MyShelfie

Prova Finale Ingegneria del Software

Team RJ45

Realizzato da:

Tirabassi Maurizio

Vianello Gabriele

Zappia Vincenzo

Laurea in Ingegneria Informatica

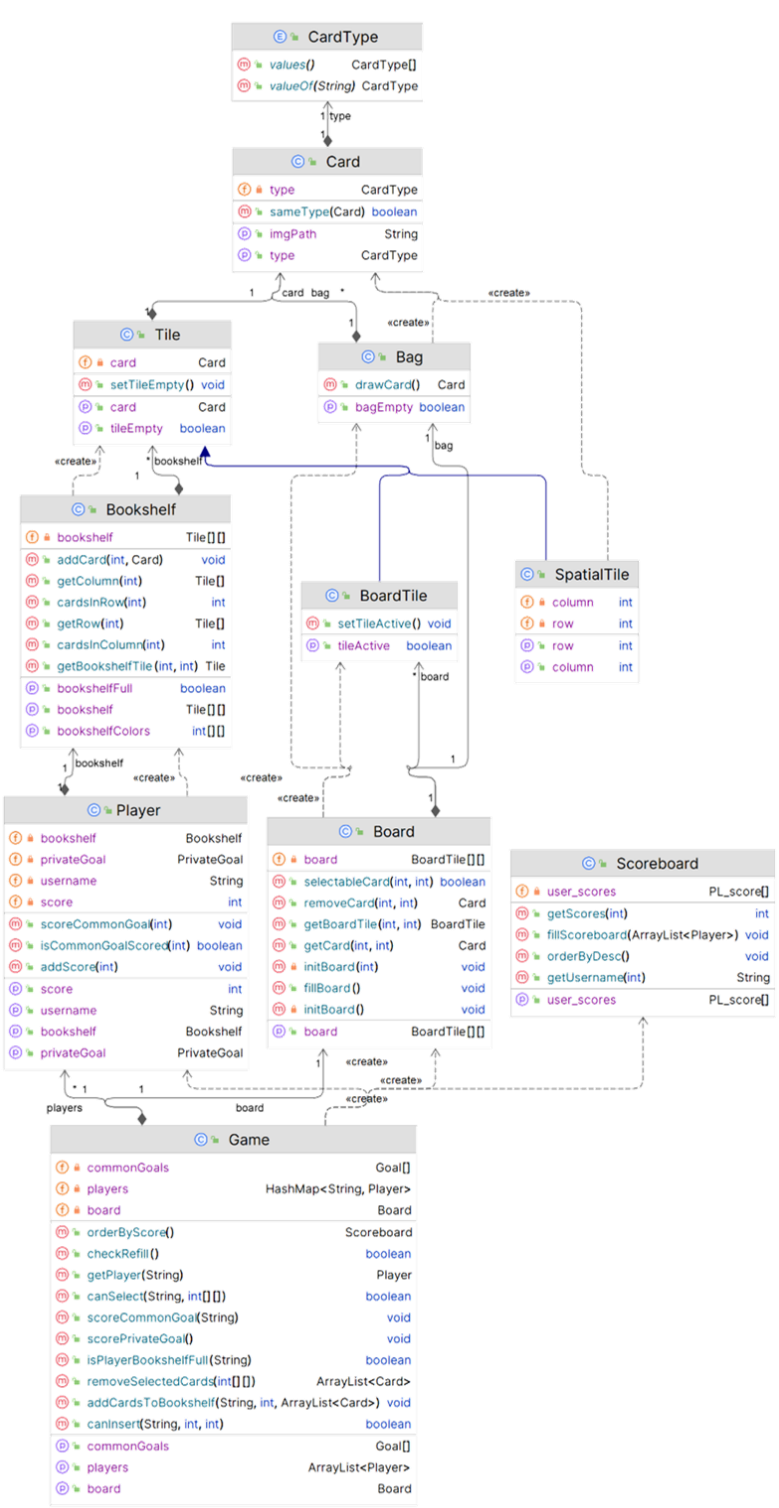
POLITECNICO DI MILANO

Indice

Introduzione

Nei prossimi paragrafi sono descritte le principali caratteristiche, le scelte progettuali ed implementative della realizzazione del gioco da tavolo *MyShelfie*.

Gioco



Caratteristiche principali della scelta delle entità di gioco:

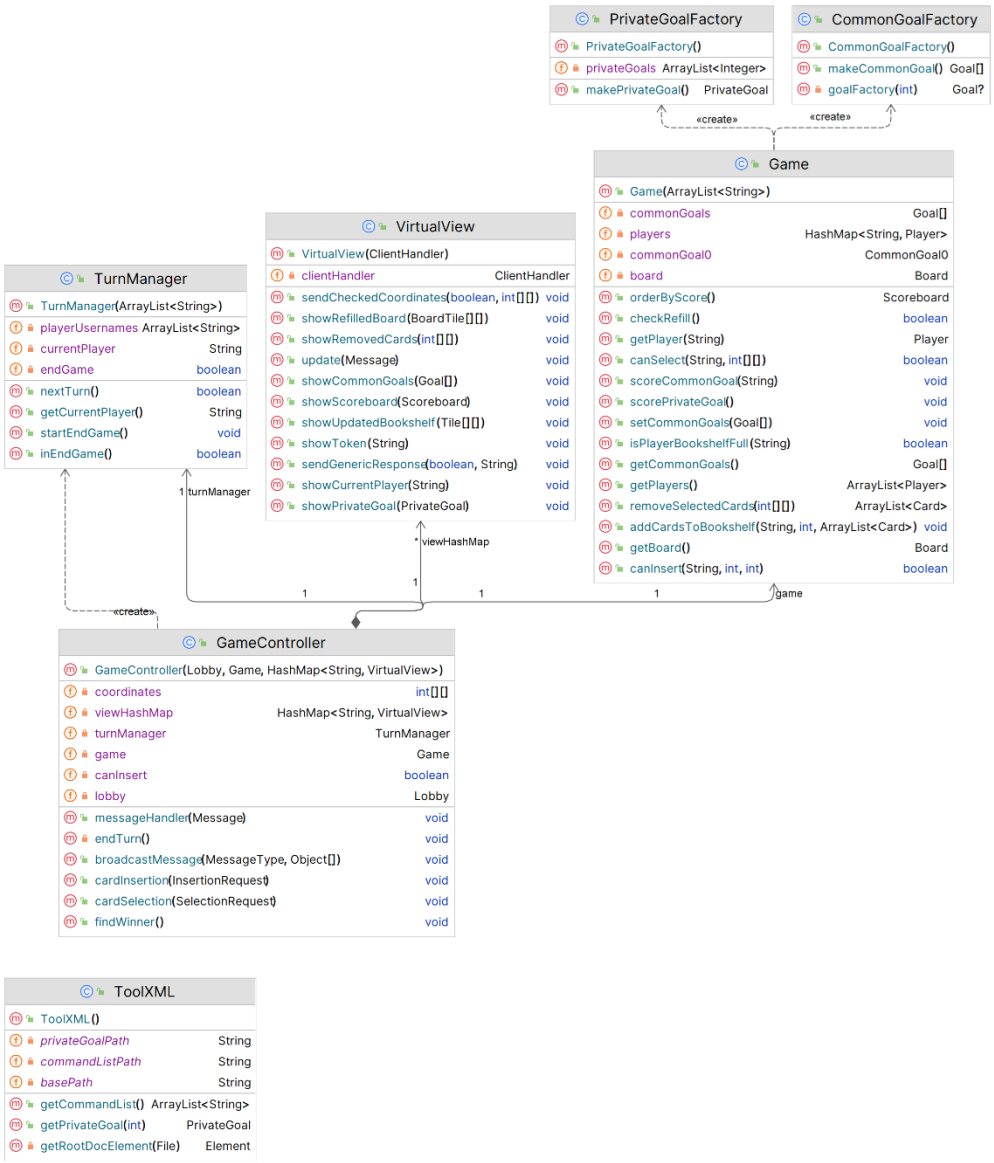
- La **Board** viene realizzata attraverso una matrice 9x9 di **BoardTile**, classe con flag *active* che identifica un'effettiva casella di gioco, così è stata ricavata la forma particolare del tabellone di gioco.
- La classe **Bookshelf** è istanziata all'interno della classe **Player**, di conseguenza **Game** non ne ha accesso diretto.
- Classe **Bag** crea le istanze di tutte le possibili 132 carte di gioco, 22 per ognuno dei 6 tipi, identificate da due attributi:

1. Enumerazione del tipo di carta
2. Nome di una delle tre possibili immagini per tipo di carta. 22/3 = 7 con il resto di 1: la scelta del gruppo di 8 - anziché 7 - carte uguali avviene nel ciclo:

```
for (int i = 0; i < 6; i++)
    for(int j = 0; j < 22; j++)
        bag.add(new
Card(cardImgName[i][j % 3],
CardType.values()[i]));
```

- Per ogni common goal viene creata una classe che ne implementa lo specifico algoritmo. All'inizio del gioco vengono scelti casualmente due common goal tra i dodici possibili attraverso **CommonGoalFactory**.

Meccaniche di gioco

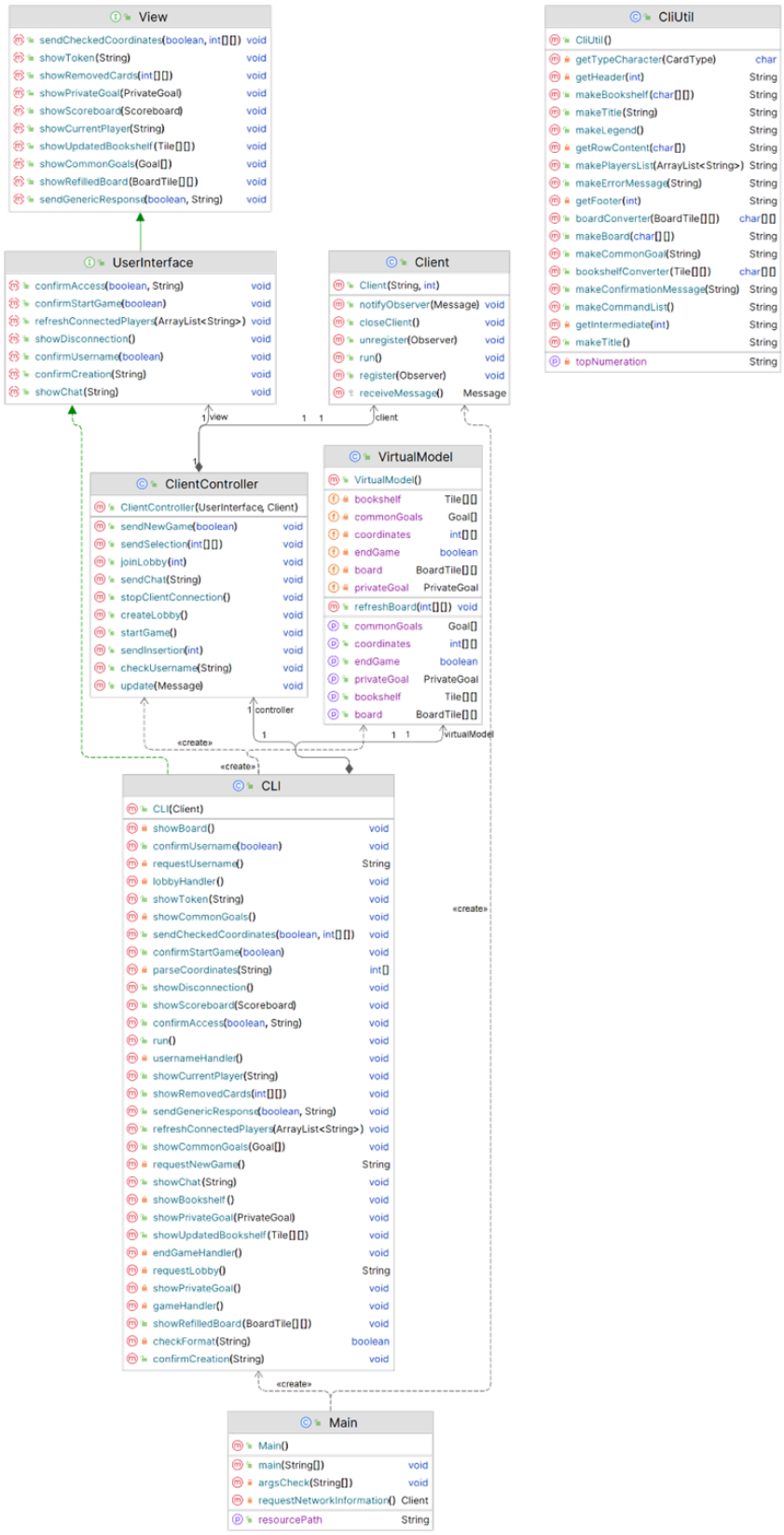


La classe **Game** implementa i metodi che determinano tutte le funzionalità di gioco, ovvero le possibili interazioni tra le entità dello stesso come, ad esempio, determinare la validità di una selezione di carte, rimuoverla dal tabellone ed inserirla all'interno della libreria di un giocatore.

È la classe **GameController** che si occupa, però, di gestire la validità logica dell'ordine in cui vengono chiamate tali funzionalità relativamente ai turni di gioco (un giocatore che non è di turno non potrà mai effettuare alcun tipo di azione) e alle fasi di uno stesso turno (il giocatore di turno non potrà mai effettuare un'inserzione prima di una selezione oppure prima che essa venga validata).

La classe **TurnManager** implementa la logica con cui vengono gestiti i turni di gioco attraverso una lista degli username dei giocatori. La poltrona si trova all'indice 0 della lista mentre il giocatore alla sua destra corrisponde all'ultimo della stessa. Con il metodo *nextTurn* si itera la lista in maniera circolare finché un giocatore non riempie la propria bookshelf, dopodiché lo scorrimento temirnerà in corrispondenza del giocatore alla destra della poltrona.

Interfaccia utente: CLI



La **CLI** viene chiamata su un thread secondario al **Main**. Gestisce il flusso di gioco attraverso quattro handler:

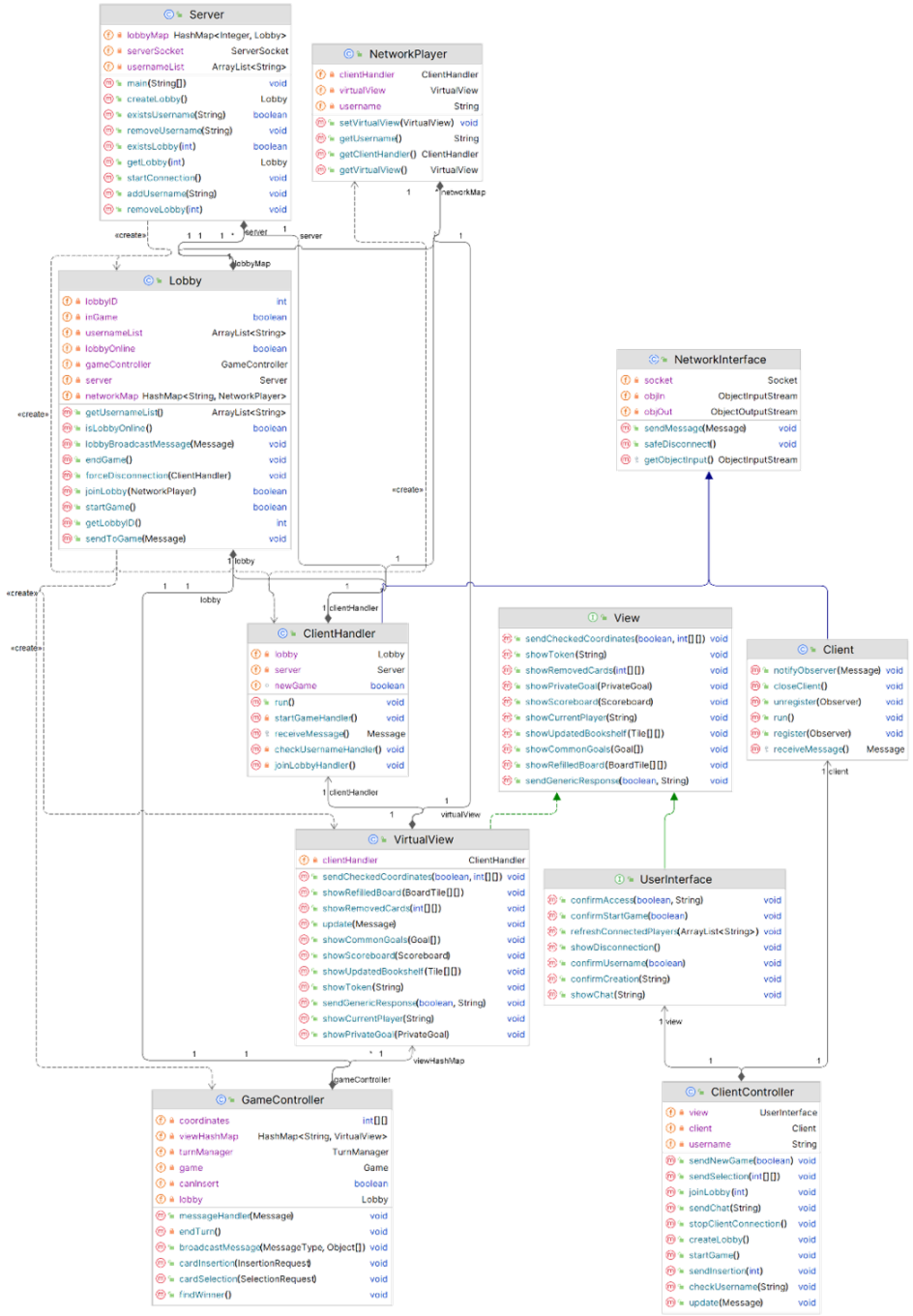
1. *usernameHandler* gestisce la scelta di uno username unico.
2. *lobbyHandler* gestisce la creazione di una nuova lobby oppure l'ingresso in una già esistente e attende l'inizio del gioco (comandato dal giocatore che crea la lobby).
3. *gameHandler* gestisce l'invio dei comandi di gioco e la visualizzazione delle entità dello stesso.
4. *endGameHandler* gestisce la possibilità di giocare di nuovo oppure chiudere l'applicazione in seguito al termine di una partita oppure alla disconnessione di un giocatore.

Tutti gli handler gestiscono il proprio flusso attendendo le risposte dal server attraverso l'utilizzo di *Object lock* bloccati dalla **CLI** e sbloccati quando il server risponde.

La **CLI** utilizza la classe **VirtualModel** all'interno della quale si tiene traccia dell'istanza aggiornata dell'entità di gioco.

Comunicazione

Il protocollo di comunicazione è basato su un sistema di classi messaggio che estendono **Message**, classe che implementa **Serializable**. In questo modo, gli oggetti generati contenenti le entità di gioco possono essere trasmessi tra server e client utilizzando *ObjectInputStream* e *ObjectOutputStream*: questi sono utilizzati nei metodi *sendMessage* e *receiveMessage* della classe astratta **NetworkInterface** estesa da **Client** e **ClientHandler**.



La classe Server si occupa di:

- Tenere traccia degli username dei giocatori attivi in tutte le lobby.
- Tenere traccia delle lobby attive.
- Accettare nuove connessioni.

Per ogni nuova connessione accettata viene creato un **ClientHandler** al quale viene associata una **VirtualView**. Dell'associazione tra *username*, **VirtualView** e **ClientHandler** si tiene traccia nella classe **NetworkPlayer**.

I messaggi generati dal server sono suddivisi in due gruppi principali:

1. Messaggi generati da GameController in seguito a cambiamenti delle entità e delle logiche di gioco.
2. Messaggi generati da Lobby che riguardano la fase di connessione e la chat.

Le due interfacce View e UserInterface ne consentono la rispettiva gestione.

Client utilizza il design pattern **Observer** per notificare **ClientController** della ricezione di nuovi messaggi da parte del server.

Safe disconnection

Nel caso in cui un giocatore perda la connessione al server viene generata un'eccezione. La nostra gestione consiste nel forzare la disconnessione dei giocatori rimanenti nella lobby, sia prima che durante il gioco, portandoli ad una schermata di scelta: creare o partecipare ad una nuova lobby oppure uscire dall'applicazione.