**SYST 17796 DELIVERABLE 1**

**By: 1.**

**DESIGN DOCUMENT TEMPLATE**

**OVERVIEW**

**1. Project Background and Description**

**Project Objective:** The primary aim of this endeavor is to develop a digital rendition of the classic card game Blackjack.

**Game Overview:** Blackjack, also known as Twenty-One, is a card game that pits the player (referred to as "User") against the computerized dealer. It utilizes a standard deck of 52 cards. The objective is to outplay the dealer using one of the following strategies:

* Achieve a total of 21 points with the initial two cards dealt (referred to as a "blackjack"), provided the dealer does not also have a blackjack;
* Attain a higher final score than the dealer without exceeding 21 points; or
* Allow the dealer to draw additional cards until their hand surpasses 21 points, resulting in a "bust."

**Game Regulations:**

* Face cards carry a value of 10, while Aces can be valued at either 1 or 11, depending on which value enhances the hand.
* Initially, each player receives two cards, with one of the dealer's cards kept hidden until the conclusion.
* During the player's turn, two choices are available: to 'Hit,' requesting another card, or to 'Stand,' retaining the current total and concluding the turn.
* Exceeding a total of 21 results in a bust, leading to an automatic win for the dealer regardless of their hand.
* If a player is dealt a total of 21 from the outset (comprising an Ace and a 10), it results in a blackjack.
* A blackjack entitles the player to receive double the amount of the initial bet.
* The dealer continues to draw cards until their total reaches 17 or higher.
* Doubling functions similarly to a hit, but the bet is doubled, and the player receives only one additional card.

**Project Vision:**

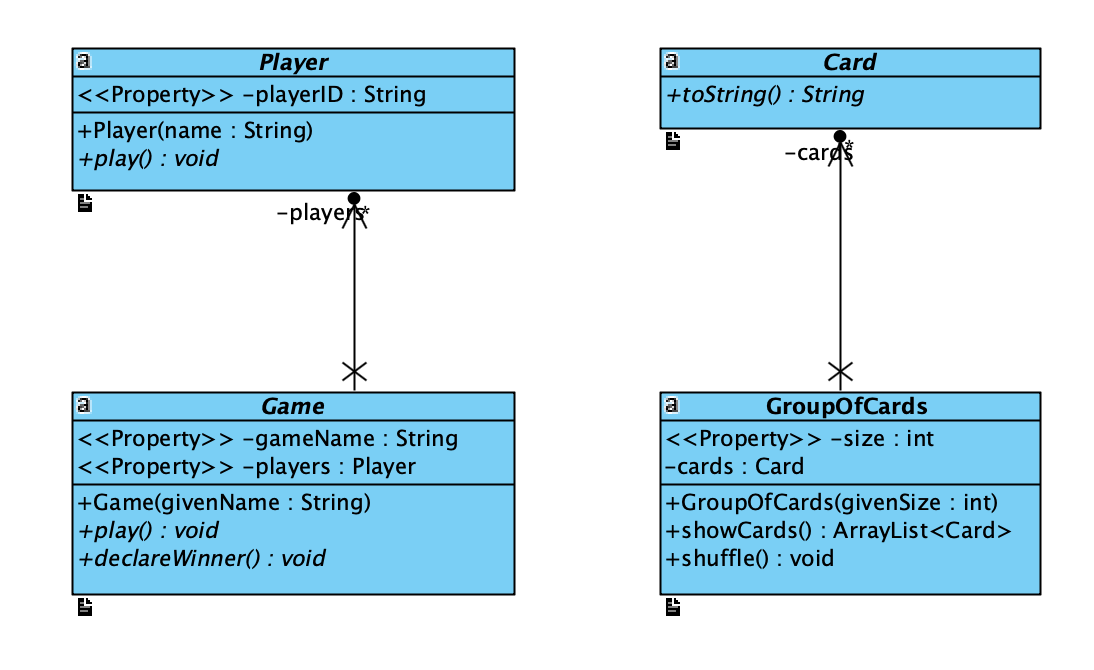
1. Expansion into Various Card Games: The project holds the potential to broaden its scope by incorporating a diverse array of card games, encompassing titles like Go Fish, War, Spot It, Uno, Skip-Bo, and beyond.
2. Enhanced Gameplay Features:
   1. Integration of Splitting Mechanism: Introduce the functionality of splitting pairs when the player holds two identical cards, thereby dividing the pair into separate hands.
   2. Doubling Bet upon Split: Splitting a pair entails doubling the initial bet, as each new hand holds the value of the original wager.
   3. Limitation on Doubling/Splitting: Restrict the ability to double or split to the initial move or the initial move following a split. d. Constraint on Split Aces: Prohibit the player from further action on split Aces.
   4. Amplified Betting Options: Allow players to double their bet on hands resulting from a split, potentially tripling or quadrupling the initial wager.
3. Transition to an Online Environment: Explore the potential for transitioning the game to an online platform, facilitating multiplayer engagement simultaneously, thus enriching the gaming experience.

**Description of the Initial Codebase:**

The provided base code serves as the foundation for further development, written in Java programming language. It comprises essential classes representing key entities involved in the game. These classes include:

1. Card: This class encapsulates the properties and behaviors of individual playing cards, such as their rank and suit.
2. Player: The Player class defines the characteristics and actions associated with each player participating in the game.
3. Group of Cards: This class represents the collection of 52 cards typically used in the game of Blackjack. It manages the deck of cards and provides functionality for shuffling and dealing cards.
4. Game: The Game class serves as the starting point for the game execution. It orchestrates the flow of gameplay, including managing players, dealing cards, and determining game outcomes.

The diagram provided illustrates the relationships between these classes, outlining the structure of the initial codebase and its components.



**2. Project Scope**

**Team Member**

|  |  |  |
| --- | --- | --- |
|  | **Team Member Name** | **Role** |
| 1 |  | Coding – Design - Implementation |
| 2 |  |  |

**Technical Scope:**

The initial set of interfaces to be implemented includes:

1. IPlayer:
   * drawCard(): Method to draw a card from the deck.
   * calcHandValue(): Method to calculate the total value of the player's hand.
   * hasBlackJack(): Method to check if the player has a blackjack.
   * hit(): Method to allow the player to request another card.
   * betCash(): Method to place a bet.
   * pushCash(): Method to handle a push (tie) situation.
   * win(): Method to handle a win scenario.
   * lose(): Method to handle a loss scenario.
2. IDealer:
   * showFirstCard(): Method specific to the dealer (computer) to reveal the first card.
3. ICard:
   * shuffle(): Method to shuffle the deck of cards.
   * drawCard(): Method to draw a card from the deck.

Upon successful implementation and passing of the corresponding tests, the project will be considered complete.

**3. High-Level Requirements for the New System:**

The new system must incorporate the following functionalities:

1. Player Registration:
   * The system should allow each player to register with the game, providing necessary details for identification.
2. Cash Registration:
   * Players should be able to register their cash with the system, establishing their initial stake for the game.
3. Betting and Doubling Down:
   * Players must have the ability to place bets and opt for the double-down option, with the system accurately calculating their accumulated cash based on wins and losses.
4. Decision-Making:
   * Players should be empowered to make strategic decisions during their turn, including the options to hit (request another card) or stand (retain the current hand).
5. Outcome Communication:
   * The system should effectively communicate the outcomes of each round, indicating whether the player has won or lost.
6. Status Monitoring:
   * Players must have continuous access to their status (score) throughout the game, ensuring transparency and informed decision-making.
7. **Implementation Plan:**

Git Repository:

* Utilize a Git repository hosted on GitHub for version control and collaboration.
* Project members will receive update notifications via email from GitHub.com.
* Project members are required to check and update the code at least once every day to ensure timely progress and delivery.

Coding Standards:

* Source file organization:
  + Documentation comments, package declaration, followed by a class comment.
  + Imports grouped with static imports listed last.
  + Class/interface signature.
* Naming conventions:
  + Use CamelCase for naming variables and methods, ensuring meaningful names.
  + Use ALL\_CAPS for constants.
* Indentation and Line Length:
  + Use 4 spaces for indentation.
  + Limit line length to 80 characters.
* Line Breaks:
  + Open brace "{" appears at the end of the same line as the declaration statement or method.
  + Closing brace "}" starts a new line by itself and is indented.
* Comments:
  + Extensively use comments where appropriate and meaningful to enhance code readability and understanding.
* Method Overrides:
  + Always use "@Override" annotation when overriding methods.

By adhering to these coding standards and practices, the development process will be streamlined, ensuring consistency, readability, and maintainability of the codebase.

**Tools used:**

* NetBeans

1. **Design Considerations:**

In the design of the Blackjack game, Object-Oriented (OO) principles are prioritized to facilitate future expansions. Key design considerations include:

**Encapsulation:**

1. Abstract Classes: Abstract classes are designed to be as generalized as possible. The Player class, for instance, encapsulates only one attribute – playerID, representing the player's name.
2. Interfaces: Interfaces are contemplated for method encapsulation, providing implementing classes with access to relevant methods while promoting modularity and abstraction.

**Delegation:**

1. Player Class: The Player class serves as an abstract entity, accommodating both the user and the dealer (Computer). This delegation ensures that both players share common functionalities.
2. Card Class: The Card class, being abstract, establishes a foundation for other card games to inherit from. This delegation enables the application of the same card functionality across various card games like Blackjack, Go Fish, etc.

**Flexibility/Maintainability:**

1. Interface Usage: Incorporating interfaces enhances program visibility and functionality. This facilitates ease of understanding for developers and allows for seamless expansion of the codebase.
2. Abstract Classes: Leveraging abstract classes such as Card and Game permits the extension of child classes, fostering flexibility for the program to accommodate other card games efficiently.

By adhering to these design considerations, the Blackjack game is not only robust and extensible but also lays a solid groundwork for accommodating future expansions and modifications.