## 

Assessment Item 4:

Debugging

Name: Jean Claude Jino Rousseau

Student ID: 11590309

Lecture: Friday 9.00 am - 12.00 pm

Due Date: 07.10.2016

Lecturer: Dr. Recep Ulusoy

Number of words: 1074

Table of Contents

[Introduction 3](#_Toc464036629)

[Analysis phase 3](#_Toc464036630)

[Bug Fixing and Improvements 4](#_Toc464036631)

[STEP 1: Improvement 1 (Welcome message and prompt for data) 4](#_Toc464036632)

[STEP 2: Bug 1 (Win/Lose balance) 5](#_Toc464036633)

[STEP 3: Bug 2 (Game stops while player still has a balance) 6](#_Toc464036634)

[STEP 4: Improvement\_2 (Display message how to quit the game) 7](#_Toc464036635)

[STEP 5: Bug 3 (The value SPADE never shows up) 8](#_Toc464036636)

[STEP 6: Bug 4 (Same result when dice are rolled) 9](#_Toc464036637)

[Replication 11](#_Toc464036638)

[Simplification 11](#_Toc464036639)

[Tracing 11](#_Toc464036640)

[Resolution 11](#_Toc464036641)

# Introduction

This report will elaborate on my task to replicate, simplify, trace, and eliminate the bugs that live in the source code about the Crown and Anchor game code that has been made available for our last assessment. The whole process will be completed using best practice debugging techniques.

# Analysis phase

Crown and Anchor games is a game that is played on a layout which is marked in a manner that is shown in picture 1 below:



**Picture 1**

The game is played with three identical dice with the faces of each dice marked with the symbols of a crown, anchor, heart, diamond, club, and spade. The game is controlled by a dealer who is not permitted to gamble. The latter is the only person who will throw the dice or activate the dice cage.

If any of the three dice fail to come to rest with one surface flat to the base of the dice cage or flat on the layout mat, the dealer shall call “No Spin/Throw”. If in any circumstances that call is made, all wagers shall be avoided on that spin/throw.

The minimum and maximum wagers shall be prominently displayed on a sign at the table. A wager on a particular symbol shall win if the symbol appears on one or more of the uppermost face of the three dice and shall lose if the symbol does not appear.

Winning wagers shall be paid at odds not less than as shown in picture 2 below:



**Picture 2**

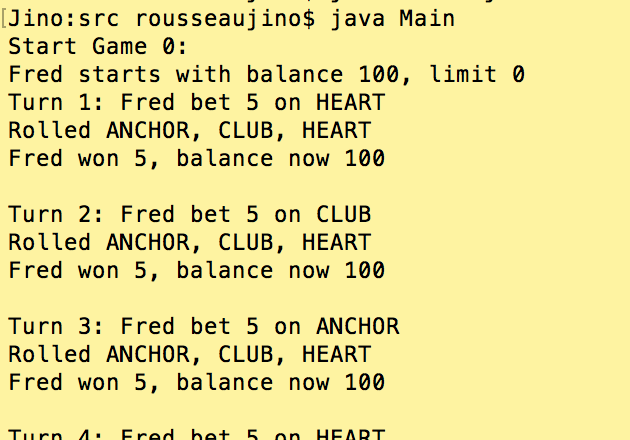
Anyone under the age of 18 years is forbidden to wager, be involved in the dealing or conduct of the game.

# Bug Fixing and Improvements

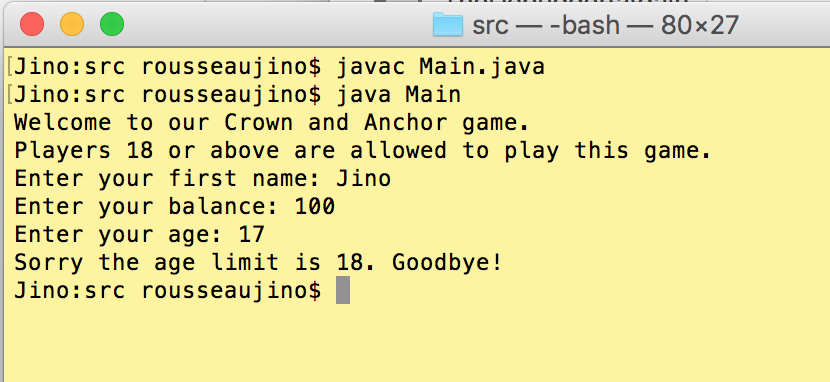
After careful analysis of the game, it can be stated that﻿﻿﻿ several bugs exist and many improvements can be made. In my attempt to improve the usability and functionalities of the game, I have developed different sequential steps to deal with all the existing issues in the game.

## STEP 1: Improvement 1 (Welcome message and prompt for data)

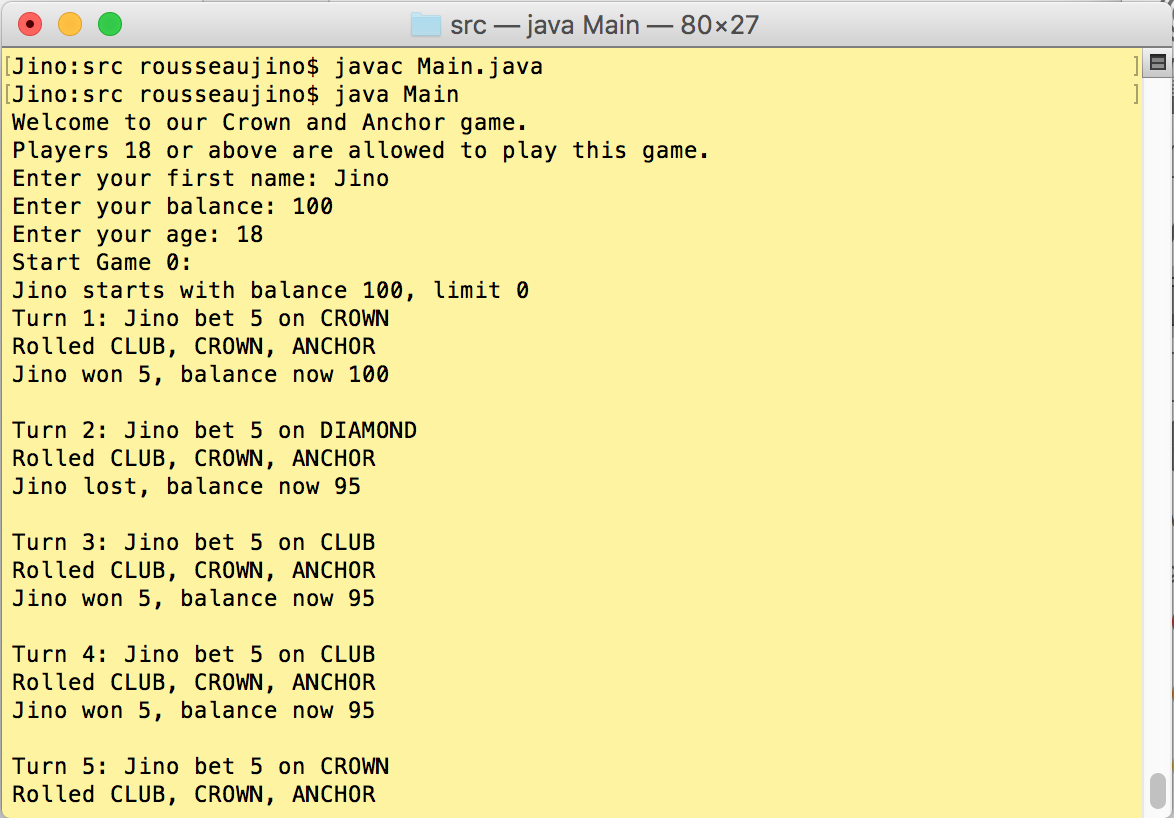
**Missing component:** Welcome message and a reminder that underage people are not allowed to play, missing at the start of the game as shown in picture 1 below. Also, the game has ‘Fred’ as player. That name and the balance cannot be changed.

 **Picture 3**

**Improvement 1:** Display a welcome message and a reminder that anyone under 18 is not allowed to play. Invite the user to input their name, balance, and age. Discard the user if the age entered is less than 18 as displayed in picture 2 below.  **﻿﻿﻿﻿﻿﻿﻿﻿﻿﻿**

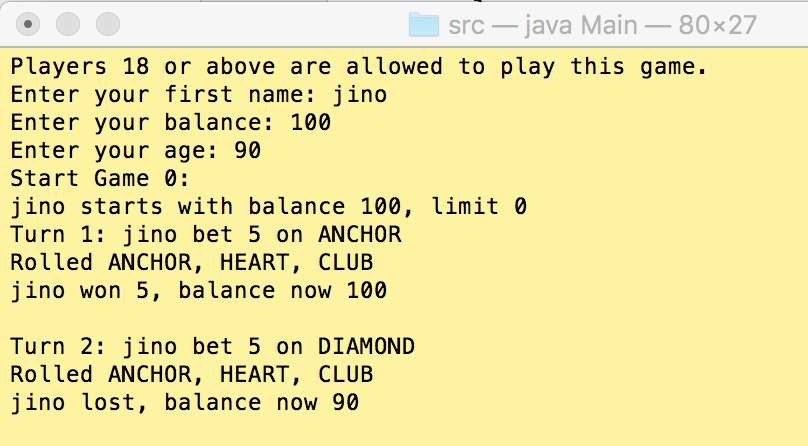
** Picture 4**

The validation is shown again below in picture 5 when the player enters a legal age.

 **Picture 5**

## STEP 2: Bug 1 (Win/Lose balance)

**Bug 1\_balance:**  The balance displayed after a player wins is that than the expected amount which is specified by the rule showed in picture 2 above. This can be shown in Picture 6 below where in turn 1, player Jino bet $5 on Anchor, wins one Anchor but end up with a balance of 100 instead of 105 (5 [for the bet] + 95 [amount left] + 5 [1 Anchor win])

 **Picture 6**

**Resolving Bug 1\_balance** Making changes to the Game.java code from:

player.takeBet(bet);

int matches = 0;

for ( Dice d : dice) {

d.roll();

if (d.getValue().equals(pick)) {

matches += 1;

}

}

int winnings = matches \* bet;

if (matches > 0) {

player.receiveWinnings(winnings);

}

return winnings;

to :

//player.takeBet(bet);

int matches = 0;

for ( Dice d : dice) {

d.roll();

if (d.getValue().equals(pick)) {

matches += 1;

}

}

int winnings = matches \* bet;

if (matches > 0) {

player.receiveWinnings(winnings);

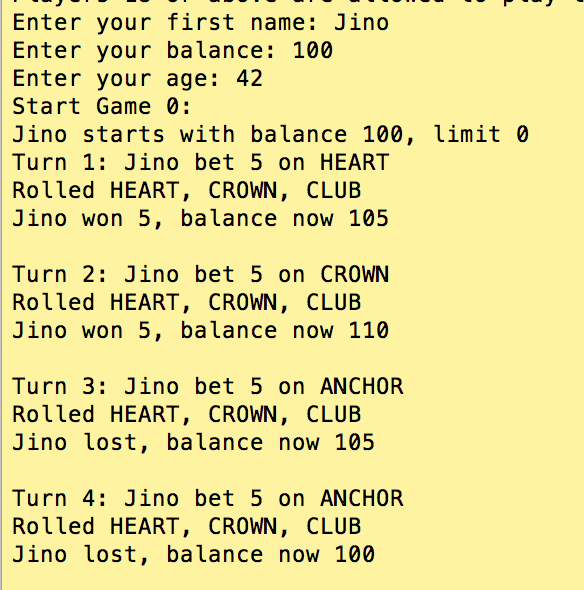
}else {

player.takeBet(bet);

}

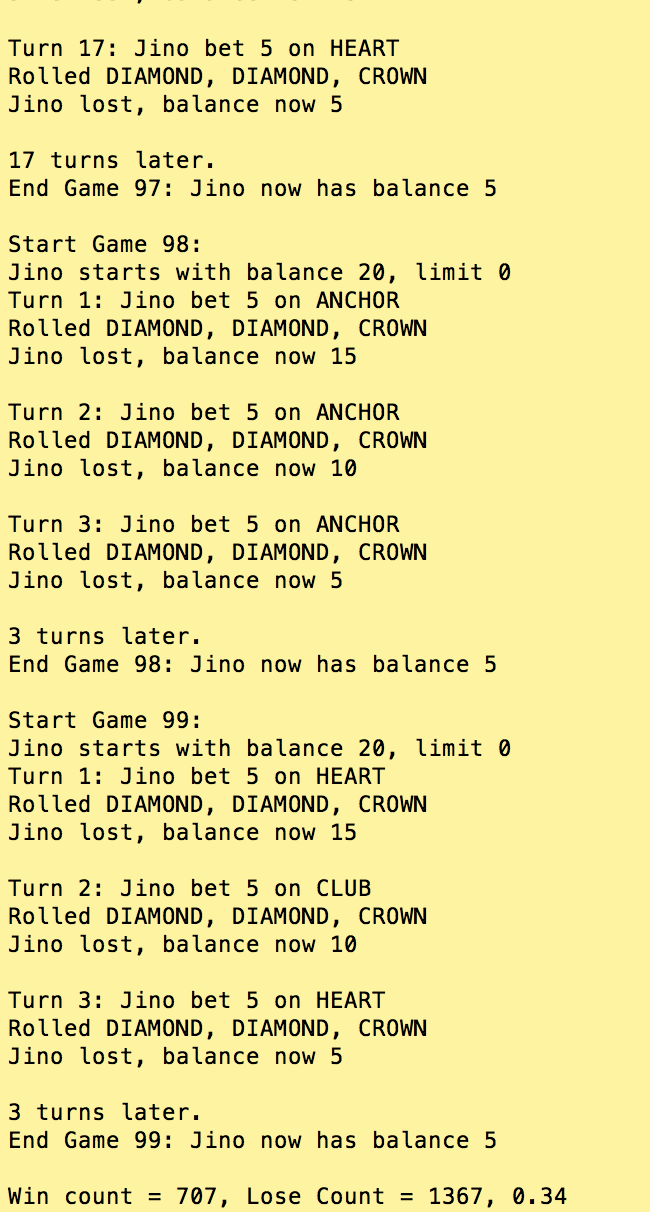
return winnings;

Produces the results displayed in picture 7 below. In turn 1, the same player bets $5 on heart, wins one, and he ends up with a balance of $105 the right amount that should be paid to him.

 **Picture 7**

## STEP 3: Bug 2 (Game stops while player still has a balance)

**Bug 2** is about the game stopping while the player still has a balance as shown in picture 8 below:

** Picture 8**

This was quickly remedied by changing the code in the Player.java code from:

/\*\* The method balanceExceedsLimitBy returns true or false if the difference between the balance and the amount is greater than the limit \*/

public boolean balanceExceedsLimitBy(int amount) {

return (balance - amount > limit);

}

to:

**public boolean balanceExceedsLimitBy(int amount) {**

**return (balance - amount >= limit);**

**}**

and in the Game.java code from:

public int playRound(Player player, DiceValue pick, int bet ) {

if (player == null) throw new IllegalArgumentException("Player cannot be null.");

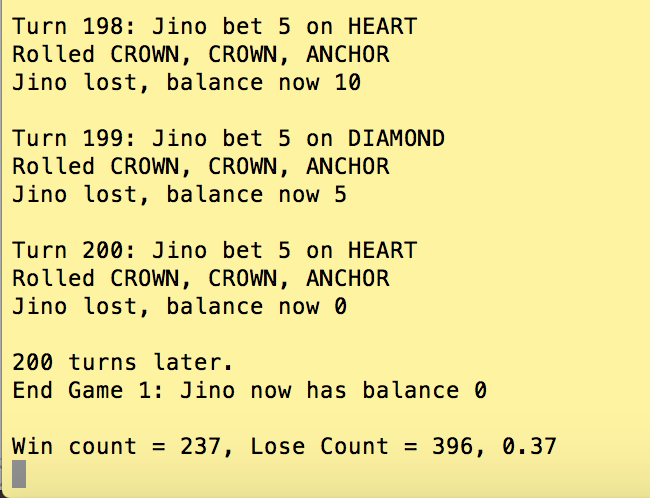
if (pick == null) throw new IllegalArgumentException("Pick cannot be negative.");

if (bet < 0) throw new IllegalArgumentException("Bet cannot be negative.");

to:

**if (bet <= 0) throw new IllegalArgumentException("Bet cannot be negative.");**

produces the result displayed in picture 9 below:

 **Picture 9**

## STEP 4: Improvement\_2 (Display message how to quit the game)

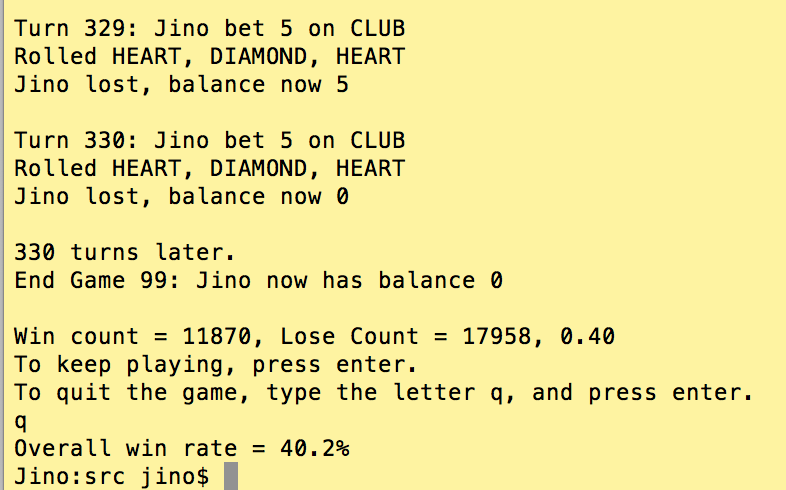
Picture 9 above show the end of a game and the prompt is waiting for the user to do something. However, there is no guide to help the user in decide what to do. I introduced **Improvement\_2 t**o the code to guide the users in their next actions. Since the code already contains the following syntax that the user is not aware of:

String ans = console.readLine();

if (ans.equals("q")) break;

I introduced some code to the program to display a message to the users to help them choose their next actions and a sample results is shown in picture 10 below.

System.out.println("To keep playing, press enter. \nTo quit the game, type the letter q, and press enter."); // Display the message for the user to know how to quite the game.

 **Picture 10**

## STEP 5: Bug 3 (The value SPADE never shows up)

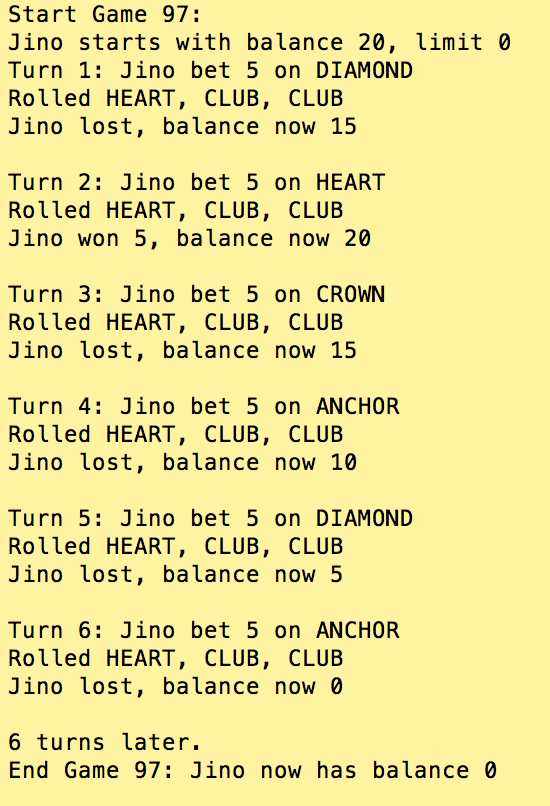
Based on the outcome of the game, it is noticed that the value of the dice ‘SPADE’ never shows up on the screen. Going through the DiceValue.java code, I found the following syntax which produced the result shown in picture 11, no ‘SPADE’.

public static DiceValue getRandom() {

int random = RANDOM.nextInt(DiceValue.SPADE.ordinal());

return values()[random];

}

 **Picture 11**

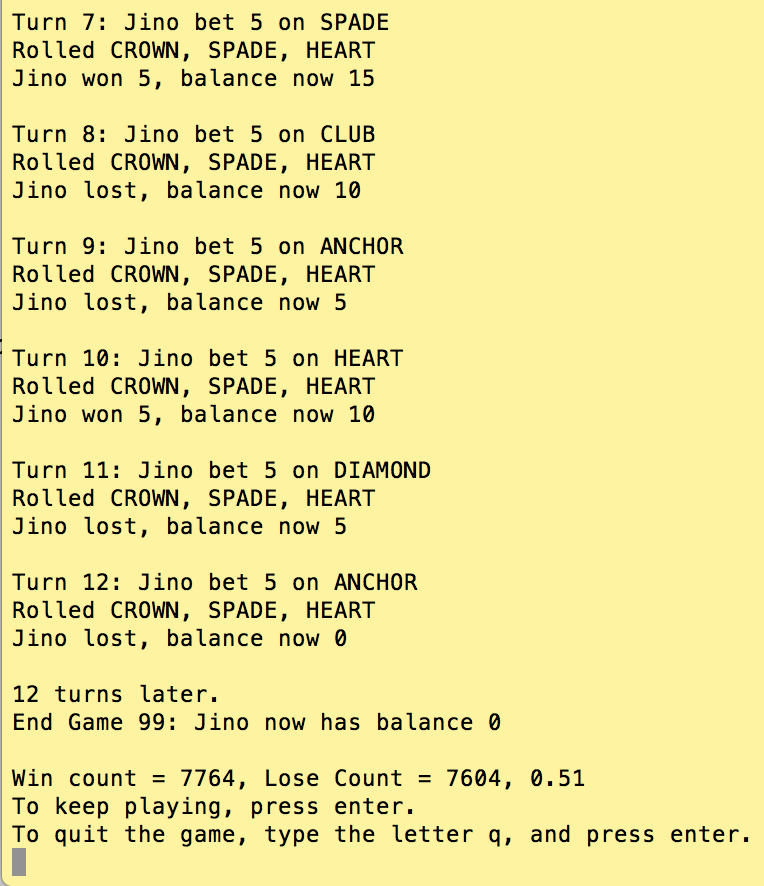
The method ‘ordinal()’ returns the ordinal of this enumeration constant (its position in its enum declaration, where the initial constant is assigned an ordinal of zero). Changing the code by adding ‘1’ to it fixed problem as shown in picture 12 below.

public static DiceValue getRandom() {

int random = RANDOM.nextInt(DiceValue.SPADE.ordinal() + 1);

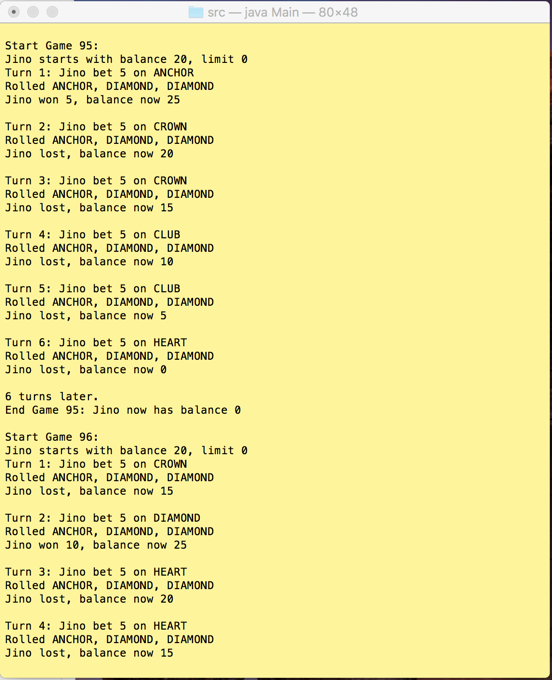
return values()[random];

}

 **Picture 12**

## STEP 6: Bug 4 (Same result when dice are rolled)

Base on the output of the game, as shown in picture 13 below, when the dice are rolled the same result pattern (ANCHOR, DIAMOND, DIAMOND) is produced regardless of the number of games being played.

 **Picture 13**

This can be resolved by improving the method getRandom displayed below from the DiceValue.java code

public static DiceValue getRandom() {

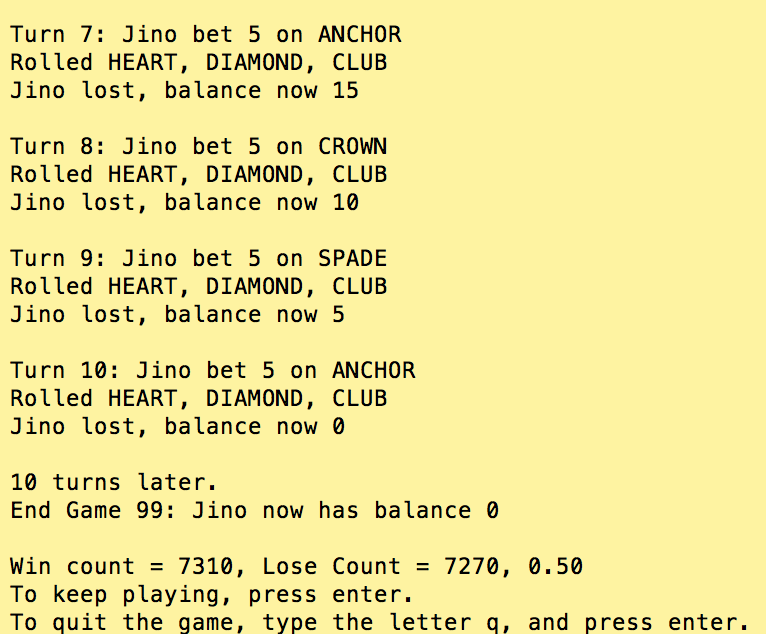
int random = RANDOM.nextInt(DiceValue.SPADE.ordinal() + 1);

return values()[random];

}

## STEP 7: Bug 5 (win:lose ration not correct)

According to one of the rules, the Crown and Anchor games have an approximate 8% bias to the house. So the win: (win+lose) ratio should approximately equal 0.42. Unfortunately, this does not appear to be the case as shown in picture 14 below.

** Picture 14**

This can be resolved by doing the following actions:

1. Create a variable called ‘winRatio’ that will represent the ratio of win: (win+lose) in the Player.java class
2. Every time the player wins, the winRatio variable is increased by one.
3. Sets the limit of the winRatio variable to 42%.
4. Once the players reach that limit and they still can play, they will lose all their remaining games in order to keep the ratio to 42%

# Replication

# Simplification

# Tracing

# Resolution