Casino Night

psp-9-7

1 Overview

In this lab you will be creating a set of classes that model objects you might find used in a casino. The first of these classes is the Card class, representing a standard playing card from a 52 card deck. The second will be a ChipBank, representing a collection of chips for wagering. Once we have these classes, we can use them in the future to make casino games much easier. Cards will have the ability to be hidden, and information about the card will be easy to extract. ChipBank will hold a balance for us, and will display the balance using counts of different chips (e.g., 1 black, 2 red). Each of these method implementations should be short, and some methods can be easily implemented by calling other methods on the same object.

2 Learning Outcomes

By the end of this project students should be able to:

- read and write programs that utilize object-oriented programming;
- read and write programs with classes and be able to create objects;
- read and write programs that use method calls;
- work effectively with a partner using pair-programming;
- write an effective report that describes the students' problem solving process.

3 Pre-Lab Instructions

Do this part before coming to lab:

- Read Chapter 9: Classes and Objects
- Develop an algorithm to express monetary values using the fewest number of bills and coins as possible. For example, $8.07 would be 1 five, 3 ones, one nickle and two pennies.

- By hand, use the algorithm you created in the previous step to break $12.63 into the smallest number of bills and coins possible.
- Be prepared to show your work to the lab aide at the beginning of lab.

4 Lab Instructions

Do this part in lab:

4.1 Restrictions

These restrictions apply only to the code contained in the classes. In your test code outside the classes, you may ignore these requirements.

- These requirements must be met without using any while loops or for loops.
- The strings "Ace", "King", "Queen", "Jack", "Spades", "Hearts", "Clubs" and "Diamonds" may only appear once in the lab.

4.2 Card

Write and test a class called "Card" that works according to the following specifications. Each instance of this class represents a single card. Just like real cards, you can create multiple card objects with the same rank and suit if you chose to do so. In fact, you'll create 52 to represent a deck of cards. Following is a list of the methods that the Card class must implement:

__init__(card_num)

The __init__() method always contains code that runs when an object is created (or initialized). This is a good place to store initial values. You may add to this method as needed later in the assignment, as you decide you need additional variables to work with. This method accepts a card_num parameter indicating which of the 52 cards we mean to create. This method will need to set one or more instance variables (e.g., suit, and rank) to record which card is being represented in this specific card object.

parameters

- card_num - between 0 and 51, indicating which card in the deck it is. The first thirteen represent the ace of spades through the king of spades, the next thirteen represent the hearts, then clubs and diamonds.

get_suit()

This method returns the suit of the card as a string, and should contain Spades, Hearts, Clubs or Diamonds depending on the card.

return

This method should return a string containing the name of the suit.
Example: “Diamonds” or “Spades”

get_rank()

This method returns the rank of the card as a string, and should contain Ace, Jack, Queen, King or the number of the card.

return

This method should return a string containing the rank of the card. Either a number or title.
Example: “7” or “King”

get_value()

This method returns a numeric value depending on the point value of the card at hand. Face cards have a value of 10, Aces are valued at 11 and other cards are valued at their numeric rank.

return

This method should return a numeric value for the card.
Example: 7 or 10

face_down()

This method flips the card face-down, as cards are initialized they should be ’face down’ by default. The class variable set by this method should be accessible in the __str__() method. This method has no parameters or return values.

face_up()

This method does the opposite of face_down(). If a card is hidden and you call face_up() it should no longer be hidden. The class variable set by this method should be accessible in the __str__() method. This method has no parameters or return values.

__str__()

This method returns the card in string representation if the card is face-up. example: “King of Hearts” or “4 of Clubs”. If the card is face-down, it should return “<facedown>” no matter what card it is.

return

Should return a string representing the full name of the card.

Example: “7 of Diamonds” or “King of Spades”

4.3 ChipBank

Write and test a class called “ChipBank” that works according to the following specifications. Following is a list of the methods and instance variables that the ChipBank class must implement:

__init__(value)

Creates a new ChipBank object with an initial monetary balance eqaul to the value passed in via the parameter. The instance variable ’balance’ will be used to record the current value of the ChipBank class and should be an integer.

parameters

- value - Numeric value to start the ChipBank balance at.

withdraw(amount)

Reduces the balance of the ChipBank by the given amount (passed in via the parameter). If the balance of the ChipBank is lower than the amount requested for withdrawal, the balance drops to zero. The function should return the amount actually withdrawn (i.e., the amount passed in, or the drained balance).

parameters

- amount - Numeric amount to withdraw. Example: 100 or 0

return

Numeric value actually withdrawn.

Examples:

If the remaining balance is 57 and ’amount’ is 100, the returned value is 57. After this call, the remaining balance is 0.

If the remaining balance is 100, and the ’amount’ is 40, the returned value is 40. After this call, the remaining balance is 60.

deposit(amount)

Increases the balance of the ChipBank by the given amount. Does not return a value.

parameters

- amount - Numeric amount to deposit. Example: 70

get_balance()

The only functionality of this method is to see the current balance of the ChipBank

parameters

- None

return

Numeric balance currently held in ChipBank.

__str__()

Returns the object as a string describing the different chips held as well as the total balance. The number of chips listed should be the minimum number to represent the total balance. You should list the number of black, green, red, and blue chips where black chips are worth 100, green are worth 25, red are worth 5, and blues are worth 1.

return

Example: a ChipBank with a balance of 163 would be represented as "1 blacks, 2 greens, 2 reds, 3 blues - totaling $163"

4.4 Testing

Test your code using the following code, following your class definitions.

```
##### DO NOT CHANGE TEST CODE #####
import random
if __name__ == "__main__":
    # Lets make a deck of cards
    deck = []
    for i in range(52):
        my_card = Card(i)
        deck.append(my_card)
        # flip over each card
        my_card.face_up()
        # print each card as we add them
        print(my_card)

    # print a random card from the deck
```

```
15      print(random.choice(deck))
16
17      # In my implementation, card number 37 is the queen of clubs
18      card = Card(37)
19      print(card)
20      # Queen of Clubs
21      print(card.get_value())
22      # 10
23      print(card.get_suit())
24      # Clubs
25      print(card.get_rank())
26      # Queen
27      card.face_down()
28      print(card)
29      # <facedown>
30      card.face_up()
31      print(card)
32      # Queen of Clubs
33
34
35      cs = ChipBank(149)
36      print(cs)
37      # 1 blacks, 1 greens, 4 reds, 4 blues - totaling $149
38      cs.deposit(7)
39      print(cs.get_balance())
40      # 156
41      print(cs)
42      # 1 blacks, 2 greens, 1 reds, 1 blues - totaling $156
43      print(cs.withdraw(84))
44      # 84
45      print(cs)
46      # 0 blacks, 2 greens, 4 reds, 2 blues - totaling $72
47      cs.deposit(120)
48      print(cs)
49      # 1 blacks, 3 greens, 3 reds, 2 blues - totaling $192
50      print(cs.withdraw(300))
51      # 192
```

4.5 Extra Credit

Extra credit can be earned by implementing additional functionality for the ChipBank class.

record(handle)

By calling this function with a file handle as a parameter, the ChipBank begins to log deposits and withdraws. Each time a deposit or withdraw is made a line should be written to the file handle showing the remaining value, and the change in balance separated by a tab. By calling this function again and passing 'None' as a parameter, logging is disabled. You should include code to capture the changes being made in the test code to output to a file.

Example file output:

```
245 -5
255 10
355 100
305 -50
```

4.6 Pep8

When you have completed the lab run pep8 against your code until all formatting errors have been corrected and your code is PEP 8 compliant. See the Getting Started lab if you need instructions on running the program, or the pep8 documentation found here.

5 Lab Report Lab Report Format and Guidelines

Each pair of students will write a single lab report together and each student will turn in that same lab report on BBLearn. Submissions from each student on a pair should be identical.

Your lab report should begin with a preamble that contains:

- The lab assignment number and name
- Your name(s)
- The date
- The lab section

It should then be followed by four numbered sections:

1. Problem Statement

In this section you should describe the problem in **your** own words. The problem statement should answer questions like:

- What are the important features of the problem?
- What are the problem requirements?

This section should also include a reasonably complete list of requirements in the assignment. Following your description of the problem, include a bulleted list of specific features to implement. If there are any specific funtions, classes or numeric requirements given to you, they should be represented in this bulleted list.

2. Planning

In the second section you should describe your planning process. Identify any input and output required. Develop an algorithm (or listing of steps) to solve the probelm. Identify variables you'll need and coding elements you'll use. List specifcs about data structures, functions, and/or classes you plan to use and why.

3. Implementation and Testing

In the third section you should describe how you implemented your plan and discuss how it went. You should also include:

- a well-commented copy of your source code with the appropriate header (i.e., lab assignment number and name, author names, date, and lab section) submitted in BBLearn as a .py file
- a screen shot of your running application / solution
- a screen shot showing pep8 compliance
- test results (as needed)

4. Reflection and Refactoring

In the last section you should reflect on the project. Make sure you solution meets all the requirements identified in Step 1, and then summarize the key aspects of your approach and the coding techniques you implemented. Also, consider things you could have done to make your solution better. This might include code organization improvements, design improvements, etc. Ask yourself if there were alternative approaches or techniques you could have employed? How would these alternatives have impacted a different solution? Discuss how you have (or could) refactor your code based on these observations.

5. Partner Rating

Every assignment you are required to rate your partner with a score -1, 0 or 1. This should be submitted in the comment section of the BBLearn submission, and not in the report document. You do not have to tell your partner the rating you assign them. A rating of 1 indicates that your partner was particularly helpful or contributed exceptional effort. A rating of 0 indcates that your partner met the class expectations of them. Rating your partner at -1 means that they

refused contribute to the project, failed to put in a resonable effort or actively blocked you from participating. Be sure to include a comment along with any -1 or +1 ratings. If a student recieves three ratings of -1 they must attend a mandatory meeting with the instructor to dicuss the situation, and recieving additional -1 ratings beyond that, the student risks losing a letter grade, or even failing the course.

Colophon

This project was developed by Dr. James Dean Palmer of Northern Arizona University.