Short Description

In this lab, you implement a simple card game. We represent a deck of 52 cards comprising 13 ranks in each of the four suits: Clubs, Diamonds, Hearts, Spades. Each suit includes an Ace, a two, a three, a four, a five, a six, a seven, an eight, a nine, a ten, a Jack, a Queen, and a King. You will create a simple game with two computer players. There are five cards in each hand and the player with more Ace cards is the winner. In case of a tie, when both players have the same number of Ace cards, the deck is created anew, the deck is shuffled and each player is dealt a new hand. All of this functionality will be implemented using Python classes[1].

Learning Outcomes

- Practice modeling real-life objects into code
- Discover new language features[2]
- Practice calling methods on class instances
- Learn how to organize your software using classes

Detailed Explanation and Things To Do

1. Below you can see a sample run of the game. In the first try, both players have one Ace card so there is no winner. Then players will shuffle all cards, and draw again. This time player two gets one Ace card but player 1 gets no Ace card. So player 2 wins the game.

```
This is the hand of player 1:
Eight of Clubs
Seven of Hearts
Six of Clubs
```

[1]https://upload.wikimedia.org/wikipedia/commons/thumb/8/81/English_pattern_playing_cards_deck.svg/1200px-English_pattern_playing_cards_deck.svg.png

[2] To create this game, you may need to use a few programming language features that have not been used yet in class. Discovering new language features and how to use them is an integral part of problem-solving in computing science and an essential skill that you should learn. Think about what you need to do, search the web for python3 programming examples, and/or use the Python documentation to help you find the programming constructs that you need. If you get stuck, ask your TA for help/hints about the programming constructs you need to use.

```
King of Spades
Ace of Hearts

This is the hand of player 2:
Queen of Hearts
Ten of Hearts
Nine of Hearts
Ace of Spades
Two of Hearts

Number of ace cards in each player's hand:
Player 1 has 1 aces
Player 2 has 1 aces

Result:
No winner, shuffle again

This is the hand of player 1:
Seven of Spades
Three of Clubs
Jack of Diamonds
Nine of Spades
Two of Diamonds

This is the hand of player 2:
Ten of Hearts
King of Hearts
Seven of Clubs
Six of Diamonds
Ace of Spades

Number of ace cards in each player's hand:
Player 1 has 0 aces
Player 2 has 1 aces

Result:
Player 2 is the winner
```

2. Your code must implement the major logical tasks, such as representing the cards and the deck as classes, displaying the cards, and determining the value of a hand by counting the aces.
3. Internally, we use an integer rank for each card, with ranks: Ace = 1, Two = 2,...Ten = 10, Jack = 11, Queen = 12, King = 13.

4. Make sure you are following code quality standards outlined in the Software Quality Tests with Classes. ***You must use getter and setter methods to ensure that Section 7.2 of Software Quality Tests is not violated.*** Here is an example of how a violation of Section 7.2 may occur and how to use getter and setter methods to avoid this violation.
5. Implement the following classes and their methods.

    a. **`Card`**

    i. **`__init__(self, rank, suit)`**
    - Description: Initialize a card with a given suit and a rank
        - rank - an integer between 1-13
        - suit- a string ("Clubs", "Diamonds", "Hearts", "Spades")

    ii. **`get_rank(self)`**
    - Description: Return the rank of a card

    iii. **`display(self)`**
    - Description: Display a card as a string with the name of the rank and the name of the suit (e.g. `'Jack of Diamonds'`, `'Three of Clubs'`)

    b. **`Deck`**

    i. **`__init__(self)`**
    - Description: Initialize a deck by generating one of each possible card using the Card class. The deck should contain one Card object for each of the 4*13=52 different cards.

    ii. **`shuffle(self)`**
    - Description: Randomly shuffle the deck of cards

    iii. **`deal(self)`**
    - Description: Return a card from the top of the deck, and remove that card from the deck.

    c. **`Player`**

    i. **`__init__(self)`**
    - Description: Initialize a player who can keep track of the cards in their hand. Initially, the hand should be empty (no cards in it).

    ii. **`add(self, card)`**
    - Description: Add a card to the player's hand
        - card - An instance of the `Card` class.

    iii. **`ace_cards(self)`**
    - Description: Returns the number of Ace cards in the player's hand.

    iv. **`display(self)`**
    - Description: Display the player's hand. This method should call the display() method of each card object in the player's hand.

6. The main function should handle:
    a. Create one deck
    b. Shuffle the deck
    c. Create both players
    d. Populate the player's hands with 5 cards each
    e. Display the player's hands
    f. Display the number of Ace cards in each player’s hand
    g. Display the winner.
    h. If the game ends in a tie, then restart the game from step a.

Submission Information

- You are required to submit the solution for this lab exercise by the due date provided. For submission purposes, the file with your code should be named:

  `cards.py`