Summary

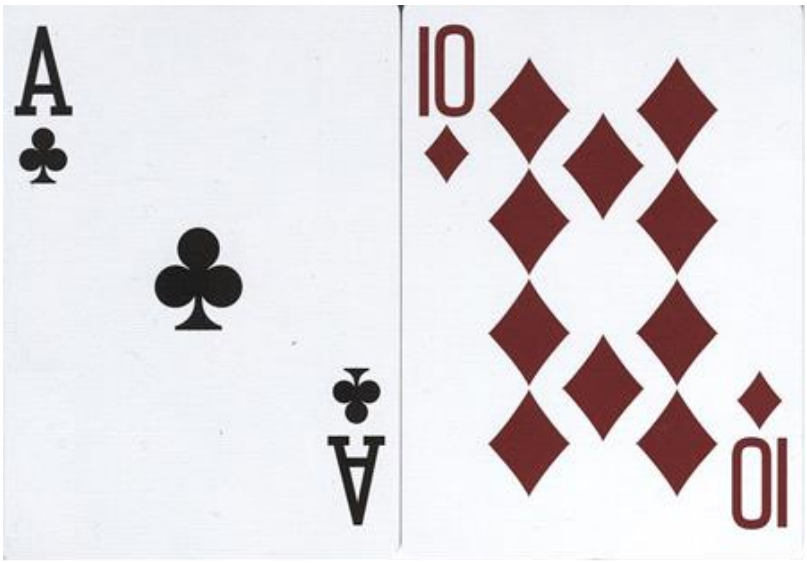
This design document provides the details for the card game of Blackjack for Fundamentals of Software Design and Development project. We will implement the lessons learned both in-class and key concepts learned in the accompanying course.

Project Background & Game Description

Project Goal:

The team's goal is to develop a single player versus computer Blackjack card game. We hope to develop the game for use within the NetBeans application and command-line based interface as our polished final product.

What Is Blackjack?



Blackjack is the American version a card game also known as twenty-one. It is a comparing type of card game that is played between one or more players and a dealer, where each player competes against the dealer and is played with one or more standard card decks of 52 cards. Each card in a game of Blackjack holds a point value: all numbered cards are equal to their face number, all Jacks, Queens, and Kings are equal to ten, and the Ace is equal to either 1 or 11 depending on the player's choice. Every round each player receives two cards and are given the option to receive another or stay with the cards they hold. The goal for a player is to achieve a higher point total than specifically the dealer, while staying at or below 21. Ideally, a player wants to achieve 21 as this means they win the round automatically.

Starting Code:

The current starting code are the four classes: "Card", "Game", "GroupOfCards", and "Player". The code is written in Java, and it is created with object-oriented principles in mind. Many of these classes will be extended to be used in subclasses for our specific game. These classes are essentially the blueprints for Objects that we will instantiate to create our Blackjack game. The starting code follows standard naming conventions (i.e., classes start with uppercase letter, while everything else starts with a lowercase letter). All the class's instance variables are private, which aligns with the principle of encapsulation. They currently have very general methods and use; this allows them to be used and changed to fit many of the card games we could choose from.

Project Scope

Team Members:

Name	Role
Volodymyr Suprun	Team Leader
Adarshpreet Singh	Team Member
Maryam Khatibzadeh Azad	Team Member
Othman Tahir	Team Member

The team operating this project will be composed of Volodymyr Suprun, Adarshpreet Singh, Maryam Khatibzadeh Azad, and Othman Tahir. We will be operating on a council system where decisions are made collectively by the team. This way the team is not tied to a single direction and can plan with the input and wisdom of all members. Volodymyr will be the team leader. Having that said, all members will be expected to contribute in an equal manner.

About Project

The project focuses on building a playable game of Blackjack using Java. Blackjack is a card game which involves the card dealer playing against the player(s) in however many rounds the individual player chooses. It is typically a gambling card game that involves real money; however, we chose to not include this and instead will track wins/losses once the program is finished. The principles of Blackjack will be implemented into the project using Java. The game will provide real time info of what is going on the dealer's and the player's side on the interface. Every team member will collaborate on the project and work on the java classes they are entrusted to code. The **completion** of the project will be determined after combining the work of all team members and testing it multiple times to ensure that the project is in accordance with the Blackjack principles/rules.

Interface

The **interface** will be focusing on the real time situation of the dealer and the player. It will display the cards allocated to the player and the card(s) of the dealer. The project will be designed in accordance with the rules of the Blackjack game. The interface will include the **hit** and **pass** button which are a part of the Blackjack game. After the completion of the current round, the winner will be displayed on the interface and the next round will begin.

High-Level Requirements

The high-level requirements for this program are the following:

1. Upon initializing a game of blackjack the player must register themself.

Upon starting the game, the dealer player is automatically initialized. The dealer is not controlled by anyone and plays the game alongside the player.

2. The deck out of which the player and dealer get their cards is initialized at the start of every new round of blackjack.

It is a standard 52 card deck. These cards are handed out randomly and there are no duplicate cards in this deck.

3. The player must have the ability to see their score at any time.

This would be the point value that the player's card hand and the dealer's card hand individually add up to. All cards between 2-10 are equal to their face value (4 = 4, 6 = 6, etc.), Jacks, Queens and Kings are 10 each, and Ace can be 1 or 11 depending on player or dealer choice. Additionally, the wins, losses, and ties of the player are updated every round end.

4. During the player's turn they have access to a few commands: Hit, Stand, and Exit.

Hit indicates the player chooses to receive another card, and Stand indicates the player wishes to play out the round with their current cards. Exit indicates the player leaving, after which the game ends.

5. The dealer must act according to regular blackjack rules.

One card the dealer receives will be face down, while the other face-up. Depending on the point total the dealer will choose to either hit or stand once the player has had a turn. At 17 points or more the dealer stands, and at 16 or below the dealer hits.

6. At the end of the round the winner is announced.

The player will be a winner if they achieved a higher point value than the dealer, or if they reached 21 points exactly. The player loses if they get fewer points than the dealer or more than 21 points. If the player achieves the same total as the dealer, it is a tie.

7. The game ends when the player decides to exit, after which the final wins, losses, and ties are shown and the program ends.

Implementation Plan

GitHub Repository URL:

https://github.com/Vlad0244/Deliverable-SYST17796-

Implementation Plan

- The team members have a chat box where we collaborate and discuss about the ideas for the project.
- The tasks are divided between the team members.

- Group video call are held time to time to discuss the project progress and ideas more effectively.
- Every team member is obliged to use GitHub.
- The GitHub will be integrated in the NetBeans. So, every time a team member makes a change in the code, they will push the code to the git repository to update the code at the remote repository and change the last modified date and modified by comments.
- Every team member is obliged to check the GitHub repository once a day to keep themselves updated with the changes and the progress of the project.
- The git repository will have a separate folder designated for java code, the UML diagrams, and the description files.
- Once every team member has finished their code, the code will be combined and run to check the functioning.
- If the project works and follow the principles of the Blackjack game, the project will be considered complete.
- The GitHub repository will be checked once for last to ensure everything is in place and the project will be considered complete.

Platforms

The following platforms will be used by the team members:

- NetBeans
- GitHub
- Visual Paradigm
- Visual Code

Design Considerations

OOP Principles Used in Current Code and Future Code

Encapsulation

The current code is distributed among multiple classes so in this way most of the functionality(methods) codes are hidden from users and that users will have no idea of how classes are being implemented or stored.

- An example of encapsulation is classes “Game”, “Player”, and “GroupOfCards”. The data variables are set to private so to be able to access them we use setter and getter methods which are set to public.
- One of our future additions will be the use of enumeration classes to set any of our cards suits and ranks. This ensures very few mistakes when initializing our decks and when working with the suits and ranks.

Flexibility/Maintainability

A large portion current code is structured in a way that allows for many of the classes to be reused or maintained to serve many purposes. Here are a few examples:

- The setter and getter methods provide flexibility to the code as they can be used multiple times in multiple classes.
- The use of abstract methods and classes. The “Card”, “Game” and “Player” classes are set to be an abstract. These classes can be used as base class and/or blueprint for future subclasses. The toString() method is set to abstract as well.
- the cards instance variable in GroupOfCards is set to be an array list so it has the possibility to store data and be dynamic for future changes.

- The setSize method will allow us to increase or decrease the size of a deck to fit an entirely different card game. This would also require changing the values in our planned enumeration classes and refactoring some values.

Delegation

The code's structure has and will have many elements of delegation. It is the process of delegating certain tasks to other classes. Here are some examples from the code and some that will be implemented in the future:

- The shuffle method in GroupOfCards makes use of the Collections framework to shuffle the ArrayList cards instead of creating code that shuffles the deck. This delegates the shuffle function to a different class and simply calls on it when needed.
- Our future code will require the manipulation of the GroupOfCards class from other classes to function. Most of the methods necessary will be within the GroupOfCards class and we will simply call on them to do function like shuffling the deck.