A Blackjack Game Program: Part 1 Loops

Introduction

Assignment Goals: The goal of this assignment is to write a program which plays a simplified version of Blackjack. Blackjack is a two player game between you and the dealer. The computer will be the dealer.

New Java Skills: Looping and Methods.

Game Rules (see http://www.pagat.com/banking/blackjack.html#objective)

- *Goal:* The aim of the game is to accumulate a higher point total than the dealer, but without going over 21. You compute your score by adding the values of your individual cards.
- *Card Values:* The cards 2 through 10 have their face value, J, Q, and K are worth 10 points each, and the Ace is worth either 1 or 11 points (player's choice).
- *Play:* At each round of the blackjack game, the players (you and the computer) receive two cards each. The score is computed to determine who wins.
- *Simplification:* In this assignment we will
    - not allow for additional cards to be dealt (hits). Do this for extra credit!
    - not worry about dealing duplicate cards, i.e. in your game, it will be possible to draw two aces of hearts. You can fix all of this later if you want, once we cover arrays.

As usual, we will use ***stepwise refinement***. In this first part, we will implement the parts of the program that require loops. In the second part (when we get to Chapter 6 Methods), we will look at how to put all of the parts together.

The Algorithm

Software development is always done in stages:

A. ***Understand what you are doing:*** Read the above rules. Get a deck of cards and play the game with someone!
B. ***Think about Output:*** What do you want the output to look like? This will determine much of the workings of the code. For example, see the sample output below (modify it if you want, but it probably should have the same basic structure)

C. ***Pseudocode***: Write the *algorithm* in pseudocode, i.e. the sequence of steps required to solve the problem, taking into consideration the sample output. For example (you fill in what is missing):

```
1. Print a welcome message to the game.
2. Prompt the user to find out if they want to play. Repeat if they enter an
   answer that is not allowed.
3. If they want to quit, end the program.
4. If they want to play the game:
     i.  deal the cards for player1
     ii. etc  // you fill in the rest …
5. repeat starting at step 2.
```

Implementing the Loops

You never want to write the code all at once. Instead, identify small tasks that can be independently implemented and thoroughly tested. Here, we will look at the tasks that require loops.

In the pseudocode, identify the input prompt loop and the game play loop.

A. ***Prompt Loop:*** Implement the prompt loop and test it. A do-while loop is probably the best type of loop to use (do you see why?).

The output might look like the following

```
Do you want to play blackjack? (y/n) maybe
Please answer y or n.
Do you want to play blackjack? (y/n) y
Great, let’s play the game.
```

Or, alternatively, it might look like

```
Do you want to play blackjack? (y/n) n
Game Over. Good bye.
```

Make sure this works before continuing. *Test every possible sequence of things that could happen.*

B. ***Dealing a Card:*** In class, we will go over how to randomly select a card. You will implement code that selects the card index and calculates the suit and face indices which are then used to generate a String name (using if-else).

This code will eventually go inside of your game play loop. To get the nice suit symbols, you can use Unicode as follows:

```
final int SPADES= 0;  // avoids magic number
…
if (suit == SPADES) {
        suitName = “\u2660";  // Unicode for a spade symbol
} else if ….
```

Make sure this works before continuing. *Test every possible sequence of things that could happen.*

C. ***Game Loop:*** Use a while-loop (do you see why?). The program should take the answer from the prompt and, if the user says yes, deal a card and print the result. Note, the prompt must be done twice - before you start the game loop (initialization) and again at the end of the game loop (to see if the while loop should be executed again). Later, we will see how to avoid all of this duplication of code.

The output should look something like:

```
WELCOME TO THE BLACKJACK GAME
Do you want to play blackjack? (y/n) maybe
Please answer y or n.
Do you want to play blackjack? (y/n) y
You have picked 8♥
Do you want to play blackjack? (y/n) y
You have picked 5♠
Do you want to play blackjack? (y/n) n
Game over. Good-bye.
```

Make sure this works before continuing. *Test every possible sequence of things that could happen.*

D. ***Clean-up:*** Clean up your code so it is nicely formatted. Use blank lines to separate sections of your code. As you might have noticed, the code is getting rather long and unwieldy. In chapter 5, we will see how to take the code and break it into reusable chunks which will make it 1) more readable, 2) have less duplication, 3) be more likely to be correct, and 4) be easier to edit.

This part of the code will not be collected. However, you will need it in order to do Part 2. If you want the instructor to look over it and give feedback, please ask. It is important that you feel confident about what you have done.

Sample Output:

```
WELCOME TO THE BLACKJACK GAME
Do you want to play blackjack? (y/n) maybe
Please answer y or n.
Do you want to play blackjack? (y/n) y

Player:  A♣  8♠     Score is 19
Dealer:  8♠  4♥     Score is 12

Congratulations!  You win.
Do you want to play blackjack? (y/n) y

Player:  K♣  1♦     Score is 11
Dealer:  6♦  A♦     Score is 17

Oops, sorry. The dealer wins.
Do you want to play blackjack? (y/n) n

Game over. Good-bye.
```