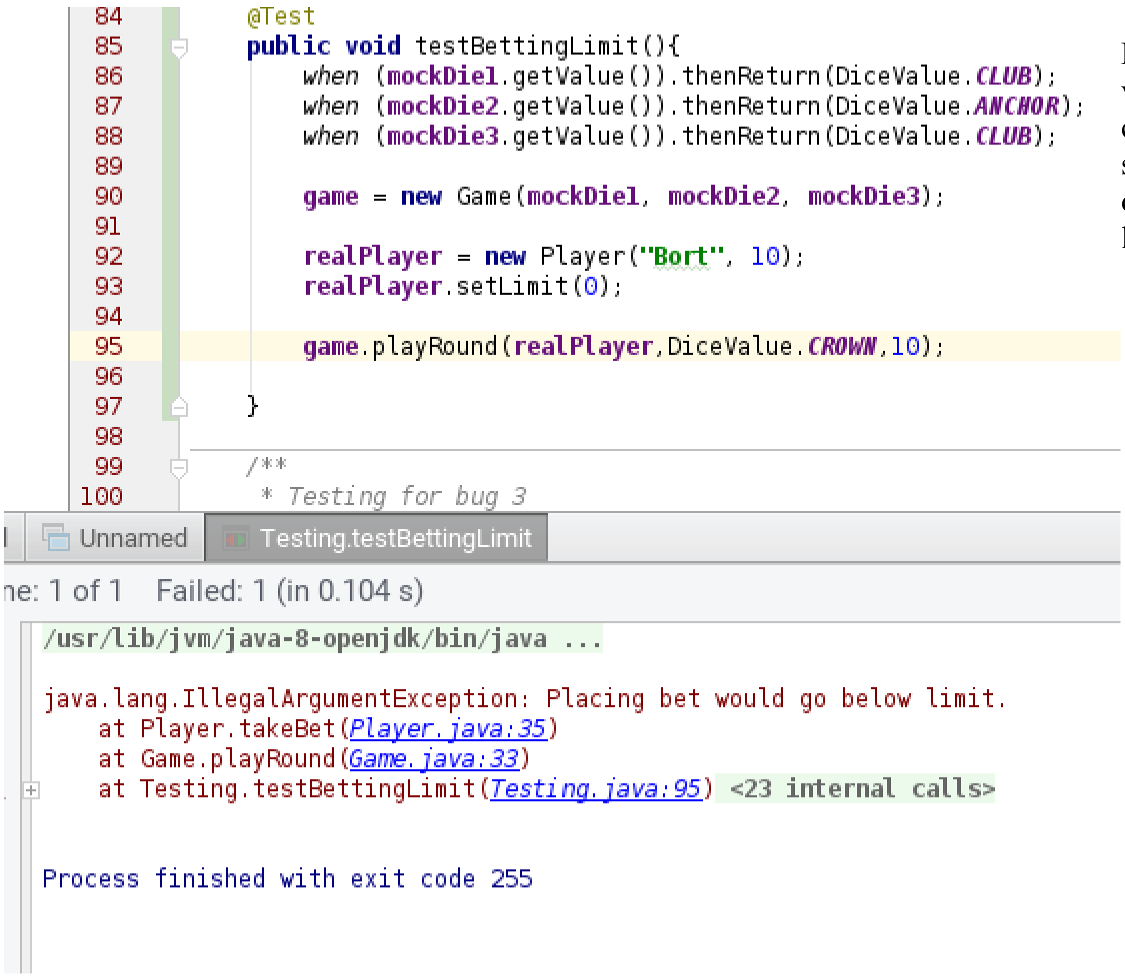
Bug #2 Player cannot reach the betting limit

Possible cause

Something is likely going on with the checks to see if the player is allowed to make a bet, given the rules state there is a board of possible bets for the player to choose from I'm assuming the smallest bet is $5

Test

Let's try to make a bet with all with players balance at $5 and the bet amount at $5



```java
    @Test
    public void testBettingLimit(){
        when (mockDie1.getValue()).thenReturn(DiceValue.CLUB);
        when (mockDie2.getValue()).thenReturn(DiceValue.ANCHOR);
        when (mockDie3.getValue()).thenReturn(DiceValue.CLUB);

        game = new Game(mockDie1, mockDie2, mockDie3);

        realPlayer = new Player("Bort", 10);
        realPlayer.setLimit(0);

        game.playRound(realPlayer,DiceValue.CROWN,10);

    }

    /**
     * Testing for bug 3
```

```
/usr/lib/jvm/java-8-openjdk/bin/java ...

java.lang.IllegalArgumentException: Placing bet would go below limit.
    at Player.takeBet(Player.java:35)
    at Game.playRound(Game.java:33)
    at Testing.testBettingLimit(Testing.java:95) <23 internal calls>


Process finished with exit code 255
```

Fiddling around with these values showed that you cannot bet your remaining spendable balance and can only bet when balance - limit – bet > 0

Fix

```java
public boolean balanceExceedsLimitBy(int amount) {
    return (balance - amount >= limit);
}
```

Simply change the greater than statement to a greater or equal to, thus allowing the bet to proceed.

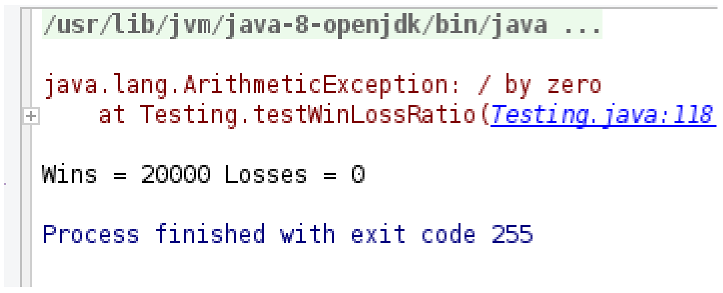
Bug #3 – win ratio is incorrect

Possible cause

I honestly have no idea, so to start with lets just run a bunch of games to see what the actual win/loss ratio is.

Testing

Create some dice and a mock player that always has balance remaining then simply play the game 10000 times to get a small sample of many games

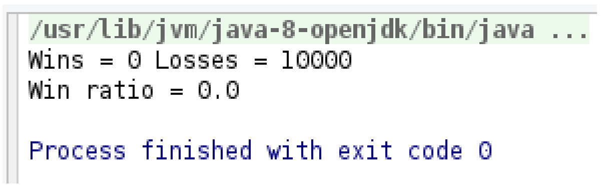


```
/usr/lib/jvm/java-8-openjdk/bin/java ...

java.lang.ArithmeticException: / by zero
    at Testing.testWinLossRatio(Testing.java:118

Wins = 20000 Losses = 0

Process finished with exit code 255
```



```
/usr/lib/jvm/java-8-openjdk/bin/java ...
Wins = 0 Losses = 10000
Win ratio = 0.0

Process finished with exit code 0
```

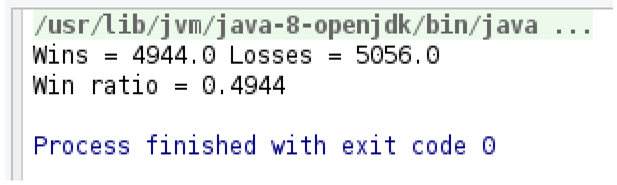
Now we know what's going on… We get one random number and that's it, this can also be shown running the provided main class, the game returns the same set of dice each play.
Further, testing utilising console output within different functions(i.e. in the function that gets the random dice value in DiceValue etc.) led to the realisation that the function roll() in the dice class only returns the random value and will not set the value that getValue returns. The only time the value is set is in the constructor.

The Fix

Add "value = DiceValue.getRandom()" to the dice roll function, return the new value instead of another random value

Part 2

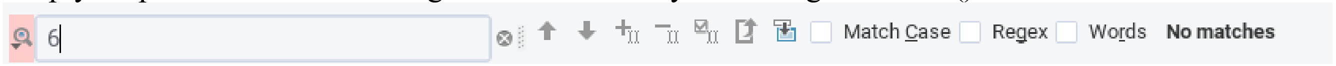
With this fixed the win ration now equals around 0.49, outside of the wanted ratio.



```
/usr/lib/jvm/java-8-openjdk/bin/java ...
Wins = 4944.0 Losses = 5056.0
Win ratio = 0.4944

Process finished with exit code 0
```

One possible issue is that when using an Integer a floating point is always rounded down, to test this I simply outputted the random integer to console every time the getRandom() in DiceValue was called

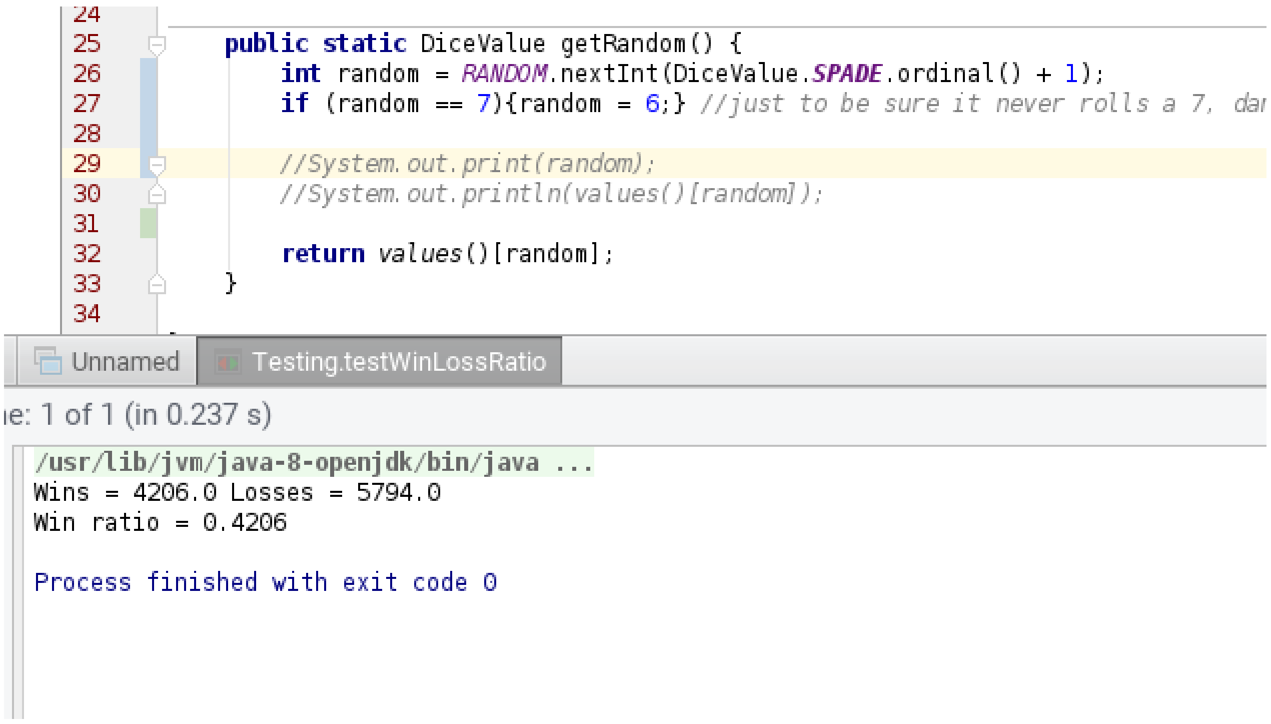


6 Match Case Regex Words No matches

Searching the output confirmed these suspicions as the number "6" did not appear in the output once, therefore the game was being played with some sort of 5 sided dice.

The Fix

Simply added 1 to the count for what the maximum value that the random integer retrieves, also added a check to make sure that in some sort of incredibly unlucky event a 7 is somehow generated(is this actually possible?). The game now selects from all dice values, as a result the average wins/losses is correct.



```java
    public static DiceValue getRandom() {
        int random = RANDOM.nextInt(DiceValue.SPADE.ordinal() + 1);
        if (random == 7){random = 6;} //just to be sure it never rolls a 7, da

        //System.out.print(random);
        //System.out.println(values()[random]);

        return values()[random];
    }
```

```
/usr/lib/jvm/java-8-openjdk/bin/java ...
Wins = 4206.0 Losses = 5794.0
Win ratio = 0.4206

Process finished with exit code 0
```