Presentazione del progetto «Chat»

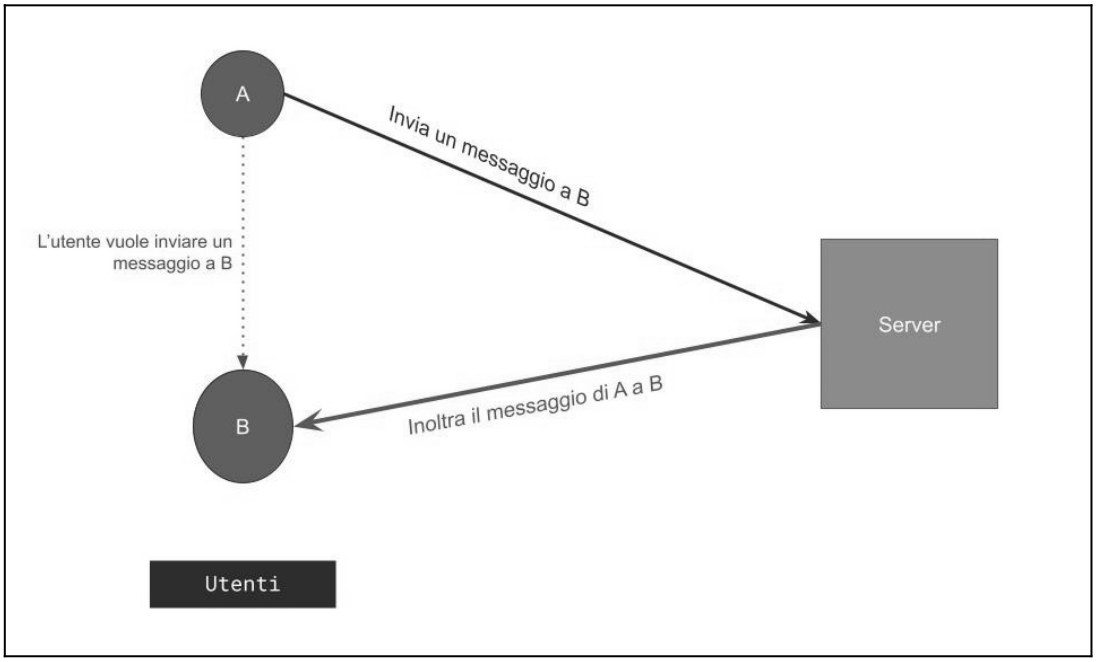
Francesco Parisio – 5BIA

Tabella dei contenuti

Introduzione

Cosa è una chat?

La *chat* è uno strumento oggidì utilizzato quotidianamente da tutti che permette la comunicazione **istantanea** tra più utenti attraverso lo scambio di messaggi testo, audio, e video. Il tipo di chat preso in esame in questo progetto è quello **centralizzato**, cioè quella composta da due parti: una parte **client**, usata dall'utente, che invia i messaggi al server; ed il **server**, appunto, che si occupa di smistare i messaggi degli utenti. Il server non è deputato all'utilizzo dell'utente dato che i messaggi sono **privati** e devono essere visti solo dai destinatari.



Per fornire un parallelismo con la realtà, consideriamo un ufficio postale. L'utente consegna all'ufficio postale la lettera, indicandone il destinatario: l'ufficio postale invia quindi la lettera, **assicurandosi che sia stata recapitata**.

Resta chiaro il fatto che server e client, sebbene interdipendenti, siano due entità **distinte**. Stabiliti degli *standard* di comunicazione, client e server devono essere capaci di comunicare anche se il loro sviluppo è realizzato da persone diverse. Ad esempio: l'invio del messaggio al corretto destinatario deve essere un compito affidato al server solamente: il client si deve occupare solo di indicarne il nome.

La chat nel progetto

La chat realizzata in questo progetto è stata pensata per scopi didattici, pertanto non include molte delle funzionalità che siamo abituati a vedere in applicazioni di messaggistica istantanea moderna. Ad esempio: non disporrà di metodi per inviare messaggi vocali, immagini o video, né di alcuna tutela

sulla sicurezza dei dati, né di creare chat di gruppo. I dettagli sui casi d'uso, e quindi sui requisiti, sono elencati nella sezione *Casi d'uso*.

Progettazione iniziale

Alcune informazioni sulle tecnologie utilizzate

Per la finalizzazione del progetto, sono state utilizzate alcune tecnologie rilevanti:

- **Socket** TCP – Un'astrazione che si basa sul protocollo di trasporto TCP della pila ISO–OSI, e rappresenta la connessione e il suo relativo canale dati tra due *end point*.
- **JavaFX –** Una *toolkit* GUI che consente di creare interfacce grafiche con Java.
- **Maven** – Un gestore di dipendenze, o *build tool*, per progetti Java.

Fuori dall'elenco, in quanto la sua posizione è opinabile, inserisco anche la programmazione concorrente, e la tecnologia alla base di questa: i **Thread**.

Queste sono implementate dalle librerie native di Java e dalle librerie ufficiali di JavaFX. Lo sviluppo sarà realizzato con il Java Development Kit alla versione 17, distribuito da OpenJDK.

Definizioni preliminari

Prima di procedere con la spiegazione del progetto, è opportuno definire alcuni termini usati nella relazione:

- **Client**: Il programma che invia i messaggi o comandi al server, ed agisce per conto dell'utente.
- **Server**: Il programma che smista i messaggi in arrivo dal client, o interpreta i comandi del client. È autonomo.
- **Messaggio**: È una stringa di testo, allegata con il nome utente del destinatario, pensato all'inoltro al server.
- **Comando**: È un messaggio che non contiene destinatari, ed è pensato per ordinare al server un'azione. E.g. un comando *di cambio nome*, che serve per chiedere al server di cambiare il nome del mittente, o un comando di *chiusura connessione*, che serve per chiudere il Socket tra il Client ed il Server.
- **Username** o **nickname:** Una stringa di testo univoca utilizzata per identificare un client. È associato ad un indirizzo IP, e di conseguenza, ad un Socket TCP.
- **Utente**: Utilizzatore del sistema Client.

Casi d’uso

I casi d’uso descrivono dettagliatamente cosa il sistema informatico deve essere in grado di fare. Siccome il progetto è composto da due parti, i casi d’uso sono stati divisi per client e server. Naturalmente, ad una azione del client, dovrà corrisponderne una del server, giacché sono *inter*dipendenti. Vengono mostrati di seguito i diagrammi dei casi d’uso con le relative tabelle descrittive.

Diagramma dei casi d’uso del Client

Caso d’uso	Si connette al server
Attori	Client
Precondizioni	
Scenario Principale	Il client si connette al server tramite Socket TCP ad un server.
Scenario Alternativo	Il client non riesce a connettersi.
Postcondizioni	Il client possiede una Socket il cui canale di

	trasferimento dati consente lo scambio di informazioni.

Caso d’uso	Inserisce il suo username
Attori	Client, Server
Precondizioni	Il client deve essere connesso al server.
Scenario Principale	Il client invia un comando al server per impostare un nome a scelta dell’utente.
Scenario Alternativo	Il client non riesce ad inserire un nuovo nome perché il nome scelto è gia in possesso di un altro utente.
Postcondizioni	Il client possiede un username, lo ha comunicato al Server: gli ha dato conferma positiva: Può comunicare ed inviare i messaggi.

Caso d’uso	Invia un messaggio
Attori	Client, Server
Precondizioni	Il client ha stabilito una connessione con il server e possiede un username validato da questi.
Scenario Principale	Il client invia un messaggio al server indicandone il destinatario, il tipo, e il contenuto.
Scenario Alternativo	Il client non riesce ad inviare un messaggio perché il destinatario indicato non è presente nella lista degli utenti connessi del server. Il client non riesce ad inviare il messaggio perché non è stato validato correttamente: il suo formato non è riconosciuto dal programma, l’utente ha commesso un errore.
Postcondizioni	Il client ha inviato un messaggio ad il destinatario inoltrandolo al server, e non ha ricevuto messaggi di errore. L’utente visualizza il suo messaggio nello storico dei messaggi ricevuti ed inviati.

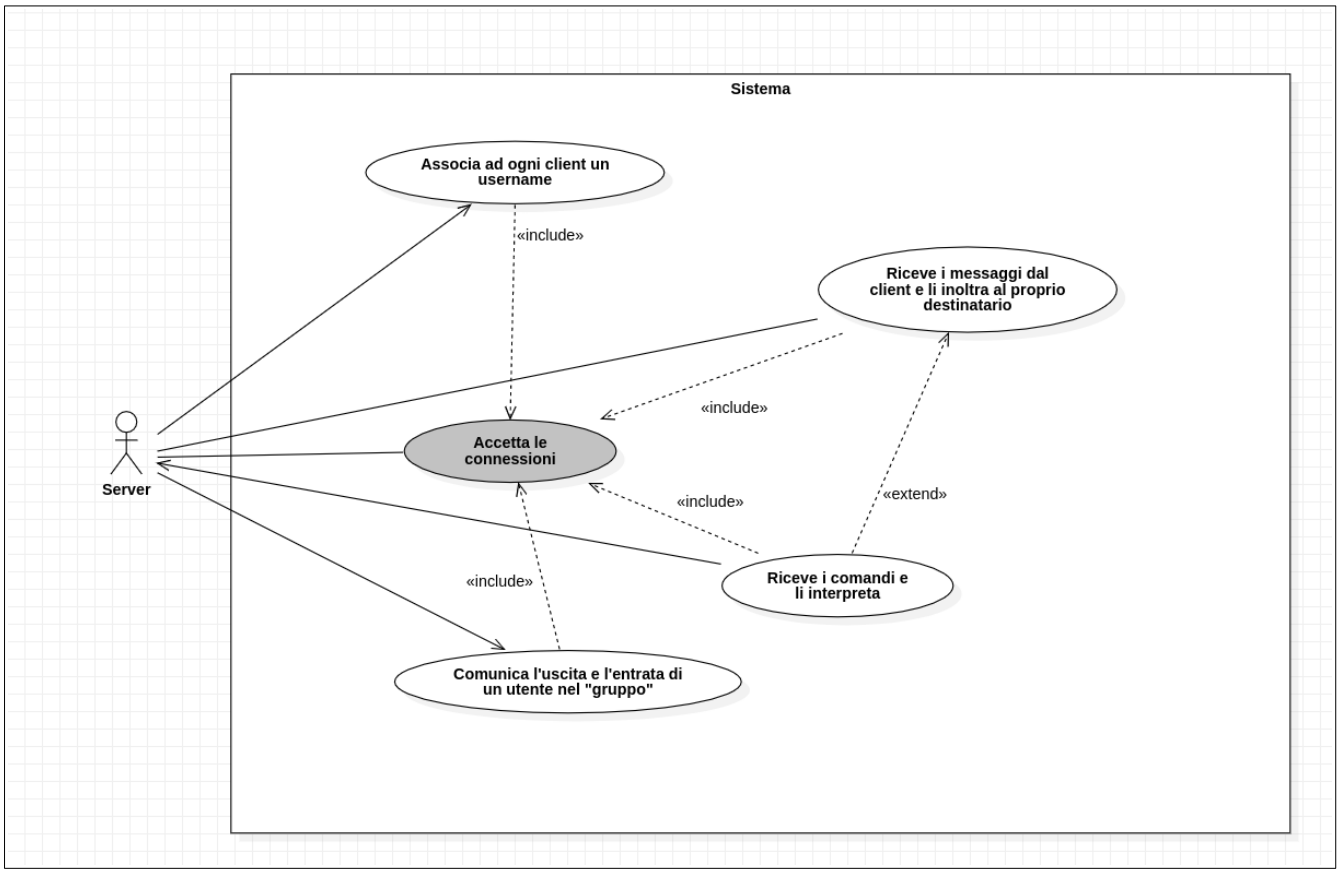
Caso d’uso	Invia un messaggio di broadcast

Attori	Client, Server
Precondizioni	Il client ha stabilito una connessione con il server e possiede un username validato da questi.
Scenario Principale	Il client invia un messaggio al server indicandone il destinatario di broadcast, il tipo, e il contenuto.
Scenario Alternativo	Il client non riesce ad inviare il messaggio perché non è stato validato correttamente: il suo formato non è riconosciuto dal programma, l'utente ha commesso un errore.
Postcondizioni	Il client ha inviato un messaggio ai client inoltrandolo al server.

Caso d'uso	Invia un comando
Attori	Client, Server
Precondizioni	Il client ha stabilito una connessione con il server e possiede un username validato da questi.
Scenario Principale	Il client invia un messaggio di tipo comando, indicando nel contenuto gli eventuali argomenti, destinandolo al server.
Scenario Alternativo	Il client riceve un messaggio di errore dal server perché il comando inviato non è stato riconosciuto. Il client non riesce ad inviare il comando perché non è stato validato correttamente: il suo formato non è riconosciuto dal programma, l'utente ha commesso un errore.
Postcondizioni	Le modifiche richieste dal comando sono soddisfatte.

Caso d'uso	Visualizza i comandi disponibili
Attori	Client
Precondizioni	
Scenario Principale	Visualizza i comandi disponibili all'utente.
Scenario Alternativo	
Postcondizioni	L'utente ha conoscenza dei comandi a lui disponibili.

Diagramma dei casi d’uso del Server



Caso d’uso	Accetta le connessioni
Attori	Server
Precondizioni	
Scenario Principale	Ogni volta che un client prova a connettersi, il server crea un Thread per gestirlo.
Scenario Alternativo	
Postcondizioni	Il server possiede un Thread che gestisce la nuova connessione. Questo è memorizzato all’interno di una lista, assieme all’indirizzo IP del client e l’username assegnatogli.

Caso d’uso	Associa ad ogni utente un nickname
Attori	Server
Precondizioni	L’utente in questione è connesso al server ed è memorizzato in una lista. Il server ha ricevuto dal

	client un comando di *cambio nome* con allegato il nuovo nome che l'utente desidera.
Scenario Principale	Il server controlla se il nome utente è disponibile e lo assegna al client.
Scenario Alternativo	Il nome utente non è disponibile, e invia al client un messaggio di errore.
Postcondizioni	Il nuovo nome scelto dall'utente è alla fine assegnato.

Caso d'uso	Riceve i messaggi dal client e li invia al proprio destinatario
Attori	Server
Precondizioni	Il server riceve un messaggio di tipo messaggio con un destinatario valido, presente nella lista dei client connessi.
Scenario Principale	Il server inoltra al destinatario il messaggio.
Scenario Alternativo	Il server non riconosce l'username, e invia un messaggio di errore.
Postcondizioni	Il messaggio è stato spedito al destinatario.

Caso d'uso	Riceve i comandi dal client e li interpreta
Attori	Server
Precondizioni	Il server riceve un messaggio di tipo comando che riconosce.
Scenario Principale	Dipendentemente da ciò che è disposto dal comando, il server gestisce la richiesta.
Scenario Alternativo	Il server non riconosce il comando e invia un messaggio di errore.
Postcondizioni	Ciò richiesto dal comando viene eseguito correttamente; le modifiche sono apportate con successo.

Caso d'uso	Comunica l'entrata e l'uscita di un utente da il "gruppo"
Attori	Server

Precondizioni	Il server riconosce un cambiamento nella lista degli utenti, oppure, il server ha ricevuto un comando di disconnessione da parte di un client.
Scenario Principale	Comunica a tutti gli utenti dell'avvenuta disconnesione o connessione di un utente al gruppo di utenti connessi.
Scenario Alternativo	
Postcondizioni	Viene inviato un messaggio consono al tipo di interazione (uscita o entrata) a tutti gli utenti della chat.

Diagramma delle classi

Vengono elencate di seguito le classi che fanno parte del progetto. È allegato lo schema UML che comprende anche gli attributi e i metodi da implementare. A queste classi non sono allegate quelle dedicate alla progettazione GUI.

Le classi in comune a tutti e due i progetti:

- **Message:** I messaggi che vengono scambiati tra client e server. Hanno un tipo, un destinatario, e un mittente. Contengono un campo opzionale chiamato "content", usato per vari scopi, dipendentemente dal tipo del messaggio. Ha un metodo statico *validate* che converte l'*user input* in un *Message*.
- **Username**: La classe che rappresenta l'username di un utente. Serve per indirizzare i Messaggi ai propri destinatari. Nel server, ogni Username è associato ad un Socket. Ha due metodi statici che ritornano un oggetto con username "server" e uno con username "everyone". Questi due nomi utenti sono proibiti e il server non può assegnarli. Intuitivamente, "server" è l'username che serve per indirizzare un messaggio al server; "everyone" manda il messaggio in broadcast.
- **Type «enumeration»**: È una enumerazione che contiene i possibili tipi che un messaggio può avere:
  - **COMMAND** è un comando, e deve necessariamente essere rivolto al server
  - **MESSAGE** è il messaggio per un utente
  - **SERVER_ANN** è un messaggio dal server per gli utenti; che comunica l'uscita o l'entrata di un client all'interno della chat oppure un messaggio di errore ad esempio.

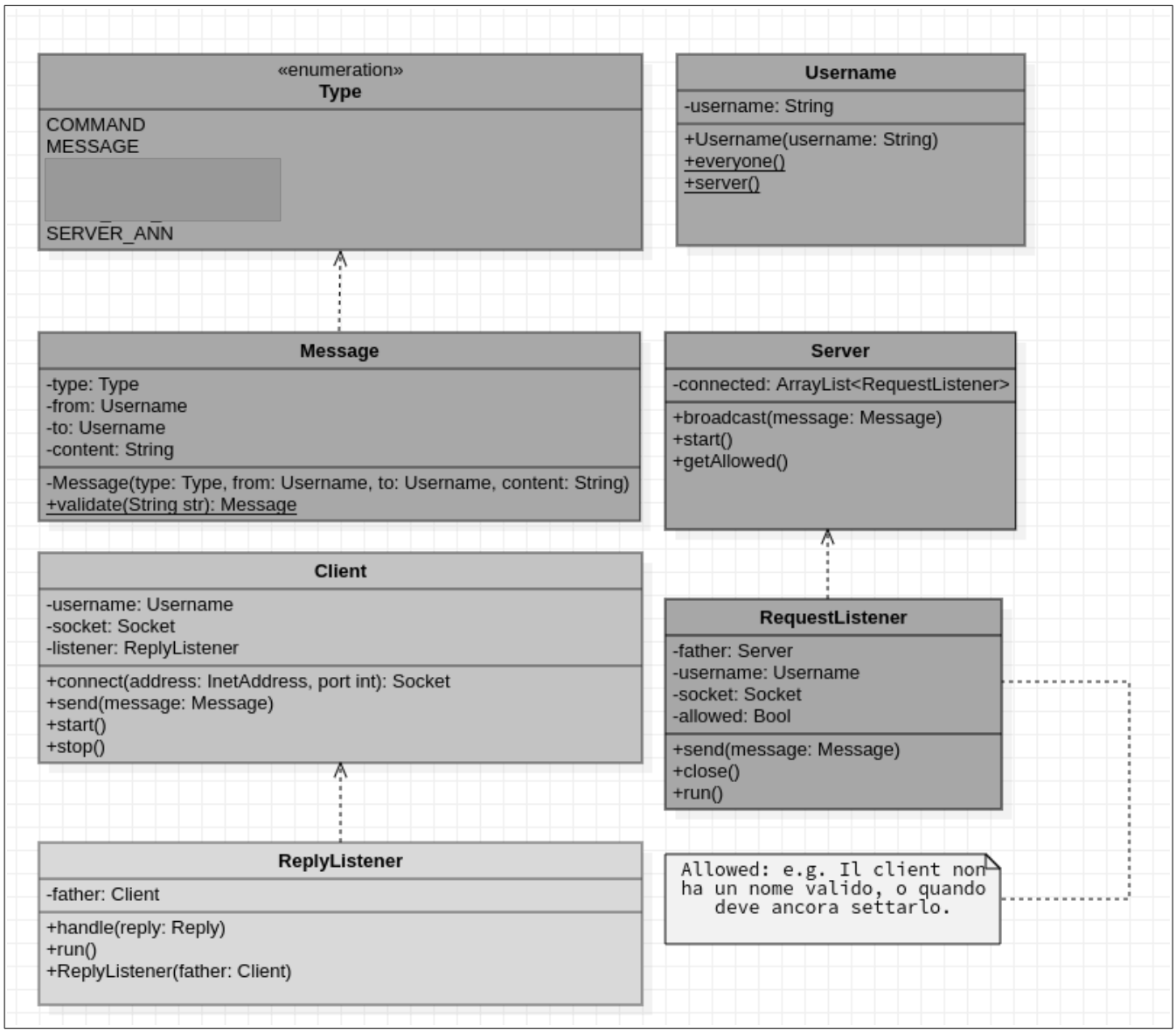
Le classi che appartengono al package "server":

- **Server**: È la classe che rappresenta il server. Ha una lista *connected* che contiene tutti i client connessi, che sono filtrati dal metodo *getAllowed* che ritorna una lista di utenti alla quale la connessione è permessa. Non è permessa agli utenti che non hanno un username valido.

- **RequestListener**: Gestisce la comunicazione con un client. Il campo *allowed* è impostato a *false* fino a quando l'username del client non è valido. *Close* può distruggere il Thread e lo elimina dalla lista *connected* in *Server*. Implementa *Runnable*.

Le classi che appartengono al package "client":

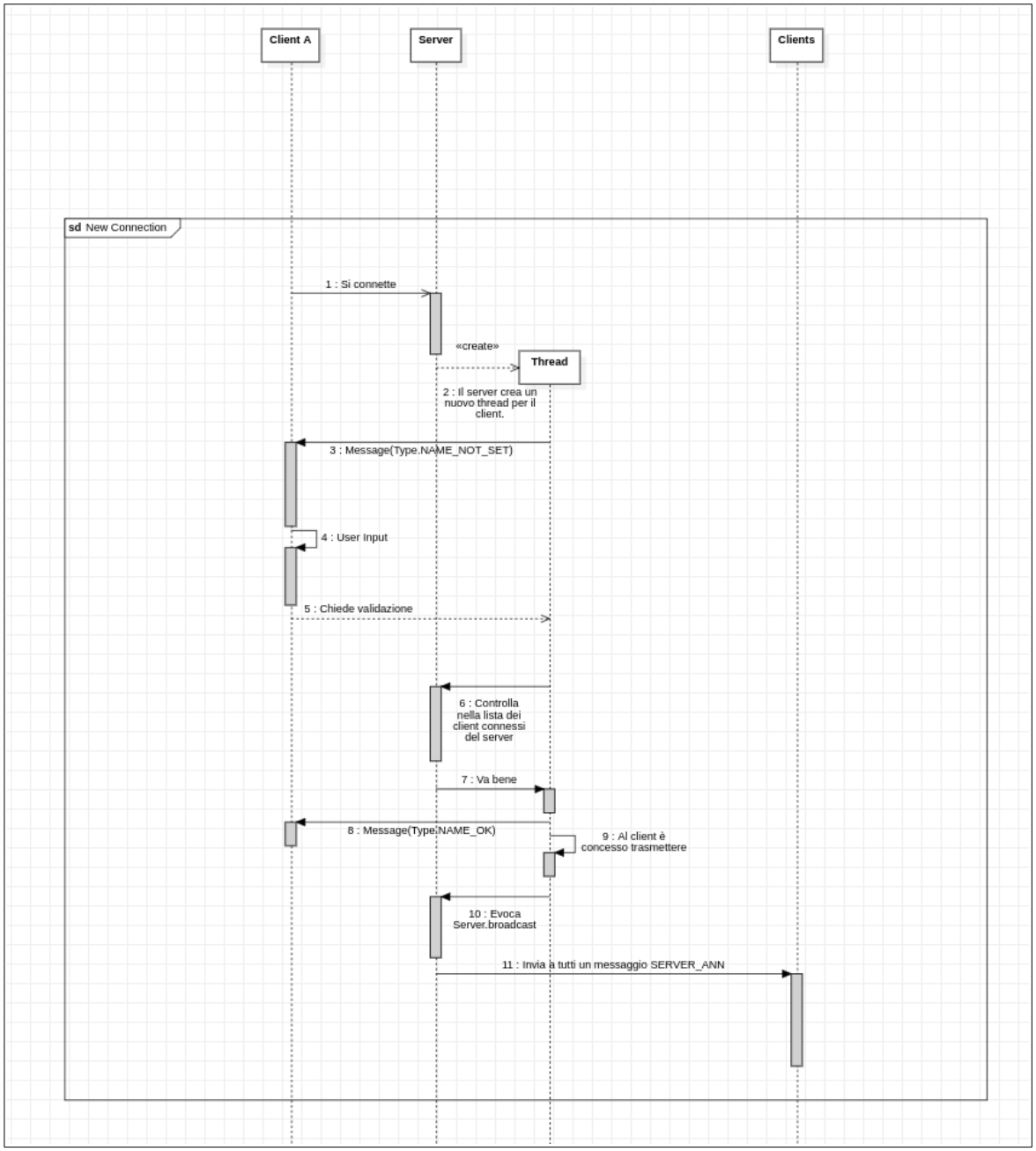
- **Client**: Rappresenta il *Client*. Ad un client corrisponde un *ReplyListener*. Questo Thread legge l'input da tastiera e invia i messaggi con *send(message)*. *Start* avvia il client, s*top* lo ferma; c*onnect* è chiamato dentro *start* per eseguire la connessione.
- **ReplyListener**: Questa è deputata alla lettura dallo stream output del Socket. Ogni messaggio inviato è gestito da questa classe, e viene *risolto* con il metodo *handle*. Implementa la classe *Runnable*.


«enumeration»
Type
COMMAND
MESSAGE
SERVER_ANN
Username
-username: String
+Username(username: String)
+everyone()
+server()
Message
-type: Type
-from: Username
-to: Username
-content: String
-Message(type: Type, from: Username, to: Username, content: String)
+validate(String str): Message
Server
-connected: ArrayList<RequestListener>
+broadcast(message: Message)
+start()
+getAllowed()
Client
-username: Username
-socket: Socket
-listener: ReplyListener
+connect(address: InetAddress, port int): Socket
+send(message: Message)
+start()
+stop()
RequestListener
-father: Server
-username: Username
-socket: Socket
-allowed: Bool
+send(message: Message)
+close()
+run()
ReplyListener
-father: Client
+handle(reply: Reply)
+run()
+ReplyListener(father: Client)
Allowed: e.g. Il client non ha un nome valido, o quando deve ancora settarlo.

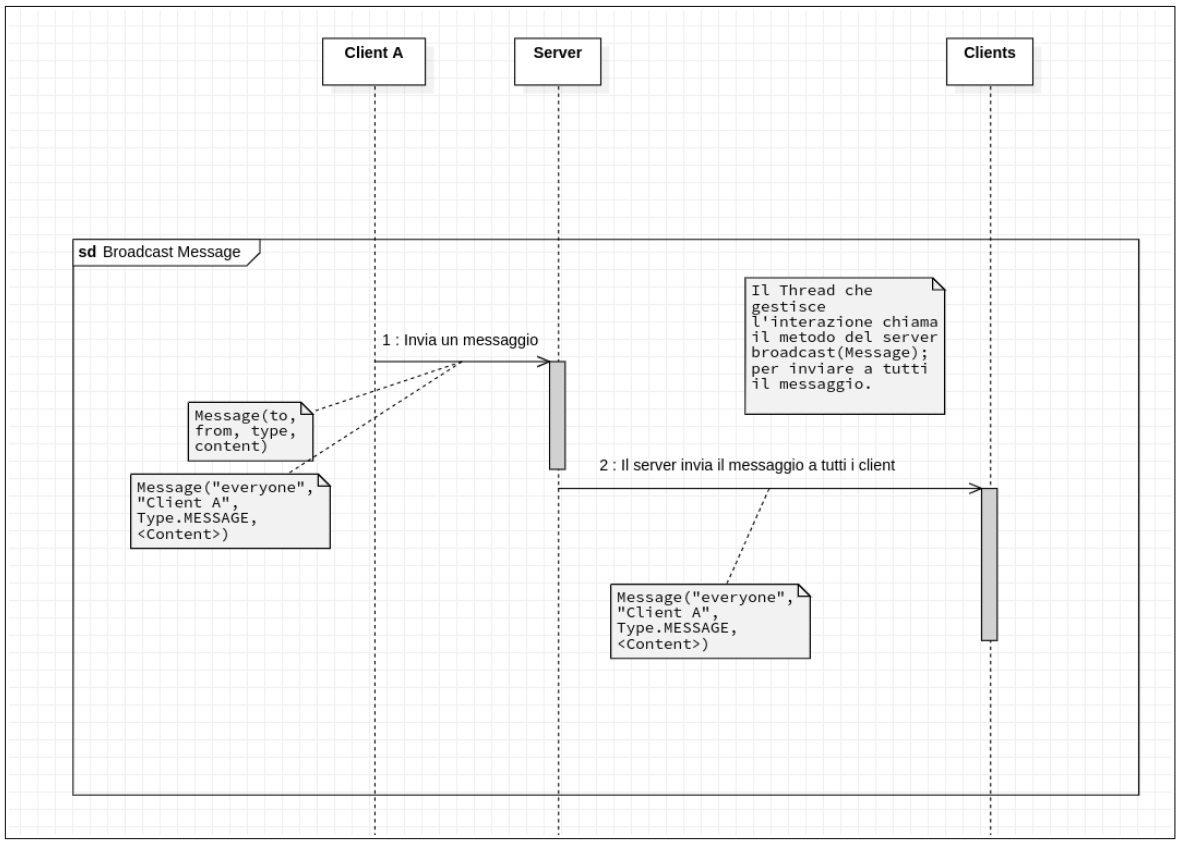
Alcuni diagrammi di sequenza

I diagrammi di sequenza sono un modo di descrivere concettualmente e graficamente le procedure e gli eventi che avvengono durante l'esecuzione, in alcuni momenti particolari. Per spiegare meglio i miei intenti, ho deciso di allegare alcuni diagrammi di sequenza.

- Il diagramma che descrive la nuova connessione. Il nome che l'utente sceglie è corretto.



- Il diagramma che descrive l’invio di un messaggio in broadcast.



Scelte progettuali

Questa sezione elenca alcune soluzioni ai problemi e quesiti che mi sono trovato ad affrontare durante il processo di progettazione.

Tecnologia utilizzata per lo scambio dei messaggi

I messaggi scambiati tra le entità del progetto debbono possedere un formato particolare che sia decifrabile da entrambi i lati e che non mostri falle: utilizzare un formato dove *campi* e *attributi* non sono standardizzati, rischia di rivelarsi una scelta fallace. Ho deciso di utilizzare il formato **JSON**, utilizzato da moltissimi sviluppatori, che conferisce al messaggio un formato flessibile e adatto a cambiamenti. L'altra scelta disponibile, sarebbe stata l'utilizzo dell'**XML**, che però è *prolisso* e *non flessibile*. Siccome la struttura del Messaggio non è definita e cambia da tipo a tipo (e.g. un messaggio NAME_OK non porta con sé nessun contenuto; al contrario, un messaggio MESSAGE bisogna di un contenuto che è difatti il messaggio da inviare) la scelta migliore è quella di utilizzare un formato JSON.

Interruzione di un thread mentre è in corso l'acquisizione da tastiera

Divisione del client in due Thread: come chiudere il processo durante l'acquisizione da tastiera da parte di un sottoprocesso? Documentandomi online, questo problema non ha trovato soluzione in quanto l'acquisizione da tastiera è un processo che avviene a livello del sistema operativo. O meglio esprimendosi, non ha una soluzione se si pretende che la conferma di chiusura debba essere ricevuta dal server. Se l'utente sta digitando da tastiera, il termine del processo può essere svolto solamente dopo che l'utente ha battuto invio. Una soluzione è questa: il metodo *send* del client esegue un controllo sull'input da tastiera: se l'utente digita "`/stop`" (affronterò successivamente la sintassi dei comandi), l'esecuzione può fermarsi senza problemi; dopotutto, è il protocollo TCP che si occupa di garantire una avvenuta ricezione del messaggio dal destinatario, per cui il client non ha bisogno di conferme dal server per spegnersi, perché vengono già inviate alla chiusura del Socket.

Questo problema tuttavia non si pone se si utilizza una interfaccia grafica, in quanto l'acquisizione da tastiera avviene in maniera differente.

Sintassi dei comandi e «campi» del messaggio JSON

Quale sintassi viene data ai comandi affinché possano essere correttamente codificati in un formato JSON? E quali sono i campi di quest'ultimo? Dato che non vi è alcuna regola sulla sintassi dei comandi, giacché si tratta di un artificio che il programmatore crea per facilitare la loro gestione, mi sono permesso di adottare quella che più piacevo. In particolare, il messaggio è stato concepito così:

<table>
<tr><th></th><th colspan="3">Sintassi dei contenuti</th></tr>
<tr><td rowspan="2">Messaggio</td><td>Prefisso</td><td>Destinatario</td><td>Contenuto</td></tr>
<tr><td>@</td><td>everyone oppure l’username di un altro utente</td><td>Il contenuto del messaggio</td></tr>
<tr><td rowspan="2">Esempio</td><td colspan="3">@everyone Ciao a tutti!</td></tr>
<tr><td colspan="3">@marco Ciao marco, come stai?</td></tr>
<tr><td rowspan="2">Comando</td><td>Prefisso</td><td>Nome comando</td><td>Argomenti</td></tr>
<tr><td>/</td><td>stop, name, et.cet.</td><td>Dipendono dal tipo di comando</td></tr>
<tr><td rowspan="2">Esempio</td><td colspan="3">/name francesco</td></tr>
<tr><td colspan="3">/stop</td></tr>
</table>

L’oggetto JSON che viene trasmesso al server è simile al seguente:

```
{“from”:”A”, “to”:”B”, “type”:”MESSAGE”, “args”:{“ciao B”,}}
```

Chiaramente, i campi avranno valori diversi a seconda del messaggio inviato dall’input. Nel prossimo capitoletto viene spiegata la struttura dei messaggi JSON dipendentemente dal tipo del messaggio.

Struttura dei messaggi JSON

COMMAND

COMMAND è un tipo di messaggio che descrive un comando, cioè una richiesta da un Client ad un Server per apportare una modifica. Ogni comando deve essere rivolto al Server, ed ogni comando, ha il tipo COMMAND. Gli argomenti del comando sono due: il primo è il tipo, il secondo è l’argomento, che è opzionale e non necessario per tutti i comandi (non a caso *Disconnessione* non ha nessun argomento).

Tipo di comando	**args**	
	Command Type	Argomento
Cambio nome	CHANGE_NAME	*Nuovo nome*

Disconnessione	DISCONNECT	

Qualche esempio:

- {“from”:”A”, “to”:”B”, “type”:”COMMAND”, “args”: {“CHANGE_NAME”, “nikola21”}}
- {“from”:”A”, “to”:”B”, “type”:”COMMAND”, “args”: {“DISCONNECT”,}}

MESSAGE

MESSAGE è il tipo di messaggio che serve all’invio di informazioni tra un Client ed un altro. È destinato all’invio del server per lo smistamento, ma deve chiaramente essere inviato al destinatario.

L’argomento è solo uno, ed è il contenuto del messaggio.

- {“from”:”A”, “to”:”B”, “type”:”MESSAGE”, “args”:{“Ciao B, come stai?”,}}

SERVER_ANN

SERVER_ANN è l’abbreviazione di Server Announcement ed è il Messaggio scambiato unicamente dal Server al Client. Come il COMMAND, ha un tipo ed un argomento:

Descrizione	**args**	
	SERVER_ANN Type	Argomento
Entra un nuovo utente	JOINED	*Username del nuovo utente*
Esce un utente gia cononesso	LEFT	*Username dell’utente uscito*
Il nome inserito è corretto	NAME_OK	
Il nome inserito non è corretto	NAME_NOT_OK	
L’utente non ha un nome	NEED_NAME	