

**Corso di**

**Tecnologie e progettazione di sistemi informatici e di telecomunicazioni**

**Classi 5AInf, 5BInf, 5CInf**

**a.s. 2016/2017**

**Relazione progetto**

**“TCPChat”**

**Studente: Roberto Cardaccia, Luca Sosti**

**Docente/i: Monica Ciuchetti e Luca Mencagli**

# Documento di Specifica dei Requisiti (Software Requirements Specification)

I requisiti sono caratteristiche, proprietà e comportamenti che l’applicazione deve avere al termine dello sviluppo. In questa fase quindi si deve riuscire ad individuare chi interagisce con il sistema e quali azioni compie.

# Introduzione e formulazione del problema

L’ applicazione richiede di realizzare tramite il protocollo TCP, una chat punto a punto per la comunicazione di due host, è richiesto di mostrare sullo standard output i messaggi inviati al server e quelli inviati dall’ host scegliendo un prompt chiaro per comprendere chi sta parlando.

Nella conversazione si possono scambiare dei messaggi speciali noti ad entrambi gli host, con alcuni comandi:

* Smile: se si vuole inviare uno smile come massaggio
* End: terminare la connessione

# Descrizione dell’architettura dell’applicazione (componenti hardware e software)

Durante la progettazione del codice ho usato i nostri PC per quanto riguarda la parte hardware mentre per quanto riguarda la parte software abbiamo utilizzato l’ ambiente di sviluppo NetBeans e la Java Virtual Machine (JVM).

## Attori

* Il server, gestisce la connessione con il client.
* Il client, che scambiano scambia messaggi nella chat.
* Il gestore, che gestisce la messaggistica nella chat
* Il protocollo TCP, che client e server devono conoscere per potersi capire nell’ interpretazione dei messaggi.
* L’ utente che avvia e manda in esecuzione il sistema mettendo prima in ascolto il server in una determinata porta.

## 2.2 Casi d’uso

Server

|  |
| --- |
| **Caso d’ uso:** attendi |
| **ID:** UC1 |
| **Attori:** Server |
| **Precondizioni:**   1. Oggetto server deve essere istanziato 2. Il server deve essere mandato in esecuzione 3. Il server deve poter avere una porta a disposizione per la connessione 4. Aspettare la connessione del client |
| **Sequenze degli eventi:** Viene avviato il main che invoca inizialmente il metodo attendi della classe server |
| **Postcondizioni:** Una volta effettuata la connessione viene stampato un messaggio di conferma a video |

|  |
| --- |
| **Caso d’ uso:** accetta |
| **ID:** UC1 |
| **Attori:** server |
| **Precondizioni:**   1. Oggetto server deve essere istanziato 2. Il server deve essere mandato in esecuzione 3. Il server deve poter avere una porta a disposizione per la connessione 4. Aspettare la connessione del client |
| **Sequenze degli eventi:** Viene accettata la connessione del socket |
| **Postcondizioni:** La connessione viene stabilita |

|  |
| --- |
| **Caso d’ uso:** inviaS |
| **ID:** UC1 |
| **Attori:** server |
| **Precondizioni:**   1. Oggetto server deve essere istanziato 2. Il server deve essere mandato in esecuzione 3. Il server deve poter avere una porta a disposizione per la connessione 4. Aspettare la connessione del client 5. Deve essere stata stabilita la connessione |
| **Sequenze degli eventi:** Viene accettata la connessione del socket |
| **Postcondizioni:** La connessione viene stabilita |

Client

|  |
| --- |
| **Caso d’ uso:** inviaC |
| **ID:** UC1 |
| **Attori:** Client |
| **Precondizioni:**   1. Si deve conoscere l’IP del server, se questo è in locale basterà sapere il local host (127.0.0.1) 2. Si deve aprire la connessione 3. Si deve inviare un messaggio dopo aver istanziato l’oggetto PrintWriter |
| **Sequenze degli eventi:** Una volta avviato il main viene creato lo stream in output e inviato il messaggio scritto attraverso lo stream |
| **Postcondizioni:** Viene inviato il messaggio scritto attraverso lo stream. |

|  |
| --- |
| **Caso d’ uso:** riceviC |
| **ID:** UC1 |
| **Attori:** Client |
| **Precondizioni:**   1. Si deve conoscere l’IP del server, se questo è in locale basterà sapere il local host (127.0.0.1) 2. Si deve aprire la connessione 3. Si deve ricevere un messaggio dopo aver istanziato l’oggetto Scanner per la ricezionedello stream in input 4. Si deve leggere la risposta del server 5. Viene mostrata a schermo la risposta |
| **Sequenze degli eventi:** Una volta avviato il main viene creato lo stream di input della classe Scanner letta la risposta e mostrata a video |
| **Postcondizioni:** Viene ricevuto il messaggio scritto attraverso lo stream di input |

|  |
| --- |
| **Caso d’ uso:** chiudiC |
| **ID:** UC1 |
| **Attori:** Client |
| **Precondizioni:**   1. Si deve conoscere l’IP del server, se questo è in locale basterà sapere il local host (127.0.0.1) 2. Si deve aprire la connessione 3. Si deve chiudere il flusso di input 4. Viene chiuso il flusso di output 5. Viene chiusa la connessione 6. Infine viene mostrato a video che la connessione è stata chiusa, altrimenti verrà generata un’eccezione |
| **Sequenze degli eventi:** Una volta avviato il main e una volta che si è finito di scambiare messaggi vene chiusa la connessione |
| **Postcondizioni:** Viene ricevuto l’output con l’avvenuta chiusura della connessione |

## Diagramma dei casi d’uso









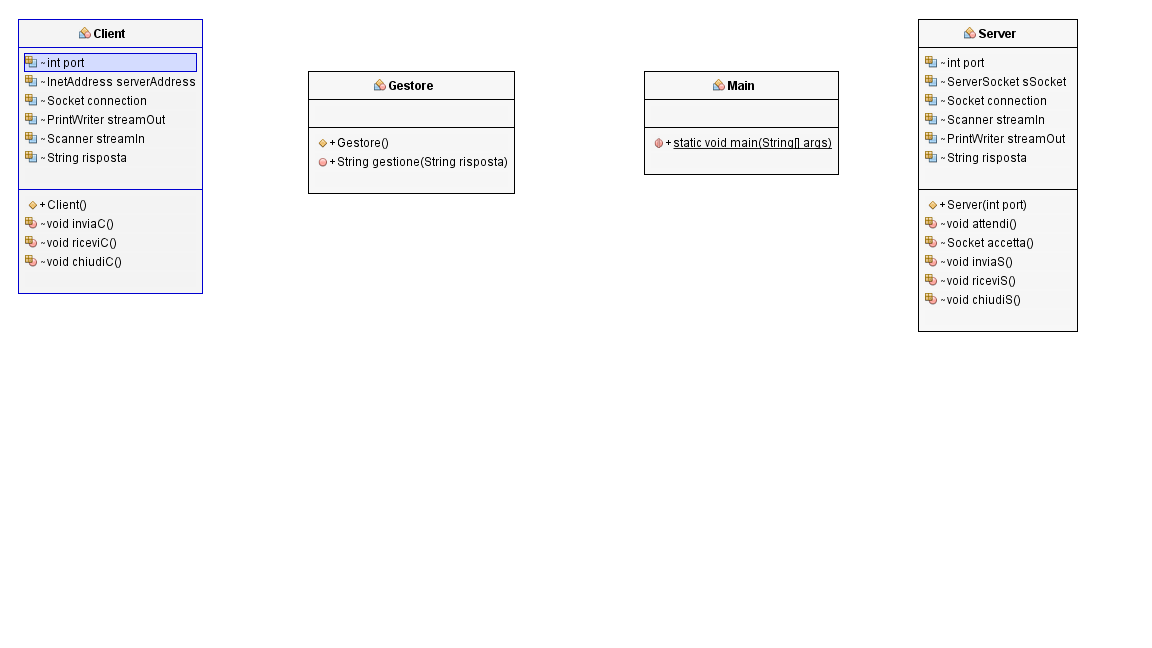
## 2.4 Vincoli e tecnologie usate

La nostra applicazione ha alcuni vincoli come ad esempio non permette, per il momento, di far dialogare più di un client con il server, inoltre non è possibile scegliere il nome utente per sapere chi sta dialogando.

L’applicazione non mostra se l’utente con cui vogliamo messaggere è disponibile in quanto è presente un solo client, infine il progetto non presenta un’interfaccia grafica, ma la comunicazione viene gestita attraverso CLI (Command Line Interface).

# Documento di progettazione ed implementazione dell’applicazione

# 1. Diagramma delle classi



# 5. Commento del codice

Il commento è riportato sul codice.

Inoltre è presente la documentazione JavaDoc attraverso un collegamento internet denominato “All Classes”.

# 7. Test dell’applicazione