CS 101 - Algorithms & Programming I

Fall 2021 - Lab 5

Due: Week of November 1, 2021

*Remember the **honor code** for your programming assignments.*
*For all labs, your solutions must conform to the CS101 style **guidelines**!*
All data and results should be stored in variables (or constants where appropriate) with meaningful names.

The objective of this lab is to learn how to use for and do-while loop to implement automated repetition. Remember that analyzing your problems and designing them on a piece of paper *before* starting implementation/coding is always a best practice.

In this particular lab, **only use the for or do-while loops**, do *not* use the while loop.

0. Setup Workspace

Start VSC and open the previously created workspace named `labs_ws`. Now, under the `labs` folder, create a new folder named `lab5`.

In this lab, you are to have two Java classes/files (under `labs/lab5` folder) as described below. A third Java file containing the revision should go under this folder as well. We expect you to submit a total of 3 files including the revision. Do *not* upload other/previous lab solutions in your submission. The user inputs in the sample runs are shown with blue color.

1. City Builder

Create a new/empty file of your own under the `lab5` folder named `Lab05_Q1.java` with a class with the same name. You, a great architect, are selected by the mayor of Potato City to run a secret construction project. The mayor will input a **string** into your program composed of digits. Each digit within this string is actually the height of a building he wants you to build. Your task is to print a blueprint of the buildings using the asterisk symbol (*) and **for** loops. Do not use while or do-while loops in this part. You may assume that the input string is valid, consisting of only digits.

Sample run:

```
Please enter city plan string: 245203157
City blueprint:
         *
         *
  *     **
 **     **
 **   * **
**** * **
**** ****
```

2. Blackjack

Create a new/empty file of your own under the `lab5` folder named `Lab05_Q2.java` with a class with the same name. Your program will be a simplified version of the popular card game Blackjack21. Even if you are familiar with the game, you should still read the instructions thoroughly, as there are quite a bit of simplifications from the original game. A couple of sample runs of the game are added at the end of the explanations. You can refer to the pictures on the last pages (under "Optional Assignment") to get a better understanding of the game.

Our version of Blackjack21 is a 1-on-1 card game, the player versus the dealer. Details of the card deck are given in section 2.1. At the beginning of the game, both the player and the dealer are dealt 2 cards each. We see both of our cards, but see only 1 of the dealer's cards. The aim of the game is to have the sum of the card values (score) in your hand be 21, or as close to 21 as possible. However, you lose if you go over 21.

The player goes first and has two choices: Hit or Stand. Hit means the player wants to pick another card. You make this move to increase your score. Stand means that the player is done picking new cards (i.e. Hitting) and their turn ends. You do this move when you are comfortable with your score (e.g. score=19 is very close to 21, picking another card could make you lose the game by going over 21). When the player's turn ends, if the player hasn't already scored 21 and won the game, the dealer starts Hitting until they either score 21 (Dealer win), score bigger than the player (Dealer win) or go over 21 (Dealer loss).

The strategy is to increase your score by hitting to get close to 21 (but not over), while also taking into consideration what the dealer's score could be.

Remark: our implementation of the game is slightly biased towards the dealer winning, but don't let that discourage you!

The game can be summarized with the following steps, each of which will be detailed in the following sections:

1. Deck creation & initial card draw.
    a. Create deck
    b. Do the initial card draw for the player and the dealer (a total of 4 cards)
    c. Remove drawn cards from the deck
2. Player's turn
    a. Hit or Stand
    b. Make decisions and possibly end (win with 21 or lose by exceeding 21) the game
3. Dealer's turn (Need not happen if the game ends in step 2.b)
    a. Hit
    b. Make decisions and end the game

2.1. Deck creation & initial card draw

- First, construct a **string** that will be used in place of a card deck. A standard card deck is composed of 13 ranks (Numbers through 1 to 10, Jack, Queen and King) in 4 suits (categories like clubs, diamonds, etc.), for a total of 52 cards. For simplicity, our deck includes only positive digits (1-9) and we also don't care about the suit of the card. All in all, we are left with a 36-card deck where each digit/card is included 4 times. Use a **for** loop to create a string corresponding to a deck and print it:

```
Starting the game with the following deck:
111122223333444455556666777788889999
```

- At the start of each game loop, the dealer randomly deals 2 cards for the player and 2 cards for themselves, for a total of 4 cards. Use **for** loop(s) to randomly select 2 cards for both the player and the dealer. Note that we also have to remove the selected cards from our deck (i.e. we are doing selection without replacement).
  Tip: "Removing" from a string can be mimicked by re-creating the string without including the character at the index you want to remove. To randomly select a card you can use the following code (assuming there are 5 cards in the deck):

```
import java.util.Random; // Similar to what we do with the Scanner
...
...
Random rand = new Random(); // Again, very similar to Scanner
int randomChoice = rand.nextInt(5); // get a random number in range [0,5)
```

- You should print the selected cards. Note that we are hiding the last card of the dealer when we are printing it. Here, `45` means the player holds cards 4 & 5 and `7?` means the dealer holds cards 7 & ? (unknown to us):

```
Dealer is now dealing cards...
Player's hand:
45
Dealer's hand:
7?
```

2.2. Player's turn

- After our initialization is complete, we now start the *core game loop*. In our Blackjack game, the player (controlled via your inputs) has two choices: Hit or Stand. Present these two choices as a menu to the player. Note that this menu has to be presented *at least once* to the user:

```
Please enter your choice:
1) Hit
2) Stand
```

- If the player enters 1 to hit, the player is given another card. Depending on the score the player has after hitting, there are 3 different paths we can take. Last 2 paths should end the game, while the first path should prompt the user again for their choice. Report these actions as in the following examples:

  The game goes on:

```
1
Hit! Dealer is giving the player a card...
Player's hand:
459
```

  Player wins:

```
1
Hit! Dealer is giving the player a card...
Player's hand:
45921

Player scored 21. Player wins!
```

  Player loses:

```
1
Hit! Dealer is giving the player a card...
Player's hand:
4598

Player scored over 21. Player lost!
```

- If the player enters 2 to stand, their turn ends and the sum of their cards are reported:

```
2
Stand! Player's turn is now over. Player's hand sums to 17.
```

2.3. Dealer's turn

- After the player's turn ends, the game may end. If not, we move on to the dealer's turn. Dealer's turn consists of Hitting *at least once until an ending condition is satisfied*. The dealer hits until they either score 21, score bigger than the player or go over 21. Report these actions as the following. Note that we are no longer hiding the dealer's second card:

  Dealer wins (case 1):

```
Dealer is drawing cards...
Dealer's hand:
753
Dealer's hand:
7535

Dealer scored more than player. Dealer wins!
```

  Dealer loses:

```
Dealer is drawing cards...
Dealer's hand:
753
Dealer's hand:
7535
Dealer's hand:
75356

Dealer scored over 21. Dealer lost!
```

  Dealer wins (case 2):

```
Dealer is drawing cards...
Dealer's hand:
753
Dealer's hand:
7536

Dealer scored 21. Dealer wins!
```

A couple of sample runs for the whole game are presented below. Here, we additionally print the last state of the deck when the game ends. You are advised to do this (not required) to make sure your card selection and removal works as intended. You can also (not required) print the score after each draw (e.g. `Player's hand (13):` instead of just `Player's hand:`), so you don't have to sum the cards in your head while debugging:

```
Starting the game with the following deck:
11112222333344445555666677778888999 9

Dealer is now dealing cards...
Player's hand:
81
Dealer's hand:
4?

Please enter your choice:
1) Hit
2) Stand
1

Hit! Dealer is giving the player a card...
Player's hand:
818

Please enter your choice:
1) Hit
2) Stand
2

Stand! Player's turn is now over. Player's hand sums to 17.

Dealer is drawing cards...
Dealer's hand:
442
Dealer's hand:
4423
Dealer's hand:
44237

Dealer scored more than player. Dealer wins!

Ending the game with the following deck:
11122233344555566667778899999
```

```
Starting the game with the following deck:
11112222333344445555666677778888999 9

Dealer is now dealing cards...
Player's hand:
16
Dealer's hand:
8?

Please enter your choice:
1) Hit
2) Stand
1

Hit! Dealer is giving the player a card...
Player's hand:
168

Please enter your choice:
1) Hit
2) Stand
1

Hit! Dealer is giving the player a card...
Player's hand:
1686

Player scored 21. Player wins!

Ending the game with the following deck:
1112222333344445555667777889999
```

(Optional Assignment, Do Not Submit) Blackjack Graphical User Interface:

If you are done with the Blackjack program of your lab, you can see the gameplay in the following user interface with minimal code changes!:

First, download the GUI folder from Moodle. Then, put the contents of the folder in the same directory as your Java file (i.e. your Java file, downloaded png images and downloaded Java files should be in the same folder). Do the following changes to use the UI:

- Add the following line as the very first line of your main function. This should open up an interface similar to the picture you see above:

```
GUI gui = new GUI();
```

- After the first time you draw cards for the player and the dealer, put the following line:

```
gui.runGUI(playerHand, dealerHand);
```

- Then, each time you add a card to the player or the dealer, use one of the following lines:

```
gui.updatePlayerHand(playerHand);
or
gui.updateDealerHand(dealerHand);
```

- You can use the following line after you give game-ending decisions:

```
gui.alert("Dealer scored over 21. Dealer lost!");
```

If all goes well, you will be playing the game from the terminal as you did before, but the graphical interface will update according to the game state. Please keep in mind that this is an optional assignment for the curious and you are not required to do it. Your TAs will not ask or answer questions regarding this part. The GUI code was written by https://github.com/uzaymacar but was adapted for this lab by E. Batuhan Kaynak.