Jasque Saydyk and Miguel Quinones

Professor Vanderberg, Swenson, and Trice

CS 126L Section 2

12 April 2017

Lab 10 - Casino Night

1. Problem Statement

In this lab, we were tasked with creating a class representing a Card and another class representing a ChipBank, then run them through the provided tester to ensure they worked properly. Both classes have a series of getters and setters, initializations, and string representations. In the Class, the three unique methods are face_down() and face_up() which manipulate the string representation of the class on how much it tells the user, and the initialization, which auto assigns a card Suit and Rank bases on the number given. For the ChipBank, the unique method is the string representation, which shows the number of chips the user possess along with their amount.

2. Planning

The implementation of these two classes are really straight-forward, as the documentation that was provided with the project clearly stated what was required for each class, though there were a few exceptions. For the Card class, the initialization required that one auto-assigns a Rank and Suit to the card given a number. While we could have used a loop, list, or some other structures to make the code less repeating, we decided that a single, giant if tree simulating a switch case was simpler, clearer, and faster to do. We also decided to make it a separate function from the initialization with the name of auto_assign_rank_and_suit(), so if the initialization function needs to be changed in the future, it will be a lot simpler to do so.

For the ChipBank class, the unique function was the string representation of the object, which required that it shows the number of chips the user possesses along with their money in their account. We decided to do this using modolus, which will give us the correct answer.

Lastly there is the extra credit function, that requires that we log all activity in the bank account into an external file. This seems to require the use of Python Logging, and we have decided not to do it as of this time.

3. Implementation and Testing

This project was pretty straight-forward, but we did run into several roadblocks. We forgot to make sure all functions called self as their first parameter initially. In the Card class, the auto_assign_rank_and_suit() method had a bug that took 30 minutes to solve where it would only output the first 11 card Ranks and Suits, and the rest would output the default None. I eventually found that the bug was an if statement I had at the very beginning of the tree that would only let in card values that were less than 12, when our original plan was to convert the numbers to a value between 0 to 12, then assign a Rank and Suit, which we decided was a bad plan.

In the ChipBank class, the string representation of the object was wrong was we failed to realize that we needed to be using integer division instead of modulus to calculate the number of chips in each category. The only other bug we experienced was a typo in the Test_file class we copied from the lab documentation were we forgot to include the parentheses for a method, thus leading it to spit out an arcane statement of the object the method is connected to.

Below is the output from the test class. The only value that doesn't align is the print(card.get_value) line that the documentation says should be 10, but we output as 37. The fact that the value says it should be 10 confuses us though. This is because, if you start from 0 to 12, the Queen should have the value of 11, not 10. In addition to this, the card value that is inputted is 37, which says that we ought to modify the incoming value, which we don't believe is necessary. If need be, we will change it to fit the correct value, but at the moment, we see no need to do so and we would even hazard to say that the value in the documentation is incorrect.

```
Ace of Spades
2 of Spades
3 of Spades
4 of Spades
5 of Spades
6 of Spades
7 of Spades
8 of Spades
9 of Spades
10 of Spades
Jack of Spades
Queen of Spades
King of Spades
Ace of Hearts
2 of Hearts
3 of Hearts
4 of Hearts
5 of Hearts
6 of Hearts
7 of Hearts
8 of Hearts
9 of Hearts
10 of Hearts
Jack of Hearts
Queen of Hearts
King of Hearts
Ace of Clubs
2 of Clubs
3 of Clubs
4 of Clubs
5 of Clubs
6 of Clubs
7 of Clubs
8 of Clubs
9 of Clubs
10 of Clubs
Jack of Clubs
Queen of Clubs
King of Clubs
Ace of Diamonds
2 of Diamonds
3 of Diamonds
4 of Diamonds
5 of Diamonds
6 of Diamonds
7 of Diamonds
8 of Diamonds
9 of Diamonds
10 of Diamonds
Jack of Diamonds
Queen of Diamonds
King of Diamonds

4 of Hearts
<facedown>
37
Clubs
Queen
<facedown>
Queen of Clubs
1 blacks, 1 greens, 4 reds, 4 blues - totaling $149
156
1 blacks, 2 greens, 1 reds, 1 blues - totaling $156
84
0 blacks, 2 greens, 4 reds, 2 blues - totaling $72
1 blacks, 3 greens, 3 reds, 2 blues - totaling $192
192
```

4. Reflection

This was a simple and fast lab to do as most of the thinking was already put into the project documentation, and both of us are familiar with the basic concepts of Object Oriented programming. The parts of the project that burned up the most time were small bugs, which teaches the lesson that, rather than microscoping into a small part of the function where the program says the error is occurring, one should scope out and look at the general flow of execution from the beginning of the function to determine what is going on. Also, it may have

been useful to adopt a more granular style of programming, though I also think that wasn't need for this lab. All in all, this was a fun, small project that illustrated the basic concepts of classes and Object Oriented programming.