CSCI 241 Assignment 10 Counting Bridge Hands

Due: Thursday, May 6, 2021 at 11:59 p.m.

Note: This is your last assignment, due the last week of classes. Please start early so you can ask questions if necessary.

*** EXTRA CREDIT OPTION***: See note on last page of this document.

Objective:

Develop *three* Java classes: one to represent a playing card, and one to hold a deck of cards, and one to hold a player's hand as seen in the Bridge card game.

Background:

Many of us have spent way too much time playing card games on a computer in recent years (sigh…). When your instructor got married, she discovered that her husband's family expected her to learn to play Bridge (and she is still learning ;-). At this point of the semester, you should be able to implement the basic building blocks of a card game.

Description:

You are now ready to use an array of objects in a project. In this assignment, you'll place **`Card`** objects in arrays representing both a deck and a player's hand of playing cards.

1. Start your work by copying the BlueJ project named **`Assign10Bridge`**. It is available as a zip file from Canvas.
2. You will see a class already in your project named **`BridgeGame.`** This class holds a `main()` method which will use the other classes you build in your project.
3. Add three (3!) new classes to your project named **`Card`, `Deck`** and **`Hand.`** These are all *instantiable* classes. No need to panic – each class is described below, and you know enough to write the code for each one.

You SHOULD NOT make any changes to the **`BridgeGame`** class. If you wish to *temporarily* comment out lines as you write the rest of the classes, go ahead. Once you are finished, your classes *must* work with the original **`BridgeGame`** `main()` method.

All work should be done in the other classes, described below:

`Card` class

Instance variables

Each **`Card`** needs 3 ***private*** data fields, (please use these **exact** names):

1. `int` **`rank`** – holds a number between 2 and 14 (inclusive) which indicates the rank of the card
2. `char` **`suit`** – holds a character representing the suit of the card (`'C'`,`'D'`,`'H'`,`'S'`) to represent one of Clubs, Diamonds, Hearts or Spades
3. `int` **`points`** – holds the number of high-card points the card is worth

Constructor - *Test by right-clicking on the class and inspecting the object created.*
This class will have one constructor which takes 1 integer and 1 character parameter, in that order, holding the rank and suit values, respectively, to which the **`Card`** object should be initialized. For example, the arguments you would use to create a **`Card`** holding the 3 of Hearts would be `(3,'H')`.

The `points` instance variable depends on the value of the rank, and here is how to set it:

- If the rank is 11, that means the card is a "jack", which is worth 1 point.
- If the rank is 12, that means the card is a "queen", which is worth 2 points.
- If the rank is 13, that means the card is a "king", which is worth 3 points.
- If the rank is 14, that means the card is an "ace", which is worth 4 points.
- Any other card is worth 0 (zero) points.

Instance methods
Test each of these by making an object on the object bench and right-clicking on it to run each method.

1. **"Getter" (Accessor) methods**
   The class should have public "getter" methods for each of the instance variables:
   - `int getRank()`
   - `char getSuit()`
   - `int getPoints()`

2. A `toString()` method that will take no parameters and return a `String`. It will create a `String` that holds the information from the card formatted in a simple style and return that `String`. Here are some examples:
```
 AH (4)         ← Ace of Hearts (worth 4 points)
 KC (3)         ←King of Clubs (worth 3 points)
 QD (2)         ←Queen of Diamonds (worth 2 points)
 JS (1)         ←Jack of Clubs (worth 1 point)
10S (0)         ←10 of Spades (worth 0 points)
 9D (0)         ← 9 of Diamonds (worth 0 points)
```

   As you can see, this String should use 2 positions for the rank (if not 10, it will need to start with the space character), followed by the suit character, followed by another space, and finally the number of high-card points it is worth in parentheses.
   Please use this exact style! I will use a comparison program to see that your output is exactly the same as mine.

Deck class

Instance variables
Each **`Deck`** needs 2 ***private*** data fields. The first is an array of **`Card`** objects (you choose the name of your array). The second, named `count`, will keep track of how many **`Card`**s remain in the deck (i.e., the array).

<u>Constructor</u> - *Test by right-clicking on the class and inspecting the object created.*

The **`Deck`** constructor takes no parameters. It first instantiates the array of `Card`s to size 52. Then, it will fill the array with `Card` objects, in this suit order (each suit has 13 cards, ranked 2-14): Clubs, Diamonds, Hearts and Spades. When this has finished, `card[0]` should be the Two of Clubs and `card[51]` should be the Ace of Spades. Use 4 loops to enter the cards (one loop for each suit). Remember to set your `count` variable correctly as well.

<u>Instance methods</u>
Test each of these by making an object on the object bench and right-clicking on it to test each method.

1. **"Getter" (Accessor) method**
   The class should have a public "getter" methods for the integer variable:
   - `int getCount()`

2. `void shuffle()`
   takes no parameters and returns nothing. It rearranges Cards in the deck. When a `Deck` object's `shuffle()` method is used, the `Card` objects in the `Deck` are randomly mixed up. Remember how to use `Math.random()`? Here is a good way to use it for shuffling: You will need two `Deck` indices to work with, because you will exchange the `Card`s held at those two locations. For each `Card`, in positions 0-51, generate a random integer between 0 and 51 (inclusive), and swap the cards at those positions in the array. For example, suppose I have reached card 14 and have randomly generated the integer 35. Then I would swap the cards at `deck[14]` and `deck[35]`. *There is an example in your textbook of how to swap entries in an array: look on page 252 (section 7.2.6) and review the "Random shuffling" example.*

3. `Card [] dealThirteenCards()`
   In a Bridge game, each of the 4 players begins with 13 cards in their hand. Our version gives 13 cards to each player all at once, rather than giving them out one at a time around the table. This method takes no parameters and returns an array of 13 `Card`s. It will create the array inside the method and place the next 13 cards available into it and return it. Remember the `count` variable in the class? Change its value each time you place a new `Card` into the array that you build to return.

4. A `toString()` method that will take no parameters and return a `String`. It will create a `String` that holds the information from the Deck formatted as specified below and return that `String`.
   This method will build a `String` of all 52 cards in the Deck, 13 cards per line. Each Card will be shown as the `Card` class `toString()` method makes it look, with a comma and space between each `Card` on the line. (See the example output page at the end of this document to view how the output would look).

`Hand` class (this is the really fun class to write! ;-)

Instance variable
Each **`Hand`** needs 1 ***private*** data field: an array of **`Card`** objects. (You can see the **`BridgeGame`** class creates 4 **`Hand`** objects.)

Constructor
The constructor takes one parameter which is an array of `Card`s. Its only action is to assign that parameter to the instance variable.

Instance methods
In the game of Bridge, a player first counts the points held in their hand of cards. There are two ways to earn points and the first way is to count the point values of the cards. You saved those numbers when you created each `Card` object. The second way is to count the *distribution* of the cards, because running out of a suit during the play of the game can mean that you can "trump" the other cards, earning points in another way. Your class will have a method for each type of counting:

1. `int countHighCardPoints()`
   takes no parameters and returns an integer. It looks through each `Card` in the Hand array and adds its points (if it has any) to a sum, which it returns.

2. `int countDistributionPoints()`
   takes no parameters and returns an integer. It needs to count the number of `Card`s in each suit:

   - a suit with 3 cards or more counts for zero points
   - a suit with 2 cards counts one point (this is called a *doubleton*)
   - a suit with 1 card counts 2 points (this is called a *singleton*)
   - a suit with 0 cards counts 3 points (this is called a *void*)

   This method will add the points together and return that number.
   Hint: it will help to start your method with a loop to count how many cards of each suit are in the hand. Remember, the cards at not in order by suit. Once you know the total number of cards in each suit, it will be easier to check for these counts.

3. A `toString()` method that will take no parameters and return a `String`. It will create a `String` that holds the information from the `Hand` formatted as specified below and return that `String`. It will be a little different, in that we wish to print 4 lines of information, with each suit's cards on different lines. The rankings of the card suits is: clubs (lowest), diamonds, hearts, and lastly, spades (highest). The cards of the suit do NOT need to be in rank sequence. If the hand has no cards in that suit, it will simply print a blank line. Again, please see the last page of this document for an example of running the `BridgeGame` program to see sample results.

As stated earlier, I have provided the `BridgeGame` class which contains a `main()` method. When you run it against your own classes, this is what you might get for output (only the first part is exact, of course, because of the shuffling):

```
Printing original deck ...
--------------------------
 2C (0),  3C (0),  4C (0),  5C (0),  6C (0),  7C (0),  8C (0),  9C (0), 10C (0),  JC (1),  QC (2),  KC (3),  AC (4),
 2D (0),  3D (0),  4D (0),  5D (0),  6D (0),  7D (0),  8D (0),  9D (0), 10D (0),  JD (1),  QD (2),  KD (3),  AD (4),
 2H (0),  3H (0),  4H (0),  5H (0),  6H (0),  7H (0),  8H (0),  9H (0), 10H (0),  JH (1),  QH (2),  KH (3),  AH (4),
 2S (0),  3S (0),  4S (0),  5S (0),  6S (0),  7S (0),  8S (0),  9S (0), 10S (0),  JS (1),  QS (2),  KS (3),  AS (4)

Printing shuffled deck ...
--------------------------
 6C (0),  2D (0),  9H (0),  AH (4),  8S (0), 10S (0),  3C (0),  JC (1),  JD (1),  4C (0),  2S (0),  JS (1),  6D (0),
 3D (0),  4S (0),  QS (2),  QC (2),  9C (0),  QH (2),  9S (0),  9D (0),  4H (0),  8H (0),  8D (0), 10C (0),  6H (0),
 3S (0),  7H (0),  5S (0),  KC (3),  2C (0),  5D (0),  AD (4),  JH (1),  4D (0),  KH (3), 10H (0),  7D (0),  6S (0),
 KS (3),  QD (2),  3H (0),  5H (0),  AS (4),  7S (0),  5C (0),  KD (3),  2H (0), 10D (0),  AC (4),  7C (0),  8C (0)

Printing North Hand:  16 high card, 0 distribution points (16 total points)
 8C (0)  7C (0)  AC (4)  5C (0)
10D (0)  KD (3)  QD (2)
 2H (0)  5H (0)  3H (0)
 7S (0)  AS (4)  KS (3)

Printing South Hand:  11 high card, 1 distribution points (12 total points)
 2C (0)  KC (3)
 7D (0)  4D (0)  AD (4)  5D (0)
10H (0)  KH (3)  JH (1)  7H (0)
 6S (0)  5S (0)  3S (0)

Printing East Hand:  6 high card, 0 distribution points (6 total points)
10C (0)  9C (0)  QC (2)
 8D (0)  9D (0)  3D (0)
 6H (0)  8H (0)  4H (0)  QH (2)
 9S (0)  QS (2)  4S (0)

Printing West Hand:  7 high card, 1 distribution points (8 total points)
 4C (0)  JC (1)  3C (0)  6C (0)
 6D (0)  JD (1)  2D (0)
 AH (4)  9H (0)
 JS (1)  2S (0) 10S (0)  8S (0)
```

Once you reach the part after the deck has been shuffled, the sequence and point values of the cards will, of course, be different every time. However, the format of the output will always look the same.

*** EXTRA CREDIT ***

Up to 5 extra points may be earned by emailing the text of your COMPLETED Card class to your instructor by 11.59 p.m. on Thursday, April 29, 2021.

Submission Requirements

1. **Electronic:** Submit the completed implementation for the `Assign10BridgeGame` BlueJ project in your CS lab account..
2. **Canvas copy:** Submit your `Card.java`, `Deck.java` and `Hand.java` files to the associated assignment entry in Canvas. I do not need a copy of the `BridgeGame.java` file.

Grading Criteria

Your program must compile without errors to be accepted.

All of the following criteria will be used to evaluate your program:

Appropriate and complete comments	8 pts
Style/Design/Organization/Efficiency	7 pts
`Card` class	5 pts
`Deck` class	15 pts
`Hand` class	15 pts
Total points:	**50 pts**