6.178 - Problem Set 1

Blackjack Utility Functions

You will complete three problem sets this IAP, each of which will compose part of a larger project: creating your own Blackjack game!

Blackjack is a card game with relatively simple rules. Each player starts with two cards in their hand and can choose to "hit" (add another card to their hand) or "stay" (stop adding cards and keep their current hand).
Now, in the standard casino style rules, the "dealer" also has their own hand and they follow slightly different rules which we'll mention later. All players play against the dealer. A player beats the dealer by having a higher total card value in their hand, but loses if their total card value exceeds 21.

Here's a link to more detailed rules (you won't be required to implement all of these rules in your game):

http://www.bicyclecards.com/how-to-play/blackjack/

For this assignment, you will implement various methods which may be useful to your game. You will be using the abstract data types *Card* and *Deck*. In BlackjackUtils.java, you will see some methods that allow you to interact with these data types and you should read their documentation to be sure you understand how to use them. You do NOT need to know or understand what is happening inside these methods, just use them as a black box. These methods are:

- `getRank(Card card)` – returns a representation of a card's rank
- `getSuit(Card card)` – returns a representation of a card's suit
- `drawCardFromDeck(Deck deck)` – draws and removes the top card from the deck
- `getBlackjackValue(Card card)` – returns the value of the card in Blackjack

  In Blackjack, all numeric cards (2-10) are worth the value printed on them. All face cards (King, Queen, Jack) are worth ten points. An ace can be worth either one point or eleven points. This methods will always return one for the value of an Ace.

Whenever you need to interact with Card and Deck, remember that you have these methods!

You will be implementing the methods found in `BlackjackUtils.java`. As you fill these in with your own code, remember that you can run the JUnit test suite in `BlackjackUtilsTest.java` at any time and see which tests you are passing. Use this to make sure your implementation is correct! **Do not change method signatures (the names of methods and their parameters), or your code will not pass the tests.**

A note about `Rank` and `Suit`

The provided methods `getRank` and `getSuit` have return types `Rank` and `Suit`, respectively. These types are enums as discussed in Lecture 2. You can compare enums using the == operator. For instance, if you wanted to check if a given `Card` cardA was a club, you could write:

```
if (getSuit(cardA) == Suit.CLUBS) {
    //cardA is a club
}else{
    //cardA is not a club
}
```

You can compare Ranks in a similar manner.

All possible values that Rank and Suit can take on can be found in the enum constant summary in their Javadocs, found at:

`http://web.mit.edu/abartow/www/cardAPIdocs/cards/Rank.html#enum.constant.summary`
`http://web.mit.edu/abartow/www/cardAPIdocs/cards/Suit.html#enum.constant.summary`

respectively.

(Note: We will be talking more about Javadocs later in the class, so do not expect to have to understand anything else on those pages, but if you want to know more, feel free to ask a TA or on Piazza.)

1 averageValue

`averageValue` takes in an array of integers (`int`) and computes the average. Take note that the average return type is a `double`!

Read the method's documentation for more information.

2 drawInitialHand

`drawInitialHand` takes in a `Deck` and simulates drawing an initial Blackjack hand. It returns an array of the two Cards drawn. These cards should be removed from the Deck.

HINT: Look at the provided function drawCardFromDeck(Deck deck)

Read the method's documentation for more information.

3 handAfterHit

`handAfterHit` takes in an array of Cards (`Card[]`) representing a hand and a `Deck`, and returns an array of cards representing what the hand would be after a player hits with that hand in Blackjack.

When a player in Blackjack hits, they draw the top card from a deck and append that card to the end of their current hand.

Read the method's documentation for more information.

4 valueOfBlackjackHand

`valueOfBlackjackHand` takes in an array of cards (`Card[]`) representing a hand in Blackjack and returns an `int` representing that hand's value according to the rules of Blackjack.

In Blackjack, all numeric cards (2-10) are worth the value printed on them. All face cards (King, Queen, Jack) are worth ten points. An Ace can be worth either eleven or one, whichever maximizes the value of the hand while keeping the value of the hand under twenty one. If the value of the hand must exceed twenty one, all aces should minimize the value of the hand (i.e. all aces should be worth one.).

Different aces can have different values in the same hand. (For instance, in a hand consisting of three aces, one of the aces is worth 11, while the other two are worth one, for a combined value for the hand of 13.)

The value of a hand is equal to the sum of the values of all cards in that hand.

HINT: Aces can have a value of either 1 or 11, so deal with non-Aces first, then deal with any Aces. Treat an Ace as 11 if it still keeps the total value at 21 or less. Otherwise, treat it as a 1.

Note that getBlackjackValue(Card card) will only treat Aces as 1's (see the documentation!) so you must use right values yourself.

Read the method's documentation for more information.

5 drawHandAsDealer

`drawHandAsDealer` takes in a `Deck` and simulates a dealer drawing their hand from that Deck, returning an array of cards (`Card[]`) representing the hand the dealer drew.

Dealers follow slightly different rules when they draw their hand: they cannot choose when to hit or stay. They are forced to continue drawing cards (i.e hitting) until the value of their hand is at least 17.

Read the method's documentation for more information.

6 compareHands

`compareHands` takes in two arrays of Cards (`Card[]`) representing hands in Blackjack. It returns a `boolean` that is true if the Blackjack score of the hand represented by the cards in the first argument is greater than or

equal to the Blackjack score of the hand represented by the cards in the second argument, and false otherwise.

Read the method's documentation for more information.

7 sameCard

sameCard takes in two `Card`s and returns a boolean that is true if the Suit and Rank of the first card is equal to the Suit and Rank of the second card, and false otherwise.

HINT: Look at the provided functions `getSuit` and `getRank`.

Read the method's documentation for more information.

8 containsCard

`containsCard` takes in an array of Cards (`Card[]`) as the first argument as well as a single desired `Card` as the second argument and returns a `boolean` that is true if the array of cards contains one or more cards that are the same (see `sameCard`) as the desired card, or false otherwise.

Read the method's documentation for more information.