COM110
Programming Assignment 5
Black Jack using object-oriented programming (OOP)

Due: Monday, Nov 18, before 11:55 pm.
Mid-point check due: Tuesday, Nov 12, before 9:30 pm

Complete the assignment given below with your assigned partner.

Project partners. For this project you will work in teams of 3. Please indicate your teams here by end of Thursday Oct 31:

Working in teams. You will submit one copy of the program you write, so here are some guidelines:

1. You should always code together whenever possible, i.e., all team members at the same workstation at the same time. You are encouraged to use the many collaboration rooms in the library that have monitors to hook up your laptops to or lab machines throughout campus including in NLH214 so that it is easy for all of you to see the code at the same time. (Coding together on a laptop is not easy because the keyboard is attached to the screen so only one person can see well at a time.)
2. Alternate who does the typing so that each person gets to code for a third of the time. Do not let the same person be the typist for all the code.
3. Set up a schedule of meeting times right away. Two or three meetings a week of 2+ hours each is probably a good start. (I would schedule meetings that total at least 6 hours a week, and keep another 3 per week on reserve.)
4. Discuss the code openly with your partners. You should challenge each other and question each other so that everyone ends up with a better understanding of everything. Having to explain stuff to each other is the *best* way to learn and check for correctness!
5. There will be a partnering survey afterward so that you'll have a chance to give feedback to me on your partnering experience. In most cases all team members will receive the same grade, but this feedback could be taken into account in the grading process.
6. If any of your partners does not reply to you within 24 hours, let me know asap!

Submission. Submit your source code by placing it in a folder called **pa5_xxxxx,** where xxxxx is the login name of one student in your programming team. Zip this folder and upload it to Moodle. *Remember to submit a directory structure that has all files needed to run your program as soon we unpack it, including the folder of playing card images, any .py files that you import and any .txt files you might make use of for the extra credit.*

Project description and specification. Your assignment is to write an object-oriented Black Jack program with at least three classes that you will define and use.

Start by reviewing the rules of Black Jack (here), and then taking some time to gain experience by playing online (here), or play with the sample assignment solution below. It's very important that you feel comfortable and familiar with the game before you proceed. (**Features of the game like doubling down and splitting will not be a required part of this assignment.**)

Start setting up the classes for your program by completing a couple (non-graphical) programming exercises in the text that will be used for the project:

1. **Start** with programming exercise 11 from chapter 10 of Zelle (on pages 360-361, or page 333-334 in the 2nd edition). It outlines how to create a **PlayingCard** class.

Notes/hints/tips:

It is a good idea to test each class as you write it. You can write your test code in a main function at the bottom of your class module, the way we have been doing in classes. Just be sure to call the main function from within an `if __name__ == "__main__"` statement, the way we learned in lecture.

- The **PlayingCard** class is quite simple, just two instance variables, suit and rank. The methods getSuit() and getRank(), are called *accessor/getter methods*. These are methods that simply have one line of code that returns the value of the corresponding instance variable. Having such *accessor methods* for a class are an example of Encapsulation (which we learned about in lecture): hiding the instance variables and internals of the class/module from any programmer who is using the class and instead providing them with a public interface for indirectly inspecting those instance variables.
- To code the `__str__` method you should hard-code a **list** of the **ranks** (from "Ace" to "King") and hard-code a small **dictionary** of the **suits** (for example, mapping "s" to "Spades" etc.), then index into the list and the dictionary respectively using the rank and suit attributes.

2. Complete programming exercise 15 from chapter 11 (page 415, or page 381 in the 2nd ed) to create a class called **Deck** that represents a deck of cards. This class will import and use your **PlayingCard** class from the first exercise above. (That's why it's important that you have already tested your **PlayingCard** class from the chapter 10 exercise above.)

Hints/tips:

- The only instance variable you should need for this class is a list of **PlayingCard** objects. You can systematically create the complete list of playing cards by nesting two loops: loop through each suit (`for s in ["d", "c", "h", "s"]`) and within each suit, loop through each possible rank (`for r in range(1,14)`).
- One easy way to complete the `shuffle()` function is then to use the random module's shuffle() function. You will need to import the random module at the beginning of the program. Below code shows how to use shuffle(function).

```
#at the very beginning of Deck.py
from random import *

#let cardList be a list instance variable in Deck class
shuffle(self.cardList)
```

3. Using these two classes (along with the Button class we have already written together in lecture/lab), you can create a third **BlackJack** class that has instance variables and methods that allow you to implement a game of Black Jack (see details below). If you still feel unclear about the rules of Black Jack, you should read up on them here (http://www.blackjack-primer.com/basic_rules.php), and take some time to play online (http://www.hitorstand.net/game_m.html), or play with the sample assignment solution below.

Of course, the graphics and trimmings and added features are up to you. A folder (called **playingcards**) of images for the cards can be downloaded from Moodle. (part of assign5.zip file) I've also included a little sample program called **cardImageTest.py** so you can review and test how to include the images into a Zelle graphical window. Put this program in the same place that you put the playingcards folder. It also (as usual) needs graphics.py.

Your final result should achieve the basic functionality of the sample program: blackjackall.pyc.

- To run the program on a Windows machine, download it in the same directory where you have saved the **playingcards** folder (so the program has access to those images). Do not save it *into* the playingcards folder, but rather *next to* the playing cardsfolder. Then double-click it. (You must be on a machine with Python 3.7.* installed. And graphics.py has to be present, as always.)
- To get this working on a Mac instead, you can do the following: download the .pyc file, the graphics.py file and the **playingcards** folder to the **Desktop**, open Terminal (in Applications/Utilities/Terminal.app), and at the prompt, type

  `cd ~/Desktop`

  and hit enter, then type

  `python3 blackjackall.pyc`.

  If this does not work for you, please let me or a TA know. These are the same steps used during Lab09 to run the diceroller.pyc file.

Use the following class design for your program, which was used to create the sample program above. (If you wish to use a class design of your own, please run it by me first for approval.) Note that this **Blackjack** class, depends on you having completed the two text book programming exercises listed above, as it uses the **PlayingCard** class and the **Deck**

class. Indeed, this class `import`s and makes use of those classes. This class also imports and makes use of the **Button** class we wrote together in lecture/lab for the dice roller program.

```
    """Attributes of this Blackjack class are as follows.

       INSTANCE VARIABLES

        dealerHand: a list of PlayingCard objects representing the
dealer's hand
        playerHand: a list of PlayingCard objects representing the
player's hand
        playingDeck: a Deck object representing the deck of cards the
game is being played with

       METHODS

        __init__(self, dHand=[], pHand=[])
            constructor that initializes instance variables
            it also gives the playingDeck an initial shuffle
        initDeal(self,gwin,xposD,yposD,xposP,yposP):
            deals out initial cards, 2 per player and
            displays dealer and player hands on graphical win
            xposD and yposD give initial position for dealer cards
            xposP and yposP are analogous
        hit(self, gwin, xPos, yPos)
            adds a new card to the player's hand and places it at
xPos, yPos
        evaluateHand(self, hand)
            totals the cards in the hand that is passed in and
returns total
            (ace counts as 11 if doing so allows total to stay under
21)
        dealearPlays(self, gwin, xPos, yPos)
            dealer deals cards to herself, stopping when hitting
"soft 17"
    """
```

Finally, note that the sample program does not implement the feature of dealing the first dealer card face down. While doing this should not be a first-order concern in your implementation, after you get everything else working, you should also go back and fix that. One of the first two initial dealer cards should be face down until the player finishes her turn and the dealer starts going. It should then be revealed.

Some extra credit ideas:

- Expand the game so that it can be played again and again (so the user doesn't have to quit after each win or loss. It is shown in the sample program, blackjackall.pyc).

- Add money to the picture: allow the player to place a bet each game and keep a running score (how much the player is currently winning or losing).
    - Add a high score screen that displays high scores after the user quits, and allows the user to enter his/her name if her score beats out any previous high scores. (The high scores can be stored in a text file so that they can be recorded and recalled each time the game is played.)
- Give the player the option of being able to double-down or split (see Black Jack rules, http://www.blackjack-primer.com/basic_rules.php, and play the game online, http://www.hitorstand.net/game_m.html).
- Any other ideas that you would like to propose… let me know!

Mid-point checking (due Tuesday, Nov 12 before 9:30pm)

For the midpoint check (5% credit), your team needs to demonstrate following requirements.

- **PlayingCard** class: Programming exercise 11 from chapter 10 of Zelle
- **Deck** class: Programming exercise 15 from chapter 11 (page 383)
- Rough logic design for your Blackjack class and discuss it with TA