Project One : Blackjack

Due: 24:00 April 19, 2018.
Scores : 20

I. Motivation

This project is aimed at helping you employ ood language to design system for cards game. To give you experience in implementing abstract data types (ADTs), using interfaces (abstract base classes), and using interface/implementation inheritance.

The classes I upload are given to you as reference.

II. Introduction

In this project, we will implement a simplified version of the card game called **Blackjack**, also sometimes called 21. It is a relatively simple game played with a standard deck of 52 playing cards. There are two principals, a **dealer** and a **player**. The player starts with a bankroll, and the game progresses in rounds called **tricks (player and dealer both finish one turn called one trick)**.

At the start of each trick, the player decides how much to wager on this trick. It can be any amount between some **minimum** allowable wager and the player's total bankroll, inclusive.

After the wager, the dealer deals a total of four cards: the first face-up to the player, the second face-up to himself, the third face-up to the player, and the fourth face-down to himself. The second card of the dealer which is face-down is called the **hole card.**

The player then examines his cards, forming a total. Each card 2-10 is worth its spot value; each face card (jack, queen, king) is also worth 10. An ace is worth either 1 or 11--whichever is more advantageous to the player. If the total includes an ace counted as 11, the total is called "**soft**", otherwise it is called "**hard**".

Play progresses first with the player, then the dealer. The player's goal is to build a turn that is as close to 21 as possible without going over---the latter is called a "**bust**", and a player who busts loses the trick without forcing the dealer to play. As long as the player believes another card will help, the player "**hits**"---asks the dealer for another card. Each of these additional cards is dealt **face-up**. This process ends either when the player decides to "**stand**"---ask for no cards, or the player busts. Note that a player can stand with two cards; one need not hit at all in a turn. If the player is dealt an ace plus any ten or face card (jack, queen, king), the player's hand is called a "**natural 21**", and the player's wager is paid off with 3 to 2 odds, without examining the dealer's cards. In other words, if the player had wagered 10, the player would win 15 (i.e., his bankroll will increase by 15) if dealt a natural 21.

If the player neither busts nor is dealt a natural 21, play then progresses to the dealer. The dealer **must** hit until he either reaches a total greater than or equal to 17 (hard or soft), or busts. If the dealer busts, the player wins. Otherwise, the two totals are compared. If the dealer's total is higher, the player's bankroll decreases by the amount of his wager. If the player's total is higher, his bankroll increases by the amount of his wager. If the totals are equal, the bankroll is unchanged; this is called a "**push**".

Note that this is a very simplified form of the game: we do not split pairs, allow double-down bets, or take insurance, etc. (http://en.wikipedia.org/wiki/Blackjack)

III. Programming Assignment

You will provide one or more implementations of four separate classes for this project: a deck of cards, a blackjack hand, a blackjack player, and a game driver. All files referenced in this specification are located in the **ProjectOne** folder.

1. The Deck Class

Your first task is to implement the following class representing a deck of cards:

```
final int DeckSize = 52;
   // A standard Deck of 52 playing card---no jokers

   Deck(){};

   // EFFECTS: constructs a "newly opened" Deck of card.  first the
   // spades from 2-A, then the hearts, then the clubs, then the
   // diamonds.  The first Card dealt should be the 2 of Spades.

   void reset(){};

   // EFFECTS: resets the Deck to the state of a "newly opened" Deck
   // of card:

   void shuffleOnce(int n){};

   // REQUIRES: n is between 0 and 52, inclusive.
   // EFFECTS: cut the Deck into two segments: the first n card,
   // called the "left", and the rest called the "right".  Note that
   // either right or left might be empty.  Then, rearrange the Deck
   // to be the first Card of the right, then the first Card of the
   // left, the 2nd of right, the 2nd of left, and so on.  Once one
   // side is exhausted, fill in the remainder of the Deck with the
   // card remaining in the other side.  Finally, make the first
   // Card in this shuffled Deck the next Card to deal.  For example,
   // shuffle(26) on a newly-reset() Deck results in: 2-clubs,
   // 2-spades, 3-clubs, 3-spades ... A-diamonds, A-hearts.

   // Note: if shuffle is called on a Deck that has already had some
   // card dealt, those card should first be restored to the Deck
   // in the order in which they were dealt, preserving the most
   // recent post-shuffled/post-reset state.

   void shuffle(int times){};
   // REQUIRES: times is the time of using shuffleOnce method to shuffle
   // cards, normally time should be at least 5.

   //EFFECTS: In each time, use Math.random to pick an cut number between
   // 13 - 39, and then call shuffleOnce method with the number;

   Card deal(){
      return null;
   };

   // EFFECTS: returns the next Card to be dealt.  If no card
   // remain, throws an instance of DeckEmptyExecption and then
   // reset the deck

   int cardsLeft(){
      return 0;
   };
   // EFFECTS: returns the number of card in the Deck that have not
   // been dealt since the last reset/shuffle.
```

The Deck Class depends on the following Card type: which is declared in card.class, and included by deck.class. The file defines SpotNames and SuitNames for you, so that SuitNames[Suit.HEARTS.ordinal()] evaluates to “Hearts”, and so on.

2. The Hand Class

Your second task is to implement the following class representing the cards in player or dealer in one turn:

```
 class HandValue {
      int  count;    // Value of hand
      boolean soft;     // true if hand value is a soft count
   };
   // OVERVIEW: A blackjack hand of zero or more cards

   private HandValue curValue;

   Hand(){};
   // EFFECTS: establishes an empty blackjack hand.

   HandValue getHandValue(){return null;};
   // EFFECTS: returns the present value of the blackjack hand.  The
   // count field is the highest blackjack total possible without
   // going over 21.  The soft field should be true if and only if at
   // least one ACE is present, and its value is counted as 11 rather
   // than 1.  If the hand is over 21, any value over 21 may be returned.
   //
   // Note: you are not allowed to change any member variables inside
   // handValue.  It is required because Players only get const Hands
   // passed to them, and therefore can only call methods like addCard,
   // to change the hand.
```

Hand class represent the cards in player's or dealer's hand in one trick.

3. The Player Class

Your third task is to implement two different blackjack players. The class for a Player is:

```
public abstract class Player {

private Hand hand;

 public int bet(int bankroll, int minimum);
   // REQUIRES: bankroll >= minimum
```

```
    // EFFECTS: returns the player's bet, between minimum and bankroll
    // inclusive

    public boolean draw(Card dealer,            // Dealer's "up card"
                  Hand hand); // Player's current hand
    // EFFECTS: returns true if the player wishes to be dealt another
    // card, false otherwise. For dealer The dealer must
    // hit until he either reaches a total greater than or equal to 17
    // (hard or soft), or busts. For player

    public void expose(Card c);
    // EFFECTS: allows the player to "see" the newly-exposed card c.
    // For example, each card that is dealt "face up" is expose()d.
    // Likewise, if the dealer must show his "hole card", it is also
    // expose()d.  Note: not all cards dealt are expose()d---if the
    // player goes over 21 or is dealt a natural 21, the dealer need
    // not expose his hole card.

    public void shuffled();
    // EFFECTS: tells the player that the deck has been re-shuffled.
}
```

The Player class depends on the Hand type. You are to implement three different derived classes for two kind of players (simple and counting) with different strategy and one for dealer.

```
public class Dealer extends Player {

    GroupOfCards memoryCard;

    @Override
    public boolean draw(Card dealer, Hand hand) {
        return false;
    }
}
```

The first derived class is the Simple player, who plays a simplified version of basic strategy for blackjack. The simple player always places the minimum allowable wager, and decides to hit or stand based on the following rules and whether or not the player has a “**hard count**” or “**soft count**”:

The first set of rules apply if the player has a “**hard count**”, i.e., his best total counts an Ace (if any) for 1, not 11.

- If the player’s hand totals 11 or less, he always hits.
- If the player’s hand totals 12, he stands if the dealer shows 4, 5, or 6; otherwise he hits.
- If the player’s hand totals between 13 and 16 inclusive, he stands if the dealer shows a 2 through a 6 inclusive; otherwise he hits.
- If the player's hand totals 17 or greater, he always stands.

The second set of rules applies if the player has a “**soft count**”---his best total includes one Ace worth 11. (Note that a hand would never count two Aces as 11 each--that's a bust of 22.)

- If the player’s hand totals 17 or less, he always hits.
- If the player’s hand totals 18, he stands if the dealer shows a 2, 7, or 8, otherwise he hits.
- If the player’s hand totals 19 or greater, he always stands.

The Simple player does nothing for expose and shuffled events.
The second derived class is the Counting player. This player counts cards in addition to playing the basic strategy. The intuition behind card counting is that when the deck has more face cards (worth 10) than low-numbered cards, the deck is favorable to the player. The converse is also true.

The Counting player keeps a running “count” of the cards he’s seen from the deck. Each time he sees (via the expose() method) a 10, Jack, Queen, King, or Ace, he subtracts one from the count. Each time he sees a 2, 3, 4, 5, or 6, he adds one to the count. When he sees that the deck is shuffled(), the count is reset to zero. Whenever the count is +2 or greater and he has enough bankroll (larger than or equal to the double of the minimum), the Counting player bets double the minimum, otherwise he bets the minimum. The Counting player should not re-implement methods of the Simple player unnecessarily.

4. The Driver program

Finally, you are to implement a driver program that can be used to simulate this version of game given your implementation of the classes described above.

You are to put your implementation of this driver program in a file named blackjack.java.

The driver program, when run, takes three arguments:

```
 <bankroll>    <tricks>        <minimum-wager>    [simple|counting]
```

The first argument is an integer denoting the player’s starting bankroll. The second argument is the maximum number of tricks to play in the simulation. The third argument is the minimum wager in one trick. **You can assume that these three integers input by the user are positive (≥1).** The final argument is one of the two strings “simple” or “counting”, denoting which of the two players to use in the simulation. For example,

```
100     3        5        simple
```

Then your program simulates the simple player playing 3 tricks with an initial bankroll of 100 and minimum wager one trick of 5.

The driver first shuffles the deck. To shuffle the deck, you choose **seven** cut number between 13 and 39 inclusive **at random**, shuffling the deck with each of these cuts. Each time the deck is shuffled, first announce it:

```
System.out.println(  "Shuffling the deck”);
```

And announce each of the seven cut points:

```
System.out.println(“cut at " + cut);
```

then be sure to tell the player via shuffle().

Then, while the player's bankroll is larger than or equal to the minimum wager and there are tricks left to be played:

- Announce the trick: *System.out.println("Trick “ + thistrick + " bankroll " + bankroll)* where the variable thistrick is the trick number, starting from 1

- If there are fewer than 20 cards left, reshuffle the deck as described above. Note that this happens only at the beginning of each trick. It does not occur during a trick even if the number of cards is fewer than 20.

- Ask the player for a wager and announce it: *System.out.println("Player bets " + wager);*

- Deal four cards: one face-up to the player, one face-up to the dealer, one face-up to the player, and one face-down to the dealer. Announce the face-up cards using print. For example:

```
Player dealt Ace of Spades
Dealer dealt Two of Hearts
```

Use the SpotNames and SuitNames arrays for this, and be sure to expose() any faceup cards to the player.

If the player is dealt a natural 21, immediately pay the player 3/2 of his bet. Note that, since we are working with integers, you'll have to be a bit careful with the 3/2 payout. For example, a wager of 5 would pay 7 if a natural 21 is dealt, since (3*5)/2 is 7 in integer arithmetic. In this case, announce the win: *System.out.println("Player dealt natural 21");*

- If the player is not dealt a natural 21, have the player play his turn. Draw cards until the player either stands or busts. Announce and expose() each card dealt as above.

- Announce the player's total *System.out.println("Player's total is " + player_count)*;

- where the variable player_count is the total value of the player's hand. If the player busts, say so: *System.out.println("Player busts")*; deducting the wager from the bankroll and moving on to the next trick.

- If the player hasn't busted, announce and expose the dealer's hole card. For example:

  Dealer's hole card is Ace of Spades

(Note: the hole card is NOT exposed if either the player busts or is dealt a natural 21.)

- If the player hasn't busted, play the dealer's turn. The dealer must hit until reaching seventeen or busting. Announce and expose each card as above.

- Announce the dealer's total *System.out.println("Dealer's total is " + dealer_count);* where the variable dealer_count is the total value of the dealer's hand. If the dealer busts, say so *System.out.println("Dealer busts");* crediting the wager from the bankroll and moving on to the next trick.

- If neither the dealer nor the player bust, compare the totals and announce the outcome. Credit the bankroll, debit it, or leave it unchanged as appropriate.

  System.out.println("Dealer wins");
  System.out.println("Player wins");
  System.out.println("Push");

- Finally, when the player either has too little money to make a minimum wager or the allotted tricks have been played, announce the outcome:

  System.out.println("Player has " + bankroll + " after " + thistrick+ " tricks");

where the variable thistrick is the current trick number. In the special case where the initial bankroll is less than the minimum, we have thistrick = 0, since the player hasn't played any trick yet.

<u>IV. Source Code Files and Compiling</u>

There are files located in the project-two.zip from our git Resources: You should copy these files into your working directory.

You need to write all the files but Spot, Suit, Card files, according to the annotations. They are discussed above and summarized below:

blackjack.java: your simulation driver
CountingPlayer.java:
Dealer.java:
SimplePlayer.java:
deck.java: your Deck implementation
groupOfCards.java:
hand.java: your Hand implementation
player.java: abstract class of your two player and one dealer

In order to guarantee that the TA compile your program successfully, you'd better to name you source code files like how they are specified above , but you are encouraged to add more appropriate methods or classes.