```
# Program: Hands Class
# Date: Nov. 29, 2022
# Programmer: Tahmid Ehsan
# Description: A blueprint made to store the hands of poker players.

# Imports aspects of other given classes
from Deck import *

# Defines class
class Hands():

    # Initializes class
    def __init__(self):
        self.rank = []
        self.value = []
        self.suit = []
        self.name = []

    # Adds information of more cards to given hand
    def add_2_hand(self,card):
        self.rank.append(card.get_rank())
        self.value.append(card.get_value())
        self.suit.append(card.get_suit())
        self.name.append(card.get_name())

    # Outputs the information of a hand
    def show_hand(self):
        return self.name

    # Displays a particular card from a hand at the user's request
    def show_specific_card(self, inc):
        return self.name[inc]

    # Clears stored information from lists about hands
    def clear_hands(self):
        self.rank.clear()
        self.value.clear()
        self.suit.clear()
        self.name.clear()

    # Changes specific cards in a hand
    def change_hand(self,current_card,rep_card):
        self.rank[current_card] = rep_card.get_rank()
```

```
  self.value[current_card] = rep_card.get_value()
  self.suit[current_card] = rep_card.get_suit()
  self.name[current_card] = rep_card.get_name()

 # Sorts the values of a given hand
def organize(self):
  self.value.sort()
  return self.value

 # Identifies what type of hand a person may have
def hand_type(self):
    suit_count = 0
    row_of_cards = 1
    of_a_kind = 0
    self.points = 0
    self.tie_breaker = 0

    # Determines if a hand is made up of the same suit
    for i in range(5):
        if (self.suit[0]== self.suit[i]):
          suit_count += 1

     # Determines how many carda are increasing in order
    for i in range(5):
        if i > 0:
            if (self.value[i] == (self.value[i-1] + 1)):
                row_of_cards += 1

     # Compares to see if first card is identical to second
    if (self.value[0] == self.value[1]):
        # If first two cards are identical, program determines
        # how many are identical in a row
        for i in self.value:
            if (i == self.value[0]):
                of_a_kind += 1
        self.tie_breaker = self.value[0]

        # If three are identical, program determines
        # if the other two are a pair
        if (of_a_kind == 3):
            if (self.value[3] == self.value[4]):
                of_a_kind += 2
```

```
    # If two are identical, program determines if there is
    # another pair and which one has the greater rank
    if (of_a_kind == 2):
        self.tie_breaker = self.value[0]
        if (self.value[2] == self.value[3]) or (self.value[3] == self.value[4]):
            of_a_kind +=4
            if (self.value[3] > self.value[1]):
                self.tie_breaker = self.value[3]

# If no pair could be found left to right, the same
# program is used to look for pairs from right to left
else:
    if (self.value[2] == self.value[3]):
        of_a_kind += 2
        self.tie_breaker = self.value[2]

if (self.value[3] == self.value[4]):
    for i in self.value:
        if (i == self.value[4]):
            of_a_kind +=1
    self.tie_breaker = self.value[4]
    if (of_a_kind == 3):
        if (self.value[0] == self.value[1]):
            of_a_kind += 2
    if (of_a_kind == 2):
        if ((self.value[2] == self.value[1]) or (self.value[1] == self.value[0])):
            of_a_kind +=4
            if (self.value[1] > self.value[4]):
                self.tie_breaker = self.value[1]

else:
    if (self.value[1]== self.value[2]):
        of_a_kind += 2
        self.tie_breaker = self.value[1]

# Program checks to make sure pairs or triples
# do not exist in the centre of the hand

if ((self.value[0] != self.value[1]) and (self.value[1] != self.value[4])):
    if (self.value[1] == self.value[3]):
        of_a_kind = 3
        self.tie_breaker = self.value[1]
```

```
# Based on the calculations from above, program
# determines what type of hand a person may have

if ((suit_count == 5) and (row_of_cards == 5)):
    self.tie_breaker = self.value[4]
    self.points = 9
    return "straight flush"

elif (of_a_kind == 4):
    self.points = 8
    return "four of a kind"

elif (of_a_kind == 5):
    self.points = 7
    return "full house"

elif (suit_count == 5):
    self.tie_breaker = self.value[4]
    self.points = 6
    return "flush"

elif (row_of_cards == 5):
    self.tie_breaker = self.value[4]
    self.points = 5
    return "straight"

elif (of_a_kind == 3):
    self.points = 4
    return "three of a kind"

elif (of_a_kind == 6):
    self.points = 3
    return "two pairs"

elif (of_a_kind == 2):
    self.points = 2
    return "pair"

else:
    self.points = 1
    self.tie_breaker = self.value[4]
    return "high card"
```

```
def game_tie_breaker(self):
    return self.tie_breaker

def game_points(self):
    return self.points
```