CMP-5015Y Assignment 2 (Java)

100227789 (jvf17ptu)

Wed, 6 Feb 2019 11:36

PDF prepared using PASS version 1.15 running on Windows 10 10.0 (amd64).

☑ I agree that by submitting a PDF generated by PASS I am confirming that I have checked the PDF and that it correctly represents my submission.

Contents

Card.java

```
1  package cw2;
2
3  import java.io.Serializable;
4  import java.util.Comparator;
5  import java.util.Random;
6  import java.util.ArrayList;
7  import java.util.Collections;
8  import java.util.Iterator;
9
10 public class Card implements Serializable, Comparable<Card> {
11
12     static final long serialVersionUID = 100L;
13
14     Rank rank;
15     Suit suit;
16
17     public Card(Rank r, Suit s) {
18         rank = r;
19         suit = s;
20     }
21
22     //CARD RANKS enum
23     public enum Rank {
24         TWO(2), THREE(3), FOUR(4), FIVE(5), SIX(6), SEVEN(7), EIGHT(8),
25         NINE(9), TEN(10), JACK(10), QUEEN(10), KING(10), ACE(11);
26
27         int value;
28
29         //Rank constructor
30         Rank(int value) {
31             this.value = value;
32         }
33
34         public static Rank[] vals = values();
35
36         //gets the next rank
37         Rank getNext() {
38             return vals[(this.ordinal() + 1) % vals.length];
39
40         }
41
42         //returns the value of the rank
43         public int getValue() {
44             return this.value;
45         }
46
47         public static Rank randomRank() {
48             Random rand1 = new Random();
49             return values()[rand1.nextInt(values().length)];
50         }
51     }
52
53     //CARD SUIT enum
54     public enum Suit {
55         CLUBS(4), DIAMONDS(3), HEARTS(2), SPADES(1);
56
57         final int value;
58
59         //Suit constructor
60         Suit(int value) {
61             this.value = value;
```

```
        }

        //selects random suit for the card
        public static Suit randomSuit() {
            Random rand = new Random();
            return values()[rand.nextInt(values().length)];
        }
    }

    @Override
    public String toString() {
        return "Rank:" + rank.value + "(" + rank + ")" + "  Suit:" + suit;
    }

    //sort cards into ascending order
    @Override
    public int compareTo(Card o) {
        if (this.rank.value > o.rank.value) {
            return 1;
        } else if (this.rank.value < o.rank.value) {
            return -1;
        }
        return 0;
    }

    //returns the max card value
    public static Card max(ArrayList card) {
        Card maxRank = null;

        //sort the list, so the last element will be the one with the highest
            value
        Collections.sort(card);

        Iterator<Card> it = card.iterator();
        while (it.hasNext()) {
            maxRank = it.next();
        }
        return maxRank;
    }

    //Sort in Descending by Rank
    static class CompareDescending implements Comparator<Card> {

        @Override
        public int compare(Card a, Card b) {
            if (a.rank.value < b.rank.value) {
                return 1;
            } else if (a.rank.value > b.rank.value) {
                return -1;
            }
            return 0;
        }
    }

    //Sort Ascending by Rank(value)
    public static class CompareRank implements Comparator<Card> {

        @Override
        public int compare(Card a, Card b) {
            if (a.rank.value > b.rank.value) {
                return 1;
            } else if (a.rank.value < b.rank.value) {
                return -1;
```

```
            }
125             return 0;
        }
127     }

129     //gives a list of card with value higher than the given
        public static ArrayList chooseGreater(ArrayList<Card> cardList, Comparator
            comp, Card card) {
131         ArrayList<Card> list = new ArrayList<>();
            int cardValue = card.rank.value;
133         Collections.sort(cardList, comp);
            for (Card c : cardList) {
135             if (c.rank.value > cardValue) {
                    list.add(c);
137             }
            }
139         return list;
        }
141
        public static void selectTest(Comparator compR, Comparator compD,
            compareCards compL) {
143
            ArrayList<Card> card = new ArrayList<>();
145         ArrayList<Card> card1 = new ArrayList<>();
            ArrayList<Card> card2 = new ArrayList<>();
147         card.add(new Card(Rank.TEN, Suit.DIAMONDS));
            card.add(new Card(Rank.FOUR, Suit.SPADES));
149         card.add(new Card(Rank.TEN, Suit.SPADES));
            card.add(new Card(Rank.TWO, Suit.CLUBS));
151         card.add(new Card(Rank.SIX, Suit.HEARTS));
            card.add(new Card(Rank.THREE, Suit.CLUBS));
153         card.add(new Card(Rank.THREE, Suit.DIAMONDS));

155         System.out.println("\nSelect Test - chooseGreater - sort by rank");
            card1 = chooseGreater(card, compR, card.get(1));
157         System.out.println("card:" + card.get(1));
            for (Card c : card1) {
159             System.out.println(c);
            }
161
            System.out.println("\nSelect Test - chooseGreater - sort descending");
163         card2 = chooseGreater(card, compD, card.get(1));
            System.out.println("card:" + card.get(1));
165         for (Card c : card2) {
                System.out.println(c);
167         }
        }
169
        public static void main(String[] argvs) {
171         ArrayList<Card> card = new ArrayList<>();
            card.add(new Card(Rank.TEN, Suit.DIAMONDS));
173         card.add(new Card(Rank.FOUR, Suit.SPADES));
            card.add(new Card(Rank.TEN, Suit.SPADES));
175         card.add(new Card(Rank.TWO, Suit.CLUBS));
            card.add(new Card(Rank.SIX, Suit.HEARTS));
177         card.add(new Card(Rank.THREE, Suit.CLUBS));
            card.add(new Card(Rank.THREE, Suit.DIAMONDS));
179
            System.out.println(card);
181
            System.out.println("\nmax(): " + max(card));
183
            System.out.println("\ncompareTo");
```

```
185         Collections.sort(card);
            for (Card c : card) {
187             System.out.println(c);
            }
189
            System.out.println("\n Sort by Rank");
191         Collections.sort(card, new CompareRank());
            for (Card c : card) {
193             System.out.println(c);
            }
195
            System.out.println("\n Sort Descending");
197         Collections.sort(card, new CompareDescending());
            for (Card c : card) {
199             System.out.println(c);
            }
201
            System.out.println("\nChoose Greater");
203         System.out.println("card:" + card.get(3));
            card = chooseGreater(card, new CompareRank(), card.get(3));
205         for (Card c : card) {
                System.out.println(c);
207         }

209         //lamda expression
            compareCards compL = (Card a, Card b) -> {
211             if ((a.rank.value > b.rank.value) || (a.rank.value == b.rank.value))
                    {
                     if (a.suit.value > b.suit.value) {
213                      return a;
                     }
215             }
                return b;
217         };
            selectTest(new CompareRank(), new CompareDescending(), (compareCards)
                compL);
219     }

221     interface compareCards {

223         Card myCards(Card a, Card b);
        }
225 }
```

Deck.java

```
1  package cw2;
2
3  import java.io.File;
4  import java.io.FileInputStream;
5  import java.io.FileOutputStream;
6  import java.io.IOException;
7  import java.io.ObjectInputStream;
8  import java.io.ObjectOutputStream;
9  import java.io.Serializable;
10 import java.util.ArrayList;
11 import java.util.Iterator;
12 import java.util.List;
13 import java.util.Random;
14
15 public class Deck implements Iterable, Serializable {
16
17     static final long serialVersionUID = 49L;
18     private static String fileName = "SPADES";
19     static int MAX_SIZE = 52;
20     Card[] cards;
21     private DeckIterator iterator;
22
23     public Deck() {
24         MAX_SIZE = 52;
25         cards = new Card[MAX_SIZE];
26         newDeck(cards);
27         shuffle(cards);
28         iterator = new DeckIterator(cards);
29     }
30
31     //return the size of the deck
32     public static int size() {
33         return MAX_SIZE;
34     }
35
36     //shuffle the deck
37     private Card[] shuffle(Card[] cards) {
38         Random rand = new Random();
39         Card temp;
40         int j = 0;
41         for (int i = 0; i < cards.length; i++) {
42             j = rand.nextInt(cards.length - 1);
43             temp = cards[i];
44             cards[i] = cards[j];
45             cards[j] = temp;
46         }
47         return cards;
48     }
49
50     //initialise new deck
51     final Card[] newDeck(Card[] cards) {
52         Card.Rank temp = Card.Rank.ACE;
53         for (int i = 0; i < 13; i++) {
54             cards[i] = new Card(temp, Card.Suit.CLUBS);
55             cards[i + 13] = new Card(temp, Card.Suit.DIAMONDS);
56             cards[i + 26] = new Card(temp, Card.Suit.HEARTS);
57             cards[i + 39] = new Card(temp, Card.Suit.SPADES);
58             temp = temp.getNext();
59         }
60         return cards;
61     }
```

```
63      //removes and returns the top card from the deck
        public Card deal() {
65          Card topCard = iterator.next();
            cards[MAX_SIZE - 1] = null;
67          Deck.MAX_SIZE = Deck.MAX_SIZE - 1;
            return topCard;
69      }

71      @Override
        //the iterator for the Deck
73      public Iterator<Card> iterator() {
            return new DeckIterator(cards);
75      }

77      //iterate ftom the last card to the the bottom card, as they are going to be
            dealt
        private class DeckIterator implements Iterator<Card> {
79
            private int nextCard;
81          private final Card[] cards;
            boolean canRemove = false;
83
            public DeckIterator(Card[] cards) {
85              this.cards = cards;
                this.nextCard = size() - 1;
87          }

89          @Override
            public boolean hasNext() {
91              if (nextCard < 0) {
                    return false;
93              }
                return true;
95          }

97          @Override
            public Card next() {
99              canRemove = true;
                Card temp = null;
101             if (!hasNext()) {
                    return null;
103             }
                temp = cards[nextCard--];
105             return temp;
            }
107     }

109     //iterates through spade cards
        public Iterator<Card> spadeIterator() {
111         return new SpadeIterator(cards);
        }
113
        //class of spade iterator
115     private class SpadeIterator implements Iterator<Card> {

117         private int spadesCounter = 0;
            private final Card[] cards;
119         private int nCards = 0;

121         private SpadeIterator(Card[] cards) {
                this.cards = cards;
123         }
```

```
125         @Override
            public boolean hasNext() {
127             return spadesCounter < 13;
            }
129
            //find next Spade card na dreturn it
131         @Override
            public Card next() {
133             Card c = null;
                while (hasNext()) {
135                 c = cards[nCards];
                    nCards++;
137                 if (c.suit.ordinal() == Card.Suit.SPADES.ordinal()) {
                        spadesCounter++;
139                     break;
                    }
141             }
                return c;
143         }
        }
145
        //write spade cards to the serializable file
147     private static void writeOut(List list) throws IOException {
            ObjectOutputStream oos = new ObjectOutputStream(new FileOutputStream(new
               File(fileName)));
149         oos.writeObject(list);
            System.out.println("\nSERIALIZED..");
151         oos.close();
        }
153
        //read spade cards from the serializable file
155     private static void readIn(String filename) throws IOException,
           ClassNotFoundException {
            List<Card> spadesDeSerialize = new ArrayList<Card>();
157         ObjectInputStream ois = new ObjectInputStream(new FileInputStream(new
               File(fileName)));
            spadesDeSerialize = (List<Card>) ois.readObject();
159         ois.close();
            System.out.println("\nDESIRIALIZE..");
161         for (Card c : spadesDeSerialize) {
                System.out.println(c);
163         }
        }
165
        public static void main(String[] argvs) throws IOException,
           ClassNotFoundException {
167         Deck deck = new Deck();
            Card temp;
169         System.out.println("DECK:");
            for (Card c : deck.cards) {
171             System.out.println(c);
            }
173
            System.out.println("\nsize of deck:" + size());
175         System.out.println("\nIterate the cards the way the are being dealt");
            Iterator<Card> itD = deck.iterator();
177         while (itD.hasNext()) {
                temp = itD.next();
179             System.out.println(temp);
            }
181
            temp = deck.deal();
```

```
183         System.out.println("\n deal:" + temp);

185         System.out.println("\n Iterate through the Spade cards in the deck");
            Iterator<Card> itS = deck.spadeIterator();
187
            List<Card> spades = new ArrayList();
189         //traverse the deck and prints the Spade cards
            while (itS.hasNext()) {
191             temp = itS.next();
                spades.add(temp);
193             System.out.println(temp);
            }
195         writeOut(spades);   //write Spade Cards to file
            readIn(fileName);   //read Spade Cards from file
197     }
    }
```

Hand.java

```
package cw2;

import cw2.Card.Rank;
import cw2.Card.Suit;
import java.util.ArrayList;
import java.util.Collection;
import java.util.Collections;
import java.util.Iterator;
import java.util.List;

public final class Hand implements Iterable {

    static final long serialVersionUID = 300L;
    private List<Card> hand = new ArrayList();
    private int hasClubs = 0;
    private int hasHearts = 0;
    private int hasDiamonds = 0;
    private int hasSpades = 0;
    int totalSuits = 0;
    final int countSuit = 0;

    //creates an empty hand
    public Hand() {
        this.hand = hand;
    }

    //create a hand and add an array of cards in it
    public Hand(Card[] cardsArr) {
        this.hand = hand;
        for (int i = 0; i < cardsArr.length; i++) {
            hand.add(cardsArr[i]);
        }
        searchSuits(hand);
        totalValues(hand);
    }

    //crete a hand copy another hand to it
    public Hand(Hand h) {
        this.hand = hand;
        for (int i = 0; i < h.hand.size(); i++) {
            hand.add(i, h.hand.get(i));
        }
        searchSuits(hand);
        totalValues(hand);
    }

    //add card c to the hand
    public void add(Card c) {
        this.hand = hand;
        hand.add(c);
        searchSuits(hand);
    }

    //add the collection c of cards to the hand
    public void add(Collection<Card> c) {
        this.hand = hand;
        for (Card card : c) {
            hand.add(card);
        }
        searchSuits(hand);
    }
```

```
62
        //add to hand another Hand h
64      public void add(Hand h) {
            this.hand = hand;
66          for (int i = 0; i < h.hand.size(); i++) {
                hand.add(h.hand.get(i));
68          }
            searchSuits(hand);
70      }

72      //remove form the hand the card c
        public boolean remove(Card c) {
74          this.hand = hand;
            for (int i = 0; i < hand.size(); i++) {
76              Rank getrank = hand.get(i).rank;
                Suit getsuit = hand.get(i).suit;
78              if ((getrank.ordinal() == c.rank.ordinal()) && (getsuit.value == c.
                    suit.value)) {
                     hand.remove(i);
80                   searchSuits(hand);
                     return true;
82              }
            }
84          return false;
        }
86
        //remove all the cards from a hand h
88      public boolean remove(Hand h) {
            this.hand = hand;
90          if (!h.hand.isEmpty()) {
                for (int i = 0; i < h.hand.size(); i++) {
92                  for (int j = 0; j < hand.size(); j++) {
                        if ((h.hand.get(i).rank.ordinal() == hand.get(j).rank.ordinal
                            ()) && (h.hand.get(i).suit.ordinal() == hand.get(j).suit.
                            ordinal())) {
94                           hand.remove(j);
                        }
96                  }
                }
98              searchSuits(hand);
                return true;
100         } else {
                return false;
102         }
        }
104
        //remove a card from a specified index from the hand
106     public Card remove(int i) {
            this.hand = hand;
108         Card temp = hand.get(i);
            hand.remove(i);
110         searchSuits(hand);
            return temp;
112     }

114     public void totalValues(List<Card> h) {
            //Card.Rank temp = Card.Rank.ACE;
116         int hasACE = 0;
            int total = 0;
118         String s = "";
            //adding frequency of ranks
120         for (Card c : h) {
                total = total + c.rank.value;
```

```
122             if (c.rank.value == Card.Rank.ACE.value) {
                    hasACE++;
124             }
            }
126         for (int i = 0; i <= hasACE; i++) {
                s += (total) + ",";
128             total = total - Rank.ACE.value + 1;
            }
130         System.out.println("total:" + s);
        }
132
        //search the hand and counts the number of each suit
134     public void searchSuits(List<Card> h) {
            this.hasClubs = hasClubs = 0;
136         this.hasDiamonds = hasDiamonds = 0;
            this.hasHearts = hasHearts = 0;
138         this.hasSpades = hasSpades = 0;

140         for (Card c : h) {
                if (c.suit.value == Suit.CLUBS.value) {
142                 hasClubs++;
                } else if (c.suit.value == Suit.DIAMONDS.value) {
144                 hasDiamonds++;
                } else if (c.suit.value == Suit.HEARTS.value) {
146                 hasHearts++;
                } else if (c.suit.value == Suit.SPADES.value) {
148                 hasSpades++;
                }
150         }
            this.totalSuits = hasClubs + hasDiamonds + hasHearts + hasSpades;
152     }

154     @Override
        public Iterator iterator() {
156         this.hand = hand;
            return new AddedIterator(hand);
158     }

160     //iterator class that traverse they hand they way the cards were added
        private class AddedIterator implements Iterator<Card> {
162
            private List<Hand> h;
164         private int nCards = h.size();
            private int counter = 0;
166
            public AddedIterator(List h) {
168             this.h = h;
            }
170
            @Override
172         public boolean hasNext() {
                if (!h.isEmpty() || counter < nCards) {
174                 return true;
                }
176             return false;
            }
178
            @Override
180         public Card next() {
                Card temp = null;
182             if (hasNext()) {
                    temp = hand.get(counter++);
184             }
```

```
            return temp;
        }
    }

    //sort hand in ascending order of ranks
    public static void sortByRank(Hand h) {
        Collections.sort(h.hand, new Card.CompareRank());
    }

    //sorts hand in ascending
    public static void sort(Hand h) {
        Collections.sort(h.hand);

    }

    //counts the number of cards on the hand that have suti s
    public int countSuit(Suit s) {
        int count = 0;
        for (Card c : hand) {
            if (c.suit.value == s.value) {
                count++;
            }
        }
        return count;
    }

    //counts the number of cards on the hand that have rank r
    public int countRank(Rank r) {
        int count = 0;
        for (Card c : hand) {
            if (c.rank.value == r.value) {
                count++;
            }
        }
        return count;
    }

    //check if it has suit s
    public boolean hasSuit(Suit s) {
        for (Card c : hand) {
            if (c.suit.value == s.value) {
                return true;
            }
        }
        return false;
    }

    @Override
    public String toString() {
        return this.hand.toString();
    }

    //returns the hand list
    public List<Card> myHand() {
        return this.hand;
    }

    public static void main(String[] argvs) {
        Card[] array = new Card[13];
        array[0] = new Card(Rank.ACE, Suit.SPADES);
        array[1] = new Card(Rank.ACE, Suit.CLUBS);
        array[2] = new Card(Rank.ACE, Suit.CLUBS);
        array[3] = new Card(Rank.FOUR, Suit.SPADES);
```

```
248         array[4] = new Card(Rank.FIVE, Suit.DIAMONDS);
            array[5] = new Card(Rank.SIX, Suit.HEARTS);
250         array[6] = new Card(Rank.SEVEN, Suit.DIAMONDS);
            array[7] = new Card(Rank.EIGHT, Suit.DIAMONDS);
252         array[8] = new Card(Rank.NINE, Suit.CLUBS);
            array[9] = new Card(Rank.TEN, Suit.DIAMONDS);
254         array[10] = new Card(Rank.JACK, Suit.DIAMONDS);
            array[11] = new Card(Rank.QUEEN, Suit.DIAMONDS);
256         array[12] = new Card(Rank.KING, Suit.DIAMONDS);

258         ArrayList<Card> cardList = new ArrayList();
            cardList.add(new Card(Rank.ACE, Suit.SPADES));
260         cardList.add(new Card(Rank.JACK, Suit.HEARTS));
            cardList.add(new Card(Rank.SEVEN, Suit.CLUBS));
262         cardList.add(new Card(Rank.SIX, Suit.HEARTS));

264         Card temp;

266         Hand handP1 = new Hand();
            Hand handP2 = new Hand(array);
268         Hand handP3 = new Hand(handP2);
            System.out.println("\nTotal of handP2:");
270         handP2.totalValues(handP2.hand);

272         System.out.println("\nHAND 2");
            System.out.println(handP2);
274
            System.out.println("\nIterate the cards the way the were added");
276         Iterator<Card> itA = handP2.hand.iterator();
            while (itA.hasNext()) {
278             temp = itA.next();
                System.out.println(temp);
280         }

282         System.out.println("\nAdd a single card:");
            handP2.add(new Card(Rank.SEVEN, Suit.CLUBS));
284         System.out.println(handP2);

286         System.out.println("\n Add a collection to the hand");
            handP2.add(cardList);
288         System.out.println(handP2);

290         System.out.println("\nAdd a hand to another hand");
            handP2.add(handP3);
292         System.out.println(handP2);

294         System.out.println("\nRemove a card");
            handP2.remove(new Card(Rank.ACE, Suit.SPADES));
296         System.out.println(handP2);

298         System.out.println("\n Remove all cards if present form another hand");
            handP2.remove(handP3);
300         System.out.println(handP2);

302         System.out.println("\nRemove a card at a specific position");
            handP2.remove(0);
304         System.out.println(handP2);

306         System.out.println("\nSort hand in ascending");
            sort(handP2);
308         System.out.println(handP2);

310         System.out.println("\n sort by rank");
```

```
            sortByRank(handP2);
312         System.out.println(handP2);

314         System.out.println("\nHAND 3:");
            System.out.println(handP3);
316         System.out.println("\nCount Suit");
            System.out.println("Suit: Clubs");
318         System.out.println(handP3.countSuit(Suit.CLUBS));

320         System.out.println("\nCount Rank");
            System.out.println("Rank: Jack");
322         System.out.println(handP3.countRank(Rank.JACK));

324         System.out.println("\nhasSuit");
            System.out.println("Suit: Clubs");
326         System.out.println(handP2.hasSuit(Suit.CLUBS));
        }
328 }
```

Trick.java

```
package cw2;

import cw2.Card.*;
import java.util.ArrayList;
import java.util.List;

public class Trick {

    public static Suit trumps;
    Suit lead;
    int playerID;
    List<Card> cardsPlayed;
    public static Card[] trumpCards = new Card[BasicWhist.NOS_PLAYERS];

    public static Card[] cardList = new Card[BasicWhist.NOS_PLAYERS];

    public Trick(int p) {    //p is the lead player
        this.playerID = p;
        cardsPlayed = new ArrayList();
        this.lead = lead;
        this.trumpCards = initTrump();  //initialise trump array to null
    }

    //resets the trump list for every trick
    public Card[] initTrump() {
        for (int i = 0; i < trumpCards.length; i++) {
            trumpCards[i] = null;
        }
        return trumpCards;
    }

    //returns the trump card
    public Suit getTrumps() {
        return trumps;
    }

    //trump cards is set at the beggining of the game and always wins
    public static void setTrumps(Suit s) {
        System.out.println("Trump Suit:" + s);
        trumps = s;
    }

    //returns the Suit of the lead card.
    public Suit getLeadSuit() {
        return this.lead;
    }

    // Records the Card c played by Player p for this trick
    public void setCard(Card c, BasicPlayer p) {
        this.cardsPlayed.add(c);
        p.myHand.remove(c);

        //if card played is trump, add it to the trump list for that round
        if (c.suit == trumps) {
            trumpCards[p.getID()] = c;
        } else {
            cardList[p.getID()] = c;
        }
        System.out.println("Player " + p.getID() + ": " + c);

    }
```

```
62
        //Returns the card played by player with id p for this trick
64      public Card getCard(BasicPlayer p) {
            return p.playedCard();
66      }

68      //being used for the humanGame to detect the user's ID
        public BasicPlayer findPlayer(int id) {
70          return BasicWhist.players[id];
        }
72
        //Finds the ID of the winner of a completed trick
74      public int findWinner() {
            int win = searchTrick();
76          System.out.println("");
            return win;
78      }

80      //search the trick for either trump cards or cards that follow suit
        public int searchTrick() {
82          int winner;
            if (searchList(trumpCards) >= 1) {    //search, if there are any trump
                cards
84              winner = searchTrumps(trumpCards);   //find the winner here
            } else {
86              winner = findPlayer(cardList);  //else if it follows suit,search here
            }
88          return winner;
        }
90
        //searching the trump card list of the round
92      public int searchTrumps(Card[] trumps) {
            Rank r = Rank.TWO;
94          int winner = 0;
            for (int i = 0; i < trumpCards.length; i++) {
96              if ((trumpCards[i] != null) && (trumpCards[i].rank.ordinal() > r.
                    ordinal())) {
                    winner = i;
98                  r = trumpCards[i].rank;
                }
100         }
            return winner;
102     }

104     //find the highest card of the round
        public int findPlayer(Card[] cards) {
106         int roundWinnerID = 0;
            Rank r = Rank.TWO;
108         for (int i = 0; i < cards.length; i++) {
                if (cards[i].rank.ordinal() > r.ordinal()) {
110                 r = cards[i].rank;
                    roundWinnerID = i;
112             }
            }
114         resetList(cards);
            return roundWinnerID;
116     }

118     //search the list to check if any trump cards
        public int searchList(Object[] array) {
120         int count = 0;
            for (int i = 0; i < array.length; i++) {
122             if (array[i] != null) {
```

```
                count++;
124             }
            }
126         return count;
        }
128
        //resets the array to null
130     public Card[] resetList(Card[] c) {
            for (int i = 0; i < c.length; i++) {
132             c[i] = null;
            }
134         return c;
        }
136 }
```

BasicPlayer.java

```
package cw2;

import cw2.Card.Suit;

public class BasicPlayer implements Player {

    public Hand myHand;
    private Suit trump;
    Strategy bs;
    Card played;

    public BasicPlayer() {
        myHand = new Hand();
        this.trump = trump;
        bs = new BasicStrategy();
    }

    //returns the curent hand
    public Hand getHand() {
        return myHand;
    }

    //adds card to the players hand
    @Override
    public void dealCard(Card c) {
        this.myHand.add(c);
    }

    //sets the strategy
    @Override
    public void setStrategy(Strategy s) {
        this.bs = s;
    }

    //Determines which of the players cards to play based on the in play trick t
       and player strategy
    @Override
    public Card playCard(Trick t) {
        Card temp;
        temp = this.bs.chooseCard(myHand, t);
        this.played = temp;
        return temp;
    }

    //let players view the trick
    @Override
    public void viewTrick(Trick t) {
        System.out.println("Trick=" + t.cardsPlayed);
    }

    //set the trump card to the players
    @Override
    public void setTrumps(Card.Suit s) {
        trump = s;
    }

    //gets the players' id
    @Override
    public int getID() {
        return BasicWhist.counterPlayer;
    }
```

```
62      //returns the played card of a player
        Card playedCard() {
64          return this.played;
        }
66
        //finds the current player
68      public BasicPlayer findPlayer(int id) {
            return BasicWhist.players[id];
70      }
    }
```

BasicStrategy.java

```
1  package cw2;
2
3  import cw2.Card.Rank;
4  import cw2.Card.Suit;
5
6  public class BasicStrategy implements Strategy {
7
8      public Integer[] scores = new Integer[BasicWhist.NOS_PLAYERS];
9
10     public BasicStrategy() {
11     }
12
13     //choose card from hand
14     @Override
15     public Card chooseCard(Hand h, Trick t) {
16         Card card = new Card(Rank.TWO, Suit.SPADES);
17         Card temp = null;
18         Card theC;
19         if (t.cardsPlayed.isEmpty()) {  //choose card for frst player to play
20             for (Card c : h.myHand()) {
21                 if (c.rank.ordinal() > card.rank.ordinal()) {
22                     card = c;
23                 }
24             }
25             t.lead = card.suit; //set the lead suit
26             System.out.println("Lead Suit:" + t.getLeadSuit());
27             return card;
28         } else {
29             for (int i = 0; i < h.myHand().size(); i++) {
30                 theC = h.myHand().get(i);
31                 //follow suit if it can
32                 if ((theC.rank.ordinal() > card.rank.ordinal()) && (theC.suit ==
                       t.getLeadSuit())) {
33                     temp = theC;
34                 } else if ((i == h.myHand().size() - 1) && (temp == null)) {
35                     temp = canTrump(h, t);
36                 }
37             }
38             return temp;
39         }
40     }
41
42     //check if it has trump card
43     Card canTrump(Hand h, Trick t) {
44         Card tmp = null;
45         for (int i = 0; i < h.myHand().size(); i++) {
46             Card c = h.myHand().get(i);
47             if (c.suit == t.getTrumps()) {
48                 tmp = c;
49             } else if (i == h.myHand().size() - 1 && (tmp == null)) {
50                 int temp = (int) ((h.myHand().size()) * Math.random());
51                 tmp = h.myHand().get(temp);
52             }
53         }
54         return tmp;
55     }
56
57     @Override
58     public void updateData(Trick t) {
59     }
```

```
61 }
```

HumanStrategy.java

```
1  package cw2;
2
3  import java.util.InputMismatchException;
4  import java.util.Scanner;
5
6  public class HumanStrategy implements Strategy {
7
8      public HumanStrategy() {
9      }
10
11     //let the user choose a card
12     @Override
13     public Card chooseCard(Hand h, Trick t) {
14         int j = 0;
15         Card played = null;
16         System.out.println("\nUSER'S TURN!");
17         System.out.println("Your Hand:");
18         for (Card c : h.myHand()) {
19             System.out.println(j + " " + c);
20             j++;
21         }
22         boolean hasSuit = true;
23         int i = 0;
24         Scanner scan = new Scanner(System.in);
25
26         do {
27             System.out.print("Enter a number in the range (0-" + (h.myHand().size
                   () - 1) + "):");
28             try {
29                 i = scan.nextInt();
30                 if (i < 0 || (i >= h.myHand().size())) {    //check input to be
                       within range
31                     System.out.println("Cards not in range, please pick one which
                           is!");
32                     System.out.println("Range (0-" + (h.myHand().size() - 1) + ")
                           ");
33                 } else {
34                     played = h.myHand().get(i);
35                     System.out.println("LEAD SUIT:" + t.getLeadSuit());
36                     System.out.println("Card Selected:" + played);
37                     if (t.getLeadSuit() == null) //if user is first, chooses the
                           lead suit
38                     {
39                         hasSuit = false;
40                     }
41                     if (played.suit != t.getLeadSuit()) {
42                         hasSuit = searchHand(h, t);  //searches the hand for a
                               lead suit
43                     } else {
44                         hasSuit = false;
45                     }
46                 }
47             } catch (InputMismatchException a) {  //catch error for non-number
                   input
48                 System.out.println("Must enter a number in the range 0-" + (h.
                       myHand().size() - 1));
49                 scan.next();
50             }
51         } while (hasSuit);
52
53         t.setCard(played, t.findPlayer(BasicWhist.humanID));
```

```
        return played;
    }

    //search hand for lead suit
    public boolean searchHand(Hand h, Trick t) {
        for (Card c : h.myHand()) {
            if (c.suit == t.getLeadSuit()) {
                System.out.println("You have to follow the Lead Suit");
                return true;
            }
        }
        return false;
    }

    @Override
    public void updateData(Trick c) {
        throw new UnsupportedOperationException("Not supported yet."); //To
            change body of generated methods, choose Tools | Templates.
    }

}
```

AdvancedStrategy.java

File not found.

BasicWhist.java

```
1  package cw2;
2
3  import cw2.Card.Suit;
4  import java.util.Collections;
5  import java.util.InputMismatchException;
6  import java.util.Scanner;
7
8  public class BasicWhist {
9
10     static final int NOS_PLAYERS = 4;
11     static final int NOS_TRICKS = 13;
12     static final int WINNING_POINTS = 7;
13     int team1Points = 0;
14     int team2Points = 0;
15     static BasicPlayer[] players;
16     BasicStrategy bs;
17     HumanStrategy human;
18     static int counterPlayer;
19     static int humanID = 2; //the user has a static ID
20
21     //constructor for the four players for the BasicGame
22     public BasicWhist(BasicPlayer[] pl) {
23         this.players = pl;
24         bs = new BasicStrategy();
25     }
26
27     //constrructor the the humanGame players
28     //includeing the BasicPlayers and the HumanPlayer
29     public BasicWhist(BasicPlayer[] p, HumanStrategy s) {
30         this.players = p;
31         this.human = s;
32     }
33
34     //deal cards to the hands
35     public void dealHands(Deck newDeck) {
36         for (int i = 0; i < 52; i++) {
37             players[i % NOS_PLAYERS].dealCard(newDeck.deal());
38         }
39         for (int i = 0; i < players.length; i++) {
40             Collections.sort(players[i].myHand.myHand());   //sort each hand in
                   ascending
41         }
42     }
43
44     //each players play a card that s added the trick
45     public Trick playTrick(Player firstPlayer) {
46         Trick t = new Trick(firstPlayer.getID());
47         int playerID = firstPlayer.getID();
48         for (int i = 0; i < NOS_PLAYERS; i++) {
49             int next = (playerID + i) % NOS_PLAYERS;
50             counterPlayer = next;
51             t.setCard(players[next].playCard(t), players[next]);
52         }
53         return t;
54     }
55
56     //play a basic game of 13 rounds
57     public void playGame() {
58         Deck d = new Deck();
59         dealHands(d);
```

```
61          int firstPlayer = (int) (NOS_PLAYERS * Math.random()); //choose a random
                player to start first
            Suit trumps = Suit.randomSuit();  //choose the trump card
63          Trick.setTrumps(trumps);       //sets trump card
            for (int i = 0; i < players.length; i++) {
65              players[i].setTrumps(trumps);
            }
67
            int i = 0;
69          //checks for the team scores, in case one team
            //scores 7 before the end of the game
71          while (i < NOS_TRICKS && checkPoints()) {
                System.out.println("\n*****STARTING NEW TRICK********* round:" + i);
73              Trick t = playTrick(players[firstPlayer]);
                for (int j = 0; j < players.length; j++) {
75                  System.out.println(j + " hand " + players[j].myHand);
                }
77              players[i % NOS_PLAYERS].viewTrick(t);
                firstPlayer = t.findWinner();   //search for round winner
79              addTeamPoints(firstPlayer);
                System.out.println("Winner of the trick =" + firstPlayer);
81              i++;
            }
83      }

85      // Method to find the winner of a trick. Note
        public void playMatch() {
87          while (team1Points < WINNING_POINTS && team2Points < WINNING_POINTS) {
                playGame();
89              System.out.println("*********************************");
            }
91          if (team1Points >= WINNING_POINTS) {
                System.out.println("Winning team is team 1 with " + team1Points);
93          } else {
                System.out.println("Winning team is team 2 with " + team2Points);
95          }
        }
97
        //play tricks
99      public Trick playHumanTrick(Player firstPlayer) {
            Trick t = new Trick(firstPlayer.getID());
101         int playerID = firstPlayer.getID();
            System.out.println("Trump Suit:" + Trick.trumps + " Player:" +
                firstPlayer.getID());
103         for (int i = 0; i < NOS_PLAYERS; i++) {
                int next = (playerID + i) % (NOS_PLAYERS);
105             counterPlayer = next;  //counterPlayer gives the id to the getID
                    method
                if (next == humanID) {
107                 this.human.chooseCard(players[next].myHand, t);
                } else {
109                 t.setCard(players[next].playCard(t), players[next]);
                }
111         }
            return t;
113     }

115     //play a game with the user
        public void playHumanGame() {
117         Deck d = new Deck();
            dealHands(d);
119
```

```
        int firstPlayer = (int) ((NOS_PLAYERS) * Math.random());//chooses the
            first player random
121     Suit trumps = Suit.randomSuit();
        Trick.setTrumps(trumps);
123     for (int i = 0; i < players.length; i++) {  //set trump card
            players[i].setTrumps(trumps);
125     }

127     Trick t;
        int i = 0;
129     while (i < NOS_TRICKS && checkPoints()) {
            System.out.println("\n*********NEXT TRICK********** " + i);
131         t = playHumanTrick(players[firstPlayer]);
            players[i % NOS_PLAYERS].viewTrick(t);
133         firstPlayer = t.findWinner();
            addTeamPoints(firstPlayer);
135         System.out.println("Winner of the Trick=" + firstPlayer);
            i++;
137     }

139 }

141 // Method to find the winner of a trick. Note
    public void playHumanMatch() {
143     while (team1Points < WINNING_POINTS && team2Points < WINNING_POINTS) {
            playHumanGame();
145         System.out.println("*****************************\n");
        }
147     if (team1Points >= WINNING_POINTS) {
            System.out.println("Winning team is team 1 with " + team1Points);
149     } else {
            System.out.println("Winning team is team 2 with " + team2Points);
151     }
    }
153
    public static void humanGame() {
155     BasicPlayer[] p = new BasicPlayer[NOS_PLAYERS];
        HumanStrategy s = new HumanStrategy();
157     Scanner scan = new Scanner(System.in);
        boolean playAgain = true;
159     String input;

161     for (int i = 0; i < p.length; i++) {
            p[i] = new BasicPlayer();   //create four players
163     }
        System.out.println("User is on team 1");
165     BasicWhist hg = new BasicWhist(p, s);
        hg.playHumanMatch();
167     do {
            hg.playHumanMatch(); //Just plays a single match
169         System.out.println("Do you want to play another game?(Y/n)");
            try {
171             input = scan.next();
                if (input.equals("N") || input.equals("n")) {
173                 System.out.println("EXITING GAME...");
                    playAgain = false;
175             } else if (input.equals("y") || input.equals("Y")) {
                    System.out.println("\n\nStarting Game...\n\n");
177                 hg.team1Points = 0;
                    hg.team2Points = 0;
179             }
            } catch (InputMismatchException a) {
181             System.out.println("okk");
```

```
                scan.next();
            }
        } while (playAgain);
    }

    public static void basicGame() {
        Scanner scan = new Scanner(System.in);
        char c;
        String input;
        boolean playAgain = true;
        BasicPlayer[] p = new BasicPlayer[NOS_PLAYERS];
        for (int i = 0; i < p.length; i++) {
            p[i] = new BasicPlayer();   //CREATE YOUR PLAYERS HERE
        }
        BasicWhist bg = new BasicWhist(p);
        do {
            bg.playMatch(); //Just plays a single match
            System.out.println("Do you want to play another game?(Y/n)");
            try {
                input = scan.next();
                if (input.equals("N") || input.equals("n")) {
                    System.out.println("EXITING GAME...");
                    playAgain = false;
                } else if (input.equals("y") || input.equals("Y")) {
                    System.out.println("\n\nStarting Game...\n\n");
                    bg.team1Points = 0;
                    bg.team2Points = 0;
                }
            } catch (InputMismatchException a) {
                System.out.println("okk");
                scan.next();
            }
        } while (playAgain);

    }

    //check the team points, in case a team scores 7 before the end of the game
    public boolean checkPoints() {
        if (team1Points == 7 || team2Points == 7) {
            return false;
        }
        return true;
    }

    //add up the points to each team
    public void addTeamPoints(int p) {
        if ((p == 0) || (p == 2)) {
            team1Points++;
        } else {
            team2Points++;
        }

        System.out.println("team " + 1 + " points:" + team1Points);
        System.out.println("team " + 2 + " points:" + team2Points);
    }

    public static void main(String[] args) {
        int choice;

        System.out.println("Choose a Game Mode:");
        System.out.println("1 = basicGame");
        System.out.println("2 = humanGame");
        Scanner scan = new Scanner(System.in);
```

```
245             choice = scan.nextInt();
                switch (choice) {
247                 case 1:
                        basicGame();
249                     break;
                    case 2:
251                     humanGame();
                        break;
253             }

255     }
    }
```

PlayerDescription.pdf

File not found.