CS 1331 Homework 5

Due Thursday, September 27th 08:00 PM

Introduction

In this assignment, you will make the underlying classes to play a game of Blackjack. This homework will focus on the implementation of Arrays and Class design.

Note that although we may provide skeletions of javadocs for some methods, you will be responsible for completing these and all javadocs.

Blackjack Rules:

(this is just a standard game of blackjack – you can skip this part if you already know how to play)

The goal in a game of Blackjack is to get as close to 21 as possible without going over. If another player is closer to 21 than you are, you lose. You start off with two cards. Typically one card is face down and only visible to yourself, while the remaining cards are publicly visible. Since we are making a single player game, the face up/face down configuration does not come into play.

Each card in blackjack is assigned a value as follows:

2	→	2
3	→	3
4	→	4
…		
10	→	10
Jack	→	10
Queen	→	10
King	→	10
Ace	→	**1 or 11** (depending on which gets you closer to 21)

You start off with 2 cards, which you sum to determine your score. After that, you have the option to repeatedly hit or stay, gaining another card each time, in attempt to get as close to 21 as possible. However, you must be careful because if you go over 21, you **bust** and lose the game.

Aces can represent either 1 or 11 depending on your needs:

5, Ace, 2	= 8 or 18, hit!
5, Ace, 2, 5	= 13... (since 23 would be a bust, the Ace defaults to a 1).

You can find more information about the game of Blackjack online, but do not worry about the rules of betting or tie breaking (usually dealer wins except when you have 21 with an Ace and “black Jack” such as the Jack of Spades)

Do not forget about javadocing. All methods in all .java files should be javadoc'd. **Please follow the javadoc guidelines outlined in Homework 3.**

To recap:

You must have **class javadocs**:

description, @author, and @version tags

You must have **method javadocs:**

description, @param, and @return tags

- `@author` For indicating who wrote the class
- `@version` Shows the version number for the class, and possibly the date
- `@param paramName` For explaining what a parameter means in a method or constructor. Note the parameter name is repeated in the javadoc
- `@return` For explaining what a method returns

5.0 Overview

You will be responsible for making a Card, DeckOfCards, and BlackjackPlayer class. We are going to also introduce the concept of proper testing during development, as you will be responsible for testing each piece of code as your write it. Depending on the success you have with these classes, we may choose to provide you with the actual User interface to be able to play an actual game of black jack against the computer.

We will provide several drivers to help you in the development and testing of your code, but please do keep in mind these do not represent the full suite of tests we will perform.

You may not use java.util.Arrays or any other class which will replicate any needed functionality for you, such as printing or sorting (although it wouldn't really help in this assignment).

5.1 Card.java

Your first class will be represent a single card from a deck of cards. It should have proper encapsulation, including getters.

1. Create two String instance variables
   - **String face** will represent the face value of the card as a String
   - **String suit** will represent the suit of a card
   - Remember to enforce proper encapsulation throughout the assignment
2. Create a constructor that takes in and assigns the face value of the card, followed by the suit of the card. It is important you enter the parameters in that order to ensure compliance with the test code
   - **public Card(String face, String suit)**
3. Create getters for the instance variables
   - **public String getFace**
   - **public String getSuit**
4. Create a **toString** method that returns a meaningful text representation of the card.
5. Although it should always be done, without mention, this is a reminded that you must javadoc all the methods in all your files.
6. Create an *optional* driver (main method) to test the Card class.

5.2 DeckOfCards.java

The deck of cards class will be responsible for creating an array that holds the deck of cards, and instantiating it with all the possible cards. A standard deck has 52 cards (when excluding jokers, as you do in Blackjack). We have provided two String arrays representing the possible face values and suits of the cards:

```
public static final String[] SUITS = {"Hearts", "Spades", "Clubs", "Diamonds"};
public static final String[] FACES = {"Ace", "2", "3", "4", "5", "6", "7",
        "8", "9", "10", "Jack", "Queen", "King"};
```

You must create every possible combination of these values to get a complete, 52 card deck. We will be testing your deck for completeness both before and after shuffling.

1. You will need **at least** the following instance variables:
   - An array of Cards: **Card[] deck**
   - An integer count of how many cards are left in the deck: **int cardsLeft**
   - A **Random** object to help in shuffling the cards. You should only instantiate this **once.**
   - Keep in mind you will also need someway to keep track of which cards have already been dealt, and which card should be dealt next.
2. You should have a parameterless constructor that:
   - Initializes the instance variables
   - Populates the deck with 52 card objects (4 suits x 13 faces)
     - You should construct these cards using values from the provided arrays.
   - After construction, the cards in the array should be ordered in a semi-logical manner
     - i.e. “Ace of Hearts”, “2 of Hearts”, “3 of Hearts”,... or “Ace of Hearts”, “Ace of Spades”,...
     - Instantiating them in this order will help ensure you do not skip any cards.
3. A shuffle method, **public void shuffle(),** that shuffles all the cards in the deck so they are no longer in order.
   - You must make sure you actually **swap** the individual Card objects in the array.
   - The deck must remain complete after shuffling – it must contain each of the 52 cards exactly once.
   - After each shuffle, all the cards should have been returned to the deck (and cardsLeft should be reset).
4. A deal method, **public Card deal()**, which returns the next card in the deck.
   - A card must only be dealt once between shuffles. Once it is dealt, it is no longer considered part of the deck.

- You should decrement cardsLeft after each card dealt.
- If there are no cards left in the deck, this method should **return null**
- Your program should never crash due to **NullPointerErrors**

5. A getter for **cardsLeft** that returns the number of cards remaining in the deck. You must follow java conventions in naming this method.
6. We have provided a static method, **public static verifyDeck(DeckOfCards cardDeck)** which will be used to verify that your deck is instantiated properly and contains all the cards, before and after shuffling
7. **public String toString()**
    - The toString method should *legibly* return the number of cards remaining in the deck, followed by all the remaining cards in the deck as a Sting.
    - You should not return cards which have been dealt since the last shuffle, as they are no longer considered as part of the deck.
    - You should not ever print something in a toString
8. **Driver**
    - We have provided some code that will test your DeckOfCards class. Please note, that this card may not cover all test cases. It is highly recommended that you write additional test cases.
    - This driver depends on the fact that you have used the same variable names and conventions outlined in this assignment.

5.3 BlackjackPlayer.java

Your blackjack player class will be responsible for representating a player in a game of Blackjack. It should be completely independent of the deck. A BlackjackPlayer has both a name, and a “hand” of cards. The object should be able to determine the all possible values of its “hand” (which cards it has), and evaluate whether the player has “**busted”** - all possible sum of card values total over 21.

We have provided a brief outline for how you might want to approach this. You must maintain the majority of the method names in order to conform with the provided test cases, but certain implementations are left onto you – such as how to evaluate multiple possible values of a hand.

1. Instance variables:
    - An array of Cards, **Card[] hand**
    - A **String playerName**, representing the name of the Player
    - Anything else you may need
2. A **Constructor** which takes in and sets [only] the name of the player
3. A **public void newHand(Card card1, Card card2)** method
    - Sets the players current hand equal to the two cards provided
    - You may assume no actions are ever taken on the BlackjackPlayer without first calling this method
    - You should not be able to add individual cards without calling this method first
4. A **public void addCard(Card card)** method which will be called if the user chooses to hit
    - This method will take a card in and add it to the array
    - You may want to change the way you store and access the array so that you do not have to worry about constantly resizing and copying the array
5. A **public void clearHand()** method
    - This should *remove* all the cards from the array.
6. **public static int evaluateCard(Card card)**
    - A static method that takes in a card object and returns an integer value corresponding to the face value of the card, as given in the introduction.
    - You must figure out how you wish to handle the case of Aces being able to represent themselves a 1 or 11. **You may modify this method header** if you feel it is necessary, as long as the next method, *handValues* still works as indicated.
    - Make sure to thoroughly document every method, especially changes from instructions.
    - The reason this method is in BlackjackPlayer rather than Card is because these values are specific to the game of Blackjack.
    - You should **keep this method static**, but you can change the return type (think about how changing the return type might help you with dealing with different ace values)
7. **public int[] handValues()** - return an array of the possible values that your hand could

represent.

  - This method returns an array of int values representing the all the possible evaluations of the player's hand. If there are no Aces, this will be an array of a single element representing the sum of the cards.
  - However, since every Ace can represent two values, you will have an additional element in the array for each Ace you have, due to the fact it could be used as either a 1 or 11.

8. **public boolean bust()** - true or false if the player has busted
   - This method will determine if the player has busted – the sum of their cards is over 21 – and return true if it has.
   - If there is *any* way for the player's total to be 21 and under, then this method should return false
9. **public int bestScore()** - returns the best score represented by the hand: greatest score <= 21
   - If the player has busted, then this method should return -1
10. **public String getPlayerName()** - returns the player's name
11. **public String toString()**
    - This method should return the Player's name, all the cards in the players hands as well as the possible values this could represent.

5.4 BlackjackPlayer Testing

You must add a main method to BlackjackPlayer.java which thoroughly tests all its features. I have provided a skeleton of the class to get you start, but make sure you complete all the test cases. You must also write your own test cases, as I have left out several scenarios you might need to consider. Passing these tests does not mean you will get an automatic A on this assignment.

5.5 Blackjack.java

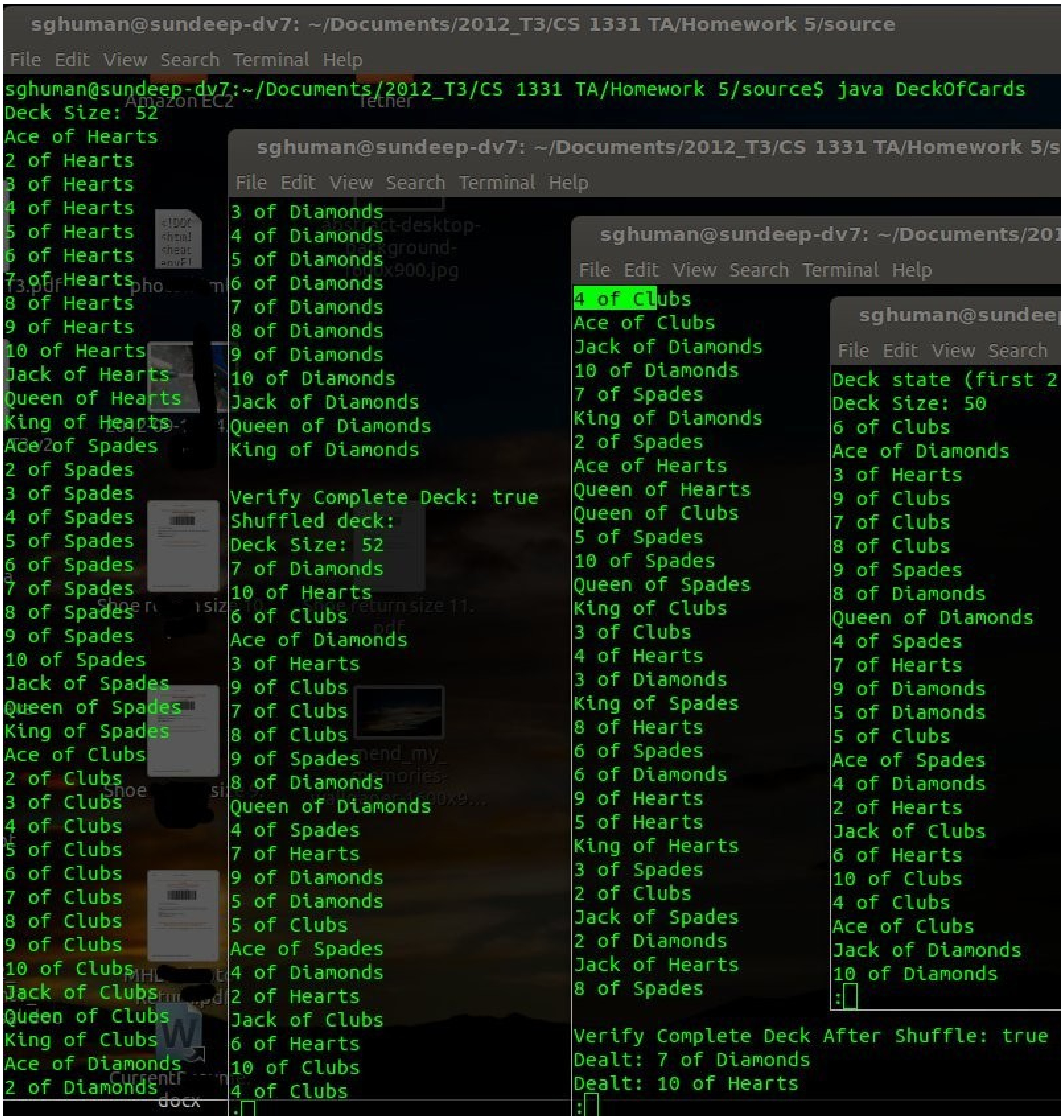
This class will implement BlackjackPlayer and run a simulation of a Blackjack game against a computer dealer.

Depending on the difficulty of the assignment, we may provide you with the class to actually play a game of Blackjack against the computer. It is very unlikely you will have to write this yourself, but for those of you who finish early and would like to challenge yourself go ahead and give it a try.

If you have properly written and tested your BlackjackPlayer class, it should work fine with any implementation we give you if we choose to follow that route.

There is some example output on the next page:

DeckOfCards:



```
sghuman@sundeep-dv7: ~/Documents/2012_T3/CS 1331 TA/Homework 5/source
sghuman@sundeep-dv7:~/Documents/2012_T3/CS 1331 TA/Homework 5/source$ java DeckOfCards
Deck Size: 52
Ace of Hearts
2 of Hearts
3 of Hearts
4 of Hearts
5 of Hearts
6 of Hearts
7 of Hearts
8 of Hearts
9 of Hearts
10 of Hearts
Jack of Hearts
Queen of Hearts
King of Hearts
Ace of Spades
2 of Spades
3 of Spades
4 of Spades
5 of Spades
6 of Spades
7 of Spades
8 of Spades
9 of Spades
10 of Spades
Jack of Spades
Queen of Spades
King of Spades
Ace of Clubs
2 of Clubs
3 of Clubs
4 of Clubs
5 of Clubs
6 of Clubs
7 of Clubs
8 of Clubs
9 of Clubs
10 of Clubs
Jack of Clubs
Queen of Clubs
King of Clubs
Ace of Diamonds
2 of Diamonds
```

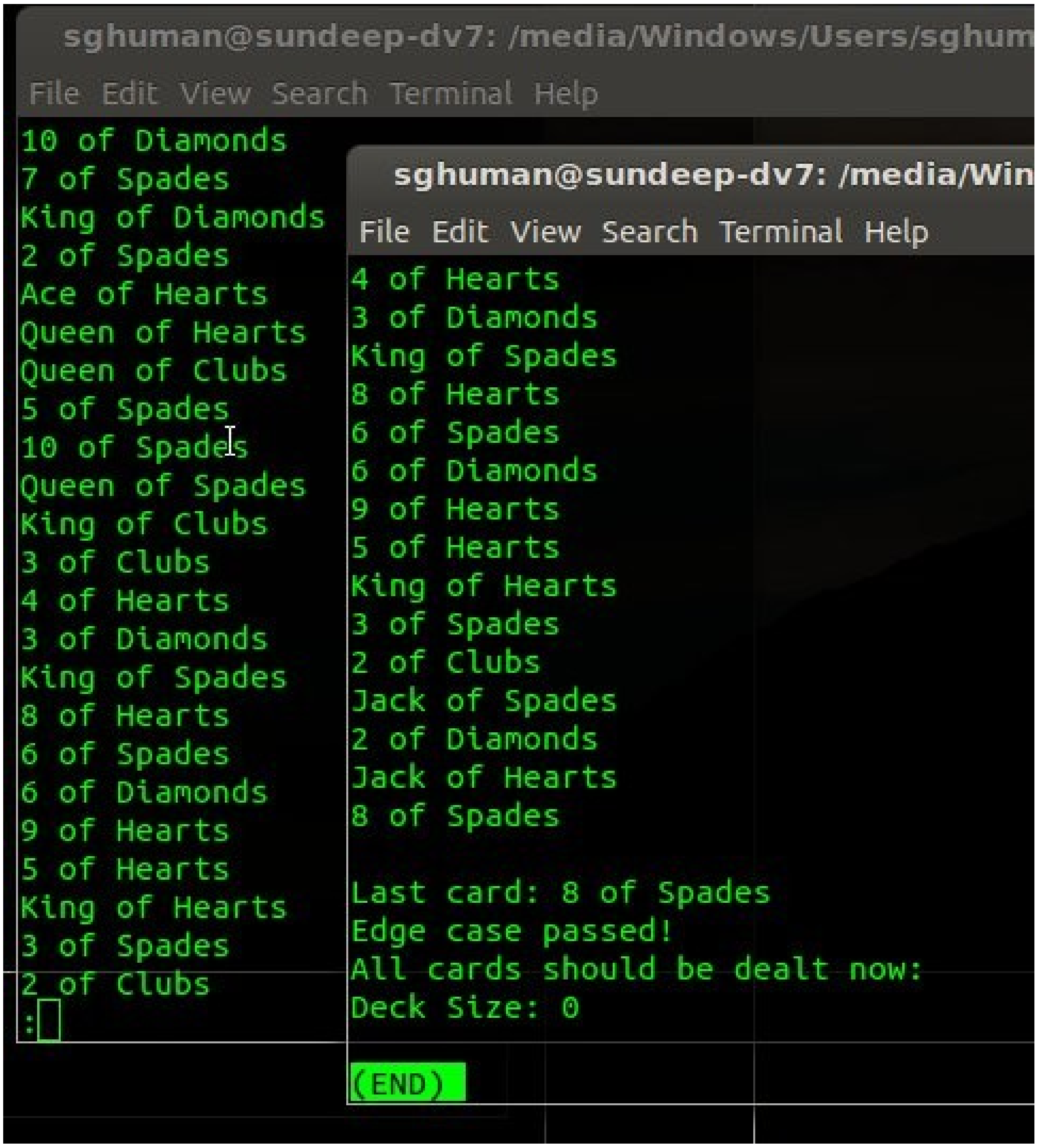
```
sghuman@sundeep-dv7: ~/Documents/2012_T3/CS 1331 TA/Homework 5/s
3 of Diamonds
4 of Diamonds
5 of Diamonds
6 of Diamonds
7 of Diamonds
8 of Diamonds
9 of Diamonds
10 of Diamonds
Jack of Diamonds
Queen of Diamonds
King of Diamonds

Verify Complete Deck: true
Shuffled deck:
Deck Size: 52
7 of Diamonds
10 of Hearts
6 of Clubs
Ace of Diamonds
3 of Hearts
9 of Clubs
7 of Clubs
8 of Clubs
9 of Spades
8 of Diamonds
Queen of Diamonds
4 of Spades
7 of Hearts
9 of Diamonds
5 of Diamonds
5 of Clubs
Ace of Spades
4 of Diamonds
2 of Hearts
Jack of Clubs
6 of Hearts
10 of Clubs
4 of Clubs
:
```

```
sghuman@sundeep-dv7: ~/Documents/201
4 of Clubs
Ace of Clubs
Jack of Diamonds
10 of Diamonds
7 of Spades
King of Diamonds
2 of Spades
Ace of Hearts
Queen of Hearts
Queen of Clubs
5 of Spades
10 of Spades
Queen of Spades
King of Clubs
3 of Clubs
4 of Hearts
3 of Diamonds
King of Spades
8 of Hearts
6 of Spades
6 of Diamonds
9 of Hearts
5 of Hearts
King of Hearts
3 of Spades
2 of Clubs
Jack of Spades
2 of Diamonds
Jack of Hearts
8 of Spades

Verify Complete Deck After Shuffle: true
Dealt: 7 of Diamonds
Dealt: 10 of Hearts
:
```

```
sghuman@sundee
Deck state (first 2
Deck Size: 50
6 of Clubs
Ace of Diamonds
3 of Hearts
9 of Clubs
7 of Clubs
8 of Clubs
9 of Spades
8 of Diamonds
Queen of Diamonds
4 of Spades
7 of Hearts
9 of Diamonds
5 of Diamonds
5 of Clubs
Ace of Spades
4 of Diamonds
2 of Hearts
Jack of Clubs
6 of Hearts
10 of Clubs
4 of Clubs
Ace of Clubs
Jack of Diamonds
10 of Diamonds
:
```


sghuman@sundeep-dv7: /media/Windows/Users/sghum
File Edit View Search Terminal Help

```
10 of Diamonds
7 of Spades
King of Diamonds
2 of Spades
Ace of Hearts
Queen of Hearts
Queen of Clubs
5 of Spades
10 of Spades
Queen of Spades
King of Clubs
3 of Clubs
4 of Hearts
3 of Diamonds
King of Spades
8 of Hearts
6 of Spades
6 of Diamonds
9 of Hearts
5 of Hearts
King of Hearts
3 of Spades
2 of Clubs
:
```

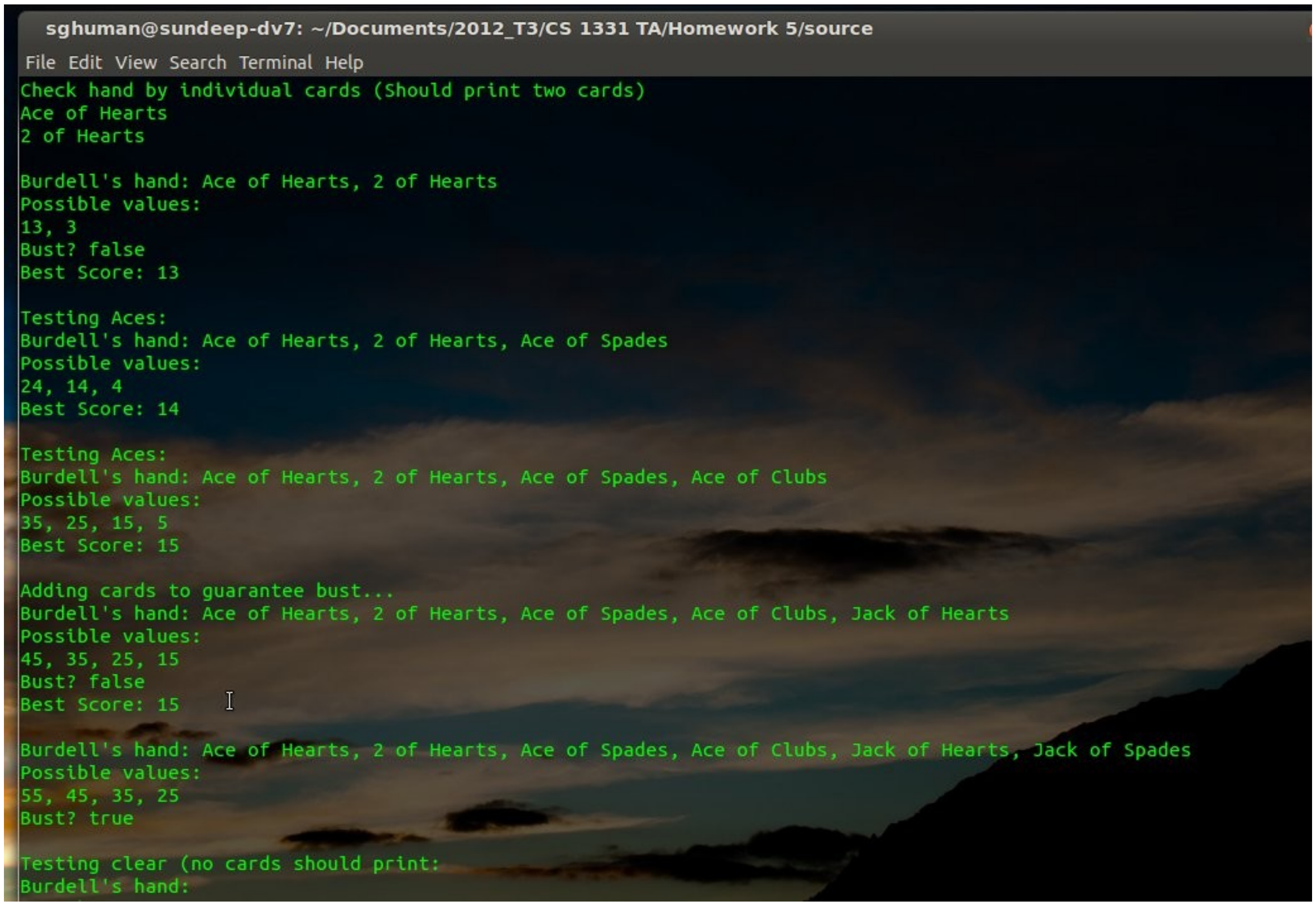
sghuman@sundeep-dv7: /media/Win
File Edit View Search Terminal Help

```
4 of Hearts
3 of Diamonds
King of Spades
8 of Hearts
6 of Spades
6 of Diamonds
9 of Hearts
5 of Hearts
King of Hearts
3 of Spades
2 of Clubs
Jack of Spades
2 of Diamonds
Jack of Hearts
8 of Spades

Last card: 8 of Spades
Edge case passed!
All cards should be dealt now:
Deck Size: 0

(END)
```

BlackjackPlayer:


sghuman@sundeep-dv7: ~/Documents/2012_T3/CS 1331 TA/Homework 5/source
File Edit View Search Terminal Help

```
Check hand by individual cards (Should print two cards)
Ace of Hearts
2 of Hearts

Burdell's hand: Ace of Hearts, 2 of Hearts
Possible values:
13, 3
Bust? false
Best Score: 13

Testing Aces:
Burdell's hand: Ace of Hearts, 2 of Hearts, Ace of Spades
Possible values:
24, 14, 4
Best Score: 14

Testing Aces:
Burdell's hand: Ace of Hearts, 2 of Hearts, Ace of Spades, Ace of Clubs
Possible values:
35, 25, 15, 5
Best Score: 15

Adding cards to guarantee bust...
Burdell's hand: Ace of Hearts, 2 of Hearts, Ace of Spades, Ace of Clubs, Jack of Hearts
Possible values:
45, 35, 25, 15
Bust? false
Best Score: 15

Burdell's hand: Ace of Hearts, 2 of Hearts, Ace of Spades, Ace of Clubs, Jack of Hearts, Jack of Spades
Possible values:
55, 45, 35, 25
Bust? true

Testing clear (no cards should print:
Burdell's hand:
```

Turn-in Procedure

Turn in the following files on T-Square. When you're ready, double-check that you have *submitted* and not just saved as draft.

- Card.java
- DeckOfCards.java
- BlackjackPlayer.java
- Any other files needed to run your program

All .java files should have a descriptive javadoc comment.

Don't forget your collaboration statement. You should include a statement with every homework you submit, even if you worked alone.

Verify the Success of Your HW Turn-In

Practice "safe submission"! Verify that your HW files were truly submitted correctly, the upload was successful, and that the files compile and run. It is solely your responsibility to turn in your homework and practice this safe submission safeguard.

7. After uploading the files to T-Square you should receive an email from T-Square listing the names of the files that were uploaded and received. If you do not get the confirmation email almost immediately, something is wrong with your HW submission and/or your email. Even receiving the email does not guarantee that you turned in exactly what you intended.
8. After submitting the files to T-Square, return to the Assignment menu option and this homework. It should show the submitted files.
9. Download copies of your submitted files from the T-Square Assignment page placing them in a new folder.
10. **Recompile and test those exact files.**
11. This helps guard against a few things.
    1. It helps insure that you turn in the correct files.
    2. It helps you realize if you omit a file or files.**
       (If you do discover that you omitted a file, submit all of your files again, not just the missing one.)
    3. Helps find last minute causes of files not compiling and/or running.

**Note: Missing files will not be given any credit, and non-compiling homework solutions will receive few to zero points. Also recall that late homework (past the grace period of 2 am) will not be accepted regardless of excuse. Treat the due date with respect. The real due date and time is 8 pm Thursday.