Simple Application –

Blackjack Game



Contents

[Project and My Aims 3](#_Toc70863796)

[Game Instructions 3](#_Toc70863797)

[How the Code/ Main Classes Work 4](#_Toc70863798)

[Enumeration 4](#_Toc70863799)

[Card Class 5](#_Toc70863800)

[Deck Class 5](#_Toc70863801)

[Blackjack Class 5](#_Toc70863802)

[Testing 5](#_Toc70863803)

[Reflection 5](#_Toc70863804)

# Project and My Aims

The programming project that I chose for the Simple Application assessment was the freestyle game, and in this case, a text-input and output version of the card game, Blackjack. I chose this particular game for the project because I felt that it was not overly complex, meanwhile being complex enough to show understanding of basic coding concepts to develop simple application. Nonetheless, I was still able to add some cool features to meet the criteria, and also make the experience more enjoyable for the user.

# Game Instructions

**Step One:**

**Running the code from BlueJ**

Navigate to the class-view window, which should have all classes laid out visually for the user to see. Right click the ‘Blackjack’ class (as this contains the main method for the code to run) and click ‘Compile’. Once compiled, right click a second time and click ‘void main (String[] args)’. The game should open on a separate window.

**Running the code from Eclipse**

Navigate to the Package Explorer on the left of the screen. Expand ‘src’ to reveal the classes, and right click the Blackjack.java class. Select ‘Run as’, then from its drop-down menu, click ‘1 Java Application’ and the game should commence in the runtime environment at the bottom of the screen. Alternatively, there is a ‘Run’ button located on the top of the screen, under the ‘Search’ tab, which is more convenient for simpler programs.

**Step Two:**

Now that the game is up and running, the dealer will give an appropriate greeting depending on the time of day, then will inform you that you have £500 to play with. The dealer will then ask much money you would like to bet. The starting bet amount can be anything from 0 to 500 pounds. Once you have decided how much you are going to bet, type it into the terminal and press Enter.

**Step Three:**

After entering the bet amount, you will be given two cards, both of which will be completely random in suit and value. The total value of your hand is automatically calculated, and is displayed below the cards. Underneath your total hand value, the dealer has one of their cards ‘showing’ (printed to a String) whereas the other one is not yet revealed. You are then given a choice to hit or stand. If you wish to hit (receive another card), you can either type ‘hit’ or the number ‘1’ into the terminal. If you wish to stand, you can type ‘stand’ or ‘2’ into the terminal. Making any of the letters capital in ‘hit’ or ‘stand’ is fine.

**Step Four:**

If you chose to hit, you will be given another card, which will add to your total hand value. There is no limit to the number of times you can hit, therefore, in a scenario where you have relatively low cards, you can keep hitting so that you can get as close to 21 as possible. However, if you hit, and your hand value goes over 21, you are then bust, and lose the round. Once you are happy with your hand value, type ‘stand’ or 2 into the terminal. The game will move on straight away and you will not receive any more cards.

**Step Five:**

After standing, the dealer’s second card is revealed, and as long as their hand value is less than 17, they will keep drawing cards until they reach or surpass this. If the dealer draws some cards and their hand is higher than yours, but equal to or lower than 21, they win. If they keep drawing cards and go over 21, they are bust and you win. Another outcome would be if you had a hand value of 19, for example, and the dealers hand reaches 18, you would win. Just like in normal Blackjack, a loss means you lose all money bet that round, and a win means that you get double your bet amount back. Finally, if there is a push (a tie) your bet amount is returned to your balance.

**Step Six:**

Lastly, after winning, losing, or resulting in a tie, the dealer will then ask if you would like to play another round. You can either type 1 for ‘Yes’, or 2 for ‘No’. If you choose to play another game, the cards are returned to the deck and another round will start, factoring in your wins/ losses. However, if you do not wish to play anymore, entering 2 will close the text input and thus end the game.

# How the Code/ Main Classes Work

## Enumeration

Since this is a card game, creating the necessary classes for the game was not too difficult. I knew that I needed a card object, a deck object (an ArrayList of cards) and a class to put the Blackjack game-logic in. In this case, the only relevant information for a card is its suit and value (value being slightly more important). The easiest way to create the sets of suits and values was using two enumeration classes, ‘Suit’ and ‘Value’. Because suits don’t matter too much in Blackjack specifically, I left the enum as a simple list of the suits. However, for Value, I decided that initialising and returning the values as integers, in the enum class, would be more far more concise than writing a switch statement in the Deck class.

## Card Class

This class defines one singular card, meaning it has to use Suit and Value. I started by creating a new private Suit (enum) called ‘suit’, and a private Value called ‘value’. The identifiers must be lowercase, otherwise Java would have issues differentiating them from the enum class, when used in statements. Next, I had to create three methods for the cards.

The first one was the card constructor, which defines that every card created must have a suit, and a value. The second was the ‘toString’ method, which textually represents the card object by returning the value, e.g., ‘KING’, and the suit, e.g., ‘HEARTS’ to a string. I stated that the word ‘of’ should always be in the middle, so that when a card is returned, it would say e.g., ‘KING of HEARTS’. This method is also an example of overriding, as it’s using the Card class’s functionality to convert cards to strings. Lastly, the final method is ‘getValue’. The purpose of which, is to return the numerical value of a card, so that hand values can be calculated, and recalculated when the player and/or dealer hits. This is an example of overriding because

## Deck Class

The deck class is an ArrayList of card objects. I created it was by making a new private ArrayList of cards called ‘deckCards’, which is the instance variable of the class. I then had to create a constructor method so that a new deck of cards can be created at the start of each game. The way I defined the number of cards that should be in a deck was via iteration using for loops. The for loops state that for every suit, there should be a full set of values (4 x 13). Once this was done, I was then able to add the cards to deckCards. The final part of the deck constructor was referencing the shuffle method, outside of both for loops.

The shuffle method consists of a separate, temporary deck which is also an ArrayList of cards. I used the java utility: ‘Random’ to generate a random number of cards (randomCardIndex) to be removed from the original deck, which can be any number from 0 to 51. This number of cards is then to be added to the temporary one, thus changing the ordering. Once this has occurred, the temporary deck then becomes equal to the original deck, meaning the cards have been shuffled successfully. There are also various one-line methods for removing, adding and drawing cards from the deck.

As in real life, operations on the deck are to shuffle, take a card, and put cards back into the deck.

## 

## Blackjack Class

# Testing

* Mention the fixation of logic and syntax errors in the development of the game
* Learned how to debug the code line-by-line using breakpoints
* Learned that when I fix errors, that does not necessarily stop future errors from occurring
* Finding and fixing errors was largely successful, however I realised that it is possible to introduce errors in the future without realising it, including breaking existing functionality. Therefore, a set of tests for the existing functionality would be useful, and would be run after every code change to ensure that nothing gets broken.

# Reflection

My original aim was to create a Graphical User Interface Blackjack game, however after some evaluation I decided that a simpler version of the game was more feasible for the amount of time given for the project. With hindsight, I realised that I should have split up the main method into smaller chunks i.e., separate components. These would then have been more easily testable and could have been tested as I wrote the code e.g., using a test-driven development approach. I will employ this recognised way of development in future projects.

Notes:

* Use screenshot to show how I refactored using Deck class to represent a player or dealer’s hand and to use a proper Hand class instead
* Cite example of overloading, looping, conditionals,