Lab Exercise: A game of cards

In this exercise you will train your skills in encapsulation, abstraction and inheritance. The purpose of the exercise is not only to get a game working which meets the requirements, but to do design a solution which is still valid in the face of changing requirements.

Many of the "tricks" you will use in this exercise are commonplace, and we will address them and give them a name later in the course, so if you finish this exercise in time, you will be able to understand when we reference it later.

In this exercise you will design and implement a simple game of cards. Basically, a Player is dealt some Cards from a Deck. When a number of cards have been dealt, the Player with the highest sum of card values is the winner of the game.

The details of the games and its constituents are as follows:

Cards:

- A Card can be of suit Red, Blue, Green or Yellow and has a card number between 1 and 8
- The suit corresponds to a multiplier:
    - Red: 1
    - Blue: 2
    - Green: 3
    - Yellow: 4
- The value of a Card is the product of the suit multiplier value and the card number. E.g., a "Green 7" has a value of 3 x 7 = 21, while a "Blue 4" has a value of 2 x 4 = 8.

Player:

- A Player has a name and can hold any number of cards.
- A Player can be dealt a Card, and can be asked for the total value of his hand.
- A Player can be asked to "show his hand", i.e. output the cards (suit and card number) of every card in his hand.

Deck:

- A Deck holds a number of cards,
- A Deck can be asked to deal a number of cards to a given player

Game:
The game controls the card game, i.e.

- Accepts new players in the game before it is started
- Commands the dealing of cards
- Can be asked to announce the winner of a game.

When the winner is announced, the game is over, and a new game can begin.

Exercise 1:
Draw a domain model of the card game. As you do, look at the details above and ask yourself:

- What is the state ("knowledge") of each domain class?
- What is the behavior ("responsibility") of each domain class?
- What other classes do the individual domain classes relate to?

Exercise 2:
Create a design of the system. In doing so, be sure to think about how to encapsulate the "knowledge" of each class and how to expose the necessary behavior of each class.

Exercise 3:

Implement your game and test it: Add 3 players, deal each player 5 cards, have them show their hands and then have the winner announced. Compare the shown hands to the announced winner - is the result correct?

Exercise 4:
A new kind of player enters the game – the *weak* player. This kind of player is just like the regular player, but the poor fellow can only hold 3 cards at any time. Whenever he is dealt a fourth card, he will "drop" one of the cards he is holding.

Redesign your solution so that the weak player may play just as ordinary players. Think about what behavior and state weak players and ordinary players share, and what is particular to the individual kinds of players. If you find that the addition of a weak player to your design requires changes to any other part of the system, be sure to discuss why and how this impact can be removed – with a proper design, this change will have *zero* impact on classes other that the Player class.

Exercise 5:
A new card suit is created, the *Gold* card. This card also has card numbers in the range 1-8, but has a multiplier of 5. Add this card to your game. Just like before, if you find that this addition impacts other classes, be sure to understand which classes and discuss how this impact can be removed – again, with a proper design, this change will have *zero* impact on classes other than the Card class.

Exercise 6:
A new game variant is introduced: In this game, it is the player with the *lowest* sum of cards that wins. Introduce this variant to the design.