UNIVERSITÀ DEGLI STUDI DI NAPOLI PARTHENOPE

PROGRAMMAZIONE 3

Francesco De Micco Matricola: 0124002556

Francesca Formisano Matricola: 0124002481

RELAZIONE - IRC CHAT

a.a 2023 - 2024

Indice

1 Traccia

Simulare una chat multiutente basata su IRC. Utilizzare un approccio client/server.

Server:

- Permette agli utenti di connettersi
- Mostra agli utenti una serie di possibili canali attivi (identificati con #)
- Rappresenta un gruppo in cui tutti gli utenti connessi possono inviare messaggi visibili a tutti coloro che sono connessi in quel canale
- Permette all'utente di cambiare il canale su cui è connesso
- Gestisce la collisione tra nomi utenti uguali
- Permette a due utenti di parlare in privato

Client:

- Si connette ad un server specificando un nome utente, non è richiesta la password
- Può richiedere la lista dei canali inviando
  Comando: `/list`
- Può connettersi ad un canale
  Comando: `/join #channel_name`
- Può vedere gli utenti connessi
  Comando: `/users`
- Può inviare messaggi
  Comando: `/msg messaggio`
- Può inviare un messaggio privato ad un utente
  Comando: `/privmsg nickname messaggio`
- Può cambiare il canale su cui è connesso in qualunque momento

Implementare l'utente amministratore che può:

- Espellere un utente dal canale
  Comando: `/kick nickname`
- Bannare/sbannare un utente dal canale
  Comando: `/ban nickname`
  Comando: `/unban nickname`
- Promuovere un utente come moderatore
  Comando: `/promote nickname`

2 Sviluppo

Il presente progetto propone la realizzazione di un sofisticato applicativo software basato su un'architettura client – server, mirato alla simulazione di una chat attraverso l'implementazione di un server multicanale basato sul protocollo di messaggistica istantanea su Internet noto come Internet Relay Chat (IRC).

Il software consentirà agli utenti di interagire in maniera versatile e dinamica. Quest'ultimi saranno in grado di partecipare a discussioni di gruppo, denominati "canali", per favorire la comunicazione all'interno di comunità tematiche. Parallelamente, il sistema offrirà la possibilità di comunicazioni dirette tra due utenti specifici, promuovendo così una comunicazione più personalizzata e mirata.

Gli utilizzatori del prodotto e di conseguenza gli attori evidenziati si distinguono principalmente in due categorie:

- **Client**: Rappresentano gli utenti attivi; essi possono scambiare messaggi con altri utenti client. Inoltre, hanno la flessibilità di cambiare canale o conversazione privata in qualsiasi momento.
- **Admin**: Rappresenta il moderatore (o moderatori) del server, rivestendo un ruolo di gestione più avanzato rispetto agli utenti standard.

Entrambe le categorie di utenti sono identificate attraverso il loro nome utente, il quale deve essere univoco nel sistema.

Il progetto, focalizzato sulla simulazione di una chat basata su IRC, prevede un utilizzo prevalente a livello locale. A tale scopo, il server che ospiterà la chat IRC sarà collocato in locale, su una dedicata macchina server. Questa scelta progettuale apre la strada a diverse applicazioni pratiche, come la comunicazione istantanea all'interno di un'azienda, integrando il prodotto in un terminale, ad esempio, di un server di storage. Ciascun server potrebbe gestire un proprio canale di comunicazione per lo staff e gli addetti alla manutenzione, compresi gli amministratori associati.

3 Tecnologie Utilizzate

3.1 Linguaggio di Programmazione

Abbiamo optato per `Java` come linguaggio principale per il suo mix di portabilità, semplicità d'uso e solidità.

3.2 Architettura utilizzata

Come richiesto, è stata scelta come architettura di comunicazione quella "client/server". L'implementazione di essa è stata resa possibile grazie allìutilizzo delle `Socket`. Le primitive per l'utilizzo di esse come quelle per la connessione e per lo scambio dei messaggi sono fornite di default dal lnguaggio `Java`.

3.3 Gestione dei Log degli Utenti

Per tenere traccia dei log degli utenti che accedono con il solo username, abbiamo implementato un `database MySQL`. Per gestire questo `database` direttamente dal codice `Java`, ci siamo affidati a `JDBC` (`Java Database Connectivity`), garantendo un accesso efficiente e sicuro ai dati.

3.4 Interfaccia Grafica

L'interfaccia grafica è stata sviluppata utilizzando la libreria `Swing` di `Java`. Grazie a `Swing`, abbiamo creato un'interfaccia intuitiva e interattiva per la chat, sfruttando una vasta gamma di componenti grafici come finestre, bottoni e campi di testo.

3.5 Riflessioni

La realizzazione di questo progetto ah richiesto impegno ed una buona conoscenza per quanto rigurarda lo sviluppo distribuito e dell'interazione con i `database` utilizzando `Java`. La combinazione di `JDBC` e `MySQL` ci ha fornito una solida base per la gestione dei dati degli utenti, mentre l'uso di `Swing` ha reso l'esperienza utente della chat estremamente piacevole e intuitiva.

4 Utilizzo dei Pattern

4.1 Singleton

Il design pattern `Singleton` è utilizzato per garantire un unico punto di accesso ad un oggetto. Il suo impiego e sua implementazione è stata utilizzata per la definizione della classe `Server` in quanto tale applicativo necessita di una sola istanza di essa.
Inoltre, essendo tale pattern dotato di due tipologie di inizializzazione, lazy ed eager inizialization si tiene a specificare che l'appproccio utilizzato è quello `Eager` in quanto sicuramente l'istanza sarà utilizzata e difatti istanziarla a prescindere può evitare incongruenze ed evenutali problemi.

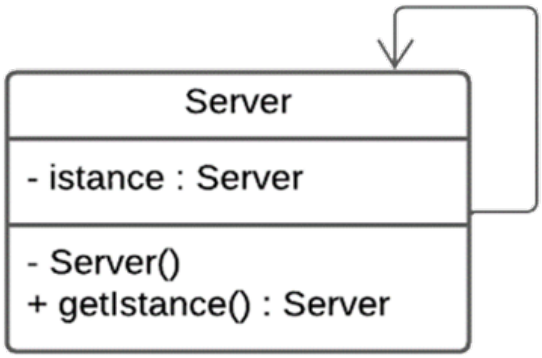

Figura 1: Schema di Singleton

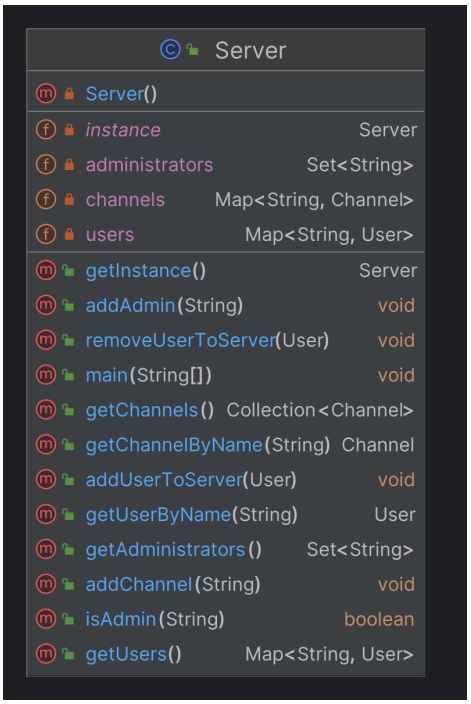

Figura 2: Implementazione Singleton

4.2 Command

Il design pattern `Command` è utilizzato per avere un modo per incapsulare e parametrizzare i comandi utente di base (ossia quelli relativi sia all'utente di default che all'admin) in modo da fornire modularità al sistema. Difatti, tali comandi possono essere utilizzati da qulunque sia il ruolo ricoperto dall'utente.
In questo modo si va a disaccoppiare il mittente della richiesta dal ricevenete garantendo estendibilità in quanto l'introduzione di eventuali nuovi comandi non comporterà sostanziali modifiche al codice, rispettando in pieno l'OCP.

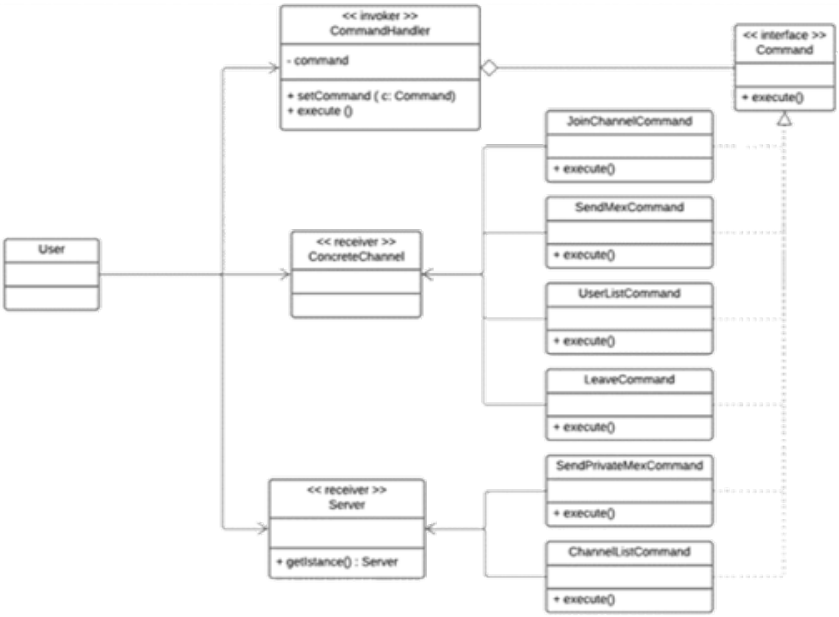

Figura 3: Schema Command

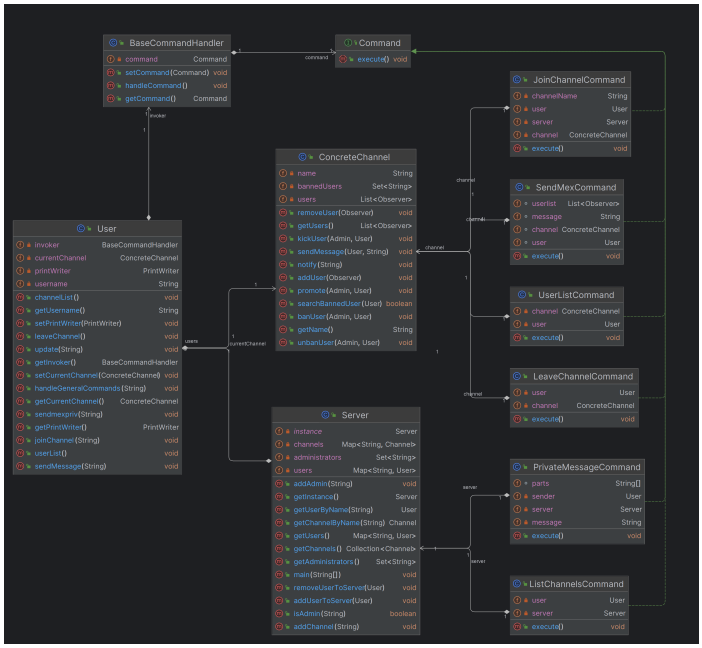

Figura 4: Implementazione Command

4.3 Factory Pattern

Il design pattern `Factory Pattern` è utilizzato per la creazione a runtime dei Canali che ospiteranno le varie chat tra utenti. È stato scelto questa semplificazione di Factory Method e Abstract Factory in quanto in prospettiva futura non vi è la necessità di consentire la creazione di più tipologie di canale, difatti non andrà a violare l'OCP.
Viene preferito ad altre tipologie di creazionali in quanto permette di fornire solo un'interfaccia generale per la creazione, nel nostro caso al Server, delegando la creazione ad una Factory specifica, che quindi nasconderà la sua logica all'esterno.

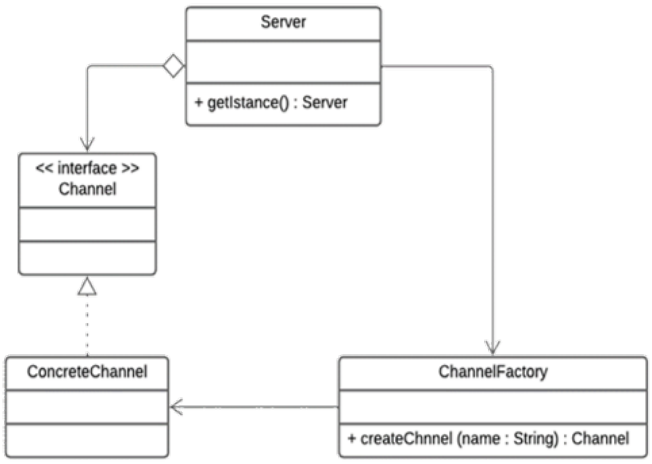

Figura 5: Schema Factory Pattern

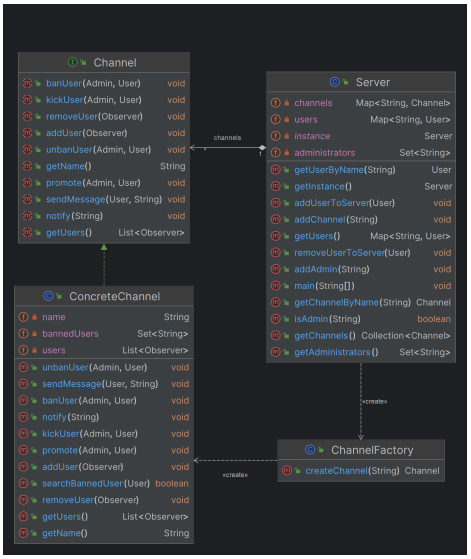

Figura 6: Implementazione Factory Pattern

4.4 Strategy

Il design pattern `Strategy` è utilizzato per l'implementazione dei comandi riservati ai moderatori non esponendo la logica all'esterno e permettere a runtime di far scegliere a run-time ed in maniera dinamica al client quale comando eseguire.
Viene utilizzato in questo modo in quanto permette di incapsulare i songoli comandi di moderatore, e riesce a renderli interscambiaili in maniera dinamica. Questo tipo di approccio risulta molto utile in quanto propone flessibilità, soprattutto nel caso in cui un'utente di base viene promosso a moderatore.

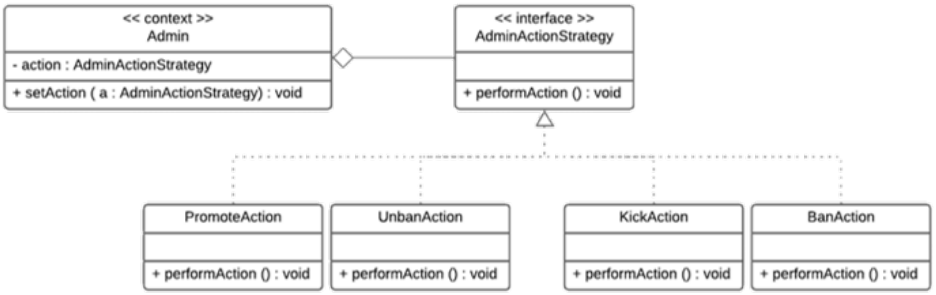

Figura 7: Schema Strategy

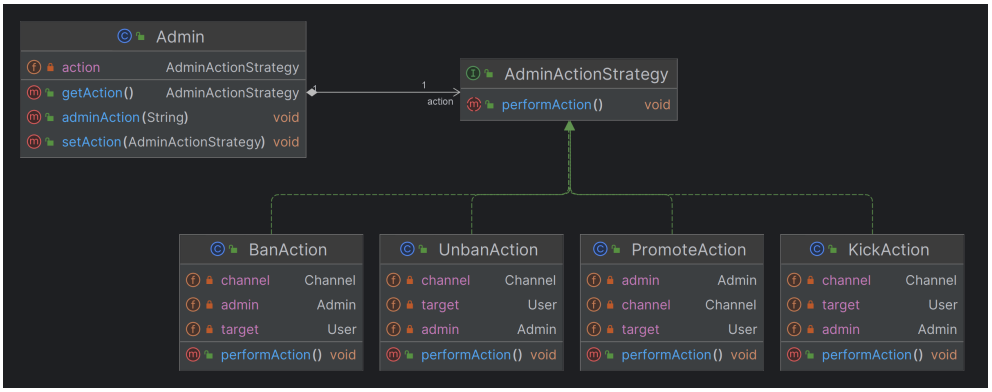

Figura 8: Implementazione Strategy

4.5 Observer

Il design pattern `Observer` è utilizzato per l'implementazione la distribuzione di eventi nei canali. In particolare per la distribuzione ed invio dei messaggi ad ogni client, che siano messaggi di altri utenti, o di semplice gestione, come l'entrare o uscire da un canale etc...
È stato preferito il suo impiego, in quanto, è capace strutturalmente, di creare un'associazione 1 a Molti, che risulta essere proprio la relazione che intercorre tra il canale (che viene "osservato") e gli utenti ad esso connesso (che "osservano" lo stato di esso).
Come è possibile vedere, in questo schema, viene inseria anche la classe "Admin", la quale, estende la classe "User", implementor diretta dell'interfaccia "Observer". Essendo stata prevista uninterfaccia di alto livello ed una tale gerarchia della classi. Il canale che procede ad aggiornare cambiamenti ai client, non saprà in grado di distinguerne la tipologia (se moderatore oppure utente di base).

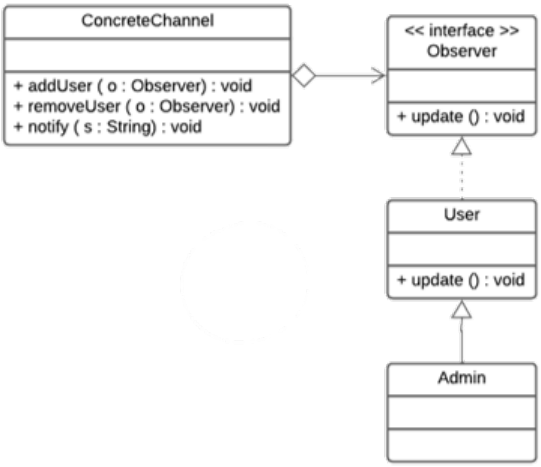

Figura 9: Schema Observer

Figura 10: Implementazione Observer

5 GitHub

Il codice sorgente di questo progetto è disponibile su GitHub al seguente indirizzo:
https://github.com/checcafor/IRC-p3proj