Project 3 - Twenty-One Card Game

CSC-121, Fall 2025

Overview

You are going to write a computer version of the classic card game "Twenty-One", also known as "Black Jack". There will be two players: one human-controlled player, and a computer-controlled "Dealer" player.

Traditional Rules of Twenty-One

The game is played with a standard 52-card Poker deck of playing cards. All players are initially dealt two cards, face down. These cards can be seen by the player that owns them, but are kept secret from other players. The dealer places one of their cards face up so that all of the other players can see it, and the dealer's other card is kept face down and secret from the other players.

Each player takes a turn where they look at their cards and decide whether to "hit" (Draw another card, face down), or "Stand" (Stop drawing cards). A player may hit as many times as they like on their turn, unless the total value of their hand reaches or exceeds 21. If a hand value is ever greater than 21, that player "busts" and loses. The dealer's turn is last. The dealer reveals all of their cards when their turn begins, and draws cards face up when they hit.

After the dealer's turn ends, all players reveal their hands. The player whose hand value is closest to 21 without going over is the winner.

Our Game's Rules

Our game will consist of two players: A human player (Here after referred to as just "the player") controlled via standard input through the terminal, and an AI controlled "dealer". At the beginning of the game, both the player and the dealer are dealt two cards each. The player will take their turn first and can see *one* of the dealer's cards, but the player's cards are kept secret from the dealer until the end of the game. (If you prefer, you can choose to only give the dealer a single face up card at the beginning of the game instead of giving the dealer one face up card and one face down card.)

The player can choose to either hit or stand, or let an AI pick for them. The player may hit as many times as they like. If the player busts, the dealer automatically wins. If the player stands, the dealer's turn begins.

The dealer will continue to hit until their total is 17 or greater. If the dealer busts, the player automatically wins. If the dealer stands, then whoever's hand total is highest is the winner. If both the player and the dealer both have the same score, the game is a "push" and there is no winner.

Hand Value

Hand value is determined by what cards a player has in their hand. Each Poker card has a suit and rank. Suits include "hearts", "diamonds", "clubs", and "spades". Suits are not used in Twenty-One, and so are ignored. Rank includes the numbers 2 through 10, the King, Queen, and Jack picture cards, and Ace. The number ranks (2-10) are worth their face value, so a 2 of Spades is worth 2 points, a 5 of Hearts is worth 5 points, etc. Picture cards (K, Q, J) are worth 10 points each. Each Ace card (A) in a player's hand is worth either 1 point or 11 points, whichever value gives the player a better score and prevents busting.

Examples

Hand: King of Diamonds, Ace of Spades = 10 + 11 = 21 points

Hand: 2 of Hearts, 8 of Hearts, Queen of Clubs = 2 + 8 + 10 = 20 points

Hand: Ace of Hearts, 7 of Spades, Ace of Clubs = 11 + 7 + 1 = 19 points
(In this hand, one Ace counts for 11 points, while the other Ace counts for 1 point.)

Hand: Jack of Diamonds, 4 of Spades, 10 of Hearts = 10 + 4 + 10 = 24 points (Bust!)

Card Representations

Cards are represented in our game as a String consisting of a single character for the suit, followed by an underscore, and then one or two characters for the rank. Suits use lowercase letters, while ranks use numbers and capital letters. The deck and player hands are represented as a String with all cards separated by a space.

Ex. The 2 of spades = `"s_2"`, the 10 of diamonds = `"d_10"`, and the ace of hearts = `"h_A"`.

Ex. A player's hand containing the ace of clubs, 3 of diamonds, and jack of clubs would be: `"c_A d_3 c_J"`.

Student Requirements and Tasks

Students must download the template file as a starting point. The template has a few methods already written for creating and shuffling the starting deck of 52 cards, and a few empty method stubs that must be filled in. Students must follow the instructions below to add in the missing functionality.

Students are expected to only use techniques discussed thus far in class up to Chapter 5 material, including writing methods. Students should avoid using topics we have not yet covered in class including arrays, lists, and additional classes. These topics will be covered later, and will be the focus of future projects.

For each of the objectives below, find the following //TODO comment in the sample code and implement the feature stated. You may tackle the tasks in any order, though we recommend students address them in the numbered order below. The documentation comments have suggested names for the input arguments, but you can use other names if you like (Make sure to update the Javadoc comment with the new name). Once you complete a method, test it by calling it with test data and seeing if the return value is what you expect.

1. //TODO: Write evalHand method.
   The evalHand method should take in a String of cards like "h_A d_Q" and return the best possible score of the hand as an int. The "best" score is the highest possible score the hand can be without busting. So, a hand with "h_A d_Q s_J" would return 21 (Ace counting as 1 point) rather than 31 (Ace counting as 11 points). If the hand will bust no matter what the variable-valued cards are scored as, then you may return any possible score for the hand. So, evalHand("h_A d_Q s_J c_A") could return 22, 32, or 42.

2. //TODO: Write getTopCard method.
   The getTopCard should take in a set of cards as a String (This could be the deck or a player's hand) and return the first card as a String. So, if the set of cards passed in was "s_2 h_5 c_Q c_10" the returned value would be the String "s_2".

3. //TODO: Write removeTopCard method.
   The removeTopCard method should take in a set of cards as a String (This could be the

deck or a player's hand) and return a new String with the first card removed. So, if the set of cards passed in was "s_2 h_5 c_Q c_10" the returned value would be the String "h_5 c_Q c_10".

4. // TODO: Write addCardToHand method.
   This method should take as input a String card and a String hand of cards. It should output a new String hand containing all of the cards of the input hand plus the input card. The card order of the new hand is not important. So, if the input card is "h_8" and the input hand is "c_2 d_J", the output hand could be "c_2 d_J h_8", or any re-ordering of those same cards. If the input card is "s_7" and the input hand is "" (Empty hand, no cards), the output hand would be "s_7".

5. //TODO: Write playerShouldHit method.
   If the player asks the AI for help on their turn (The "Play for me" option), you should call the playerShouldHit boolean method. It takes in the player's hand and the dealer's visible card as input, and returns either True if the player should hit or False if the player should stand. Use the supplied input cards to decide for the player what they should do.

   You should also modify the documentation comment where it says "Strategy: N/A" and explain the strategy your method makes with its decision. When does it decide to hit or stand, and why? Make sure to also explain this strategy in your video.

6. //TODO: Write main method game logic here.
   Your game logic should all go here in your main method. Write logic to play one round of Twenty-One, with the player's turn first and the dealer's turn last.

   Deal two cards to each player to start the game, and reveal only one of the dealer's cards to the player. Present the player with 3 options: "Hit", "Stand", or "Play for me". If the player's total is exactly 21 (Even with their starting hand), they should automatically stand without being presented the options.

   When the player stands, the dealer takes their turn. The dealer hits until their hand total is 17 or greater, then stands. If either the player or dealer busts on their turn, the game immediately ends with that player losing. (HINT: You can use the `return` statement to end a `void` method early.)

   You should occasionally print out the player's hand, the visible dealer's cards, and values of both the player's hand and dealer's hand. How you format and print the information, and when you choose to print it, is up to you. You should test playing the game yourself to see when presenting this information is most useful for the player.

   In your game logic, you should call the other methods you have written. I.E. Whenever you need to get the score of a player's hand, you should call `evalHand` with that player's hand as the input argument. Whenever you need to add a card to a player's

hand, you should call `addCardToHand`. Try to call methods whenever it seems appropriate and useful rather than duplicating code.

** Extra Credit **

After completing the main objectives above, copy your code into a new .java file and class named TwentyOneJokers. In this new file, add two joker cards into the deck. Jokers are wild. Each joker card can represent any value from 1 through 11, whatever value gives the player a better hand score.

Change whatever code you think is necessary for jokers to work in the game, including the pre-written methods for constructing the deck.

If you complete this part of the project, submit both your TwentyOne.java and TwentyOneJokers.java source files. ***A working jokers program is worth 1 point of extra credit on Midterm 2.***

Examples

The hand Ace, 5, Joker should return 21 (Ace counting as 11, Joker counting as 5).

The hand Queen, 8, Joker, Joker should be worth 21 points (Jokers counting as 1 and 2 points).

Example Program Output 1

```
Java
Player's Hand: d_5 c_2

Dealer's Hand: c_4 ?

Player's Total: 7, Dealer's Total: 4

** PLAYER: Choose an action. **

(1) Hit

(2) Stand

(3) Pick for me

1

Player's Hand: d_5 c_2 d_A

Dealer's Hand: c_4 ?

Player's Total: 18, Dealer's Total: 4

** PLAYER: Choose an action. **

(1) Hit

(2) Stand

(3) Pick for me

2

Player's Final Hand: d_5 c_2 d_A

Player's Total: 18

Dealer's Hand: c_4 d_8

Player's Total: 18, Dealer's Total: 12

Dealer's Hand: c_4 d_8 s_5

Player's Total: 18, Dealer's Total: 17

Dealer's Final Hand: c_4 d_8 s_5
```

```
Dealer's Total: 17

Player's Total: 18, Dealer's Total: 17

Player wins.
```

Example Program Output 2

```
Java
Player's Hand: d_2 d_A

Dealer's Hand: s_7 ?

Player's Total: 13, Dealer's Total: 7

** PLAYER: Choose an action. **

(1) Hit

(2) Stand

(3) Pick for me

3

Player's Hand: d_2 d_A d_5

Dealer's Hand: s_7 ?

Player's Total: 18, Dealer's Total: 7

** PLAYER: Choose an action. **

(1) Hit

(2) Stand

(3) Pick for me

3

Player's Final Hand: d_2 d_A d_5

Player's Total: 18
```

```
Dealer's Hand: s_7 c_7

Player's Total: 18, Dealer's Total: 14

Dealer's Hand: s_7 c_7 h_6

Player's Total: 18, Dealer's Total: 20

Dealer's Final Hand: s_7 c_7 h_6

Dealer's Total: 20

Player's Total: 18, Dealer's Total: 20

Dealer wins.
```

Tips & Tricks

- If you're stuck on where to start with writing the main method game logic, try calling and printing out the return values from the `makeDeck()` and `makeShuffledDeck()` methods.

- Understanding the format of the cards will help you write the other methods.

- Methods in Java can only return one value. To mimic the action of drawing a card from the deck, you'll need to first call `getTopCard(deck)` to get the card on the top of the deck, and then call `removeTopCard(deck)` to make a new deck String with that top card removed.

- Did you know that the Scanner object can scan a String? It's true! Also, the default delimiter for tokenizing input with scanner.next() is a blank space character (" ").

- You can play an online free version of Twenty-One/Back Jack to learn the rules of the game. Here's one:
  https://www.arkadium.com/games/blackjack/

Notes on genAI usage:

You are encouraged to use an AI tool while writing your code. For example, you may want to ask it to explain a concept related to your program, such as what operator to use to check if two values are equal. Or, if you find you have a syntax error, you could leverage genAI as part of your debugging process. However, you should not trust AI tools blindly, or copy and paste AI-generated code without understanding how it works. You should not use concepts that we have not yet covered in class, even if AI suggests them. Keep in mind that you will need to explain your code on video, and must understand the code to do so.

Explain your work

Manual

In addition to your project code .java files and video, you must write a short manual to accompany your project. Name your plain text manual "README.txt". Include any information you deem necessary to running and using the program.

Video

A final part of the project is to explain your work and demonstrate your understanding of the program you created.

You will share your explanation in a 5-minute video after creating a detailed outline of the points that you want to make. Specifically, you must explain the following:

- The intended functionality of the program you decided to create and why you chose to create that program
- Your process and the steps you took to design and develop this program
- How the code executes in one of your most complex conditional statements
- The lessons you learned through this project

Each part should be at least 1-minute long. You can use a standard cellphone or webcam to record your video. If you use a screen recording program, we still need to be able to hear your voice explaining the code and project.

Please submit your video using a standard video format, like .mp4 format. If you are unable to submit the video file along with your submission on canvas for any reason, you may instead upload the video to a free service such as YouTube or Vimeo and include a URL link to the video with your project submission in a plain text file named "video.txt". Your video must remain accessible to grading staff until grading is complete.

Grading - Deliverables and Evaluation Criteria

You will be responsible for the following deliverables. The weighting of each component is indicated below. It's important that you refer to the Project 1 Rubric for detailed information about how each deliverable will be evaluated.

Checkpoint (10%)

Each student must meet with the professor or a TF in an in-person meeting before the project deadline to discuss their project, and any problems they may be encountering. Feedback will be offered to help students with building their project. Be sure to ask the professor/TF to count your project check-in.

Code submission (60%)

You must submit your code by the due date. *Please ensure it is possible to successfully compile and run your Java program!* If any additional instructions are required to run your project, please include them in your manual.

Manual (10%)

You must submit a plain text manual file explaining how to use your program.

Video Explanation (20%)

You must submit your video (Or a plain text file with a link to your video) containing your explanation along with your detailed outline by the due date.