syst 17796 Deliverable 1

design document

# Overview

## Project Background and Description

The program is a Java implementation of the classic card game Blackjack using object-oriented programming principles. The game is played between the player and the dealer, with the objective being to have a hand value that is closer to 21 than the dealer's hand, without going over 21 (which is called "busting").

Here's an overview of the program's logic and rules:

The game starts by shuffling a standard 52-card deck and dealing two cards to the player and two cards to the dealer, with one of the dealer's cards face down.

The player is then prompted to choose whether to "hit" (receive another card) or "stand" (keep their current hand). If the player chooses to hit, another card is dealt to the player. The player can continue to hit until they decide to stand or until their hand value exceeds 21.

Once the player has finished his or her turn, it's time to the dealer's turn. The dealer reveals their face-down card and hits until their hand value is at least 17.

Once both the player and the dealer have finished their turns, the game is over and the winner is determined. If the player's hand value is greater than 21, the player busts and loses the game. If the dealer's hand value is greater than 21, the dealer busts and the player wins. If neither player busts, the player with the hand value closest to 21 without going over wins. If both player and dealer have the same hand value, the game ends in a tie.

## Project Scope

We divide the project into four members. Firstly, we all choose a game BlackJack for the implementation. Here is the collaboration of our group members.

Viren Panchal: Implementation of java code for a blackjack game, Creating Git repository, adding members, and pushing the project, the Class diagram on the git repository.

Jeel Mavani: Helping during the implementation of java code and solving the errors.

Dhwani Soni: make a Class diagram using NetBeans and Visual paradigm professional and help Noopur to make a design document.

Noopur Patel: Make a Design Document and help Dhwani to make a class diagram.

## High-Level Requirements

* The game should have a deck of cards that can be shuffled and dealt to the players.
* The game should have two players: the player and the dealer.
* At the beginning of the game, each player should be dealt two cards from the deck.
* The player should be able to see one of the dealer's cards.
* The player should be able to hit (receive another card) or stand (keep their current hand).
* If the player's hand exceeds 21, they lose the game.
* Once the player stands, it's the dealer's turn.
* The dealer should hit until their hand is at least 17.
* If the dealer's hand exceeds 21, the player wins.
* If the player's hand is greater than the dealer's hand and less than or equal to 21, the player wins.
* If the dealer's hand is greater than the player's hand and less than or equal to 21, the dealer wins.
* If both the player and the dealer have the same hand value, the game ends in a tie.
* The game should keep track of the player's and dealer's hands and their current values.
* The game should provide an interface for the player to interact with the game (e.g., buttons for hitting and standing).
* The game should provide feedback to the player on their current hand value and whether they have won or lost the game.
* The game should provide options for the player to start a new game or quit the game.

## Implementation Plan

Here is the URL of the Git repository: <https://github.com/virenpanchal1001/Group11_Gujjus.git>

We use netbeans to implement the java code on blackjack. We also use GitHub to make a repository and Visual Paradigm Professional to make a class diagram.

## Design Considerations

* The current code is structured using the following Object-Oriented principles:
* Encapsulation: The code uses encapsulation to hide the internal details of the classes and provide a well-defined interface for interaction with the classes. For example, the **Hand** class encapsulates the **cards** list, and provides methods like **addCard**, **getCards**, and **calculateHandValue** for interaction with the cards in the hand. Similarly, the **Deck** class encapsulates the **cards** list and provides methods like **shuffle** and **dealCard** for interaction with the deck.
* Potential for improvement: Currently, the **Card** class does not have any encapsulation, and its variables are public. It would be better to make the variables private and provide accessor and mutator methods to access and modify the variables.
* Inheritance: The code uses inheritance to model the relationships between the classes. For example, the **BlackJackHand** class inherits from the **Hand** class to add the **isBlackJack** and **isBust** methods for determining if a hand is a blackjack or if it is bust. Similarly, the **Deck** class inherits from the **CardCollection** class to add the **shuffle** method for shuffling the deck.
* Potential for improvement: Currently, there are no other examples of inheritance in the code. However, if the code were to be extended to include other types of card games, it could benefit from using inheritance to model the relationships between the classes.
* Polymorphism: The code uses polymorphism to allow different classes to be treated as instances of a common parent class. For example, the **Hand** class can contain any type of card, not just blackjack cards, and the **Deck** class can contain any type of card, not just blackjack cards.
* Potential for improvement: Currently, there are no other examples of polymorphism in the code. However, if the code were to be extended to include other types of card games, it could benefit from using polymorphism to allow different types of cards to be used in the same deck and hand.
* Overall, the current code follows good Object-Oriented principles, but there is room for improvement in terms of encapsulation, inheritance, and polymorphism.