Object Oriented Programming

Assignment 3

Card Games

Marks available: 100

Percentage of overall marks from this assignment: 35%

Date issued: 10/09/20

Deadline: 30/11/20, 17:00

Introduction

For this assignment you will create some classes to enable the creation of card games. For the final part of this assignment you will focus on the game 'Pontoon'.

Submission instructions

Submit your work by the deadline above as a **zip** archive. Submit only .java source files in your archive unless told otherwise.

Make absolutely sure you have are not submitting .class files instead of .java files.

Each Java file must have a package declaration at the topic of it (these declarations must be on the very first line of the file). The final part of this declaration varies according to the assignment and exercise. The package declaration for this assignment is:

```
package com.bham.pij.assignments.pontoon;
```

Do not copy and paste this package declaration from this pdf! You will get unexpected characters in your files that will render them untestable. Type it in.

Your package declaration must be on the very first line of your file. Do not put anything above the package declaration, not even comments.

Do not forget to add the semicolon at the end of the package declaration. All characters in a package name should be lower case.

In each of these exercises, you are not restricted to only creating the required methods but you must create **at least** the required methods.

All of the required methods must be `public`.

None of the required methods should be declared as `static`.

Do not use non-ASCII characters anywhere in your files, even as comments.

All strings can be treated as case insensitive unless otherwise stated.

IMPORTANT: Do NOT put any code that gets user input in **ANY of the classes you submit**. If you do, your work can't be tested. This means that you should not use `Scanner`.

Introduction

For this assignment you will create a number of classes that could be used to develop a card game. Each exercise involves the creation of a single class.

You need to create all of the classes yourself. There is no prepared class for you to download for this assignment. You must create the required classes. If you choose to create other classes that is up to you.

The only validation that is required is stated in the exercises below.

As you do each exercise, you are strongly recommended to check that the functionality you added in the *previous* exercise still works. This is called *regression testing*.

Please read the following statements carefully.

You are not allowed to include the gathering of user input in any of the classes you submit. You may ask how do you create a game of Pontoon without that? The answer is that you only need to create the classes below. You do **not** need to actually develop the game of Pontoon itself.

However, for your own testing purposes, and amusement, you may wish to use the classes below to develop a Pontoon game, but that is for your own purposes only. We do not want that. Doing so will allow you to check that your code works as expected.

If you choose to create the game of Pontoon with your classes, do not add that code to the `Pontoon` class below. Create another class that contains the actual game logic itself. This other class will probably use the Scanner therefore do **NOT** submit it.

The game of Pontoon

Pontoon is a game that has many different names, and is known all over the world. The main attribute of this game is that the players are trying to get a hand of cards with the value 21 or less, but as close to 21 as possible if less than 21. A hand of cards with a value of more than 21 is 'bust' and is not valid.

The game is often played by two players with one acting as the 'bank'. You can simply consider it a game of two equal players. At the start of the game, each player gets **two cards** dealt face down. One of the

players turns both of their cards. If the value of their two cards sums to less than 16, they must receive another card, or, they must 'twist'. If the value of their cards is between 16 and 21 (inclusive) they can choose to 'stick'. Once the player is either happy with the value of their cards (and it has not exceeded 21) they stick and the game moves to the other player. The other player follows the same process and attempts to get a better hand than the first player.

For the purposes of this assignment, the rank order of hands in the game Pontoon is as follows.

(1) A 'Pontoon' is a hand with an ACE and a single card with the value 10, i.e, it comprises two cards. The value of this hand is 21. This is the best ranked hand in Pontoon and beats all other hands (apart from another 'Pontoon').

(2) A 'Five Card Trick' is a hand comprising five cards where the value of the hand does not exceed 21.

(3) A hand with any number of cards totalling 21.

(4) Hands with a total value of 20 or less, the nearer to 21 the better.

A hand with a value of more than 21 is 'bust' and is worthless.

Exercise 1: The `Card` class [25 marks]

For this exercise you should create a class called `Card`. This class represents a single playing card. If you are not familiar with playing cards at all, do some background research first.

A playing card has a *suit* and a *value*. Each card also has a numerical value, described below.

The suit of a playing card is one of the following:

CLUBS, HEARTS, DIAMONDS, SPADES

The values of playing cards are as follows. This list also shows,in parentheses, the *numerical* value of each card:

ACE (1 or 11)
TWO (2)
THREE (3)
FOUR (4)
FIVE (5)
SIX (6)
SEVEN (7)
EIGHT (8)
NINE (9)
TEN (10)
JACK (10)
QUEEN (10)
KING (10)

The `Card` class should declare public `static` enums for *suit*s and *value*s. The suit enum must be called `Suit`. The value enum must be called `Value`. The members of the enums should be declared using all

upper-case characters (as shown above). The `Value` enum should contain the values listed above (e.g. ACE, TWO, THREE, ... etc.). There is no need to explicitly store the numerical value of each card. That will be computed as discussed below.

In addition to the two enums above, this class should also have the following methods:

Getters and setters for the card's suit and value.

A method to compute the numerical value of a card. This method should have the following signature:

```
public ArrayList<Integer> getNumericalValue()
```

This method must return an `ArrayList` because if a card has the value ACE it has two possible numerical values (1 and 11). If the card is an ACE the method should return both of those values in low to high order (i.e. index 0 contains the value 1, in this case). In all other cases this method returns an `ArrayList` containing one value, which is one of the numerical values stated above.

This class must also provide a constructor that receives a `Suit` and a `Value` as parameters.

Exercise 2: The `Player` class [25 marks]

The `Player` class represents a player in a card game. The main responsibility of this class, however, is to represent the player's 'hand' of cards. A 'hand' is a collection of cards held by a player. The number of cards per player varies depending on the game and the current state of the game. That detail is not dealt with by this class, however.

The `Player` class should have the following methods:

```
public Player(String name)
```

This is the constructor. It stores the player's name.

```
public String getName()
```

Returns the player's name.

```
public void dealToPlayer(Card card)
```

This method deals a card to a player, i.e. it adds a card to a player's hand.

```
public void removeCard(Card card)
```

This method removes a card from a player's hand.

```
public ArrayList<Integer> getNumericalHandValue()
```

This method returns all of the possible numerical values of a hand. This will comprise multiple values if the hand contains ACE cards since each ACE card can have the value 1 or 11. The numerical values of the hand should be returned in low to high order (i.e. the lowest value is returned at index 0).

`public int getBestNumericalHandValue()`

This method returns the maximum numerical value of the player's hand of cards.

`public ArrayList<Card> getCards()`

Returns the cards in the player's hand.

`public int getHandSize()`

Returns the number of cards in the player's hand.

Exercise 3: The `Deck` class [30 marks]

The `Deck` class represents a whole deck of cards. There are 52 cards in a deck which is comprised of 13 cards from each of the four suits. Each suit has the following same 13 cards:

ACE, TWO, THREE, FOUR, FIVE, SIX, SEVEN, EIGHT, NINE, TEN, JACK, QUEEN, KING.

There are no repeated cards in the deck. The 'TWO of HEARTS' and 'TWO of DIAMONDS', for example, are distinct cards.

The `Deck` class has the following methods:

`public Deck()`

The constructor. This constructor should create the deck of cards.

`public void reset()`

This method re-creates a full deck of cards in an existing deck. Thus, this method will always result in the creation of 52 distinct cards in the deck.

`public void shuffle()`

This method shuffles the deck. After this method has been called the current cards that are still in the deck (which might be less than 52) should be shuffled. In general this means that if a card exists at a particular index in the deck, it should exist at a different index after the method has been called[1].

`public Card getCard(int i)`

This method returns the card at the given index.

`public Card dealRandomCard()`

This method deals a random card. A random card should be selected from those remaining in the deck and returned. The selected card should be removed from the deck.

[1]Due to the nature of random number generation, in a small number of cases a card might get shuffled back into the same position. You do not need to deal with this small possibility.

`public int size()`

Returns the number of cards currently in the deck.

Exercise 4: The `CardGame` class [10 marks]

This is an `abstract` class that will be used as the base class for other card game classes.

This class should declare the following methods.

`public CardGame (int nplayers)`

The constructor. This constructor creates the deck and sets the number of players for this game.

`public abstract void dealInitialCards()`

This abstract method deals the number of initial cards to each player in the game.

`public abstract int compareHands(Player hand1, Player hand2)`

This abstract method compares the hands of two players. If hand1 is better than hand2 the method should return -1. If hand2 is better than hand1 the method should return +1. If the two hands are equal then the method should return 0.

`public Deck getDeck()` returns the deck.

`public Player getPlayer(int i)`

This method gets the player at the index. This method assumes you have added the correct number of players to the game.

`public int getNumPlayers()` gets the number of players in the game.

Exercise 5: The `Pontoon` class [10 marks]

This class extends the `CardGame` class and should implement the abstract methods from its parent.