EECS 2030: Lab 6

(1.5 % of the final grade, may be done in groups of up to three students)

Motivation

The purpose of this lab is to practice using enumerated types in Java.

Part 1: Getting Started

Download a zip file containing the Lab 6 Eclipse project.

Import the project into Eclipse by doing the following:

1. Under the **File** menu choose **Import...**
2. Under **General** choose **Existing Projects into Workspace** and press **Next**
3. Click the **Select archive file** radio button, and click the **Browse...** button.
4. In the file browser that appears, navigate to your download directory (exactly where this is depends on what computer you working on; on the lab computers the file will probably appear in your home directory)
5. Select the file **lab6.zip** and click **OK**
6. Click **Finish**.

Explore the existing methods and the test cases, try to understand the purpose of each line. For example, what parameters the constructors are expected to take, what the output should be, and why some operation(s) should be disallowed.

Part 2: Design and Implementation

The following classes are included:

Card

CardDeck

CardSuit

CardValue

Poker

PokerHand

and

CardDemo

You are to implement the first six classes in such a way that the main method in the "demo" class works correctly and produces some correct output, and the classes satisfy the following:

- **CardDeck**[1] consists of 52 distinct cards (4 suits with 13 values); when created, it does not need to be shuffled

[1] https://en.wikipedia.org/wiki/Standard_52-card_deck

- the ordering of the suits and the ranks is not important; the suggested order is alphabetical for suits, and 2-to-10, J, Q, K, A for the ranks.
- once the **Card** is created, its rank and the suit should not change
- **Deck** and **PockerHand** should be Iterable (note that you may just reuse one of the underlying iterators to provide this functionality)
- **Poker** should implement 2 methods checking for two possible poker hands[2], see the comments inside the classes for details. Hint: remember how one can count the frequency of words in text?
- whenever you pass references around, remember about differences between composition and aggregation
- **Card**, **Deck**, **PockerHand** should override the **Object**'s **toString** method in order to produce the desired output from the main method provided.
- try to avoid code duplication as much as possible.
- most constructors and methods will require just a couple of lines of code (1–3), with the exception of the **Hand** constructor (about 10 lines), and the **Poker** static methods (about 12 lines)

The tester class is provided mainly to illustrate some of the intended behaviour. Feel free to modify it any way you desire during the development; only the first 6 classes will be considered during the grading process. The output of the original tester (note that the output will be different every time you run the code:

```
Drawing one card: THREE HEARTS
Deck size before drawing 5: 51
Deck size  after drawing 5: 46
[EIGHT CLUBS, TEN HEARTS, THREE DIAMONDS, FIVE SPADES, TEN DIAMONDS]
Has a pair: true

Searching for double pairs
Pair found: [TEN SPADES, NINE DIAMONDS, FOUR DIAMONDS, NINE CLUBS, TEN HEARTS]
Pair found: [ACE HEARTS, ACE DIAMONDS, THREE SPADES, JACK DIAMONDS, THREE CLUBS]
Pair found: [KING SPADES, JACK CLUBS, JACK SPADES, THREE HEARTS, THREE DIAMONDS]
Pair found: [FOUR SPADES, KING CLUBS, KING SPADES, TEN DIAMONDS, FOUR HEARTS]
Total pairs found: 4
```

One of the ways to test your **Poker** class implementation is to check if the number of single or double pairs you find in 1000, 10000, 1000000 random decks is consistent with the expected percentages[3].

If you have questions or think that you found an error, don't hesitate to post your questions on the course forum on Moodle, or contact the instructor directly (andriyp@cse.yorku.ca).

[2] https://en.wikipedia.org/wiki/List_of_poker_hands
[3] https://en.wikipedia.org/wiki/Poker_probability

Grading

The assignment will be graded using *the Common Grading Scheme for Undergraduate Faculties*[4]. We look at whether the code passes the unit tests, satisfies the requirements of this documents, and whether it conforms to the code style rules.

Submission

Find all the `java` files in your project and submit them electronically via Moodle (no zipping is required). There should be 6 files (or 7 if tester class is submitted) in total.

If working in a group, make only one submission and include a `group.txt` file containing the names and the student numbers of the group members. The deadline is firm. Contact the instructor *in advance* if you cannot meet the deadline explaining your circumstances.

Academic Honesty

Direct collaboration (e.g., sharing your work results across groups) is not allowed (plagiarism detection software may be employed). However, you're allowed to discuss the assignment requirements, approaches you take, etc. Also, make sure to state any sources you use (online sources – including web sites, old solutions, books, etc.). Although using outside sources is allowed – with proper citing, if the amount of non-original work is excessive, your grade may be reduced.

[4] https://secretariat-policies.info.yorku.ca/policies/common-grading-scheme-for-undergraduate-faculties/