

## Debugging the Crown and Anchor Game

### Bug 1: Game doesn't pay out correct level

**Hypothesis 1:** playRound method is not returning the correct winning bet amount.

**Experiment:** Game.Java has method named playRound which returns the winning. I am assuming playRound is not changing the player's balance correctly

#### **Test Output:.**

Below is the test screenshot from JUnit test:

```
<terminated> IncorrectPayout [JUnit] C:\Program Files\Java\jre1.8.0_111\bin\javaw.exe (15Oct, 2017)

----- Start In All matched method! -----
CROWN
CROWN
CLUB
----- End In All matched method! -----
The balance should be 130 but is 100

----- Start In Two matched method! -----
ANCHOR
ANCHOR
ANCHOR
----- End In Two matched method! -----
The balance should be 120 but is 90

----- Start In One matched method! -----
ANCHOR
HEART
HEART
----- End In One matched method! -----
The balance should be 110 but is 90

----- Start In Zero Match Method! -----
ANCHOR
DIAMOND
HEART
----- End In Zero Match Method! -----

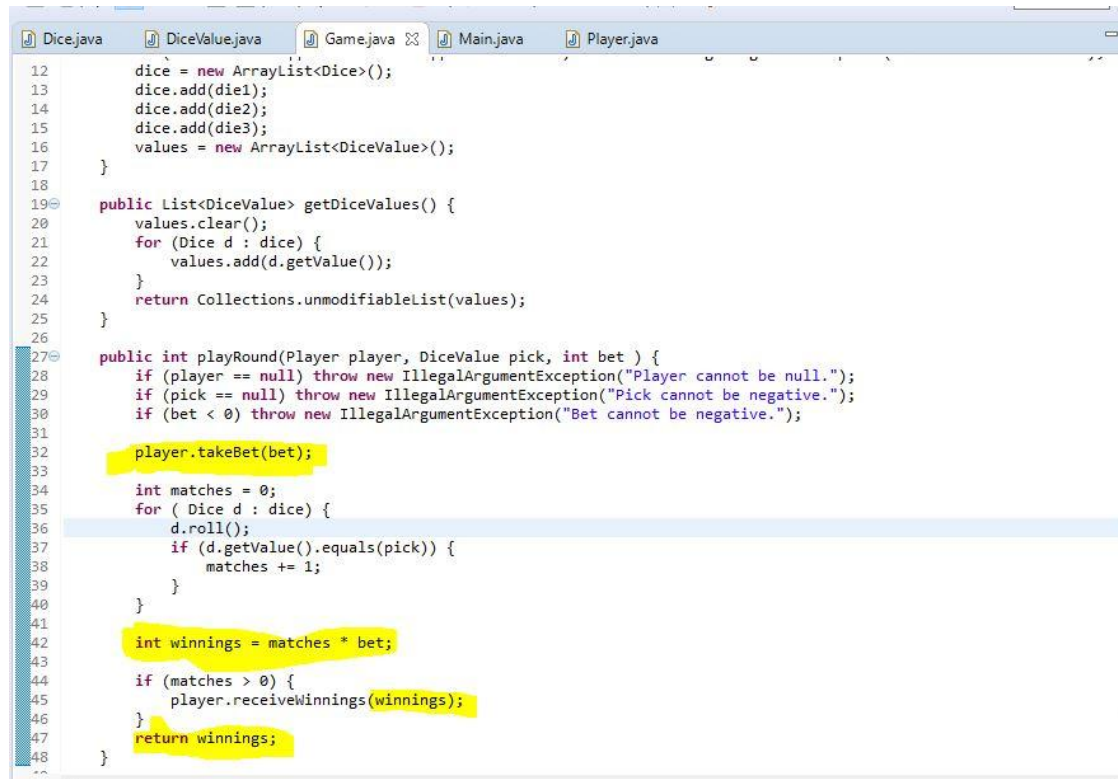
Failure Trace
java.lang.AssertionError: expected:<30> but was:<10>
    at company.IncorrectPayout.testWinningForAllMatch(Inco
```

Figure 1: Game class, playRound method test screenshot

**Conclusion:** Since methods are returning correct value, there is no error in this method, thus the hypothesis is wrong.

**Hypothesis 2:** Maybe the balance becomes incorrect is just after takeBet is called. (i.e. to use correct terminology the player's balance becomes 'infected' by takeBet).

let us see this screenshot:



```
12     dice = new ArrayList<Dice>();
13     dice.add(die1);
14     dice.add(die2);
15     dice.add(die3);
16     values = new ArrayList<DiceValue>();
17 }
18
19 public List<DiceValue> getDiceValues() {
20     values.clear();
21     for (Dice d : dice) {
22         values.add(d.getValue());
23     }
24     return Collections.unmodifiableList(values);
25 }
26
27 public int playRound(Player player, DiceValue pick, int bet ) {
28     if (player == null) throw new IllegalArgumentException("Player cannot be null.");
29     if (pick == null) throw new IllegalArgumentException("Pick cannot be negative.");
30     if (bet < 0) throw new IllegalArgumentException("Bet cannot be negative.");
31
32     player.takeBet(bet);
33
34     int matches = 0;
35     for ( Dice d : dice) {
36         d.roll();
37         if (d.getValue().equals(pick)) {
38             matches += 1;
39         }
40     }
41
42     int winnings = matches * bet;
43
44     if (matches > 0) {
45         player.receiveWinnings(winnings);
46     }
47     return winnings;
48 }
```


## Experiment:

To test these ones, I put step as follows:

1. At the end of playRound, and have a look at the player's balance.
2. Then, step to just before takeBet and make sure the balance is still the same

End of playRound:

```
46
47 public int playRound(Player player, DiceValue pick, int bet)
48 {
49     if (player == null) throw new IllegalArgumentException("Pla
50     if (pick == null) throw new IllegalArgumentException("Pick
51     if (bet < 0) throw new IllegalArgumentException("Bet cannot
52
53     player.takeBet(bet);
54
55     int matches = 0;
56     for (Dice d : dice)
57     {
58         d.roll();
59         if (d.getValue().equals(pick))
60         {
61             matches += 1;
62         }
63     }
64
65     int winnings = matches * bet;
66
67     if (matches > 0)
68     {
69         player.
70     }
71     return winnings;
72 }
```




Debugger window showing Player object state:

- player= Player (id=18)
  - balance= 95
  - limit= 0
  - name= "Fred" (id=22)

Just before takeBet:

```
47 public int playRound(Player player, DiceValue pick, int bet)
48 {
49     if (player == null) throw new IllegalArgumentException("Play
50     if (pick == null) throw new IllegalArgumentException("Pick
51     if (bet < 0) throw new IllegalArgumentException("Bet cannot
52
53     player.takeBet(bet);
54
55
56
57
58
```




Debugger window showing Player object state:

- player= Player (id=18)
  - balance= 95
  - limit= 0
  - name= "Fred" (id=22)

The player's balance is still 95.

Stepping to just after takeBet:

```
47 public int playRound(Player player, DiceValue pick, int bet)
48 {
49     if (player == null) throw new IllegalArgumentException("Player
50     if (pick == null) throw new IllegalArgumentException("Pick cann
51     if (bet < 0) throw new IllegalArgumentException("Bet cannot be
52
53     player.takeBet(bet);
54
55
56
57
58
59
```



Debugger window showing Player object state:

- player= Player (id=18)
  - balance= 90
  - limit= 0
  - name= "Fred" (id=22)

Yes, the balance has changed to 90

### Conclusion:

Everything that we required to verify our hypothesis has come true, so hypothesis 2 is verified it is takeBet that causes this infection.

### Bug 2: Player cannot reach betting limit:

#### Limit set to 0, but game ends with player still with 5 (dollars) remaining.

**Hypothesis 1:** balanceExceedsLimitBy does not include the minimum limit.

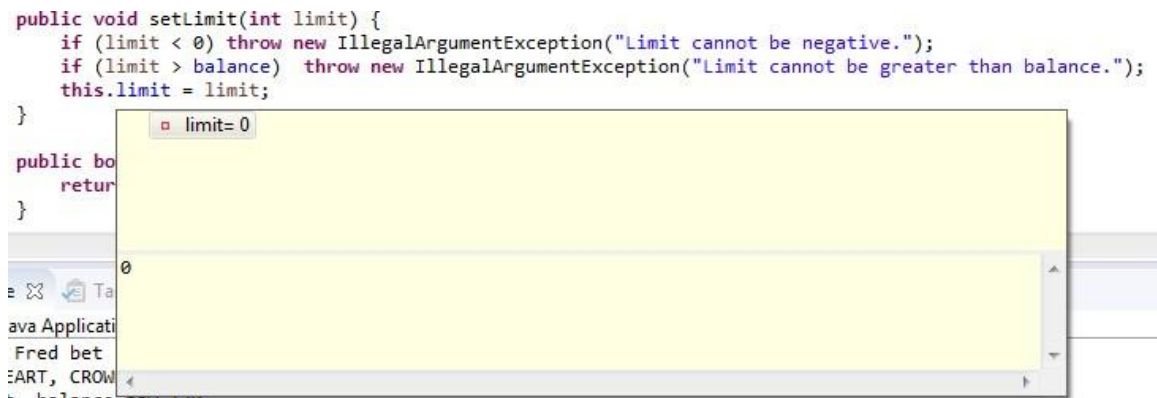
**Experiment:** It is identified that the bug problem is probably with the condition, where the game must stop while the player exceeds the balance by bet amount. This method is included in the Player.java as:

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

So in this case, if the limit is 0, the game would continue as long as the balance - bet > 0. So in the case that the bet is 5 as in main file, when the balance is 5, the game would stop as "5 - 5 > 0" is false.

From the above guess, three hypotheses are proposed:

**Hypothesis 1:** setLimit() method correctly assigns limit as zero.



- **Hypothesis 2:** balanceExceedsLimitBy does not include the minimum limit. After run, the game as at balance = 5, limit = 0 so (2) is verified



**Hypothesis 3:** The game ends when the limit is reached. (TRUE)

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

The testing of the hypotheses is conducted by putting breakpoints at `return (balance - amount > limit);` which is in ( Player.java)

**Bug 3: Odds in the game do not appear to be correct. Crown and Anchor games have an approximate 8% bias to the house. So the win : (win+lose) ratio should approximately equal 0.42. This does not appear to be the case.**

The methods that generate a new DiceValue for each dice occur in the Main.java, Game.java and Dice.java.

Starting from the **Main.java**, three dices of the game are called:

```
BufferedReader console = new BufferedReader(new InputStreamReader(System.in));

Dice d1 = new Dice();
Dice d2 = new Dice();
Dice d3 = new Dice();

Player player = new Player("Fred", 100);
Game game = new Game(d1, d2, d3);
List<DiceValue> cdv = game.getDiceValues();
```

The **Main.java** also implies that the assigning of dice occurs in the *playRound* method of *Game.java*.

```
while (player.balanceExceedsLimitBy(bet) && player.getBalance() < 200)
{
    turn++;
    DiceValue pick = DiceValue.getRandom();

    System.out.printf("Turn %d: %s bet %d on %s\n",
        turn, player.getName(), bet, pick);

    int winnings = game.playRound(player, pick, bet);
    cdv = game.getDiceValues();

    System.out.printf("Rolled %s, %s, %s\n",
        cdv.get(0), cdv.get(1), cdv.get(2));

    if (winnings > 0) {
        System.out.printf("%s won %d, balance now %d\n\n",
            player.getName(), winnings, player.getBalance());
        winCount++;
    }
}
```

The rolling of dice occurs in *playRound* method as

```
public int playRound(Player player, DiceValue pick, int bet ) {
    if (player == null) throw new IllegalArgumentException("Player cannot be null.");
    if (pick == null) throw new IllegalArgumentException("Pick cannot be negative.");
    if (bet < 0) throw new IllegalArgumentException("Bet cannot be negative.");

    // player.takeBet(bet);

    int matches = 0;
    for ( Dice d : dice) {
        d.roll();
        if (d.getValue().equals(pick)) {
            matches ++;
        }
    }

    int winnings = matches * bet;

    if (matches > 0) {
        player.receiveWinnings(winnings);
    }

    else player.takeBet(bet);
    return winnings;
}
```

The rolling in **Dice.java** occurs as:

```
public Dice() {
    value = DiceValue.getRandom();
}

public DiceValue getValue() {
    return value;
}

public DiceValue roll() {
    return DiceValue.getRandom();
}
```

So from this, it could be identified that:


- The *value* variable in **Dice.java** is invariant throughout the game as it is just assigned in the constructor of Dice, but no changing occurs further in the file.
  - The compare of the pick is with the *d.getValue()*, which is the value of the Dice originally, rather than the value of the dice after rolling.
  - The *roll()* method, although assigns a new DiceValue to the dice, the comparison of DiceValue is with the *getValue()* method, which returns the invariant variable *value*.
  - The three dices are set at the beginning of the game in **Main.java**, with the given *value* at the setting of constructor. If the *value* variable in Dice(), the dices may be reused again throughout the game.
- 
- **Hypothesis 1:** the *roll()* method does not change the value of DiceValue (incorrect).
  - **Hypothesis 2:** the result of each turn is decided by comparing the pick with the *value* of the dice (correct).
  - **Hypothesis 3:** three dices are assigned with each value and reused throughout the game (incorrect).

To test the hypotheses, one breakpoint is put within the loop to monitor the changing state of DiceValue. It is verified as at the start of the game, three dice values in *cdv* are:

```
Player player = new Player("Fred", 100);
Game game = new Game(d1, d2, d3);
List<DiceValue> cdv = game.getDiceValues();
```

```
int totalWins =
int totalLosses
```

```
while (true)
{
```



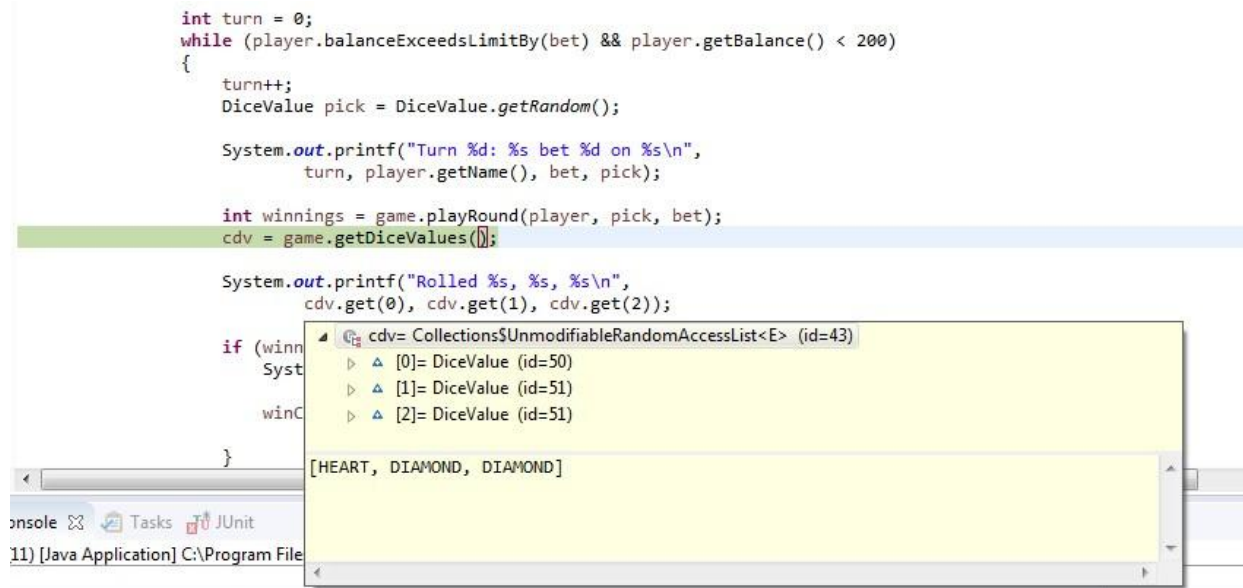
```
cdv= Collections$UnmodifiableRandomAccessList<E> (id=43)
  [0]= DiceValue (id=50)
  [1]= DiceValue (id=51)
  [2]= DiceValue (id=51)
```

```
[HEART, DIAMOND, DIAMOND]
```

```
Tasks JUnit
pplication] C:\Program
```



but after rolling, three dices value are still the same:

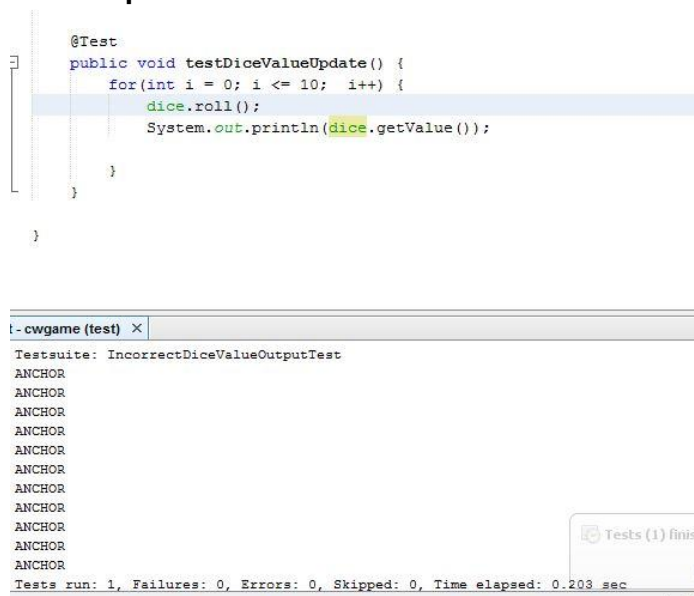


**Bug 4: Combination of three symbols never changes, first combination of symbols is repeated through the whole game.**

**Hypothesis:** Rolled dice Value is not update each time dice is rolled.

**Experiment::** if the displayed three dice values were Crown, Club and Anchor and user has picked the Anchor, then user should have won, but the prompt is player lost, which means the balance payout working fine, only dice value is not update after rolled each time. Since in Main class uses Game class getDiceValues to get the rolled dice values, and the this method uses Dice class getValue, for that reason I think the bug is in Dice class getValue not being updated.

**Test Output:**





As we can see in the screenshot above that, even though the dice is looped ten times, and each time value is same.

**Conclusion:**

The hypothesis was correct, the value is not being update in Dice class, thus the values never changes in each turn.