Bluejack

SE 115 Project Description

Introduction

Bluejack is a card game played by two people. It is similar to Blackjack, but with different types of cards, which will be described shortly. Please read this description carefully, it is written to answer all of your questions.

The implementation of your Bluejack game will be played by **two players**: the computer versus the human player.

The Game

Bluejack is played by three decks of cards. The **game deck** is made up of four sets of cards that range between 1 and 10. The sets are colored blue, yellow, red, and green. The other two decks are called **player decks** which will have 10 cards each. They are formed as follows:

- The game deck is shuffled.
- Repeat the following 5 times:
    - The card at the top is given to the computer,
    - The card at the bottom is given to the user.

After this first step, both player decks have 5 cards. The remaining five cards are randomly generated as follows:

- Three cards with random colors and values between 1 to 6, with an additional sign, that is either a plus (+) or a minus (-). These cards are either positive or negative, depending on the random sign.
- The remaining two cards have an 80 percent chance of being a signed card. However, if the user is lucky (20 percent!), then:
    - One card can be a flip (+/-) card. Flip cards change the sign of the last played card.
    - One card can be a double (x2) card. Double cards double the value of the last played card.
    - Flip cards and double cards do not have a color, they only change the values or the sign of the last played card.

The chance to get one flip and one double card is 0.2*0.2 = 0.04: that is 4 percent!

Now that the player decks are ready, only 4 of these cards are randomly picked as the hand of the player.

The purpose of the game is to get a score of 20 using cards from the main and player decks. The cards in the hands of the players can be used when necessary. The first player to win three sets is the winner. However, if one of the players uses all blue cards to get a score of 20, they automatically win the game. Hence, the name Bluejack.

Since the computer deals the hands, the player starts the game. The game is played as follows:

- The current player asks for a card from the game deck to be placed on their board.
- The player can either **end** their turn, or depending on the sum of the cards in their board can choose to **stand**, or play one of the cards in their hand. Playing a card also ends the turn.
- If the player chooses to stand, it cannot draw any more cards from the deck, and waits for the opponent to be done.
- The player who is closest to but not over 20 wins the set.
- If a player is over 20 once their turn is over, then the player automatically loses the set. This is called a **bust**.

The board for each player can hold up to 9 cards.

A player wins the game in the following cases.

- Once both players stand, the player closest to but not over 20 wins the set.
- The player wins if the other player busts.
- If the player's board is full (that is: 9 cards are placed on it) and their sum is less than or equal to 20, then the player wins.

It is possible that two players stand at the same score. In such a case the game is tied, and no player wins the set.

A Sample Game

Since there is no real Bluejack deck, here is a sample game of three sets.

Before the game begins, the game deck is shuffled, player decks are generated and four cards from the player decks are randomly picked for each player.

Computer Hand	X X X X (we cannot see their hand)	This is the beginning of the game. The cards are dealt. We have a red +4, a green +2, a red -1, and a flip card. We will use the flip card when necessary. Now it is our turn, so we get the top card from the game deck.
Computer Board	Empty	
Player Board	Empty	
Player Hand	+4 +2 -1 +/-	

Computer Hand	X X X X (we cannot see their hand)	We get a blue +4 card. Since the game deck is made up of only positive values, we will end our turn, and let the computer take a card.
Computer Board	Empty	
Player Board	+4	
Player Hand	+4 +2 -1 +/-	

Computer Hand	X X X X (we cannot see their hand)	The computer gets a yellow +10 and ends its turn.
Computer Board	+10	

Player Board	+4	
Player Hand	+4 +2 -1 +/-	

Computer Hand	X X X X (we cannot see their hand)	We get a green +10! Our score is 14. Even if we bust, we can either use our -1 card, or our flip card, to turn things around. So let's end our turn.
Computer Board	+10	
Player Board	+4 +10	
Player Hand	+4 +2 -1 +/-	

Computer Hand	X X X X (we cannot see their hand)	Computer gets a green +6 card, and so the score of the computer is now 16. So far, no player played any of their cards in their hands.
Computer Board	+10 +6	
Player Board	+4 +10	
Player Hand	+4 +2 -1 +/-	

Computer Hand	X X X X (we cannot see their hand)	We get a green +5 card, and we are now 19. This is where we **stand**, hoping that the computer does not make it to 20.
Computer Board	+10 +6	
Player Board	+4 +10 +5	
Player Hand	+4 +2 -1 +/-	

Computer Hand	X X X X (we cannot see their hand)	The computer gets a yellow +6 card, which makes it 22, but it still has a chance to save it if any of its cards can turn things around. We wait for the computer to complete its turn.
Computer Board	+10 +6 +6	
Player Board	+4 +10 +5	
Player Hand	+4 +2 -1 +/-	

Computer Hand	X X O X (we cannot see their hand)	The computer uses a red -2 card from its hand, and now the computer has a score of 20. We lose this set. Kaya: 0, CPU: 1
Computer Board	+10 +6 +6 -2	
Player Board	+4 +10 +5	
Player Hand	+4 +2 -1 +/-	

Computer Hand	X X O X (we cannot see their hand)	The boards are cleared for the second set. We continue to draw cards from the game deck. The CPU has used one of its cards. We still have four. We begin by drawing the next card from the game deck.
Computer Board		
Player Board		
Player Hand	+4 +2 -1 +/-	

Computer Hand	X X O X (we cannot see their hand)	We start with a yellow +9 card, and end our turn.
Computer Board		
Player Board	+9	
Player Hand	+4 +2 -1 +/-	

Computer Hand	X X O X (we cannot see their hand)	The computer gets a red +5 card and ends its turn.
Computer Board	+5	
Player Board	+9	
Player Hand	+4 +2 -1 +/-	

Computer Hand	X X O X (we cannot see their hand)	After getting a green +5 card, we now have a score of 14 and end our turn.
Computer Board	+5	
Player Board	+9 +5	
Player Hand	+4 +2 -1 +/-	

Computer Hand	X X O X (we cannot see their hand)	After getting a red +10 card, the computer now has a score of 15 and ends its turn.
Computer Board	+5 +10	
Player Board	+9 +5	
Player Hand	+4 +2 -1 +/-	

Computer Hand	X X O X (we cannot see their hand)	We get a green +7 card. We have a score of 21, but now we can play our red -1 card.
Computer Board	+5 +10	

Player Board	+9 +5 +7	
Player Hand	+4 +2 -1 +/-	

Computer Hand	X X O X (we cannot see their hand)	Our score is now 20. We did not win, however, we will wait for the computer to get a card.
Computer Board	+5 +10	
Player Board	+9 +5 +7 -1	
Player Hand	+4 +2 +/-	

Computer Hand	X X O X (we cannot see their hand)	The computer also scores 20, the set is **tied**. We begin a new round.
Computer Board	+5 +10 +5	
Player Board	+9 +5 +7 -1	
Player Hand	+4 +2 +/-	

Computer Hand	X X O X (we cannot see their hand)	The board is cleared, and I get a green +10, then the CPU gets a red +7.
Computer Board	+7	
Player Board	+10	
Player Hand	+4 +2 +/-	

Computer Hand	X X O X (we cannot see their hand)	I get a green 8, and reach 18. Now I can use my green +2.
Computer Board	+7	
Player Board	+10 +8	
Player Hand	+4 +2 +/-	

Computer Hand	X X O X (we cannot see their hand)	I have 20, but we will wait for the CPU.
Computer Board	+7	
Player Board	+10 +8 +2	
Player Hand	+4 +/-	

Computer Hand	X X O X (we cannot see their hand)	CPU busts, and I win. Kaya: 1, CPU: 1
Computer Board	+7 +10 +4	
Player Board	+10 +8 +2	
Player Hand	+4 +/-	

The game continues in this fashion until one of the players reaches 3.

Version Control System: Git

We want you to learn the common "version control system" software that is used everywhere: Git. A version control system helps us keep track of our changes in time. It is also used as a collaboration tool, which means it also helps you develop software with other developers. Git is one of many version control systems, but currently the most popular one. Therefore, it is vital for you to learn what Git is, what it does, how it works, and how to use it properly. This project is a great starting point because you will be using Git on your own, without the complication of other users, and you will get to learn the basics at your own pace.

Git software is available at https://git-scm.com/ and it contains documentation, along with videos, at https://git-scm.com/doc. However, the documentation also talks about some more advanced things you can do with it. We have already uploaded an offline video to Panopto about Git. Please watch it and start using Git in your project from the very first day.

While Git can be used on your own local system, it is mostly used with a remote (online) server. The most common one is the GitHub, at https://github.com/. Registration is free, so make sure you create a profile if you haven't done so already. You can create a repository for your project, and keep your source code online. Make sure you keep your Bluejack project private, so that no one else can access it.

Requirements

In any software project, we write down a list of requirements. These requirements have to be implemented because it is what the software should do in a given environment. Some of them are "functional" - which are about the functions the software should have, and some of them are "non-functional" - which are about the conditions that the software will be running. You are expected to implement all of them.

Non-functional Requirement 1: The program must be implemented in the Java programming language.

Rationale: Since we are learning programming using Java, this is expected. You are expected to use Java, but refrain from using additional libraries, such as GUI. You will not get any additional points if the project is in 3D.

Non-functional Requirement 2: The development must be done on a **private** Git repository.

Rationale: While you are free to use any Git server, we strongly suggest that you create a GitHub account. Every time you improve your project, commit your changes to the repository, so that we can see your progress at the end of the semester. Install the "GitHub

Desktop" program and you will be ready to go. **The repository must be private**. No one must be able to see it, but you.

Non-functional Requirement 3: The program must be delivered as a JAR file.

Rationale: Instead of several class files, pack the program into a JAR file, and submit it as a Jar file.

Functional Requirement 1: The program must be able to create a game deck.

Rationale: This is a basic requirement: without the cards the game would be unplayable.

Functional Requirement 2: The program must be able to shuffle the deck.

Rationale: Although this is only required at the very beginning it is a very good challenge for SE 115. You must **not** use a shuffle method from the JDK, but rather **implement your own**. Find a shuffling algorithm, learn how it works, and implement it.

Functional Requirement 3: The program must be able to create player decks, as described in this document.

Rationale: The player deck will be used to select the cards for the player hand. Please read the explanation in this document to prepare the player deck.

Functional Requirement 4: The program must be able to move cards from the game deck to the players and the boards.

Rationale: There must be only one copy of a card - this requires careful planning: how will the cards be moved from the deck into the player's hand, or the board, or when the player takes all the cards, where will the cards go?

Functional Requirement 5: The program must be able to take turns between players, and let the player and the computer to either draw a card from the game deck, or stand.

Functional Requirement 6: The program must be able to store a "game history" on a file which stores the latest 10 games, names of the players who played them, and the date.

Rationale: You will learn how to read and write to a file in the upcoming weeks. The game history list must be updated after each game is played. Once there are more than 10, the last ones should be removed and only 10 must stay in the list. The game history must be printed as Kaya:3 - Computer:0, 2023.10.29, where Kaya is the player name.

Functional Requirement 7: The program must be able to play for the computer.

Rationale: The program must be able to choose the appropriate card for the computer. You are welcome to improve your game strategy. Make sure you discuss it in the report.

Report

A report should accompany your source code. The report must be written formally. In this report, discuss how you have implemented all the functional requirements, discuss your design choices, the challenges you have faced and how you have solved them.

This can only be done if you keep taking notes during the development. When we read the report, we must understand how you have developed the program. So, make sure you do your best to talk about what you have done in detail. The more detailed it is, the better.

The report must be in PDF format. If you submit in another format, such as "docx" or "pages" or anything else, we will not accept it, and you will not get any points.

Grading

All functional requirements are 10 points each; this is 70 points.

Report is 20 points.

Git logs will be examined, it is 5 points. This is a separate PDF file that contains screenshots. However, we can ask you to show us your Git repository during the oral exam, or we can email you earlier to give access to us as a contributor.

There will be an oral exam at the end of the semester: 5 points.

Exceptions:

- If the program is not running, then you get zero.
- If you do not submit your JAR file, then we cannot run your program, and you get zero.
- If the program crashes on an input, you will lose a large amount of points. Test your code.
- If you cannot answer the questions in the oral exam, you can lose more than 5 points. For example, if we ask you a question about “shuffling” and you cannot answer it, then you will not get any points from the shuffling requirement. If you do not show up to the oral exam, **we can take away as many points as we like.** That is because we will assume that we have asked you some number of questions, and you were not able to answer them.

Submitting Your Project

This project is not a team project. It is an individual project. You have to submit the following files

- A ZIP file that contains your source code.
- A JAR file that runs your program.
- A PDF file that contains your project report.
- A PDF file that contains screenshots of your Git commits.

The due date is 24.12.2023, 23:00. Late submissions will NOT be accepted.