CMP-5015Y Assignment 2 (Java)

100214063 (`uyx17kku`)

Wed, 5 Feb 2020 14:53

PDF prepared using PASS version 1.17 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 pgCW;
2
3  import java.io.*;
4  import java.util.ArrayList;
5  import java.util.Collections;
6  import java.util.Comparator;
7  import java.util.Random;
8
9
10 ////////////////LAMBDAS
11 public class Card implements Serializable, Comparable<Card> {
12     static final long serialVersionUID = 12345;
13     private static final String fName = "serialization.txt";
14     public final Rank rank;
15     public final Suit suit;
16
17     //creates card with with rank and suit parameters
18     public Card(Rank rank, Suit suit) {
19         this.rank = rank;
20         this.suit = suit;
21     }
22
23     //calculate rank difference between 2 cards based on index in the rank enum
24     public static int difference(Card card1, Card card2) {
25         return Math.abs(card1.rank.ordinal() - card2.rank.ordinal());
26     }
27
28     //actual difference in value of the card
29     public static int differenceValue(Card card1, Card card2) {
30         return Math.abs(card1.rank.getValue() - card2.rank.getValue());
31     }
32
33     private static Object readIn() throws IOException, ClassNotFoundException {
34         System.out.println("Reading ////////////////////");
35         ObjectInputStream a = new ObjectInputStream(new FileInputStream(new File(
               fName)));
36         return a.readObject();
37     }
38
39     private static void writeOut(Serializable obj) throws IOException {
40         ObjectOutputStream a = new ObjectOutputStream(new FileOutputStream(new
               File(fName)));
41         a.writeObject(obj);
42         a.close();
43     }
44
45     private static void selectTest(Card c) {
46         Card c1 = new Card(Rank.ACE, Suit.DIAMONDS);
47         Card c2 = new Card(Rank.TWO, Suit.CLUBS);
48         Card c3 = new Card(Rank.THREE, Suit.CLUBS);
49         Card c4 = new Card(Rank.THREE, Suit.HEARTS);
50         Card c5 = new Card(Rank.THREE, Suit.CLUBS);
51         Card c6 = new Card(Rank.TWO, Suit.HEARTS);
52         Card c7 = new Card(Rank.TWO, Suit.CLUBS);
53
54         ArrayList a = new ArrayList();
55         a.add(c1);
56         a.add(c2);
57         a.add(c3);
58         a.add(c4);
59         a.add(c5);
```

```
        a.add(c6);
        a.add(c7);
        a.add(c);
        System.out.println("\nOriginal list:");
        for (int i = 0; i < a.size(); i++) {
            System.out.println(a.get(i));
        }
        //for compareTO list descending
        System.out.println("\nComparison 1(compareTo):");
        for (int i = 0; i < a.size(); i++) {
            Comparator<Card> comp = Card::compareTo;
            Collections.sort(a, comp);
            System.out.println("changed version: " + a.get(i));
        }
        System.out.print("CompareAscending\n");
        for (int i = 0; i < a.size(); i++) {
            Comparator<Card> dsc = new compareAscending();
            Collections.sort(a, dsc);
            System.out.println("changed version: " + a.get(i));
        }
    }

    public static void main(String[] args) throws IOException,
        ClassNotFoundException {
        Card c8 = new Card(Rank.EIGHT, Suit.HEARTS);
        Card c9 = new Card(Rank.ACE, Suit.SPADES);
        System.out.println("Card Created:" + c8 + " and " + c9 + " (to show to
            string method)");
        System.out.println("Card difference value= " + Card.differenceValue(c8,
            c9));
        System.out.println("Card difference = " + Card.difference(c8, c9));
        selectTest(c8);
        System.out.print("Other Methods test on: " + c9);
        System.out.print("\ngetPrevious () =" + c8.getRank().getPrevious());
        System.out.print("\ngetValue() =" + c8.getRank().getValue());
        System.out.print("\ngetRank () =" + c8.getRank());
        System.out.print("\ngetSuit() =" + c8.getSuit() + "\n");
        writeOut(c8);
        System.out.println("Trying to serialize: " + c8);
        //De-serialization
        System.out.println("deserialized: " + readIn());
    }

    //get rank - Accessor
    public Rank getRank() {
        return this.rank;
    }

    //get suit - Accessor
    public Suit getSuit() {
        return this.suit;
    }

    public int compareTo(Card card) {
        int cardComp = card.rank.compareTo(this.rank);
        if (cardComp == 0) {
            cardComp = this.suit.compareTo(card.suit);
        }
        return cardComp;
    }

    public String toString() {

```

```
        return rank + " of " + suit;
    }

    //for rank - constructor
    public enum Rank {
        TWO(2), THREE(3), FOUR(4), FIVE(5), SIX(6), SEVEN(7), EIGHT(8), NINE(9),
           TEN(10), JACK(10), QUEEN(10), KING(10), ACE(11);
        final int value;

        Rank(int x) {
            value = x;
        }

        //get value of card (rank)
        public int getValue() {
            return value;
        }

        //returns the rank value before it
        public Rank getPrevious() {
            if (this.equals(Rank.TWO))
                return Rank.ACE;
            else
                return Rank.values()[this.ordinal() - 1];
        }
    }

    //for suit- constructor
    public enum Suit {
        CLUBS, DIAMONDS, HEARTS, SPADES;

        //random suit method
        public Suit randSuit() {

            return Suit.values()[new Random().nextInt(Suit.values().length)];
        }
    }

    //compares rank and put it in ascending order
    public static class compareAscending implements Comparator<Card> {
        public int compare(Card o1, Card o2) {
            return (o1.rank.ordinal() - o2.rank.ordinal());
        }
    }

    //compares suit based on order in the enum
    public static class compareSuit implements Comparator<Card> {
        public int compare(Card o1, Card o2) {
            return o2.compareTo(o1);
        }
    }
}
```

Deck.java

```
1  package pgCW;
2
3  import java.io.*;
4  import java.util.ArrayList;
5  import java.util.Iterator;
6  import java.util.Random;
7
8
9  public class Deck implements Serializable, Iterable {
10     static final long serialVersionUID = 12355;
11     private static final String fName = "deck.txt";
12     private final ArrayList<Card> deck = new ArrayList<>();
13
14     public Deck() {
15         for (Card.Suit suit : Card.Suit.values()) {
16             for (Card.Rank rank : Card.Rank.values()) { //for every suit in enum
                   for every rank,deck created
17                 deck.add(new Card(rank, suit)); //new card created
18             }
19         }
20     }
21
22     private static Object readIn() throws IOException, ClassNotFoundException {
23         System.out.println("serialising ////////////////////");
24         ObjectInputStream b = new ObjectInputStream(new FileInputStream(new File(
               fName)));
25         return b.readObject();
26     }
27
28     private static void writeOut(Serializable obj) throws IOException {
29         ObjectOutputStream b = new ObjectOutputStream(new FileOutputStream(new
               File(fName)));
30         b.writeObject(obj);
31         b.close();
32     }
33
34     public static void main(String[] args) throws IOException,
           ClassNotFoundException {
35         Deck a = new Deck();
36         System.out.println("Original Deck:" + a);
37         a.shuffle();
38         System.out.println("\nShuffled :\n" + a);
39         System.out.println("Size:" + a.size());
40         System.out.println("\nDeal: (top of shuffled deck)\n" + a.deal());
41         System.out.println("\nNew Deck: " + a.newDeck());
42         writeOut(a);
43         System.out.println("////////////////////");
44         //De-serialization
45         System.out.println("\ndeserialized: (missing first card of shuffled deck
               as it was dealt)" + readIn());
46         a.deck.clear();
47         Card c1 = new Card(Card.Rank.TEN, Card.Suit.DIAMONDS);
48         Card c2 = new Card(Card.Rank.TEN, Card.Suit.SPADES);
49         Card c3 = new Card(Card.Rank.TWO, Card.Suit.CLUBS);
50         Card c4 = new Card(Card.Rank.SIX, Card.Suit.HEARTS);
51         a.deck.add(c1);
52         a.deck.add(c2);
53         a.deck.add(c3);
54         a.deck.add(c4);
55         System.out.println("Deck created : \n" + a);
56         //System.out.println("\nIterated"+a.iterator()+"\n");
```

```
57  ////////////////////////////////////NOT WORKING
            writeOut(a);
59          //System.out.println("Trying to serialize: " + a);
            //De-serialization
61          System.out.println("deserialized: " + readIn());


63
        }
65
        public ArrayList<Card> getDeck() {
67          return this.deck;
        }
69
        //find size
71      public int size() {
            return deck.size();
73      }

75      //create new deck
        public Deck newDeck() {
77          return new Deck();
        }
79
        //iterator
81      public Iterator<Card> iterator() {
            return new OddEvenIterator(deck);
83      }

85      public void shuffle() {
            Random random = new Random();
87          random.nextInt();

89          for (int i = 0; i < deck.size(); i++) {
                int change = i + random.nextInt(deck.size() - i);
91              //Card changing index
                Card c = deck.get(i);
93              deck.set(i, deck.get(change)); //changing deck
                deck.set(change, c);
95          }
        }
97
        public Card deal() {
99          return deck.remove(0);
        }
101
        public String toString() {
103         StringBuilder s = new StringBuilder();

105         for (Card card : deck) {
                //str is appending card and newline
107             s.append(card).append("\n");
            }
109         return s.toString();
        }
111
        public static class OddEvenIterator implements Iterator<Card> {
113         int nextCard;
            int oddEven;
115         ArrayList<Card> deck;

117         public OddEvenIterator(ArrayList<Card> deck) {
                //next card at -2 so that when 2 is added it starts at index 0
119             this.deck = deck;
```

```
            this.oddEven = 0;
            this.nextCard = -2;
        }

        public boolean hasNext() {
            if (nextCard < deck.size() - 2)
                return true;
                //for even number
            else if (oddEven == 0) {
                //Change Values for odd numbers to be travesed
                nextCard = -1;
                oddEven = 1;
                return true;
            }
            return false;
        }

        //nextcard
        public Card next() {
            if (hasNext())
                return deck.get(nextCard += 2);
            return null;
        }
    }
}
```

Hand.java

```
package pgCW;

import java.io.IOException;
import java.io.Serializable;
import java.util.*;


public class Hand implements Serializable, Iterable {
    static final long serialVersionUID = 12365;
    Collection<Card> hand;
    private int[] countRank;
    private int[] suitArray;

    //creates new instance of card
    public Hand() {
        this.suitArray = new int[4]; // to tally how much of each suit
        this.countRank = new int[13]; //to tally how much of each rank
        this.hand = new ArrayList();
    }

    //card array to hand
    public Hand(Card[] cards) {
        for (Card a : cards) {
            hand.add(a);
        }
    }

    //new instance of Hand
    public Hand(Hand newH) {
        for (Object card : newH) {
            hand.add((Card) card);
        }
    }

    public static void main(String[] args) throws IOException,
       ClassNotFoundException {
        Hand a = new Hand();
        Hand b = new Hand();
        Card c1 = new Card(Card.Rank.TEN, Card.Suit.DIAMONDS);
        Card c2 = new Card(Card.Rank.TEN, Card.Suit.SPADES);
        Card c3 = new Card(Card.Rank.TWO, Card.Suit.CLUBS);
        Card c4 = new Card(Card.Rank.SIX, Card.Suit.HEARTS);
        Card c5 = new Card(Card.Rank.THREE, Card.Suit.SPADES);
        Card c6 = new Card(Card.Rank.FIVE, Card.Suit.CLUBS);
        Card c7 = new Card(Card.Rank.FIVE, Card.Suit.HEARTS);
        a.add(c1);
        a.add(c2);
        a.add(c3);
        a.add(c4);
        b.add(c6);
        b.add(c7);
        System.out.println("Original Hand:" + a);
        a.add(c5);
        System.out.println("\nAdding card to hand:\n" + a);
        System.out.println("\nhand size:" + a.handSize());
        System.out.println("\nRemoving 4th card:" + a.remove(4) + "\n Hand:\n" +
           a);
        a.add(b);
        System.out.println("\nAdding another hand to first hand\n" + a);
        a.remove(b);
        System.out.println("\nRemoving second hand\n" + a);
```

```
60          a.add(b);

62          System.out.println("\nCount Rank\n" + a.countRank);
            System.out.println("\nSuit Array\n" + a.suitArray);
64
        }
66
        //get Rank array
68      public int[] getRankArray() {
            return countRank;
70      }

72      //get Suit array
        public int[] getSuitArray() {
74          return suitArray;
        }
76
        //adding a single card
78      public void add(Card a) {
            countRank[a.getRank().getValue()]++; //adding to tally of value
80          suitArray[a.getSuit().ordinal()]++; // adding to tally of suit
            hand.add(a);
82      }

84      //adding a collection
        public void add(Collection<Card> cardCollection) {
86          for (Card card : cardCollection) {
                countRank[card.getRank().getValue()]++; //add to tally
88              suitArray[card.getSuit().ordinal()]++; //add to tally
                hand.add(card);
90          }
        }
92
        //adding a hand
94      public void add(Hand hand2) {
            for (Object card : hand2) {
96              Card a = (Card) card;
                countRank[((Card) card).getRank().getValue()]++; //add to tally
98              suitArray[((Card) card).getSuit().ordinal()]++;// add to tally
                hand.add(a);
100         }
        }
102
        //remove certain card
104     public boolean remove(Card certainCard) {
            for (Card card : hand) {
106             if (certainCard == card) {
                    countRank[card.getRank().getValue()]--; // remove card from tally
108                 suitArray[card.getSuit().ordinal()]--; //remove card from tally
                    hand.remove(card);
110                 return true;
                }
112         }
            return false;
114     }

116     public int handSize() {
            return hand.size();
118     }

120     //remove a certain hand
        public boolean remove(Hand certainHand) {
122         int removedCards = 0;
```

```
        for (Object card : certainHand) { //given certain hand remove those cards
124         Card certainCard = (Card) card;
            if (hand.contains(certainCard)) {
126             countRank[((Card) card).getRank().getValue()]--;//decrease tally
                suitArray[((Card) card).getSuit().ordinal()]--;//decrease tally
128             hand.remove(certainCard);
                removedCards++;
130         }
        }
132     //Only return true if all cards given were removed.
        return removedCards == certainHand.handSize();
134 }

136 //remove card from  a location in hand
    public Card remove(int rem) {
138     int place = 0; // place of card
        for (Card card : hand) {
140         if (place == rem) //loop through for when the position matches
            {
142             countRank[card.getRank().getValue()]--;//decrease tally
                suitArray[card.getSuit().ordinal()]--;//decrease tally
144             hand.remove(card); //remove the card in that location
                return card;
146         }
            place++; //look at next postition
148     }
        return null;
150 }

152 public boolean isFlush() {
        if (hand.isEmpty()) {
154         return false;
        } else {
156         //get the first suit and compare the rest to its suit
            Card card = (Card) hand.toArray()[0];
158         Card.Suit suit = card.getSuit(); //Get first suit
            for (int i = 1; i < hand.size(); i++) {
160             Card a = (Card) hand.toArray()[i];
                Card.Suit newSuit = a.getSuit();
162             if (suit != newSuit) //Compare first suit with other suit.
                    return false; //false if they dont match
164         }
            return true;
166     }
    }
168
    public boolean isStraight() {
170     Collections.sort((ArrayList<Card>) hand);
        //loops through all cards to check
172     for (int i = 0; i < hand.size(); i++) {
            //check if the cards are in consecutive order
174         Card card = (Card) hand.toArray()[i];
            if (card.getRank().getValue() != card.getRank().getValue())
176             return false;
        }
178     return true;
    }
180
    public Iterator<Card> iterator() {
182     Iterator<Card> b = new Iterator<Card>() {
            private int index = 0;
184
            //check if there is a next card
```

```
186             public boolean hasNext() {
                    return index < hand.size();
188             }

190             public Card next() {
                    if (hasNext()) {
192                     return (Card) hand.toArray()[index++];
                    }
194                 return null;
                }
196         };
            return b;
198     }

200     public String toString() {
            StringBuilder s = new StringBuilder();
202         for (Card card : hand) {
                //appends card and newline
204             s.append(card).append("\n");
            }
206         return s.toString();
        }
208
        //compares rank and put it in descending order
210     public static class compareDescending implements Comparator<Card> {
            public int compare(Card o1, Card o2) {
212             return (o2.rank.ordinal() - o1.rank.ordinal());
            }
214     }
    }
```

BasicStrategy.java

```
1  package pgCW;
2
3
4
5  /*
6   * You should implement a new class BasicStrategy that implements the Strategy
       interface provided.  Basic strategy should
7  1.  Never cheat
8  2.  Always play all the cards of the lowest value possible
9  */
10
11 public interface Strategy {
12
13     /**
14      *
15      * @param b    the bid the player has to follow.
16      * @param h       The players current hand
17      * @param cheat true if the Strategy has decided to cheat (by call to cheat()
            )
18      *
19      * @return a Bid with the cards to pass to the game and the Rank. This will
            be
20      * different to the rank of thecards if the player is cheating!
21      *
22      */ {
23         //Checking whether the player has cards that is equal to the current
24         //bid rank or one above
25         //If so false is returned if not the player will have to cheat
26         ////////////////////not working
27         Bid b = new Bid();
28         Hand h = new Hand();
29         h.countRank;
30         return !h.countRank[b.getRank().getValue() > 0 || h.countRank(b.getRank()
               .getPrevious().getValue()) > 0; return true;
31     }
32
33     {
34         return true;
35     }
36
37 getValue()
38
39     /**
40      * Decides on whether to cheat or not
41      *
42      * @param b the bid this player has to follow (i.e the
43      *          bid prior to this players turn.
44      * @param h The players current hand
45      * @return true if the player will cheat, false if not
46      */
47     boolean cheat(Bid b, Hand h);
48     //if the count of that value across all suits is more than 4 because you have
           some in your hand to check
49     if(b.getCount()>4-h.countRank(b.getRank().
50
51         Bid chooseBid(Bid b, Hand h, boolean cheat);))
52
53     /**
54      * @param b the current bid
55      * @return true if this player is going to call cheat  on the last play b
56      */
```

```
57
    boolean callCheat(Hand h, Bid b);
59  return false
}
```

BasicPlayer.java

File not found.

BasicCheat.java

File not found.

HumanStrategy.java

File not found.

ThinkerStrategy.java

File not found.

MyStrategy.java

File not found.

StrategyFactory.java

File not found.

Bid.java

```
package pgCW;


public class Bid {
    Hand h;
    Card.Rank r;

    public Bid() {
        h = new Hand();
        r = Card.Rank.ACE;
    }

    public Bid(Hand hand, Card.Rank bidRank) {
        h = hand;
        r = bidRank;
    }

    public Hand getHand() {
        return h;
    }

    public void setHand(Hand hand) {
        h = hand;
    }

    public int getCount() {
        return h.handSize();
    }

    public Card.Rank getRank() {
        return r;
    }

    public void setRank(Card.Rank rank) {
        r = rank;
    }

    public String toString() {
        return h.handSize() + " x " + r;
    }

}
```