Poker: Texas HoldEm Manual

Per Astrom, Lindsay Knupp, Guillermo Torres, Callie Valenti

Table of Contents

Introduction

We have created a multiplayer, graphical user interface based game of Texas HoldEm Poker. To play, users first connect to the server using the server's IP address. The server's IP address is displayed on the server's screen and must be verbally exchanged to the players (if playing on multiple computers) or copied over from an adjacent run window (if playing on one computer). Then, a JavaFX window pops up where game play is recorded. Each player is dealt two cards initially which can be viewed at the bottom right corner of the screen. For the first round, a common table card is placed on the table so that each connected user can view it. Each player then has the choice to either bet, check, or fold. To bet, users type in an amount into the box and select bet. To check or fold, users press the respective buttons. After a user takes their turn, they must type the word "Go" into the terminal to progress gameplay. After all players have taken their turn, the pot on the table is updated with everyone's bet amount and the round is finished. Game play continues in a similar fashion until there are a total of five cards on the table. Finally, a winner is declared and awarded the total pot amount.

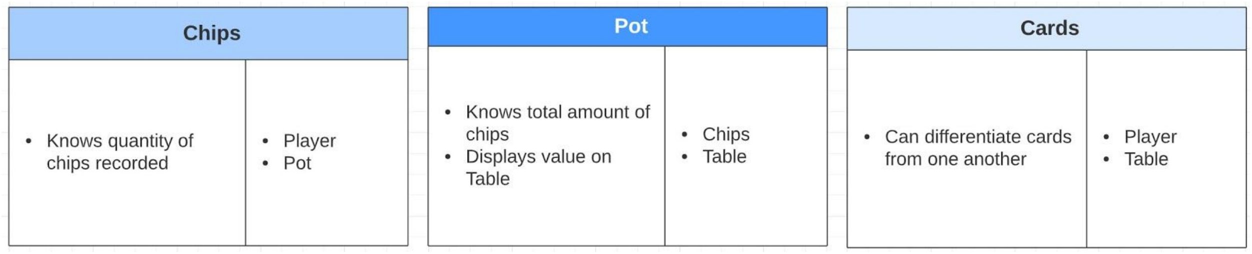
User Stories

Priority:	Story:
1	As a dealer, I want to be able to deal cards to the players, so that the game can start
2	As a dealer, I want to display cards to the screen, so that the game can start
3	As a player, I want to be able to look at multiple cards, so that I can view my hand
4	As a player, I want to be able to check, bet, or fold, so that I can complete my turn
5	As a player, I want to be able to connect to a network with other players
6	As a player, I want to be able to view the table
7	As a player, I want to be able to view the same set of cards as the other players
8	As a player, I want to be able to see the shared Pot on the Table
9	As a player, I want to be able to interact with game to select the action I want to take and how much I want to bet
10	As a dealer, I want to be able to synchronize the game between all of the players
11	As a player, I want to be able to see the different hands I can get
12	As a player, I want to see the amount of chips I have during the game

OOD

Initial Game Flow

When first tasked with the goal of creating a Poker Game, we thought about the different classes we would need to represent the gameflow. Our first few user stories focused on differentiating between a player and a dealer. What specific responsibilities or objects would a player need to possess? We decided that we would need to create classes to represent betting chips, a pot, and a deck of cards.

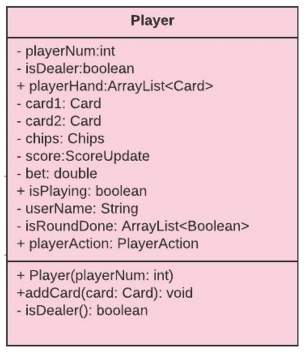


Chips	
• Knows quantity of chips recorded	• Player • Pot

Pot	
• Knows total amount of chips • Displays value on Table	• Chips • Table

Cards	
• Can differentiate cards from one another	• Player • Table

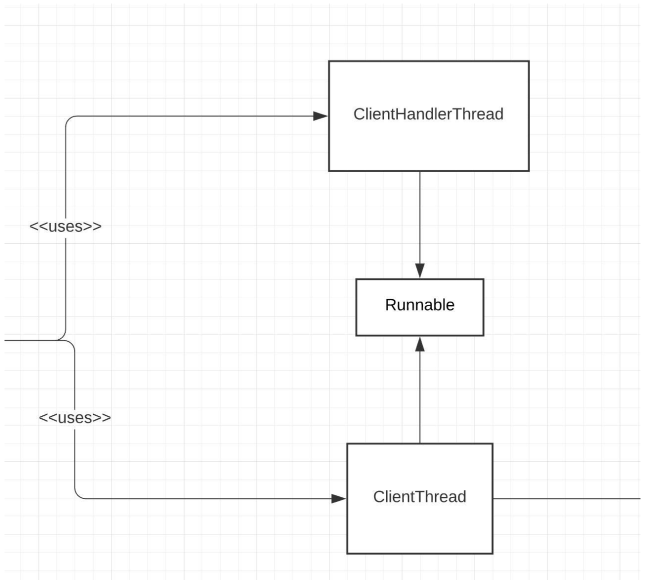
These objects represented the essential components of a Poker game. From there we were able to encapsulate all of the information with a Player class. We decided that every time a person would connect to the game, a new Player object would be created where they could keep track of their chips, personal cards, and their actions. Eventually, as we continued adding details to the game, our Player class became more and more complex.

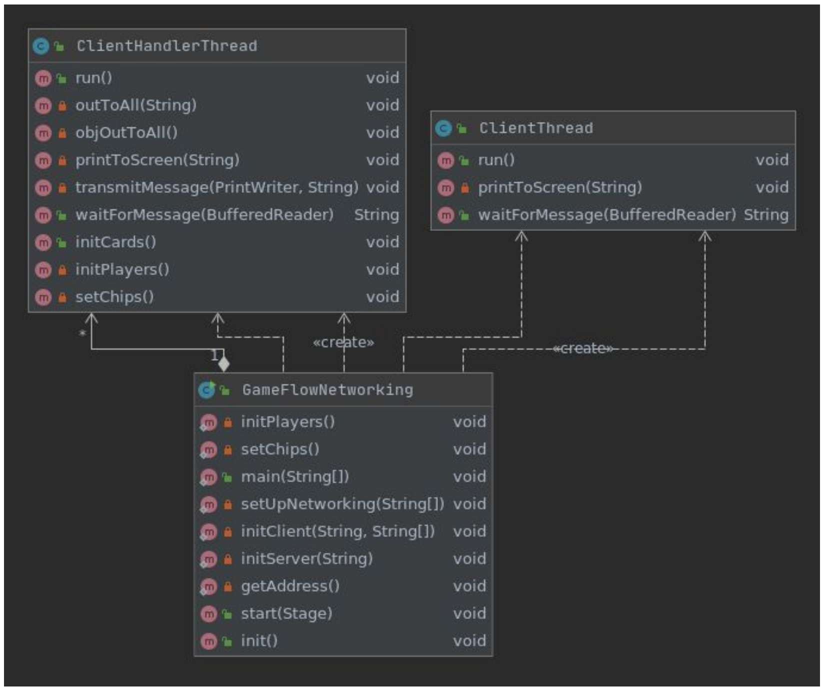


```
Player
- playerNum:int
- isDealer:boolean
+ playerHand:ArrayList<Card>
- card1: Card
- card2: Card
- chips: Chips
- score:ScoreUpdate
- bet: double
+ isPlaying: boolean
- userName: String
- isRoundDone: ArrayList<Boolean>
+ playerAction: PlayerAction
+ Player(playerNum: int)
+addCard(card: Card): void
- isDealer(): boolean
```

Networking

After we established the initial game flow for a round of Poker with all of the necessary components, we were able to start networking. We needed to use multi-threading in order to connect multiple clients to one server. Thus, we created a ClientHandlerThread for the server and a ClientThread for the clients implementing the Runnable interface. Next, we knew we had to have another object, like a Table. The table class is the model responsible for the logic behind the table section on the graphical user interface and encapsulated the previously mentioned classes.





Overview of more complex networking

After a secure connection was established between the Clients and Server we were able to create the Table object.

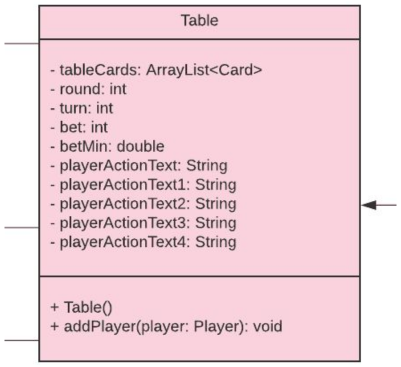
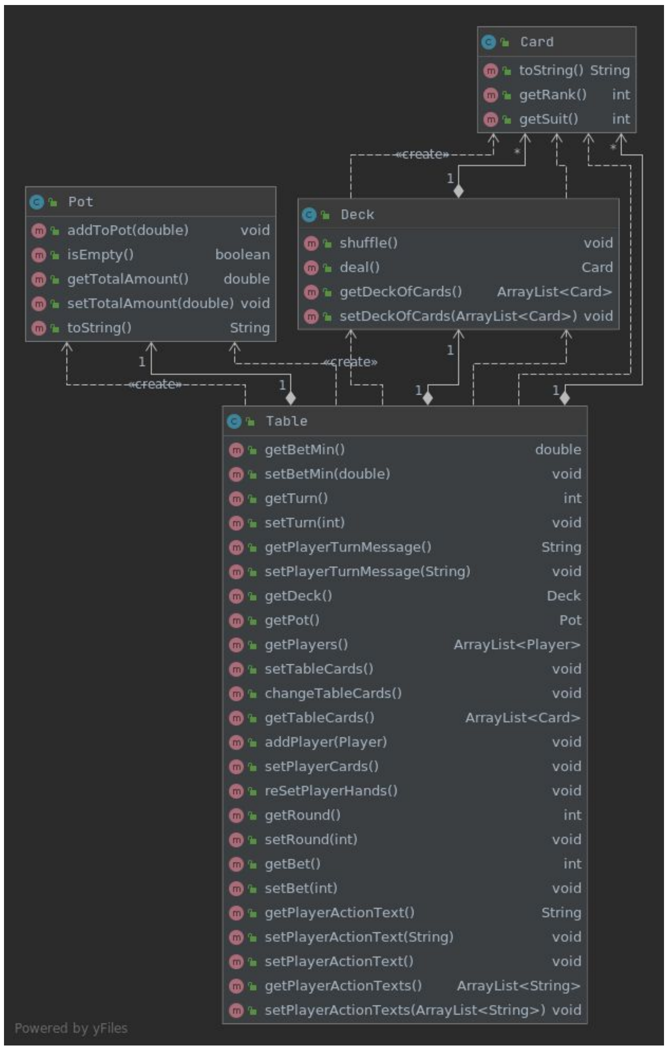


Table
- tableCards: ArrayList<Card>
- round: int
- turn: int
- bet: int
- betMin: double
- playerActionText: String
- playerActionText1: String
- playerActionText2: String
- playerActionText3: String
- playerActionText4: String
+ Table()
+ addPlayer(player: Player): void

The table included fields like the Table cards, the player actions, and "counter" fields which kept track of the round, who's turn it was, and the amount bet.

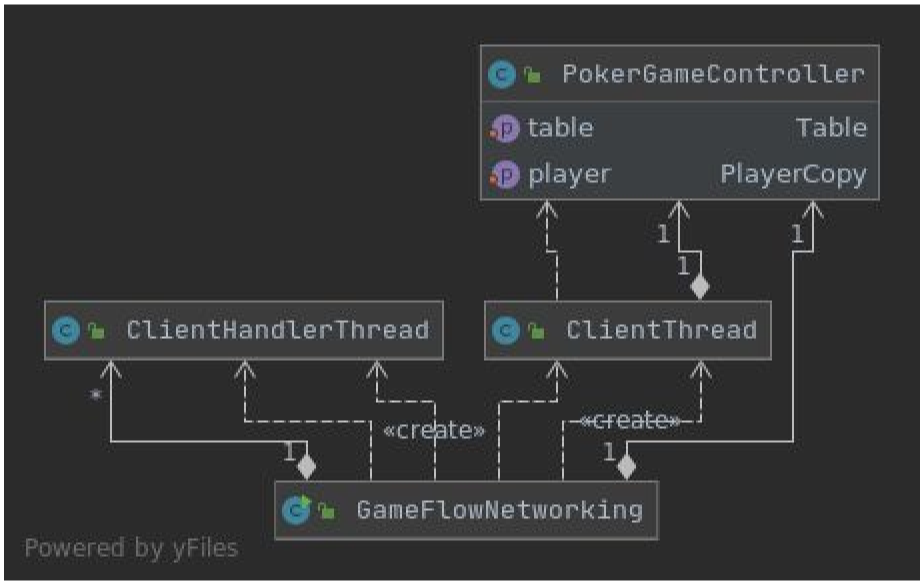


The Table class needed to be connected to the classes which were shared with everyone who was connected to the game.

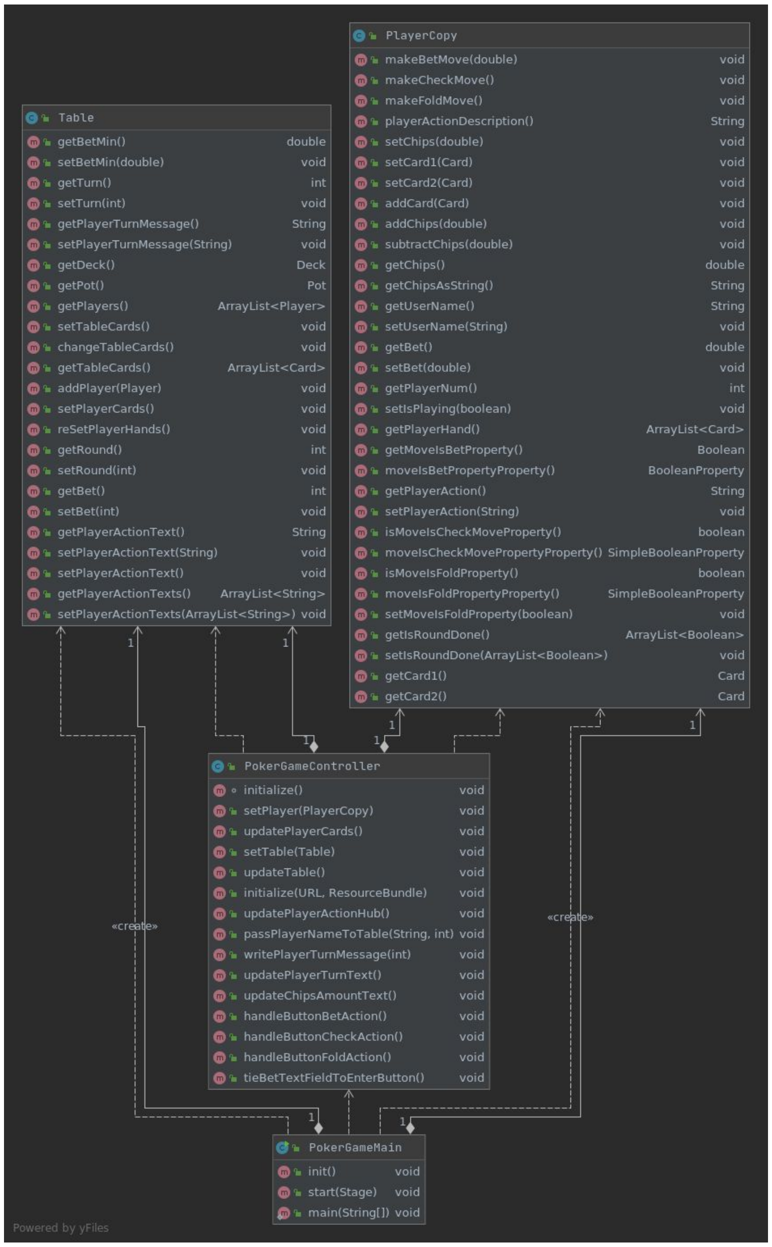
GUI

Finally, after being able to send the Table object between the Server and the Clients, we needed to integrate the GUI. In order for all the clients to be able to view the Table, we needed to implement the controller inside the ClientThread class. This way, whenever a new client connected to the Server, a GUI would be generated that they could interact with.

When designing the GUI, the idea was to look for an appealing and simple visualization for the game. We wanted to use the green poker table background for the Pane that visualizes the playing cards and I think that the green used in each rectangle inside helps distinguish the difference between where information will be visualized. For the Pane on the right of the GUI, we decided to show a list with the different possible hands in Texas hold'em focusing on new poker players who do not know all the hands. In the bottom Pane, the team decided to make it each player's functionality. There each individual player will be to place their bets, check, and fold their cards and also see their individual cards and chip amount. The GUI was made to seem simple for new players and be easy to follow.



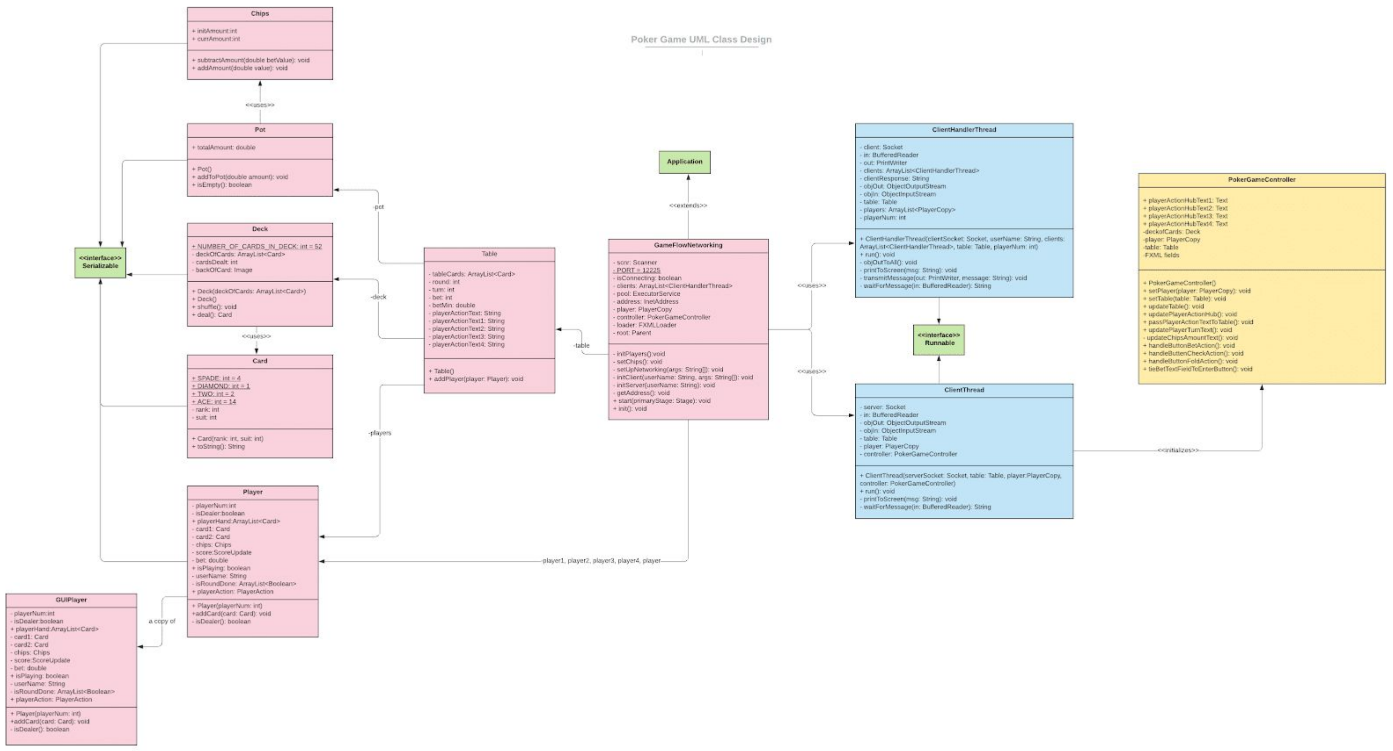
Although it is not possible to display the fxml view file as a UML diagram in IntelliJ, we can see how the controller class is part of the ClientThread. We also see how it binds the Table and Player classes. The controller then binds these with the view.



PlayerCopy
makeBetMove(double) void
makeCheckMove() void
makeFoldMove() void
playerActionDescription() String
setChips(double) void
setCard1(Card) void
setCard2(Card) void
addCard(Card) void
addChips(double) void
subtractChips(double) void
getChips() double
getChipsAsString() String
getUserName() String
setUserName(String) void
getBet() double
setBet(double) void
getPlayerNum() int
setIsPlaying(boolean) void
getPlayerHand() ArrayList<Card>
getMoveIsBetProperty() Boolean
moveIsBetPropertyProperty() BooleanProperty
getPlayerAction() String
setPlayerAction(String) void
isMoveIsCheckMoveProperty() boolean
moveIsCheckMovePropertyProperty() SimpleBooleanProperty
isMoveIsFoldProperty() boolean
moveIsFoldPropertyProperty() SimpleBooleanProperty
setMoveIsFoldProperty(boolean) void
getIsRoundDone() ArrayList<Boolean>
setIsRoundDone(ArrayList<Boolean>) void
getCard1() Card
getCard2() Card
Table
getBetMin() double
setBetMin(double) void
getTurn() int
setTurn(int) void
getPlayerTurnMessage() String
setPlayerTurnMessage(String) void
getDeck() Deck
getPot() Pot
getPlayers() ArrayList<Player>
setTableCards() void
changeTableCards() void
getTableCards() ArrayList<Card>
addPlayer(Player) void
setPlayerCards() void
reSetPlayerHands() void
getRound() int
setRound(int) void
getBet() int
setBet(int) void
getPlayerActionText() String
setPlayerActionText(String) void
setPlayerActionText() void
getPlayerActionTexts() ArrayList<String>
setPlayerActionTexts(ArrayList<String>) void
1
1
1
1
1
1
PokerGameController
initialize() void
setPlayer(PlayerCopy) void
updatePlayerCards() void
setTable(Table) void
updateTable() void
initialize(URL, ResourceBundle) void
updatePlayerActionHub() void
passPlayerNameToTable(String, int) void
writePlayerTurnMessage(int) void
updatePlayerTurnText() void
updateChipsAmountText() void
handleButtonBetAction() void
handleButtonCheckAction() void
handleButtonFoldAction() void
tieBetTextFieldToEnterButton() void
«create»
«create»
1
1
PokerGameMain
init() void
start(Stage) void
main(String[]) void
Powered by yFiles

A more detailed view of the GUI using SceneBuilder. The GUI consists of an fxml view file, a controller class, and two models: player and table.

Lastly, as we had all the components created, we were able to successfully run a game of Poker with multiple clients.


Poker Game UML Class Design
Chips
+ initAmount:int
+ currAmount:int
+ subtractAmount(double betValue): void
+ addAmount(double value): void
<<uses>>
Pot
+ totalAmount: double
+ Pot()
+ addToPot(double amount): void
+ isEmpty(): boolean
Deck
+ NUMBER_OF_CARDS_IN_DECK: int = 52
- deckOfCards: ArrayList<Card>
- cardsDealt: int
- backOfCard: Image
+ Deck(deckOfCards: ArrayList<Card>)
+ Deck()
+ shuffle(): void
+ deal(): Card
<<interface>> Serializable
<<uses>>
Card
+ SPADE: int = 4
+ DIAMOND: int = 1
+ TWO: int = 2
+ ACE: int = 14
- rank: int
- suit: int
+ Card(rank: int, suit: int)
+ toString(): String
Player
- playerNum:int
- isDealer:boolean
+ playerHand:ArrayList<Card>
- card1: Card
- card2: Card
- chips: Chips
- score:ScoreUpdate
- bet: double
+ isPlaying: boolean
- userName: String
- isRoundDone: ArrayList<Boolean>
+ playerAction: PlayerAction
+ Player(playerNum: int)
+addCard(card: Card): void
- isDealer(): boolean
a copy of
GUIPlayer
- playerNum:int
- isDealer:boolean
+ playerHand:ArrayList<Card>
- card1: Card
- card2: Card
- chips: Chips
- score:ScoreUpdate
- bet: double
+ isPlaying: boolean
- userName: String
- isRoundDone: ArrayList<Boolean>
+ playerAction: PlayerAction
+ Player(playerNum: int)
+addCard(card: Card): void
- isDealer(): boolean
-pot
-deck
-players
Table
- tableCards: ArrayList<Card>
- round: int
- turn: int
- bet: int
- betMin: double
- playerActionText: String
- playerActionText1: String
- playerActionText2: String
- playerActionText3: String
- playerActionText4: String
+ Table()
+ addPlayer(player: Player): void
-table
Application
<<extends>>
GameFlowNetworking
- scnr: Scanner
- PORT = 12225
- isConnecting: boolean
- clients: ArrayList<ClientHandlerThread>
- pool: ExecutorService
- address: InetAddress
- player: PlayerCopy
- controller: PokerGameController
- loader: FXMLLoader
- root: Parent
- initPlayers():void
- setChips(): void
- setUpNetworking(args: String[]): void
- initClient(userName: String, args: String[]): void
- initServer(userName: String): void
- getAddress(): void
+ start(primaryStage: Stage): void
+ init(): void
-player1, player2, player3, player4, player
<<uses>>
<<uses>>
ClientHandlerThread
- client: Socket
- in: BufferedReader
- out: PrintWriter
- clients: ArrayList<ClientHandlerThread>
- clientResponse: String
- objOut: ObjectOutputStream
- objIn: ObjectInputStream
- table: Table
- players: ArrayList<PlayerCopy>
- playerNum: int
+ ClientHandlerThread(clientSocket: Socket, userName: String, clients: ArrayList<ClientHandlerThread>, table: Table, playerNum: int)
+ run(): void
- objOutToAll(): void
- printToScreen(msg: String): void
- transmitMessage(out: PrintWriter, message: String): void
- waitForMessage(in: BufferedReader): String
<<interface>> Runnable
ClientThread
- server: Socket
- in: BufferedReader
- objOut: ObjectOutputStream
- objIn: ObjectInputStream
- table: Table
- player: PlayerCopy
- controller: PokerGameController
+ ClientThread(serverSocket: Socket, table: Table, player:PlayerCopy, controller: PokerGameController)
+ run(): void
- printToScreen(msg: String): void
- waitForMessage(in: BufferedReader): String
<<initializes>>
PokerGameController
+ playerActionHubText1: Text
+ playerActionHubText2: Text
+ playerActionHubText3: Text
+ playerActionHubText4: Text
-deckofCards: Deck
-player: PlayerCopy
-table: Table
-FXML fields
+ PokerGameController()
+ setPlayer(player: PlayerCopy): void
+ setTable(table: Table): void
+ updateTable(): void
+ updatePlayerActionHub(): void
+ passPlayerActionTextToTable(): void
+ updatePlayerTurnText(): void
- updateChipsAmountText(): void
+ handleButtonBetAction(): void
+ handleButtonCheckAction(): void
+ handleButtonFoldAction(): void
+ tieBetTextFieldToEnterButton(): void