CMP-5015Y Assignment 2 (Java)

100108964 (kzy14tcu)

Wed, 6 Feb 2019 12:56

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

```
/*
 * To change this license header, choose License Headers in Project Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
package cards;

import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.ObjectInputStream;
import java.io.ObjectOutputStream;
import java.io.Serializable;
import java.util.ArrayList;
import java.util.Collections;
import static java.util.Collections.list;
import java.util.Comparator;
import java.util.Iterator;
import java.util.List;
import java.util.Objects;
import java.util.Random;
/*
 *
 * @author Nahim
 */
public class Card implements Serializable, Comparable<Card>,Comparator<Card> {

    private static final long serialVersionUID = 100L;
    private Rank rank;
    private Suit suit;

    /**
     * Enum to hold Rank and suit values for cards
     */
    public enum Rank {

        TWO(2,1),THREE(3,2),FOUR(4,3),FIVE(5,4),SIX(6,5),SEVEN(7,6),EIGHT(8,7),
           NINE(9,8),TEN(10,9),JACK(10,10),QUEEN(10,11),
        KING(10,12),ACE(11,13);

        private int cardValue;
        private int cardRank;
        public static Rank[] values = values();


        private Rank(int value, int cardRank)
        {
            this.cardValue = value;
            this.cardRank = cardRank;
        }

        //getNext method - returns next enum value
        public Rank getNext()
        {
            return values()[(ordinal()+1)%values().length];
        }

        //returns previous
        public Rank getPrevious(){
            return values()[(ordinal()-1)%values.length];
```

```
61          }
63          public int getValue()
            {
65              return cardValue;
            }
67
            public int getCardRank(){
69              return cardRank;
            }
71
        };
73
        /**
75       * Enumeration to store Suit values
         */
77      public enum Suit
        {
79          CLOVERS, DIAMONDS, HEARTS, SPADES;
81          public static Suit[] values = values();
83          /**
             * method that returns random suit
85           * @return value of suit
             */
87          public static Suit getRandomSuit()
            {
89              Random random = new Random();
                int randomSuit = new Random().nextInt(values.length);
91              //return Suit.values()[random.nextInt(Suit.values().length)];
                return values[randomSuit];
93          }
95      };
97
        /**
99       * Default Constructor
         * @param rank
101      * @param suit
         */
103     public Card(Rank rank, Suit suit)
        {
105         this.rank = rank;
            this.suit = suit;
107     }
109     /**
         * Test constructor
111      */
        public Card()
113     {
115     }
117     /**
         *
119      * @return random suit
         */
121     public Suit getRandomSuit()
            {
123             Random random = new Random();
```

```
125             return Suit.values()[random.nextInt(Suit.values().length)];
        }
127
    /**
129      * Method to set Rank and value
     * @param rank
131      */
    public void setRank(int rank)
133 {

135         this.rank.getValue();
    }
137
    /**
139      *
     * @return rank
141      */
    public Rank getRank()
143 {
        return rank;
145 }

147 /**
     * Method to set suit
149  * @param suit
     */
151 public void setSuit(Suit suit)
    {
153     this.suit = suit;
    }
155
    /**
157  *
     * @return suit
159  */
    public Suit getSuit()
161 {
        return suit;
163 }

165    // implement natural order
    @Override
167 public int compareTo(Card o) {
        int rankCompare = rank.compareTo(o.rank);
169     return rankCompare != 0 ? rankCompare : suit.compareTo(o.suit);
    }
171
    @Override
173 public int compare(Card o1, Card o2) {
        int n = o1.compareTo(o2);
175     if(n == 1 || n == -2){
            return -1;
177     }
        return 0;
179 }

181 /**
     * Sort in Ascending order
183  * @param one
     * @param two
185  * @return
     */
```

```
187     public  int compareTo(Card one, Card two){
            if(this.rank.compareTo(two.rank) < 0){
189             return -1;
            }
191         else if(this.rank.compareTo(two.rank) == 0){
                if(this.suit.compareTo(two.suit) < 0){
193                 return -2;
                }
195         else if(this.suit.compareTo(two.suit) == 0){
                return 0;
197         }
            else if(this.suit.compareTo(two.suit) > 0){
199             return 2;
            }
201         }

203         return 1;
        }
205
        /**
207      * Method to get the max card in list
         * @param cards
209      * @return
         */
211     public static Card max(List<Card> cards){
            Card max = cards.get(0);
213         Iterator<Card> it = cards.iterator();

215         while(it.hasNext()){
                int compare = max.compareTo(it.next());
217             if(compare == 1){
                    max = it.next();
219             }
            }
221         return max;
        }
223
        @Override
225     public int hashCode() {
            int hash = 5;
227         hash = 19 * hash + Objects.hashCode(this.rank);
            hash = 19 * hash + Objects.hashCode(this.suit);
229         return hash;
        }
231
        @Override
233     public boolean equals(Object obj){
            if(obj == null || !(obj instanceof Card)){
235             return false;
            }
237         Card c = (Card)obj;
            return this.compareTo(c) == 0;
239     }


241
        /**
243      * Nested class sort descending order of Rank
         */
245     public static class CompareDescending implements Comparator<Card>{

247         @Override
            public int compare(Card o1, Card o2) {
249             int rank = o1.compareTo(o2);
```

```
            if(rank == 0)
            {
               // return o1.suit.ordinal() - o2.suit.ordinal();#
                return -1;
            }
            else{
                return 0;
            }
        }

}
    /**
     * Nested class sort into ascending order by suit then rank
     */
    public class CompareRank implements Comparator<Card>{

        @Override
        public int compare(Card t, Card t1) {
            if(t.rank.ordinal() > t1.rank.ordinal()){
                return 1;
            }
            else if(t.rank.ordinal() < t1.rank.ordinal()){
                return -1;
            }else{
                return 0;
            }

     }


}
     /**
      * Returns a ArrayList of cards passing a comparator based on
      * being greater than the card passed through the parameter.
      * @param card
      * @param c
      * @param compare
      * @param acard
      * @return
      */
     public ArrayList<Card> chooseGreater(ArrayList<Card> card, Card c,Comparator<
        Card> compare, Card acard){
         Iterator<Card> iterator = card.iterator();
         ArrayList<Card> newList = new ArrayList();

         while(iterator.hasNext()){
             Card c1 = iterator.next();
             if(compare.compare(acard, c1) == -1){
                 newList.add(c);
             }
         }
         return newList;
     }


     /**
      * Uses two comparators and lambda expression to call ChooseGreater method
         with
      * a list of cards. Lambda implements logic that Card A is greater than Card
         B
      * if Rank of A is greater than Rank B or if the ranks are equal, than Card A
           is greater than B if the
      * suit is greater than B
```

```
309         * @param list
            * @param card
311         * @return
            */
313        public ArrayList<Card> selectTest(ArrayList<Card> list, Card card){
               Comparator<Card> Rank = new CompareRank();
315            Comparator<Card> descending = new CompareDescending();
               ArrayList<Card> aList = new ArrayList();
317
               list.forEach(cards ->{
319                if(Rank.compare(card, card) == -1){
                       aList.add(cards);
321                }
                   else if(card.getRank() == cards.getRank()){
323                    if(descending.compare(card, card) == -1){
                           aList.add(card);
325                    }
                   }
327            });

329            return aList;
           }
331

333        /**
            * Method to save Serialisation object
335         * @param fileName
            * @throws IOException
337         * @throws ClassNotFoundException
            */
339        public void saveFile(String fileName) throws IOException,
               ClassNotFoundException{
               FileOutputStream in = new FileOutputStream(fileName);
341            try(ObjectOutputStream out = new ObjectOutputStream(in)){
                   out.writeObject(this);
343            }
       }
345

347        /**
            * Method to load serialisation Card object
349         * @param filename
            * @return
351         * @throws FileNotFoundException
            * @throws IOException
353         * @throws ClassNotFoundException
            */
355        public Card load(String filename) throws FileNotFoundException, IOException,
               ClassNotFoundException{
               try{
357                FileInputStream in = new FileInputStream(filename);
                   Card l = null;
359                try{
                       ObjectInputStream fin = new ObjectInputStream(in);
361                    l = (Card)fin.readObject();
                   }catch(IOException e){
363                }
                   return l;
365            }catch(IOException e){
               }
367
               return null;
369        }
```

```
371
373
375     /**
         * toString to return cards
377      * @return
         */
379     @Override
        public String toString(){
381         StringBuilder st = new StringBuilder();
            st.append(this.rank).append(" ").append(this.suit);
383         return st.toString();
        }
385
        public static void main(String[] args){
387         //Test Card.java methods

389         //Add cards to hand
            ArrayList<Card> cards = new ArrayList();
391         cards.add(new Card(Rank.ACE,Suit.SPADES));
            cards.add(new Card(Rank.QUEEN, Suit.DIAMONDS));
393         cards.add(new Card(Rank.EIGHT,Suit.HEARTS));
            cards.add(new Card(Rank.JACK, Suit.CLOVERS));
395
            Card ACEOFSPADE = new Card(Rank.ACE,Suit.SPADES);
397

399         //sorted list in ascending order
            System.out.println(cards + "\n");
401
            //print by rank
403         Collections.sort(cards,Card.CompareRank());

405         ArrayList<Card> choosegreater = chooseGreater(cards,compareRank());
            System.out.println(choosegreater);
407     }

409 }
```

Deck.java

```
/*
 * To change this license header, choose License Headers in Project Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
package cards;

import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.ObjectInputStream;
import java.io.ObjectOutputStream;
import java.io.Serializable;
import java.lang.reflect.Array;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.Collections;
import java.util.Iterator;
import java.util.List;
import java.util.ListIterator;

/**
 *
 * @author Nahim
 */
public class Deck implements Iterable<Card>, Serializable {
    int fixedSize = 52;
    private List<Card> cards = Arrays.asList(new Card[fixedSize]);
    private int cardsLeft;
    private static final long serialVersionUID = 49;


    /**
     * Deck constructor - creates list of cards and fills deck with cards
     */
    public Deck(){
        cards = new ArrayList();
        for(Card.Suit suit : Card.Suit.values()){
            for(Card.Rank rank : Card.Rank.values()){
                cards.add(new Card(rank,suit));
            }
        shuffle();

        }

    }

    /**
     * Method to shuffle deck using Collections
     */
    public void shuffle(){
        Collections.shuffle(cards);

    }

    /**
     * Method to deal deck
     * @return
     */
    public Card deal(){
```

```
        Card top = null;
        //if card is available
        if(cards.size() > 0){
            //get top card
            top = this.cards.get(0);
            //remove from deck
            this.cards.remove(top);
            //cardsleft--
            this.cardsLeft--;
        }else{
            //make a new deck
            newDeck();
    }
        return top;
    }

    /**
     *
     * @return size of hand list
     */
    public int size(){
        return this.cards.size();
    }

    /**
     * Method to create new deck
     */
    public void newDeck(){
        cards.clear();
        Deck newDeck = new Deck();
        newDeck.shuffle();
        this.cards = newDeck.cards;

        //this.cardsLeft = 52;
        shuffle();

    }

    /**
     * Method to save the deck in SpadeIterator order
     * @param fileName
     * @throws IOException
     * @throws ClassNotFoundException
     */
    public void saveFile(String fileName) throws IOException,
       ClassNotFoundException{
        FileOutputStream in = new FileOutputStream(fileName);
        try(ObjectOutputStream out = new ObjectOutputStream(in)){
            out.writeObject(this);
        }
    }

    public Deck load(String file) throws FileNotFoundException, IOException,
       ClassNotFoundException{
        Deck d = null;
        try{
            FileInputStream in = new FileInputStream(file);
            ObjectInputStream fin = new ObjectInputStream(in);
            d = (Deck)fin.readObject();
        }catch(IOException e){
            e.printStackTrace();
        }
        return d;
```

```
123     }
125     /**
         * write file
127      * @param out
         * @throws IOException
129      */
        public void write(ObjectOutputStream out) throws IOException{
131         out.defaultWriteObject();
            Card c;
133         Iterator<Card> iterator = SpadeIterator();
            ArrayList<Card> list = new ArrayList();
135
            while(iterator.hasNext()){
137             c = iterator.next();
                System.out.println(c.toString());
139             list.add(c);
            }
141         out.writeObject(list);
143     }
145     public Card getCard(int c){
            return cards.get(c);
147     }
149
         public void Reverse(List<Card> cards){
151          this.cards = cards;
         }
153
        @Override
155      public Iterator<Card> iterator(){
             return new DeckIterator(cards);
157
         }
159
161     /**
         * Nested class to traverse Deck in order cards are delt
163      */
        public class DeckIterator implements Iterator<Card>{
165
            private List<Card> cards;
167         private Deck deck;
            private int position;
169
            DeckIterator(List<Card> cards){
171             this.cards = new ArrayList();
                this.deck = deck;
173             this.position = cards.size();
            }
175
            // return postion
177          @Override
             public boolean hasNext() {
179             return position > 0;
             }
181
             @Override
183          public Card next() {
                 while(hasNext()){
185                  Card spade = cards.remove(position--);
```

```
                if(spade.getSuit().compareTo(Card.Suit.SPADES) == 0)
187                 return spade;
             }
189          return deck.getDeck().get(position);
         }
191
        @Override
193      public void remove(){
             cards.remove(position);
195      }

197  }

199  public List<Card> getDeck(){
         return cards;
201  }

203  /**
      * Iterator that traverses spades in the deck
205   * @return
      */
207  public Iterator<Card> SpadeIterator(){
         return new SpadeIterator(cards);
209  }

211  private class SpadeIterator implements Iterator<Card>{

213      private Deck d;
         private Card c;
215      int position = 0;

217
         public SpadeIterator(List<Card> cards){
219          cards = new ArrayList();
         }
221
         @Override
223      public boolean hasNext() {
            return position < size();
225      }

227      @Override
         public Card next() {
229          position++;
             if(hasNext()){
231              for(Card c : d){
                     if(c.getSuit().equals(Card.Suit.SPADES)){
233                      cards.add(c);
                     }
235              }
             }
237          throw new UnsupportedOperationException("Not supported yet."); //To
                 change body of generated methods, choose Tools | Templates.
         }
239
  }
241
  public static void main(String[] args) throws FileNotFoundException,
     IOException, ClassNotFoundException{
243      //Deck test

245      Deck deck = new Deck();
```

```
247             Iterator<Card> it2 = deck.iterator();
                while(it2.hasNext()){
249                 System.out.println(it2.next().toString());
            }
251
            //test serialization
253         Iterator<Card> it = deck.iterator();

255         while(it.hasNext()){
                System.out.println(it.next() + "");
257     }
            try{
259             FileOutputStream fout = new FileOutputStream("deck.ser");
                ObjectOutputStream out = new ObjectOutputStream(fout);
261             out.writeObject(deck);
                out.close();
263         }catch(IOException ex){
                ex.printStackTrace();
265         }

267         it = deck.SpadeIterator();
            while(it.hasNext()){
269             System.out.println(it.next().toString() + "");
            }
271
            deck.newDeck();
273
            try{
275             FileInputStream fin = new FileInputStream("deck.ser");
                ObjectInputStream in = new ObjectInputStream(fin);
277             deck = (Deck)in.readObject();
                in.close();
279         }catch(IOException ex){
                ex.printStackTrace();
281         }


283


285
    }
287
    }
```

Hand.java

```
/*
 * To change this license header, choose License Headers in Project Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
package cards;

import java.util.ArrayList;
import java.util.Arrays;
import java.util.Collections;
import java.util.Iterator;
import cards.Card.Rank;
import cards.Card.Suit;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.ObjectInputStream;
import java.io.ObjectOutputStream;

/**
 *
 * @author Nahim
 */
public class Hand implements Iterable<Card>{
    //cards in a hand
    private ArrayList<Card> hand;
    //keeps track of cards
    private int noofCards;

    private static final long serialVersionUID = 300L;

    // default constructor
    public Hand(){
        hand = new ArrayList();
        noofCards = hand.size();

    }
    //hand constructor
    public Hand(Card[] arrayofCards){
        //adds card array to hand
        hand.addAll(Arrays.asList(arrayofCards));
        noofCards = hand.size();
    }

    public int countSuit(Suit countSuit){
        int counter = 0;

        for(int i = 0; i < hand.size(); i++){
            if(hand.get(i).getSuit().equals(countSuit)){
                counter++;
            }
            else{
                counter--;
            }

        }
        return counter;
    }

    public int countRank(Rank countRank){
```

```
62          int count = 0;

64          for(Card c : hand){
                if(c.getRank() == countRank){
66                  count++;
                }
68          }
            return count;
70      }

72      public void addSingleCard(Card aCard){
            //add a single har to this hand
74          hand.add(aCard);
            noofCards = hand.size();
76      }

78      public void addCardCollection(Card[] arrayOfCards){
            //add array of cards to hand
80          hand.addAll(Arrays.asList(arrayOfCards));
            noofCards = hand.size();
82      }

84      public void addHand(Hand aHand){
            //add hand to this hand
86          ArrayList<Card> temp = aHand.hand;
            hand.addAll(temp);
88          noofCards = hand.size();
        }
90
        public boolean removeSingleCard(Card removeCard){
92          //check hand if empty
            if(!hand.isEmpty()){
94              for(int i = 0; i<hand.size(); i++){
                    //remove if matches the card in the hand
96                  if(hand.get(i).equals(removeCard)){
                        hand.remove(i);
98                  }
                }
100             return true;
            }else{
102             //return false if it does not match
                return false;
104         }
        }
106
        public boolean removeAllCards(Hand collectionOfCards){
108         boolean remove = false;

110         if(!hand.isEmpty() && collectionOfCards.hand.size() <= hand.size()){
                for(Card myCard : hand){
112                 //check if all passed cards are in this hand
                    for(Card card1 : hand){
114                     if(card1.equals(myCard)){
                            remove = true;
116                     }else{
                            return false;
118                     }
                }
120             }
            }else{
122             remove = false;
            }
124         if(remove == true){
```

```
            for(int i = 0; i<hand.size(); i++){
                for(int j = 0; j < collectionOfCards.hand.size(); j++){
                    hand.remove(i);
                }
            }
        }
        return remove;
    }

    public Card remCardAtPostion(int card){
        Card aCard = null;

        //if in a valid position
        if(hand.size() >= card){
            //set card to aCard value
            aCard = hand.get(card);
            //remove it from the hand
            hand.remove(hand.get(card));
            //remove no of cards
            this.noofCards--;

        }
        return aCard;

    }

    @Override
    public Iterator<Card> iterator() {
        //return list
        return hand.listIterator();
    }

    public void sortAscending(){
        Collections.sort(hand);
    }

    public void sortRank(){
        Collections.sort(hand, Card.CompareRank());
    }

    public ArrayList<Card> getHand(){
        return this.hand;
    }


    public boolean hasSuit(Suit suitInHand){
        for(Card c : hand){
            if(c.getSuit() == suitInHand){
                return true;
            }
        }
        return false;
    }

    @Override
    public String toString(){
        //toString method
        StringBuilder sb = new StringBuilder();
        for(int i = 0; i< hand.size(); i++){
            sb.append(hand.get(i)).append("\n");
        }

        return sb.toString();
```

```
188     }

190     /**
         * Method to save serialization object
192      * @param fileName
         * @throws IOException
194      * @throws ClassNotFoundException
         */
196      public void saveFile(String fileName) throws IOException,
            ClassNotFoundException{
         FileOutputStream in = new FileOutputStream(fileName);
198      try(ObjectOutputStream out = new ObjectOutputStream(in)){
             out.writeObject(this);
200      }
    }
202
       /**
204     * Serialization method to load object
        * @param file
206     * @return
        * @throws FileNotFoundException
208     * @throws IOException
        * @throws ClassNotFoundException
210     */
        public Hand load(String file) throws FileNotFoundException, IOException,
            ClassNotFoundException{
212     Hand h = null;
        try{
214         FileInputStream in = new FileInputStream(file);
            ObjectInputStream fin = new ObjectInputStream(in);
216         h = (Hand)fin.readObject();
        }catch(IOException e){
218         e.printStackTrace();
        }
220     return h;
    }
222

224  public static void main(String[] args){

226      //Hand test

228      //add cards - create new hand
         Hand hand = new Hand();
230      Card card = new Card(Rank.ACE,Suit.SPADES);
         hand.addSingleCard(new Card(Rank.EIGHT,Suit.CLOVERS));
232      Hand hand2 = new Hand();
         hand2.addHand(hand);
234
         System.out.println(hand.toString());
236      System.out.println(hand.countRank(Rank.FOUR));
         System.out.println(hand.countSuit(Suit.CLOVERS));
238      System.out.println(hand.noofCards);

240      //hasSuit
         System.out.println(hand.hasSuit(Suit.DIAMONDS));
242      hand.sortAscending();

244      System.out.println("Sort by Rank: ");
         hand.sortRank();
246      System.out.println("Sort by Suit: ");

248      //remove card
```

```
            System.out.println(hand.remCardAtPostion(0));
250         System.out.println(hand.removeSingleCard(card));

252         //print hand
            System.out.println(hand);
254


256


258     }

260 }
```

Trick.java

```
package whist;
import cards.Card;
import cards.Card.Suit;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.Collections;


/**
 * Skeleton class for storing information about whist tricks
 * @author ajb
 */
public class Trick{
   public static Suit trumps;
   private Card[] trick;
   private Card leadCard;
   private int leadPlayer;
   private Card topCard;

   public Trick(int p){
   leadPlayer = p;
   trick = new Card[BasicGame.NOS_PLAYERS];
   this.topCard = topCard;
   }    //p is the lead player

   public static void setTrumps(Suit s){
       trumps=s;
   }

   public Suit getTrumps(){
       return trumps;
   }

   public Card getTopCard(){
       return topCard;
   }

/**
 *
 * @return the Suit of the lead card.
 */
    public Suit getLeadSuit(){
        return leadCard.getSuit();
    }

    public void setLeadCard(Card cards){
        leadCard = cards;
    }
/**
 * Records the Card c played by Player p for this trick
 * @param c
 * @param p
 */
    public void setCard(Card c,Player p){

        if(p.getID() == leadPlayer){
            leadCard = c;
        }
       topCard = trick[findWinner()];
    }
/**
```

```
62   * Returns the card played by player with id p for this trick
     * @param p
64   * @return
     */
66      public Card getCard(Player p){
            return trick[p.getID()];
68      }

70  /**
     * Finds the ID of the winner of a completed trick
72   */
        public int findWinner(){
74          ArrayList<Card> finalCards = new ArrayList();
            ArrayList<Card> trumpCard = new ArrayList();
76          Boolean isTrump = false;
            Card winCard = null;
78
            for(Card c : trick){
80              if(c != null){
                    if(c.getSuit() == getLeadSuit()){
82                      finalCards.add(c);
                    }
84                  if(c.getSuit() == trumps){
                        finalCards.add(c);
86                      isTrump = true;
                    }
88              }
            }
90
            if(isTrump){
92              for(Card c : finalCards){
                    if(c.getSuit() == trumps){
94                      trumpCard.add(c);
                    }
96              }
                Collections.sort(trumpCard);
98              winCard = trumpCard.get(0);
            }
100
            return Arrays.asList(trick).indexOf(winCard);
102     }

104     Card[] getCards() {
            throw new UnsupportedOperationException("Not supported yet."); //To
                change body of generated methods, choose Tools | Templates.
106     }
    }
```

BasicPlayer.java

```
/*
 * To change this license header, choose License Headers in Project Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
package whist;

import cards.Card;
import cards.Hand;
import java.util.Collections;

/**
 *
 * @author Nahim
 */
class BasicPlayer implements Player {

    private Hand h;
    private Strategy s;
    private Trick t;
    private int id;

    BasicPlayer(int id, Hand h, Strategy s) {
        this.h = h;
        this.s = s;
        this.id = id;
    }

    @Override
    public void dealCard(Card c) {
        h.addSingleCard(c);
    }

    @Override
    public void setStrategy(Strategy s) {
        this.s = s;
    }

    /**
     * players card played based on the trick t passed
     * @param t
     * @return
     */
    @Override
    public Card playCard(Trick t) {
        Card card = s.chooseCard(this.h, t);
        Collections.sort(h.getHand());
        System.out.println("Card played " + card);
        this.h.removeSingleCard(card);
        return card;
    }
    @Override
    public void viewTrick(Trick t) {
        s.updateData(t);
    }

    @Override
    public void setTrumps(Card.Suit s) {
       Trick.trumps = s;
    }

```

```
    @Override
63  public int getID() {
        return id;
65  }

67 }
```

BasicStrategy.java

```
/*
 * To change this license header, choose License Headers in Project Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
package whist;

import cards.Card;
import cards.Deck;
import cards.Hand;

/**
 *
 * @author Nahim
 */
public class BasicStrategy implements Strategy {

    int id;
    Trick t;
    Hand trumpCard = new Hand();
    Hand leadCards = new Hand();
    Hand noTrumpOrLeadCards = new Hand();

    public BasicStrategy(int id){
        this.id = id;
    }

    //discard lowest card
    public Card getLowestCard(){
        if(!leadCards.getHand().isEmpty()){
            leadCards.sortRank();
            return leadCards.getHand().get(leadCards.getHand().size()-1);
        }else if(!noTrumpOrLeadCards.getHand().isEmpty()){
            noTrumpOrLeadCards.sortRank();
            return noTrumpOrLeadCards.getHand().get(noTrumpOrLeadCards.getHand().
                size()-1);
        }else{
            trumpCard.sortRank();
            return trumpCard.getHand().get(trumpCard.getHand().size()-1);
        }
    }

    @Override
    public Card chooseCard(Hand h, Trick t) {
      //this trick equals to passed trick
      this.t = t;

      Boolean first = true;
      Boolean partnerTurn = false;
      Boolean partnerWinning = false;
      Boolean topTrump = false;
      int partner = 0;

      Card[] trick = t.getCards();

      for(Card card : trick){
          if(card != null){
              first = false;
          }
      }
```

```
61          //first player
            if(first){
63              h.sortRank();
                return h.getHand().get(0);
65          }

67          //check if cards in hand are trump or lead
            for(Card card : h.getHand()){
69              if(card.getSuit() == t.getLeadSuit()){
                    leadCards.addSingleCard(card);
71              }else if(card.getSuit() == t.trumps){
                    trumpCard.addSingleCard(card);
73              }else
                    noTrumpOrLeadCards.addSingleCard(card);
75          }

77          if(id == 0){
                partner = 2;
79          }
            if(id == 1){
81              partner = 3;
            }
83          if(id ==2){
                partner = 0;
85          }else{
                partner = 1;
87          }

89          //check if partner has had turn
            if(trick[partner] != null){
91              partnerTurn = true;
            }
93
            //if partner has not has thier turn
95          if(partnerTurn){
                if(trick[partner] == t.getTopCard()){
97                  partnerWinning = true;
                }
99          }

101         //determine top card in trick is a trump
            if(t.getTopCard().getSuit() == t.trumps && t.trumps != t.getLeadSuit()){
103             topTrump = true;
            }
105
            //if partner is winning, discard lowest card
107         if(partnerWinning){
                return getLowestCard();
109         }else if(leadCards.getHand().get(0).getRank().getCardRank() > t.getTopCard
                ().getRank().getCardRank() && topTrump == false){
              return leadCards.getHand().get(0);
111           //play trump card
            }else if(!trumpCard.getHand().isEmpty()){
113             trumpCard.sortRank();
                if(topTrump == false){
115                 return trumpCard.getHand().get(0);
                }//if top card is trump, play a card higher than it
117             if(topTrump == true){
                    if(trumpCard.getHand().get(0).getRank().getCardRank()> t.getTopCard
                        ().getRank().getCardRank()){
119                      return trumpCard.getHand().get(0);
                    }
121             }
```

```
        }

        return getLowestCard();

    }

    @Override
    public void updateData(Trick c) {
       t = c;
    }

    public static void main(String[] args){
        BasicStrategy s = new BasicStrategy(0);
        Deck deck = new Deck();
        Hand hand1 = new Hand();
        Trick t = new Trick(0);

        for(int i=0; i<13; i++){
            hand1.addSingleCard(deck.deal());
        }

        Trick.setTrumps(Card.Suit.DIAMONDS);
        System.out.print("Trump is Diamonds ");

        //create player 1
        BasicPlayer player1 = new BasicPlayer(0,hand1,s);

        hand1.sortAscending();

        //output player 1 hand
        System.out.println(hand1.toString());
        t.setCard(player1.playCard(t), player1);
    }


}
```

HumanStrategy.java

```
/*
 * To change this license header, choose License Headers in Project Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
package whist;

import cards.Card;
import cards.Hand;
import java.util.ArrayList;
import java.util.InputMismatchException;
import java.util.Scanner;

/**
 *
 * @author Nahim
 */
public class HumanStrategy implements Strategy {

    private Hand leadCards = new Hand();
    private Card chosenCard =null;
    private boolean isFirst = true;
    private int id;
    private Trick trick;
    private Hand hand;

    public HumanStrategy(int id){
        this.id = id;
    }

    @Override
    public Card chooseCard(Hand h, Trick t) {
        trick = t;
        hand = h;

        Scanner sc = new Scanner(System.in);
        boolean validCard = true;
        boolean chooseCard = true;
        boolean answer = true;
        chosenCard = null;
        isFirst = true;

        leadCards.getHand().clear();

        //check if first player
        for(Card c : t.getCards()){
            if(c != null){
                isFirst = false;
                break;
            }
        }

        if(!isFirst){
            //check if cards in hand are trump or lead
            for(Card c : h.getHand()){
                if(c.getSuit() == t.getLeadSuit()){
                    leadCards.addSingleCard(c);
                }
            }
        }
```

```
62          h.sortAscending();
            System.out.println("" + t.toString());
64          System.out.println("HAND ");
            System.out.println(PlayerHand());
66
            int handSize = hand.getHand().size()-1;
68          System.out.println("Pick a card " + "between 0" + "-" + handSize + " ");
            int acard = 0;
70          while(!validCard){
                try{
72                  acard = sc.nextInt();
                }catch(InputMismatchException e){
74                  acard = -1;
                     sc.next();
76              }
            }
78
            int card = acard;
80


82          //Card chosen less than 0 or bigger than hand size
            if(card < 0 || card >= hand.getHand().size()){
84              validCard = false;
            }
86
            while(!validCard){
88              System.out.println("Pick a valid card " + handSize);
                sc.next();
90          }
            card = sc.nextInt();
92
            //card is valid if between 0 and handsize
94          validCard = !(card < 0 || card >= hand.getHand().size());


96
            chosenCard = hand.getHand().get(card);
98


100             return chosenCard;
        }
102
        public String PlayerHand(){
104         StringBuilder st = new StringBuilder();
            ArrayList<Hand> hand = new ArrayList();
106         for(int i=0; i<hand.size();i++){
                st.append("(").append(i).append(")").append(hand.get(i).toString()).
                   append("\n");
108         }
            return st.toString();
110     }

112     @Override
        public void updateData(Trick c) {
114       trick = c;
        }
116
    }
```

AdvancedStrategy.java

File not found.

BasicWhist.java

```
1  package whist;
2
3  import cards.Deck;
4  import cards.Card.Suit;
5
6  /**
7   *
8   * @author ajb
9   */
10 public class BasicGame {
11     static final int NOS_PLAYERS=4;
12     static final int NOS_TRICKS=13;
13     static final int WINNING_POINTS=7;
14     int team1Points=0;
15     int team2Points=0;
16     Player[] players;
17     public BasicGame(Player[] pl){
18
19     }
20     public void dealHands(Deck newDeck){
21         for(int i=0;i<NOS_TRICKS;i++){
22             players[i%NOS_PLAYERS].dealCard(newDeck.deal());
23         }
24     }
25     public Trick playTrick(Player firstPlayer){
26         Trick t=new Trick(firstPlayer.getID());
27         int playerID=firstPlayer.getID();
28         for(int i=0;i<NOS_PLAYERS;i++){
29             int next=(playerID+i)%NOS_PLAYERS;
30             t.setCard(players[next].playCard(t),players[next]);
31         }
32         return t;
33     }
34     public void playGame(){
35         Deck d=new Deck();
36         dealHands(d);
37         int firstPlayer=(int)(NOS_PLAYERS*Math.random());
38         Suit trumps=Suit.getRandomSuit();
39         Trick.setTrumps(trumps);
40         for(int i=0;i<NOS_PLAYERS;i++)
41             players[i].setTrumps(trumps);
42
43         for(int i=0;i<NOS_TRICKS;i++){
44             Trick t=playTrick(players[firstPlayer]);
45             System.out.println("Trick ="+t);
46             firstPlayer=t.findWinner();
47             System.out.println("Winner ="+firstPlayer);
48
49
50         }
51     }
52 /**
53  * Method to find the winner of a trick.
54  * @param t: current trick
55  * @return the index of the winning player
56  */
57     public void playMatch(){
58         team1Points=0;
59         team2Points=0;
60         while(team1Points<WINNING_POINTS && team2Points<WINNING_POINTS){
61             playGame();
```

```
        }
63      if(team1Points>=WINNING_POINTS)
            System.out.println("Winning team is team1 1 with"+team1Points);
65      else
            System.out.println("Winning team is team2 1 with"+team2Points);
67
    }
69  public static void playTestGame(){
        Player[] p = new Player[NOS_PLAYERS];
71      for(int i=0;i<p.length;i++){
            p[i]=null;//CREATE YOUR PLAYERS HERE
73
        }
75      BasicGame bg=new BasicGame(p);
        bg.playMatch(); //Just plays a single match
77  }
    public static void main(String[] args) {
79      playTestGame();
    }
81
}
```

PlayerDescription.pdf

File not found.