



ITC205

Assignment 4

Bug Fixing Journal

Lecks Chester

11254915

Bug Fixing Journal

Contents

Bug 1 – Game Does Not Pay Out at Correct Levels	2
User Test	2
Test Output (Failing)	2
Debug Log	3
Before and After Screenshots	4
Test Output (Fixed)	4
User Test (Fixed)	5
Bug 2 – Player cannot reach betting limit	5
User Test	5
Test Output (Failing)	6
Debug Log	6
Before and After screenshots	6
Test output (Fixed).....	7
User Test Output (Fixed)	7
Bug 3 – Odds in the game do not appear to be correct	7
User Test	7
Test Output (Failing)	8
Debug Log	8
Before and After Screenshots/Fixed Test Output	10
User Test Output (Fixed)	12
Additional Bugs.....	12
Notes	12
References	12

Bug 1 – Game Does Not Pay Out at Correct Levels

User Test

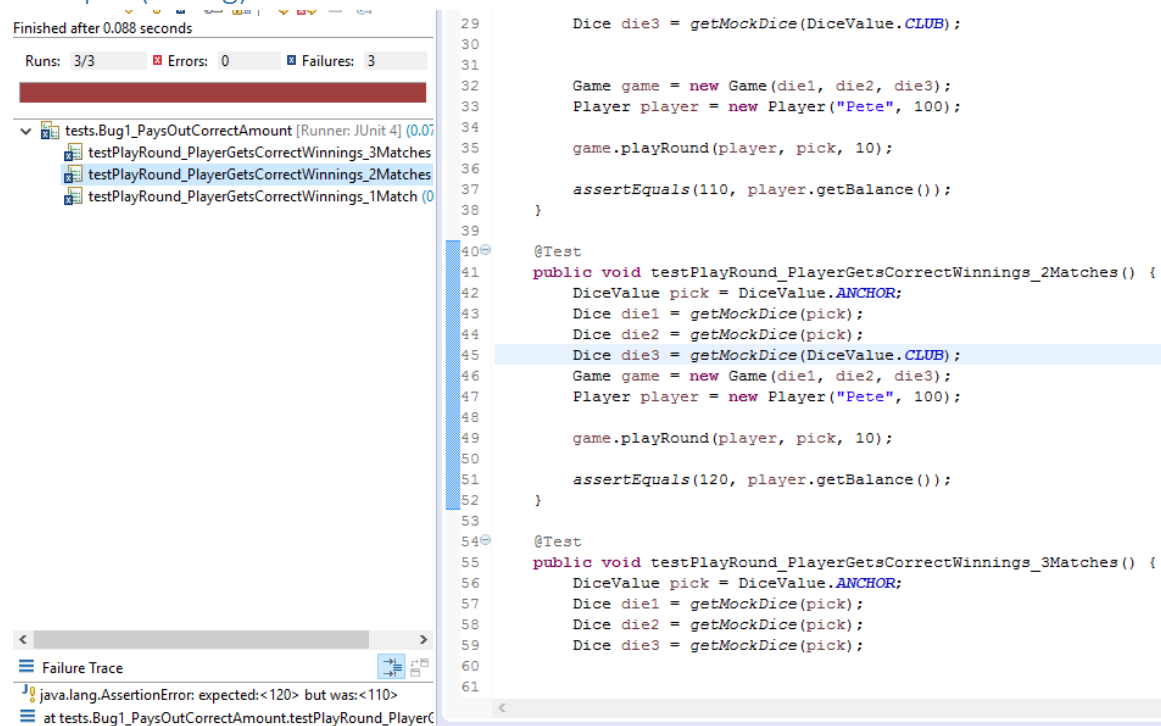
Click Run and observe that when Fred wins, his balance increases appropriately. This test fails, for example:

```
Turn 47: Fred bet 5 on CLUB
Rolled DIAMOND, HEART, ANCHOR
Fred lost, balance now 15

Turn 48: Fred bet 5 on DIAMOND
Rolled DIAMOND, HEART, ANCHOR
Fred won 5, balance now 15
```

*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 (Failing)



```
Finished after 0.088 seconds
Runs: 3/3 Errors: 0 Failures: 3

tests.Bug1_PaysOutCorrectAmount [Runner: JUnit 4] (0.0)
  testPlayRound_PlayerGetsCorrectWinings_3Matches
  testPlayRound_PlayerGetsCorrectWinings_2Matches
  testPlayRound_PlayerGetsCorrectWinings_1Match (0)

40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61

Dice die3 = getMockDice(DiceValue.CLUB);

Game game = new Game(die1, die2, die3);
Player player = new Player("Pete", 100);

game.playRound(player, pick, 10);

assertEquals(110, player.getBalance());

@Test
public void testPlayRound_PlayerGetsCorrectWinings_2Matches() {
    DiceValue pick = DiceValue.ANCHOR;
    Dice die1 = getMockDice(pick);
    Dice die2 = getMockDice(pick);
    Dice die3 = getMockDice(DiceValue.CLUB);
    Game game = new Game(die1, die2, die3);
    Player player = new Player("Pete", 100);

    game.playRound(player, pick, 10);

    assertEquals(120, player.getBalance());
}

@Test
public void testPlayRound_PlayerGetsCorrectWinings_3Matches() {
    DiceValue pick = DiceValue.ANCHOR;
    Dice die1 = getMockDice(pick);
    Dice die2 = getMockDice(pick);
    Dice die3 = getMockDice(pick);

    assertEquals(120, player.getBalance());
}
```

Failure Trace

```
java.lang.AssertionError: expected:<120> but was:<110>
    at tests.Bug1_PaysOutCorrectAmount.testPlayRound_PlayerC
```

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

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

Result: Pass.

Conclusion: **Player.getBalance** works as intended. Hypothesis is incorrect.

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

Hypothesis: **Player.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: **Player.takeBet** works as intended. Hypothesis is incorrect.

Hypothesis: **Player.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: **Player.receiveWinnings** works as intended. Hypothesis is incorrect.

*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:

```
39         matches += 1;
40     }
41 }
42
43 int winnings = matches * bet;
44
45 if (matches > 0) {
46     player.receiveWinnings(winnings);
47 }
48 return winnings;
49 }
50
51 }
52 }
```

Name	Value
> this	Game (id=54)
> player	Player (id=55)
> pick	DiceValue (id=38)
bet	10
matches	2
winnings	20

And after the bug is fixed:

```
42
43 int winnings = 0;
44
45 if (matches > 0) {
46     winnings = (matches+1) * bet;
47     player.receiveWinnings(winnings);
48 }
49 return winnings;
50 }
```

Name	Value
> this	Game (id=55)
> player	Player (id=56)
> pick	DiceValue (id=38)
bet	10
matches	2
winnings	30

Test Output (Fixed)

All tests ran successfully with the bug fix included:

Runs: 7/7

Errors: 0

Failures: 0

tests.Bug1_PaysOutCorrectAmount [Runner: JUnit 4] (0.057 s)

- testTakeBet (0.000 s)
- testPlayRound_PlayerGetsCorrectWinnings_3Matches (0.054 s)
- testPlayRound_PlayerGetsCorrectWinnings_NoMatches (0.000 s)
- testGetBalance (0.000 s)
- testPlayRound_PlayerGetsCorrectWinnings_2Matches (0.001 s)
- testPlayRound_PlayerGetsCorrectWinnings_1Match (0.001 s)
- testReceiveWinnings (0.001 s)

User Test (Fixed)

Output for a winning bet is given below:

```
Turn 239: Fred bet 5 on ANCHOR
Rolled DIAMOND, CLUB, CLUB
Fred lost, balance now 20

Turn 240: Fred bet 5 on DIAMOND
Rolled DIAMOND, CLUB, CLUB
Fred won 10, balance now 25
```

Bug 2 – Player cannot reach betting limit

User Test

Run the program repetitively until the player stops because he has reached his limit (and not doubled his money). Ensure that the players balance is 0. Test fails because the balance is 5 when the player stops for this reason:

```
239 turns later.
End Game 99: Fred now has balance 5
```

*Because the game runs in a **while** loop using **(player.balanceExceedsLimitBy(bet) && player.getBalance() < 200)**, it's clear that there's an issue with the Player class not checking whether the balance is exceeded correctly. Initial tests to show the bug will be to test **player.balanceExceedsLimitBy**.*

*Note that part of the issue here seems to be design; we don't want to know if the balance exceeds a limit, we want to know if the balance exceeds OR EQUALS a limit. Something like **Player.canBet(int amount)** might be more appropriate, but I believe changing it is beyond the scope of the assignment.*

Test Output (Failing)

Finished after 0.011 seconds

Runs: 3/3 Errors: 0 Failures: 1

tests.Bug2_CannotReachBettingLimit [Runner: JUnit 4] (0.000 s)

- testBalanceExceedsLimitBy_MoreThanBalance (0.000 s)
- testBalanceExceedsLimitBy_AllOfBalance (0.000 s)
- testBalanceExceedsLimitBy_SomeOfBalance (0.000 s)

Failure Trace

```
java.lang.AssertionError: expected:<true> but was:<false>
    at tests.Bug2_CannotReachBettingLimit.testBalanceExceedsLimitBy_AllOfBalance
```

```

2
3 import static org.junit.Assert.*;
9
10 public class Bug2_CannotReachBettingLimit {
11
12     @Before
13     public void setUp() throws Exception {
14     }
15
16     @Test
17     public void testBalanceExceedsLimitBy_SomeOfBalance() {
18         Player player = new Player("Pete", 5);
19         boolean result = player.balanceExceedsLimitBy(4);
20         assertEquals(true, result);
21     }
22
23     @Test
24     public void testBalanceExceedsLimitBy_AllOfBalance() {
25         Player player = new Player("Pete", 5);
26         boolean result = player.balanceExceedsLimitBy(5);
27         assertEquals(true, result);
28     }
29
30     @Test
31     public void testBalanceExceedsLimitBy_MoreThanBalance() {
32         Player player = new Player("Pete", 5);
33         boolean result = player.balanceExceedsLimitBy(6);
34         assertEquals(false, result);
35     }
36 }
37

```

Again, as expected from the bug report, the test fails.

Debug Log

Because the faulty function is only 1 line, it is immediately clear what is wrong.

Hypothesis: `balanceExceedsLimitBy`'s calculations are incorrect.

Test (JUnit): `testBalanceExceedsLimitBy_AllOfBalance`

Result: FAIL. Returns false.

Conclusion: `balanceExceedsLimitBy` is not being calculated correctly so the hypothesis is correct.

Before and After screenshots

As the function is only 1 line, there is not a very appropriate place to pause the script and inspect variables. $(5-5 > 0) == \text{false}$:

```

28     }
29
30     public boolean balanceExceedsLimitBy(int amount) {
31         return (balance - amount > limit);
32     }
33

```

this		Player (id=46)
balance	5	
limit	0	
name	"Pete" (id=55)	
amount	5	

But $(5-5 \geq 0) == \text{true}$:

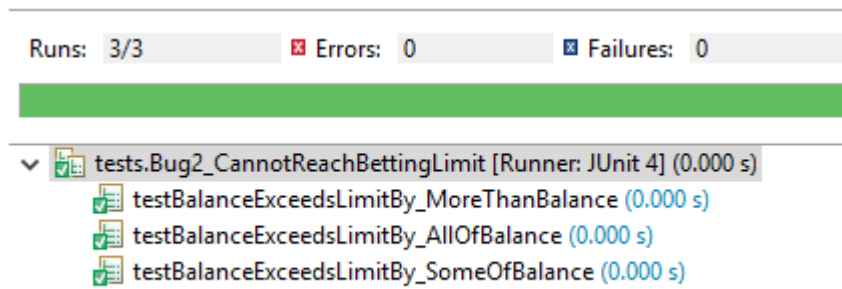
```

29
30     public boolean balanceExceedsLimitBy(int amount) {
31         return (balance - amount >= limit);
32     }
33

```

this		Player (id=46)
balance	5	
limit	0	
name	"Pete" (id=55)	
amount	5	

Test output (Fixed)



User Test Output (Fixed)

```
176 turns later.  
End Game 99: Fred now has balance 0
```

At this point there is an obvious nearby bug, but because it has not yet 'been reported' or caused any issues, I'll leave it to discover using more 'scientific' methods.

Bug 3 – Odds in the game do not appear to be correct

User Test

Run the program repetitively and observe that the resulting win/loss ratio is close to the expected win/loss ratio of 0.42.

The test currently fails because the ratio fluctuates wildly, from 0.2 to 0.6.

```
Win count = 5919, Lose Count = 3919, 0.60
```


Test Output (Failing)

Runs: 1/1 Errors: 0 Failures: 1

tests.Bug3_IncorrectOdds [Runner: JUnit 4] (0.050 s)

test_playRound_Odds (0.050 s)

```

9
10 import java.util.Random;
11
12 import org.junit.Test;
13
14 import crown_and_anchor.Dice;
15 import crown_and_anchor.DiceValue;
16 import crown_and_anchor.Game;
17 import crown_and_anchor.Player;
18
19 public class Bug3_IncorrectOdds {
20
21     @Test
22     public void test_playRound_Odds() {
23         Random random = new Random(999);
24         Player player = new Player("Pete", 100000000);
25         Game game = new Game(new Dice(), new Dice(), new Dice());
26         int wins = 0;
27         int rounds = 500000;
28
29         for(int i = 0; i < rounds; i++)
30         {
31             DiceValue pick = DiceValue.values()[random.nextInt(6)];
32             int winnings = game.playRound(player, pick, 1);
33             if(winnings > 0)
34                 wins++;
35         }
36         assertEquals(0.42d, (double)wins/(double)rounds, 0.01d);
37     }
38 }

```

Failure Trace

java.lang.AssertionError: expected:<0.42> but was:<0.332808>
at tests.Bug3_IncorrectOdds.test_playRound_Odds(Bug3_IncorrectOdds.java)

Failure Trace

java.lang.AssertionError: expected:<0.42> but was:<0.501058>
at tests.Bug3_IncorrectOdds.test_playRound_Odds(Bug3_IncorrectOdds.java)

Failure Trace

java.lang.AssertionError: expected:<0.42> but was:<0.167528>
at tests.Bug3_IncorrectOdds.test_playRound_Odds(Bug3_IncorrectOdds.java)

Debug Log

Interestingly, no matter how many iterations of **playRound** are run with random picks, certain values come up for the win/loss ratio. 0.16, 0.33, 0.50 are the ones I have observed. This makes me think that the results aren't entirely random, and that something is happening during initialization that changes the result of **playRound**.

Being a dice game, the randomness is obviously at least partly controlled by the dices. Because of this, the first thing to test is that the dices work.

Hypothesis: Dices aren't random.

Test (JUnit): Assert that the chance of each of the 6 faces rolling on the dice is 1/6.

Result: Fail. The odds for all faces except Spade is 0.20. The odds for Spade are 0.

Conclusion: Hypothesis is correct; the bug is somewhere in **Dice.roll** which is a single line that calls **DiceValue.getRandom**.

Hypothesis: `DiceValue.getRandom` doesn't return a random value.

Test (JUnit): Assert that the chance of each of the 6 faces being returned by `DiceValue.getRandom` is 1/6.

Result: Fail. The odds for all faces except Spade is 0.20. The odds for Spade are 0.

Conclusion: Hypothesis is correct; the bug is somewhere in `DiceValue.getRandom` which is a single line that calls `DiceValue.getRandom`.

The problem line is obviously in line 26, `int random = RANDOM.nextInt(DiceValue.SPADE.ordinal());`. According to (Oracle, 2016A), `Enum.ordinal()` will return the zero-based index of the enum. In this case it would be 5. According to (Oracle, 2016B), `Random.nextInt(value)` returns a value between 0 (inclusive) and the specified integer, 5 (exclusive). This means that 4 can never be returned, which explains the results of the previous hypothesis results.

Hypothesis: Changing line 26 to `int random = RANDOM.nextInt(values().length)` will result in the game functioning correctly.

Test (JUnit): Assert that the chance of each of the 6 faces rolling on the dice is 1/6 with the line changed.

Result: Partial Success. The Dices now work correctly.

Conclusion: While the dices now work correctly, the game still gives the same results.

Looking at how the `Game.playRound` interacts, with the Dice, it doesn't actually use the value returned by `roll`, but instead gets the value using `Dice.getValue`. For this reason, the next step is to check that the value returned by `Dice.getValue` is the same as that returned by `roll`.

Hypothesis: `Dice.getValue` is not returning the random values.

Test (JUnit): Assert that `Dice.getValue` returns the value that was rolled.

Result: Fail. `Dice.getValue` returns the correct value 1/6 of the time.

Conclusion: The issue is in `Dice.getValue`. Hypothesis is correct.

The next step is to look at where the value being returned by `Dice.getValue` is being set. It is only being set in the constructor, which explains my initial suspicions regarding something happening during initialization that is determining the results in some way.

Hypothesis: Setting value to the random face before returning it from `Dice.roll` will make the game work properly.

Test (JUnit): Assert that the chance of winning the game is 0.42 after making the change.

Result: SUCCESS. The Dices now work correctly.

Conclusion: The hypothesis is correct; the game odds are now correct.

The odds of the game now appear to be correct.

Before and After Screenshots/Fixed Test Output

Without fixes:

tests.Bug3_IncorrectOdds [Runner: JUnit 4] (1.111 s)

- test_getRandomDiceValue_OddsOfAnchor (0.051 s)
- test_getRandomDiceValue_OddsOfClub (0.059 s)
- test_getRandomDiceValue_OddsOfCrown (0.058 s)
- test_getRandomDiceValue_OddsOfDiamond (0.089 s)
- test_getRandomDiceValue_OddsOfHeart (0.057 s)
- test_getRandomDiceValue_OddsOfSpade (0.056 s)
- test_getValueReturnsRolledValue (0.074 s)
- test_playRound_Odds (0.296 s)
- test_rollDice_OddsOfAnchor (0.060 s)
- test_rollDice_OddsOfClub (0.063 s)
- test_rollDice_OddsOfCrown (0.062 s)
- test_rollDice_OddsOfDiamond (0.061 s)
- test_rollDice_OddsOfHeart (0.063 s)
- test_rollDice_OddsOfSpade (0.060 s)

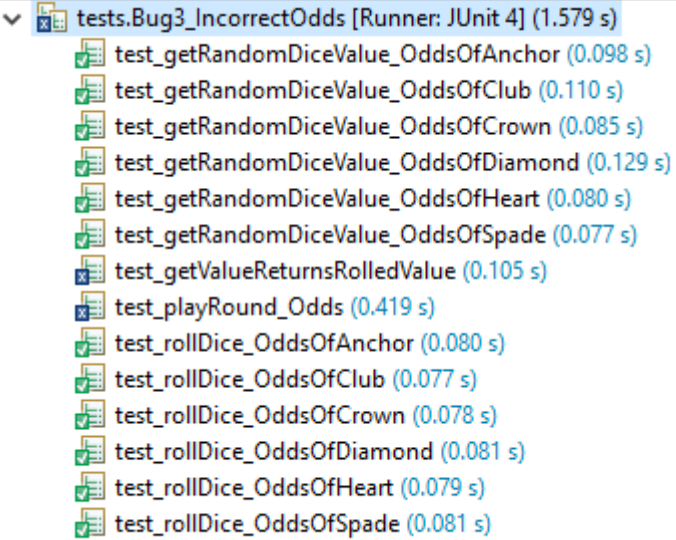
java.lang.AssertionError: expected:<0.42> but was:<0.3335292>
at tests.Bug3_IncorrectOdds.test_playRound_Odds(Bug3_IncorrectOdds.java:34)

java.lang.AssertionError: expected:<0.16666666666666666> but was:<0.0>
at tests.Bug3_IncorrectOdds.test_getRandomDiceValue_OddsOfSpade(Bug3_IncorrectOdds.java:220)

java.lang.AssertionError: expected:<0.16666666666666666> but was:<0.1997344>
at tests.Bug3_IncorrectOdds.test_getRandomDiceValue_OddsOfHeart(Bug3_IncorrectOdds.java:205)

```
25 public static DiceValue getRandom() {
26     int random = RANDOM.nextInt(DiceValue.SPADE.ordinal());
27     return values()[random];
28 }
29
30
31
32
33
34 public DiceValue roll() {
35     return DiceValue.getRandom();
36 }
```

With first fix:



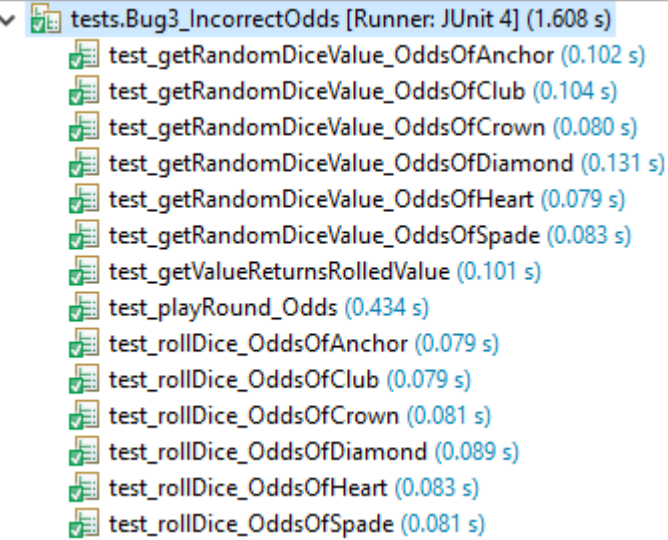
tests.Bug3_IncorrectOdds [Runner: JUnit 4] (1.579 s)

- test_getRandomDiceValue_OddsOfAnchor (0.098 s)
- test_getRandomDiceValue_OddsOfClub (0.110 s)
- test_getRandomDiceValue_OddsOfCrown (0.085 s)
- test_getRandomDiceValue_OddsOfDiamond (0.129 s)
- test_getRandomDiceValue_OddsOfHeart (0.080 s)
- test_getRandomDiceValue_OddsOfSpade (0.077 s)
- test_getValueReturnsRolledValue (0.105 s)
- test_playRound_Odds (0.419 s)
- test_rollDice_OddsOfAnchor (0.080 s)
- test_rollDice_OddsOfClub (0.077 s)
- test_rollDice_OddsOfCrown (0.078 s)
- test_rollDice_OddsOfDiamond (0.081 s)
- test_rollDice_OddsOfHeart (0.079 s)
- test_rollDice_OddsOfSpade (0.081 s)

```
25 public static DiceValue getRandom() {
26     int random = RANDOM.nextInt(values().length);
27     return values()[random];
28 }
```

java.lang.AssertionError: expected:<0.42> but was:<0.4998846>
at tests.Bug3_IncorrectOdds.test_playRound_Odds(Bug3_IncorrectOdds.java:34)

With both fixes:



tests.Bug3_IncorrectOdds [Runner: JUnit 4] (1.608 s)

- test_getRandomDiceValue_OddsOfAnchor (0.102 s)
- test_getRandomDiceValue_OddsOfClub (0.104 s)
- test_getRandomDiceValue_OddsOfCrown (0.080 s)
- test_getRandomDiceValue_OddsOfDiamond (0.131 s)
- test_getRandomDiceValue_OddsOfHeart (0.079 s)
- test_getRandomDiceValue_OddsOfSpade (0.083 s)
- test_getValueReturnsRolledValue (0.101 s)
- test_playRound_Odds (0.434 s)
- test_rollDice_OddsOfAnchor (0.079 s)
- test_rollDice_OddsOfClub (0.079 s)
- test_rollDice_OddsOfCrown (0.081 s)
- test_rollDice_OddsOfDiamond (0.089 s)
- test_rollDice_OddsOfHeart (0.083 s)
- test_rollDice_OddsOfSpade (0.081 s)

```
14 public DiceValue roll() {
15     value = DiceValue.getRandom();
16     return value;
17 }
```

User Test Output (Fixed)

Note that the win/loss ratio does vary slightly, but is always 0.42 +/- 0.01. This is because the sample size is not large enough to always be exactly 0.42.

```
Win count = 8593, Lose Count = 11898, 0.42
```

Additional Bugs

In order to find any additional bugs, I'll now write unit tests for each of the classes (except Main) to ensure all their functions work as intended.

Unit testing the **Dice** class came up with no further bugs.

Unit testing the **DiceValue** enum came up with no further bugs.

Unit testing the **Game** class came up with no further bugs.

Unit testing the **Player** class came up with no further bugs.

Notes

There are some changes that I don't think are appropriate for the assignment that could be made:

- As discussed above, the name of **balanceExceedsLimitBy** and **balanceExceedsLimit** should probably be changed to something like **canBet(amount)** and **isAtLimit()**. While this is the intended usage for **balanceExceedsLimitBy**, it's not clear whether it is for **balanceExceedsLimit** as the method is not used by the program. For this reason, I will leave it with the inconsistent behaviour for this assignment.
- The player can bet 0 for some reason. This could get annoying for the dealer.
- The player cannot place multiple bets on a single roll, which seems to be a key mechanic of the game.
- Main has not been tested/debugged because I assume this is just a test program and a quick glance didn't spot any obvious errors.
- The output is not particularly clear; it doesn't say what the 0.42 means, but I assume it is just a test function anyway.

References

Oracle. (2016A, October 1). *Class Enum<E extends Enum<E>>*. Retrieved from Oracle Help Center: <https://docs.oracle.com/javase/7/docs/api/java/lang/Enum.html>

Oracle. (2016B, October 1). *Class Random*. Retrieved from Oracle Help Center: <https://docs.oracle.com/javase/7/docs/api/java/util/Random.html>