# Documentation

1. First requirement - Player with single instance to achieve this task thread functionality is used to get the desired output.
2. Second requirement - Player with 2 process id’s will be able to communicate with inter-process communication i.e. TCP socket is used.
3. Both the requirements are fulfilled in one single project and a shell script is used to run the programs.
4. To differentiate both the working argument is used while running the shell - **./start.sh socket** is used to work in IPCS mode if no argument is passed the default mode is single threaded version where it will use only one single instance for communication.
5. The definition and working of each class are given below:

File – Main.java

Definition - The Main class serves as the entry point for the application, enabling users to specify the operational mode (server, client, or normal) via command-line arguments.

Working:

* When the program is launched, it checks if at least one command-line argument is provided.
* no arguments are provided, it displays a usage message indicating how to run the program and exits.
* If an argument is provided, it determines the mode based on the argument value (server, client, or normal).
* Depending on the mode, it invokes the appropriate main method of the corresponding class (PlayerServer, PlayerClient, or PlayerMain), passing an empty string array as arguments.
* If an unknown mode is provided, it displays an error message and exits.

File – Message.java

Definition - The Message class represents a communication unit, containing sender identification and message content.

Working:

* The Message class contains information about a message's sender and content.
* When creating a message, you specify the sender's ID and the message content.
* You can retrieve the sender's ID and content from a message using specific methods.
* Additionally, the class allows messages to be serialized for transmission over networks or saving to files.

File – MessageReceiver.java

Definition - The MessageReceiver class coordinates the reception and processing of messages for a specific player within the application.

Working:

* The MessageReceiver class encapsulates the functionality to handle message reception for a designated player.
* Upon execution, it triggers the player's processMessages() method, enabling the player to process incoming messages seamlessly in a concurrent manner.

File – Player.java

Definition - The Player class models a participant in a messaging system, facilitating message handling, transmission, and response within the application.

Working:

* The Player class represents a participant in a messaging system.
* It manages message reception, processing, and transmission between players.
* Upon receiving a message, it processes it by appending a counter and sends a response to the sender.
* The class utilizes a message queue for buffering incoming messages and ensures concurrent processing via its processMessages() method.

File – PlayerMain.java

Definition - PlayerMain coordinates message exchange between players in a multi-threaded environment.

Working:

* Initializes player instances and manages message transmission.
* Utilizes multi-threading for concurrent message handling.
* Facilitates message reception and processing via dedicated threads.
* Gracefully terminates communication upon completion.

File – PlayerClient.java

Definition -The PlayerClient serves as a client-side component facilitating communication with a server via message exchange.

Working:

* Establishes a connection with the server.
* Sends an initial message to initiate the conversation.
* Receives and processes messages from the server iteratively.
* Constructs and sends subsequent messages based on received responses, facilitating bidirectional communication.

File – PlayerServer.java

Definition – The PlayerServer acts as a server-side component, handling communication with clients through message exchange.

Working:

* Starts a server socket and waits for incoming connections from clients.
* Accepts client connections and initiates communication.
* Receives messages from clients, processes them, and constructs responses.
* Sends response messages back to clients, facilitating bidirectional communication.