Università degli studi di Napoli Parthenope

Corso di reti di calcolatori e Lab.

Traccia progetto: ***Università***

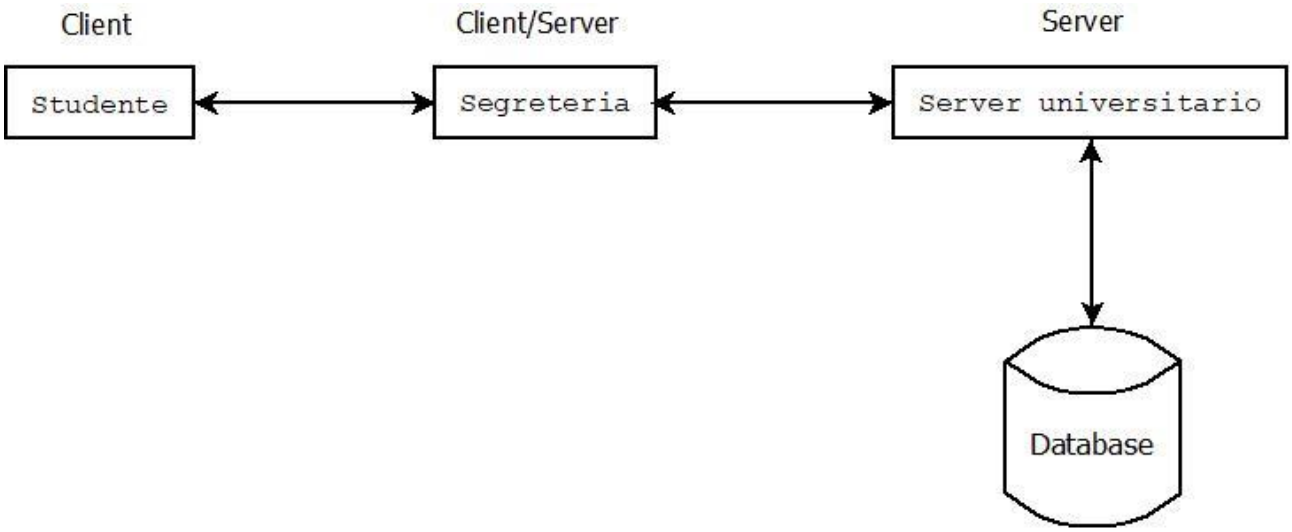
Daniele Biagio De Luca - 0124002504

Descrizione del progetto

Si vuole scrivere un'applicazione client/server concorrente per la gestione degli esami universitari. In particolare:

- La segreteria deve poter inserire nuovi esami sul server dell'università, inoltrare le richieste di prenotazione degli studenti al server universitario e fornire le date degli esami richieste dallo studente.
- Lo studente può visualizzare le date di esame di un determinato corso e inviare una prenotazione a una di esse alla segreteria.
- Il server universitario riceve sia l'aggiunta di nuovi esami, che le richieste di prenotazioni inoltrate da parte della segreteria, inviando il numero di prenotazione.

Schema dell'architettura

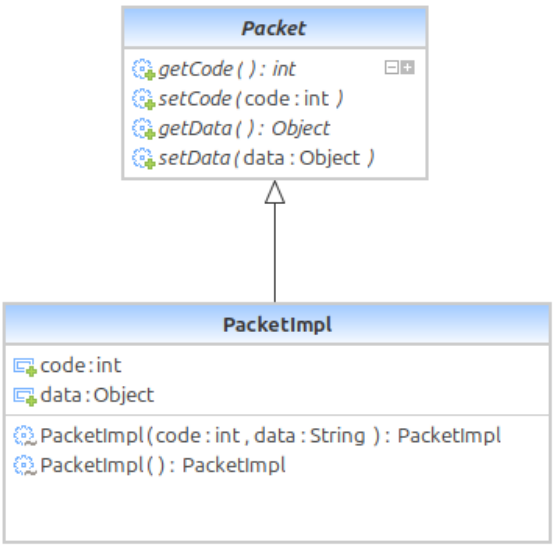


L'architettura dell'applicazione si basa sulle seguenti entità: **lo studente** (*che funge da client*), **la segreteria** (*ricopre il ruolo sia di client che di server*) e il **server universitario** (*server*). Per effettuare le proprie operazioni, lo studente invia le richieste al server della segreteria, che le inoltra al server universitario. Quest'ultimo effettua quindi le elaborazioni necessarie e restituisce, a ritroso, il risultato. L'inserimento di nuovi esami viene effettuato tramite l'applicazione client della segreteria.

Per la memorizzazione delle prenotazioni degli studenti e degli esami, si è optato l'utilizzo di un database relazionale (**Mysql**), le cui operazioni di recupero e inserimento dati vengono eseguite solo dal server universitario. Come linguaggio di programmazione è stato utilizzato Java, in quanto portabile e fornisce un'API per la gestione delle socket (sia TCP che UDP) semplice da utilizzare e ben documentata.

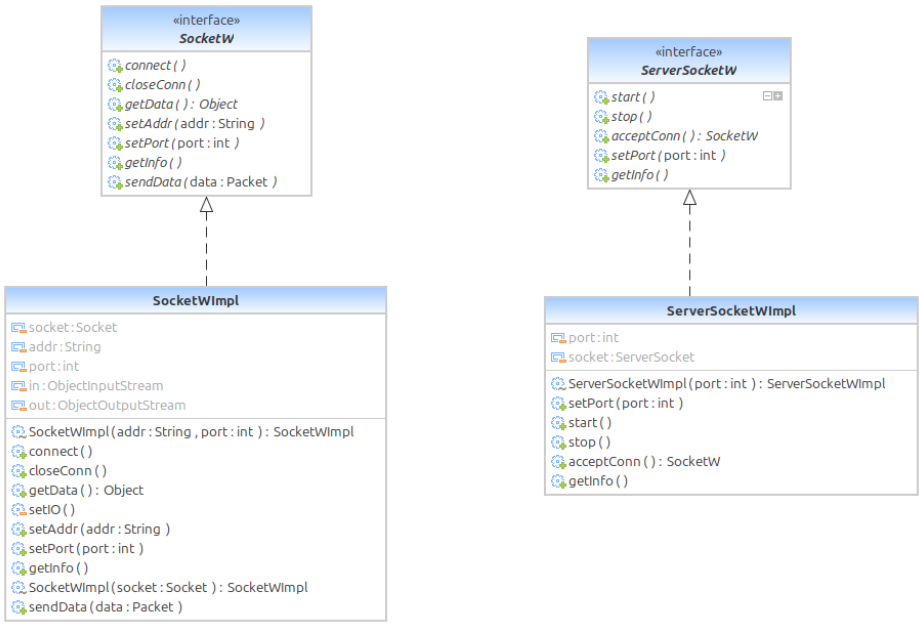
Dettagli implementativi dei Client/Server

Per lo sviluppo del sistema sono state create le seguenti classi:



Packet

Classe astratta che definisce un pacchetto che contiene i dati da mandare tramite la socket, tra cui un codice che funge sia da codice richiesta (nel momento in cui viene inviato al server), sia da codice esito (nel momento in cui viene inviato al client). I dati vengono memorizzati nel campo data di tipo **Object**. In questo modo possono essere inseriti dati di tipo diverso all'interno del pacchetto. Un'alternativa è parametrizzare il tipo del campo *data*. I metodi della classe si occupano semplicemente di impostare e ottenere i dati dai campi.



SocketW

Interfaccia che definisce una socket con i relativi metodi. Viene implementata dalla classe **SocketWImpl**. Di seguito vi sono i metodi più importanti:

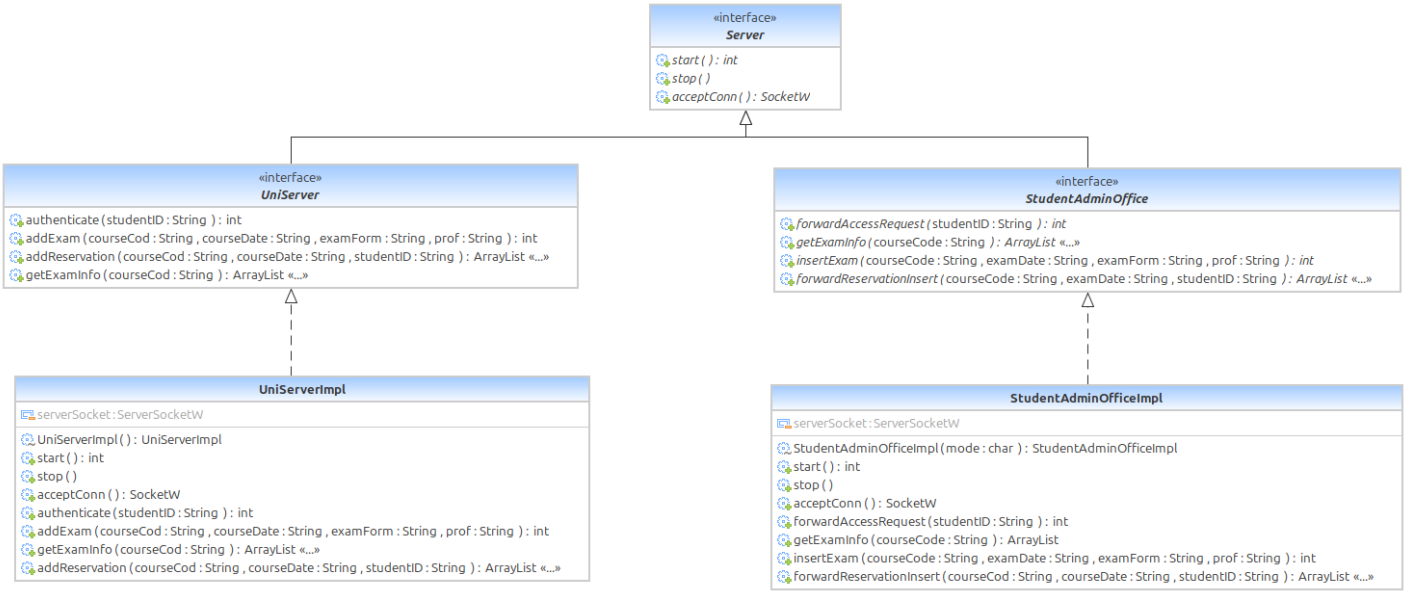
- Il metodo *connect()* effettua la connessione ad un server tramite l'indirizzo IP e la sua porta;

- *closeConn()* chiude la connessione aperta stabilita precedentemente;
- *setIO()* si occupa di impostare i flussi di input e output della socket, che rispettivamente leggono e scrivono oggetti. Viene invocato in entrambi i costruttori;
- Il metodo *sendData()* invia i dati attraverso la socket, sotto forma di oggetto di classe **Packet**. Il metodo *getData()*, invece, restituisce dati in ingresso sotto forma di oggetto di classe **Packet**.

ServerSocketW

Interfaccia che definisce una socket server con i relativi metodi. Viene implementata dalla classe **SocketServerWImpl.**

- *start()* ha lo scopo di aprire il server, mettendolo in ascolto sulla porta memorizzata nel campo port;
- *stop()* interrompe l'ascolto del server sulla porta;
- *acceptConn()* accetta una connessione in arrivo al server, restituendo un oggetto di tipo **SocketW** per scrivere e leggere dati.



Server

Interfaccia che definisce un server generico. I metodi sono semplicemente *start(),* che restituisce un numero intero in base all'esito dell'operazione (un valore diverso da zero indica che si è verificato un errore); *stop()* interrompe il server mentre *acceptConn()* accetta una connessione tramite la socket server e restituisce un oggetto di tipo **SocketW**.

UniServer

Interfaccia che estende **Server** e definisce le operazioni eseguite dal server universitario. Viene implementata da **UniServerImpl**.

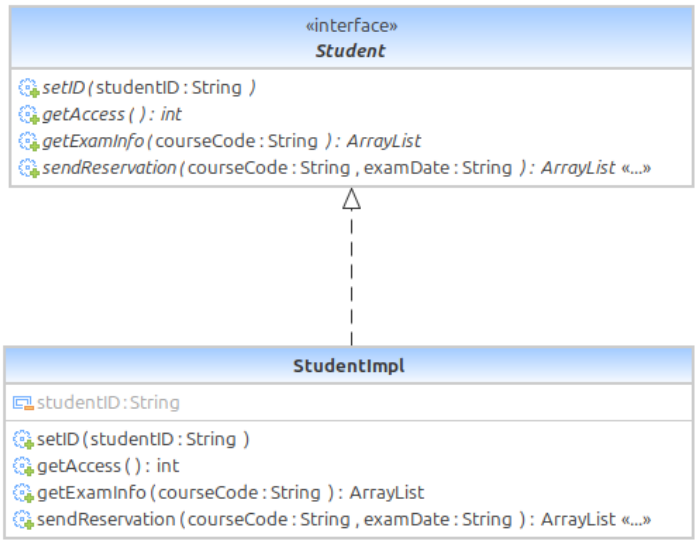
- Il metodo *authenticate()* ha lo scopo di autenticare uno studente data la sua matricola. Restituisce un intero che sarà uguale a 0 se la matricola inserita è corretta;
- *getExamInfo()* restituisce le date di esame di un corso, dato il suo codice. L'output è un array contenente sia le informazioni richieste che un codice di esito;

- *addReservation()* si occupa di inserire la prenotazione di uno studente ad un esame. Richiede il codice del corso, la data dell'esame e la matricola dello studente. Restituisce un array contenente sia il n. di prenotazione, che il codice di esito;
- *addExam()* inserisce un esame di un determinato corso. Tale operazione viene richiesta dalla segreteria, la quale deve passare le informazioni necessarie quali il codice corso, la data dell'esame, la modalità d'esame e il docente.

StudentAdminOffice

Interfaccia che estende **Server** e definisce le operazioni lato server e client eseguite dalla segreteria. Viene implementata da **StudentAdminOfficeImpl**.

- Il metodo *forwardAccessRequest()* inoltra, al server universitario, la richiesta di autenticazione da parte di uno studente. Restituisce il codice di esito da inserire nel pacchetto da inviare al client;
- *getExamInfo()* si occupa di inoltrare, al server universitario, la richiesta di visione delle date di esame di un corso da parte dello studente. Il metodo restituisce un array contenente sia le informazioni richieste, sia il codice di esito;
- Il metodo *insertExam()* invia la richiesta di inserimento di un esame al server universitario. I parametri richiesti sono il codice del corso, la data dell'esame, la modalità dell'esame e il docente;
- *forwardReservationInsert()* inoltra la richiesta di prenotazione di un esame, da parte di uno studente, al server universitario.

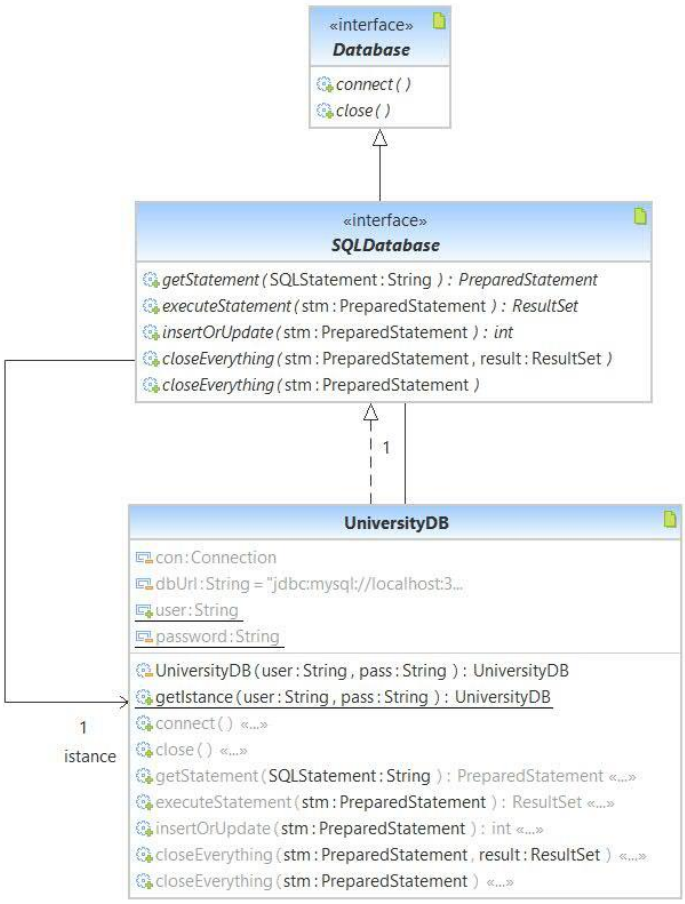


Student

Interfaccia che definisce uno studente generico e le operazioni relative. Viene implementata dalla classe **StudentImpl**, che presenta un campo per memorizzare la matricola dello studente.

- *getAccess()* ha il compito di autenticare lo studente tramite la sua matricola. Invia al server della segreteria la richiesta, e riceve l'esito;

- Il metodo *getExamInfo()* ricava le date di esame di un corso individuato dal codice corso inserito in input. Restituisce un array contenente sia le informazioni richieste, che il codice di esito;
- Tramite il metodo *sendReservation(),* viene inviata, al server della segreteria, la richiesta di prenotazione ad un esame di un determinato corso. Restituisce in output un array di interi contenente sia il numero di prenotazione che il codice di esito.



Database

Le interfacce **Database** e **SQLDatabase** sono state create al fine di gestire la connessione al database e le varie operazioni che si possono effettuare. La prima definisce semplicemente i metodi per connettersi al database e per chiudere la connessione, mentre la seconda definisce i metodi necessari per effettuare varie operazioni con database di tipo SQL. Tale interfaccia è stata creata in modo tale da poter ospitare altri metodi per eseguire operazioni diverse da quelle tipiche (reperimento e inserimento dati). **UniversityDB** implementa **SQLDatabase** ed è progettata sia per connettersi a qualsiasi database SQL che per restituire una sola istanza di sé stessa. Per la connessione a Mysql, è stato utilizzato il driver *Connector/J*.

Parti rilevanti di codice

La seguente immagine illustra il codice dell'applicazione principale del server universitario:

```java
// Applicazione principale del server universitario
package gestionale_Uni;

public class UniServerApp
{

    public static void main(String[] args)
    {
        // Oggetto di classe UniServer
        UniServer server = new UniServerImpl();
        if(server.start() == 0) // Controlla se l'avvio del server non dà errori
        {
            while(true)
            {
                SocketW socket = server.acceptConn();   // Accetta connessioni
                // Se il socket restituito non è null...
                if(socket != null)
                {
                    // Avvia un thread, passando il socket appena ottenuto e l'oggetto originale UniServer
                    new UniServerThread(socket,server);
                }
            }
        }
    }
}
```

Tramite il metodo *start()* dell'oggetto **server** di tipo **UniServer,** l'applicazione si mette in ascolto sulla porta precedentemente impostata (1027 in questo caso). L'applicazione prosegue l'esecuzione se l'operazione di apertura server non restituisce errori e, tramite un ciclo while infinito, rimane in ascolto sulla porta. Nel momento in cui arriva una connessione, questa viene accettata con il metodo *acceptConn()*, che restituisce una socket di tipo **SocketW** per la comunicazione con il client. Una volta accettata la connessione, viene creato un thread concorrente per servire la richiesta appena arrivata, passando la socket per la comunicazione e il riferimento all'oggetto memorizzato nella variabile **server** (questo per evitare di creare un nuovo oggetto di tipo UniServer nel thread).

```java
// Classe che definisce un thread con tutte le operazioni effettuate dal server universitario
package gestionale_Uni;

import java.util.ArrayList;

public class UniServerThread implements Runnable
{
    // Variabili necessarie per l'esecuzione
    private UniServer server;
    private SocketW socket;
    private int reqCode, statusCode, resultCode, reservationNumb;
    private String studentID, courseID,examDate, examForm, prof;
    private ArrayList examInfo;
    private ArrayList<String> data;
    private ArrayList<Integer> result;
    private Packet p, s;

    /* Costruttore che ha come parametri il socket restituito dalla SocketServer e l'oggetto originale
     * 'server' di UniServer */
    UniServerThread(SocketW socket, UniServer originalObj)
    {
        // Imposta i campi socket e server e fa partire il thread
        this.socket = socket;
        this.server = originalObj;
        new Thread(this).start();
    }

    // Metodo che contiene le operazioni che il server deve effettuare
    @Override
    public void run()
    {
        try
```

```java
    public void run()
    {
        try
        {
            // Ottiene un pacchetto dati dalla comunicazione e ricava il codice
            p= (Packet) socket.getData();
            reqCode = p.getCode();
            System.out.println("Codice richiesta: " + reqCode + "\n");
            // Effettua le operazioni in base al codice richiesta contenuto nel pacchetto
            switch(reqCode)
            {
            case 0:     // Codice 0 -> Autenticazione studente
                // Estrae dal pacchetto la matricola
                studentID = (String) p.getData();
                System.out.println("Pronto ad autenticare lo studente di matricola " + studentID + "\n");
                // Verifica che la matricola sia corretta e restituisce il codice di esito dell'operazione
                resultCode = server.authenticate(studentID);
                System.out.println("Richiesta di autenticazione servita\n");
                // Imposta il pacchetto risposta con il codice esito e lo invia al server segreteria
                s = new PacketImpl();
                s.setCode(resultCode);
                socket.sendData(s);
                break;
            case 1: // Codice 1 -> Richiesta visione esami
                // Estrae dal pachetto il codice del corso
                courseID = (String) p.getData();
                // Invoca il metodo apposito che restituisce le info sottoforma di array
                examInfo = server.getExamInfo(courseID);
                s = new PacketImpl();
                // Estrae dall'array restiutito il codice di esito e lo inserisce nel pacchetto risposta
                statusCode= (int) examInfo.get(examInfo.size() - 1);
```

```java
                statusCode= (int) examInfo.get(examInfo.size() - 1);
                examInfo.remove(examInfo.size() - 1);
                s.setCode(statusCode);
                // Inserisce nel pacchetto l'array delle informazioni e lo invia al server della segreteria
                s.setData((ArrayList<ArrayList<String>>) examInfo);
                socket.sendData(s);
                System.out.println("Richiesta visione esami servita\n");
                break;
            case 2: // Codice 2 -> Inserimento esame
                System.out.println("Richiesta di inserimento esame ricevuta");
                // Estrae dal pacchetto le informazioni da inserire nel database
                data = (ArrayList<String>) p.getData();
                courseID = data.get(0);
                examDate = data.get(1);
                examForm = data.get(2);
                prof = data.get(3);
                // Effettua l'aggiunta dell'esame, ottenendo il codice di esito della operazione
                resultCode = server.addExam(courseID, examDate, examForm, prof);
                // Imposta il pacchetto risposta inserendo il codice di esito e lo invia
                s = new PacketImpl();
                s.setCode(resultCode);
                socket.sendData(s);
                System.out.println("Operazione conclusa");
                break;
            case 3: // Codice 3 -> Inserimento prenotazione
                System.out.println("Richiesta di prenotazione per esame ricevuta\n");
                // Estrae i dati per la prenotazione
                data = (ArrayList<String>) p.getData();
                courseID = data.get(0);
                examDate = data.get(1);
                studentID = data.get(2);
                // Invoca il metodo apposito, che restituisce il codice di esito e il n. di prenotazione
```

```java
                data = (ArrayList<String>) p.getData();
                courseID = data.get(0);
                examDate = data.get(1);
                studentID = data.get(2);
                // Invoca il metodo apposito, che restituisce il codice di esito e il n. di prenotazione
                result = server.addReservation(courseID, examDate, studentID);
                // Imposta il pacchetto risposta col codice esito e con i dati, inviandolo
                s = new PacketImpl();
                statusCode = (int) result.get(result.size() - 1);
                reservationNumb = (int) result.get(0);
                s.setCode(statusCode);
                s.setData(reservationNumb);
                socket.sendData(s);
                break;
            default:
                System.out.println("Codice richiesta non valido\n");
                break;
            }
        }
        catch(Exception e)  // Gestione eccezioni
        {
            System.out.println("Errore generico interno..." + e.getMessage() + "\n");
        }
        finally
        {
            // Chiude la connessione
            socket.closeConn();
        }
    }

}
```

Le immagini precedenti mostrano, invece, le operazioni eseguite in un thread dell’applicazione del server universitario. Nel costruttore vengono impostati i campi

riguardanti l'oggetto di tipo **UniServer** e **SocketW** con i valori precedentemente passati, per poi far partire il server. Il server riceve dal client un pacchetto contenente dei dati e un codice, per identificare la richiesta effettuata. In base al codice, la richiesta viene servita in modo diverso invocando il metodo apposito. Successivamente viene creato un pacchetto dati inserendo il codice di esito dell'operazione, gli eventuali dati e questo viene inviato al client. Infine la connessione con il client viene chiusa.

In generale, le stesse operazioni vengono effettuate dal server della segreteria.

Il significato dei codici richiesta varia in base alle entità coinvolte nella comunicazione.

Studente -> Segreteria

0: Autenticazione studente;
1: Richiesta visione esame;
2: Richiesta prenotazione esame.

Segreteria -> Server universitario

0: Inoltro richiesta autenticazione studente;
1: Inoltro richiesta visione esame;
2: Inserimento esame;
3: Inoltro richiesta prenotazione esame.

Istruzioni per l'esecuzione

È necessario eseguire i file dal terminale, aprendo finestre di terminale diverse. È consigliato entrare nella cartella *bin* del progetto e aprirle da lì. I file vanno eseguiti in uno specifico ordine: *UniServerApp*, *StudentAdminOfficeSApp*, *StudentAdminOfficeCApp* (per l'inserimento di nuovi esami), *StudentClientApp*. I file devono essere aperti con il semplice comando **Java,** eccetto per il primo file, che va eseguito con il seguente comando: *java -cp .:$HOME/gestionale_Uni/lib/mysql-connector-j-9.1.0.jar gestionale_Uni.UniServerApp* . In questo modo verrà incluso anche il driver per la connessione al database Mysql.