POKER GAME USING JAVA

Team members:

Name: Akash Shankar SRN: PES1UG20CS028

Name: Ansh CS SRN: PES1UG20CS061

Name: Anirudh C SRN: PES1UG20CS056

Name: Aditya E SRN: PES1UG20CS019

Team ID: A9

Section: A

Aim :

To recreate a fully functional game of poker which consists of 4 players and one dealer using java programming language.

Technicals details :

● We plan to implement a client server architecture, which consists of a main server(dealer) and 3 client servers (players).

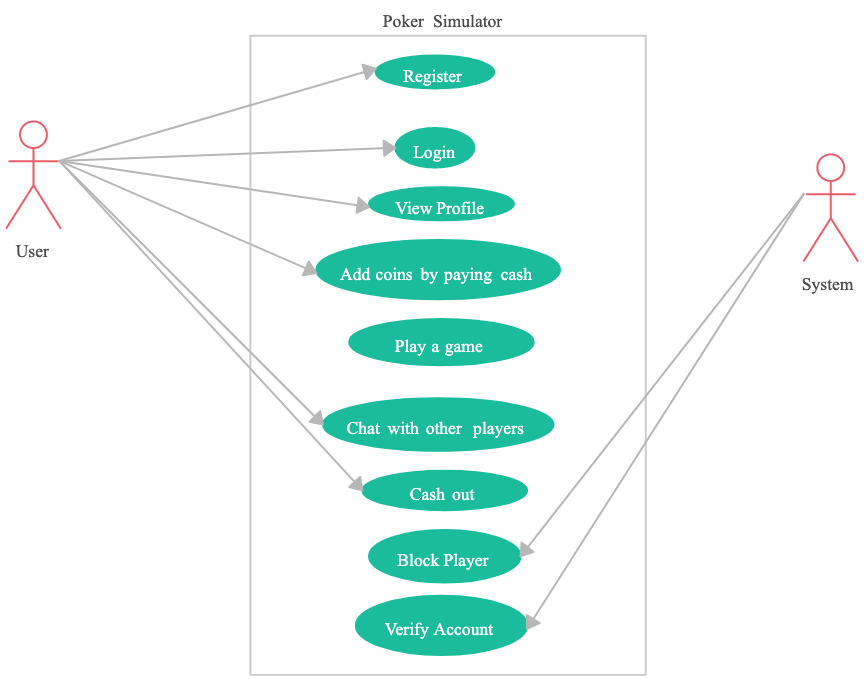
● The clients can only communicate with the dealer and not with one another. Players will not be allowed to view each other's cards.

● We plan to implement the game using a web based UI.

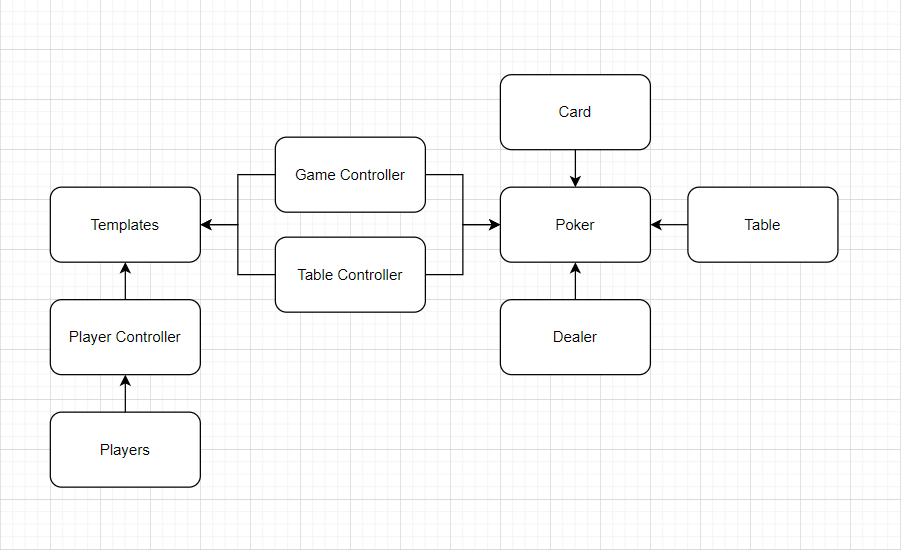
● Classes :

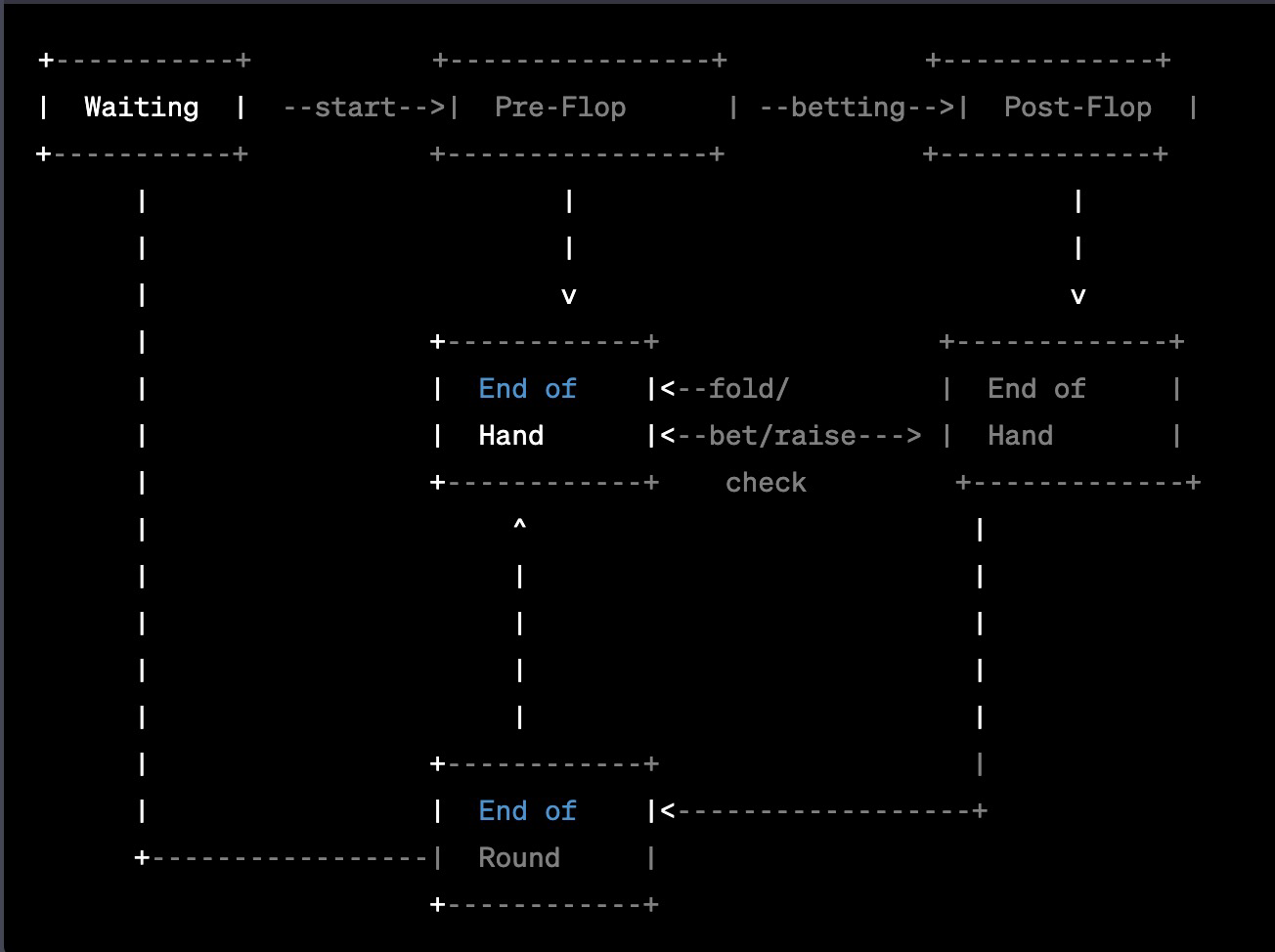
* Dealer : The dealer is one of the classes which contains the main deck of cards and is responsible for shuffling and distributing cards to players and revealing the Flop, Turn and River cards. It is also responsible for deciding who has the best hand and has won the game.
* Player : The player is allowed to make calls, checks and fold. He is also allowed to view current pot size and amount of money with him.
* Card : It specifies which suit a particular card belongs to (hearts, clubs, spade, diamond) and its value (2,3,4,5,6,7,8,9,10,J,Q,K,A)
* Pokesh - handles all the game logic.
* Game Controller - controls the working of the game by calling necessary functions from Pokesh.
* Player Controller - deals with accepting players for the game and moves made by each player.
* Table Controller - deals with activities that happen on the poker table like shuffling cards, dealing cards, etc.
* Table - model class containing the logic behind table.
* Choose\_Winner - chooses the winner using the game logic.

Use Case Diagram



Class Diagram



State Diagram

Activity Diagram



Design Principles and Patterns Used:

Observer Pattern: This pattern can be used to implement the communication between the Model and the View. The Model can notify the View whenever there is a change in the game state (e.g., a player's turn, a new card is dealt, etc.) and the View can update itself accordingly.

Command Pattern: This pattern can be used to encapsulate player actions as commands that can be executed or undone.

Factory Pattern: This pattern can be used to create different types of objects (e.g., cards, players, hands, etc.) in a centralized way. A factory class can be created for each type of object, and the factory can create objects based on the parameters passed to it.

Singleton Pattern: This pattern can be used to ensure that there is only one instance of certain classes (e.g., the game controller). This can be useful in ensuring that the game state is consistent and that there are no conflicts between different instances of the same class.

Open-Closed Principle (OCP): The system should be open for extension but closed for modification. For example, the Game class should be designed in a way that new types of poker games can be easily added without modifying the existing code.

Interface Segregation Principle (ISP): Clients should not be forced to depend on interfaces they do not use. For example, if the Table class has methods for shuffling and dealing cards, then the Player class should not be forced to implement these methods if it does not use them.