

```
1  """
2  Models a single card
3  """
4
5  JACK = 11
6  QUEEN = 12
7  KING = 13
8  ACE = 14
9  HEARTS = "H"
10 DIAMONDS = "D"
11 SPADES = "S"
12
13 class Card:
14
15     def __init__(self, rank, suit):
16         """
17         Constructor
18
19         :param rank: an integer for the rank of the card
20         :param suit: a string for the suit of the card
21         """
22         self.__card = {"rank": rank, "suit": suit}
23
24     def get_rank(self):
25         """
26         Gets the rank of the card
27
28         :return: an integer for the rank of the card
29         """
30         return self.__card["rank"]
31
32     def get_suit(self):
33         """
34         Gets the suit of the card
35
36         :return: an integer for the suit of the card
37         """
38         return self.__card["suit"]
39
40     def __str__(self):
41         """
```

```
42         Returns the readable version of the card
43
44         :return: a string for the readable version of the card
45         """
46         rank = self.get_rank()
47         suit = self.get_suit()
48
49         if rank == JACK:
50             rank_string = "Jack"
51
52         elif rank == QUEEN:
53             rank_string = "Queen"
54
55         elif rank == KING:
56             rank_string = "King"
57
58         elif rank == ACE:
59             rank_string = "Ace"
60
61         else:
62             rank_string = str(rank)
63
64         if suit == HEARTS:
65             suit_string = "Hearts"
66
67         elif suit == DIAMONDS:
68             suit_string = "Diamonds"
69
70         elif suit == SPADES:
71             suit_string = "Spades"
72
73         else:
74             suit_string = "Clubs"
75
76         return rank_string + " of " + suit_string
77
78 def __confirm_result():
79     card = Card(11, "C")
80
81     print(card)
82
```

```
83 if __name__ == "__main__":  
84     __confirm_result()
```

```
1  """
2  Models a deck of cards
3  """
4
5  from card import *
6  import random
7
8  RANKS = [2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14]
9  SUITS = ["H", "D", "S", "C"]
10 EMPTY = 0
11 TOP_CARD = 0
12 MAX_HAND = 5
13
14 class Deck:
15
16     def __init__(self):
17         """
18         Constructor
19         """
20         self.__deck = []
21
22         for rank in RANKS:
23             for suit in SUITS:
24                 self.__deck.append(Card(rank, suit))
25
26         self.__shuffle()
27
28     def __shuffle(self):
29         """
30         Shuffles the deck
31
32         :return: a list for the shuffled deck
33         """
34         return random.shuffle(self.__deck)
35
36     def __deal(self):
37         """
38         Deals and returns a single card by removing the top
39         card of the deck
40
41         :return: a Card object which is the top card of the
```

```
40 deck
41     """
42     if self.size() == EMPTY:
43         return None
44
45     else:
46         return self.__deck.pop(TOP_CARD)
47
48 def size(self):
49     """
50     Returns the number of cards left in the deck
51
52     :return: an integer for cards left in the deck
53     """
54     return len(self.__deck)
55
56 def __str__(self):
57     """
58     Returns the readable version of the deck
59
60     :return: a string for the readable version of the deck
61     """
62     return_string = ""
63
64     for card in self.__deck:
65         return_string += str(card) + "\n"
66
67     return return_string
68
69 def list_o_cards(self):
70     """
71     Creates a list of Card objects
72
73     :return: a list of Card objects
74     """
75     list_o_cards = []
76
77     for i in range(MAX_HAND):
78         list_o_cards.append(self.__deal())
79
80     return list_o_cards
```

```
81
82 def __confirm_result():
83     deck = Deck()
84
85     print(deck)
86
87 if __name__ == "__main__":
88     __confirm_result()
```

```
1 """
2 Testing utilities. Do not modify this file!
3 """
4
5 VERBOSE = True
6 num_pass = 0
7 num_fail = 0
8
9 def assert_equals(msg, expected, actual):
10     """
11     Check whether code being tested produces
12     the correct result for a specific test
13     case. Prints a message indicating whether
14     it does.
15     :param: msg is a message to print at the beginning.
16     :param: expected is the correct result
17     :param: actual is the result of the
18     code under test.
19     """
20     if VERBOSE:
21         print(msg)
22
23     global num_pass, num_fail
24
25     if expected == actual:
26         if VERBOSE:
27             print("PASS")
28             num_pass += 1
29     else:
30         if not VERBOSE:
31             print(msg)
32         print("**** FAIL")
33         print("expected: " + str(expected))
34         print("actual: " + str(actual))
35         if not VERBOSE:
36             print("")
37             num_fail += 1
38
39     if VERBOSE:
40         print("")
41
```

```
42
43 def fail_on_error(msg,err):
44     """
45     if run-time error occurs, call this to insta-fail
46
47     :param msg: message saying what is being tested
48     :param err: type of run-time error that occurred
49     """
50     global num_fail
51     print(msg)
52     print("*** FAIL")
53     print(err)
54     print("")
55     num_fail += 1
56
57
58 def start_tests(header):
59     """
60     Initializes summary statistics so we are ready to run
61     tests using
62     assert_equals.
63     :param header: A header to print at the beginning
64     of the tests.
65     """
66     global num_pass, num_fail
67     print(header)
68     for i in range(0,len(header)):
69         print("=",end="")
70     print("")
71     num_pass = 0
72     num_fail = 0
73
74 def finish_tests():
75     """
76     Prints summary statistics after the tests are complete.
77     """
78     print("Passed %d/%d" % (num_pass, num_pass+num_fail))
79     print("Failed %d/%d" % (num_fail, num_pass+num_fail))
80     print()
```



```
1 """
2 A simple poker game
3
4 :author: Chris Hegang Kim
5 :note: I affirm that I have carried out the attached academic
6       endeavors with full academic honesty,
7       in accordance with the Union College Honor Code and the course
8       syllabus.
9 """
10
11 from poker_hand import *
12
13 CONTINUE = True
14 TWO_MAX_HAND = 10
15
16 def main():
17     deck = Deck()
18     continue_game = CONTINUE
19     total_point = 0
20
21     while continue_game and deck.size() > TWO_MAX_HAND:
22         myhand = PokerHand(deck.list_o_cards())
23         other_hand = PokerHand(deck.list_o_cards())
24         result = myhand.get_result(other_hand)
25
26         print("my hand: ", myhand)
27         print("other hand: ", other_hand)
28
29         player_input = input("Who is the winner? (Type my hand
30 , other hand, or tie)")
31
32         if player_input == result:
33             total_point += 1
34
35         else:
36             continue_game = not CONTINUE
37
38     print("Game is over, and your total point is ",
39           total_point)
40
41 if __name__ == "__main__":
```

```
38     main()
```

```
1  """
2  Models a 5-card hand of cards
3  """
4
5  import copy
6  from deck import *
7
8  TIE = 0
9  MAX_HAND = 5
10 FIRST_CARD = 0
11 LAST_CARD = 1
12 FLUSH = 4
13 TWO_PAIR = 3
14 PAIR = 2
15 HIGH_CARD = 1
16
17 class PokerHand:
18
19     def __init__(self, list_o_cards):
20         """
21         Constructor
22
23         :param list_o_cards: a list of Card objects
24         """
25         self.__hand = copy.deepcopy(list_o_cards)
26
27     def __add_card(self, card):
28         """
29         Adds the card to the hand
30
31         :param card: a Card object
32         :return: the PokerHand object with the additional card
33         """
34         return self.__hand.append(card)
35
36     def __get_ith_card(self, index):
37         """
38         Returns the card at the given index
39
40         :param index: an integer greater or equal to 0
41         :return: a Card object at the given index
```

```

42         """
43         return self.__hand[index]
44
45     def __str__(self):
46         """
47         Returns the readable version of the hand
48
49         :return: a string for the readable version of the hand
50         """
51         return_string = ""
52
53         for card in self.__hand:
54             return_string += str(card) + "\n"
55
56         return return_string
57
58     def compare_to(self, other_hand):
59         """
60         Determines which of two poker hands is worth more.
61         Returns an int
62         which is either positive, negative, or zero depending
63         on the comparison.
64
65         :param self: The first hand to compare
66         :param other_hand: The second hand to compare
67         :return: a negative number if self is worth LESS than
68         other_hand,
69         zero if they are worth the SAME (a tie), and a
70         positive number if
71         self is worth MORE than other_hand
72         """
73
74         hand_point = self.__get_point()
75         other_point = other_hand.__get_point()
76
77         result = hand_point - other_point
78
79         if result == TIE:
80             if hand_point == FLUSH or hand_point == HIGH_CARD:
81                 self.__get_rank_list()
82                 other_hand.__get_rank_list()

```

```

79         for i in range (MAX_HAND):
80             result = self.__get_ith_rank(i) -
other_hand.__get_ith_rank(i)
81
82             if result ≠ TIE:
83                 return result
84
85         elif hand_point = TWO_PAIR or hand_point = PAIR
:
86
87         for i in range (self.__rank_list_size()):
88             result = self.__get_ith_rank(i) -
other_hand.__get_ith_rank(i)
89
90             if result ≠ TIE:
91                 return result
92
93         return result
94
95     def __get_point(self):
96         """
97         Gets the point of the hand
98
99         :return: an integer for the score of the hand
100        """
101        if self.__is_flush():
102            hand_score = FLUSH
103
104        elif self.__is_two_pair():
105            hand_score = TWO_PAIR
106
107        elif self.__is_pair():
108            hand_score = PAIR
109
110        else:
111            hand_score = HIGH_CARD
112
113        return hand_score
114
115     def __get_rank_list(self):
116         """

```

```
117         Gets the list with ranks in the descending order
118
119         :return:
120         """
121         rank_list = []
122
123         for card in self.__hand:
124             rank_list.append(card.get_rank())
125
126         rank_list.sort(reverse = True)
127
128         self.__rank_list = rank_list
129
130     def __get_ith_rank(self, index):
131         """
132         Returns the rank at the given index
133         :param index: an inter for the index of the rank
134         :return: an integer at the given index
135         """
136         return self.__rank_list[index]
137
138     def __rank_list_size(self):
139         """
140         Returns the size of the rank list
141
142         :return: an integer for the size of the rank list
143         """
144         return len(self.__rank_list)
145
146     def __copy(self):
147         """
148         Stores a copy of the original value
149
150         :param original: a value for the copy
151         :return: a copied value with the new reference
152         """
153         return copy.deepcopy(self)
154
155     def __size(self):
156         """
157         Returns the number of cards left in the hand
```

```

158
159         :return: an integer for cards left in the hand
160         """
161         return len(self.__hand)
162
163     def __remove_ith_card(self, index):
164         """
165         Removes the card with the given index
166
167         :param index: an integer for the index of the card
168         :return: a PokerHand object without the removed card
169         """
170         return self.__hand.remove(self.__hand[index])
171
172     def __is_flush(self):
173         """
174         Checks whether the hand is flush
175
176         :return: True if all cards have the same suit
177         """
178         for i in range(1, MAX_HAND):
179             if self.__get_ith_card(i).get_suit() != self.
__get_ith_card(i - 1).get_suit():
180                 return False
181
182         return True
183
184     def __is_two_pair(self):
185         """
186         Checks whether the hand is two pair
187
188         :return: True if the hand has 2 pairs of the same
rank
189         """
190         current_hand = self.__copy()
191
192         i = 1
193         total_pair = 0
194         pair_rank = []
195         other_rank = []
196

```

```
197         while current_hand.__size() > i:
198             if current_hand.__get_ith_card(FIRST_CARD).
get_rank() == current_hand.__get_ith_card(i).get_rank():
199                 total_pair += 1
200
201                 pair_rank.append(current_hand.__get_ith_card(
FIRST_CARD).get_rank())
202
203                 current_hand.__remove_ith_card(i)
204                 current_hand.__remove_ith_card(FIRST_CARD)
205
206                 i = 1
207
208             else:
209                 i += 1
210
211             if current_hand.__size() == i:
212                 other_rank.append(current_hand.__get_ith_card
(FIRST_CARD).get_rank())
213
214                 current_hand.__remove_ith_card(FIRST_CARD)
215
216                 i = 1
217
218             if current_hand.__size() == LAST_CARD:
219                 other_rank.append(current_hand.__get_ith_card
(FIRST_CARD).get_rank())
220
221                 current_hand.__remove_ith_card(FIRST_CARD)
222
223         if total_pair == 2:
224             pair_rank.sort(reverse = True)
225             other_rank.sort(reverse = True)
226
227             pair_rank.extend(other_rank)
228
229             self.__rank_list = pair_rank
230
231         return True
232
233     return False
```



```

234
235     def __is_pair(self):
236         """
237         Checks whether the hand is a pair
238
239         :return: True if the hand has a pair of the same rank
240         """
241         current_hand = self.__copy()
242
243         i = 1
244         total_pair = 0
245         pair_rank = []
246         other_rank = []
247
248         while current_hand.__size() > i:
249             if current_hand.__get_ith_card(FIRST_CARD).
get_rank() == current_hand.__get_ith_card(i).get_rank():
250                 total_pair += 1
251
252                 pair_rank.append(current_hand.__get_ith_card(
FIRST_CARD).get_rank())
253
254                 current_hand.__remove_ith_card(i)
255                 current_hand.__remove_ith_card(FIRST_CARD)
256
257                 i = 1
258
259             else:
260                 i += 1
261
262                 if current_hand.__size() == i:
263                     other_rank.append(current_hand.__get_ith_card
(FIRST_CARD).get_rank())
264
265                     current_hand.__remove_ith_card(FIRST_CARD)
266
267                     i = 1
268
269                 if current_hand.__size() == LAST_CARD:
270                     other_rank.append(current_hand.__get_ith_card
(FIRST_CARD).get_rank())

```

```
271
272         current_hand.__remove_ith_card(FIRST_CARD)
273
274     if total_pair == 1:
275         pair_rank.sort(reverse = True)
276         other_rank.sort(reverse = True)
277
278         pair_rank.extend(other_rank)
279
280         self.__rank_list = pair_rank
281
282         return True
283
284     return False
285
286 def get_result(self, other_hand):
287     """
288     Gets the result according to the given value
289
290     :return: a string according to the given value
291     """
292     result = self.compare_to(other_hand)
293
294     if result > 0:
295         return "my hand"
296
297     elif result < 0:
298         return "other hand"
299
300     else:
301         return "tie"
302
303 def __confirm_result():
304     deck = Deck()
305     myhand = PokerHand(deck.list_o_cards())
306     other_hand = PokerHand(deck.list_o_cards())
307     result = myhand.get_result(other_hand)
308
309     print("my hand: ", myhand)
310     print("other hand: ", other_hand)
311     print("result: ", result, " won.")
```

312

313 **if** __name__ == "__main__":

314 __confirm_result()

```
1 from testing import *
2 from poker_hand import *
3
4 def __test_poker_hand():
5     start_tests("Starts testing poker_hand module")
6     __test_compare_to()
7     finish_tests()
8
9 def __test_compare_to():
10     __flush_v_two_pair()
11     __flush_v_flush_high_card1()
12     __flush_v_flush_high_card2()
13     __flush_v_flush_high_card3()
14     __flush_v_flush_high_card4()
15     __flush_v_flush_high_card5()
16     __flush_v_flush_tie()
17
18     __two_pair_v_pair()
19     __two_pair_v_two_pair_high_card1()
20     __two_pair_v_two_pair_high_card2()
21     __two_pair_v_two_pair_high_card3()
22     __two_pair_v_two_pair_tie()
23
24     __pair_v_high_card()
25     __pair_v_pair_high_card1()
26     __pair_v_pair_high_card2()
27     __pair_v_pair_high_card3()
28     __pair_v_pair_high_card4()
29     __pair_v_pair_tie()
30
31     __high_card_v_high_card1()
32     __high_card_v_high_card2()
33     __high_card_v_high_card3()
34     __high_card_v_high_card4()
35     __high_card_v_high_card5()
36     __high_card_v_high_card_tie()
37
38 def __flush_v_two_pair():
39     hand = PokerHand([Card(14, "H"), Card(14, "H"), Card(8, "H"), Card(7, "H"), Card(10, "H")])
40     other_hand = PokerHand([Card(14, "H"), Card(14, "D"), Card
```

```
40 (8, "S"), Card(8, "C"), Card(9, "H"))]
41     msg = "Starts testing flush vs two pair"
42     expected = 1
43     actual = hand.compare_to(other_hand)
44
45     assert_equals(msg, expected, actual)
46
47 def __flush_v_flush_high_card1():
48     hand = PokerHand([Card(14, "H"), Card(13, "H"), Card(12, "
H"), Card(11, "H"), Card(10, "H")])
49     other_hand = PokerHand([Card(13, "H"), Card(12, "H"), Card
(11, "H"), Card(10, "H"), Card(9, "H")])
50     msg = "Starts testing flush vs flush (first high card)"
51     expected = 1
52     actual = hand.compare_to(other_hand)
53
54     assert_equals(msg, expected, actual)
55
56 def __flush_v_flush_high_card2():
57     hand = PokerHand([Card(14, "H"), Card(13, "H"), Card(12, "
H"), Card(11, "H"), Card(10, "H")])
58     other_hand = PokerHand([Card(14, "H"), Card(12, "H"), Card
(11, "H"), Card(10, "H"), Card(9, "H")])
59     msg = "Starts testing flush vs flush (second high card)"
60     expected = 1
61     actual = hand.compare_to(other_hand)
62
63     assert_equals(msg, expected, actual)
64
65 def __flush_v_flush_high_card3():
66     hand = PokerHand([Card(14, "H"), Card(13, "H"), Card(12, "
H"), Card(11, "H"), Card(10, "H")])
67     other_hand = PokerHand([Card(14, "H"), Card(13, "H"), Card
(11, "H"), Card(10, "H"), Card(9, "H")])
68     msg = "Starts testing flush vs flush (third high card)"
69     expected = 1
70     actual = hand.compare_to(other_hand)
71
72     assert_equals(msg, expected, actual)
73
74 def __flush_v_flush_high_card4():
```

```
75     hand = PokerHand([Card(14, "H"), Card(13, "H"), Card(12,
    "H"), Card(11, "H"), Card(10, "H")])
76     other_hand = PokerHand([Card(14, "H"), Card(13, "H"),
    Card(12, "H"), Card(10, "H"), Card(9, "H")])
77     msg = "Starts testing flush vs flush (fourth high card)"
78     expected = 1
79     actual = hand.compare_to(other_hand)
80
81     assert_equals(msg, expected, actual)
82
83 def __flush_v_flush_high_card5():
84     hand = PokerHand([Card(14, "H"), Card(13, "H"), Card(12,
    "H"), Card(11, "H"), Card(10, "H")])
85     other_hand = PokerHand([Card(14, "H"), Card(13, "H"),
    Card(12, "H"), Card(11, "H"), Card(9, "H")])
86     msg = "Starts testing flush vs flush (fifth high card)"
87     expected = 1
88     actual = hand.compare_to(other_hand)
89
90     assert_equals(msg, expected, actual)
91
92 def __flush_v_flush_tie():
93     hand = PokerHand([Card(14, "H"), Card(13, "H"), Card(12,
    "H"), Card(11, "H"), Card(10, "H")])
94     other_hand = PokerHand([Card(14, "H"), Card(13, "H"),
    Card(12, "H"), Card(11, "H"), Card(10, "H")])
95     msg = "Starts testing flush vs flush (tie)"
96     expected = 0
97     actual = hand.compare_to(other_hand)
98
99     assert_equals(msg, expected, actual)
100
101
102 def __two_pair_v_pair():
103     hand = PokerHand([Card(14, "H"), Card(14, "D"), Card(8, "
    S"), Card(8, "C"), Card(10, "H")])
104     other_hand = PokerHand([Card(14, "H"), Card(14, "D"),
    Card(8, "S"), Card(9, "C"), Card(10, "H")])
105     msg = "Starts testing two pair vs pair"
106     expected = 1
107     actual = hand.compare_to(other_hand)
```

```
108
109     assert_equals(msg, expected, actual)
110
111
112 def __two_pair_v_two_pair_high_card1():
113     hand = PokerHand([Card(14, "H"), Card(14, "H"), Card(8, "H"), Card(8, "H"), Card(9, "H")])
114     other_hand = PokerHand([Card(13, "H"), Card(13, "H"), Card(11, "H"), Card(11, "H"), Card(9, "H")])
115     msg = "Starts testing two pair vs two pair (first high pair card)"
116     expected = 1
117     actual = hand.compare_to(other_hand)
118
119     assert_equals(msg, expected, actual)
120
121
122 def __two_pair_v_two_pair_high_card2():
123     hand = PokerHand([Card(14, "H"), Card(14, "H"), Card(8, "H"), Card(8, "H"), Card(9, "H")])
124     other_hand = PokerHand([Card(14, "H"), Card(14, "H"), Card(7, "H"), Card(7, "H"), Card(9, "H")])
125     msg = "Starts testing two pair vs two pair (second high pair card)"
126     expected = 1
127     actual = hand.compare_to(other_hand)
128
129     assert_equals(msg, expected, actual)
130
131
132 def __two_pair_v_two_pair_high_card3():
133     hand = PokerHand([Card(14, "H"), Card(14, "H"), Card(8, "H"), Card(8, "H"), Card(10, "H")])
134     other_hand = PokerHand([Card(14, "H"), Card(14, "H"), Card(8, "H"), Card(8, "H"), Card(9, "H")])
135     msg = "Starts testing two pair vs two pair (third high card)"
136     expected = 1
137     actual = hand.compare_to(other_hand)
138
139     assert_equals(msg, expected, actual)
```

```
140
141
142 def __two_pair_v_two_pair_tie():
143     hand = PokerHand([Card(14, "H"), Card(13, "H"), Card(12,
144         "H"), Card(11, "H"), Card(10, "H")])
145     other_hand = PokerHand([Card(14, "H"), Card(13, "H"),
146         Card(12, "H"), Card(11, "H"), Card(10, "H")])
147     msg = "Starts testing two pair vs two pair (tie)"
148     expected = 0
149     actual = hand.compare_to(other_hand)
150
151     assert_equals(msg, expected, actual)
152
153 def __pair_v_high_card():
154     hand = PokerHand([Card(14, "H"), Card(14, "D"), Card(8, "
155         S"), Card(9, "C"), Card(10, "H")])
156     other_hand = PokerHand([Card(14, "H"), Card(13, "D"),
157         Card(8, "S"), Card(9, "C"), Card(10, "H")])
158     msg = "Starts testing pair vs high card"
159     expected = 1
160     actual = hand.compare_to(other_hand)
161
162     assert_equals(msg, expected, actual)
163
164 def __pair_v_pair_high_card1():
165     hand = PokerHand([Card(14, "H"), Card(14, "D"), Card(8, "
166         S"), Card(9, "C"), Card(10, "H")])
167     other_hand = PokerHand([Card(13, "H"), Card(13, "D"),
168         Card(8, "S"), Card(9, "C"), Card(10, "H")])
169     msg = "Starts testing pair vs pair (first high pair card
170         )"
171     expected = 1
172     actual = hand.compare_to(other_hand)
173
174     assert_equals(msg, expected, actual)
175
176 def __pair_v_pair_high_card2():
177     hand = PokerHand([Card(14, "H"), Card(14, "D"), Card(8, "
178         S"), Card(9, "C"), Card(11, "H")])
179     other_hand = PokerHand([Card(14, "H"), Card(14, "D"),
180         Card(8, "S"), Card(9, "C"), Card(10, "H")])
```



```
172     msg = "Starts testing pair vs pair (second high card)"
173     expected = 1
174     actual = hand.compare_to(other_hand)
175
176     assert_equals(msg, expected, actual)
177
178 def __pair_v_pair_high_card3():
179     hand = PokerHand([Card(14, "H"), Card(14, "D"), Card(8, "
180         S"), Card(10, "C"), Card(11, "H")])
181     other_hand = PokerHand([Card(14, "H"), Card(14, "D"),
182         Card(8, "S"), Card(9, "C"), Card(11, "H")])
183     msg = "Starts testing pair vs pair (third high card)"
184     expected = 1
185     actual = hand.compare_to(other_hand)
186
187     assert_equals(msg, expected, actual)
188
189 def __pair_v_pair_high_card4():
190     hand = PokerHand([Card(14, "H"), Card(14, "D"), Card(9, "
191         S"), Card(10, "C"), Card(11, "H")])
192     other_hand = PokerHand([Card(14, "H"), Card(14, "D"),
193         Card(8, "S"), Card(10, "C"), Card(11, "H")])
194     msg = "Starts testing pair vs pair (fourth high card)"
195     expected = 1
196     actual = hand.compare_to(other_hand)
197
198     assert_equals(msg, expected, actual)
199
200 def __pair_v_pair_tie():
201     hand = PokerHand([Card(14, "H"), Card(14, "D"), Card(9, "
202         S"), Card(10, "C"), Card(11, "H")])
203     other_hand = PokerHand([Card(14, "H"), Card(14, "D"),
204         Card(9, "S"), Card(10, "C"), Card(11, "H")])
205     msg = "Starts testing pair vs pair (tie)"
206     expected = 0
207     actual = hand.compare_to(other_hand)
208
209     assert_equals(msg, expected, actual)
210
211 def __high_card_v_high_card1():
```

```
207     hand = PokerHand([Card(14, "H"), Card(13, "D"), Card(12,
    "S"), Card(11, "C"), Card(10, "H")])
208     other_hand = PokerHand([Card(13, "H"), Card(12, "D"),
    Card(11, "S"), Card(10, "C"), Card(9, "H")])
209     msg = "Starts testing high card vs high card (first high
    card)"
210     expected = 1
211     actual = hand.compare_to(other_hand)
212
213     assert_equals(msg, expected, actual)
214
215 def __high_card_v_high_card2():
216     hand = PokerHand([Card(14, "H"), Card(13, "D"), Card(12,
    "S"), Card(11, "C"), Card(10, "H")])
217     other_hand = PokerHand([Card(14, "H"), Card(12, "D"),
    Card(11, "S"), Card(10, "C"), Card(9, "H")])
218     msg = "Starts testing high card vs high card (second high
    card)"
219     expected = 1
220     actual = hand.compare_to(other_hand)
221
222     assert_equals(msg, expected, actual)
223
224 def __high_card_v_high_card3():
225     hand = PokerHand([Card(14, "H"), Card(13, "D"), Card(12,
    "S"), Card(11, "C"), Card(10, "H")])
226     other_hand = PokerHand([Card(14, "H"), Card(13, "D"),
    Card(11, "S"), Card(10, "C"), Card(9, "H")])
227     msg = "Starts testing high card vs high card (third high
    card)"
228     expected = 1
229     actual = hand.compare_to(other_hand)
230
231     assert_equals(msg, expected, actual)
232
233 def __high_card_v_high_card4():
234     hand = PokerHand([Card(14, "H"), Card(13, "D"), Card(12,
    "S"), Card(11, "C"), Card(10, "H")])
235     other_hand = PokerHand([Card(14, "H"), Card(13, "D"),
    Card(12, "S"), Card(10, "C"), Card(9, "H")])
236     msg = "Starts testing high card vs high card (fourth high
```

```
236     card)"
237     expected = 1
238     actual = hand.compare_to(other_hand)
239
240     assert_equals(msg, expected, actual)
241
242 def __high_card_v_high_card5():
243     hand = PokerHand([Card(14, "H"), Card(13, "D"), Card(12,
244         "S"), Card(11, "C"), Card(10, "H")])
245     other_hand = PokerHand([Card(14, "H"), Card(13, "D"),
246         Card(12, "S"), Card(11, "C"), Card(9, "H")])
247     msg = "Starts testing high card vs high card (fifth high
248         card)"
249     expected = 1
250     actual = hand.compare_to(other_hand)
251
252     assert_equals(msg, expected, actual)
253
254 def __high_card_v_high_card_tie():
255     hand = PokerHand([Card(14, "H"), Card(13, "D"), Card(12,
256         "S"), Card(11, "C"), Card(10, "H")])
257     other_hand = PokerHand([Card(14, "H"), Card(13, "D"),
258         Card(12, "S"), Card(11, "C"), Card(10, "H")])
259     msg = "Starts testing high card vs high card (tie)"
260     expected = 0
261     actual = hand.compare_to(other_hand)
262
263     assert_equals(msg, expected, actual)
264
265 if __name__ == "__main__":
266     __test_poker_hand()
```