```
 1 class Card:
 2     """
 3     Class for the Card object
 4 
 5     """
 6 
 7     def __init__(self, rank, suit):
 8         """
 9         initializes Card object
10 
11         :param self: instance of Card
12         :param rank: cards number or face value
13         :param suit: suit value of Card (heart,diamond,
   club, or spade)
14         :return:
15         """
16         self.__card = {'rank': rank, 'suit': suit}
17 
18 
19 
20     def get_rank(self):
21         """
22         getter method for this cards rank
23         :return:  rank value
24         """
25 
26         return self.__card['rank']
27 
28     def get_suit(self):
29         """
30         getter method for this cards suit
31         :return: suit value
32         """
33         return self.__card['suit']
34 
35     def __str__(self):
36 
37         rank = str(self.get_rank())
38 
39         if rank == "11":
40             rank = "Jack"
41         if rank == "12":
42             rank = "Queen"
43         if rank == "13":
44             rank = "King"
45         if rank == "14":
```

```
46              rank = "Ace"
47
48          suit = self.get_suit()
49
50          if suit == "S":
51              suit = "Spades"
52          if suit == "D":
53              suit = "Diamonds"
54          if suit == "C":
55              suit = "Clubs"
56          if suit == "H":
57              suit = "Hears"
58
59
60
61
62          return "[ " + rank + " of " + suit + " ]"
63
```

```
 1 import random
 2
 3 from card import Card
 4
 5
 6 class deck:
 7     """
 8     model of a deck
 9     """
10
11     def __init__(self):
12         """
13         initialize deck object
14         """
15         self.__cardList = []
16         deck.generate( self )
17         deck.shuffle( self )
18
19     def get_card_list(self):
20         """
21         getter method for cardList
22
23         :return: cardList
24         """
25         return self.__cardList
26
27     def size(self):
28         """
29         returns size of the deck left
30
31         :return: size
32         """
33
34
35         card_list = self.get_card_list()
36         return len( card_list )
37
38     def generate(self):
39         """
40         generates a deck of 52 cards
41
42         :return: none
43         """
44
45         for i in range( 0, 4 ):
46             for j in range( 2, 15 ):
```

```
47
48                  if i == 0:  # Heart suit
49                      suit = "H"
50
51                  if i == 1:  # Diamond suit
52                      suit = "D"
53
54                  if i == 2:  # spade suit
55                      suit = "S"
56
57                  if i == 3:  # club suit
58                      suit = "C"
59
60                  self.__cardList.append( Card( j, suit ) )
61
62      def shuffle(self):
63          for i in range( 3 ):
64              random.shuffle( self.__cardList )
65
66      def deal(self):
67          """
68          draws a Card from the deck
69          :return: Card
70          """
71
72          if self.__cardList.__len__() != 0:
73              return self.__cardList.pop()
74
75          else:
76              return None
77
78      def __str__(self):
79          """
80          prints deck of cards
81          :return: none
82          """
83          string = "Hand \n"
84          for card in self.get_card_list():
85              string += card
86              string += "\n"
87
88          return string
89
```

```
 1
 2 class PokerHand:
 3     """
 4     models a poker hand
 5     """
 6
 7     def __init__(self, hand_list=None):
 8         if hand_list is None:
 9             hand_list = []
10         self.__hand = hand_list
11
12     def get_hand(self):
13         """
14         getter method for hand(list of cards)
15
16         :return: hand
17         """
18         return self.__hand
19
20     def add_card(self, card):
21         """
22         append a card to a hand
23
24         :param card: card being added
25         :return: None
26         """
27         self.__hand.append( card )
28
29     def get_ith_card(self, index):
30         if 0 <= index < len( self.get_hand() ):
31             return self.get_hand()[index]
32         else:
33             return None
34
35     def deal_poker_hand(self, deck):
36         """
37         this function adds 5 cards from the deck to the
   hand
38
39         :param deck: deck that cards are being drawn from
40         :return:
41         """
42
43         for i in range( 5 ):
44             self.__hand.append( deck.deal() )
45
```

```
46      def what_is_it(self):
47          """
48          evaluates the hand
49
50          :return: index[0] - hand type      index[1] - pair
    values                                  index[2]
51                   4 for flush                    pairs = [] (
    empty list) - if there are no pairs    list of
52                   3 for 2pair                    pairs = [rank1
    ,] - if 1 pair                          highcard values
53                   2 for pair                     pairs = [rank1
    , rank2] - if 2 pairs                  (all cards not in a
    pair)
54                   1 for highcard
55          """
56
57          pairs = []
58          highcards = []
59
60          ranks = sorted( [card.get_rank() for card in self.
    get_hand()] )
61          suits = [card.get_suit() for card in self.get_hand
    ()]
62
63          for r in set( ranks ):
64              if ranks.count( r ) == 4:
65                  pairs.append( r )
66                  pairs.append( r )
67
68              if ranks.count( r ) == 3:
69                  pairs.append( r )
70
71              if ranks.count( r ) == 2:
72                  pairs.append( r )
73
74              else:
75                  highcards.append( r )
76
77          if all( s == suits[0] for s in suits ):
78              return 4, pairs, highcards
79
80          if len( pairs ) == 2:
81              return 3, pairs, highcards
82
83          if len( pairs ) == 1:
84              return 2, pairs, highcards
```

```
 85         else:
 86 
 87             return 1, pairs, highcards
 88 
 89     def compare_to(self, other_hand):
 90         """
 91         Determines which of two poker hands is worth more. Returns an int
 92         which is either positive, negative, or zero depending on the comparison.
 93         :param self: The first hand to compare
 94         :param other_hand: The second hand to compare
 95         :return: a negative number if self is worth LESS than other_hand,
 96         zero if they are worth the SAME (a tie), and a positive number if
 97         self is worth MORE than other_hand
 98 
 99          """
100 
101         this_hand_type = self.what_is_it()[0]
102         other_hand_type = other_hand.what_is_it()[0]
103 
104         this_hand_pairs = sorted( self.what_is_it()[1], reverse=True )
105         other_hand_pairs = sorted( other_hand.what_is_it()[1], reverse=True )
106 
107         this_hand_highcards = sorted( self.what_is_it()[2], reverse=True )
108         other_hand_highcards = sorted( other_hand.what_is_it()[2], reverse=True )
109 
110         if this_hand_type > other_hand_type:  # if this hand is a higher type
111             return 1
112         if this_hand_type < other_hand_type:  # if this hand is a lower type
113             return -1
114         if this_hand_type == other_hand_type:  # if same hand type
115             if len( this_hand_pairs ) != 0:  # makes sure there are pairs
116                 if self.compare_to_helper( this_hand_pairs, other_hand_pairs ) == 0:  # if pairs tie
117                     return self.compare_to_helper(
```

```
117 this_hand_highcards,
118 
    other_hand_highcards )  # compare highcards result
119                 else:
120                     return self.compare_to_helper(
    this_hand_pairs,
121 
    other_hand_pairs )  # compare pairs result
122             else:  # if no pairs then just compare the
    highcards
123                 return self.compare_to_helper(
    this_hand_highcards,
124 
    other_hand_highcards )  # compare highcards result
125 
126     def compare_to_helper(self, this_hand_list,
    other_hand_list):
127         """
128 
129         :param this_hand_list: sorted list of ranks from '
    this' hand
130         :param other_hand_list: sorted list of ranks from
    other hand
131         :return:  1 if this_hand_list contains first
    instance of a greater rank,
132                   -1 if other_hand_list contains first
    instance of a greater rank,
133                   and 0 if all ranks are the same
134         """
135         for rank1, rank2 in zip( this_hand_list,
    other_hand_list ):
136             if rank1 > rank2:
137                 return 1
138             if rank1 < rank2:
139                 return -1
140             if rank1 == rank2:
141                 continue
142         return 0
143 
144     def __str__(self):
145         """
146         prints all cards in hand
147         :return: none
148         """
149         string = ""
150         for card in self.get_hand():
```

```
151             string += str( card ) + "\n"
152
153         return string
154
```

```python
from deck import deck
from hand import PokerHand

"""
By: Ian Sulley

Honor Code:
I affirm that I have carried out the attached academic endeavors with full academic honesty,
in accordance with the Union College Honor Code and the course syllabus.

"""


def generate_hand(deck):
    """
    generates a hand from the given deck
    :param deck: deck used to form hand
    :return: hand
    """
    new_hand = PokerHand()
    if (len( deck.get_card_list() ) >= 5):
        new_hand.deal_poker_hand( deck )
        return new_hand
    if (len( deck.get_card_list() ) < 5):  # if the cards in the deck drops below 5

        new_deck = generate_deck()  # generate a new deck

        new_deck.extend(deck.get_card_list())
        deck = new_deck
        new_hand.deal_poker_hand( deck )

        return None


def generate_deck():
    """
    creates a new deck
    :return: deck
    """

    new_deck = deck()
    return new_deck

```

```
44
45 def main():
46
47     my_deck = generate_deck() #make a deck of cards
48     score_count = 0 #keep track of score
49
50     while True:
51
52         if len( my_deck.get_card_list() ) < 10:
53             break
54
55         hand_1 = generate_hand( my_deck )
56         hand_2 = generate_hand( my_deck )
57
58         evaluated_winner = hand_1.compare_to( hand_2 )
59
60         print( "Hand 1: \n" )
61         print( hand_1 )
62         print( "\n" )
63
64         print( "Hand 2: \n" )
65         print( hand_2 )
66         print( "\n" )
67
68         user_winner = int( input( "Who is the winner?(1, 2
   , or 0 if tie): " ) )
69
70         while user_winner != 1 and user_winner != 2 and
   user_winner != 0:
71             user_winner = int( input( "Sorry, your answer
   was invalid, please enter 1, 2, or 0.: " ) )
72
73         print( user_winner )
74         if user_winner == 2:
75             user_winner = -1
76         if user_winner == evaluated_winner:
77             score_count += 1
78             print( "Correct! 1 point awarded." )
79         else:
80             print( "Sorry, Wrong Answer." )
81             print( evaluated_winner )
82             game_over = True
83
84     print( "Congrats, You've made it through the entire
   deck!" )
85     print( "Your score: " + str( score_count ) )
```

```
 86      print( "Thanks for playing!" )
 87
 88
 89 if __name__ == '__main__':
 90      main()
 91
```

```
 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
    tests using
61      assert_equals.
62      :param header: A header to print at the beginning
63      of the tests.
64      """
65      global num_pass, num_fail
66      print(header)
67      for i in range(0,len(header)):
68          print("=",end="")
69      print("")
70      num_pass = 0
71      num_fail = 0
72
73  def finish_tests():
74      """
75      Prints summary statistics after the tests are complete.
76      """
77      print("Passed %d/%d" % (num_pass, num_pass+num_fail))
78      print("Failed %d/%d" % (num_fail, num_pass+num_fail))
79      print()
80
```

```
 1 from testing import *
 2 from hand import *
 3 from card import *
 4
 5
 6 ######TESTS###########
 7 def test_all():
 8     start_tests( "Starting All Tests" )
 9     test_all_flushes()
10     test_all_two_pair()
11     test_all_pair()
12     test_all_highcard()
13
14
15
16 #####FLUSH TESTS#####
17
18 def test_all_flushes():
19     start_tests( "Testing Flushes" )
20     compare_flushes1()
21     compare_flushes2()
22     compare_flushes_tie()
23     compare_flush_2pair()
24     compare_flush_pair()
25     compare_flush_hi()
26     finish_tests()
27
28
29 # Flush1 vs Flush2 (Flush 1 wins highcard is greater)
30 def compare_flushes1():
31     hand1 = PokerHand( [Card( 13, "S" ), Card( 12, "S" ),
   Card( 9, "S" ), Card( 7, "S" ), Card( 3, "S" )] )
32     hand2 = PokerHand( [Card( 4, "C" ), Card( 2, "C" ),
   Card( 7, "C" ), Card( 5, "C" ), Card( 10, "C" )] )
33
34     expected_answer = 1
35     actual_answer = hand1.compare_to( hand2 )
36     assert_equals( "Testing Flush1 vs Flush2 (Flush 1 wins
   ; Highcard is greater)",
37                    expected_answer,
38                    actual_answer )
39
40
41 # Flush1 vs Flush2 (Flush 2 wins highcard is greater)
42 def compare_flushes2():
43     hand1 = PokerHand( [Card( 4, "C" ), Card( 2, "C" ),
```

```
43 Card( 7, "C" ), Card( 5, "C" ), Card( 10, "C" )] )
44     hand2 = PokerHand( [Card( 13, "S" ), Card( 12, "S" ),
   Card( 9, "S" ), Card( 7, "S" ), Card( 3, "S" )] )
45
46     expected_answer = -1
47     actual_answer = hand1.compare_to( hand2 )
48     assert_equals( "Testing Flush1 vs Flush2 (Flush 2 wins
   highcard is greater)",
49                    expected_answer,
50                    actual_answer )
51
52
53 # Flush1 vs Flush2 (Tie)
54 def compare_flushes_tie():
55     hand1 = PokerHand( [Card( 4, "C" ), Card( 2, "C" ),
   Card( 7, "C" ), Card( 5, "C" ), Card( 10, "C" )] )
56     hand2 = PokerHand( [Card( 7, "S" ), Card( 5, "S" ),
   Card( 4, "S" ), Card( 2, "S" ), Card( 10, "S" )] )
57
58     expected_answer = 0
59     actual_answer = hand1.compare_to( hand2 )
60     assert_equals( "Testing Flush1 vs Flush2 Tie",
61                    expected_answer,
62                    actual_answer )
63
64
65 # Flush vs 2 pair
66 def compare_flush_2pair():
67     hand1 = PokerHand( [Card( 7, "S" ), Card( 5, "S" ),
   Card( 11, "S" ), Card( 2, "S" ), Card( 10, "S" )] )
68     hand2 = PokerHand( [Card( 4, "H" ), Card( 4, "D" ),
   Card( 8, "S" ), Card( 8, "C" ), Card( 6, "D" )] )
69
70     expected_answer = 1
71     actual_answer = hand1.compare_to( hand2 )
72     assert_equals( "Testing Flush1 vs 2pair (Flush 1 wins)"
   ,
73                    expected_answer,
74                    actual_answer )
75
76
77 # Flush vs pair
78 def compare_flush_pair():
79     hand1 = PokerHand( [Card( 7, "S" ), Card( 5, "S" ),
   Card( 4, "S" ), Card( 2, "S" ), Card( 10, "S" )] )
80     hand2 = PokerHand( [Card( 10, "H" ), Card( 4, "D" ),
```

```
 80 Card( 9, "S" ), Card( 10, "C" ), Card( 6, "D" )] )
 81
 82     expected_answer = 1
 83     actual_answer = hand1.compare_to( hand2 )
 84     assert_equals( "Testing Flush1 vs pair (Flush 1 wins)"
    ,
 85                    expected_answer,
 86                    actual_answer )
 87
 88
 89 # Flush vs high Card
 90 def compare_flush_hi():
 91     hand1 = PokerHand( [Card( 7, "S" ), Card( 5, "S" ),
    Card( 4, "S" ), Card( 2, "S" ), Card( 10, "S" )] )
 92     hand2 = PokerHand( [Card( 3, "H" ), Card( 4, "D" ),
    Card( 10, "S" ), Card( 8, "C" ), Card( 6, "D" )] )
 93
 94     expected_answer = 1
 95     actual_answer = hand1.compare_to( hand2 )
 96     assert_equals( "Testing Flush1 vs highcard (Flush 1
    wins)",
 97                    expected_answer,
 98                    actual_answer )
 99
100
101 #####2 PAIR TESTS#####
102
103 def test_all_two_pair():
104     start_tests( "Testing 2pair" )
105     compare_2pair_flush()
106     compare_2pair_2pair_1()
107     compare_2pair_2pair_2()
108     compare_2pair_2pair_3()
109     compare_2pair_2pair_4()
110     compare_2pair_2pair_5()
111     compare_2pair_2pair_tie()
112     compare_2pair_pair()
113     compare_2pair_hi()
114     finish_tests()
115
116
117 # 2pair vs Flush
118 def compare_2pair_flush():
119     hand1 = PokerHand( [Card( 4, "H" ), Card( 4, "D" ),
    Card( 8, "S" ), Card( 8, "C" ), Card( 6, "D" )] )
120     hand2 = PokerHand( [Card( 7, "S" ), Card( 5, "S" ),
```

```
120 Card( 11, "S" ), Card( 2, "S" ), Card( 10, "S" )] )
121
122     expected_answer = -1
123     actual_answer = hand1.compare_to( hand2 )
124     assert_equals( "Testing 2pair vs Flush",
125                    expected_answer,
126                    actual_answer )
127
128
129 # 2pair1 vs 2pair2 (2pair1 wins higher of pair values is
    greater)
130 def compare_2pair_2pair_1():
131     hand1 = PokerHand( [Card( 4, "H" ), Card( 6, "D" ),
    Card( 10, "S" ), Card( 10, "C" ), Card( 4, "D" )] )
132     hand2 = PokerHand( [Card( 4, "H" ), Card( 4, "D" ),
    Card( 8, "S" ), Card( 8, "C" ), Card( 6, "D" )] )
133
134     expected_answer = 1
135     actual_answer = hand1.compare_to( hand2 )
136     assert_equals( "Testing 2pair1 vs 2pair2 (2pair1 wins
    higher of pair values is greater)",
137                    expected_answer,
138                    actual_answer )
139
140
141 # 2pair1 vs 2pair2 (2pair2 wins higher of pair values is
    greater)
142 def compare_2pair_2pair_2():
143     hand1 = PokerHand( [Card( 4, "H" ), Card( 4, "D" ),
    Card( 8, "S" ), Card( 8, "C" ), Card( 6, "D" )] )
144     hand2 = PokerHand( [Card( 14, "H" ), Card( 14, "D" ),
    Card( 8, "S" ), Card( 8, "C" ), Card( 6, "D" )] )
145
146     expected_answer = -1
147     actual_answer = hand1.compare_to( hand2 )
148     assert_equals( "Testing 2pair1 vs 2pair2 (2pair2 wins
    higher of pair values is greater)",
149                    expected_answer,
150                    actual_answer )
151
152
153 # 2pair1 vs 2pair2 (2pair1 wins lower of pair values is
    greater)
154 def compare_2pair_2pair_3():
155     hand1 = PokerHand( [Card( 4, "H" ), Card( 4, "D" ),
    Card( 8, "S" ), Card( 8, "C" ), Card( 6, "D" )] )
```

```
156     hand2 = PokerHand( [Card( 3, "H" ), Card( 4, "D" ),
    Card( 8, "S" ), Card( 8, "C" ), Card( 3, "D" )] )
157
158     expected_answer = 1
159     actual_answer = hand1.compare_to( hand2 )
160     assert_equals( "2pair1 vs 2pair2 (2pair1 wins lower of
     pair values is greater)",
161                    expected_answer,
162                    actual_answer )
163
164
165 # 2pair1 vs 2pair2 (2pair2 wins lower of pair values is
    greater)
166 def compare_2pair_2pair_4():
167     hand1 = PokerHand( [Card( 3, "H" ), Card( 4, "D" ),
    Card( 8, "S" ), Card( 8, "C" ), Card( 3, "D" )] )
168     hand2 = PokerHand( [Card( 4, "H" ), Card( 4, "D" ),
    Card( 8, "S" ), Card( 8, "C" ), Card( 6, "D" )] )
169
170     expected_answer = -1
171     actual_answer = hand1.compare_to( hand2 )
172     assert_equals( "Testing 2pair1 vs 2pair2 (2pair2 wins
    lower of pair values is greater)",
173                    expected_answer,
174                    actual_answer )
175
176
177 # 2pair1 vs 2pair2 (2pair1 wins, both pairs same but
    highcard wins)
178 def compare_2pair_2pair_5():
179     hand1 = PokerHand( [Card( 4, "H" ), Card( 4, "D" ),
    Card( 8, "S" ), Card( 8, "C" ), Card( 6, "D" )] )
180     hand2 = PokerHand( [Card( 4, "H" ), Card( 4, "D" ),
    Card( 8, "S" ), Card( 8, "C" ), Card( 2, "D" )] )
181
182     expected_answer = 1
183     actual_answer = hand1.compare_to( hand2 )
184     assert_equals( "Testing 2pair1 vs 2pair2 (2pair1 wins
    , both pairs same but highcard wins)",
185                    expected_answer,
186                    actual_answer )
187
188
189 # 2pair1 vs 2pair2 (tie)
190 def compare_2pair_2pair_tie():
191     hand1 = PokerHand( [Card( 4, "H" ), Card( 4, "D" ),
```

```
191 Card( 8, "S" ), Card( 8, "C" ), Card( 6, "D" )] )
192     hand2 = PokerHand( [Card( 4, "H" ), Card( 4, "D" ),
    Card( 8, "S" ), Card( 8, "C" ), Card( 6, "D" )] )
193
194     expected_answer = 0
195     actual_answer = hand1.compare_to( hand2 )
196     assert_equals( "Testing 2pair1 vs 2pair2 (tie)",
197                    expected_answer,
198                    actual_answer )
199
200
201 # 2pair vs pair
202 def compare_2pair_pair():
203     hand1 = PokerHand( [Card( 4, "H" ), Card( 4, "D" ),
    Card( 8, "S" ), Card( 8, "C" ), Card( 6, "D" )] )
204     hand2 = PokerHand( [Card( 10, "H" ), Card( 4, "D" ),
    Card( 9, "S" ), Card( 10, "C" ), Card( 6, "D" )] )
205
206     expected_answer = 1
207     actual_answer = hand1.compare_to( hand2 )
208     assert_equals( "Testing 2pair vs pair",
209                    expected_answer,
210                    actual_answer )
211
212
213 # 2pair vs highcard
214 def compare_2pair_hi():
215     hand1 = PokerHand( [Card( 4, "H" ), Card( 4, "D" ),
    Card( 8, "S" ), Card( 8, "C" ), Card( 6, "D" )] )
216     hand2 = PokerHand( [Card( 3, "H" ), Card( 4, "D" ),
    Card( 10, "S" ), Card( 8, "C" ), Card( 6, "D" )] )
217
218     expected_answer = 1
219     actual_answer = hand1.compare_to( hand2 )
220     assert_equals( "2pair vs highcard",
221                    expected_answer,
222                    actual_answer )
223
224
225 #####PAIR TESTS#####
226
227 def test_all_pair():
228     start_tests( "Testing pairs" )
229     compare_pair_flush()
230     compare_pair_2pair()
231     compare_pair_pair_1()
```

```
232     compare_pair_pair_2()
233     compare_pair_pair_3()
234     compare_pair_pair_4()
235     compare_pair_hi()
236     compare_pair_pair_tie()
237     finish_tests()
238
239
240 # pair vs flush
241 def compare_pair_flush():
242     hand1 = PokerHand( [Card( 10, "H" ), Card( 4, "D" ),
    Card( 9, "S" ), Card( 10, "C" ), Card( 6, "D" )] )
243     hand2 = PokerHand( [Card( 4, "C" ), Card( 2, "C" ),
    Card( 7, "C" ), Card( 5, "C" ), Card( 10, "C" )] )
244
245     expected_answer = -1
246     actual_answer = hand1.compare_to( hand2 )
247     assert_equals( "Testing pair vs flush",
248                    expected_answer,
249                    actual_answer )
250
251
252 # pair vs 2pair
253 def compare_pair_2pair():
254     hand1 = PokerHand( [Card( 10, "H" ), Card( 4, "D" ),
    Card( 9, "S" ), Card( 10, "C" ), Card( 6, "D" )] )
255     hand2 = PokerHand( [Card( 4, "H" ), Card( 4, "D" ),
    Card( 8, "S" ), Card( 8, "C" ), Card( 6, "D" )] )
256
257     expected_answer = -1
258     actual_answer = hand1.compare_to( hand2 )
259     assert_equals( "Testing pair vs 2pair",
260                    expected_answer,
261                    actual_answer )
262
263
264 # pair1 vs pair2 (pair1 wins; high pair)
265 def compare_pair_pair_1():
266     hand1 = PokerHand( [Card( 10, "H" ), Card( 4, "D" ),
    Card( 9, "S" ), Card( 10, "C" ), Card( 6, "D" )] )
267     hand2 = PokerHand( [Card( 2, "H" ), Card( 4, "D" ),
    Card( 9, "S" ), Card( 9, "C" ), Card( 6, "D" )] )
268
269     expected_answer = 1
270     actual_answer = hand1.compare_to( hand2 )
271     assert_equals( "pair1 vs pair2 (pair1 wins; high pair
```

```
271 )",
272                     expected_answer,
273                     actual_answer )
274
275
276 # pair1 vs pair2 (pair2 wins; high pair)
277 def compare_pair_pair_2():
278     hand1 = PokerHand( [Card( 10, "H" ), Card( 4, "D" ),
    Card( 9, "S" ), Card( 10, "C" ), Card( 6, "D" )] )
279     hand2 = PokerHand( [Card( 12, "H" ), Card( 4, "D" ),
    Card( 9, "S" ), Card( 12, "C" ), Card( 6, "D" )] )
280
281     expected_answer = -1
282     actual_answer = hand1.compare_to( hand2 )
283     assert_equals( "Testing pair1 vs pair2 (pair2 wins;
    high pair)",
284                     expected_answer,
285                     actual_answer )
286
287
288 # pair1 vs pair2 (pair1 wins; highcard)
289 def compare_pair_pair_3():
290     hand1 = PokerHand( [Card( 10, "H" ), Card( 4, "D" ),
    Card( 12, "S" ), Card( 10, "C" ), Card( 6, "D" )] )
291     hand2 = PokerHand( [Card( 10, "H" ), Card( 4, "D" ),
    Card( 9, "S" ), Card( 10, "C" ), Card( 6, "D" )] )
292
293     expected_answer = 1
294     actual_answer = hand1.compare_to( hand2 )
295     assert_equals( "Testing pair1 vs pair2 (pair1 wins;
    highcard)",
296                     expected_answer,
297                     actual_answer )
298
299
300 # pair1 vs pair2 (pair2 wins; highcard)
301 def compare_pair_pair_4():
302     hand1 = PokerHand( [Card( 10, "H" ), Card( 4, "D" ),
    Card( 9, "S" ), Card( 10, "C" ), Card( 6, "D" )] )
303     hand2 = PokerHand( [Card( 10, "H" ), Card( 13, "D" ),
    Card( 9, "S" ), Card( 10, "C" ), Card( 6, "D" )] )
304
305     expected_answer = -1
306     actual_answer = hand1.compare_to( hand2 )
307     assert_equals( "Testing pair1 vs pair2 (pair2 wins;
    highcard",
```

```
308                     expected_answer,
309                     actual_answer )
310
311
312 # pair1 vs pair2 (tie)
313 def compare_pair_pair_tie():
314     hand1 = PokerHand( [Card( 10, "H" ), Card( 4, "D" ),
    Card( 9, "S" ), Card( 10, "C" ), Card( 6, "D" )] )
315     hand2 = PokerHand( [Card( 10, "H" ), Card( 4, "D" ),
    Card( 9, "S" ), Card( 10, "C" ), Card( 6, "D" )] )
316
317     expected_answer = 0
318     actual_answer = hand1.compare_to( hand2 )
319     assert_equals( "Testing pair1 vs pair2 (tie)",
320                     expected_answer,
321                     actual_answer )
322
323
324 # pair vs high Card
325 def compare_pair_hi():
326     hand1 = PokerHand( [Card( 10, "H" ), Card( 4, "D" ),
    Card( 9, "S" ), Card( 10, "C" ), Card( 6, "D" )] )
327     hand2 = PokerHand( [Card( 3, "H" ), Card( 4, "D" ),
    Card( 9, "S" ), Card( 2, "C" ), Card( 6, "D" )] )
328
329     expected_answer = 1
330     actual_answer = hand1.compare_to( hand2 )
331     assert_equals( "Testing pair vs high Card",
332                     expected_answer,
333                     actual_answer )
334
335
336 #####HIGHCARD TESTS#####
337 def test_all_highcard():
338     start_tests( "Testing Highcards" )
339     compare_hi_flush()
340     compare_hi_2pair()
341     compare_hi_pair()
342     compare_hi_hi_1()
343     compare_hi_hi_2()
344     compare_hi_hi_tie()
345     finish_tests()
346
347
348 # highcard vs flush
349 def compare_hi_flush():
```

```
350     hand1 = PokerHand( [Card( 10, "H" ), Card( 4, "D" ),
    Card( 9, "S" ), Card( 2, "C" ), Card( 6, "D" )] )
351     hand2 = PokerHand( [Card( 4, "C" ), Card( 2, "C" ),
    Card( 7, "C" ), Card( 5, "C" ), Card( 10, "C" )] )
352
353     expected_answer = -1
354     actual_answer = hand1.compare_to( hand2 )
355     assert_equals( "Testing Flush1 vs Flush2 (Flush 1 wins
    ; Highcard is greater)",
356                    expected_answer,
357                    actual_answer )
358
359
360 # highcard vs 2pair
361 def compare_hi_2pair():
362     hand1 = PokerHand( [Card( 10, "H" ), Card( 4, "D" ),
    Card( 9, "S" ), Card( 2, "C" ), Card( 6, "D" )] )
363     hand2 = PokerHand( [Card( 4, "H" ), Card( 4, "D" ),
    Card( 8, "S" ), Card( 8, "C" ), Card( 6, "D" )] )
364     expected_answer = -1
365     actual_answer = hand1.compare_to( hand2 )
366     assert_equals( "Testing highcard vs 2pair",
367                    expected_answer,
368                    actual_answer )
369
370
371 # highcard vs pair
372 def compare_hi_pair():
373     hand1 = PokerHand( [Card( 10, "H" ), Card( 4, "D" ),
    Card( 9, "S" ), Card( 2, "C" ), Card( 6, "D" )] )
374     hand2 = PokerHand( [Card( 10, "H" ), Card( 4, "D" ),
    Card( 9, "S" ), Card( 10, "C" ), Card( 6, "D" )] )
375
376     expected_answer = -1
377     actual_answer = hand1.compare_to( hand2 )
378     assert_equals( "Testing highcard vs pair",
379                    expected_answer,
380                    actual_answer )
381
382
383 # highcard1 vs highcard2 (highcard1 wins)
384 def compare_hi_hi_1():
385     hand1 = PokerHand( [Card( 10, "H" ), Card( 4, "D" ),
    Card( 9, "S" ), Card( 12, "C" ), Card( 6, "D" )] )
386     hand2 = PokerHand( [Card( 10, "H" ), Card( 4, "D" ),
    Card( 9, "S" ), Card( 2, "C" ), Card( 6, "D" )] )
```

```
387
388     expected_answer = 1
389     actual_answer = hand1.compare_to( hand2 )
390     assert_equals( "Testing highcard1 vs highcard2 (
    highcard1 wins)",
391                    expected_answer,
392                    actual_answer )
393
394
395 # highcard1 vs highcard2 (highcard2 wins)
396 def compare_hi_hi_2():
397     hand1 = PokerHand( [Card( 10, "H" ), Card( 4, "D" ),
    Card( 9, "S" ), Card( 2, "C" ), Card( 6, "D" )] )
398     hand2 = PokerHand( [Card( 10, "H" ), Card( 4, "D" ),
    Card( 9, "S" ), Card( 12, "C" ), Card( 6, "D" )] )
399
400     expected_answer = -1
401     actual_answer = hand1.compare_to( hand2 )
402     assert_equals( "Testing highcard1 vs highcard2 (
    highcard2 wins)",
403                    expected_answer,
404                    actual_answer )
405
406
407 # highcard1 vs highcard2 (tie)
408 def compare_hi_hi_tie():
409     hand1 = PokerHand( [Card( 10, "H" ), Card( 4, "D" ),
    Card( 9, "S" ), Card( 2, "C" ), Card( 6, "D" )] )
410     hand2 = PokerHand( [Card( 10, "H" ), Card( 4, "D" ),
    Card( 9, "S" ), Card( 2, "C" ), Card( 6, "D" )] )
411
412     expected_answer = 0
413     actual_answer = hand1.compare_to( hand2 )
414     assert_equals( "Testing highcard1 vs highcard2 (tie)",
415                    expected_answer,
416                    actual_answer )
417
418
419
420
421 if __name__ == "__main__":
422     test_all()
423
424
425
426
```

```
427
428
429
430
431 """
432  _____  _____  _____
433 |_   _||  |  ||   __|
434   | |  |     ||   __|
435   |_|  |__|__||_____|
436
437
438  _____  _____  ____
439 |   __||   | ||    \
440 |   __|| | | ||  |  |
441 |_____||_|___||____/
442
443 """
444
```