CS5004 Northeastern University, Silicon Valley, Prof Mark, Spring 2024

Homework 02: Hand of Cards

The task in this homework is to create and operate on a "Hand of Cards," where ***Card*** should be a newly defined concrete Class, and ***Hand*** should be a newly defined generic interface whose elements may be Cards but could possibly be some other reference type implementing the same signatures (perhaps dice). You will practice by using ArrayList<E>, a Generic type, to *implement* the Hand interface, and you will practice by directly creating a *concrete* class that overrides equality when you implement Card. For details about ArrayList<E>, please review: https://docs.oracle.com/javase/8/docs/api/java/util/ArrayList.html.

Hand<C> (presumably a subset of one or more Decks[1] of Cards) should be defined by a generic interface. (Initially, we are talking about Cards, but a Hand could potentially hold other data types, such as dice or the result of a mapping operation on a Hand.) The operations provided by Hand should include a Constructor to create an empty Hand, an **add(C c)** method to add one <C> to the "front" of a Hand, a **discard(index)** method to remove the index'th <C> from a Hand (after verifying that the index is valid), a **get(index)** method to get the index'th <C> from the Hand (ensuring that the index is within bounds), a **getSize** method to return the number of <C>s currently in the Hand, a boolean **isEmpty** method to check if a Hand is empty, and a **find**(<C>) method to return the index of the first item in the Hand that is equal to it, or -1 if the Hand does not contain that particular <C>. There should also be a **sortHand** method that accepts a higher-order comparison Function as an argument and calls Collections.sort on the Hand.

Finally, there should be a **getHand** (filter) method to return a subset of a Hand based on a Predicate (such as "suitColor is RED" or "rank > 10"). There should also be a fold/reduce method called **rankSum()** that returns the sum of the ranks of the cards in the hand. Finally, there should be a mapping method, **getMap(Function f),** that returns a Hand<C> where C is some other reference type, such as mapping each Card to its Color or its Rank, based on a higher-order function f.

Playing cards may be directly defined using a concrete class, **Card**. A Card has two fields: **suit** (an Enum) and **rank** (an Integer from 1 to 13, with Ace being 1, Jack being 11, Queen being 12, and King being 13.). It should include a constructor that accepts a suit value and a rank value. It should throw an IllegalArgumentException if the rank is out of range. There should be **getters** for the suit and rank, a **getColor** method that returns an Enum RED or BLACK depending on whether the suit is **DIAMONDS/HEARTS** versus **CLUBS/SPADES**, a **toString** method, an **equals** method, and a **hashCode** method. Two Cards are **equal** if both their suit and rank are equal. The Card class should also implement Comparable<Card> based on rank. A separate Class should implement **Comparator<Card>** based on the suit, with CLUBS < HEARTS < DIAMONDS < SPADES.

[1] For this assignment, assume that decks include 52 cards; they do not include Jokers. It is possible that more than one deck could be involved in a game, so two Cards that are not identical might nevertheless be equal.

Your solution should include Javadocs in the proper format, consistent with this description. There should also be a minimum of 2 JUnit assertions per method. Tests for the Constructor and Getters may be combined where appropriate. What matters is the number of *assertions*, not the number of *Tests*, because a single Test method can include multiple assertions. Tests should include sorting Hands based on the **compareTo** method and sorting them based on the **compare** method, as well as testing Cards for equality with both equal and unequal pairs. In your code for testing methods involving higher-order functions/predicates, please include one example with a single-method named class, one with an anonymous class, and one with a lambda expression.