Lab 10 Casino Night

Elliot Graves Brigham Ray

11-26-2016

Lab Section 4

Casino Night

Understand the Problem:

The programmer must create a program with 2 classes, chipBank and Card that

mimic the results of a casino game. When the test code is implemented it should return certain values from the string methods in each class that are meant to be returned as a result. The str method in the Card class should return the Card's name (1, 2, 3, king, queen, jack) and its suit if the card is not face down. If the card is face\_up then the return of the str method is “<face down.>”.

Assumptions: The class must be changed because there is a name error \_\_call\_\_ meaning the test code object name is too similar to the name of the class. The Card class name is changed to CardBaby to compensate for the error which is received when the code first runs.

Plan the Solution:

To start, the first class created is the Card Class. At first the class is called Card but changed to CardBaby because there are too many similarities between the object in the test code and the Class name.

class CardBaby():

The first method gives each object created the attributes necessary for use in the next methods within the Card class. These include self.suit, self.value, self.name, and self.cardnum. These attributes are used in other methods.

def \_\_init\_\_(self, card\_num):

self.name = ""

self.card\_num = card\_num

# card\_dictionary = {}

# self.name = name

# card\_dictionary.update({name: card\_num})

if self.card\_num <= 13:

self.suit = "Spades"

if self.card\_num == 1:

self.name = "Ace"

self.value = "11"

elif self.card\_num == 2:

self.name = "2"

self.value = self.name

elif self.card\_num == 3:

self.name = "3"

self.value = self.name

elif self.card\_num == 4:

self.name = "4"

self.value = self.name

elif self.card\_num == 5:

self.name = "5"

self.value = self.name

elif self.card\_num == 6:

self.name = "6"

self.value = self.name

elif self.card\_num == 7:

self.name = "7"

self.value = self.name

elif self.card\_num == 8:

self.name = "8"

self.value = self.name

elif self.card\_num == 9:

self.name = "9"

self.value = self.name

elif self.card\_num == 10:

self.name = "10"

self.value = self.name

elif self.card\_num == 11:

self.name = "Jack"

self.value = 10

elif self.card\_num == 12:

self.name = "Queen"

self.value = 10

elif self.card\_num == 13:

self.name = "King"

self.value = 10

elif self.card\_num <= 26:

self.card\_num -= 13

self.suit = "Hearts"

if self.card\_num == 1:

self.name = "Ace"

self.value = "11"

elif self.card\_num == 2:

self.name = "2"

self.value = self.name

elif self.card\_num == 3:

self.name = "3"

self.value = self.name

elif self.card\_num == 4:

self.name = "4"

self.value = self.name

elif self.card\_num == 5:

self.name = "5"

self.value = self.name

elif self.card\_num == 6:

self.name = "6"

self.value = self.name

elif self.card\_num == 7:

self.name = "7"

self.value = self.name

elif self.card\_num == 8:

self.name = "8"

self.value = self.name

elif self.card\_num == 9:

self.name = "9"

self.value = self.name

elif self.card\_num == 10:

self.name = "10"

self.value = self.name

elif self.card\_num == 11:

self.name = "Jack"

self.value = 10

elif self.card\_num == 12:

self.name = "Queen"

self.value = 10

elif self.card\_num == 13:

self.name = "King"

self.value = 10

self.card\_num += 13

elif self.card\_num <= 39:

self.card\_num -= 26

self.suit = "Clubs"

if self.card\_num == 1:

self.name = "Ace"

self.value = "11"

elif self.card\_num == 2:

self.name = "2"

self.value = self.name

elif self.card\_num == 3:

self.name = "3"

self.value = self.name

elif self.card\_num == 4:

self.name = "4"

self.value = self.name

elif self.card\_num == 5:

self.name = "5"

self.value = self.name

elif self.card\_num == 6:

self.name = "6"

self.value = self.name

elif self.card\_num == 7:

self.name = "7"

self.value = self.name

elif self.card\_num == 8:

self.name = "8"

self.value = self.name

elif self.card\_num == 9:

self.name = "9"

self.value = self.name

elif self.card\_num == 10:

self.name = "10"

self.value = self.name

elif self.card\_num == 11:

self.name = "Jack"

self.value = 10

elif self.card\_num == 12:

self.name = "Queen"

self.value = 10

elif self.card\_num == 13:

self.name = "King"

self.value = 10

self.card\_num += 26

elif self.card\_num <= 52:

self.suit = "Diamonds"

self.card\_num -= 39

if self.card\_num == 1:

self.name = "Ace"

self.value = "11"

elif self.card\_num == 2:

self.name = "2"

self.value = self.name

elif self.card\_num == 3:

self.name = "3"

self.value = self.name

elif self.card\_num == 4:

self.name = "4"

self.value = self.name

elif self.card\_num == 5:

self.name = "5"

self.value = self.name

elif self.card\_num == 6:

self.name = "6"

self.value = self.name

elif self.card\_num == 7:

self.name = "7"

self.value = self.name

elif self.card\_num == 8:

self.name = "8"

self.value = self.name

elif self.card\_num == 9:

self.name = "9"

self.value = self.name

elif self.card\_num == 10:

self.name = "10"

self.value = self.name

self.value = self.name

elif self.card\_num == 11:

self.name = "Jack"

self.value = 10

elif self.card\_num == 12:

self.name = "Queen"

self.value = 10

elif self.card\_num == 13:

self.name = "King"

self.value = 10

self.card\_num += 39

else:

print("This card doesn't exist")

The next methods return the various values of the attributes we assigned to an object created.

def get\_suit(self):

return str(self.suit)

def get\_rank(self):

return str(self.name)

def get\_value(self):

return str(self.value)

The next 2 methods determine if a card object is either face up or face down. The return of the methods is visualized in the str class which returns the card name and its suit if it is face up, but returns the str “face down.” if it is face down.

def face\_up(self):

global card\_face

card\_face = str(self.name + " of " + self.suit)

return card\_face

def face\_down(self):

global card\_face

card\_face = "<face down.>"

return card\_face

def \_\_str\_\_(self):

global card\_face

if card\_face != "<face down.>" and self.card\_num > 0:

return (str(self.name) + " of " + str(self.suit))

elif card\_face == "<face down.>" and self.card\_num > 0:

return "<face down.>"

The next class is the Chipbank class.

class ChipBank():

The first init method determines the object's initial self.value when it is first created like in a poker game when someone bets an initial amount.

def \_\_init\_\_(self, chipValue):

self.value = chipValue

The next two methods, deposit and withdraw, determine the amount when chips are either deposited or withdrawn. The user can deposit an infinite amount of chips although they can only take out as many chips as there are in the chipBank. If they take out all the chips the total can at the very least be equal to zero.

def withdraw(self, amount):

if self.value - amount < 0:

extra = -1 \* (self.value - amount)

amount\_withdrawn = amount - extra

self.value -= amount\_withdrawn

return amount\_withdrawn

else:

self.value -= amount

return amount

def deposit(self, amount):

self.value += amount

The last method visualizes the total amount in the chip bank in terms of chips. Each different chip color has a different value and the least amount of chips that can be used are used to display the total number of chips that the object has.

def \_\_str\_\_(self):

blacks = int(self.value / 100)

remainder\_after\_blacks = self.value % 100

greens = int(remainder\_after\_blacks / 25)

remainder\_after\_greens = remainder\_after\_blacks % 25

reds = int(remainder\_after\_greens / 5)

remainder\_after\_reds = remainder\_after\_greens % 5

blues = remainder\_after\_reds

return (str(blacks) + " blacks " + str(greens) + " greens " +

str(reds) + " reds " + "and " + str(blues) +

" blues, totaling " + str(self.value))

Implementation and Testing:

The code runs according to plan with no apparent issues. In the test code, card 37 should represent the Jack, not the Queen as it shows in the example code. When the code runs it shows that the card 37 is actually supposed to by the Jack instead of the Queen.

Reflect:

The code runs pretty much exactly as it should, although the first init method in the Card class is extremely redundant, a loop could have been used to do the exact same thing as the init method is supposed to do. Why a loop couldn't be used is unkown but it would have eliminated pretty much half of the code that doesn't need to be there. In terms of the other class though, there is no redundancy that doesn't need to be there.

Refactor:

A loop could be used in the first class to keep from the unnecessary if and elif statements that made the code extremely ambiguous.