6.178 - Problem Set 2

Card and Deck

Let's continue building our Blackjack game!

For this assignment, you will implement the abstract data types `Card` and `Deck` which you used as "black boxes" in problem set 1. You will be using a lot of knowledge learned during Lecture 4 so do not be alarmed if a lot of this looks unfamiliar to you. If you want to get a head start you can look at the lecture slides (already up on Stellar!) or look at the Java tutorials: `https://docs.oracle.com/javase/tutorial/java/javaOO/`

You will be implementing the methods found in `Card.java` and `Deck.java`. As you fill these in with your own code, remember that you can run the JUnit test suites in `CardTest.java` and `DeckTest.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.**

1 Card.java

1.1 Suit

Fill in `enum Suit` { } with the 4 suits found in a 52 card deck.

1.2 Rank

Fill in `enum Rank` { } with the 13 ranks found in a 52 card deck.

1.3 Card constructor

Fill in the constructor for `Card(Rank rank, Suit, suit)` { } to create a new Card.

HINT: You need some way of keeping track of the rank and suit.

1.4 getSuit

Fill in the method `getSuit()` { } to return the suit of the card.

This suit of the card should be of type `Suit` which you created in part 1.

Read the method's documentation for more information.

1.5 getRank

Fill in the method `getRank() { }` to return the rank of the card.

The rank of the card should be of type `Rank` which you created in part 2.

Read the method's documentation for more information.

1.6 toString

Fill in the method `toString() { }` to return a string representation of the `Card`.

A `Card` representation consists of its rank and suit. In order to pass the tests and get full credit, you must return the specific representation as detailed here:

- The representation starts with the rank and is followed by the suit.
- The rank is represented by either the number or the initial of the rank. Example a two is ”2” and an ace is a ”A”.
- The suit is represented by its initial, but appears in lower case unlike the rank. Example a diamond is a ”d”.

Read the method's documentation for more information.

1.7 Object Contract

Fill in `equals` and `hashCode` to fulfill the object contract that we learned about in class.

As a brief refresher:

- `equals(Object other) { }` should test the equality between this `Card` object and another object, `other`, which currently has a type of `Object`.
- Equality should always be reflexive, symmetric, transitive, and consistent.
- `hashCode() { }` should return an integer that describes the current `Card` object.
- The hashcode should always be the same for two `Card`s that are deemed equal by the `equals` method.

As always, read the method's documentation for more information.

2 Deck.java

2.1 Deck constructors

Fill in the two different constructors for `Deck`: one for a standard 52 card deck, one for a custom deck created from a list of `Card`s.

`Deck() { }` creates a deck for a standard 52 card deck in the following order: spades, diamonds, clubs, hearts each from ace to king.

`Deck(List<Card> cards) { }` creates a deck from a given `List` of `Card`s. The constructor should not modify what cards are in the deck.

Read the method's documentation for more information.

2.2 draw

Fill in the method `draw() { }` which is very similar to `drawCardFromDeck(Deck deck)`. The difference is that the `draw` method is called from a `Deck` object whereas the `drawCardFromDeck` method uses a `Deck` as a parameter.

This `draw` method should return the top `Card` of the `Deck` and remove it from the `Deck`.

Read the method's documentation for more information.

2.3 shuffle

Fill in the method `shuffle() { }` to shuffle and reorder all the cards in the `Deck`.

This method should not return anything. Instead it should change the current `Deck`.

Read the method's documentation for more information.

2.4 getCards

Fill in the method `getCards() { }` to get all the cards in the `Deck`, ordered in the same way as in the `Deck` itself.

You should try not to return any fields of the `Deck` directly to protect against outsiders changing the `Deck`.

Read the method's documentation for more information.

2.5 mostCommonRanks

Fill in the method `mostCommonRanks() { }` to find what the most common ranks in the `Deck` are. This should return *all* the ranks that occur the most often. If two ranks both occur 3 times in the `Deck` and all other ranks occur only 1 time, then both ranks should be returned. If all ranks occur exactly the same amount of times (for example 1 time), then all ranks should be returned.

Read the method's documentation for more information.

2.6 suitFrequency

Fill in the method `suitFrequency() { }` to report how many times each suit appears in the `Deck`. The return type is `Map<Suit, Integer>` which means that we want you to connect a `Suit` to an `Integer` in a data structure.

The `Suit` should be the same as the one you implemented earlier in `Card`. All 4 suits should appear in the returned `Map` even if there are 0 `Card`s with that `Suit`.

Read the method's documentation for more information.

2.7 toString

Fill in the `toString() { }` method to return a string representation of a `Deck`.

A `Deck` representation consists of the string representations of all the `Card`s in the `Deck`. As in `Card`, we want a specific representation which is as follows:

- The representation starts with a "[" character.
- This is followed by all the `Card`s. Each `Card` has its own string representation and should be separated by commas.
- The last character should be a "]" character.

Read the method's documentation for more information.

2.8 Object Contract

Fill in `equals` and `hashCode` to fulfill the object contract that we learned about in class.

For two decks to be equal, they must have the same cards in the same order.

As a brief refresher:

- `equals(Object other) { }` should test the equality between this `Deck` object and another object, `other`, which currently has a type of `Object`.
- Equality should always be reflexive, symmetric, transitive, and consistent.
- `hashCode() { }` should return an integer that describes the current `Deck` object.
- The hashcode should always be the same for two `Deck`s that are deemed equal by the `equals` method.

As always, read the method's documentation for more information.