ASSIGNMENT 4
Black Jack

COMP-202, Fall 2015

Due: Friday, December 4th, 2015 (23:59)

You must do this assignment individually and, unless otherwise specified, you should follow all the instructions. Graders have the discretion to deduct up to 15% of the value of this assignment for deviations from the general instructions and regulations.

Part 1, Question 1:	30 points
Part 1, Question 2:	10 points
Part 1, Question 3:	60 points
Part 2, Bonus:	10 points
	110 points total

Free pass usage

If you wish to use your "free pass" (see the course webpage for more details), you are required to email the TA who is marking your assignment *before* the deadline. Please see the table below, which is referenced by the *last* name of the student, to determine your TA. Note that this is also the person whom you should see if you have questions or other complains about your marks.

TA Name	**Email Address**	**Student Last Name Range**
Teng Long	teng.long@mail.mcgill.ca	A-BAN
Andrew Holliday	andrew.holliday@mail.mcgill.ca	BAO-BUDD
Carlos Gonzalez Oliver	carlos.gonzalezoliver@mail.mcgill.ca	BUDE-CHR
Hua Qun (Robin) Yan	huaqun.yan@mail.mcgill.ca	CHS-DOW
Feras Abu Talib	ta_feras@hotmail.com	DOX-GAG
David Bourque	david.bourque@mail.mcgill.ca	GAH - HARB
Paul Husek	paul.husek@mail.mcgill.ca	HARC - JON
Babak Samari	babak@cim.mcgill.ca	JOO - KOR
Seyyed Mozafari	sh.mozafari@mail.mcgill.ca	KOS - LI, H
Egor Katkov	egor.katkov@mail.mcgill.ca	LI, I - MA
Chenghui Zhou	chenghui.zhou@mail.mcgill.ca	MAC - MUR
Jonathan Campbell	jonathan.campbell@mail.mcgill.ca	MUS - PARE
Neeth Kunnath	neeth.kunnath@mail.mcgill.ca	PARF - RENT
Ayush Jain	ayush.jain@mail.mcgill.ca@mail.mcgill.ca	RENU - SHIM
Tzu-Yang (Ben) Yu	tzu-yang.yu@mail.mcgill.ca	SHIN - TANG
Mohammad Patoary	mohammad.patoary@mail.mcgill.ca	TANN - WANG, Q
Xiaozhong Chen	xiaozhong.chen@mail.mcgill.ca	WANG, T - YANI
Stephanie Laflamme	stephanie.laflamme@mail.mcgill.ca	YAY- Z (end)

Part 1

Note: Since this is a continuation of assignment 3, there is no warm up.

For this assignment, a 75% non-compilation penalty will apply. This means that if your code does not compile, you will receive a maximum of 25% for the question. If you are having trouble getting your code to compile, please contact your instructor or one of the TAs or consult each other on the discussion boards.

Question 1: Updating CardPile to use ArrayList (30 points)

For this question, you will add an additional method to the `CardPile` *class. You will also modify the way we store the private property* `cards` *to take advantage of a new library object that we have used, specifically* `ArrayList`

The solutions for A3 will be released before A4 is due, so you can work with those. However, they can not be released until all students have handed in their A3's (approximately November 25th).

In assignment 3, we implemented a `CardPile` class to represent a pile of cards. We did this using a private property of type `Card[]`. Now, we will modify our implementation to use `ArrayList<Card>` instead. This will improve the ease of use of some of the methods.

First, remove the private property `Card[] cards` and replace it with a private property `ArrayList<Card> cards`. Your code in the `CardPile` class will no longer compile. Update all of your methods so that they work with the new type.

You will notice that most methods are much simpler with `ArrayList<Card>` since the array list class takes care of some of the messier aspects of the array. You will also notice that once you finish making the update, if you've done everything properly, all your other classes from assignment three still work as expected. Make sure to re-test your card game class from assignment three to ensure you've made the updates correctly! This is one of the main advantages to declaring class attributes as private instead of public.

What you have done in this question is known as *refactoring*, which must be done quite frequently to any code base to keep it organized.

Hint: There is a library method in the `Collections` class that will let you shuffle an `ArrayList`. You can find more information at
`http://docs.oracle.com/javase/6/docs/api/java/util/Collections.html#shuffle(java.util.List)`

Question 2: Adding two new methods to `CardPile` (10 points)

In the previous assignment, we wrote a method `makeFullDeck` that produced a `CardPile` of 52 cards. In this assignment, we will add two new methods:

- `makeFullDeck(int n)` which is a `static` method and produces a pile of cards in which each of the 52 cards are represented `n` times. For example, if `n` is 3, then there should be 3 of each possible card, for a total of 156 `Card`s. This allows us to play card games that require more than one deck.

  Note that you must keep your old `makeFullDeck` method in addition to the new method. Since this new method takes different input, we can *overload* the method name. Moreover, this new method is much easier to write if you make clever use of your old `makeFullDeck` method.

- `getNumCards()` which takes nothing as input and returns the number of `Card`s in the `CardPile`.

Question 3: BlackJack (60 points)

For this question you will implement a game of blackjack. An (abridged)[1] *version of the rules follows.*

A player is playing the game against the casino (dealer). Each card is assigned point values (see below), and the goal is to have a hand of cards with as close to a score of 21 as possible without going over. A score of 21 is the best possible score. If it is obtained with only two cards, then it is called **blackjack**.

Both the player and the dealer are each initially given two cards. The player can see both their own cards, but only the second of the dealer's cards.

After the cards are dealt, the player chooses whether to `hit` or `stay`. If they choose to hit, then they get an additional card. If they choose to stay, then the player's turn is over and the dealer gets to play. The player will continue to choose until they either stay or go over 21. *If the player goes over 21 (this is known as "busting"), then the player loses automatically no matter what the dealer does.*

If the player stays at a score of 21 or lower, the dealer then plays. The dealer will *always* hit until their total is 18 or higher, at which point they will always stay.

If neither player busts, then the player whose score is closest to 21, but not over 21 wins. If both the player and the dealer have the same score, then they tie (also known as a "push").

One last rule is that a player who is dealt "blackjack" (2 cards that total 21) automatically wins without the dealer getting to play at all *unless* the dealer is also dealt "blackjack" (2 cards that add up to 21) in which case it is a tie. This means that if the player gets "blackjack," the dealer does not get a chance to play to try to get 21 as well.

Point values of each card: Two through Nine are worth 2-9 points each. 10, Jack, Queen, and King are each worth 10 points. Ace is worth either 1 or 11, whichever gives a better score. The suit is irrelevant.

For example, a hand of: 2 of Spades, 8 of Diamonds, King of Hearts is worth 20 points.
A hand of Ace of Spades, 8 of Diamonds, King of Hearts is worth 19 points.
A hand of Ace of Spades, King of Hearts is worth 21 points (and is blackjack).
A hand of Ace of Spades, Ace of Spades, Nine of Clubs is worth 21 points.

For further details and examples see: `http://www.pagat.com/banking/blackjack.html`. Remember that you are only required to implement this simplified version of the game for this assignment.

In order to do implement this, you will define a class `Blackjack` and are *required* to write the following helper methods in addition to your main method:

- A static method `getScore` that takes as input a Card object and returns its score in blackjack. In this method, a card whose value is ACE should return 11.
- A static method `countValues` that takes as input a `CardPile` object and returns its total blackjack score. This method should return the best possible score for a `CardPile` where "best" means as close to 21 as possible without going over.
- A static method `playRound` that takes a `CardPile` as input and returns nothing. This method should execute one round of blackjack and in doing so will remove cards from the `CardPile` that was input. It must also use a `Scanner` to obtain user input, as the user needs to decide after each new card is dealt, whether to `hit` or `stay`. (Note: You can use `.next()` to read the next "word" typed by the user.) It is required that this method print all of the information that the user should have access to as well as the result of each round (including the full hand of the dealer).

In your main method, you must create a CardPile of 4 complete decks of cards and then play blackjack until there are fewer than or equal ($\leq$) to 10 cards left in the deck.

[1] Note that we have made several simplifications to the rules to make it easier to implement. Specifically, there are none of the more advanced rules such as "splitting" or "insurance".

Part 2: Bonus

Question 4: Gambling: Playing Blackjack for Money (10 points)

In order to obtain 10 bonus points[2], modify the Blackjack class in the following ways to allow for betting money. Note that if you choose to do the bonus question, you should still submit only one version of the `Blackjack.java` file since none of the changes should conflict. Make sure to double check, in this case, that the code for your bonus question does not adversely affect the code for your previous questions.

- Define an enum corresponding to the different possible outcomes of a blackjack round: DEALER_WINS, PLAYER_WINS, TIE, BLACKJACK. This enum goes inside of the Blackjack class and is called Results.
- Modify the `playRound` method to return the result of the round instead of void.
- Modify the main method to obtain the initial pile of chips using args[0]. Assume it is an integer value.
- Before each round, use a Scanner object to obtain the user's bet for this round - make sure they only bet chips that they have. You may assume the user enters an integer.
- After the round is over, update their chip pile according to the rules below.
- The game now must also end if the player runs out of chips or chooses to leave. They can choose to leave by betting a negative amount.

Betting rules:
The user can bet only once before each round begins. If they lose, then they lose their bet. If they tie, their total chip count is unchanged. If they win, they win the amount that they bet. If they get blackjack then they win 1.5 times their bet (round down if the result is a fraction).
For example, if they have 50 chips, bet 10, and win, then they have 60 chips after the round. If they got blackjack, then they would have 65 chips at the end of the round. If they lost they would have 40 chips, and if they tied they would have 50 chips.

What To Submit

You have to submit one zip file with all your files in it to MyCourses under Assignment 4. If you do not know how to zip files, please ask any search engine or friends. Google might be your best friend with this, and a lot of different little problems as well.

These files should all be inside your zip.

Please name your zip file A4_*StudentId*.zip where StudentId should be replaced by whatever your actual student id is.

`CardPile.java`
`Blackjack.java`
`confession.txt (optional)` - You should write in this file any information that you think is useful for the TA to mark the assignment. This should include things you were not sure of as well as parts of your code that you don't think it will work. Of course, like a confession, this will draw the TA's attention to the part of your code that doesn't work, but he/she will probably be more lenient than if he/she has to spend a lot of time looking for your error. It demonstrates that even though you couldn't solve the problem, you understand roughly what is going on.

[2]The bonus points will be added to your assignment four grade. If your assignment four grade is higher than 100 percent, then the bonus points will "carry over" to other assignments. If your assignments total more than 100 percent, then it can still help further improve your overall class grade when averaged with the test scores.