**SYST 17796 DELIVERABLE 1**

**By: 1. Heymann Sharma**

**2. Manh Hung Dao**

**DESIGN DOCUMENT**

**OVERVIEW**

**1. Project Background and Description**

**Project goal:** The goal of the project is to create a Blackjack card game.

**Brief Description of the Game**

Blackjack (or Twenty One) is a comparing card game between player -User and a dealer – Computer. It is played with one card collection (decks) of 52 cards. The objective of the game is to beat the dealer in one of the following ways:

* Get 21 points on the player's first two cards (called a "blackjack") without a dealer blackjack;
* Reach a final score higher than the dealer without exceeding 21; or
* Let the dealer draw additional cards until their hand exceeds 21 ("busted").

*Game Rules:*

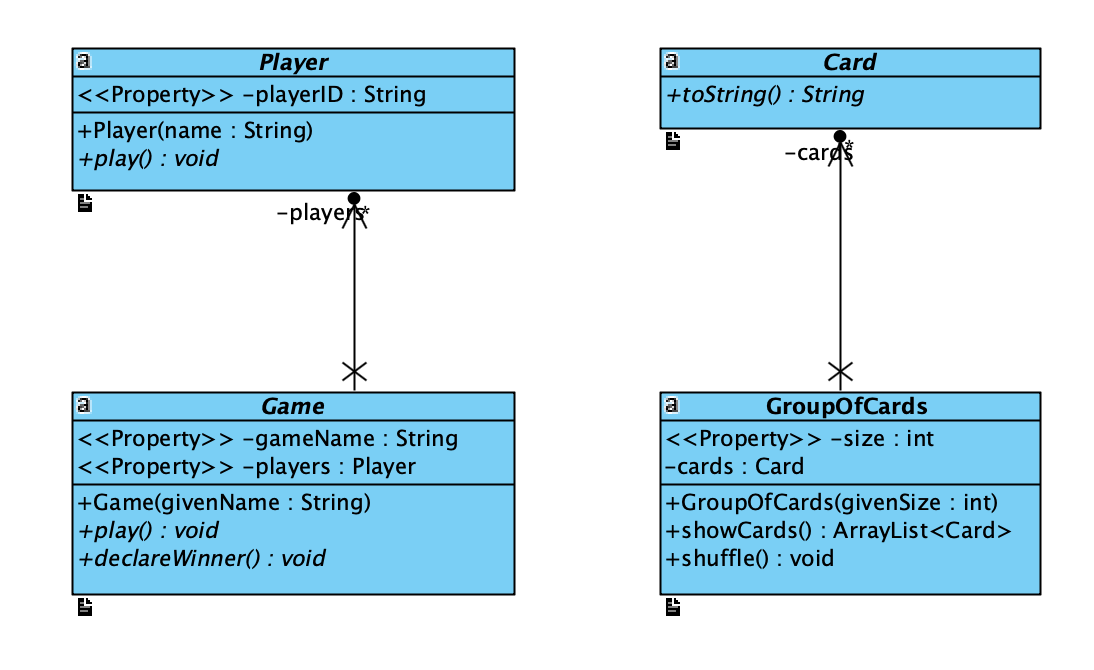
* Face cards are worth 10. Aces are worth 1 or 11, whichever makes a better hand.
* Each player starts with two cards, one of the dealer's cards is hidden until the end.
* At player’s turn, player has 2 options: To 'Hit' is to ask for another card, or to 'Stand' is to hold your total and end your turn.
* If player go over 21, it is a bust, and the dealer wins regardless of the dealer's hand.
* If player is dealt 21 from the start (Ace & 10), player gets a blackjack.
* Blackjack means player win twice the amount of the bet.
* Dealer will hit until dealer’s cards total 17 or higher.
* Doubling is like a hit, only the bet is doubled and player only gets one more card.

**Project vision:**

1. The project could be extended to include other card games, such as Go Fish, War, Spot It, Uno, Skip-Bo, etc.
2. The game could be extended to handle the case of Split:
   1. Split can be done when player have two of the same card - the pair is split into two hands.
   2. Splitting also doubles the bet, because each new hand is worth the original bet.
   3. Player can only double/split on the first move, or first move of a hand created by a split.
   4. Player cannot play on two aces after they are split.
   5. Player can double on a hand resulting from a split, tripling or quadrupling you bet.
3. The game could be extended to online environment, with multi-players at the same time

**Description of the current base code**

Base code is given as the starting point for development. The code is written in Java, with initial classes for stake-holders in the game, including the Card, Player, Group of Cards (52 cards used in Blackjack), Game (the start of the Game), as shown in the diagram:



Describe the project goals and final vision. Include a brief description of how to play the game you have chosen and a reference to the rules of the game you have chosen. Also describe the current starting base code. Use technical terms to describe the code including what language it is written in, any patterns you can see and any coding conventions used.

**2. Project Scope**

**Team Member**

|  |  |  |
| --- | --- | --- |
|  | **Team Member Name** | **Role** |
| 1 | Heymann Sharma | Coding – Design - Implementation |
| 2 | Manh Hung Dao | Design – Testing – Documentation |
| *3* | *Calling for additional team member to share workload* |  |
| *4* | *Calling for additional team member to share workload* |  |

**Technical Scope:**

The following are the initial consideration of interfaces to be implemented:

* 1. IPlayer, including:
     + drawCard()
     + calcHandValue()
     + hasBlackJack()
     + hit()
     + betCash()
     + pushCash()
     + win()
     + lose()
  2. IDealer, including:
     + showFirstCard() – this method is applied to Dealer (Computer) only
  3. ICard, including:
     + shuffle()
     + drawCard()

When all the interfaces are implemented, and the running tests are satisfied – project is completed!

**3. High-Level Requirements**

The new system must include the following:

* Ability for each player to register with the game
* Ability for player to register their cash
* Ability for player to place a bet / double-down, and calculate his/her accumulated cash
* Ability for player to decide to hit / to stand at his/her turn
* Ability for the game to communicate a win or loss
* Ability for players to know their status (score) at all times

**4. Implementation Plan**

**Git repository:** [**https://github.com/hungdmcan/SYST-17796-PROJECT**](https://github.com/hungdmcan/SYST-17796-PROJECT)

Project members receive update notification from Github.com via email.

Project member with check / update code at least once every day (once/day) to ensure the on-time delivery of the project.

**Coding standards:**

* The source file is organized with documentation comment, package declaration, followed by a class comment, imports grouped (static last), class/interface signature
* Naming using Camel-Casing with meaning, with method name using verb and CONSTANT using ALLCAPS
* Indentation: 4 spaces / Line length: 80 characters
* Line break: open brace “{” appears at the end of the same line as the declaration statement / method and closing brace “}” starts a new line by itself indented
* Comments should be done extensively where appropriate & meaningful
* Always use @Override when overriding

**Tools used:**

* NetBeans with VP
* Javadoc for documentation generation

**5. Design Considerations**

In the design of the Blackjack game, OO principles are considered so that it can be used as the basis for future expansion.

Some key design considerations are as below:

** Encapsulation**

1. Abstract Classes are designed as the most generalized as possible, with Player only have one encapsulated attribute, i.e. playerID, which is the player’s name.
2. Interfaces are being considered with method encapsulation, which will help to provide implementing classes with access to appropriate methods.

** Delegation**

1. Class Player is an abstract class. The user of the program is a Player who has a hand, and the dealer (Computer) is also a Player.
2. Class Card is an abstract class. There are other card games, that can be inheritance of this Card Class, such as Blackjack, Go Fish, etc. Therefore, this relationship can be used to be applied to other game for future expansion.

** Flexibility/Maintainability**

1. The use of interfaces helps to provide the clear visibility of the program functionality. This helps developers with easy understanding as well as expanding of the codes for future expansion.
2. The use of abstract classes (Card, Game…) allow the extension of child classes, which provide flexibility for the program to be used for other card games.