COMP1721 Object-Oriented Programming

Coursework 2

1 Introduction

Your task is to write some classes that could be used in simulations of the game of five-card draw poker. You may find that the following Wikipedia pages are helpful in understanding the task and judging whether your code is behaving correctly:

```
https://en.wikipedia.org/wiki/Five-card_draw
https://en.wikipedia.org/wiki/List_of_poker_hands
https://en.wikipedia.org/wiki/Poker_probability
```

There are two different levels of solution: basic and full. We have provided you with three classes that should be used as the basis for these solutions. We have also given you unit tests for the classes and a program that uses the classes to estimate the probabilities of different poker hands.

There are 35 marks available for this assignment, mostly for implementation of two classes that pass the tests. The remaining marks are awarded for aspects such as effective use of Java, coding style and commenting of classes, proper submission and use of Git (see Section 7).

2 Preparation

It is important that you follow the instructions below *precisely*.

1. Download `cwk2-files.zip` from Minerva. **Put this file in the top level of your repository**—i.e., the directory containing the `cwk1`, `cwk2` and `exercises` subdirectories.
2. Open a terminal window in the top level directory of your repository and unzip the Zip archive with the command `unzip cwk2-files.zip`. If you are asked whether you wish to overwrite a README file, say yes.
3. Make sure that you have the following files and subdirectories visible in `cwk2`:

   ```
   README.md       gradle          settings.gradle
   build.gradle    gradlew         src
   config          gradlew.bat
   ```

   IMPORTANT: Make sure that this is exactly what you see! For example, you should *not* have a subdirectory of `cwk2` that is itself named `cwk2`. Fix any problems with the directory structure before proceeding any further.
4. Remove `cwk2-files.zip`. Use Git to add and commit the new files, then push your commit up to gitlab.com. The following commands, executed in the terminal at the top level of your repository, will achieve all of this:

   ```
   git add cwk2
   git commit -m "Initial files for Coursework 2"
   git push origin main
   ```

3 Classes Provided

You are provided with three classes, which should be used in your solution:

- `Card`, representing a single playing card
- `CardCollection`, representing a collection of playing cards
- `CardException`, representing errors that can occur in code that manipulates playing cards

Implementations of the classes can be found in `src/main/java/comp1721/cwk2`. Take some time to study this code. **Note that you are not allowed to make any changes to these three classes** when implementing your solution. We will check for this when marking!

You will need to create two new classes:

- `Deck`, representing a standard deck of playing cards
- `PokerHand`, representing a hand of cards in a game of five-card draw poker

Before you begin implementing these classes, think carefully about the best way of reusing the code that we have provided.

4 Basic Solution

4.1 `Deck` and `PokerHand`

Requirements for these two classes are described below. **Both of them should first be implemented in skeleton form, containing the minimal amount of code that will allow the tests to compile and run.** That way, you can use the tests to guide you towards correct implementations of the classes.

Minimum requirements for the `Deck` class are:

- A default constructor that creates a deck containing the standard 52 playing cards, arranged by suits and then in rank order.
- A `size` method that returns the number of cards in the deck.
- An `isEmpty` method that returns `true` if the deck is empty of cards, `false` otherwise.
- A `contains` method with a `Card` parameter that returns `true` if the deck contains the specified card, `false` otherwise.
- A `discard` method that empties the deck of all its cards.
- A `deal` method that removes the first card in the deck and returns it.
- A `shuffle` method that rearranges cards in the deck randomly.

Unless otherwise stated above, these methods have no parameters and return nothing. Note that the `shuffle` method can be implemented very simply using the Java API (see the `Collections` utility class).

Minimum requirements for `PokerHand` are:

- A default constructor that creates an empty hand.
- A constructor with a `String` parameter that specifies the cards that should be added to the hand, using two-character abbreviations for the cards. For example, an argument of `"2D JC"` should result in the Two of Diamonds and Jack of Clubs being added to the hand.
- A `toString` method, overriding the default version, which returns a string in which cards are shown in two-character form, separated by spaces—e.g., `"2D JC 7H"`.
- Methods `size` and `discard`, which behave just like the corresponding methods of `Deck`.
- Method `discardTo`, with a `Deck` parameter, which empties the hand of cards and returns each of them to the specified deck.

`PokerHand` will also need **predicate methods** that each return a boolean value to indicate whether a hand is one of the specific types of hand in poker:

```
isPair              isFullHouse
isTwoPairs          isFlush
isThreeOfAKind      isStraight
isFourOfAKind
```

At this stage, you should create all of these as stub methods that return `false`, so that the tests compile. You can run all the tests with

```
./gradlew test
```

You should write **doc comments** for the `Deck` and `PokerHand` classes, describing what each class represents and identifying you as the author. You should also write doc comments for each public method in the two classes. See the provided classes for examples of how to write doc comments, and read the article at `https://oracle.com/technetwork/java/javase/documentation/index-137868.html`

You can generate HTML documentation from doc comments with

```
./gradlew javadoc
```

Documentation will appear in `build/docs/javadoc`. Open `index.html` in a browser to check that your doc comments have rendered sensibly.

5 Full Solution

5.1 `PokerHand`

To complete `PokerHand`, implement each of the predicate methods `isPair`, `isTwoPairs`, `isFullHouse`, etc, so that they return `true` when appropriate. If you need guidance on why the test for a particular predicate method is failing, examine the test implementation in `Full.java`. You'll find this file in the directory `core/src/test/java/comp1721/cwk2`.

5.2 Program

You are provided with a program to estimate the probabilities of poker hands. You can run this with

```
./gradlew run
```

This will generate output similar to that shown in Figure 1. The probabilities of the different hands should be similar to those seen here if you have implemented the classes correctly.

```
50,000 hands dealt

P(Pair)            = 42.100%
P(Two Pair)        =  4.676%
P(Three of a Kind) =  2.174%
P(Straight)        =  0.388%
P(Flush)           =  0.190%
P(Full House)      =  0.170%
P(Four of a Kind)  =  0.028%
```

Figure 1: Example of terminal output from `PokerStats`

The program will also create a log file named `log.txt`. This contains a record of all the hands dealt by the program. An example is shown in Figure 2.

```
Q♣4♥3♣9♣K♠
3♦Q♥2♦T♦A♣
3♥5♥2♥9♥T♥  Flush
J♠T♠A♠8♥K♦
7♥6♥6♣8♣A♦  Pair
J♣A♥K♣4♠3♠
T♣7♠J♦Q♦4♣
5♠6♠2♠J♥7♦
8♠6♦9♦7♣5♣  Straight
7♥3♦K♥K♣4♠  Pair
5♠J♣7♣Q♥8♠
A♣K♦A♥A♦3♥  3 of a Kind
8♦9♥2♠A♠5♣
3♠J♠2♥8♥Q♣
4♣6♣J♥2♣T♠
4♥7♦9♠Q♠T♣
T♦9♦6♠T♥6♦  2 Pairs
7♠5♦2♦J♦9♣
```

Figure 2: Example of log file output from `PokerStats`

6 Submission

Generate a Zip archive of your code with

```
./gradlew submission
```

This will create a file named `cwk2.zip`. This file should be submitted via the link provided in Minerva.

The deadline for submissions is **5 pm on 23 May 2022**.

7 Marking

Marks for this assignment are assigned as follows:

16	Tests for basic solution
9	Tests for full solution
4	Effective use of Java
4	Coding style and doc comments in classes
2	Correct submission and use of Git
35	