**COMMUNICATION PROTOCOL**

Chapter 1 Server Architecture

Chapter 2 Client’s Backend Architecture

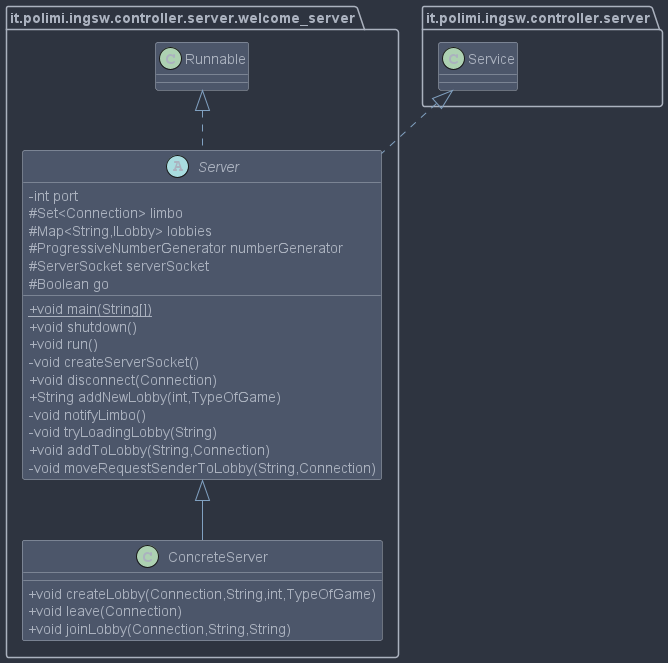
Chapter 3 Communication protocol’s messages

**Chapter 1 Server Architecture**

The first class to be constructed when the server application is launched is the Server class.

The server class has only one goal: to welcome the new login requests on the server.

**How the server works**



The Server class sole goal is to secure the connection with the client.

The server class constructor accepts any port number but has 9001 as its default. Once instantiated it can be started by calling the “run” method. The main method in the server jar automatically starts the server.

The server is started by writing on console:

*java -jar server.jar [port]*

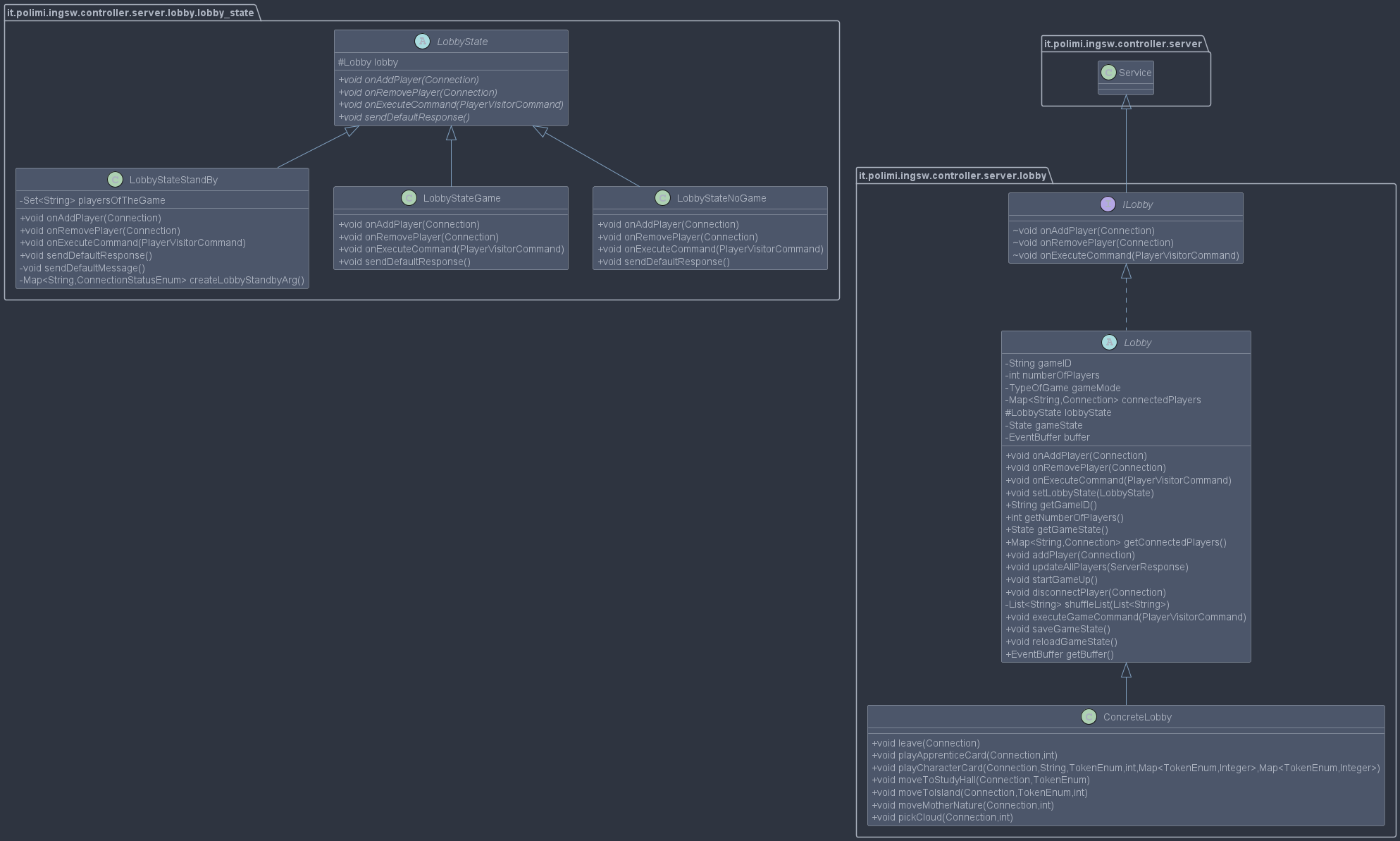
Once the server starts its ServerSocket waits for the first connection request from a client.

When a connection is accepted, its Socket is stored inside a “Connection” class, whose routine runs on a separated thread (more on that later).

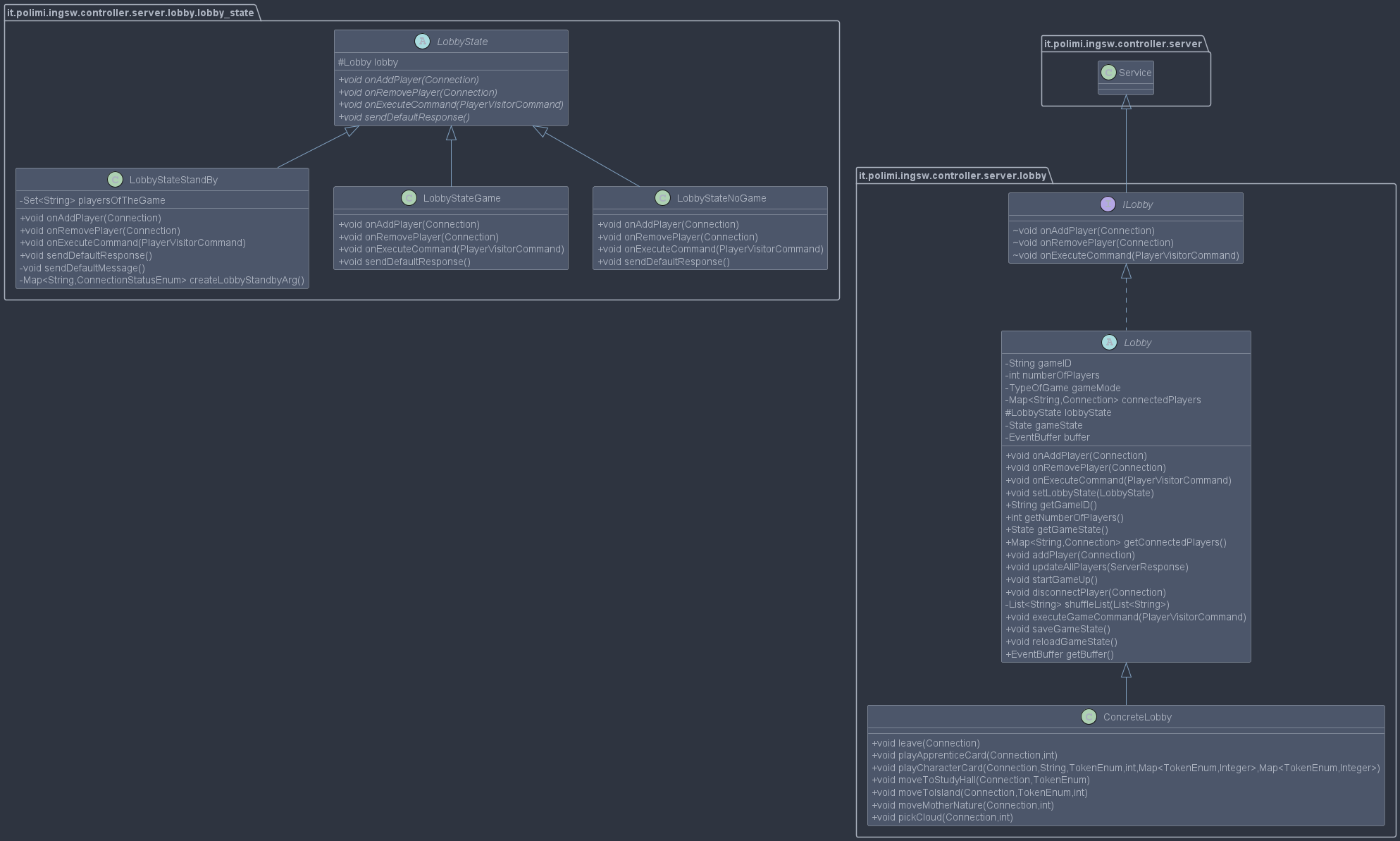
Lastly, the Connection is stored on a Set called limbo, which hosts all the Connection of clients which haven’t joined any lobby yet.

Any client inside the limbo can either join a lobby or create a new one (a Set of available and loadable lobbies is sent when the client joins and is refreshed every time a new lobby is created or loaded from the disk).

**How the lobbies work**

****A lobby hosts players Connections and has access to its game instance, it behaves differently based on whether there’re missing player or not.

A lobby can have three states: no game, stand by and game.

****

A lobby hosts players Connections and has access to its game instance, it behaves differently based on whether there’re missing player or not.

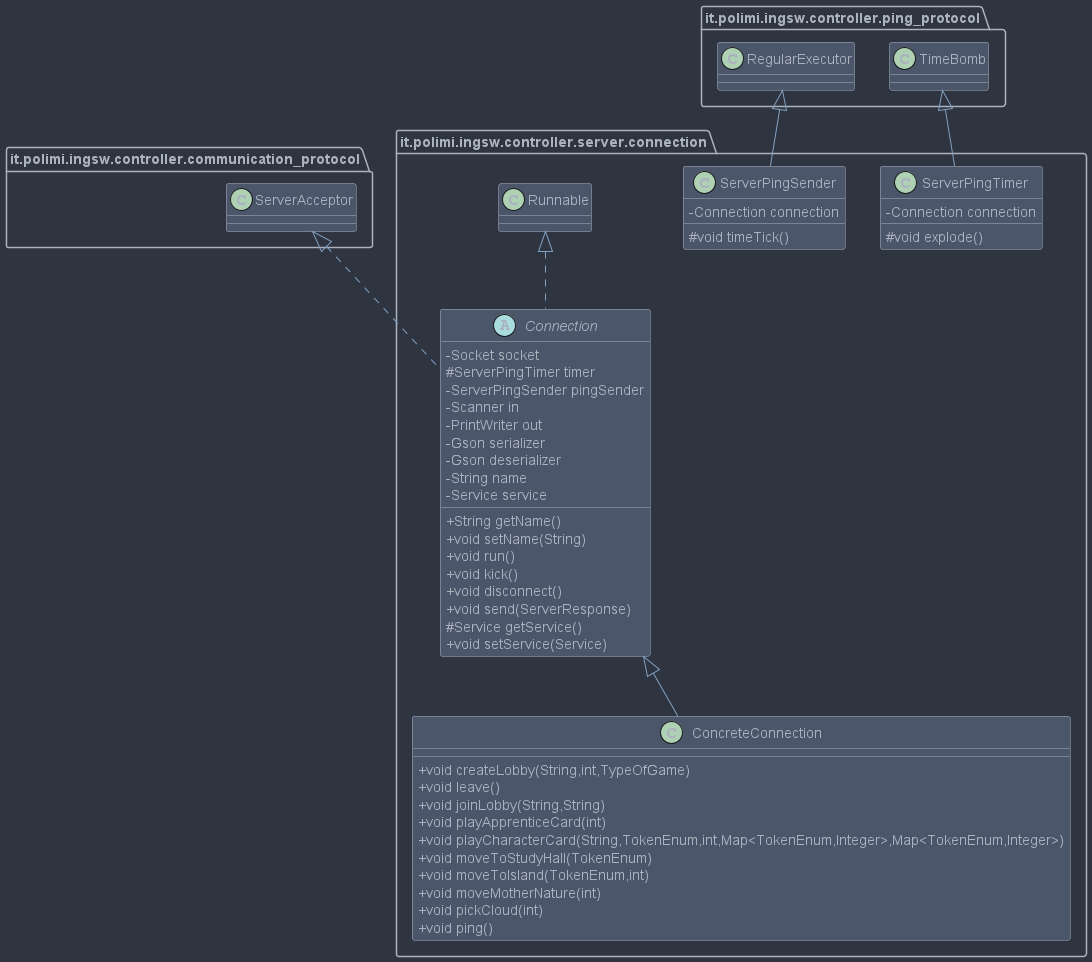
When a new lobby is created it starts in the no game state, in this lobby any player may join or leave with little consequence; once the number of desired players (declared on the lobby creation request) is reached it enters the game state.

In the game state players can either play the game by issuing commands or leave, but other clients cannot join it, for the lobby is full. If at any point a player quits or its connection is lost (the ping mechanism reaches its time limit), the lobby enters the stand-by state.

The stand-by state works in a similar manner to the no-game state, but only the previously connected players are allowed to join; once all the previous players are connected again, the games start again at the point in time in which the lobby went stand-by. The stand-by state is the default state of lobbies of loaded games.

**Connections**

A Connection class is created every time the Server Socket accepts a connection request.



Once the connection starts three threads are created: the listening thread, the ping timer, and the ping sender.

The listening thread waits for a line from the client’ socket input stream (The thread sleeps, there is no “busy-wait”); the line is then converted into a client request via a Gson deserializer; lastly, the request is sent to the service to whom the Connection is linked to (service is just an interface which is implemented by Server and Lobby): the Connection seldom elaborates the content of the client requests, notable exceptions are the leave request and the ping.

The ping timer is, as one could expect, a counter that must be refreshed by ping messages from the client; if, for any reason, the client fails to send a ping in the requested time, the Connection assumes the client went offline and closes its connection.

The ping sender, instead, regularly send a ping message to the client to communicate its alive (our client assumes that the connection has been lost if it does not receive any message from the server in a certain time frame).

The Connection sends server responses to the client by serializing them using a Gson serializer.

When the disconnection procedure starts all the threads are stopped.

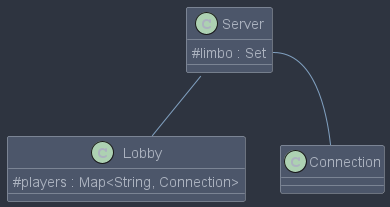
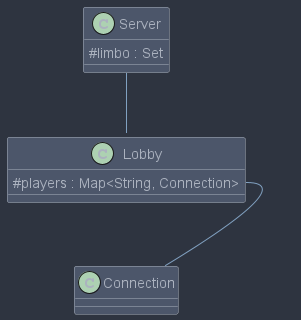
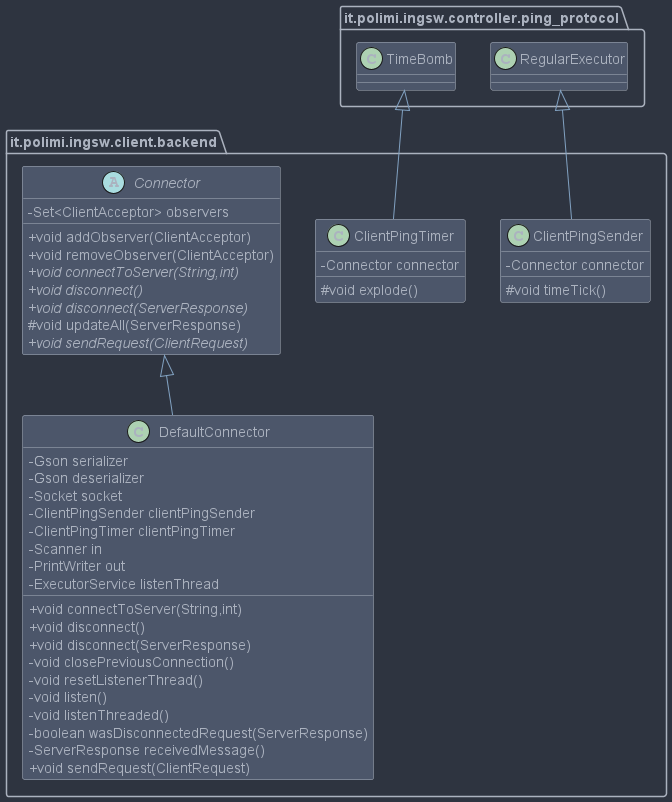
 

Figure 1: client inside limbo

Figure 2:client inside lobby

**Chapter 2 Client Architecture**

The client can communicate with the server by using a Connector class.

An implementation of this class is the Default Connector.

The Default connector has three main operations:

1. connect to server
2. disconnect
3. send request

Just like the Connection class its routines run on three threads, which have similar purposes:

1. listen
2. ping timer
3. ping sender

Despite sharing the same purposes their code is slightly different in the concrete classes, since they interact with different entities, and they send/receive different messages (client requests and server responses do not implement the same interface nor should they).

Just like the Connection, if the connection is lost (i.e., the ping timer countdown reaches zero) no disconnect request is issued since the connection is assumed to have been lost.

The client can connect to the server after the client app has been launched by using the connector method “connectToServer(ip, port)” or can be done at launch by writing ip and port as args (in the command line interface at least).

To activate the CLI type:

java -jar CLI.jar [ip port]

Type either both the ip and the port or neither.

**Chapter 3 Communication protocol’s messages**

The communication between the Server and its client is granted by the exchange of two kinds of messages: Client Requests and Server Responses.

Both the connector in the client and the Connection class on the server can only read messages of one kind and only send messages of the other.

As one could expect, if one of the two classes sends a kind of message, that kind is the one received by the other class, thus:

1. The Connection class listens for Client Requests and sends Server Responses
2. The Connector class listens for Server Responses and sends Client Requests

The separation of these two kinds of messages is guaranteed by the fact that they do implement different interfaces (Off topic: in both cases it is important that the class itself is correctly passed throughout the socket because both interfaces are visitors in a visitor pattern, albeit they are related to different acceptors).

**Client Requests**

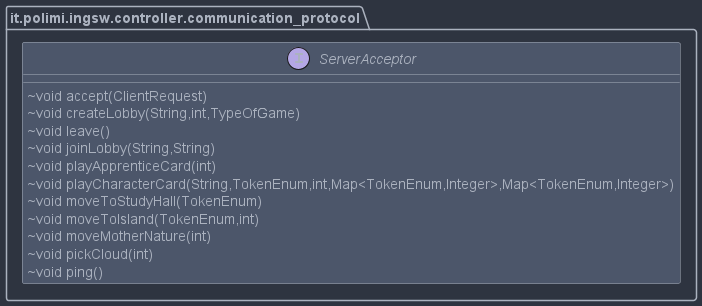
1. Create Lobby (hostname, number of players, game mode): creates a lobby, the client’s name becomes the hostname for the purposes of the game; the other two numbers indicate the desired number of players and the game mode (either normal or expert). While already inside a lobby, requests to create a lobby are ignored.
2. Join Lobby (player name, game id): the client tries to join the lobby “game id”, their game name becomes the given name for the purposes of the game. Just like Create Lobby, while already inside a game lobby, join requests are ignored. The request may be rejected by the server in two cases:
   1. No lobby corresponds to the given game id
   2. The player’s name in such lobby has already been taken
3. Leave: tells the server the client is leaving
4. Ping: resets the connection ping timer
5. Game Request: there is no generic game request but several specific ones (bear in mind that game requests are computed only if the client is engaged in a game match, otherwise they are simply ignored)
   1. Play Apprentice Card (card id): to play the desired apprentice card
   2. Move To Study Hall (token): adds the desired token to the player study hall from the entrance hall
   3. Move to Island (token, island index): moves the desired token from the player entrance hall to the designated island
   4. Move Mother Nature (movement): moves mother nature by the desired number of tiles
   5. Pick Cloud (cloud index): adds the tokens on the designated cloud to the player’s entrance hall
   6. Play Character Card (card name, token, island index, tokens to remove, tokens to add): plays the desired card while giving all the necessary arguments (bear in mind that some cards will refuse the command if needless arguments are passed); to avoid any complications, both the CLI and the GUI have builder methods for creating the correct requests.

All game commands are just requests to modify the model and thus they might be refused and throw back, exclusively to the sender, an error message (in the form of a game event).

**Server Responses**

1. Server Welcome: contains a set of available lobbies as well as a set of loadable lobbies
2. Lobbies Status: similar to Server Welcome, but it is passed to every client in the limbo whenever a new lobby is added
3. Error: contains an error message
4. Disconnected: informs that the player has been disconnected
5. Lobby Joined: sent when a player connects to or disconnects from a lobby in the no game state, contains the name of the players currently in the lobby
6. Lobby Standby: sent to all players in a stand-by lobby, contains information of all the players connection statuses
7. Server ping: ping message regularly sent to the clients to keep alive the connection
8. Game Status: sent when the game starts and every time a command is successfully executed; it contains a simplified version of the game model -in the form of a DTO- and an ordered list of game events useful for explaining the evolution of the model. The game DTO is passed every turn for ease of use for the client (especially the CLI) but also for giving essential info at the start of the game as well as for easily reloading the model at any point in time.

As mentioned above, both the Client Requests and the Server Responses are visitors, and their acceptors are “Server Acceptor” and “Client Acceptor” respectively.

 Immagine che contiene testo

Descrizione generata automaticamente

*Server Acceptor Interface Client Acceptor Interface*