CMP-5015Y Assignment 2 - Blackjack in Java

100135292 (yqm15fqu)

Wed, 7 Feb 2018 15:23

PDF prepared using PASS version 1.12 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 blackjackgame;
2
3  import java.io.*;
4  import java.util.*;
5
6  public class Card implements Serializable, Comparable<Card> {
7
8      //Serialisation ID
9      static final long serialVersionUID = 112;
10
11     //final ensures each rank or suit value is defined
12     private final Suit suit;
13     private final Rank rank;
14
15     //Enum to set up the Rank for the cards
16     public enum Rank {
17         TWO(2), THREE(3), FOUR(4), FIVE(5), SIX(6), SEVEN(7), EIGHT(8), NINE(9),
18         TEN(10), JACK(10), QUEEN(10), KING(10), ACE(11);
19
20         final int cardValue;
21         // returns the values for each rank
22         Rank(int getValue) {
23             cardValue = getValue;
24         }
25         //returns the previous card
26         public Rank getPrevious() {
27             if (this.cardValue == 2) {
28                 return ACE;
29             } else {
30                 return values()[this.ordinal() - 1];
31             }
32         }
33         //returns the card value
34         public int getValue() {
35             return this.cardValue;
36         }
37     }
38     //enum to set up the suit for cards
39     enum Suit {
40         CLUBS(1), DIAMONDS(2), HEARTS(3), SPADES(4);
41
42         final int suitType;      //final ensures suitType is always a set value
43
44         Suit(int getValue) {
45             suitType = getValue;
46         }
47     }
48     //Constructor for a card
49     public Card(Rank rank, Suit suit) {
50         this.rank = rank;
51         this.suit = suit;
52     }
53     // returns the suit of a card
54     public Suit getSuit() {
55         return suit;
56     }
57     // returns the rank of a card
58     public Rank getRank() {
59         return rank;
60     }
61     // sums the value of 2 cards
```

```
    public static int sum(Card card1, Card card2) {
63      return card1.rank.getValue() + card2.rank.getValue();
    }
65  // boolean that returns true if the value of 2 cards is equal to 21
    public static boolean isBlackJack(Card card1, Card card2) {
67      if (sum(card1, card2) == 21 && (card1.rank == card1.rank.ACE
                || card2.rank == card2.rank.ACE)) {
69          return true;
        }
71      return false;
    }
73  // Class made to compare cards into ascending order
    public static class CompareAscending implements Comparator<Card> {
75
        //2 cards are compared, 1 is returned if card 1 is greater than card 2
77      @Override
        public int compare(Card card1, Card card2) {
79          if (card1.getRank().ordinal() > card2.getRank().ordinal()) {
                return 1;
81          } else if (card1.getRank().ordinal() == card2.getRank().ordinal()) {
                if (card1.getSuit().ordinal() < card2.getSuit().ordinal()) {
83                  return 1;
                } else {
85                  return -1;
                }
87          } else {
                return -1;
89          }
        }
91  }
    //class made to compare suits of cards
93  public static class CompareSuit implements Comparator<Card> {

95      //2 cards are compared, if card 1 is greater than card 2, -1 is returned
        @Override
97      public int compare(Card card1, Card card2) {
            if (card1.getSuit().ordinal() < card2.getSuit().ordinal()) {
99              return -1;
            } else if (card1.getSuit().ordinal() == card2.getSuit().ordinal()) {
101             if (card1.getRank().ordinal() < card2.getRank().ordinal()) {
                    return -1;
103             } else {
                    return 1;
105             }
            } else {
107             return 1;
            }
109     }
    }
111 //compareTo method that returns -1 if tha card passed is less than the card
    //being called on.
113 @Override
    public int compareTo(Card card) {
115     if (this.rank.getValue() > card.getRank().getValue()) {
            return -1;
117     } else if (this.getRank().getValue()
                < card.getRank().getValue()) {
119         return 1;
        }
121     return 0;
    }
123
    //toString that prints the name of a card e.g TWO OF HEARTS and returns
```

```
125     //it as a string.
        @Override
127     public String toString() {
            String cardInfo = this.getRank() + " OF " + this.getSuit();
129
            return cardInfo;
131     }

133     //--------------------TESTING, REMOVE LATER------------------------//
        public static void main(String[] args) {
135
            //creating cards giving rank and suit
137         Card tenDiamond = new Card(Rank.TEN, Suit.DIAMONDS);
            Card tenSpades = new Card(Rank.TEN, Suit.SPADES);
139         Card twoClubs = new Card(Rank.TWO, Suit.CLUBS);
            Card sixHearts = new Card(Rank.SIX, Suit.HEARTS);
141         Card aceSpades = new Card(Rank.ACE, Suit.SPADES);
            Card aceHearts = new Card(Rank.ACE, Suit.HEARTS);
143
            //ArrayList of card objects
145         ArrayList<Card> deck = new ArrayList();

147         //adding cards to the card arraylist
            deck.add(tenDiamond);
149         deck.add(tenSpades);
            deck.add(twoClubs);
151         deck.add(sixHearts);

153         // testing toString
            System.out.println("Cards added to an ArrayList using"
155                 + " the deck constructor: \t" + deck.toString());
            //testing sum
157         System.out.println("Sum being tested: " + sum(aceHearts, tenSpades));
            //testing isBlackJack
159         System.out.println("isBlackJack being"
                    + " tested: " + isBlackJack(aceHearts, tenSpades));
161         //testing compareTo
            System.out.println(tenDiamond.compareTo(tenSpades));
163
            //   CompareAscending x = new CompareAscending();
165         //  System.out.println(x.compare(tenDiamond, tenSpades));
        }
167
}
```

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 blackjackgame;

import java.util.*;
import java.io.*;
import blackjackgame.Card.*;
import java.util.logging.Level;
import java.util.logging.Logger;

/**
 *
 * @author Robert
 */
public class Deck implements Serializable, Iterable {
     static final long serialVersionUID = 111;

    //creating a static ArrayList of Cards.
    public static ArrayList<Card> Deck;

    //Creating a new deck and populating the Deck with all possible suits/cards.
    public Deck() {
        Deck = new ArrayList<Card>();
        for (Suit suit : Suit.values()) {
            for (Rank rank : Rank.values()) {
                Deck.add(new Card(rank, suit));
            }
        }
    }
    //method for shuffling the deck
    public void shuffleDeck() {
        int deckSize = Deck.size();
        //random seed generated
        Random random = new Random();

        //swap the positions of the generated seed index and the current card
        for (int i = 0; i < deckSize; i++) {
            Card curCard = Deck.get(i);
            int randIndex = i + random.nextInt(deckSize - i);
            Deck.set(i, Deck.get(randIndex));
            Deck.set(randIndex, curCard);

        }
    }
    //remove the top card of the shuffled deck and return it.
    public Card deal() {
        return Deck.remove(0);
    }

    //return the size of the deck currently
    public int size() {
        return Deck.size();
    }
    //clear the current deck and re-populate it.
    public void newDeck() {
        Deck.clear();
        for (Suit suit : Suit.values()) {
            for (Rank rank : Rank.values()) {
```

```
62                  Deck.add(new Card(rank, suit));
                }
64          }
        }
66      // iterator created.
        @Override
68      public Iterator iterator() {
            return new SecondCardIterator();
70      }
        //secondCardIterator class made that is serializable
72      private static class SecondCardIterator implements Iterator<Card>,
                Serializable {
74
            int position = 0;
76          //generate a null card
            Card card = null;
78          //check there's a next card and return true if there is
            @Override
80          public boolean hasNext() {
                return position < Deck.size() - 1;
82          }
            //if true is returned from has next, go 2 cards ahead
84          @Override
            public Card next() {
86              if (hasNext() == true) {
                    Card otherCard = Deck.get(position);
88                  position += 2;
                    return otherCard;
90              } else {
                    return null;
92              }
            }
94      }
        // toString that returns each card in a deck.
96      @Override
        public String toString() {
98          String print = "";

100         if (Deck.size() == 0) {
                return "Deck contains no cards, did you drop it?";
102         } else {
                Deck.forEach((item) -> {
104                 System.out.println(item.getRank() + "\t OF " + item.getSuit());
                });
106         }
            return "";
108     }

110     public static void main(String[] args) {
            //test deck made;
112         Deck j = new Deck();
            //show deck before shuffle
114         System.out.println("Before Shuffle: \n" + j);
            //create a new secondCardIterator
116         SecondCardIterator example;
            example = new SecondCardIterator();
118
            //shuffle deck and print
120         j.shuffleDeck();
            System.out.println("After Shuffle: \n" + j.toString());
122
            // testing size, newDeck and deal methods
124         System.out.println(j.size());
```

```
            System.out.println(j.deal());
            System.out.println(j.deal());
            System.out.println(j.size());

            System.out.println(j.toString());

            j.newDeck();
            System.out.println(j.size());
            // testing size, newDeck and deal methods

          //  j.shuffleDeck();
            for (int i = 0; i < 10; i++) {
                System.out.println(example.next());
            }
            System.out.println("\n\n\n\n");
////////////////////////////////////////////////////////////////////////

            //               SERIALIZ(s)ATION read
            System.out.println("SERIALIZATION TEST");

            SecondCardIterator test = new SecondCardIterator();
            String filename = "SecondCardIteratorDeck.ser";
            j.newDeck();
            try {
                FileOutputStream fos = new FileOutputStream(filename);
                ObjectOutputStream out = new ObjectOutputStream(fos);
                while (test.hasNext() == true) {
                    out.writeObject(test.next());
                }
                out.close();
            } catch (Exception ex) {
                ex.printStackTrace();
            }
//                          SERIALIZ(s)ATION ouput

            System.out.println("\n\n\n\n\n\n Hello\n\n\n");

            //attempt to print the cards. works but error is always thrown
            //(No time left to fix)
            try {
                FileInputStream fis = new FileInputStream(filename);
                ObjectInputStream in = new ObjectInputStream(fis);
                Card readCard;
                int counter = 0;

                while((readCard = (Card) in.readObject()) != null || counter == 25){
                    System.out.println(readCard.toString());
                    counter++;
                }
                in.close();

                System.out.println(j);
            } catch (FileNotFoundException ex) {
                 Logger.getLogger(Deck.class.getName()).log(Level.SEVERE, null, ex);
             } catch (IOException ex) {
                 Logger.getLogger(Deck.class.getName()).log(Level.SEVERE, null, ex);
             } catch (ClassNotFoundException ex) {
                 Logger.getLogger(Deck.class.getName()).log(Level.SEVERE, null, ex);
             }

    }

}
```

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 blackjackgame;

import blackjackgame.Card.*;
import blackjackgame.Deck.*;
import java.io.*;

import java.util.*;

/**
 *
 * @author yqm15fqu
 */
public class Hand implements Serializable, Iterable {

    static final long serialVersionUID = 102;

    //create an array for later use
    private int[] rankSum = new int[13];
    private ArrayList<Integer> handValues = new ArrayList();
    private ArrayList<Card> hand;

    //create a new hand.
    public Hand() {
        this.hand = new ArrayList();
    }
    // create a hand list that takes a card array and adds it to the hand
    public Hand(Card[] hand) {
        this.hand = new ArrayList(Arrays.asList(hand));
    }
    // Create a new hand and take another hands cards. add them to the hand
    public Hand(Hand passedCards) {
        this.hand = new ArrayList(passedCards.hand);
    }
    //returns a hand and its cards
    public ArrayList<Card> getHand() {
        return this.hand;
    }

    //add a card to the hand.
    public void add(Card card) {
        this.hand.add(card);
    }
    // add a collection of cards to the hand
    public void add(Collection<Card> card) {
        this.hand.addAll(card);
    }
    // add a hand to the hand.
    public void add(Hand hand) {
        this.hand.addAll(hand.getHand());
    }
    // remove a card, count the ranks and the hand. return the removed card
    public boolean remove(Card card) {
        boolean cardHolder = this.hand.remove(card);
        this.countRank();
        this.handSum();
        return cardHolder;
```

```
    }
    // remove a hand by assigning it to a bool, remove all cards then count.
    public boolean remove(Hand hand) {
        boolean tempHand = this.hand.removeAll(hand.getHand());
        this.countRank();
        return tempHand;
    }
    //remove a card at a specific position
    public Card remove(int num) {
        Card removedCard = this.hand.remove(num);
        this.countRank();
        return removedCard;
    }
    //count the ranks that are currently in hand using a case and switch
    public void countRank() {
        //initialise rankSum so each position is 0,
        for (int i = 0; i < rankSum.length; i++) {
            rankSum[i] = 0;
        }
        for (Card hand1 : this.hand) {
            switch (hand1.getRank()) {
                case TWO:
                    this.rankSum[0]++;
                    System.out.println("TWO in Hand");
                    break;
                case THREE:
                    this.rankSum[1]++;
                    System.out.println("THREE in Hand");
                    break;
                case FOUR:
                    this.rankSum[2]++;
                    System.out.println("FOUR in Hand");
                    break;
                case FIVE:
                    this.rankSum[3]++;
                    System.out.println("FIVE in Hand");
                    break;
                case SIX:
                    this.rankSum[4]++;
                    System.out.println("SIX in Hnad");
                    break;
                case SEVEN:
                    this.rankSum[5]++;
                    System.out.println("SEVEN in Hand");
                    break;
                case EIGHT:
                    this.rankSum[6]++;
                    System.out.println("EIGHT in Hand");
                    break;
                case NINE:
                    this.rankSum[7]++;
                    System.out.println("NINE in Hand");
                    break;
                case TEN:
                    this.rankSum[8]++;
                    System.out.println("TEN in Hand");
                    break;
                case JACK:
                    this.rankSum[9]++;
                    System.out.println("JACK in Hand");
                    break;
                case QUEEN:
                    this.rankSum[10]++;
```

```
125                     System.out.println("QUEEN in Hand");
                        break;
127                 case KING:
                        this.rankSum[11]++;
129                     System.out.println("KING in Hand");
                        break;
131                 case ACE:
                        this.rankSum[12]++;
133                     System.out.println("ACE in Hand");
                        break;
135                 default:
                        break;
137             }

139         }
        }
141     //arrayList that sums all the cards in hand and returns all possible soft
        //and hard values.
143     public ArrayList<Integer> handSum() {
            handValues.clear();
145         int possibleSum = 0;
            int foundAces = 0;
147
            //Checks all cards to see if a value is 11, if so add foundAces by 1
149         for (int j = 0; j < this.hand.size(); j++) {
                possibleSum += this.hand.get(j).getRank().getValue();
151             if (this.hand.get(j).getRank().getValue() == 11) {
                    foundAces++;
153             }
            }
155         //add the highest possible hard value to hand.
            handValues.add(possibleSum);
157         for (int i = 1; i <= foundAces; i++) {
                //remove 10 for every ace found. add the new value to possibleSum
159             possibleSum -= 10;
                handValues.add(possibleSum);
161         }
            //return an arraylist of all possible hand values.
163         return handValues;
        }
165
        //creates a new iterator
167     @Override
        public Iterator iterator() {
169         return this.hand.iterator();
        }
171     // sorts the cards in ascending order
        public void sortAscending() {
173         Collections.sort(hand, new CompareAscending());
        }
175     //sorts the cards in descending order
        public void sortDescending() {
177         Collections.sort(hand);
        }
179     //counts the amount a suit in hand passed in as an argument.
        public int countSuit(Suit suit) {
181         int Suit = 0;
            for (Card x : hand) {
183             if (x.getSuit() == suit) {
                    Suit++;
185             }
            }
187         return Suit;
```

```
    }
189 //counts the amount of a rank in a hand passed in as an argument.
    public int countRank(Rank rank) {
191     int Rank = 0;
        for (Card x : hand) {
193         if (x.getRank() == rank) {
                Rank++;
195         }
        }
197     return Rank;
    }
199 //toString to show what hand contains.
    @Override
201 public String toString() {
        return "Hand contians: " + hand;
203 }
    //bool that returns true if the cards are greater than the passed value.
205 public boolean isOver(int x) {
        if (handValues.get(handValues.size() - 2) > x) {
207         return true;
        } else {
209         return false;
        }
211 }

213 //reverses the order of the current hand.
    public void reverseHand() {
215     Collections.reverse(this.hand);
    }
217

    public static void main(String[] args) {
219     //create a new hand
        Hand hand = new Hand();
221     //create cards
        Card card1 = new Card(Rank.FIVE, Suit.CLUBS);
223     Card card12 = new Card(Rank.TWO, Suit.CLUBS);
        Card card13 = new Card(Rank.ACE, Suit.CLUBS);
225     Card card14 = new Card(Rank.THREE, Suit.CLUBS);
        Card card2 = new Card(Rank.ACE, Suit.DIAMONDS);
227     Card card22 = new Card(Rank.TWO, Suit.DIAMONDS);

229     //add cards to hand
        hand.add(card1);
231     hand.add(card12);
        hand.add(card13);
233     hand.add(card14);
        hand.add(card2);
235     hand.add(card22);

237     //testing for countRank()
        hand.countRank();
239     //testing for handSum()
        hand.handSum();
241
        //testing for sortAscending()
243     hand.sortAscending();

245     //testing for toString
        System.out.println(hand.toString());
247
        //testing for isOver
249     System.out.println(hand.isOver(115));
```

```
251             //testing for reverseHand
                hand.reverseHand();
253
                //printing the hand
255             System.out.println(hand);

257         }

259 }
```

BlackjackTable.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 blackjackgame;

import blackjackgame.Card.*;
import java.util.*;

/**
 *
 * @author Robert
 */
public class BlackjackTable {

    public static void humanGame() {

        BlackjackTable table = new BlackjackTable();
        BlackjackDealer gameDealer = new BlackjackDealer();
        int playerNo = 0;
        int gameCycles = 0;
        Scanner scan = new Scanner(System.in);
        Scanner cont = new Scanner(System.in);

        System.out.println("\n\n -- Deck Shuffled --\n\n");
        gameDealer.dDeck.shuffleDeck();

        HumanPlayer player1 = new HumanPlayer();
        BasicPlayer player2 = new BasicPlayer();

        List<Player> humanExample = new ArrayList();

        humanExample.add(player1);
        humanExample.add(player2);

        gameDealer.assignPlayers(humanExample);

        System.out.println("Please enter the amount of rounds you would like"
                + " to play: ");

        gameCycles = scan.nextInt();

        while (gameCycles != 0 && gameCycles > 0) {

            System.out.println("\n\n -- Bets made --\n\n");
            for (Player player : humanExample) {
                player.makeBet();
            }
            System.out.println("\n\n -- Bets Taken --\n\n");
            gameDealer.takeBets();

            System.out.println("\n\n -- First Cards Delt --\n\n");
            gameDealer.dealFirstCards();

            System.out.println("\n\n -- Players Play --\n\n");

            for (Player player : humanExample) {
                System.out.println("Player" + playerNo);
                gameDealer.play(player);
                playerNo++;
```

```
            }
63
            System.out.println("\n\n -- dealer deals now --\n\n");
65          gameDealer.playDealer();

67          System.out.println("\n\n -- settle bets --\n\n");
            gameDealer.settleBets();
69          gameCycles--;

71          if (gameCycles == 0) {
                System.out.println("Would you like to continue? y/n");
73              String h = cont.nextLine();
                if (h.charAt(0) == 'y' || h.charAt(0) == 'Y') {
75                  System.out.println("Please enter the amount of rounds "
                            + "you would like to play: ");
77                  gameCycles = scan.nextInt();
                } else {
79                  System.out.println("Thanks for playing!");
                }
81
            }
83          playerNo = 0;
        }
85
    }
87
    public static void basicGame() {
89      BlackjackTable table = new BlackjackTable();
        BlackjackDealer gameDealer = new BlackjackDealer();
91      int playerNo = 0;
        int gameCycles = 0;
93      Scanner scan = new Scanner(System.in);
        Scanner cont = new Scanner(System.in);
95
        System.out.println("\n\n -- Deck Shuffled --\n\n");
97      gameDealer.dDeck.shuffleDeck();

99      BasicPlayer player1 = new BasicPlayer();
        BasicPlayer player2 = new BasicPlayer();
101     BasicPlayer player3 = new BasicPlayer();
        BasicPlayer player4 = new BasicPlayer();
103
        List<Player> example = new ArrayList();
105
        example.add(player1);
107     example.add(player2);
        example.add(player3);
109     example.add(player4);

111     gameDealer.assignPlayers(example);

113     System.out.println("Please enter the amount of rounds you would like"
                + " to play: ");
115     gameCycles = scan.nextInt();
        while (gameCycles != 0 && gameCycles > 0) {
117
            System.out.println("\n\n -- Bets made --\n\n");
119         for (Player player : example) {
                player.makeBet();
121         }
            System.out.println("\n\n -- Bets Taken --\n\n");
123         gameDealer.takeBets();
```

```
125             System.out.println("\n\n -- First Cards Delt --\n\n");
                gameDealer.dealFirstCards();
127
                System.out.println("\n\n -- Players Play --\n\n");
129
                for (Player player : example) {
131                 System.out.println("Player" + playerNo);
                    gameDealer.play(player);
133                 playerNo++;
                }
135
                System.out.println("\n\n -- dealer deals now --\n\n");
137             gameDealer.playDealer();

139             System.out.println("\n\n -- settle bets --\n\n");
                gameDealer.settleBets();
141             gameCycles--;

143             if (gameCycles == 0) {
                    System.out.println("Would you like to continue? y/n");
145                 String h = cont.nextLine();
                    if (h.charAt(0) == 'y' || h.charAt(0) == 'Y') {
147                     System.out.println("Please enter the amount of rounds "
                                + "you would like to play: ");
149                     gameCycles = scan.nextInt();
                    } else {
151                     System.out.println("Thanks for playing!");
                    }
153
                }
155             playerNo = 0;
            }
157     }

159     public static void intermediateGame() {

161         BlackjackTable table = new BlackjackTable();
            BlackjackDealer gameDealer = new BlackjackDealer();
163         int playerNo = 0;
            int gameCycles = 0;
165         Scanner scan = new Scanner(System.in);
            Scanner cont = new Scanner(System.in);
167
            System.out.println("\n\n -- Deck Shuffled --\n\n");
169         gameDealer.dDeck.shuffleDeck();

171         IntermediatePlayer player1 = new IntermediatePlayer();
            IntermediatePlayer player2 = new IntermediatePlayer();
173         IntermediatePlayer player3 = new IntermediatePlayer();
            IntermediatePlayer player4 = new IntermediatePlayer();
175
            List<Player> intermediateExample = new ArrayList();
177
            intermediateExample.add(player1);
179         intermediateExample.add(player2);
            intermediateExample.add(player3);
181         intermediateExample.add(player4);

183         gameDealer.assignPlayers(intermediateExample);

185         System.out.println("Please enter the amount of rounds you would like"
                    + " to play: ");
187
```

```
        gameCycles = scan.nextInt();

        while (gameCycles != 0 && gameCycles > 0) {

            System.out.println("\n\n -- Bets made --\n\n");
            for (Player player : intermediateExample) {
                player.makeBet();
            }
            System.out.println("\n\n -- Bets Taken --\n\n");
            gameDealer.takeBets();

            System.out.println("\n\n -- First Cards Delt --\n\n");
            gameDealer.dealFirstCards();
            System.out.println("DEALER CARD!!: "
                    + gameDealer.dHand.getHand().get(0));
            player1.viewDealerCard(gameDealer.dHand.getHand().get(0));
            player2.viewDealerCard(gameDealer.dHand.getHand().get(0));
            player3.viewDealerCard(gameDealer.dHand.getHand().get(0));
            player4.viewDealerCard(gameDealer.dHand.getHand().get(0));

            System.out.println("\n\n -- Players Play --\n\n");

            for (Player player : intermediateExample) {

                System.out.println("Player" + playerNo);
                gameDealer.play(player);
                playerNo++;
            }

            System.out.println("\n\n -- dealer deals now --\n\n");
            gameDealer.playDealer();

            System.out.println("\n\n -- settle bets --\n\n");
            gameDealer.settleBets();
            gameCycles--;

            if (gameCycles == 0) {
                System.out.println("Would you like to continue? y/n");
                String h = cont.nextLine();
                if (h.charAt(0) == 'y' || h.charAt(0) == 'Y') {
                    System.out.println("Please enter the amount of rounds "
                            + "you would like to play: ");
                    gameCycles = scan.nextInt();
                } else {
                    System.out.println("Thanks for playing!");
                }

            }
            playerNo = 0;
        }

    }

    public static void advancedGame() {

        BlackjackTable table = new BlackjackTable();
        BlackjackDealer gameDealer = new BlackjackDealer();

        int playerNo = 0;
        int gameCycles = 0;

        Scanner scan = new Scanner(System.in);
        Scanner cont = new Scanner(System.in);
```

```
251
            System.out.println("\n\n -- Deck Shuffled --\n\n");
253         gameDealer.dDeck.shuffleDeck();

255         //create players
            AdvancedPlayer player1       = new AdvancedPlayer();
257         BasicPlayer player2          = new BasicPlayer();
            HumanPlayer player3          = new HumanPlayer();
259         IntermediatePlayer player4   = new IntermediatePlayer();

261         List<Player> advancedExample = new ArrayList();
            //assign to array
263         advancedExample.add(player1);
            advancedExample.add(player2);
265         advancedExample.add(player3);
            advancedExample.add(player4);
267         //assign players
            gameDealer.assignPlayers(advancedExample);
269
            System.out.println("Please enter the amount of rounds you would like"
271                 + " to play: ");
            //how many games to play
273         gameCycles = scan.nextInt();

275         while (gameCycles != 0 && gameCycles > 0) {

277             System.out.println("\n\n -- Bets made --\n\n");
                for (Player player : advancedExample) {
279                 player.makeBet();
                }
281             System.out.println("\n\n -- Bets Taken --\n\n");
                gameDealer.takeBets();
283
                System.out.println("\n\n -- First Cards Delt --\n\n");
285             gameDealer.dealFirstCards();
                System.out.println("DEALER CARD!!: "
287                         + gameDealer.dHand.getHand().get(0));
                player1.viewDealerCard(gameDealer.dHand.getHand().get(0));
289             player2.viewDealerCard(gameDealer.dHand.getHand().get(0));
                player3.viewDealerCard(gameDealer.dHand.getHand().get(0));
291             player4.viewDealerCard(gameDealer.dHand.getHand().get(0));

293             System.out.println("\n\n -- Players Play --\n\n");
                for (Player player : advancedExample) {
295
                    System.out.println("Player" + playerNo);
297                 gameDealer.play(player);
                    playerNo++;
299             }

301             System.out.println("\n\n -- dealer deals now --\n\n");
                gameDealer.playDealer();
303             //count cards for advanced
                for (Player player : advancedExample) {
305                 player1.viewCards(player.getHand().getHand());
                }
307             player1.viewCards(gameDealer.dHand.getHand());


309
                System.out.println("\n\n -- settle bets --\n\n");
311             gameDealer.settleBets();
                gameCycles--;
313
```

```
            if (gameCycles == 0) {
                System.out.println("Would you like to continue? y/n");
                String h = cont.nextLine();
                if (h.charAt(0) == 'y' || h.charAt(0) == 'Y') {
                    System.out.println("Please enter the amount of rounds "
                            + "you would like to play: ");
                    gameCycles = scan.nextInt();
                } else {
                    System.out.println("Thanks for playing!");
                }

            }
            playerNo = 0;
        }

    }

    public static void main(String[] args) {

            basicGame();
//            humanGame();
//            intermediateGame();
//            advancedGame();

            ////////////////Didn't have time to finish saving or to be able to
            ////////////////save to text file. Sorry!!
    }
}
```

BlackjackDealer.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 blackjackgame;

import java.util.*;

/**
 *
 * @author yqm15fqu
 */
public class BlackjackDealer implements Dealer {

    //for assigning players to the dealer
    List<Player> currentPlayers = new ArrayList();
    //dealer hand
    Hand dHand = new Hand();
    //dealer deck
    Deck dDeck = new Deck();
    //for player positions
    int playerPosition = 0;
    //dealer hand total
    int dTotal = 0;
    //players bets
    int playerBets = 0;

    //adds players to the arraylist associated with the dealer
    @Override
    public void assignPlayers(List<blackjackgame.Player> p) {
        if (p.size() > 7) {
            System.out.println("There are too many playes for this game,"
                    + " a maximum of 8 plays only");
        } else if (p.size() <= 7) {
            currentPlayers.addAll(p);
        }
    }
    //takes the players bets for all the players
    @Override
    public void takeBets() {
        for (Player currentPlayer : currentPlayers) {
            playerBets += currentPlayer.getBet();
        }
        System.out.println("All bets recieved, as a total of Ãč" + playerBets
                + " from " + currentPlayers.size() + " players\n\n");
        playerBets = 0;
    }
    //checks theres enough cards for the round, if not it makes a new deck and
    //shuffles. then deals 2 cards to each player and one for the dealer.
    @Override
    public void dealFirstCards() {

        if ((dDeck.size() - (currentPlayers.size() * 2)) <= 13) {
            dDeck.newDeck();
            dDeck.shuffleDeck();

            for (Player currentPlayer : currentPlayers) {
                currentPlayer.newDeck();
            }
        }
```

```
63          dHand.remove(dHand);

65          for (Player currentPlayer : currentPlayers) {

67              currentPlayer.newHand();
                currentPlayer.takeCard(dDeck.deal());
69              currentPlayer.takeCard(dDeck.deal());

71              System.out.println("Hand size: "
                        + currentPlayer.getHandTotal() + "\n");
73          }
            dHand.add(dDeck.deal());
75      }

77      //makes each player call hit while true, then checks for blackjacks and bust
        @Override
79      public int play(blackjackgame.Player p) {
            while (p.hit() == true) {
81              p.takeCard(dDeck.deal());
            }
83          if (p.blackjack() == true) {
                System.out.println("Total: " + p.getHandTotal() + "\n");
85              return p.getHandTotal();
            }
87          if (p.isBust() == true) {
                System.out.println("Player is bust!\n");
89              return p.getHandTotal();
            }
91          System.out.println("Total: " + p.getHandTotal() + "\n");
            return p.getHandTotal();
93      }

95      //dealer draws cards until 17 or higher
        @Override
97      public int playDealer() {
            System.out.println(dHand);
99
            while (scoreHand(dHand) < 17) {
101             dHand.add(dDeck.deal());
                dTotal = scoreHand(dHand);
103             System.out.println(dHand);
            }
105         System.out.println(dTotal);
            return dTotal;
107     }

109     //checks the hand values for each player.
        @Override
111     public int scoreHand(Hand h) {
            ArrayList<Integer> x = h.handSum();
113         int hTotal = x.get(0);

115         int check = 1;
            if (hTotal > 21) {
117             for (int i = 1; i < x.size() - 1; i++) {
                    hTotal = x.get(check);
119                 if (hTotal >= 17 && hTotal <= 21) {
                        System.out.println(hTotal);
121                     return hTotal;
                    } else {
123                     check++;
                    }
```

```
125             }
                if (hTotal > 21) {
127                 System.out.println("Hand is bust");
                    return hTotal;
129             }

131         }

133         return hTotal;
        }
135
        //settles all bets for all players and pays amoutns out
137     @Override
        public void settleBets() {
139
            int playerTotal = 0;
141         int playerBet = 0;

143         for (Player currentPlayer : currentPlayers) {
                playerBet = currentPlayer.getBet();
145             playerTotal = currentPlayer.getHandTotal();

147             if (currentPlayer.blackjack() == true && dTotal != 21
                        && dHand.getHand().size() > 2) {
149                 currentPlayer.settleBet((playerBet * 2)
                            + playerBet);
151                 System.out.println("BlackJack for player" + playerPosition
                            + "! " + (((playerBet) * 2) + playerBet) + " won!");
153             } else if (dHand.getHand().size() == 2 && dTotal == 21
                        && currentPlayer.blackjack() == false) {
155                 currentPlayer.settleBet(-playerBet);
                    System.out.println("Player" + playerPosition + " loses "
157                         + playerBet);
                } else if (dHand.getHand().size() == 2 && dTotal == 21
159                     && currentPlayer.blackjack() == true) {
                    currentPlayer.settleBet(playerBet);
161                 System.out.println("Dealer & Player" + playerPosition + "Both "
                            + "have BlackJack! Player" + playerPosition + "'s bet"
163                         + " returned.");
                } else if (dTotal > 21 && playerTotal <= 21) {
165                 currentPlayer.settleBet(playerBet * 2);
                    System.out.println("Dealer is bust, player" + playerPosition
167                         + " wins " + (playerBet * 2));
                } else if (currentPlayer.isBust() == true) {
169                 currentPlayer.settleBet(-playerBet);
                    System.out.println("Player" + playerPosition + " is bust");
171             } else if (dTotal <= 21 && playerTotal < dTotal) {
                    currentPlayer.settleBet(-playerBet);
173                 System.out.println("Dealer has a total " + dTotal + ". Player"
                            + playerPosition + " has "
175                         + playerTotal + ". Dealer Wins");
                } else if (playerTotal > dTotal && playerTotal <= 21) {
177                 currentPlayer.settleBet(playerBet * 2);
                    System.out.println("Dealer has a total " + dTotal + ". Player"
179                         + playerPosition + " has " + playerTotal
                            + ". Player Wins " + (playerBet * 2));
181             } else if (playerTotal == dTotal) {
                    currentPlayer.settleBet(playerBet);
183                 System.out.println("Dealer has a total " + dTotal + ". Player"
                            + playerPosition + " has " + playerTotal + ". DRAW! "
185                         + "bets returned");
                }
187             playerPosition++;
```

```
        }
        //removes bust players
        for (int i = 0; i < currentPlayers.size(); i++) {
            if (currentPlayers.get(i).getBalance() <= 0) {
                currentPlayers.remove(i);
                System.out.println("Player has run out of money, OH NO!!!");
                System.out.println("Player forcefully "
                        + "removed from the table");
            }
        }
        //returns players current balances.
        for (Player currentPlayer1 : currentPlayers) {
            System.out.println("\n\nPlayer" + playerPosition + "'s balance: "
                    + currentPlayer1.getBalance());
            playerPosition++;
        }
        playerPosition = 0;
    }

}
```

BasicPlayer.java

```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 blackjackgame;

import java.util.*;

/**
 *
 * @author yqm15fqu
 */
public class BasicPlayer implements Player {

    Hand playerHand = new Hand();
    //cardcount for later
    int cardCount = 0;
    //Player balance
    int playerBalance = 200;
    //amount bet
    int playerBet = 0;

    //remove old hand, make a new one
    @Override
    public Hand newHand() {
        playerHand.remove(playerHand);
        return playerHand;
    }

    //make a bet
    @Override
    public int makeBet() {

        if (playerBalance - playerBet < 0 || playerBalance <= 0) {
            System.out.println("player doesn't have enough money");
            playerBet = 0;
            return playerBalance;
        }
        playerBet = 10;
        playerBalance -= playerBet;
        System.out.println("10 bet");
        return playerBet;
    }
    //return the bet
    @Override
    public int getBet() {
        return playerBet;
    }
    //return balance
    @Override
    public int getBalance() {
        return playerBalance;
    }
    //Hit is tested. if hand value is < 17 take a hit, if it's over 21 check
    //other possible values, if still over 21 bust. if lower value becaus of ACE
    // return lower value.
    @Override
    public boolean hit() {
        ArrayList<Integer> x = playerHand.handSum();
        int basicVal = x.get(0);
```

```
63          if (basicVal < 17) {
                return true;
65          }

67          if (x.get(0) > 21) {

69              for (int i = 1; i < x.size(); i++) {
                    basicVal = x.get(i);
71                  if (basicVal >= 17 && basicVal <= 21) {
                        System.out.println(basicVal + "Jo");
73                      return false;
                    } else if (basicVal < 17) {
75                      return true;
                    }
77              }
            }
79          if (basicVal > 21) {
                return false;
81          }

83          return false;
        }
85
        // add a card to the hand
87      @Override
        public void takeCard(Card c) {
89          System.out.println(c.toString() + " taken");
            playerHand.add(c);
91      }

93      //add or remove the balance that is won in the game
        @Override
95      public boolean settleBet(int p) {
            if (p >= 0) {
97              playerBalance += p;
                playerBet = 0;
99              return true;
            } else if (p < 0) {
101             playerBet = 0;
                return true;
103         }
            playerBet = 0;
105         return false;
        }
107
        //return the total of the hand. the highest value below or equal to 21 if
109     //possible
        @Override
111     public int getHandTotal() {
            ArrayList<Integer> x = playerHand.handSum();
113
            if (x.get(0) > 21) {
115             for (int i = 1; i < x.size(); i++) {
                    if (x.get(i) <= 21) {
117                     return x.get(i);
                    }
119             }
            }
121         return x.get(0);
        }
123
        //checks 2 cards are passed, if they are then calls the previous is black
```

```
125     //jack method and gets the return from it and returns true if true.
        @Override
127     public boolean blackjack() {
            List<Card> x = new ArrayList();
129         x = playerHand.getHand();

131         if (x.size() == 2) {
                Card card1 = x.get(0);
133             Card card2 = x.get(1);

135             if (Card.isBlackJack(card1, card2) == true) {
                    return true;
137             }
            }
139         return false;
        }
141
        //checks all possible values, if all are above 21 returns true, returns
143     //false if a possible value is below 22
        @Override
145     public boolean isBust() {
            int bustCount = 0;
147         List<Integer> x = playerHand.handSum();

149         for (int i = 0; i < x.size(); i++) {
                if (x.get(i) > 21) {
151                 bustCount++;
                }
153         }

155         if (bustCount == x.size()) {
                return true;
157         } else {
                return false;
159         }
        }
161
        //return the hand
163     @Override
        public Hand getHand() {
165         return this.playerHand;
        }
167     //view the dealers card
        @Override
169     public void viewDealerCard(Card c) {
            Card dealerCard = c;
171
        }
173     //tells players the deck has been shuffled, resets card count
        @Override
175     public void newDeck() {
            System.out.println("The Deck has been shuffled.");
177         cardCount = 0;
        }
179
        //records values for all cards that are played each round and updates card
181     //count accordingly. used for advanced player.
        @Override
183     public void viewCards(List<Card> cards) {
            int tenUp = 0, midRange = 0, sixDown = 0;
185
            for (Card card1 : cards) {
187             if (card1.getRank().getValue() >= 10) {
```

```
                tenUp++;
189             } else if (card1.getRank().getValue() >= 7
                        && card1.getRank().getValue() <= 9) {
191                 midRange++;
                } else if (card1.getRank().getValue() <= 6
193                         && card1.getRank().getValue() >= 1) {
                    sixDown++;
195             }
            }
197         cardCount = ((-tenUp) + sixDown);
        }
199
    }
```

HumanPlayer.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 blackjackgame;

import java.util.*;

/**
 *
 * @author yqm15fqu
 */
public class HumanPlayer extends BasicPlayer {

    Scanner scan = new Scanner(System.in);
    //lets the player decide if they want to hit or not
    @Override
    public boolean hit() {
        ArrayList<Integer> x = playerHand.handSum();
        int check = 0;

        for (int i = 0; i < x.size(); i++) {
            if (x.get(i) > 21) {
                check++;
            }
        }
        if (check == x.size()) {
            return false;
        }

        System.out.println("Your hand is: " + playerHand);
        System.out.println("Do you want to hit? y/n");
        String hitOrNo = scan.next();

        if ("y".equals(hitOrNo) || "Y".equals(hitOrNo)) {
            return true;
        }

        return false;
    }
    //lets the player choose if they
    @Override
    public int makeBet() {

        System.out.println("How much would you like to bet? (Your balance is: "
                + playerBalance + ") : ");

        playerBet = scan.nextInt();

        if (playerBet < 10 || playerBet > 500 || playerBet > playerBalance) {
            while (playerBet < 10 || playerBet > 500
                    || playerBet > playerBalance) {
                System.out.println("You can't bet more than you have."
                        + " Your balance is: " + playerBalance);
                playerBet = scan.nextInt();

            }
            playerBalance -= playerBet;
        }
        playerBalance -= playerBet;
```

```
62          return playerBet;
        }
64
    }
```

IntermediatePlayer.java

```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 blackjackgame;

import blackjackgame.Card.*;
import java.util.ArrayList;

/**
 *
 * @author yqm15fqu
 */
public class IntermediatePlayer extends BasicPlayer {

    Card dealerCard;

    //hit logic for advanced player. checks if the possible values are >1 to
    //determine if there is an ace in hand or not, then follows appropriate
    //logic
    @Override
    public boolean hit() {

        ArrayList<Integer> x = playerHand.handSum();
        ArrayList<Card> cards = playerHand.getHand();
        int basicVal = x.get(0);

        if (x.size() > 1) {
            System.out.println(x.get(x.size() - 1));
            basicVal = x.get(x.size() - 1);
            if (dealerCard.getRank().getValue() >= 7 && basicVal < 17) {
                if (basicVal == 9 || basicVal == 10 || basicVal >= 17) {
                    return false;
                }
                return true;
            } else if (dealerCard.getRank().getValue() <= 6 && basicVal < 12) {
                if (basicVal != 9 || basicVal != 10 || basicVal >= 12) {
                    return false;
                }
                return true;
            } else {
                return false;
            }
        }
        if (x.size() == 1) {
            if (dealerCard.getRank().getValue() >= 7 && basicVal < 17) {
                return true;
            } else if (dealerCard.getRank().getValue() <= 6 && basicVal < 12) {
                return true;
            } else {
                return false;
            }
        }
        System.out.println("Some how outside of loop conditions.");
        return false;
    }
    //so the advanced player can count cards
    @Override
    public void viewDealerCard(Card c) {
        dealerCard = c;
```

```
63          }
    }
```

AdvancedPlayer.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 blackjackgame;

import java.util.*;

/**
 *
 * @author yqm15fqu
 */
public class AdvancedPlayer extends BasicPlayer {

    Card dealerCard;


    @Override
    public int makeBet() {
        if(cardCount <= 1){
            System.out.println("10 bet");
            playerBet = 10;
            playerBalance -= playerBet;
        }
        if(cardCount > 1){
            System.out.println(10*cardCount + " bet");
            playerBet = (10*cardCount);
            playerBalance -= playerBet;
        }

        return playerBet;
    }

    @Override
    public boolean hit() {

        ArrayList<Integer> x = playerHand.handSum();
        ArrayList<Card> cards = playerHand.getHand();
        int basicVal = x.get(0);

        if (x.size() > 1) {
            System.out.println(x.get(x.size() - 1));
            basicVal = x.get(x.size() - 1);
            if (dealerCard.getRank().getValue() >= 7 && basicVal < 17) {
                if (basicVal == 9 || basicVal == 10 || basicVal >= 17) {
                    return false;
                }
                return true;
            } else if (dealerCard.getRank().getValue() <= 6 && basicVal < 12) {
                if (basicVal != 9 || basicVal != 10 || basicVal >= 12) {
                    return false;
                }
                return true;
            } else {
                return false;
            }
        }
        if (x.size() == 1) {
            if (dealerCard.getRank().getValue() >= 7 && basicVal < 17) {
                return true;
```

```
62              } else if (dealerCard.getRank().getValue() <= 6 && basicVal < 12) {
                    return true;
64              } else {
                    return false;
66              }
            }
68          System.out.println("Some how outside of loop conditions.");
            return false;
70      }

72      @Override
        public void viewDealerCard(Card c) {
74          dealerCard = c;
        }
76  }
```