COMPUTER SCIENCE 11A (FALL TERM, 2016)
PROGRAMMING IN JAVA

PROGRAMMING ASSIGNMENT 7

OVERVIEW

This assignment tests your understanding of everything covered so far, including classes and objects.

Your job for this assignment is two parts. One part will be as a designer of objects, and the other will be as a user of objects.

You will be designing a program that allows you to play the card games War and Blackjack at the terminal against the computer, at Antonella's Cyber Casino. To make designing the game easier, you will create two classes, which are specified here:

PART 1: CARD CLASS

Card class - the class representing a single card in a deck. Every card has three properties. The first is a "suit," which is either "Hearts", "Diamonds," "Spades", or "Clubs". A card also has a color, where Hearts and Diamonds are red, and Spades and Clubs are black. Finally, each card has a value from 1-13 (inclusive). Some values have special names: the 1 is an Ace, 11 is a Jack, 12 is a Queen, and 13 is a King.

Your first job is to design a class for a card, which each of these properties. It should have these methods:

1. **public Card(int value, String suit)** - initializes the Card and it's variables. For example, Card(12, "Hearts") will create a red 12 of hearts.
2. **public int getValue( )** - returns value of the card
3. **public String getColor( )** - returns color of the card
4. **public String getSuit( )** - returns suit of the card
5. **public String toString()** - returns string representation of the card, E.G. "King of Hearts." Or "7 of clubs"

PART 2: DECK CLASS

In an entire deck, there are 52 cards, each of the 13 values for the four different suits. You should implement a class to handle all the logistics of the deck, so that as a user of the class, all you have to do is pick cards from the deck, and then discard them once you're done with them. This class has a bit more logic in it, so here's what you have to do.

1. **public Deck()**
   a. This method will initialize the deck - creating an array of each of the 52 possible cards. After generating them, you should also shuffle them in the array (shuffling method is defined below). This method should also initialize a discard pile array,

which currently has nothing in it. You may want to initialize other variables too, to assist with your other methods.

2. **public void shuffle()**
   a. This method shuffles all the items in the deck. You can be creative with your shuffling algorithm, but I suggest you implement this one (the first algorithm under "The Modern Algorithm"). This is in "pseudo code," which means it's not in a real programming language, so it will be your job to convert this into real java code. Make sure if there are some "null" elements in your deck, that you don't shuffle them into the deck.
3. **public Card drawNextCard()**
   a. This method will give you the next card in the deck. Initially, this will be the card at index 0, then index 1, then index 2… up until index 51. After index 51, drawNextCard should take all the cards in the discard pile, put them in the deck array, empty the discard pile, shuffle them, and then return the first one.
4. **public void discard(Card c)**
   a. This method will add the card into the discard pile. Initially it will add the card to the first index of the discard pile, then the second index, etc…

Hints:

1. Remember that you can have an array of an object you created. For example, if you have a Card class, you can make: Card[ ] deck = new Card[52]; - which will be an array of size 52 that keeps cards.
2. Remember that assignments for Arrays are by reference. If you want the deck to be a copy of everything in the discard pile, you can just do:
   currentDeck = discardPile.clone();
   Then to empty the discard pile, you can just say: discardPile = new Card[52];
3. Be careful with shuffling that you don't accidentally shuffle "null" objects into the deck. This won't happen in the beginning when the deck is full, but when you're making the contents of the discard pile into the new deck, there will probably be some "null" elements at the end.
4. To help with debugging, add print statements in your classes that narrative what's going on, as well as printing the class variables. Then, make a main method that uses these classes in simple ways, and make sure they do what you predict. Here is an example of a way you can debug your class.

Feel free to add any other methods to these classes.

PART 3: CLIENT CODE

Once you have these classes done, and have tested them to make sure they work, you can start using these objects to write the games in our casino, in a new file Casino.java. The first one is mandatory, and the second one you can do for extra credit. Either way, each game should be in its own method.

Game 1: Simple War

This game is simpler than the normal game of war, but the object is to have a card with a higher value than the dealer. The user makes a bet. Then, both the user and the computer draw a card. If the user's card has a larger value than the computer's card, then the user wins, and gets the value of their bet added to their total. Otherwise, the computer wins, and the user loses their bet. After each round, discard the cards. Keep playing until the user runs out of money, or until they say they don't want to play anymore.

Game 2: Blackjack (Extra Credit!)

These rules will be simpler than the normal game of blackjack. The goal of this game is to get a set of cards that's value is as close to 21 as possible, without going over 21. Each round, the user makes a bet. Then, the user is given two cards. After these two cards, they can either "stand," which means that their score will be the value of all their cards, or "hit," which means that they will get another card from the deck, and this value is added to the total value. The user's turn is over when they stand, when they have five cards, or when their total is over 21.

Then, the dealer goes. The dealer also starts with two cards from the deck, but has to follow very specific rules (which is what makes it extremely good for a computer). If the hand has a value greater than 16, the dealer stands. Otherwise, the dealer hits.

In Blackjack, the cards have special values. Any card that has a value of over 10 (Jack, Queen, and King) will be worth 10 points. In addition, an Ace can either be worth 1 point, or 11 points, whichever is larger, but not over 21. Don't worry about the case where there are more than 1 Ace in your hand, assume they are all worth 11, or all worth 1. If it's too complicated, you can make a version where the ace is always worth 1.

At the end of each round, whoever got the higher score wins. If the user wins, they get the value of their bet added to their total money. If the computer wins, the user loses their bet. If both the user and the dealer have the same score or got over 21, it's a tie. Keep playing until the user quits, or has no more money.

The main method

Finally, you should have a main method that allows the users to play these games. If you only implemented one game, it should just start the game. If you implemented both games, you should allow the user to choose between the two. An example of one way to deal with this is given below, but design the game however you like. There are many choices that are left up to you regarding the design of the game. For example, what do you do when the user runs out of money. How do you decide how much the user starts with? Will you implement a betting minimum and maximum?

PART 4: WRITEUP

Create a text file, README.txt, where you briefly describe the design choices you made in your program, both in the objects you created, and in your main client code.

Grading:

For this assignment, you should limit yourself to the Java features covered in class so far (lecture 32). Although we will cover other topics while you are working on this assignment, do not use any of those features.

You will be graded on

- **External Correctness:** The output of your program should match exactly what is expected. Programs that do not compile will not receive points for external correctness.
- **Internal Correctness:** Your source code should follow the stylistic guidelines shown in class. Also, remember to include the comment header at the beginning of your program.

Submission:

Submit four files, README.txt, Card.java (which is your card class), Deck.java (the deck class), and Casino.java (the class with the main method). Put them into a zip file called PA7_[your_username].zip. For example, if your brandeis email is dilant@brandeis.edu, you should submit PA7_dilant.zip. This assignment will be due on **Thu, Dec 1 @ 11:00pm.**

NOTE: Make sure that your submission is in the correct format, and that all your files work as you intend them to before submitting. Additionally, make sure you are submitting the .java files, not the .class files. Any submissions not in a zip file will receive a zero, and any assignments that don't compile or that don't have .java files will receive a zero. There will be no exceptions to this rule.

Example Output:

A sample run of the program is shown below (for a version with both games), but be sure to design the program the way you want to, don't just copy this input/output. Be creative!

```
Welcome to the casino. You will start off with $1000

Enter 1 for war, 2 for blackjack, 3 to quit:  2
How much do you bet?  100
Current hand:
[6 of Clubs, 3 of Spades, null, null, null]
Value: 9
Enter 1 to hit, enter any other int to stand: 1
Current hand:
[6 of Clubs, 3 of Spades, Jack of Clubs, null, null]
Value: 19
Enter 1 to hit, enter any other int to stand: 2
Your final score: 19

My turn
Current hand:
[Queen of Spades, Ace of Spades, null, null, null]
Value: 11 or 21
I'm going to stand
My final score: 21
I win
Money Left: 900.0
Do you want to keep playing? Y/N:  Y
How much do you bet?  100
Current hand:
[Ace of Hearts, 5 of Spades, null, null, null]
Value: 6 or 16
Enter 1 to hit, enter any other int to stand: 1
Current hand:
[Ace of Hearts, 5 of Spades, 8 of Spades, null, null]
Value: 14
Enter 1 to hit, enter any other int to stand: 1
Current hand:
[Ace of Hearts, 5 of Spades, 8 of Spades, 8 of Clubs, null]
Value: 22
This is over 21, sorry.
Your final score: -1

My turn
Current hand:
[9 of Hearts, King of Hearts, null, null, null]
Value: 19
I'm going to stand
My final score: 19
I win
```

```
Money Left: 800.0
Do you want to keep playing? Y/N:  Y
How much do you bet?  500
Current hand:
[2 of Clubs, King of Diamonds, null, null, null]
Value: 12
Enter 1 to hit, enter any other int to stand: 1
Current hand:
[2 of Clubs, King of Diamonds, 10 of Diamonds, null, null]
Value: 22
This is over 21, sorry.
Your final score: -1

My turn
Current hand:
[10 of Clubs, 3 of Hearts, null, null, null]
Value: 13
I'm going to hit
Current hand:
[10 of Clubs, 3 of Hearts, 6 of Hearts, null, null]
Value: 19
I'm going to stand
My final score: 19
I win
Money Left: 300.0
Do you want to keep playing? Y/N:  N


Enter 1 for war, 2 for blackjack, 3 to quit:  1
The rules of this game are simple. If you have the higher card, you
win. If I have the higher card, I win. If it's a tie, neither of us win

How much do you bet?  100
Your card: 8 of Hearts
My card: 2 of Diamonds
Congrats, you won!

Your new total is: 400.0
Do you want to keep playing? Y/N: Y
How much do you bet?  200

Your card: King of Spades
My card: 7 of Clubs
Congrats, you won!

Your new total is: 600.0
Do you want to keep playing? Y/N: Y
How much do you bet?  500
```

```
Your card: 7 of Spades
My card: 10 of Spades
Sorry, I won

Your new total is: 100.0
Do you want to keep playing? Y/N: Y
How much do you bet?  100

Your card: 9 of Diamonds
My card: 3 of Clubs
Congrats, you won!

Your new total is: 200.0
Do you want to keep playing? Y/N: Y
How much do you bet?  500
You bet too much. Enter a bet that is under 200.0
How much do you bet?  200

Your card: 5 of Hearts
My card: Jack of Diamonds
Sorry, I won
Your new total is: 0.0
Your total is: 0.0
Thanks for playing!
```