Computer Science 2

Lab 01

Appalachian State University

Updated: 04/07/2023

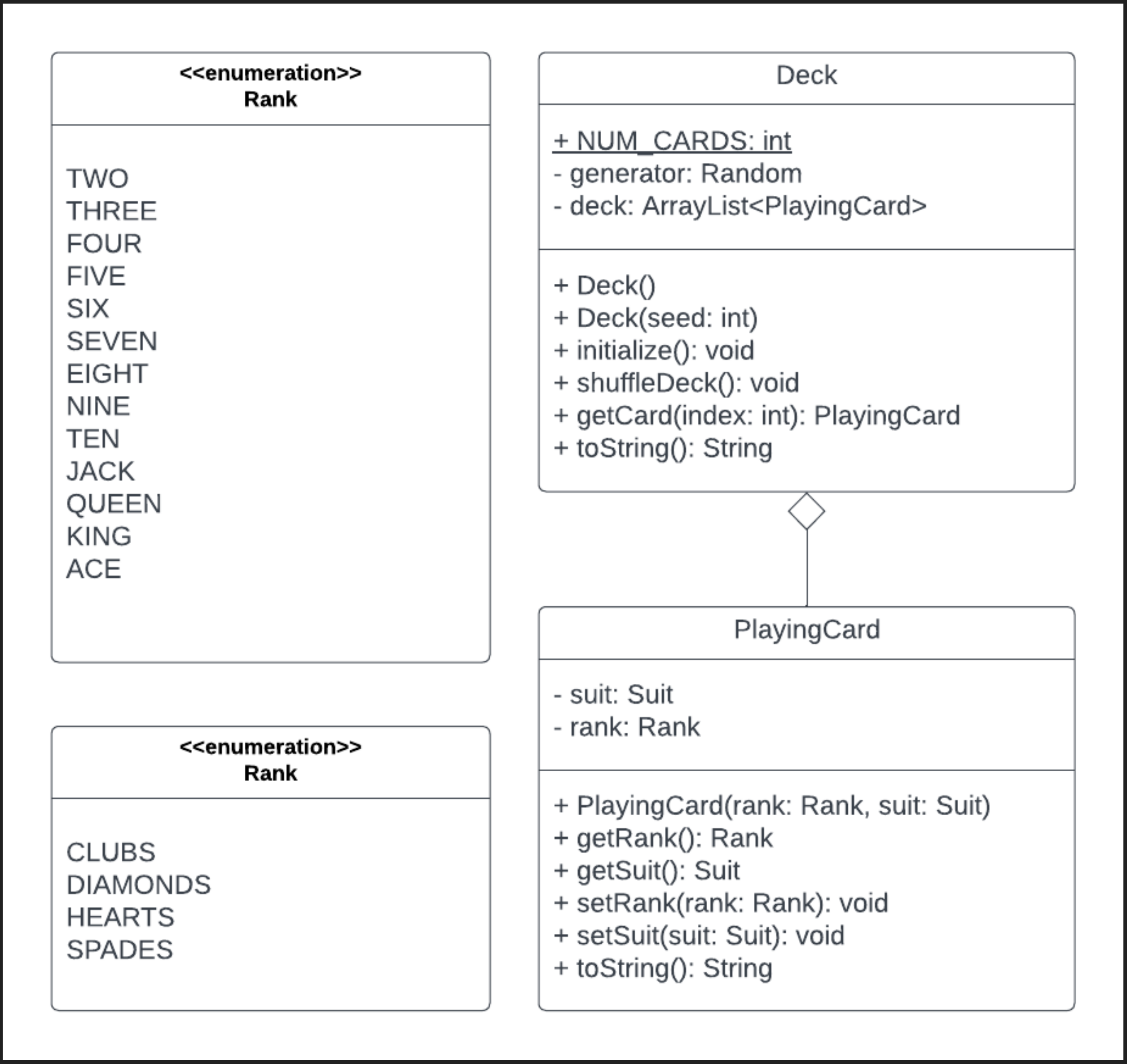
Author: Willow Sapphire

Table of Contents

For a printer-friendly version of the instructions with black text on a white background, click here

Part 1: UML Skeleton

Use the following UML to construct the two enumerated types and skeletons for the two classes in their appropriate files located in src/main/deck.



Notes

- Remember to put package declarations at the top of each file. These are all in the deck package.
- Both enumerated types and both classes are public
- The UML for the enumerated types tells you exactly what the values should be.
- You do not know everything you need to know to fill out the classes, but you can construct skeletons for each of them.

When Finished

- The `make compile` command should work with no errors
- The `make rank` command should pass all the tests
- The `make suit` command should pass all the tests

Part 2: The PlayingCard Class

The playing card class is very simple. It has two fields, a rank and a suit, they each have getters and setters, and the constructor initializes both fields to the parameters passed in.

The only other method in the class is the toString method. This should return a string that describes the card. In this case, we want it to return “RANK of SUIT” ex. “TWO of CLUBS”. I suggest you use String.format and do not explicitly call the toString methods of the enumerated types. However, you write this method in whatever way you please.

When you are done, the `make playingCard` command should pass all the tests.

Part 3: The Deck Class

The deck class is a little bit more complex. You can see from the UML that it is an aggregation of PlayingCards, which means it has a “has a” relationship with the PlayingCard class. You can also see that it has an ArrayList of PlayingCards, which is where the aggregation appears.

Deck Fields

Do not statically initialize any fields.

NUM_CARDS

- This field represents the number of cards in the deck, which is 52.
- Since it is in SCREAMING_SNAKE_CASE, you can infer that it should be final.
- Since it is underlined in the UML, it should also be static.
- Only declare it. Do not statically initialize it.

generator

- This field is used to generate random numbers for the shuffle method.
- Remember to import its type from the util package.
- Only declare it. Do not statically initialize it.

deck

- This field contains all of the PlayingCard objects in the deck.
- Remember to import its type from the util package.
- Only declare it. Do not statically initialize it.

Deck Methods

Deck: No-Arg

- This constructor sets the generator field to a new Random object. The generator is used in the shuffle method. It then calls the initialize method.

Deck: One-Arg

- This constructor also sets the generator field, but this time sets it to a new Random object that you pass the seed that was given to the constructor. This allows a user to replicate the outcome of the shuffle method by using the same seed in their own random number generator. After setting the generator, it also calls the initialize method.

initialize

- This method sets the deck field to a new ArrayList containing every type of playing card. The deck should have the playing cards in order of Rank followed by Suit. For example, two of clubs, two of diamonds, two of hearts, two of spades, three of clubs, three of diamonds, etc…
- I recommend using nested For-Each loops to accomplish this method. You may do it however you want as long as you use loops and do not hard-code every card that you are adding to the deck.
- This method is public so that users can reset their deck if they want.

shuffle

- This method moves all the cards in the deck field around to pseudo randomly arrange them.
- Here is how the algorithm works

  ```
  Loop, starting at the last index down to 1
  choose a random number between 0 and the current index
  swap the card at the current index with the card at the
  random index chosen
  ```

getCard

- Returns a PlayingCard object at the provided index in the deck.

toString()

- Returns a string representation of the deck.
- This should return the output of each card's toString method, in order, each separated by a comma and a space.
- There should not be a trailing comma and space at the end.
- You will need to loop through the deck and concatenate the toString output of each card onto a string along with a comma and space.

- When you reach the last card, you should only concatenate its toString output (no comma and space).

When you are done, the `make playingCard` command should pass all the tests. The `make test` command should also pass all the tests since, at this point, you should be done with the lab.