Program Documentation

1. Class Description

GameConfig.java	This includes sharable fields such as playerScore, playerTurn, maxGuess, and isEnd. These fields are used to check each player's game conditions.
GameLobby.java	This class is a game lobby where players are joined (accepted) into the thread (eventually to the server). Afterwards, it initialises the game conditions and runs the game based on the number of players. The communication log is also written in this class.
GuessGameThread.java	This is committed to running the actual game. It includes the game flow (operations).
MultiPlayer.java	This class is a player (client) side. It is a method where every process is included. Hence, this class is technically a player itself.
MultiServer.java	This opens a game lobby thread and record the communication log into the text file.
ServerCoordinator.java	This is a helper class for the server, therefore, it checks all the validations for both the server and the clients, generates the secret code and compare player guess code with the secret code. As well as that, it writes down game log into the text file.
SingleServer.java	This is a single client player class. It works almost similar to MultiPlayer class.
SinglePlayer.java	This is for a single playe game. It only allows one client connection (backlog is 1) and communicates with the client directly.

2. Program operation

Single Player Game

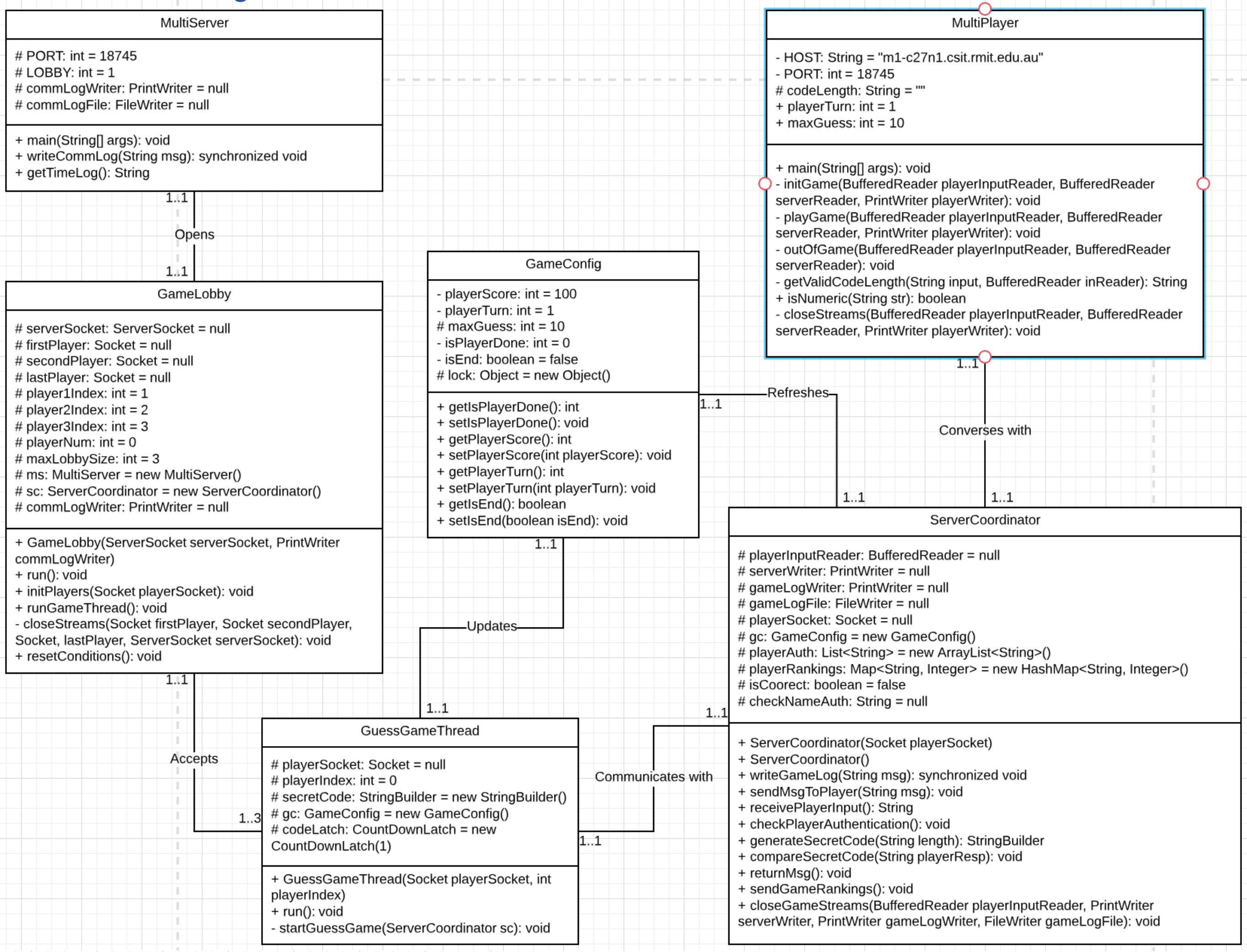
- **SingleServer**
  - Once the server has been run, it waits for a player to be connected to it. Afterwards, it asks the player to set the secret code length. Then, the server generates the secret code. As soon as the user is notified that "the secret code has been generated", the game starts. During the game, it checks the user's guess code with the secret code and gives back some hints (correct digits with correct positions, and correct digits with incorrect positions). This runs 10 times unless the player guesses before the turn reaches out to the maximum guess (10). At the end, it tells the player about the player's score, the secret code and a few more messages.
- **SingleClient**
  - The player is connected to the server once this has been run. At the beginning, the player is asked to set the secret code length. This also checks the code length validation. After sending code length to the server, the player waits until the server generates the secret code successfully. Then, the game starts with a few messages. The player guesses 10 times unless the player guesses the secret code before reaching out the max guess (10). During the process, the program checks input validation (whether or not the player guess is numeric, valid length etc.) At the end, the player is notified if they have won or lost. Both cases, the player is announced about their game play details including score, secret code and a few more ending messages.

Multi Player Game

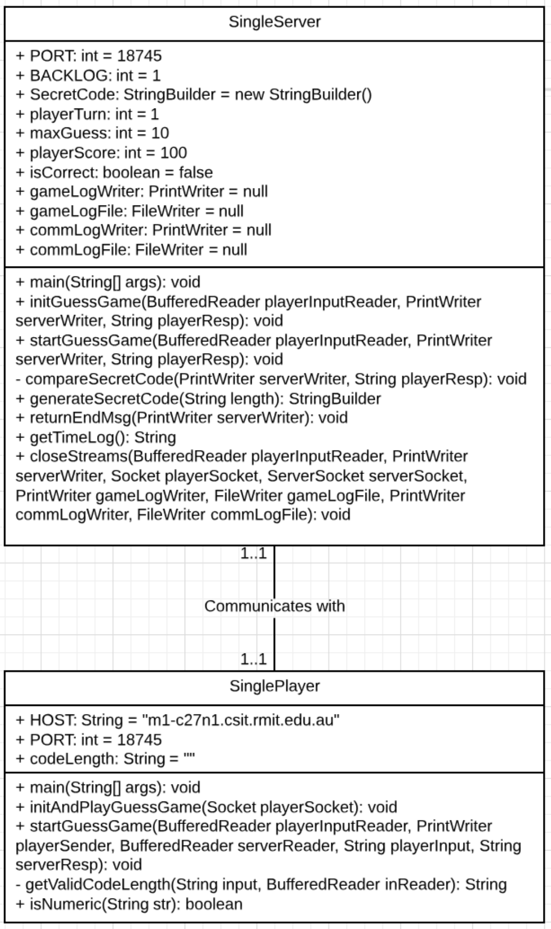
- **MultiServer**
  - Once this class has been operated, it opens a game lobby (thread). In the game lobby, it waits for players (sockets) to join (accepted) the game. While waiting for player connections, it asks the user whether or not he wants to wait for another player connection. If say yes, it waits for another connection, otherwise, just runs the game based on the number of players (threads) connected to the server. The GuessGameThread is committed to manipulating the communication with player(s). Hence, it declares an object of ServerCoordinator which is responsible for helping server work and proceeds the game flow. To begin with, it initialises the game conditions by asking the first player for the secret code length and generates the secret code. Every single communication (receiving and sending responses) are happened in ServerCoordinator class. There is also the GameConfig class where all the shared variables are stored. Through this class' variables, players are able to store their own game play details. Once the game has been started, it loops 10 times unless players guess the code correctly, or the players forfeit. At the end, the server sends player's game details to proper users and ranking to all connected users. After sending the messages, the server can be either terminated or play the game again.
- **MultiPlayer**
  - There are 2 cases being able to happen. If there is only one connection in the server, it is rather similar to single player game, but, in this case, the user needs to sign up (register) into the server with his first name, and during the game, the player is able to forfeit. In the other case (multiple users are connected), the first user is asked to set the secret code length, and the other users are asked to press any key to ready for the game. During the actual game flow, players have 10 guess chances normally unless they guess the code correctly or forfeit. Every player input has validation check from the methods included in the class. At the end of the game flow, players are announced about their score, secret code, and every player's score and ranking. Lastly, they are asked to press "q" key to quit the game.

Multi Player Game

3. Class diagram


MultiServer
# PORT: int = 18745
# LOBBY: int = 1
# commLogWriter: PrintWriter = null
# commLogFile: FileWriter = null
+ main(String[] args): void
+ writeCommLog(String msg): synchronized void
+ getTimeLog(): String
1..1
Opens
1..1
GameLobby
# serverSocket: ServerSocket = null
# firstPlayer: Socket = null
# secondPlayer: Socket = null
# lastPlayer: Socket = null
# player1Index: int = 1
# player2Index: int = 2
# player3Index: int = 3
# playerNum: int = 0
# maxLobbySize: int = 3
# ms: MultiServer = new MultiServer()
# sc: ServerCoordinator = new ServerCoordinator()
# commLogWriter: PrintWriter = null
+ GameLobby(ServerSocket serverSocket, PrintWriter commLogWriter)
+ run(): void
+ initPlayers(Socket playerSocket): void
+ runGameThread(): void
- closeStreams(Socket firstPlayer, Socket secondPlayer, Socket, lastPlayer, ServerSocket serverSocket): void
+ resetConditions(): void
1..1
Accepts
1..3
GameConfig
- playerScore: int = 100
- playerTurn: int = 1
# maxGuess: int = 10
- isPlayerDone: int = 0
- isEnd: boolean = false
# lock: Object = new Object()
+ getIsPlayerDone(): int
+ setIsPlayerDone(): void
+ getPlayerScore(): int
+ setPlayerScore(int playerScore): void
+ getPlayerTurn(): int
+ setPlayerTurn(int playerTurn): void
+ getIsEnd(): boolean
+ setIsEnd(boolean isEnd): void
1..1
Updates
1..1
1..1
Refreshes
1..1
MultiPlayer
- HOST: String = "m1-c27n1.csit.rmit.edu.au"
- PORT: int = 18745
# codeLength: String = ""
+ playerTurn: int = 1
+ maxGuess: int = 10
+ main(String[] args): void
- initGame(BufferedReader playerInputReader, BufferedReader serverReader, PrintWriter playerWriter): void
- playGame(BufferedReader playerInputReader, BufferedReader serverReader, PrintWriter playerWriter): void
- outOfGame(BufferedReader playerInputReader, BufferedReader serverReader): void
- getValidCodeLength(String input, BufferedReader inReader): String
+ isNumeric(String str): boolean
- closeStreams(BufferedReader playerInputReader, BufferedReader serverReader, PrintWriter playerWriter): void
1..1
Converses with
1..1
ServerCoordinator
# playerInputReader: BufferedReader = null
# serverWriter: PrintWriter = null
# gameLogWriter: PrintWriter = null
# gameLogFile: FileWriter = null
# playerSocket: Socket = null
# gc: GameConfig = new GameConfig()
# playerAuth: List<String> = new ArrayList<String>()
# playerRankings: Map<String, Integer> = new HashMap<String, Integer>()
# isCoorect: boolean = false
# checkNameAuth: String = null
+ ServerCoordinator(Socket playerSocket)
+ ServerCoordinator()
+ writeGameLog(String msg): synchronized void
+ sendMsgToPlayer(String msg): void
+ receivePlayerInput(): String
+ checkPlayerAuthentication(): void
+ generateSecretCode(String length): StringBuilder
+ compareSecretCode(String playerResp): void
+ returnMsg(): void
+ sendGameRankings(): void
+ closeGameStreams(BufferedReader playerInputReader, PrintWriter serverWriter, PrintWriter gameLogWriter, FileWriter gameLogFile): void
1..1
Communicates with
1..1
GuessGameThread
# playerSocket: Socket = null
# playerIndex: int = 0
# secretCode: StringBuilder = new StringBuilder()
# gc: GameConfig = new GameConfig()
# codeLatch: CountDownLatch = new CountDownLatch(1)
+ GuessGameThread(Socket playerSocket, int playerIndex)
+ run(): void
- startGuessGame(ServerCoordinator sc): void

Single Player Game


SingleServer
+ PORT: int = 18745
+ BACKLOG: int = 1
+ SecretCode: StringBuilder = new StringBuilder()
+ playerTurn: int = 1
+ maxGuess: int = 10
+ playerScore: int = 100
+ isCorrect: boolean = false
+ gameLogWriter: PrintWriter = null
+ gameLogFile: FileWriter = null
+ commLogWriter: PrintWriter = null
+ commLogFile: FileWriter = null
+ main(String[] args): void
+ initGuessGame(BufferedReader playerInputReader, PrintWriter serverWriter, String playerResp): void
+ startGuessGame(BufferedReader playerInputReader, PrintWriter serverWriter, String playerResp): void
- compareSecretCode(PrintWriter serverWriter, String playerResp): void
+ generateSecretCode(String length): StringBuilder
+ returnEndMsg(PrintWriter serverWriter): void
+ getTimeLog(): String
+ closeStreams(BufferedReader playerInputReader, PrintWriter serverWriter, Socket playerSocket, ServerSocket serverSocket, PrintWriter gameLogWriter, FileWriter gameLogFile, PrintWriter commLogWriter, FileWriter commLogFile): void
1..1
Communicates with
1..1
SinglePlayer
+ HOST: String = "m1-c27n1.csit.rmit.edu.au"
+ PORT: int = 18745
+ codeLength: String = ""
+ main(String[] args): void
+ initAndPlayGuessGame(Socket playerSocket): void
+ startGuessGame(BufferedReader playerInputReader, PrintWriter playerSender, BufferedReader serverReader, String playerInput, String serverResp): void
- getValidCodeLength(String input, BufferedReader inReader): String
+ isNumeric(String str): boolean