Exercise 5:
Advanced Inheritance

TECHNISCHE
UNIVERSITÄT
DARMSTADT

Software Engineering Design & Construction
WS 2016/17 - Dr. Michael Eichberg, M.Sc. Matthias Eichholz

Although this exercise is not graded, it is highly recommended to also do them on your own. Just looking at a solution is much easier in comparison to actually coming up with it. Support can be found in the forum:
`https://www.fachschaft.informatik.tu-darmstadt.de/forum/viewforum.php?f=234`

Introduction

In this exercise, you will design and implement parts of a general, abstract framework for different kinds of card games and then instantiate it for specific examples. Your solution should be as simple as possible. As a first step, make yourself familiar with the code. Add your implementations in the corresponding places and test them thoroughly.

Card games often share a common structure. Moreover, the rules of many traditional card games are not set in stone and can vary by region or be tweaked by a group of players. For an example, see the rules of Mau Mau at `http://en.wikipedia.org/wiki/Mau_Mau_(card_game)`. Variants of Mau Mau can be played with different card decks, various rules assigned to different cards and other rule tweaks. Examples:

- You can play with any standard French or German deck. Uno, with its proprietary deck, can be regarded as a Mau Mau variant as well.
- When a player plays a 2 or 7, the next player has to take two cards from the stock, unless she has a 2 or 7 as well.
- The next player has to skip a turn (commonly assigned to 8s).
- A player can set suit (commonly assigned to Jacks).
- A certain card reverses the turn order.

Task 1 The Cards Layer

As you can read on `http://en.wikipedia.org/wiki/Playing_card`, there are different kinds or decks of playing cards. Most cards are ranked – e.g., 2, 3, ..., 10, Jacks, Queens, Kings, Ace – and suited (`http://en.wikipedia.org/wiki/Suit_(cards)`). Not all of them, however, are both suited and ranked. There also exist different kinds of ranks (Full 52, Piquet 32 etc) and suits (French, German, etc).

You should implement a composable class hierarchy similar to the Smart Home example from the lecture that models different aspects of playing cards in different modules. The base module looks as follows:

```
trait Cards {
  val deck: Seq[Card]
  type Card <: TCard

  trait TCard {
    def name: String
  }

  val cardOrdering: Ordering[Card]
}
```

Model suited and ranked cards in their own subtraits `SuitedCards` and `RankedCards` and compose them into a trait `SuitedAndRankedCards`. They might need to define additional abstract member types similar to `Card` above. Write

two traits `FullRankedCards` and `PiquetRankedCards` that fix the ranks to the standard 52 (2 to Ace) and 32 (7 to Ace) ranks, respectively. Also write a trait `FrenchSuitedCards` that fixes the suits to the four suits available in a French deck. As a concrete example, create a trait `FullFrenchCards` for a standard French 52 deck as described on `http://en.wikipedia.org/wiki/Standard_52-card_deck`. (You don't have to do the following, but think how you would create a Piquet French deck, a German deck, or an Uno deck).

Put traits in a reasonable inheritance/mix-in hierarchy. Each trait should stay general (e.g. don't fix the ranks in `SuitedCards`), but make as many members (types and values) as concrete as you can (e.g., do implement value `deck` in the topmost trait as you can).

Attention: When using IntelliJ, at some point you might encounter issues with the type checker while the program still compiles (IntelliJ doesn't support path-dependent types very well).

Task 2 The Base Layer

Write a trait `Game` that uses trait `Cards` and defines all general components for a Mau Mau game. Document what your types and traits represent. Your design should include the following:

- The number of players should be configurable.
- A player should be able to receive cards (used for the initial hand and during the game as a penalty) and play a card.
- Implement the logic of a single player's turn (not an entire turn for all players) in a method `playersTurn(p: Player)`. The logic must check that the player is playing according to the rules and is not cheating.

Instantiate at least two different games of Mau Mau with different decks and different rules (explain your rules in a comment). Write a few tests that test the turn logic and that the checks against cheating or in place. For that, you need to implement one or more dummy players.

Note that we do not ask you to implement the game logic of an entire game of Mau Mau, just the components and the logic of the turn of a single player from above. Your rules must at least include that:

- One can play a card if it corresponds to the suit or value of the open card on the pile.
- There must be at least 2 cards with special meaning (take two, skip a turn, reverse the turn order, choose suit etc)

Task 3 Safety

Explain in a comment of your `Game` trait why a player cannot play cards from a wrong card deck. For example, in a game of Mau Mau with a Standard 52 deck, a player cannot play cards from another French, German or Uno deck. What exactly prevents a player from doing so?

Task 4 In Java

Think about how you would design your framework in Java 8. Add a comment to your Scala `Game` trait that names the difficulties that would arise compared to the Scala version. Hint: think about the features of Scala you use in your solution and how your design benefits from them.