Assignment 03: Blackjack

CS101 - Intro To Computer Science

Spring 2024

In this assignment, we develop a modified version of Blackjack. You will learn to work with random numbers and implement a nontrivial program flow, and our modifications are both to simplify the algorithm and to provide some extra practice.

Prerequisites:

- Branching
- Loops

1. Project Setup

The zip file contains a folder called `03_blackjack`, which in turn contains a VS Code project. Extract the folder from the zip file and open it with VS Code to begin implementing and running your assignment. The folder contains the skeleton code for your submission.

2. Blackjack

Blackjack is a card game where players draw cards to try to obtain the highest total value without exceeding 21. The player with the largest card value wins, but if a player has more than 21, they "bust" and instantly lose.

At the beginning of each game each player draws two cards. A player can then request a card by saying “hit”, and can continue to say “hit” until they decide they have obtained enough cards. If they have enough cards they say “stop”.

We will play a simple version with only two players, the user and the dealer. The player will first assert their moves by typing them into the terminal and the dealer will play according to a random algorithm. In particular, the dealer will randomly choose between a “hit” or “stop” with an equal probability for either option.

The cards in our deck will take on integer values from 2 through 11 (including 2 and 11). We will assume that our deck has an infinite number of each card, so do not worry about drawing the same card too many times.

Implement the following flow for the game:

2.1. Two cards are dealt to the user.

2.2. Two cards are dealt to the dealer.

2.3. The user’s cards are displayed on the terminal.

2.4. The user is prompted whether they would like to hit. If they do so, they obtain another card.

2.5. If the overall value of their cards exceeds 21, then the user has "busted". They have therefore lost and the game is finished.

2.6. As long as the user continues to hit, the prompt will continue to be repeated and each time the user obtains a new card their sum is checked for being a bust, and if they bust then they lose and the game is finished.

2.7. As soon as the user chooses "stand" we let the dealer play.

2.8. The dealer plays by choosing to "hit" or "stand" at equal probability. If the dealer chooses "hit" they obtain a new card. If the dealer’s cards exceed 21 points during this process, the dealer has busted and therefore lost and the game is finished.

2.9. Once the dealer has chosen "stand" the game is finished.

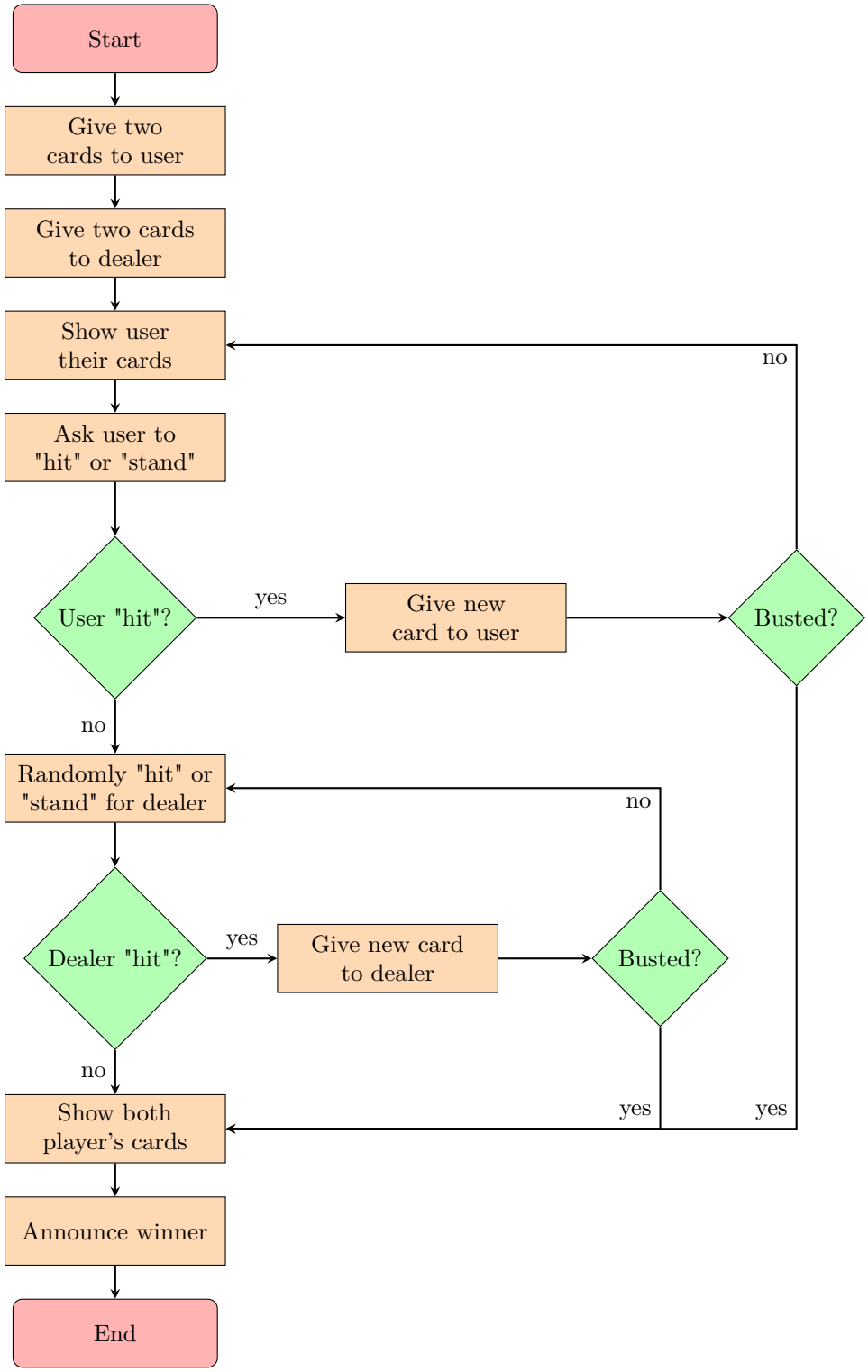
2.10. Once the game is finished the players (dealer and user) show their cards.

2.11. If the player busted then the dealer wins and if the dealer busted then the player wins.

2.12. If neither player busted then the player with the higher total value wins.

2.13. Regardless of how the game ends, whether by a bust or by a win, the program must show the user what cards they had and what cards the dealer had at the end of the game and announce the result of the game. Either there is a winner, a tie, or one of the players has bust.

A flow chart for the game is shown on the next page and after that there is a sequence of example outputs that your program should match.


Start
Give two cards to user
Give two cards to dealer
Show user their cards
Ask user to "hit" or "stand"
User "hit"?
yes
Give new card to user
Busted?
no
no
Randomly "hit" or "stand" for dealer
Dealer "hit"?
yes
Give new card to dealer
Busted?
no
no
yes
yes
Show both player's cards
Announce winner
End

Here are some sample outputs. Your program should be able to provide similar sequences of outputs. Be careful to match the capitalization and spaces of the various lines.

- user stands, dealer wins

```
Welcome to Blackjack!
Your cards are: 2, 4
Would you like to hit or stand?
stand
The dealer stands.
Your cards are: 2, 4
The dealer's cards are: 8, 7
Dealer wins!
```

- dealer wins

```
Welcome to Blackjack!
Your cards are: 3, 7
Would you like to hit or stand?
hit
Your cards are: 3, 7, 4
Would you like to hit or stand?
hit
Your cards are: 3, 7, 4, 2
Would you like to hit or stand?
stand
The dealer hits.
The dealer hits.
The dealer hits.
The dealer stands.
Your cards are: 3, 7, 4, 2
The dealer's cards are: 8, 3, 4, 3, 3
Dealer wins!
```

- user wins

```
Welcome to Blackjack!
Your cards are: 8, 2
Would you like to hit or stand?
hit
Your cards are: 8, 2, 8
Would you like to hit or stand?
stand
The dealer hits.
The dealer stands.
Your cards are: 8, 2, 8
The dealer's cards are: 3, 3, 7
You win!
```

- tie

```
Welcome to Blackjack!
Your cards are: 5, 6
Would you like to hit or stand?
hit
Your cards are: 5, 6, 3
Would you like to hit or stand?
stand
The dealer hits.
The dealer hits.
The dealer stands.
Your cards are: 5, 6, 3
The dealer's cards are: 2, 6, 2, 4
Tie!
```

- dealer busts

```
Welcome to Blackjack!
Your cards are: 5, 6
Would you like to hit or stand?
hit
Your cards are: 5, 6, 3
Would you like to hit or stand?
stand
The dealer hits.
The dealer has bust!
Your cards are: 5, 6, 3
The dealer's cards are: 8, 8, 10
You win!
```

- user busts

```
Welcome to Blackjack!
Your cards are: 5, 6
Would you like to hit or stand?
hit
Your cards are: 5, 6, 3
Would you like to hit or stand?
hit
You have bust!
Your cards are: 5, 6, 3, 10
The dealer's cards are: 8, 7
Dealer wins!
```

3. Please note:

3.1. In `Blackjack.java`'s main method, a `Random` object is already provided. Use it to generate the random numbers you need in the assignment.

3.2. Use `java.util.Random.nextInt()` to draw cards and `java.util.Random.nextBoolean()` to make decisions for the dealer (`true` implies that dealer hits).

3.3. How you store the player's cards is up to you. You are allowed to use data structures not yet covered in the lecture as well.

3.4. Make sure to stick to the flow chart. For example: give the user the cards first, then the dealer, not the other way around.

4. Unit Unit Tests

This assignment includes unit tests to make sure that your code conforms to our expected set of methods. Unit tests are used for a variety of reasons, but two key reasons are what we are doing here, to define how code should be invoked, and also to make sure that code continues to perform consistently over time. The unit tests will only build and run AFTER you have defined the various methods required for this assignment. Unit tests can be run by following the instructions in Figure 1. Note that these unit tests are VERY similar to how your assignments are graded - the autograder we use is implemented by running similar unit tests and giving marks if you pass them.

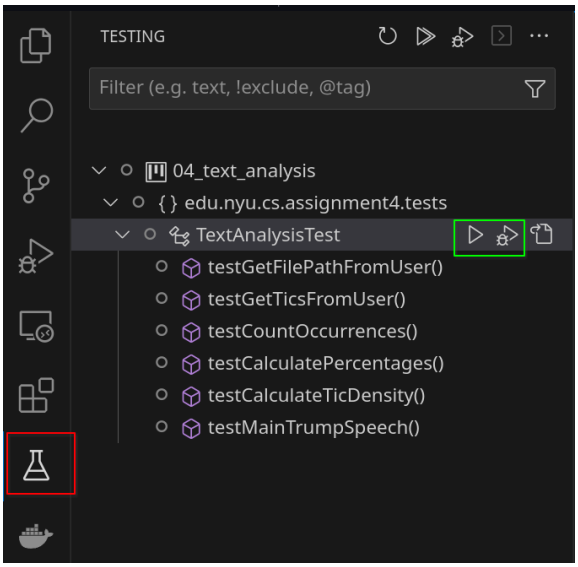


Figure 1: The red square indicates the Test tab and the green square indicates how to run or debug the tests. If you highlight the individual you should see play/debug buttons appear next to them. If you press those buttons you will run just a single unit test.

5. Submission

Submit the following files:

- Blackjack.java