**Black Jack Game**

Software Design Process

Revision History

| **Date** | **Revision** | **Description** | **Author** |
| --- | --- | --- | --- |
| 10/17/2024 | 1.0 | Draft for Software Design Specification | Everyone |
| 10/19/2024 | 1.1 | Update UML diagram | Everyone |
| 10/20/2024 | 1.2 | Update Use cases | Carolina Barbecho |
| 10/24/2024 | 1.3 | Added new Use Cases | Brandon Sandoval |
| 10/26/2024 | 1.4 | Updated UML Diagram with more clear methods | Nikolas Alvarado |
| 10/26/2024 | 1.5 | Interface Design - UI | Carolina Barbecho |
| 10/29/2024 | 1.6 | Evaluated Sequence Diagrams | Nikolas Alvarado |
| 10/29/2024 | 1.7 | Updated Document to clarify certain sections | Matthew Carreon |
| 10/29/2024 | 1.8 | Updated Use Cases to more uniformly handle classes. | Nikolas Alvarado |
| 10/30/2024 | 1.9 | Added github link to section 1.2 | Santiago Ruiz Arias |
| 10/30/2024 | 2.0 | Added Server-Client Class Diagram | Santiago Ruiz Arias |
| 12/02/2024 | 3.0 | Update Server-Client Class Diagram | Santiago Ruiz Arias |
| 12/03/2024 | 3.1 | Update Document to Reflect from Phase 3 | Matthew Vincent Carreon |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

**Table of Contents**

[**1. INTRODUCTION 4**](#_heading=h.kbhn8bij7wzd)

[1.1. Purpose 4](#_heading=h.3znysh7)

[1.2. Overview 4](#_heading=h.l8bsezdwyfz)

[**2. INTERFACE DESIGN 5**](#_heading=h.1fob9te)

[2.1 UI 5](#_heading=h.jomz95k8ym35)

[**3. ARCHITECTURAL DESIGN 6**](#_heading=h.35nkun2)

[3.1 Components for Blackjack Game 6](#_heading=h.toc565z5h1ds)

[3.2 Client - Server Architecture and Multithreading 6](#_heading=h.u6m28saebew4)

[**4. DETAILED DESIGN 8**](#_heading=h.sxlrixf45mgc)

[4.1 System Components 8](#_heading=h.5bp23sozgu1d)

[4.2 Functional responsibilities 8](#_heading=h.drkjund7e45x)

[4.3 User Interfaces 8](#_heading=h.8vp6at4di7v)

[4.4 Unit States / State changes / Enum 8](#_heading=h.xy4xs2a9pz20)

[4.5 Data and Control Interaction Between Units 8](#_heading=h.bhaz5vw3o6aa)

[4.6 Data Packaging and Implementation 8](#_heading=h.4s63b3q91imz)

[4.7 Algorithms and Data Structures 8](#_heading=h.adl1w0lqjdvz)

[**5. REFERENCE 9**](#_heading=h.e0dm7kk3mrm)

[5.1 Use Case Specification Document 9](#_heading=h.ilcndki6dyue)

[5.2 Classes 18](#_heading=h.luguw3ulg917)

[5.3 Class Diagram 27](#_heading=h.cwz9uo1h3d74)

[5.4 Sequence Diagram 27](#_heading=h.wuw2ve53bb98)

# INTRODUCTION

## Purpose

This document outlines the software design specification, detailing the structure and functionality of the system.

## Overview

This project includes the interface design, architectural components, and detailed class structures required to implement the game. This document serves as a comprehensive guide for the development team, facilitating a clear understanding of the game’s requirements and design considerations.

This and all other Documents and files for this project can be found on the project github:

<https://github.com/Barbecho90/Group5SoftwareEng>

# INTERFACE DESIGN

The interface design aims to create an intuitive and user friendly environment for players and dealers participating in the Blackjack game.

## 2.1 UI

1.1 First Screen

→ Account & Login

→ username

→ password

→ Role

→ Exit

1.2 Main Menu

→ username or id

→ View Balance

→ Add funds

→ withdraw Funds

→ Join Game (Players /Disable for dealers)

→ Create Game (Dealers / Disable for players)

1.3 GamePlay Screen

→ Bet

→ Display player’s hand

→ Dealers hand

→ actions

→ Hit

→ Stand

→ Split

→ Double Down

→ deal Cards (dealers/ Disable for players)

→ Leave Room (Only when the dealer have not deal cards)

# ARCHITECTURAL DESIGN

## 3.1 Components for Blackjack Game

The system is composed of several classes, each with a specific role:

**Account:** Contains attributes such as username, password, and balance. It includes methods for depositing and withdrawing funds..

**User**: Base class containing common attributes and methods for all users.

**Player class**: Extends User with attributes and methods specific to players including betting and hand management.

**Dealer class**: Extend User, representing the dealer with its own hand management and game rules.

**Card**: Represents a single playing card with suit and value attributes.

**Deck**: Represents a standard deck of 52 playing cards with suit and values attributes.

**Shoe**: Represents 4 decks of cards.

Methods for shuffling, dealing cards, and checking if the deck is empty.

**Hand**: Represent a players or dealer’s current set of cards with methods to calculate hand value.

**Table:** Manages individual tables, including the list of players, maximum capacity, and whether the table is full.

## 3.2 Client - Server Architecture and Multithreading

The Blackjack game will adopt a client-server architecture, where the server manages the game state and handles interactions between multiple clients.(Players and dealers)

**Server**: The server will handle the game logic, including card dealing, player actions and win/loss calculations.

**Client**: Each player and dealer will interact with the game through a client application, which will send requests to the server.

To enhance performance and responsiveness, the server and the client will utilize multithreading. This approach will allow servers to handle multiple client connections simultaneously and for clients to handle multiple tasks concurrently.

* Thread Management: each client connection where the server process requests concurrently.
* Game State Synchronization: Server will maintain a consistent game state across all clients.
* UI Thread: it handles user interactions and updates the UI.
* Connection Thread: Manage multiple clients
* Game Logic Thread: manage the game logic such as: (total hand values, split, double Down, bets, hit, stand, turns, and payments )

Client - server Multithreading will improve the user experience reducing latency, real time updates about game status such as card deals and other player’s actions without noticeable delays. Also, it will allow the client to manage simultaneous interactions efficiently.

# REFERENCE AND DETAILED DESIGN

## 4.1 Use Case Specification Document

**Account:**

**Use Case ID:** UC00

**Use Case Name:** Account andlogin

**Relevant Requirements: “**user authentication”

**Primary Actor:**

User: players/dealer

**Pre-conditions:**

The app is functional and can be used, and there must be credentials already for the player/dealer.

**Post-conditions:**

Users access the system and the available options are tailored to their specific role.

**Basic Flow or Main Scenario:**

The user opens the system interface.

Create an account with username, password and select the role.

The user logs into the pre-made account.  
When a user login a new menu shows options: balance, withdraw funds, deposit funds , or join a game.

**Extensions or Alternate Flows:**

The user logs into the pre-made account, already having a DealerRole

A new menu shows options: balance, create a new table.

**Exceptions:**

If the system is unavailable, it will display a message informing the user of the inconvenience.

**Related Use Cases:**

Add funds(UC02)

Deal cards(UC06)

Withdraw Funds(UC03)

**Use Case ID:** UC01

**Use Case Name:** Join Game

**Relevant Requirements:** Users join the game.

**Primary Actor:**

User: players/dealer

**Pre-conditions:**

The app is functional and can be used and the user must have been logged in.

Players need to have funds before joining a game.

**Post-conditions:**

Users access the system and can use features based on the player/dealer role.

**Basic Flow or Main Scenario:**

Player decides to join the game

Player is prompted to determine their bet before joining

The player have access to the black jack game

Wait for the dealer to deal cards.

**Extensions or Alternate Flows:**

The dealer login in the app.

Dealer is at the table waiting for players to deal a hand.

**Exceptions:**

If the system is unavailable/full, it will display a message informing the user of the inconvenience.

If the player cannot provide a minimum bet, they will not be let into the game

If the player is already in the game and cannot bet, player will be removed.

**Related Use Cases:**

Add funds(UC02)

Withdraw Funds(UC03)

Manage Hand(UC04)

Deal cards(UC06)

**Use Case ID:** UC02

**Use Case Name:** Add funds

**Relevant Requirements:** Users add funds to their account

**Primary Actor:**

User: players

**Pre-conditions:**

The app is functional and accessible and player login.

**Post-conditions:**

The user’s account balance is updated with the added funds.

**Basic Flow or Main Scenario:**

The user navigates to the “Add Funds” in the system/app.

The user enters the amount they wish to add.

The systems check if the amount is a number greater than 0.

If the input is a valid number:

The system updates the user’s account balance.

If the input is invalid:

The system prompts the user to enter a valid numerical amount.

**Extensions or Alternate Flows:**

If the player is a dealer, then a message will be displayed that funds cannot be accessed.

**Exceptions:**

If the system is unavailable, it will display a message informing the user of the inconvenience.

**Related Use Cases:**

Join Game (UC01)

Withdraw Funds(UC03)

**Use Case ID:** UC03

**Use Case Name:** Withdraw Funds

**Relevant Requirements:**

**Primary Actor:**

User: players

**Pre-conditions:**

The app is functional and accessible.

Players are logged in.

The balance amount must be greater than 0.

**Post-conditions:**

The player's account balance is updated after the withdrawal.

A confirmation message is displayed to the user for the withdrawal.

**Basic Flow or Main Scenario:**

The player navigates to the “Withdraw Funds: section of the app.

The player enters the amount they wish to withdraw.

The system checks if the amount is valid and if the player has sufficient

funds.

The system processes the withdrawal.

The system updates the player’s account balance.

**Extensions or Alternate Flows:**

If the input is invalid or funds are insufficient, the system prompts the user to enter a valid amount.

If the user cancels the transaction, they are returned to the previous screen.

**Exceptions:**

Dealers do not have the option to withdraw funds.

**Related Use Cases:**

Join Game (UC01)

Add Funds (UC02)

**Use Case ID:** UC04

**Use Case Name:** Manage Hand

**Relevant Requirements:** Players can interact with their hand during gameplay.

**Primary Actor:**

Players/dealer

**Pre-conditions:**

The app is functional and accessible.

**Post-conditions:**

The player’s hand is updated based on their actions(hit, stand, split, double down)

**Basic Flow or Main Scenario:**

**For Players:**

The players view their current hand, which displays the cards dealt

UC006.

The player decides on an action:

**Hit**

The player selects the “Hit” option.

The system deals one additional card to the player’s hand.

If the total exceeds 21, the player is bust, and the player loses immediately.

**Stand**

The player selects the "Stand" option.

The system retains the current hand and prepares for the next player’s turn.

**Split:**

**Condition:** Player is dealt a pair.

The system enables the "Split" option.

The player can select the option.

The dealer split the player's hand and dealt two new cards, and allowed the player to play each hand separately.

**Double Down:**

**Condition:** Cards dealt equal to 9,10 or 11

The system enables the “Double Down” option

The player can select the option.

The player bet is double

The system deals one more card to the player’s hand.

The player's turn ends after this action.

**For dealers:**

The dealer reveals their face-down card at the end of all players' turns if one player or more does not bust.

The dealer's final hand is compared to each player's hand to determine winners.

The dealer decides on actions based on house rules: (if player bust they have to collect their bet)

Hit:

* 1. If the dealer’s hand value is less than 17, the dealer must hit.
  2. The dealer deals one additional card to the dealer’s hand.
  3. If the total hand value exceeds 21, the dealer is bust.

Stand:

* 1. If the dealer’s hand value is 17 or higher, the dealer stands.

**Extensions or Alternate Flows:**

Players will not be able to perform illegal moves.

If the game round ends, the system updates player balances accordingly.

**Exceptions:**

If the system has any issue, an error message is displayed.

**Related Use Cases:**

Login (UC00)

Join Game (UC01)

Add Funds (UC02)

Withdraw Funds (UC03)

Leave Game(UC05)

Deal cards (UC06)

**Use Case ID:** UC05

**Use Case Name:** Leave Game

**Relevant Requirements:** Players can leave a game table when they are done playing.

**Primary Actor:**

Player, Dealer

**Pre-conditions:**

The app is functional and accessible and logged in.

The player has joined a game.

It is not the player’s turn.

**Post-conditions:**

The player is removed from the game table.

The player’s current game state is updated accordingly.

Other players and the dealer are notified of the player’s departure.   
Table state and attributes are updated.

**Basic Flow or Main Scenario:**

The player decides to leave the game and selects the “Leave Game” option. This will only appear upon the “Bet” option before a game starts, or during a game when it is not their turn.

The Player returns to the main menu with proper updates to their account  
Table states and attributes are updated

**Extensions or Alternate Flows:**

Dealer attempts to leave the game when it has not started.  
Everyone returns to the main menu with proper updates to their account  
Table is deleted.

**Exceptions:**

If the system is unavailable, it will display a message informing the user of the inconvenience.

**Related Use Cases:**

Login (UC00)

Join Game (UC01)

Add Funds (UC02)

Withdraw Funds (UC03)

Manage Hand (UC04)

Leave Game(UC05)

**Use Case ID:** UC06

**Use Case Name:** Deal cards

**Relevant Requirements:** Dealers deal cards to players and themselves at the beginning of each round.

**Primary Actor:**

Dealer

**Pre-conditions:**

The app is functional and accessible.

The dealer joins the game.

The table must have at least one player.

The game round has started, and the players have placed their bets.

**Post-conditions:**

The dealer deals two face-up cards at the beginning of the game to each player.

The dealer deals to himself one face-up card and one face-down card at the beginning of the game.

**Basic Flow or Main Scenario:**

The dealer verifies that all players have placed their bets.

The dealer shuffles the shoe if shoe is unavailable.

The dealer begins dealing cards.

Players receive two cards face up.

The dealer’s first card is face down, and the second is face up.

The dealer then deals another card according to Manage hand UC004

**Extensions or Alternate Flows:**

Dealers deal cards to players based on their actions during the game.

The dealer identifies that it is the player’s turn to take action.

The dealer verifies the player’s request to hit.

The dealer draws the next card from the deck:

The dealer takes one card from the top of the shuffled deck.

The dealer places the drawn card face up in front of the player.

**Exceptions:**

If the system is unavailable, it will display a message informing the user of the inconvenience.

If the shoe runs low on cards during the dealing process, the dealer must shuffle and redeal before proceeding.

**Related Use Cases:**

Join Game (UC01)

Manage Hand (UC04)

Leave Game(UC05)

**Use Case ID:** UC07

**Use Case Name:** Shuffle Cards

**Relevant Requirements:** Dealer in the game

**Primary Actor:**

Dealer

**Pre-conditions:**

The app is functional and accessible.

The dealer joins the game.

The table must have at least one dealer.

The dealer has chosen to shuffle cards

**Post-conditions:**

**Basic Flow or Main Scenario:**

The dealer verifies that all players have placed their bets.

The dealer shuffles the shoe.

The shoe is now created with a fresh set of cards.

**Extensions or Alternate Flows:**

**Exceptions:**

If the system is unavailable, it will display a message informing the user of the inconvenience.

Shoe cannot be reshuffled if the shoe card amount is not requiring a shuffle

**Related Use Cases:**

**Use Case ID:** UC08

**Use Case Name:** Create Table (Create Game)

**Relevant Requirements:** Dealer is present.

**Primary Actor:**

User: dealer

**Pre-conditions:**

The app is functional and can be used and the user must have been logged in.

The user attempting the method is a Dealer

**Post-conditions:**

Dealer will have one table created for them to host a Blackjack game..

**Basic Flow or Main Scenario:**

Dealer decides to create a game.

Wait for the dealer to begin game and deal cards

**Extensions or Alternate Flows:**

**Exceptions:**

If the system finds that the Dealer already has a Table in their name, an error ill be sent to prevent another instance being made.

**Related Use Cases:**

**Use Case ID:** UC09

**Use Case Name:** Start Table (Start Game)

**Relevant Requirements:** Dealer is present, Player(s) are present

**Primary Actor:**

User: dealer

**Pre-conditions:**

The user attempting the method is a Dealer

At least 1 player is present at the table, already inputted a bet

**Post-conditions:**

The blackjack game will begin, and will select the first player to manage their hand.

**Basic Flow or Main Scenario:**

Dealer selects to “Start Game”  
Dealer is prompted to shuffle and deal cards from the shoe.

Player’s states will be updated accordingly to provide a turn based system

**Extensions or Alternate Flows:**

**Exceptions:**

**Related Use Cases:**

Manage Hand (UC04)

**Use Case ID:** UC10

**Use Case Name:** Betting

**Relevant Requirements:** Manage hand.

**Primary Actor:**

User: player

**Pre-conditions:**

The app is functional and can be used and the user must have been logged in.

The user attempting the method is a Player

Player is in the table

The player must have available funds

**Post-conditions:**

The player is able to bet their amount to play a game

**Basic Flow or Main Scenario:**

Dealer decides to create a game.

Game is waiting for players to join

Player joins the game

Player makes bet to play

**Extensions or Alternate Flows:**

Player does not bet

Game skips player

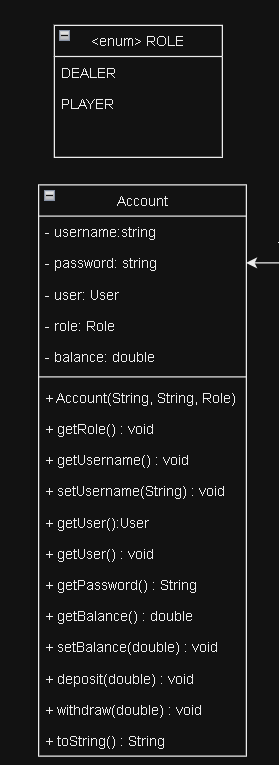
**Exceptions:**

If the system finds that the Dealer already has a Table in their name, an error ill be sent to prevent another instance being made.

**Related Use Cases:**

## 

## 4.2 Classes

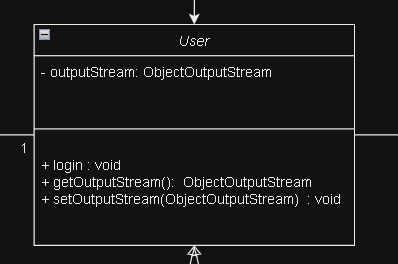
****

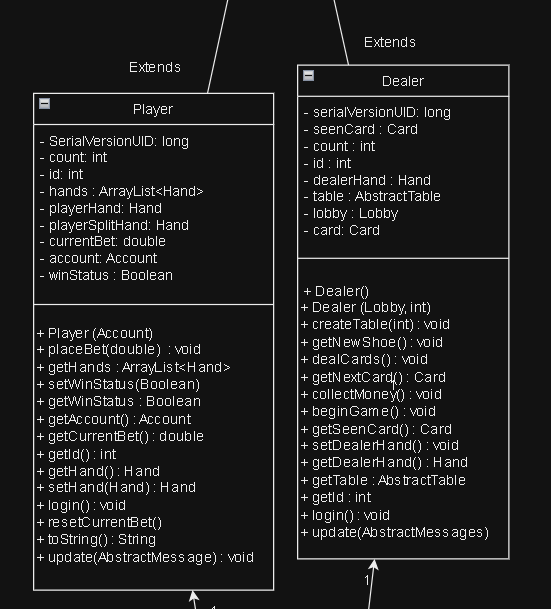
**Class Name:** Account

**Attributes and Methods:**

* username & password to create an account.
* user: Sets the type of account
* deposits(amount): Adds the specified amount to the balance.
* Role: represents the role of the user.
* balance: Represents the amount of money in a user account
* withdraw(amount): Subtracts the specified amount from the account balance.
* deposit:(amount): Adds the specified amount to the account balance
* getUsername: Returns the username of the account
* setUsername: Sets the username of the account
* getUser: get the User object
* getPassword: Returns the password of the account
* getBalance: Returns the current balance of the account
* setBalance(double): Sets the balance of the account
* getRole: returns the role that is linked with the account.
* rank : Represents the ranks, with face cards counted as 10 and Ace as 1 or 11.
* toString: Prints out the account information

**Class Name:** User



****

**Attributes and Methods:**

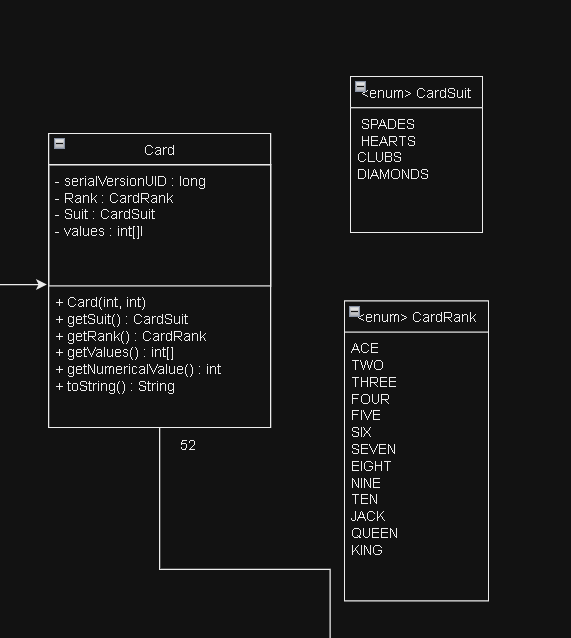
**Player**

* count: static methods for the id
* id: id for the player account
* hands: Has a list of hands
* playerHand: What is current in the players hand
* playerSplitHand: Cards that can be split
* currentBet: double that hold the best
* account: Holds the proper account for the user
* winStatus: true or false given that the player is a winner
* Player(Account): Constructor for the player account
* placeBet(Double): Places a bet given input
* getHands: Return the hands in the player
* setWinStatus: Sets the win status
* getWinStatus: Returns the win status
* getAccount: Returns the account
* getCurrentBet: Returns the current bet
* getId: Returns the Id
* getHand: Returns the hand
* setHand: Sets the hand object
* login: Login for the player
* resetCurrentBet: Resets the current bet
* update(AbstractMessage): Updates the player account for the server.

**Dealer**

* seenCard: Holds the screen hard from the dealer
* count: static methods for the id
* id: id for the dealer account
* dealerHand: The cards the dealer has in their hand
* table: Hold the table made by the dealer
* lobby: holds the lobby made by the dealer
* Card: holds the card object
* Dealer(Lobby, Int): Creates the lobby with a unique number
* createTable(int): Creates the table to show up in a list
* dealCards: deals the cards to the user
* getNextCard: Gets the next card within the shoe
* collectMoney(): collects the money from all the players
* beginGame: Starts the game
* getSeenCard(): From the dealer hand object, get their card
* setDealerhand(Hand dealerHand): set the dealer hand
* getDealerhand(): Returns the dealers hand
* getTable: Returns the table
* getId: Returns the id
* update(AbstractMessage): Updates the dealer object

**Class Name:** Card



**Attributes and Methods:**

* suit : Represents the four suits in a deck of cards.
* rank : Represents the ranks, with face cards counted as 10 and Ace as 1 or 11.
* Card class: Has a constructor that initializes the suit and rank.
* getSuit: This method returns the suit value of the card.
* getRank: This method returns the name of the rank as a string.
* getValues: Returns the values
* getNumericalValue: Returns an numerical value of the card

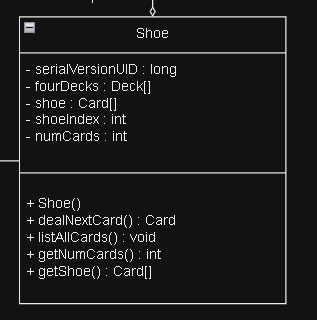
**Class Name:** Deck



**Attributes and Methods:**

* deck: represents a full deck of cards.
* Deck() : Constructor generates a standard 52-card deck by combining suits and values.
* shuffleDeck: shuffles the cards.
* getDeck(): gets the deck of cards
* getCard(int): returns the card object from deck

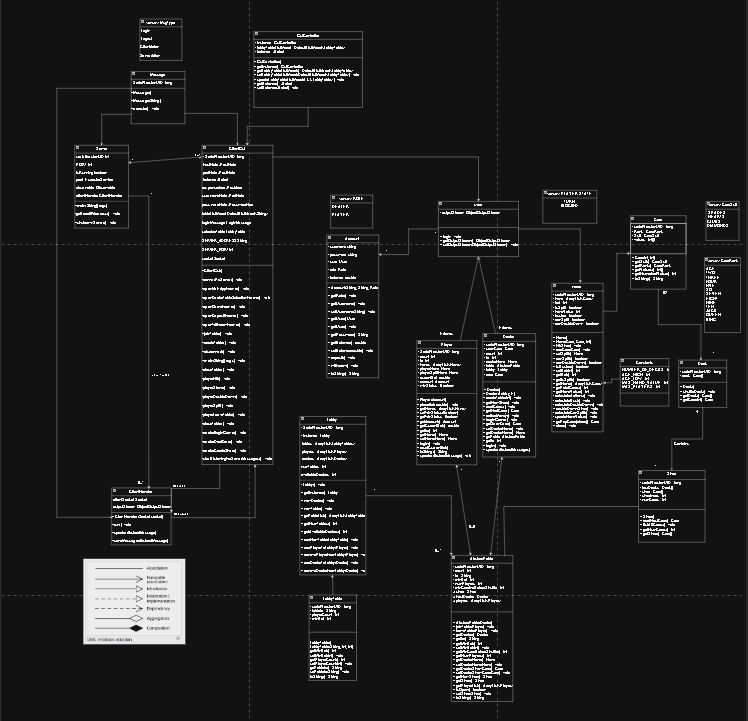
**Class Name:** Shoe



**Attributes and Methods:**

* fourDecks : This attribute holds multiple Deck objects, each representing a standard deck of cards.
* shoe: This attributes hold the multiple cards
* shoeIndex: holds the index of the shoe
* numCards: holds the number of cards in the shoe
* Shoe(): constructor that uses the number of decks.
* dealNextCard(): deals the next card in the shoe
* getNumCards(): return the number of cards
* getShoe(): return the shoe object

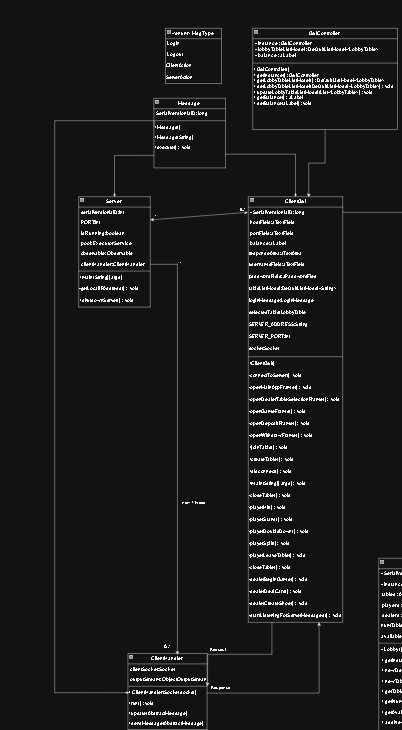
## 4.3 Class Diagram

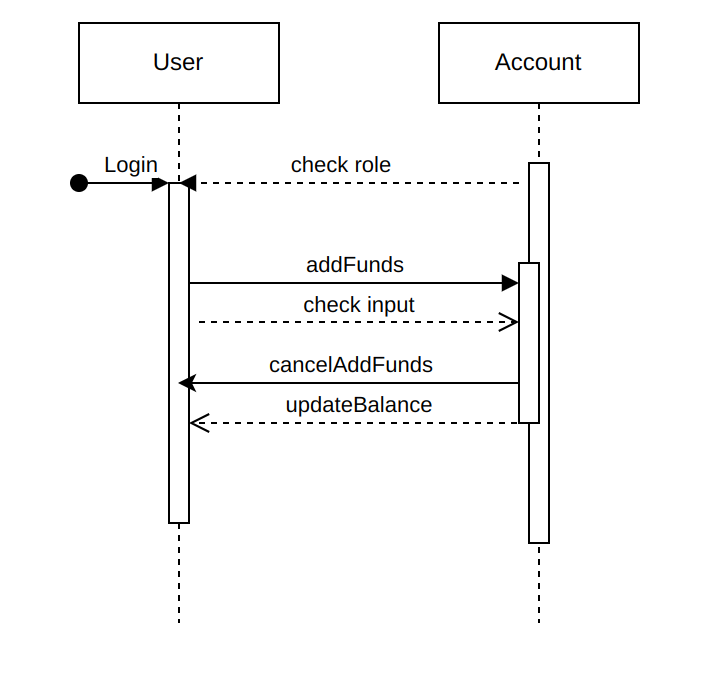
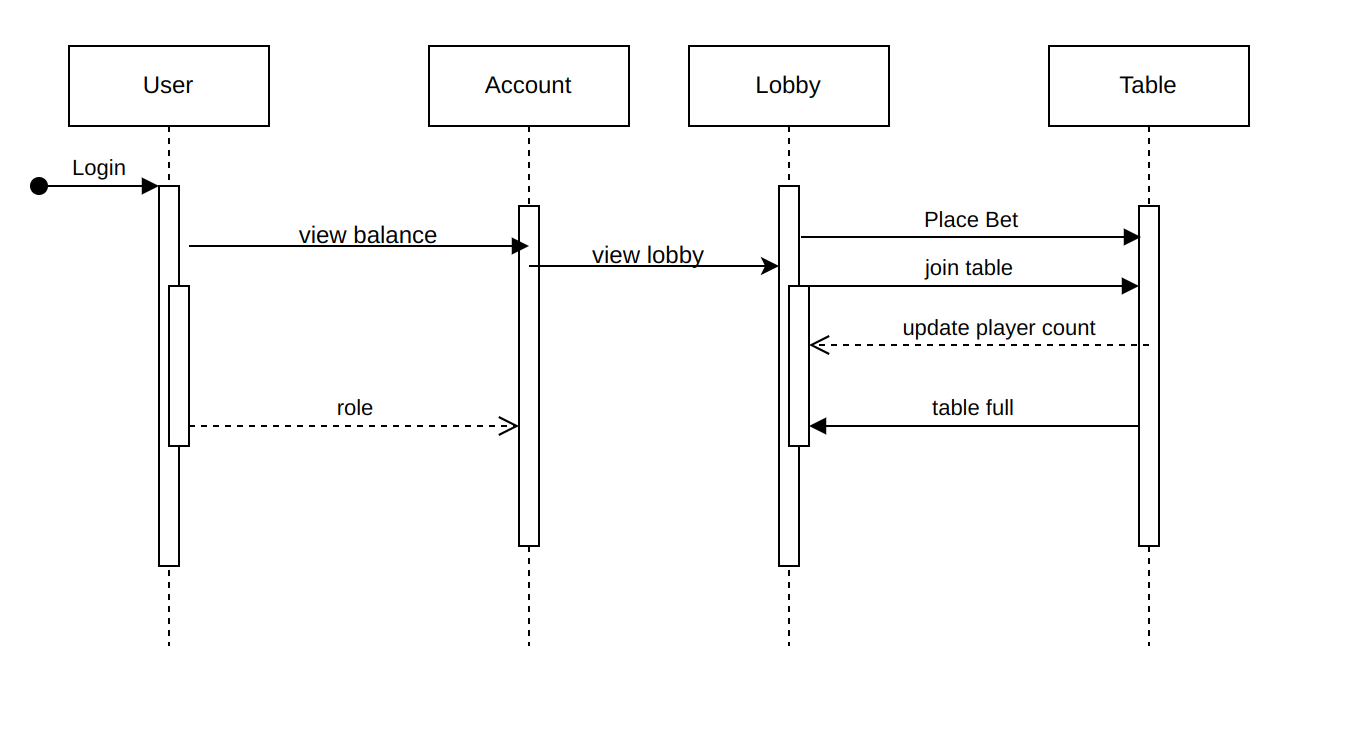


## 

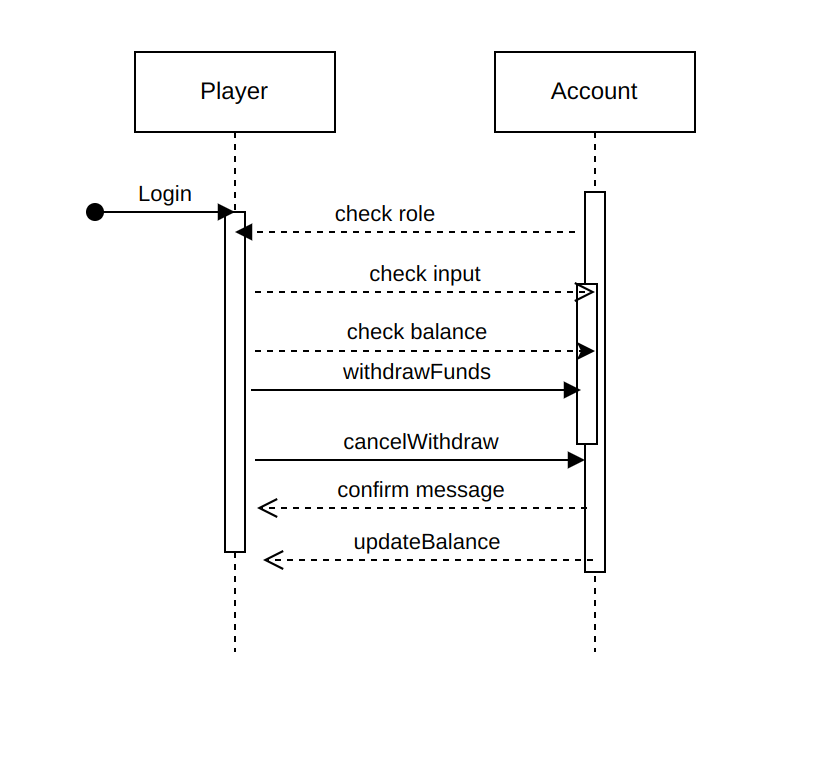
## 4.4 Sequence Diagram

Join Game

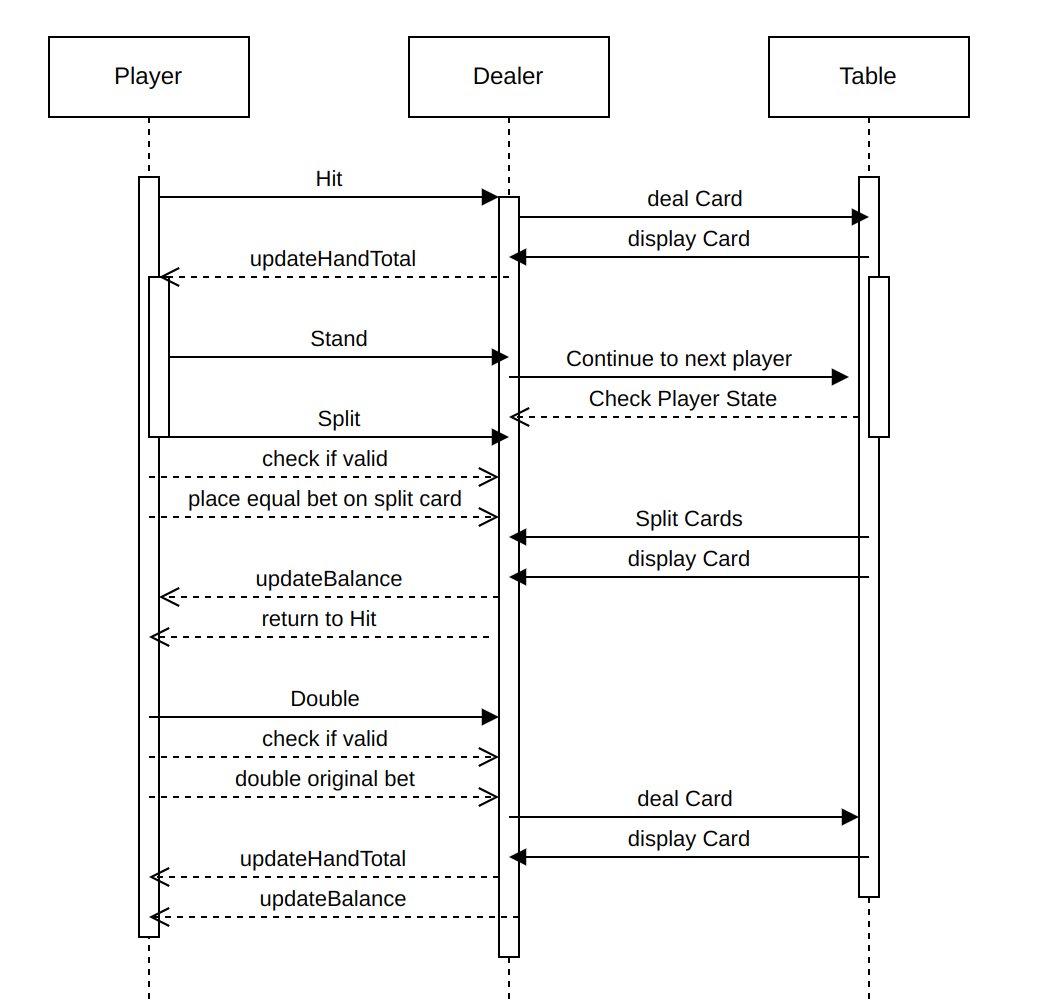




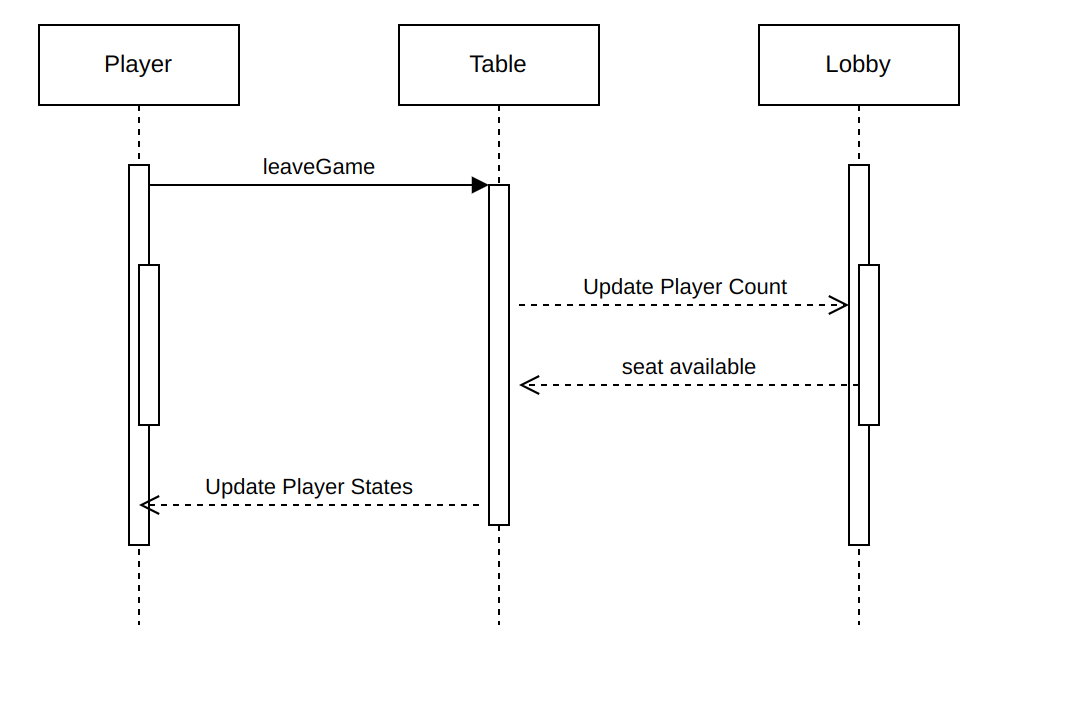
Add Funds

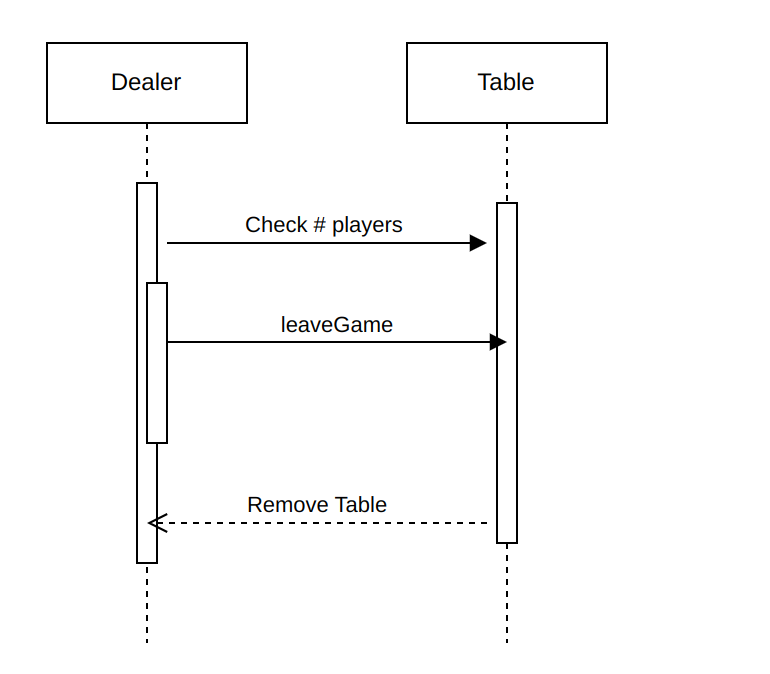
Withdraw 

Manage Hand:



Leave Game:





Deal Cards:

