INTRODUCTION TO PROGRAMMING
DM550, DM857, DS801 (Fall 2019)

Exam project: Part II — Deadline: 23h59 on Friday, November 22nd, 2019

Overview

In this part, your task is to develop the low-level classes required to implement the card game: classes `Suit`, `Card` and `Player`.

Class `Suit`

This class has exactly four objects, corresponding to the four possible suits in a deck of cards (Spades, Hearts, Clubs and Diamonds).

Note. In order for the client for this class to work properly, it is important that the objects in this class have the correct names. Check the attached documentation for this class for these names.

Class `Card`

This card has exactly forty objects, corresponding to the forty cards in the deck of cards used by the game. Each object should have a public attribute corresponding to the suit of the card it represents. This class should also provide the following additional methods:

- a method `int points()` returning the number of points this card is worth (for scoring purposes);
- a method `boolean isHigherThan(Card other)` that returns `true` if this card is higher than `other` (assuming they both have the same suit);
- a method `String toString()` returning a textual representation of this card.

Note. In order for the client for this class to work properly, it is important that the objects in this class have the correct names. Check the attached documentation for this class for these names.

Class `Player`

Objects of this class represent individual players, characterized by their name, the cards in their hand (at most 3), and the set of cards they have collected throughout this game (at most 40). This class should provide the following methods:

- a constructor that takes as argument a `String` and creates a new player with the given name;
- a method `String name()` that returns this player's name;
- a method `Card[] hand()` that returns an array containing the cards in this player's hand;
- a method `int cardsInHand()` that returns the total number of cards currently in this player's hand;
- a method `void addToHand(Card card)` that adds `card` to the cards in this player's hand;
- a method `void addToCollectedCards(Card card)` that adds `card` to the cards in this player's pile of collected cards;
- a method `void removeFromHand(Card card)` that removes `card` from the cards in this player's hand (if it is there);
- a method `Card[] collectedCards()` that returns an array containing the cards collected by this player throughout the current game.

Expected results

You must hand in classes `Suit.java`, `Card.java` and `Player.java` implementing the contracts described above. Your classes should be providers for the remaining classes, which are provided in compiled form. Two implementations of class `Play` are also provided — a text-based one and a graphical one. Your classes should be compatible with both of them, as well as with your original implementation developed in the first part of the project.

Since all the client classes have the same name, you can test your implementation with different clients in two ways. You can make several directories (one for each client) and copy all the common class files (including the ones you implemented) plus the client to each of them, and run `Play` from the directory corresponding to each test; or you can replace `Play.class` with the implementation you want to test. (Note that the GUI implementation also includes a few `Play$*.class` files, which also must be included.)

You should also hand in a report describing the design of your classes – your choice of attributes, whether you provide any additional getters and setters, which universal methods you override, and any other design choices that you think are important to document. One of the members of the group must be clearly identified on the title page as the coordinator for this part of the project; the coordinator cannot be the same as in the first part (unless the group does not have enough elements). The source code for the developed classes should be included as appendix.

The report will be the basis for the evaluation.

Suggestions

It is recommended that you develop and test the three classes independenly. You should test them by dedicated code, and you should also check that they follow the contract by integrating them with the other given classes.