Java Review Assignment

Re-submit Assignment

Due Jan 20, 2017 by 11:59pm **Points** 100 **Submitting** a file upload
File Types zip

Overview

We will be building a console program that lets a user play a simplified version of the Blackjack card game.

Resources

Download **Console.java** and use it to interact with the Java console.

Blackjack Rules

To play Blackjack you use a standard 52 card deck. In Blackjack, the player plays against a dealer.

- Both player and dealer are dealt two cards
  - Player: face up
  - Dealer: face down
- Each card is given a numerical value
- The goal of the game is for the sum of your cards to be as close to, but not above, 21
- Each card has a rank (2-10, Jack, Queen, King, Ace) and suit (Spades, Clubs, Diamonds, Hearts)
  - 2-10: face value (eg. 2-10)
  - Jack, Queen, King: 10
  - Ace: 11 (simplified)
  - Card values are by rank
- After cards are dealt the player can:
  - Hit: draw another card
    - You can repeatedly draw more cards
    - If the sum of your cards is ever above 21 you lost (bust)
  - Stand: keep your hand the way it is
- After the player stands, the dealer will then hit repeatedly until they win or go bust

*Note: This is a simplified version of the rules. For more information: **link (https://en.wikipedia.org/wiki/Blackjack)** .*

Note: Here is a video that shows the rules of Blackjack. Our rules above are much simpler than those stated in this video. For example, we don't allow splits, betting, double down or insurance.
link.** **(https://www.youtube.com/watch?v=-9YGKFdP6sY)

(https://www.youtube.com/watch?v=-9YGKFdP6sY)

Card Class

Create a card class to represent a card from a **standard 52-card deck** (https://en.wikipedia.org/wiki/Standard_52-card_deck) . Your class should include:

- Suit and rank field
- A parameterized and default constructor (assume the default card is the Ace of Spades)
- Getters and setters for each field
- An appropriate toString() method

Game Classes: CardGame

Create an abstract class, called "CardGame", that stores fields and methods that are shared among most card games. Here is a class diagram that for the CardGame class:

CardGame
- gameName: String - welcomeMessage: String
+ CardGame(name: String) + abstract shuffle() + abstract deal() : Card + abstract playRound()

Note: the main purpose of the CardGame class is to hold a "game name" and "welcome message" for different types of card games. The abstract methods shuffle(), deal() and playRound() will be written in a child class.

CardGame Fields:

- **gameName:** the name of the game
- **welcomeMessage:** a message that introduces the game

CardGame Methods

- **public CardGame(String name)**
  - A single parameterized constructor that assigns your fields.

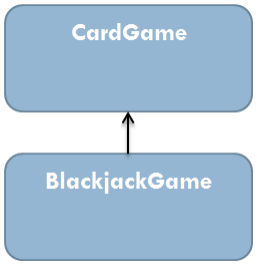
- **public abstract void shuffle()**
  - This method will shuffle the deck of cards being used in the game
  - You will need to use the Random class to generate random integers here
- **public abstract Card deal()**
  - Plays a single hand in the game
- **public abstract void playRound()**
  - With Blackjack this is a single match between the player and dealer
  - Deals a single card from the deck to the user (notice the return type)

Here is an example output of a single call to playRound():

```
Play a Blackjack game!
Welcome to my BlackJack Game!
You are dealt a(n) 10 of Hearts
You are dealt a(n) 4 of Clubs
Total: 14
Hit? (Y/N):
Y
You are dealt a(n) 4 of Spades
Total: 18
Hit? (Y/N):
Y
You are dealt a(n) Queen of Spades
Total: 28
Dealer wins!
Play another? (Y/N)
N
Thank you for playing my game!
```

Game Classes: BlackjackGame

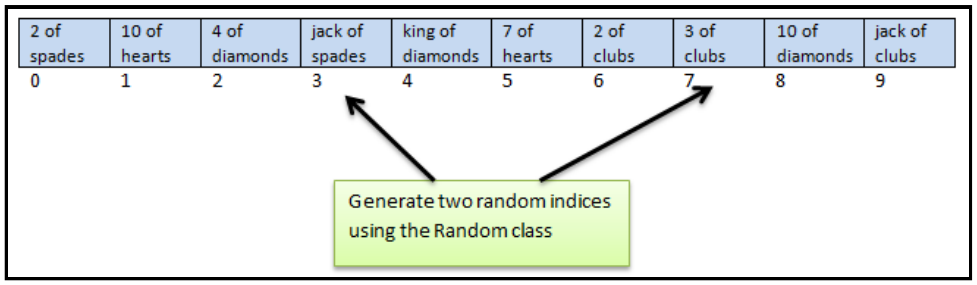
Create a child class of the "CardGame" class called "BlackjackGame." Most of your code for this assignment will be inside the BlackjackGame class. The BlackjackGame class will let a user play several games of Blackjack on the Java console.

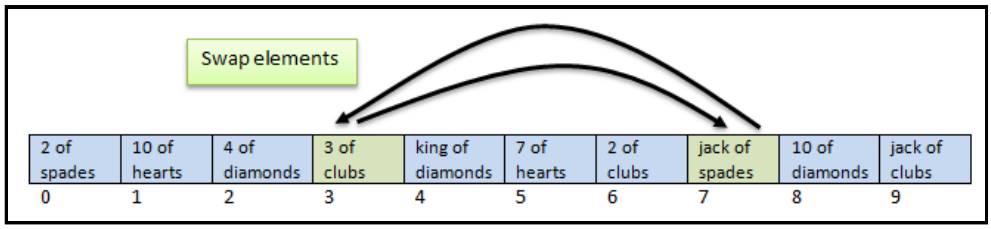


Note: You BlackjackGame class is responsible for implementing the abstract methods shuffle(), deal() and playRound(). Together these methods should allow you to play the game.

Feel free to add methods or fields to the BlackjackGame class. Although, your design should adhere to the following:

- You should not be storing the name of the game or welcome message in this class! Pass the following values up to your parent class
  - *gameName: "Blackjack"*
  - *welcomeMessage: "Play a Blackjack game!\nWelcome to my BlackJack Game!"*
- Your constructor should generate a new random deck of cards
  - Your cards should be stored in an array (eg. Card[] cards)
  - Your array should have all 52 cards in a **standard 52-card deck (https://en.wikipedia.org/wiki/Standard_52-card_deck)** .
  - *Note: you will need to loop over both the possible suit and ranks to generate each card in a deck*
- Your shuffle() method should shuffle your array of cards
  - *Note: you can approximate shuffling by repeatedly and randomly picking two elements in the array and swapping their position*





- Your playRound() method should
  - deal the user two cards and report the cards dealt and their total

```
You are dealt a(n) 8 of Clubs
You are dealt a(n) 3 of Spades
Total: 11
```

  - ask the user whether they want to hit or stand

```
Hit? (Y/N):
Y
```

- if the user hits report the card dealt and the new total

```
You are dealt a(n) 4 of Spades
Total: 15
Hit? (Y/N):
Y
```

- after the user has decided to stand or gone bust, it then deals the dealer hand and reports the winner

```
Hit? (Y/N):
N
Dealer dealt 4 of Spades
Dealer total: 4
Dealer dealt 5 of Spades
Dealer total: 9
Dealer dealt 7 of Diamonds
Dealer total: 16
Dealer dealt 7 of Spades
Dealer total: 23
Dealer loses!
```

Sample Output

```
Play a Blackjack game!
Welcome to my BlackJack Game!
You are dealt a(n) 8 of Clubs
You are dealt a(n) 3 of Spades
Total: 11
Hit? (Y/N):
Y
You are dealt a(n) 4 of Spades
Total: 15
Hit? (Y/N):
Y
You are dealt a(n) Jack of Diamonds
Total: 25
Dealer wins!
Play another? (Y/N)
N
Thank you for playing my game!
```

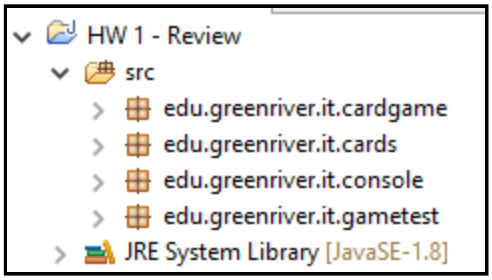
```
You are dealt a(n) 7 of Spades
You are dealt a(n) Ace of Diamonds
Total: 17
Hit? (Y/N):
N
Dealer dealt 4 of Spades
Dealer total: 4
Dealer dealt 5 of Spades
Dealer total: 9
Dealer dealt 7 of Diamonds
Dealer total: 16
Dealer dealt 7 of Spades
Dealer total: 23
Dealer loses!
Play another? (Y/N)
```

```
You are dealt a(n) 8 of Spades
You are dealt a(n) Ace of Diamonds
Total: 18
Hit? (Y/N):
N
Dealer dealt 5 of Clubs
Dealer total: 5
Dealer dealt 5 of Clubs
Dealer total: 10
Dealer dealt 3 of Diamonds
Dealer total: 13
Dealer dealt 7 of Clubs
Dealer total: 20
Dealer wins!
Play another? (Y/N)
```

Note: The user should be able to play several rounds of BlackJack, as seen from the output above.

Requirements

- Each of your files should be placed in appropriate packages



- Don't leave integers literals "littered" about your code. Turn them into constants!!!
  - eg. the deck size (52), the maximum total before bust (21), the number of times you choose to swap elements in your shuffle() method
- Your code should follow our style guide, which can be found **here**.

Java Review Rubric

Criteria	Ratings		Pts
Card class created successfully (fields, constructors, getters/setters & toString())	10.0 pts Full Marks	0.0 pts No Marks	10.0 pts
CardGame class is abstract, has correct fields and three abstract methods.	5.0 pts Full Marks	0.0 pts No Marks	5.0 pts
BlackjackGame is a child class of CardGame and passes a gameName & welcomeMessage up to its parent class.	5.0 pts Full Marks	0.0 pts No Marks	5.0 pts
BlackjackGame correctly creates a standard 52-card deck.	5.0 pts Full Marks	0.0 pts No Marks	5.0 pts
shuffle() method in BlackjackGame correctly shuffles the array of cards.	5.0 pts Full Marks	0.0 pts No Marks	5.0 pts
playRound() method lets you play a single match of Blackjack. You can hit, go bust, or stand.	20.0 pts Full Marks	0.0 pts No Marks	20.0 pts
playRound() method repeatedly deals to the computer until the computer wins or loses.	10.0 pts Full Marks	0.0 pts No Marks	10.0 pts
The user is prompted to "continue play" after a match completes. The user can then play multiple matches on the Java console.	10.0 pts Full Marks	0.0 pts No Marks	10.0 pts
Styles: naming conventions, brace placement, spacing, commented code, redundancy, packages and magic numbers.	20.0 pts Full Marks	0.0 pts No Marks	20.0 pts
Formal documentation: full Javadocs & comment block header.	10.0 pts Full Marks	0.0 pts No Marks	10.0 pts
			Total Points: 100.0