361 Fall 2019, Homework 1

This is a starter Java project for Homework 1. The assignment has two problems: key word in context and a card game similar to rummy. A description of each of the problems is given below.

Note that, in addition to this file, the project contains many hints and explanations in the form of javadoc comments. You can use the command `mvn javadoc:javadoc` to compile the javadoc to HTML.

Clone this repository and import to your IDE to get started.

Problem 1 Key Word in Context

The goal of the Key Word in Context (KWIC) problem is to build a special kind of index for a document. KWIC is often used as an example problem in software engineering. Parnas treats this problem at length in the article 'On the criteria to be used in decomposing systems into modules'.

A KWIC index, also called a concordance, is an index to a document where the entries, listed alphabetically, also include some of the context (i.e. the words surrounding the entry). This repository contains an example KWIC index for a short text document implemented as linked HTML files. The example consists of three files.

- `frankensteinsample.txt`: the original document, a short sample from the novel Frankenstein by Mary Shelley (public domain)
- `frankensteinsample.html`: an HTML version of the same text
- `kwic-frankenstein.html`: KWIC index as an HTML file linked to `frankensteinsample.html` (Note: since the sample is short enough to fit on a single screen we won't see the effect of the named anchor links. These files are included to show how your program should format HTML output. I have also made a larger KWIC example available on the moodle.)

As shown in the example, your program should be able to read in a `.txt` document and produce two HTML files: an HTML version of the original text, and an HTML KWIC index that links into the previous file.

Create your main program by implementing the interface `KeywordInContext`. The `TestKWIC` class gives 6 unit tests that your implementation should be able to pass. However, your program will also be evaluated based on the HTML files it outputs. Passing the unit tests alone is not sufficient.

Review the larger KWIC example on moodle to clarify what your program should do.

Tasks for KWIC Problem

- Provide an implementation of the `KeywordInContext` interface that passes the unit tests in the class `TestKWIC`
- Find a long text document (for example from Project Gutenberg) and use your program to create a KWIC index following the examples provided. Submit your HTML files along with code. Make sure to choose a document such that the index ends up no larger than 5 MB.

Problem 2 PlayableRummy

`PlayableRummy` is an interface for the game logic of a variation of the popular card game "rummy". Together with the unit tests `TestRummyCode`, an implementation of this interface is meant to be an exercise in object-oriented development. An implementation of this type should provide a constructor with the signature `Rummy(String... players)` where the elements of `players` are the names of the players.

The expected behavior of an implementation is outlined in the unit tests that have been provided. The methods of this interface are only concerned with gameplay. There are no methods for dealing with scores. The first player to discard their hand, consistent with the rules, wins the game. The possible states of the game depend on the current player and the various phases of the play as described in the enum `Steps`.

Improper uses for this type are intended to be handled with unchecked exceptions as described in `RummyException`.

The accompanying unit tests assume certain conventions. A **non-standard** deck of 65 playing cards with 5 suits and 13 ranks is assumed. The 5 suits are Clubs, Diamonds, Hearts, Spades, and Moons. Individual cards are referenced by short 2-3 character strings. The first group of characters indicates rank while the last group indicates the suit. For example, the string `"3D"` would represent the 3 of Diamonds. The string `"10C"` represents the 10 of Clubs. The regular expression `"(A|2|3|4|5|6|7|8|9|10|J|Q|K)(C|D|H|S|M)"` defines valid card descriptions. To give some more examples, `"AM"` represents the Ace of Moons, `"2D"` represents the Two of Diamonds, `"QH"` represents the Queen of Hearts. When forming runs, an Ace can be treated as high or low. For example, `AC,2C,3C` and `JC,QC,KC,AC` are both to be understood as valid runs.

The order of the players should be consistent with the order seen in the array returned from `getPlayers()`. The first element of the array is the name of the first player, second of the second player and so on. Play progresses and cards are dealt in the corresponding order. Play always begins with the first player.

A new instance of this type should be in the WAITING step. The methods `rearrange` and `shuffle` are meant to allow the client to put the deck into a certain state before play begins. The method `initialDeal` deals cards to the players and moves the game to the first turn of the first player in the DRAW step. Play continues until a player discards or melds all of their cards. The game should go to the FINISHED step when that happens.

To have the current player declare rummy, call `declareRummy` during the MELD step. The game moves to the RUMMY step. In this step, the player puts down cards as normal and ends with the `finishMeld` method. An exception should be thrown if the player declared rummy but was not able to put down and discard their hand. Any cards that were put down are treated as normal melds, and the play proceeds to the discard phase.

Description of the standard rules of rummy. Note that the game we are implementing is a variation of the standard rules.

Tasks for Rummy Problem

- Before coding, read this description and other supporting materials and produce a *domain model* using the UML class diagram notation
- Provide an implementation of the `PlayableRummy` interface such that all 30 unit tests in the class `TestRummyCode` pass
- After implementation, sketch the programming classes as a UML class diagram and note differences with the domain model (it is ok to draw the diagrams by hand and submit a scan)
- Submit a zip file of your project folder along with the domain model and class diagrams. Also, describe how you followed any design patterns