Debugging Log Book

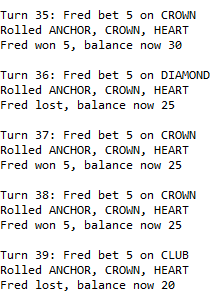
# Bug 1

## Hypothesis:

The program appears to be running correctly and as it should as the payouts are matching what is outlined in the rules, however, it could be that the user is becoming confused as the system does not advise the user on the balance change when they make their bet. This leaves the user feeling as though they are not winning anything.

## Evidence to support the hypothesis

As shown below Fred bets 5 on Crown and won 5 credits which made his balance 30, he then bets another 5 on Diamond and loses which changes his balance to 25, this shows that the bet is successfully being taken for the bet as he did not win anything. In the next game Fred bets 5 credits on Crown and wins 5 and his balance stays at 25. This indicates that the application is working correctly as Fred has received the 5 credits as per the rules.



## Tracing

In order to prove my hypothesis I created 4 tests in JUnit under the test class Bug1Test, each test was run 3 times. The Tests were as follows:

* Test 1: Test that 1 match repays the betted amount as per the game rules
* Test 2: Test that 2 matches repays 2 times the betted amount as per the game rules
* Test 3: Test that 3 matches repays 3 times the betted amount as per the game rules
* Test 4: Test that 0 matches does not repay the betted amount as per the game rules.
* Test 5: Test to show the implemented change.

The test outputs for Bug1Test are:

**Test 1:**

Test 1 - Single Match

Start Game 0:

Turn 1: Fred bet 5 on CROWN

Rolled CROWN, ANCHOR, HEART

Fred won 5, balance now 100

Test 1 - Single Match

Start Game 1:

Turn 1: Fred bet 5 on CROWN

Rolled CROWN, ANCHOR, HEART

Fred won 5, balance now 100

Test 1 - Single Match

Start Game 2:

Turn 1: Fred bet 5 on CROWN

Rolled CROWN, ANCHOR, HEART

Fred won 5, balance now 100

**Test 2:**

Test 2 - Double Match

Start Game 0:

Turn 1: Fred bet 5 on CROWN

Rolled CROWN, CROWN, HEART

Fred won 10, balance now 105

Test 2 - Double Match

Start Game 1:

Turn 1: Fred bet 5 on CROWN

Rolled CROWN, CROWN, HEART

Fred won 10, balance now 110

Test 2 - Double Match

Start Game 2:

Turn 1: Fred bet 5 on CROWN

Rolled CROWN, CROWN, HEART

Fred won 10, balance now 115

**Test 3:**

Test 3 - Triple Match

Start Game 0:

Turn 1: Fred bet 5 on CROWN

Rolled CROWN, CROWN, CROWN

Fred won 15, balance now 110

Test 3 - Triple Match

Start Game 1:

Turn 1: Fred bet 5 on CROWN

Rolled CROWN, CROWN, CROWN

Fred won 15, balance now 120

Test 3 - Triple Match

Start Game 2:

Turn 1: Fred bet 5 on CROWN

Rolled CROWN, CROWN, CROWN

Fred won 15, balance now 130

**Test 4:**

Test 4 - No Match

Start Game 0:

Turn 1: Fred bet 5 on CROWN

Rolled HEART, ANCHOR, HEART

Fred lost, balance now 95

Test 4 - No Match

Start Game 1:

Turn 1: Fred bet 5 on CROWN

Rolled HEART, ANCHOR, HEART

Fred lost, balance now 90

Test 4 - No Match

Start Game 2:

Turn 1: Fred bet 5 on CROWN

Rolled HEART, ANCHOR, HEART

Fred lost, balance now 85

## Resolution:

As can be seen from the testing and outputs the application is behaving as it should in relation to the payouts. In order to resolve this issue I have added a print statement advising of the starting balance before Fred makes his bet in order to remove the player’s confusion. As this is a print statement it will not affect any of the existing code.

## Output after resolution implemented:

As can be seen in the output below the application is now displaying the balance to the user after the change has been implemented.

Test 5 - Implemented Change

Start Game 0:

Starting Balance: 100

Turn 1: Fred bet 5 on CROWN

Rolled HEART, ANCHOR, HEART

Fred lost, balance now 95

Test 5 - Implemented Change

Start Game 1:

Starting Balance: 95

Turn 1: Fred bet 5 on CROWN

Rolled HEART, ANCHOR, HEART

Fred lost, balance now 90

Test 5 - Implemented Change

Start Game 2:

Starting Balance: 90

Turn 1: Fred bet 5 on CROWN

Rolled HEART, ANCHOR, HEART

Fred lost, balance now 85

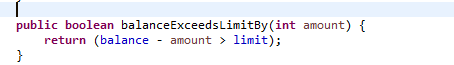
# Bug 2

## Hypothesis:

I believe that bug 2 is occurring in the balanceExceedsLimitBy method within the Player class, this is because the Boolean expression is returning false before the balance is getting to 0 due to use of greater than rather that greater than equal to.

## Evidence to support the hypothesis:

As shown below, the method is returning true if the balance minus amount is greater than limit, this means that If the balance minus amount is equal to the limit it will not run.



## Tracing:

In order to prove my hypothesis I created 2 tests in JUnit under the test class Bug2Test, the first test was set to loop 3 times and the second test once. The Tests were as follows:

* Test 1: Test that starts with a balance of 15 and calls 3 losing games to verify that the game does indeed end with an error. As can be seen the 3rd test fails when Fred tries to make a bet of 5 on crown which would leave him with a 0 balance.
* Test 2: Test that starts with a balance of 5 and overrides the balanceExceedsLimitBy method to return false so that the method is by passed in order to prove that this method is causing the issue.

**Test 1:**

Test 1 - Verify Hypothesis

Start Game 0:

Fred starts with balance 15, limit 0

Turn 1: Fred bet 5 on CROWN

Rolled HEART, ANCHOR, HEART

Fred lost, balance now 10

Test 1 - Verify Hypothesis

Start Game 1:

Fred starts with balance 10, limit 0

Turn 1: Fred bet 5 on CROWN

Rolled HEART, ANCHOR, HEART

Fred lost, balance now 5

Test 1 - Verify Hypothesis

Start Game 2:

Fred starts with balance 5, limit 0

Turn 1: Fred bet 5 on CROWN

**Test 2:**

Test 2 - Verify game reaches balance of 0

Start Game 1:

bet 5 on CROWN

Rolled HEART, ANCHOR, HEART

Fred lost, balance now 0

## Resolution:

As can be seen from the test the application is failing when the players balance is reaching 5 credits due to the method balanceExceedsLimitBy. My resolution is to implement the changes I suggested in my hypothesis which is to change the logic from greater than to greater than equals to. This should not affect other parts of the program as this method is only used as a check for this part of the game.

## Output after resolution implemented:

As can be seen in the output below the application is now going to the balance of 0 after the change has been implemented.

Test 1 - Verify Hypothesis

Start Game 0:

Fred starts with balance 15, limit 0

Turn 1: Fred bet 5 on CROWN

Rolled HEART, ANCHOR, HEART

Fred lost, balance now 10

Test 1 - Verify Hypothesis

Start Game 1:

Fred starts with balance 10, limit 0

Turn 1: Fred bet 5 on CROWN

Rolled HEART, ANCHOR, HEART

Fred lost, balance now 5

Test 1 - Verify Hypothesis

Start Game 2:

Fred starts with balance 5, limit 0

Turn 1: Fred bet 5 on CROWN

Rolled HEART, ANCHOR, HEART

Fred lost, balance now 0

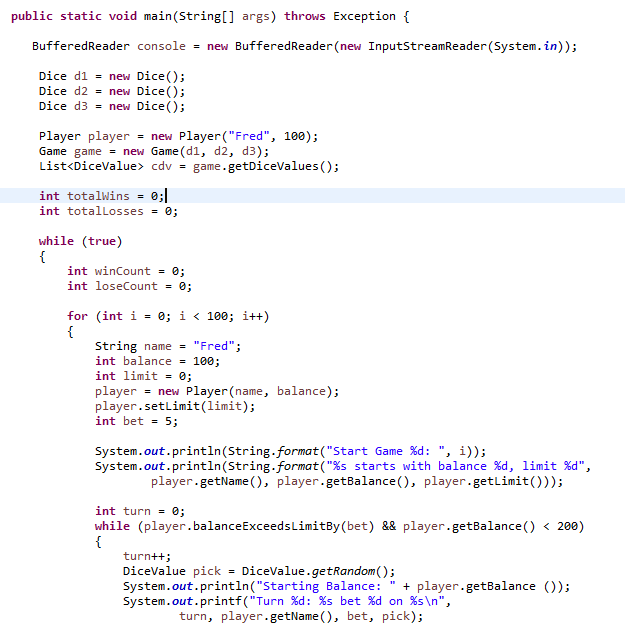
# Bug 3

## Hypothesis:

I believe that bug 3 is being caused due to the dice not being recreated in the Main class. In Main it appears that the dice are only created once at the beginning when the application is run, this means that the dice stay the same for all of the games in each run, rather than being randomly generated for each game.

## Evidence to support the hypothesis:

As can be seen below the variables d1, d2, d3 are created at the beginning of the main method and then are added to the game object. This all occurs outside of the inner while loop which means that the dice are not being recreated for each game.



## Tracing:

In order to prove my hypothesis I created 2 tests in JUnit under the test class Bug3Test, the first test was set to loop 3 times and the second test once. The Tests were as follows:

* Test 1: Test is a run of the main method as is without any changes to verify that the bug is definitely in the Main class and main method. As you can see from the run the win rate is a lot higher than 42% and the rolled dice are repeating.
* Test 2: is with the changes to the main method as stated in the hypothesis, as can be seen this change fixed the dice remaining the same with each turn of the game. Also the percentage is closer to the 42% however it is still slightly high at approximately 48%.

**TEST 1:**

Starting Balance: 5

Turn 64: Fred bet 5 on CROWN

Rolled ANCHOR, CLUB, CROWN

Fred won 5, balance now 5

Starting Balance: 5

Turn 65: Fred bet 5 on ANCHOR

Rolled ANCHOR, CLUB, CROWN

Fred won 5, balance now 5

Starting Balance: 5

Turn 66: Fred bet 5 on HEART

Rolled ANCHOR, CLUB, CROWN

Fred lost, balance now 0

66 turns later.

End Game 99: Fred now has balance 0

Win count = 2984, Lose Count = 2000, 0.60

Overall win rate = 59.9%

**Test 2:**

Starting Balance: 15

Turn 35: Fred bet 5 on DIAMOND

Rolled CROWN, HEART, CLUB

Fred lost, balance now 10

Starting Balance: 10

Turn 36: Fred bet 5 on DIAMOND

Rolled ANCHOR, ANCHOR, CROWN

Fred lost, balance now 5

Starting Balance: 5

Turn 37: Fred bet 5 on HEART

Rolled DIAMOND, DIAMOND, ANCHOR

Fred lost, balance now 0

37 turns later.

End Game 99: Fred now has balance 0

Win count = 2354, Lose Count = 2522, 0.48

Overall win rate = 48.3%

## New Hypothesis:

I now believe that there is also an issue with the results being returned when the getRandom method in the DiceValue class generates the dice. I believe that the RANDOM.nextInt value needs to have a + 1 to rectify the issue.

## Evidence to support the hypothesis:

After a for loop completed 1000 loops of the getRandom method, the output was as follows:

Number of Hearts: 208

Number of Clubs: 180

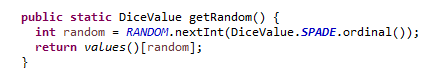
Number of Anchors: 195

Number of Crowns: 207

Number of Diamonds: 210

Number of Spades: 0

As can be seen the application is not generating all of the dice values that it should be, it is not reaching the spade. This is due to the RANDOM.nextInt using the ordinal positioning, which causes the generated number to be up to but not actually the last ordinal position.



## Tracing:

In order to prove my hypothesis I created a test in JUnit under the test class Bug3Test, the first test was used as evidence to support my hypothesis and looped 1000 times using the getRandom method to get the values. Due to the test identifying the bug I will use this test after making a change to the getRandom method to verify the fix.

**Test 1:**

Number of Hearts: 208

Number of Clubs: 180

Number of Anchors: 195

Number of Crowns: 207

Number of Diamonds: 210

Number of Spades: 0

**Test 2:**

Number of Hearts: 168

Number of Clubs: 169

Number of Anchors: 155

Number of Crowns: 169

Number of Diamonds: 169

Number of Spades: 170

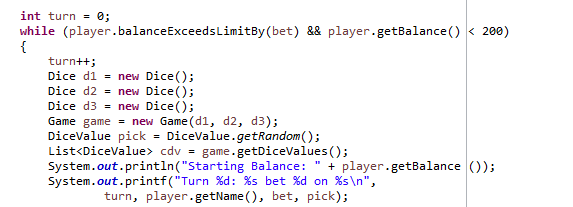
## Resolution:

Implement the +1 so that the method random method will reach spades. This will now mean that there are more faces on the dice and will fix the issues with the game win percentages. Also implement the changes listed in the first hypothesis as well

**Change 1:**



**Change 2:**



## Output after resolution implemented:

As can be seen in the output below the application is now producing a 42% win rate after the changes have been implemented, as well as the differing dice faces for each roll.

Starting Balance: 10

Turn 45: Fred bet 5 on ANCHOR

Rolled HEART, DIAMOND, HEART

Fred lost, balance now 5

Starting Balance: 5

Turn 46: Fred bet 5 on SPADE

Rolled CLUB, CROWN, HEART

Fred lost, balance now 0

46 turns later.

End Game 99: Fred now has balance 0

Win count = 1649, Lose Count = 2297, 0.42