Laboratory 4: Black Jack!

Java API
ArrayList
Lab 04 Documentation Included
matrix package "external" documentation Included

Download Lab04.zip for all of the additional supporting files that you will need to compile and run. Extract the files into your working directory

You can find the rules of blackjack all over the Internet. To get an idea of what you are trying to accomplish in this lab, I'll demonstrate the final solution. The dealer stands on all 17s. Doubling after splitting is always allowed. Multiple splitting is always allowed.

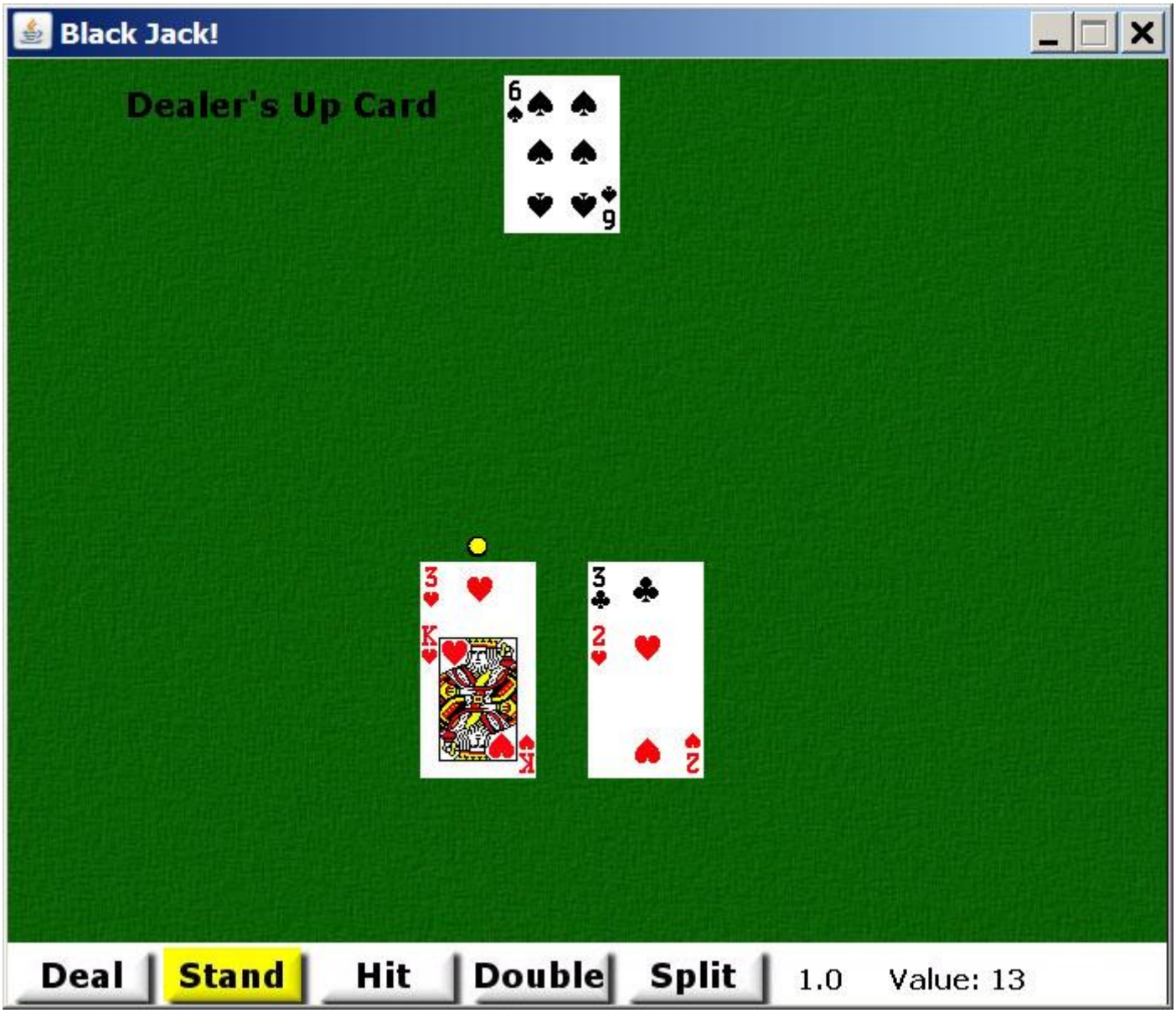
The yellow highlighting in BlackJack represents the optimal decision given your cards and the dealer's up card. Refer to the table below for the complete optimal strategy for blackjack. Across is the dealer's up card (T means a 10 or any face card). Down is your hand value. The table entries are as follows:

- **S** = Stand
- **H** = Hit
- **D** = Double Down
- **P** = Split

The A/ table entries represent soft hands of two or more cards (where one card is an ace). Splitting is only relevant at the bottom of the table where the possible duplicate card values are listed.

The row value listed next to the hand total indicates the row in the strategy array (see **BlackJackPlayer**) where the desired move for the corresponding hand total can be found. After the row has been determined by the hand total, simply go to the column corresponding to the dealer's up card. **Note:** the hand totals begin with 2 (two aces, row 0) and go up to 21 (row 19).

Dealer's Up Card	2	3	4	5	6	7	8	9	T	A
<= 8 (row 0 - 6)	H	H	H	H	H	H	H	H	H	H
9 (row 7)	H	D	D	D	D	H	H	H	H	H
10 (row 8)	D	D	D	D	D	D	D	D	H	H
11 (row 9)	D	D	D	D	D	D	D	D	D	H
12 (row 10)	H	H	S	S	S	H	H	H	H	H
13 (row 11)	S	S	S	S	S	H	H	H	H	H
14 (row 12)	S	S	S	S	S	H	H	H	H	H
15 (row 13)	S	S	S	S	S	H	H	H	H	H
16 (row 14)	S	S	S	S	S	H	H	H	H	H
>= 17 (row 15 - 19)	S	S	S	S	S	S	S	S	S	S
A/2 (row 20)	H	H	H	D	D	H	H	H	H	H
A/3 (row 21)	H	H	H	D	D	H	H	H	H	H
A/4 (row 22)	H	H	D	D	D	H	H	H	H	H
A/5 (row 23)	H	H	D	D	D	H	H	H	H	H
A/6 (row 24)	H	D	D	D	D	H	H	H	H	H
A/7 (row 25)	S	D	D	D	D	S	S	H	H	H
A/8 (row 26)	S	S	S	S	S	S	S	S	S	S
A/9 (row 27)	S	S	S	S	S	S	S	S	S	S
A/10 (row 28)	S	S	S	S	S	S	S	S	S	S
2/2 (row 29)	P	P	P	P	P	P	H	H	H	H
3/3 (row 30)	P	P	P	P	P	P	H	H	H	H
4/4 (row 31)	H	H	H	P	P	H	H	H	H	H
5/5 (row 32)	D	D	D	D	D	D	D	D	H	H
6/6 (row 33)	H	P	P	P	P	H	H	H	H	H
7/7 (row 34)	P	P	P	P	P	P	H	H	H	H
8/8 (row 35)	P	P	P	P	P	P	P	P	P	P
9/9 (row 36)	P	P	P	P	P	S	P	P	S	S
10/10 (row 37)	S	S	S	S	S	S	S	S	S	S
A/A (row 38)	P	P	P	P	P	P	P	P	P	P



Lab:

- Randomizing Object Arrays (Shuffling)
- For-Each Statement
- Object Arrays as Parameters/Return Values
- ArrayList (Using Generics)
- Primitive Arrays

Part I: Decks of Cards

For this part and the subsequent parts of this lab, refer both to the instructions below and the instructions given in the .java files.

Download the **Card** class (it's in the zip file). Although not necessary for this lab, the compareTo method for the Card class will compare by face value first, then by suit. A sorted Deck of Cards will have all the Aces first in the order Spades, Hearts, Clubs, Diamonds. Then the 2s in the same suit order, and so forth.

Complete the constructor in the Decks class. Create the requested number of standard 52 card decks (check user input). You will need to use a nested for loop. After you have all the required Cards, shuffle all of the of Cards together.

Complete the shuffle method in the **Decks** class. Use the Permutation class, which requires the Random class. After all of the Cards are shuffled, reset count so that all of the Cards in the Decks are again available.

Look at the deal method (the code for this method has been given to you). The deal method uses the count instance variable to return the next Card in the Decks. After a Card has been selected from the Decks, count is decremented in preparation for the next call to deal. If there are no more Cards left to deal, shuffle is called to reset the Decks.

Part II: BlackJackHand

Complete the BlackJackHand class. A **BlackJackHand** object holds a minimum of two Cards, but the maximum number that the hand will hold is not known. Thus, it is convenient to use an **ArrayList** to hold the Cards in the BlackJackHand. Make your ArrayList able to hold only Card objects. The constructor will accept two Cards, which are placed in the ArrayList. You must also determine whether the hand is a soft hand or not. Simply determine if one of the two Cards is an Ace, in which case the hand is soft. In addition to the constructor, complete the following methods:

- public void drawDealerHand(Graphics g, int x, int y, boolean done)
  - Draws this BlackJackHand (a player hand) to the screen.

- private int[] sum()
  - Helper method to computes the value of this BlackJackHand.
  - Two values are determined (can be computed simultaneously).
  - The first total is obtained with all aces being counted as 1.
  - The second total is obtained with one ace being counted as 11, if there is at least one ace.
  - Both totals are returned in an integer array!
  - This is to help the calling method (see the next method) to determine if this BlackJackHand is soft or not.

- public int handValue()
  - Computes the value of this BlackJackHand.
  - Determines whether this BlackJackHand is soft or not using both totals returned from the helper sum method.
  - If the second total is less than 21 but the second total and the first total are the same,
  - either total is used and this BlackJackHand is not soft.
  - Otherwise, the second total is used and this BlackJackHand is soft.
  - If the second total is greater than 21, the first total is used and this BlackJackHand is not soft.

- Remember to set soft to true if the hand is soft.

- public void hit(Card card)
  - Hits this BlackJackHand by adding the Card passed in
  - to the ArrayList storing the Cards in this BlackJackHand.
  - This method should be very short.
- public boolean canSplit()
  - This method determines whether the BlackJackHand can be split or not.
  - In order qualify for a possible split, the BlackJackHand can only have two Cards, and they must have the same value.
  - This means that any two cards with value 10 can be split, i.e. a King and a Queen can be split.

- public BlackJackHand[] split(Card card1, Card card2)
  - If the BlackJackHand can be split, this method will split the BlackJackHand.
  - The two Cards in the current BlackJackHand become one Card each for two new BlackJackHands.
  - The next two Cards for the two new BlackJackHands are passed in as parameters.
  - You will need to create new BlackJackHands.

Part III: BlackJackPlayer

Complete the BlackJackPlayer class. The BlackJackPlayer class holds all of the player's hands. Like the number of cards in a BlackJackHand, the maximum number of hands that the player will have is not known (due to splitting). BlackJackPlayer keeps track (through the **index** instance variable) of which hand the player is currently playing. The BlackJackPlayer class also **has-a** BlackJackStrategy which must be instantiated in the constructor. Complete the following methods:

- public void draw(Graphics g, int width, int height)

- public void split(Card card1, Card card2)
  - The player wants to split.
  - You need to remove the hand that is being split from the ArrayList.
  - Create two new hands and insert them into the appropriate place in the ArrayList.

- public void doubleDown(Card dealt)
  - call a method located in BlackJackHand

- public int result(BlackJackHand dealer)

Part IV: BlackJackStrategy

Complete the BlackJackStrategy class to obtain the desired move given a player's hand total and the dealer's up card. The desired strategy is read in from a text file and stored in a matrix (**BasicMatrixInterface**, done for you). Use the row and column information provided above to determine the optimal strategy. Your result will be highlighted in yellow when the game is running. The user is not required to use the optimal strategy, however. Assume a BlackJackHand method isSoft() is available, which you will write in the next part. Note the following convention when reading from the text file:

- 1 = Stand
- 2 = Hit
- 3 = Double
- 4 = Split

Complete the following methods within BlackJackStrategy:

- public char getMove(BlackJackHand player, BlackJackHand dealer)
    - Extracts the desired move (excluding splits) from the strategy array for hard and soft hands.
    - Use the hand value to make it easier to identify the necessary row from the strategy array.
    - Returns a 'S' if the correct play is to stand.
    - Returns an 'H' if the correct play is to hit.
    - Returns a 'D' if the correct play is to double.
    - **Hint:** if the player's hand is soft, you need to add 7 to the player's hand value to get the row for the correct play, otherwise 2 should be subtracted from the hand value.
    - **Hint:** if the dealer's up card is an Ace (which has the value of 1) you need to extract the correct play from column 9, otherwise a 2 should be subtracted from the up-card value.
    - Note: the game will interpret a 'D' as an 'H' if the hand cannot be doubled.
    - This class only has the responsibility of reading from the matrix.

- public boolean shouldSplit(BlackJackHand player, BlackJackHand dealer)
    - Determines if a split is the desired move if the first two cards in a hand have the same value.
    - Use the (modified) hand value to make it easier to identify the necessary row from the strategy matrix.
    - Returns a false if a split is not the correct play or true if a split is the correct play according to the strategy.
    - **Hint**: if the player's hand is soft, you need to extract the correct play from row 38, otherwise 27 should be added to the face value.
    - **Hint**: if the dealer's up card is an Ace (which has the value of 1) you need to extract the correct play from column 9, otherwise a 2 should be subtracted from the up-card value.

Only one submission per team is necessary, but please make sure to include both names at the top of your source code, as well as in the comments section when submitting the lab, so both people can get credit.