Blackjack, written in Java

by Nikola Viazmenski

13 February 2016

1 What is Blackjack?

Blackjack is one of the most famous gambling house card games of the modern day. Played widely across the world, the main objective is to reach a hand value as close to 21 as possible without going over. The eponymous blackjack hand is a hand that, in the first draw, yields a value of precisely 21, and usually has a higher payout than other winning hands in casinos. Blackjack's fame perhaps could come from its simplicity; it has very simple and easy-to-learn rules.

The main rules of Blackjack are as follows.

1. Reach a hand value as close to 21 as possible.
2. Do not 'bust', or go over 21.
3. Have a higher card value than your opponent - the dealer.

1.1 Playing the game

The game is started with dealing two cards to each player and two cards, one face down and the other face-up, to the dealer. A player would then decide if they want to 'hit' - add another card to their hand, or 'stand' - end their turn. When all players have either ended their turn or busted, the dealer reveals their second card and proceed to hit or stand as well. A special rule surrounds the dealer - they are not allowed to hit if their card value is 17 or over. After the dealer's turn ends, wins and losses are evaluated, and bet payouts (if any) are calculated and paid out.

1.2 Scoring

Scoring in Blackjack is a slight modification of the traditional card values of the international 52-card deck.

- Number cards are worth their face value.
- Face cards (Kings, Queens, and Jacks) are worth 10 points.

- Aces are worth either 1 or 11 points - if someone has a hand with an Ace that would bust on an Ace value of 11, the Ace's value is reduced from 11 to 1.

2 Preparatory methods

In the Java rendition of Blackjack, several preparatory methods and pieces of code are used to create the deck that is used for play, shuffling it to allow for appropriate random chance, and giving the cards the values they need for the game of Blackjack to be played correctly.

2.1 Deck creation

The deck is created by a script that first generates a larger array of size 52 for all of the cards, then sub-arrays of sizes 4 and 13 for the suits and the cards in each suit. The arrays are filled with a for-loop method. If the card indices match certain numbers within the suits, they are designated as face cards and are given the appropriate names. Otherwise, simply the number is added. A proper "of Suit" name is appended at the end of each card.

2.2 Deck shuffling

Deck shuffling is taken care of by the method 'shuffle', which operates by using a modified bubble-sorter algorithm. By adding an element of randomness with the method 'rand', the deck can be pseudo-shuffled an immense amount of times by switching the positions of a random set of two cards. Given enough shuffle operations (in practice, one thousand operations was sufficient), the deck can be fully randomized.

2.3 Card valuing

Card values are assigned using another method, 'value'. This is an overloaded method - it can accept single strings (cards), arrays of cards (static dealt hands), and ArrayList objects(dynamic arrays (hands) to which pieces of data of any kind (cards) can be added or removed.) This code refers to the 'aces' method, expanded on in Section 4.2, *Ace dual-value logic.* Cards are valued based on their first character - if it is a number, as assigned in the deck generation, it is given an integer value equivalent to that number. If it is a King, Queen, Jack, or 10, it is assigned a value of 10 using the

```
card.charAt()
```

method. If it is an ace, it is assigned a value of 11 with the same method - however, this value is subject to change. This change is detailed further in Section 4.2, *Ace dual-value logic.*

3 Blackjack gameplay

A further set of methods and pieces of code are used for dealing cards to the player and the dealer, hide the dealer's card, utilize the valuation methods described in Section 2 to calculate the values of the playing hands, allow for hits, stands, and busts, and decide who wins and loses.

3.1 Dealing cards

Dealing cards is dealt with (no pun intended) by the 'deal' method, which returns a card from the top of the deck - the amount of cards left in the deck is determined by the integer 'count'. This returned card is then added to an ArrayList object in the game, which acts as the 'dynamic hand' - a hand to which cards can be added or removed.

3.2 Gameplay

Blackjack gameplay is initiated as soon as the program is initialized. Cards are automatically generated, shuffled, and dealt. When the former processes are completed, the user is given information regarding their cards and their hand value, as well as the dealer's exposed card. They are then prompted to either hit (in which the user inputs an H) or to stand (in which the user inputs an S). When they hit, the 'deal' method adds another card to their hand, which is stored in an ArrayList object. Once the player ends their turn by choosing to stand, which triggers the change in a boolean value that decides whether the hit-loop should continue or not, the dealer's hand is revealed, the dealer plays (how the dealer plays is covered in Section 4.3, *Dealer automation*), their card value is evaluated, and the player is told that they have either won or lost.

4 Extended logic

The program also contains various methods of logic that allow for the more intricate rules of Blackjack to be successfully implemented and executed. This includes logic for win/loss evaluation, ace dual-valuing, and how the dealer's play is automated.

4.1 Win/loss evaluation

Win/loss evaluation is taken care of by a small logic piece. The main operation is written below.

```
if( (value(hand)>value(dealer_hand) && value(hand)<22) | (value(dealer_hand) > 21) ){
System.out.println( "You Win!");
}
else{System.out.println( "You Lose!");}
```

4.2 Ace dual-value logic

This section covers the overloaded method 'aces'.
The general function of the method is to implement Blackjack's rule surrounding the dual value of aces - they are worth a simultaneous 1 and 11, based on the hand. The generalization of the implementation is as follows. It ties slightly into the method 'value', described in Section 2.3, *Card valuing.*

1. Assign the ace value as 11 by default.
2. Utilize the 'aces' method to count how many aces are in a hand.
3. If a hand would bust with the ace value being 11, reduce the ace count by 1 and reduce the hand value by 10.

4.3 Dealer automation

The dealer's play is automated by a while loop. It will repeatedly hit for the dealer until the dealer's hand exceeds 17, in line with the rules of the game. The main action is written below.

```
while( value(dealer_hand)<17 ){
dealer_hand.add( deal(deck) );
}
```

5 Full source code – annotated according to section

```
import java.util.*;
import java.util.Scanner;

public class Cards{

static int count=52; //the count represents the number of cards remaining in the deck

// Section 2.2

public static int rand(int high){
return (int) (high*Math.random()+1);
}

public static void shuffle(String[] the_deck, int switches){
String temp;
int a; int b;
for(int i=0; i<switches; i++){
a = rand(52);
b = rand(52);
temp = the_deck[a-1];
the_deck[a-1] = the_deck[b-1];
the_deck[b-1] = temp;
}
}
// end Section 2.2
// Section 3.1
public static String deal(String[] the_deck){
count=count-1;
return the_deck[count];}
// end Section 3.1
// Section 4.2
public static int aces(String the_card){
if(the_card.charAt(0)=='A'){
return 1;}
else{
return 0;}
}

public static int aces(String[] the_hand){
int sum=0;
for(int i=0; i<the_hand.length;i++){
sum = sum + aces(the_hand[i]);
}
return sum;
}

public static int aces(ArrayList the_hand){
int sum=0;
for(int i=0; i<the_hand.size();i++){
sum = sum + aces(the_hand.get(i).toString());
}
return sum;
}
// end Section 4.2
// Section 2.3
public static int value(String the_card){
char first = the_card.charAt(0);
if (first=='1'|first=='J'|first=='Q'|first=='K'){
return 10;
}
else if(first=='A'){
return 11;}
else{
return Character.getNumericValue(first);
}
```

```
}

public static int value(String[] the_hand){
int sum=0;
for(int i=0; i<the_hand.length;i++){
sum = sum + value(the_hand[i]);
}
return sum;
}

public static int value(ArrayList the_hand){
int sum=0;
int num_aces=aces(the_hand);
for(int i=0; i<the_hand.size();i++){
sum = sum + value(the_hand.get(i).toString());
}
while(num_aces>0 && sum>21){
sum=sum-10;
num_aces=num_aces-1;
}
return sum;
}
// end Section 2.3


public static void main(String[] args){

Scanner scan = new Scanner(System.in);
// Section 2.1
String[] deck = new String[52];
String[] suit = new String[4];
int[] card = new int[13];

for (int i=0; i<card.length; i++){
card[i]=i+1;}
String cardName;
suit[0] = "Clubs";
suit[1] = "Diamonds";
suit[2] = "Hearts" ;
suit[3] = "Spades";


for(int i=0; i<4; i++){
for(int j=0; j<13; j++){
if(j==0){cardName="Ace";}
else if(j==10){cardName="Jack";}
else if(j==11){cardName="Queen";}
else if(j==12){cardName="King";}
else {cardName=Integer.toString(card[j]);}
deck[ 13*i+j ]= cardName + " of " +suit[i];
}
}
shuffle(deck, 1000);
// Section 3.2
String say;
boolean state=true;

ArrayList hand = new ArrayList();
ArrayList dealer_hand = new ArrayList();
dealer_hand.add( deal(deck) );
dealer_hand.add( deal(deck) );
hand.add( deal(deck) );

while(state){

hand.add( deal(deck) );

System.out.println("Dealer showing: " + dealer_hand.get(1));
System.out.println("Contents of hand: " + hand);
System.out.println("Your score is: " + value(hand));

if(value(hand)>21){
System.out.println("Bust!");
break;
}

System.out.println( "Hit[H] or stand[S]?");
say=scan.nextLine();
if(say.equals("H")){state=true;}
else{state=false;}
}
// Section 4.3
while( value(dealer_hand)<17 ){
dealer_hand.add( deal(deck) );
}


System.out.println("Dealer has: " + dealer_hand);
System.out.println("Dealer score is: " + value(dealer_hand));
// Section 4.1
if( (value(hand)>value(dealer_hand) && value(hand)<22) | (value(dealer_hand) > 21) ){
System.out.println( "You Win!");
```

```
}
else{System.out.println( "You Lose!");}
```