

Course Project Phase 1

This is the first phase of 3 phase project we will be doing in this course.

The project is to build a card game. The game is a simplified version of the Blackjack game. We will be building this in three steps, each of which will hopefully be easy to understand and implement. In the first phase of the project, you will create classes required to play any card game. Specifically, download the `objects.py` file that is provided. It has test-code in the `main()` method. You should define the three classes listed below in this file, uncomment the test code and run it. See sample runs below.

1. Create a Card class: Class to represent a playing card with the following requirements:

- a. Attributes:
 - i. **suit**: a public attribute representing suit of the card. Possible values: **"Spades", "Hearts", "Diamonds" and "Clubs"**
 - ii. **rank**: a public attribute representing rank of the card. Possible values: **"Ace", "2", "3", "4", "5", "6", "7", "8", "9", "10", "Jack", "Queen", "King"**.
- b. Properties:
 - i. **value**: A public read-only property that represents the point value of the card. It is only dependent on the rank of the card: An Ace is worth 11 points, Jack/Queen/King each is 10 points, and the rest of the cards are worth their rank value. E.g., Deuce (a "2") is worth 2 points, a "3" is worth 3 points, etc. *Hint: You can use the **int** function to convert the rank of the non-face cards to the corresponding point value.*
- c. A constructor:
 - i. This takes two required arguments: first one is rank and second one is suit and initializes the corresponding public attributes. *For consistency, make sure the order of the arguments is maintained – the first one is the rank and the second one is the suit. So `Card("King", "Hearts")` will create a card object representing "King of Hearts" and `Card("7", "Spades")` will create a card object representing "7 of Spades".*
- d. Methods
 - i. **displayCard**: a method that takes no parameters and returns a string of the form *"rank of suit suitsymbol"*. E.g., "King of Hearts ♥", "10 of Diamonds ♦", or "3 of Clubs ♣" *Hint: For your convenience a dictionary is provided at the top of the `objects.py` file that has these symbols corresponding to each suit. Use this dictionary here.*

2. Create a Deck class: Class to represent a deck of cards that is used to play a card game.

This class has the following requirements:

- a. Attributes:
 - i. **__deck**: a private attribute which is a list of objects of Card class.

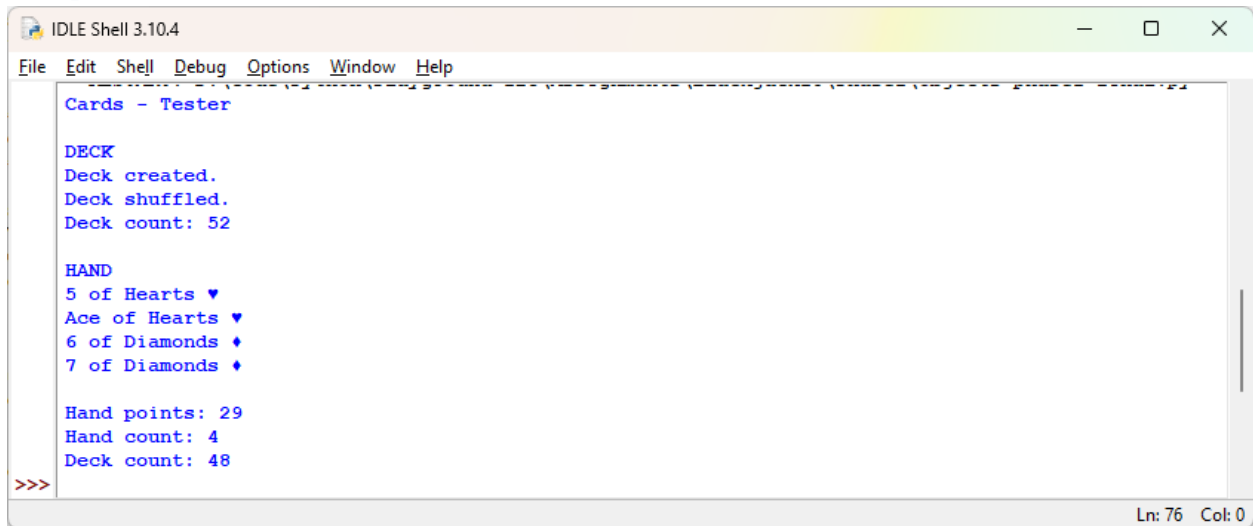
- b. Properties:
 - i. **count**: A read-only property equal to the count of cards currently in the `__deck` list.
- c. A constructor:
 - i. This takes no arguments. It initializes `__deck` attribute with 52 objects of Card class with one of each card type. That's 13 with suit = "Spades" and ranks from "2" to "Ace", 13 with suit = "Hearts" and ranks from "2" to "Ace", and so on, so that `__deck` has all unique 52 cards. These can be in perfect order. There are many ways to fill in this list of 52, using loop constructs. *Hint: One possible way is to use two lists, one initialized to 4 possible values of suit and other initialized to 13 possible values of rank. Then use a nested loop to loop over these two creating all possible pairings of suit and rank to create the 52 cards. But feel free to use any other way to use loops here.*
- d. Methods
 - i. **shuffle**: a method with no arguments that shuffles the deck to a random order. *Hint: Use the shuffle method of the random module that takes a list and shuffles in place.*
 - ii. **dealCard**: a method with no arguments that removes a card from the deck and returns it. It should return None if no cards left in the deck.

3. **Create a Hand class**: Class to represent the dealer's hand or the player's hand. This class has the following requirements:

- a. Attributes:
 - i. `__cards`: a private attribute which is a list of objects of Card class.
- b. Properties:
 - i. **count**: A read-only property equal to the count of cards currently in the `__cards` list.
 - ii. **points**: A read-only property equal to total points of the cards in the hand.
- c. A constructor: This takes no arguments. It initializes `__cards` attribute to an empty list.
- d. Methods:
 - i. **addCard**: a method that takes one argument of Card class object and appends it to the `__cards` list.
 - ii. **displayHand**: a method with no arguments that prints the hand by printing the strings returned by displayCard method of each of the cards in the `__cards` list.

Add these class definitions to the objects.py file, uncomment the test code and run it. See the sample runs below. Submit the file with the name of the form first_last_objects.py. Be sure to add the title comment-block at the top of the file giving details such as your name, class name, date etc

Sample Run 1:



```

IDLE Shell 3.10.4
File Edit Shell Debug Options Window Help
Cards - Tester

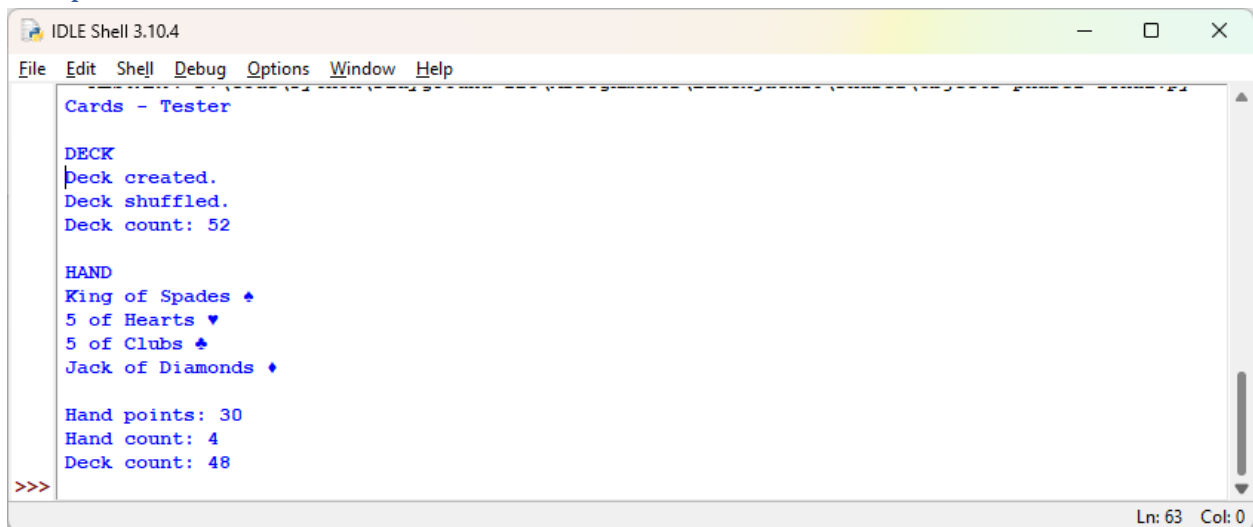
DECK
Deck created.
Deck shuffled.
Deck count: 52

HAND
5 of Hearts ♥
Ace of Hearts ♥
6 of Diamonds ♦
7 of Diamonds ♦

Hand points: 29
Hand count: 4
Deck count: 48
>>>
Ln: 76 Col: 0

```

Sample Run 2:



```

IDLE Shell 3.10.4
File Edit Shell Debug Options Window Help
Cards - Tester

DECK
Deck created.
Deck shuffled.
Deck count: 52

HAND
King of Spades ♠
5 of Hearts ♥
5 of Clubs ♣
Jack of Diamonds ♦

Hand points: 30
Hand count: 4
Deck count: 48
>>>
Ln: 63 Col: 0

```