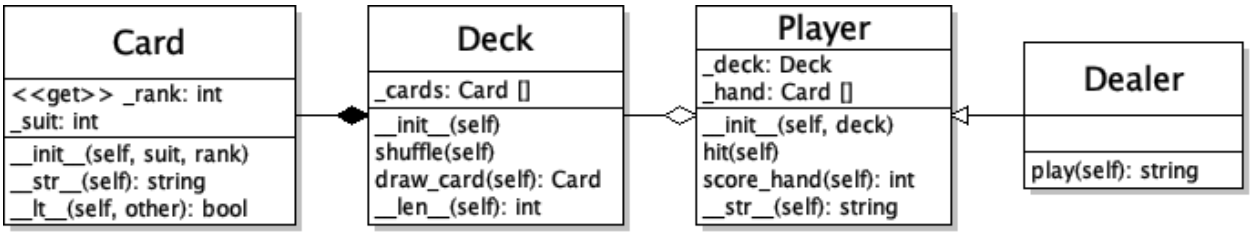
CECS 277 – Lab 7 – Inheritance

Blackjack

Create a program that plays a simplified version of the card game Blackjack. Two cards will be dealt to the player (the user) and to the dealer (the computer). Whoever has the highest score without going over 21 wins the round. Keep track of the number of rounds the player and the dealer have won.



Create a Card class (created in a separate file named 'card.py'):

Attributes:

1. `_rank` – an integer index 0-12, representing the card ranks of ['2', '3', '4', '5', '6', '7', '8', '9', '10', 'Jack', 'Queen', 'King', 'Ace']. Create a property for this attribute.
2. `_suit` – an integer index 0-3, representing the card suits of ['Clubs', 'Diamonds', 'Hearts', 'Spades']

Methods:

1. `__init__(self, suit, rank)` – assign the parameters to the attributes.
2. `__str__(self)` – returns a string in the format 'rank of suit' (ex. 'King of Clubs'). Hint: remember that the rank and suit are index values to the lists above.
3. `__lt__ (self, other)` – compares the ranks of the self and other cards. Returns true if the self rank is less than the other rank.

Create a Deck class (created in a separate file named 'deck.py'):

Attributes:

1. `_cards` – a list of Card objects that are in the deck.

Methods:

1. `__init__(self)` – initializes a standard deck of 52 cards. Thirteen ranks of each of the four suits.
2. `shuffle(self)` – shuffles the deck. Hint: the shuffle function in the random class can be passed a list.
3. `draw_card(self)` – remove the topmost card from the deck and return it.
4. `__len__(self)` – return the number of cards remaining in the deck.

Create a Player class (created in a separate file named 'player.py'):

Attributes:

1. `_deck` – a reference to the deck of cards that both the player and the dealer use.
2. `_hand` – a list of Cards that the player is currently holding.

Methods:

1. `__init__(self, deck)` – sets the deck attribute, deals two cards to the player's hand from the deck, then sorts the hand.
2. `hit(self)` – adds another card from the deck to the player's hand and resorts them.

3. `score(self)` – totals up the cards in the player's hand and returns that score. Card ranks 2-10 are face value, Jack, Queen, and King are 10, and Aces are scored at 11 if the total score of the other cards is less than 22, otherwise they count as 1. Hint1: since the cards are sorted, aces will always be counted last, so you will always know the total of the other cards. Hint2: it is handy to have a list of values for the ranks of the cards (remember that the card's rank attribute is an index value).
4. `__str__(self)` – displays each of the cards in the player's hand and the score of that hand.

Create a Dealer class (created in a separate file named 'dealer.py') that inherits from Player:
Methods:

1. `play(self)` – plays a round for the dealer. The dealer hits on scores 16 or lower, and stays on scores 17 or more. If their score goes over 21, then they bust. Make a loop that repeatedly hits until the dealer's score is more than 16. Build and return a string that repeatedly shows the dealer's hand, their score, hits, and whether they busted. Hint: you can use str(self) to call the str method of the Player class for the dealer.

Main file (created in a separate file named 'main.py') – create the following functions:

1. `display_winner(pScore, dScore, points)` – displays the winner of the round based on the player's and dealer's hand scores. If one player busted, then the other is the winner. If both players busted, then neither is the winner. If neither player busted, then compare the scores to see who won, if they tied, then nobody wins. The points parameter is a 2-item list that stores the number of rounds the player and dealer have won. Increment the appropriate counter given the winner of the round, then display the points.

When the program starts, construct a Deck and shuffle it. Create a loop that repeats until the user chooses to quit the game. Construct a Player object and pass it the deck, display their cards and score using the str method. Then repeatedly prompt them to hit until they choose to stay. If they bust, then display that they busted and end that loop. Then it's the dealer's turn. Construct a Dealer object, pass it the deck, and have it play the round. Then display the winner of the round by calling the display_winner function. If the user plays for several rounds, then the Deck will become depleted, reconstruct and reshuffle the deck before it runs out of cards (~12-15 cards remaining).

Example Output:

```
-Blackjack-

Player's Cards:
2 of Clubs
King of Clubs
Score = 12
1. Hit
2. Stay
Enter choice: 1

Player's Cards:
2 of Clubs
King of Clubs
King of Hearts
Score = 22
Bust!

Dealer's Cards:
Queen of Diamonds
Ace of Clubs
Score = 21

Dealer wins.
Player's points: 0
Dealer's points: 1
Play again? (Y/N): y

Player's Cards:
```

```
6 of Spades
10 of Clubs
Score = 16
1. Hit
2. Stay
Enter choice: 2

Dealer's Cards:
5 of Clubs
9 of Hearts
Score = 14
Dealer Hits!

Dealer's Cards:
4 of Diamonds
5 of Clubs
9 of Hearts
Score = 18

Dealer Wins.
Player's points: 0
Dealer's points: 2
Play again? (Y/N): y

Player's Cards:
3 of Clubs
8 of Diamonds
Score = 11
1. Hit
2. Stay
Enter choice: 1

Player's Cards:
3 of Clubs
8 of Diamonds
10 of Diamonds
Score = 21
1. Hit
2. Stay
Enter choice: 2

Dealer's Cards:
4 of Clubs
7 of Clubs
Score = 11
Dealer Hits!

Dealer's Cards:
4 of Clubs
7 of Clubs
Queen of Hearts
Score = 21

Tie
Player's points: 0
Dealer's points: 2
Play again? (Y/N): n
```

Notes:

1. Create 6 files: card.py, deck.py, player.py, dealer.py, check_input.py, and main.py.
2. Check the user's input using the get_int_range and get_yes_no functions.
3. Please do not create any extra attributes, methods, or parameters. You may create additional functions in main.py as needed.
4. Use a decorator to make the property for the Card's rank attribute.
5. Do not create global variables or use the attributes globally. Only access the attributes using the class's methods. Do not call any methods or attributes using the underscores.
6. Use docstrings to document the class, each of its methods, and the functions in main.py. See the lecture notes and the Coding Standards reference document for examples.
7. Please place your name, date, and a brief description of the program in a comment block at the top of your program (main.py). Place brief comments throughout your code.
8. This is a simplified version of the game Blackjack. Please do not add extra features such as betting, splitting, naturals, etc. You can add those features later for yourself, but please do not turn them in as it makes grading the assignments more difficult.
9. Thoroughly test your program before submitting:
    a. Make sure the cards are displayed properly in sorted order (by rank not suit).

b. Make sure that the score is correct for the cards displayed.
c. Make sure that the user input is validated.
d. Make sure that the user can continue hitting until they bust.
e. Make sure that the dealer can only hit up to a score of 16.
f. Make sure that the winner is reported correctly and points are incremented.
g. Make sure that the program refreshes the deck before it runs out of cards.

Blackjack Rubric – Time estimate: 5 hours

Blackjack **10 points**	Correct. 2 points	A minor mistake. 1.5 points	A few mistakes. 1 point	Several mistakes. 0.5 points	No attempt. 0 points
Card and Deck classes: 1. Created in a separate files. 2. Has correct attributes and property. 3. Deck is composed of Card objects. 4. Has correct methods.					
Player class (in separate file)**:** 1. Has deck and hand attributes. 2. Has methods: __init__, hit, score, and __str__. 3. __init__ draws two cards from the Deck and places them in the hand. 4. cards in hand are sorted 5. score calculates and returns the points in the Player's hand					
Dealer class (in separate file)**:** 1. Inherits from Player class. 2. Has play method that builds and returns the string representing the dealer's round of play. 3. Dealer hits on 16 and below.					
Main file (in separate file)**:** 1. Constructs Deck, Player, and Dealer. 2. Has loop to repeat until user quits. 3. Displays Player's hand and score. 4. Repeatedly prompts user to hit. 5. Dealer correctly plays round. 6. Correctly displays winner and score.					
Code Formatting: 1. Correct spacing. 2. Meaningful variable names. 3. No exceptions thrown. 4. No global variables or accessing attributes directly. 5. Correct documentation.					