Object-Oriented Design: War Card Game

Now let's design a card game. If you've never played War before (or if you have), here's the variant of the game that we will play. The deck is split evenly between two players. On each turn, both players reveal their first card. The player showing the card with the highest rank takes both cards and adds them to his or her deck, and reshuffles their hand. If there is a tie, the players reveal their next cards. If there is no longer a tie, the player with the highest rank takes all four cards on the revealed stack into their hand (and reshuffles their hand). Otherwise, both players continue revealing their next cards until the tie is broken. The game continues until one player has collected all 52 cards.

What classes and methods do we need? Each card has a rank and a suit, so it makes sense to make a **Card** class with these attributes. In our game, "greater" cards are those with higher rank; cards with the same rank are "equal", no matter what suit they come from.

A **Hand** is a collection of Cards, so we'll make that another class. We can give and take cards from a Hand, and also shuffle the cards in a Hand. We'll want to know how many Cards are in a Hand, so we'll need a `num_cards` method, among others.

A **Deck** is a special kind of Hand, populated with (in our case) the standard 52 cards. We can `deal` cards from a Deck.

A **Player** is also a Hand of cards, but with a (person) name also.

Finally, the **Game** class implements all of the game logic. This class implements methods for dealing hands, taking a turn, and playing a game to determine a winner.

Card class

In [ ]:
```
class Card():
    suit_str = {'S': "\u2660", 'H': "\u2661", 'C': "\u2663", 'D': "\u2662"}
    rank_str = {n: str(n) for n in range(2,11)}
    rank_str[11] = 'J'; rank_str[12] = 'Q'; rank_str[13] = 'K'; rank_str[14] = 'A'

    def __init__(self, rank, suit):
        """ rank: integer from 2 to (including) 14
            suit: 'S' for Spades, or 'H' for Hearts,
                  or 'C' for Clubs, or 'D' for Diamonds
        """
        assert suit in "SHCD"
        assert rank in range(2,15)
        self.rank = rank
        self.suit = suit

    def __gt__(self, other): #other might be None
        return self.rank > other.rank if other else True

    def __lt__(self, other):
        return self.rank < other.rank if other else False

    def __eq__(self, other):
        return self.rank == other.rank if other else False

    def __str__(self):
        return "{}{}".format(self.rank_str[self.rank], self.suit_str[self.suit])

    def __repr__(self):
        return "Card({},'{}')".format(self.rank, self.suit)
```

In [ ]:
```
Card(2,'H') == Card(2,'S')
```

In [ ]:
```
Card(2,'H') < None
Card(2, 'H')
str(Card(2, 'H'))
```

In [ ]:
```python
cards = [Card(3,'S'), Card(14,'D'), Card(10,'D'), Card(14,'H')]
print("cards:", cards) # note: str(<list>) creates a string of the list of <repr of list elements>
print("max:", max(cards))
print("min:", min(cards))
print("position of max card:", cards.index(max(cards)))
print("sorted:", [str(c) for c in sorted(cards)])
print("reverse sorted:", [c for c in sorted(cards, reverse=True)])
```

Hand class

In [ ]:
```python
import random
class Hand():
    def __init__(self):
        self.cards = []

    def receive_cards(self, cards):
        """ Receive sequence of cards into hand """
        for c in cards:
            self.receive_card(c)

    def receive_card(self, card):
        """ Receive single card into bottom of hand
            (if card is not None) """
        pass

    def shuffle(self):
        """ Shuffle the deck by rearranging the cards in random order. """
        if self.cards:
            random.shuffle(self.cards)

    def give_card(self):
        """ Remove and return the card at the top of the hand. If there
            are no more cards, return None """
        pass

    def give_back_cards(self):
        """ Remove and returns all cards in the hand """
        pass

    def num_cards(self):
        pass

    def __str__(self):
        return str([str(card) for card in self.cards])
```

In [ ]:
```python
h = Hand()
c = Card(2,'S')
h.receive_card(c)
print("num_cards:", h.num_cards())
print("alt_num_cards:", h.alt_num_cards)
```

Deck class

In [ ]:
```
class Deck(Hand):
    def __init__(self):
        super().__init__()
        self.receive_cards(Deck.build_deck())
        self.shuffle()

    @staticmethod
    def build_deck():
        """
        Return a list of the 52 cards in a standard deck.

        Suits are "H" (Hearts), "S" (Spades), "C" (Clubs), "D" (Diamonds).
        Ranks in order of increasing strength the numbered cards
        2-10, 11 Jack, 12 Queen, 13 King, and 14 Ace.
        """
        suits = {"H", "S", "C", "D"}
        return [Card(rank, suit) for rank in range(2,15) for suit in suits]

    def deal(self):
        return self.give_card()
```

In [ ]:
```
d = Deck()
print(d)
```

Player class

In [ ]:
```
class Player(Hand):
    def __init__(self, name):
        self.name = name
        super().__init__()

    def draw_card(self, card):
        """ Add card to the player's hand. """
        self.receive_card(card)

    def reveal_card(self):
        """ Remove and return the first card in the hand. """
        return self.give_card()

    def __repr__(self):
        return "Player(" + repr(self.name) + ")"
```

Game class

In [ ]:

```python
class Game():
    def __init__(self, players):
        self.players = players
        self.deck = Deck()

    def deal_hands(self):
        """
        Deal cards to both players. Each player takes one card at
        a time from the deck.
        """
        pass

    def turn(self, do_print=False):
        """
        Reveal cards from both players. The player with the higher
        rank takes all the cards in the pile.
        """
        pile = [p.reveal_card() for p in self.players]

        # If there is a tie, get the next cards from each
        # player and add them to the cards pile.
        while pile[0] == pile[1]:
            if do_print: print("war:", [str(c) for c in pile], "TIE!")
            for i in range(len(self.players)):
                pile.insert(i, self.players[i].reveal_card())

        winner = self.players[0] if pile[0] > pile[1] else self.players[1]
        winner.receive_cards(pile)
        winner.shuffle()
        if do_print: print("war:", [str(c) for c in pile], "=>", winner.name)

    def play(self, do_print=False):
        """
        Keep taking turns until a player has won (has all 52 cards).
        Return the winning player.
        """
        self.deal_hands()
        while all(p.num_cards() < 52 for p in self.players):
            self.turn(do_print)

        for p in self.players:
            if p.num_cards() == 52:
                if do_print: print(p.name + " wins!")
                return p

    def play_n_times(self, n):
        for p in self.players:
            p.wins = 0

        for i in range(n):
            # Return all cards players are holding back to the deck
            for p in self.players:
                self.deck.receive_cards(p.give_back_cards())
            self.deck.shuffle()
            self.deal_hands()
            winner = self.play()
            winner.wins += 1
        print("\nPlayed", n, "hands")
        for p in self.players:
            print("  ", p.name, "wins:", p.wins)
```

Let's try it out...

In [ ]:

```python
game = Game([Player("Amy"), Player("Joe")])
game.play(do_print = True)
```

Many games

Let's extend so we can run many games and see who wins the most

```
In [ ]: game = Game([Player("Amy"), Player("Joe")])
        game.play_n_times(3)
        game.play_n_times(30)
        game.play_n_times(100)
```

Try with more than two players

```
In [ ]: game = Game([Player("Amy"), Player("Brad"), Player("Carl")])
        game.play_n_times(3)
        game.play_n_times(30)
        game.play_n_times(100)
```

Oops. We hard coded the game to only expect two players, so Carl can never win! We'll leave it as an exercise for the reader to go back and generalize `Game` to fix.