Università degli Studi di Napoli
"Parthenope"

DIPARTIMENTO DI SCIENZE E TECNOLOGIE
CORSO DI RETI DI CALCOLATORI

Telepass

Gaudio Lorenzo
012400/2500

Progetto realizzato per lo svolgimento dell'esame di Reti di calcolatori,integrato a Programmazione III ed Ingegneria del software ed InterazioneUomo-Macchina

Anno Accademico 2023/24

Indice

Capitolo 1
Descrizione del progetto

1.1 Principali caratteristiche del progetto

Il progetto "Telepass" è un'applicazione client-server che facilita l'accesso agli utenti come amministratori o utenti standard e gestisce la simulazione di ingressi e uscite da un casello o altre operazioni per l'amministratore quali revoca o inserimento del dispositivo Telepass.

Il sistema è composto dalle seguenti classi:

ClientAccesso: Un client che consente agli utenti di selezionare il tipo di accesso (amministratore o utente) e comunica con il server di accesso per ottenere l'autorizzazione.

ServerAccesso: Un server che gestisce le richieste di accesso degli utenti, autorizzando gli accessi come amministratori o utenti e gestendo le comunicazioni con il client.

ServerAccessoHandler: Un gestore di thread che gestisce le richieste di accesso degli utenti sul lato del server di accesso, comunicando con il client e inviando risposte in base al tipo di accesso richiesto.

ClientSimulazione: Un client che consente agli utenti di simulare l'ingresso o l'uscita da un casello, comunicando con il server di simulazione per inviare i dati e ricevere conferme.

ServerSimulazione: Un server che gestisce le simulazioni di ingresso e uscita da un casello, interagendo con i client di simulazione e inviando i risultati al server centrale per l'archiviazione.

ServerSimulazioneHandler: Un gestore di thread che gestisce le simulazioni di ingresso e uscita sul lato del server di simulazione, comunicando con i client di simulazione e inviando i risultati al server centrale.

1.2 Punti chiave del progetto:

Gestione dei Dispositivi Telepass: Il sistema gestisce l'inserimento e la rimozione dei dispositivi Telepass per le macchine, consentendo agli amministratori di aggiungere nuovi dispositivi, rimuovere quelli esistenti e gestire le operazioni di conferma o annullamento.

Gestione degli Errori e delle Eccezioni: Il sistema gestisce eventuali errori o eccezioni durante le operazioni di accesso o simulazione. Fornisce messaggi di errore significativi agli utenti e registra le eccezioni per fini di debug e manutenzione, garantendo la robustezza del sistema.

Interfaccia Utente: Gli utenti interagiscono con il sistema attraverso un'interfaccia utente intuitiva, che fornisce una panoramica chiara dei menu disponibili e delle opzioni di input per avviare le simulazioni o gestire gli accessi.

Capitolo 2
Descrizione e schema dell’architettura

2.1 Descrizione dell’architettura

2.1.1 ServerAccesso

Il ServerAccesso è il fulcro dell'autenticazione degli utenti. Esso accetta le richieste di accesso dai client, autentica gli utenti come amministratori o utenti standard, e gestisce le comunicazioni tra il client e il server centrale. Il ServerAccessoHandler gestisce ciascuna connessione in un thread separato, gestendo le richieste di accesso e comunicando con il client.

2.1.2 ServerSimulazione

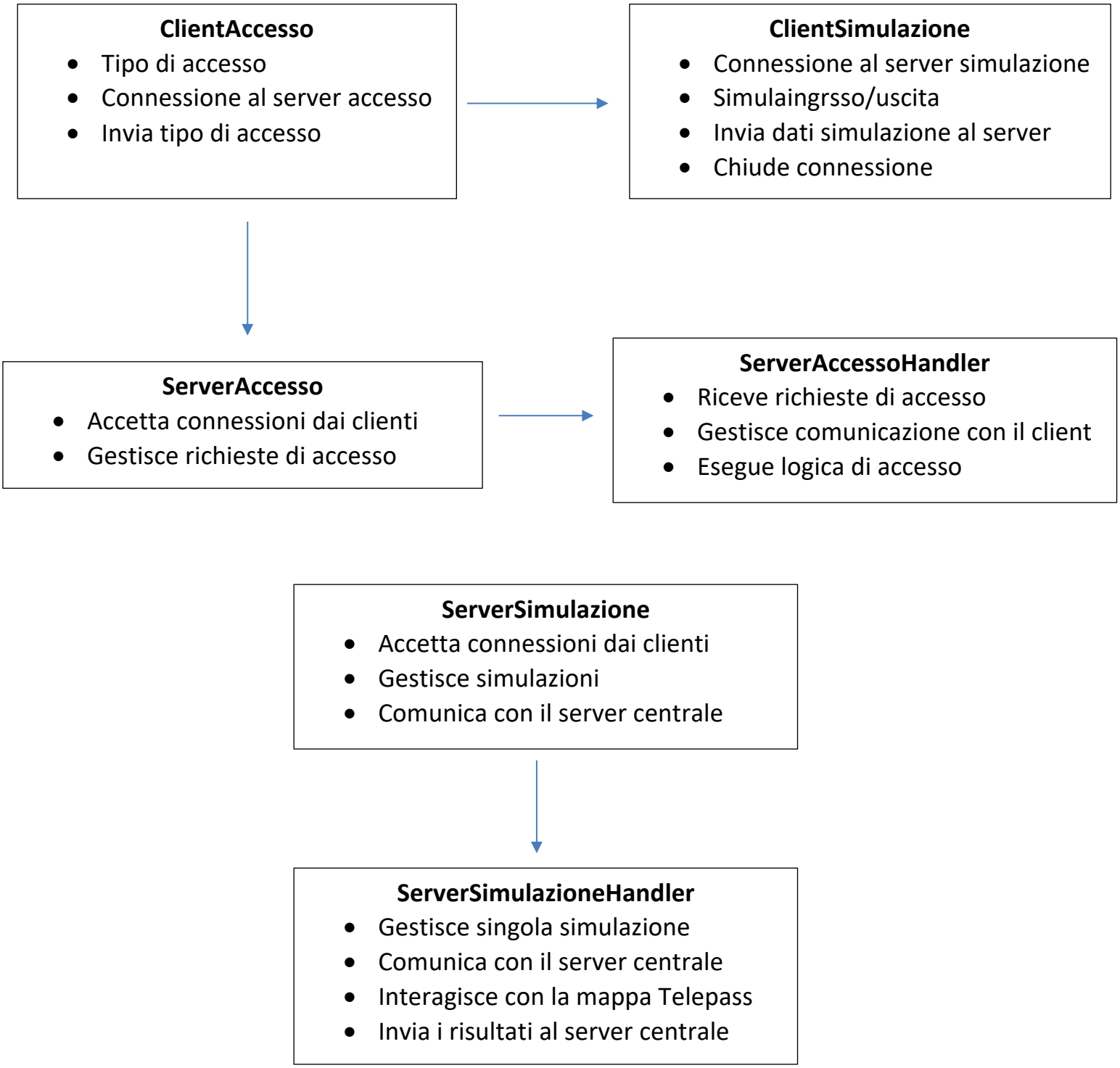
Il ServerSimulazione è responsabile della simulazione degli ingressi e delle uscite da un casello. Accetta le connessioni dai client di simulazione, gestisce le simulazioni in thread separati (ServerSimulazioneHandler) e invia i risultati al server centrale per l'archiviazione.

2.1.3 ClientAccesso e ClientSimulazione

Il ClientAccesso consente agli utenti di selezionare il tipo di accesso desiderato (amministratore o utente) e comunica con il ServerAccesso per ottenere l'autorizzazione. Il ClientSimulazione consente agli utenti di simulare l'ingresso o l'uscita da un casello, comunicando con il ServerSimulazione per inviare dati e ricevere conferme.

2.2 Schema dell'architettura

Questo progetto utilizza un'architettura Client-Server, in cui il server centrale gestisce direttamente le richieste di accesso degli utenti. Una volta ricevute, le instrada ai server di simulazione per l'elaborazione. I server di simulazione completano le operazioni richieste e inviano i risultati



Capitolo 3
Implementazione dei Componenti Client-Server

3.1 ClientAccesso

Connessione ai server:

- La classe ClientAccesso si connette prima al server di accesso sulla porta 54321 e successivamente al server di simulazione sulla porta 8000.
- Viene utilizzato il costrutto try-with-resources per garantire la chiusura automatica delle risorse (Socket, PrintWriter, BufferedReader) al termine del blocco try.

Gestione del tipo di accesso:

- L'utente inserisce il tipo di accesso (amministratore o utente) che viene inviato al server di accesso.
- Il server di accesso risponde con un messaggio che viene stampato a console.

Menù Utente:

- Se l'utente è di tipo "utente", viene visualizzato un menu con opzioni per simulare l'ingresso al casello o uscire dall'applicazione.
- L'utente può inserire la propria scelta (1 o 2) e inviare i dati di ingresso al server di simulazione.

Menù Amministratore:

- Se l'utente è di tipo "amministratore", viene visualizzato un menu con opzioni per inserire o rimuovere dispositivi Telepass o uscire dal menu.
- L'amministratore inserisce la propria scelta (1, 2 o 3) e invia le operazioni corrispondenti al server di simulazione.

Chiusura dello scanner:

- Alla fine, lo scanner viene chiuso per liberare le risorse.

```java
public class ClientAccesso {
    public static void main(String[] args) {
        // Inizializzazione di uno scanner per la lettura dell'input da console
        Scanner scanner = new Scanner(System.in);

        try {
            // Connessione al server di accesso
            try (Socket accessoSocket = new Socket("localhost", 54321);
                 PrintWriter accessoWriter = new PrintWriter(accessoSocket.getOutputStream(), true);
                 BufferedReader accessoReader = new BufferedReader(new InputStreamReader(accessoSocket.getInputStream()))) {

                // Richiesta del tipo di accesso
                System.out.println("Inserisci il tipo di accesso (amministratore/utente): ");
                String tipoAccesso = scanner.nextLine();
                accessoWriter.println(tipoAccesso);

                // Lettura e stampa della risposta dal server di accesso
                String rispostaServer = accessoReader.readLine();
                System.out.println("Risposta dal server: " + rispostaServer);

                // Connessione al server di simulazione
                try (Socket simulazioneSocket = new Socket("localhost", 8000);
                     PrintWriter simulazioneWriter = new PrintWriter(simulazioneSocket.getOutputStream(), true);
                     BufferedReader simulazioneReader = new BufferedReader(new InputStreamReader(simulazioneSocket.getInputStream()))) {

                    // Gestione del menu in base al tipo di accesso
                    if ("utente".equals(tipoAccesso)) {
                        menuUtente(scanner, simulazioneWriter, simulazioneReader);
                    } else if ("amministratore".equals(tipoAccesso)) {
                        menuAmministratore(scanner, simulazioneWriter);
                    }
                }
            }
        } catch (IOException e) {
            e.printStackTrace();
        } finally {
            // Chiusura dello scanner per liberare le risorse
            scanner.close();
        }
    }
    private static void menuUtente(Scanner scanner, PrintWriter simulazioneWriter, BufferedReader simulazioneReader) throws IOException {
        int scelta;
        while (true) {
            // Menu Utente
            System.out.println("Menu Utente:");
            System.out.println("1. Simulazione - Entrare al casello");
            System.out.println("2. Esci");
            System.out.print("Inserisci la tua scelta: ");

            // Leggi l'input come stringa
            String input = scanner.nextLine();

            // Converti l'input in un numero intero
            try {
                scelta = Integer.parseInt(input);
            } catch (NumberFormatException e) {
                System.out.println("Input non valido. Inserisci un numero corrispondente alla tua scelta.");
                continue;
            }

            switch (scelta) {
                case 1:
                    // Simulazione di ingresso al casello
                    System.out.print("Inserisci la targa: ");
                    String targa = scanner.nextLine();
                    simulazioneWriter.println("INGRESSO:" + targa);
                    System.out.println("Dati di ingresso inviati con successo");
                    break;
                case 2:
                    // Uscita dall'applicazione
                    return;
                default:
                    System.out.println("Scelta non valida. Riprova.");
                    break;
```

```java
    private static void menuAmministratore(Scanner scanner, PrintWriter simulazioneWriter, BufferedReader simulazioneReader, PrintWriter accessoWriter) throws IO
        while (true) {
            // Menu Amministratore
            System.out.println();
            System.out.println("Menu Amministratore:");
            System.out.println("1. Inserire Telepass per una macchina");
            System.out.println("2. Rimuovere Telepass per una macchina");
            System.out.println("3. Esci");
            System.out.print("Inserisci la tua scelta: ");
            int scelta = scanner.nextInt();
            scanner.nextLine(); // Consuma la linea rimanente

            switch (scelta) {
                case 1:
                    // Inserimento Telepass
                    System.out.print("Inserisci la targa del dispositivo Telepass da aggiungere: ");
                    String targaAdd = scanner.nextLine();
                    System.out.println();
                    System.out.println("Targa aggiunta al dispositivo telepass!");
                    simulazioneWriter.println("INSERISCI_TELEPASS:" + targaAdd);
                    break;
                case 2:
                    // Rimozione Telepass
                    System.out.print("Inserisci la targa del dispositivo Telepass da rimuovere: ");
                    String targaRemove = scanner.nextLine();
                    System.out.println();
                    System.out.println("Richiesta di rimozione inoltrata!");
                    simulazioneWriter.println("RIMUOVI_TELEPASS:" + targaRemove);
                    break;
                case 3:
                    // Uscita dal menù amministratore
                    return;
                default:
                    System.out.println("Scelta non valida. Riprova.");
                    break;
            }
```

3.1 ClientSimulazione

Connessione server di simulazione:

- Il client si connette al server di simulazione che ascolta sulla porta 8000.

```
// Connessione al server di simulazione sulla porta 8000
Socket socket = new Socket( host: "localhost",  port: 8000);
```

Inizializzazione degli strumenti di I/O:

- Il PrintWriter viene utilizzato per inviare dati al server.
- Lo Scanner viene utilizzato per leggere l'input dell'utente da console.

```
// Inizializzazione del PrintWriter per inviare dati al server
PrintWriter writer = new PrintWriter(socket.getOutputStream(),  autoFlush: true);

// Inizializzazione dello scanner per leggere l'input dell'utente da console
Scanner scanner = new Scanner(System.in);
```

Menù di simulazione per l’utente:

- Il menu visualizza le opzioni disponibili per l'utente, che possono scegliere di simulare l'ingresso al casello, l'uscita dal casello o di uscire dal programma.

```
// Visualizzazione del menu di simulazione
System.out.println("Menu di simulazione:");
System.out.println("1. Simulazione ingresso casello");
System.out.println("2. Simulazione uscita casello");
System.out.println("3. Esci");
```

Gestione della scelta dell'utente:

- Legge la scelta dell’utente e attua le azioni corrispondenti

```
// Leggi la scelta dell'utente
System.out.print("Inserisci la tua scelta: ");
int scelta = scanner.nextInt();
```

Simulazione di ingresso/uscita o chiusura della connessione:

- A seconda della scelta dell'utente, vengono chiamati i metodi simulaIngresso o simulaUscita per inviare i dati di simulazione appropriati al server.

```
                    // Simulazione di ingresso al casello
                    simulaIngresso(writer);
                    break;
                case 2:
                    // Simulazione di uscita dal casello
                    simulaUscita(writer);
                    break;
                case 3:
                    // Chiudi la connessione e esci dal ciclo
                    socket.close();
                    return;
                default:
                    System.out.println("Scelta non valida. Riprova.");
            }
        }
```

Metodi di simulazione:

- Questi metodi contengono la logica di simulazione specifica per l'ingresso e l'uscita. In questo caso, vengono inviati messaggi semplici al server di simulazione.

```
    private static void simulaIngresso(PrintWriter writer) {
        // Implementazione della logica di simulazione per l'ingresso al casello
        String datiSimulazione = "Ingresso al casello";
        // Invia i dati di simulazione al server
        inviaDatiDiSimulazione(writer, datiSimulazione);
    }

    1 usage
    private static void simulaUscita(PrintWriter writer) {
        // Implementazione della logica di simulazione per l'uscita dal casello
        String datiSimulazione = "Uscita dal casello";
        // Invia i dati di simulazione al server
        inviaDatiDiSimulazione(writer, datiSimulazione);
```

Metodi di invio dati simulazione:

- Questo metodo invia effettivamente i dati di simulazione utilizzando il metodo PrintWriter.

```
    private static void inviaDatiDiSimulazione(PrintWriter writer, String datiSimulazione) {
        // Invia i dati di simulazione al server di simulazione
        writer.println(datiSimulazione);
        System.out.println("Simulazione inviata con successo.");
```

3.3 ServerAccesso

Inizializzazione del ServerSocket:

- Il server si mette in ascolto sulla porta 54321 per accettare connessioni in ingresso dai client di accesso.

```
ServerSocket serverSocket = new ServerSocket( port: 54321);
```

Loop di accettazione delle connessioni:

- Il server entra in un loop infinito in cui accetta connessioni dai client. Quando una connessione viene accettata, viene creato un nuovo thread (ServerAccessoHandler) per gestire la richiesta del client. Per gestire le richieste concorrenti da parte di più client, il server crea un nuovo thread per ogni connessione accettata. Ogni thread sarà responsabile di comunicare con un singolo client attraverso il suo socket.

```
13          while (true) {
14              // Accettazione della connessione in ingresso da parte di un client
15              Socket clientSocket = serverSocket.accept();
16
17              // Gestione della connessione del client in un thread separato
18              Thread thread = new Thread(new ServerAccessoHandler(clientSocket));
19              thread.start(); // Avvio del thread per gestire la connessione del client
20          }
```

3.4 ServerAcessoHandler

Costruttore:

- Il costruttore inizializza l'istanza della classe con il socket del client. Questo socket è utilizzato per stabilire la connessione con il client.

```
10      public class ServerAccessoHandler implements Runnable {
            3 usages
11          private Socket clientSocket;
12
13          // Costruttore per inizializzare il ServerAccessoHandler con il Socket del client
            1 usage
14          public ServerAccessoHandler(Socket clientSocket) {
15              this.clientSocket = clientSocket;
16          }
```

Implementazione dell'interfaccia Runnable e gestione della comunicazione:

- La classe implementa l'interfaccia Runnable, consentendo l'esecuzione del suo codice in un thread separato quando viene creato un nuovo thread. All'interno del blocco try-with-resources, vengono inizializzati i flussi di input (reader) e output (writer) per la comunicazione con il client. Questo approccio assicura che i flussi vengano chiusi correttamente al termine dell'uso.

```
76          @Override
77          public void run() {
78              // ...
79          }
```

```
76      try (
77              BufferedReader reader = new BufferedReader(new InputStreamReader(clientSocket.getInputStream()));
78              PrintWriter writer = new PrintWriter(clientSocket.getOutputStream(), true)
79              ) {
80              // ...
81              } catch (SocketException e) {
82              // ...
83              } catch (IOException e) {
84              // ...
85              }
```

Lettura dell'input dal client:

- Viene creato un loop per leggere continuamente l'input proveniente dal client. Il metodo readLine() restituisce una stringa contenente i dati inviati dal client.

```
26                  String inputLine;
27                  // Loop per leggere continuamente l'input dal client
28                  while ((inputLine = reader.readLine()) != null) {
```

Gestione delle richieste:

- Il blocco if-else gestisce due tipi di input: dati di simulazione e richieste di accesso degli utenti.

```
31                  if (inputLine.startsWith("DATI_SIMULAZIONE:")) {
32                      // Estrae i dati di simulazione dall'input
33                      String datiSimulazione = inputLine.substring("DATI_SIMULAZIONE:".length());
34                      // Stampa i dati di simulazione ricevuti
35                      System.out.println("Ricevuti dati di simulazione: " + datiSimulazione);
36                      // Registra i dati di simulazione
37                      logDatiSimulazione(datiSimulazione);
38                      // Invia conferma al client che i dati di simulazione sono stati ricevuti
39                      writer.println("Conferma ricezione dati simulazione");
40                  } else {
41                      // Gestisce le richieste di accesso dell'utente
42                      switch (inputLine) {
43                          case "amministratore":
44                              // Se l'utente è un amministratore, conferma l'accesso
45                              System.out.println("Accesso amministratore autorizzato.");
46                              writer.println("Accesso amministratore confermato");
47                              break;
48                          case "utente":
49                              // Se l'utente è un utente, conferma l'accesso
50                              System.out.println("Accesso utente autorizzato.");
51                              writer.println("Accesso utente confermato");
52                              break;
53                          default:
54                              // Se il tipo di accesso non è valido, invia un messaggio di errore al client
55                              System.out.println("Tipo di accesso non valido.");
56                              writer.println("Tipo di accesso non valido");
57                              break;
```

Gestione dei dati di simulazione:

- Se il messaggio inizia con "DATI_SIMULAZIONE:", vengono estratti i dati di simulazione, registrati e viene inviata una conferma al client.

```
32                      String datiSimulazione = inputLine.substring("DATI_SIMULAZIONE:".length());
33                      // Stampa i dati di simulazione ricevuti
34                      System.out.println("Ricevuti dati di simulazione: " + datiSimulazione);
35                      // Registra i dati di simulazione
36                      logDatiSimulazione(datiSimulazione);
37                      // Invia conferma al client che i dati di simulazione sono stati ricevuti
38                      writer.println("Conferma ricezione dati simulazione");
```

Gestione delle richieste di accesso e delle eccezzioni:

- Se l'input non è un dato di simulazione, la classe gestisce le richieste di accesso degli utenti, distinguendo tra amministratori e utenti standard.

```
                    switch (inputLine) {
                        case "amministratore":
                            // Se l'utente è un amministratore, conferma l'accesso
                            System.out.println("Accesso amministratore autorizzato.");
                            writer.println("Accesso amministratore confermato");
                            break;
                        case "utente":
                            // Se l'utente è un utente, conferma l'accesso
                            System.out.println("Accesso utente autorizzato.");
                            writer.println("Accesso utente confermato");
                            break;
                        default:
                            // Se il tipo di accesso non è valido, invia un messaggio di errore al client
                            System.out.println("Tipo di accesso non valido.");
                            writer.println("Tipo di accesso non valido");
                            break;
                    }
                }
            }
        } catch (SocketException e) {
            // Gestisce eccezioni di connessione resettata
        } catch (IOException e) {
            // Gestisce altre eccezioni di I/O
            e.printStackTrace();
        }
    }
```

Registrazione e stampa dei dati

- Questo metodo stampa i dati di simulazione ricevuti su System.out.

```
    // Metodo per registrare i dati di simulazione
    1 usage
    private void logDatiSimulazione(String dati) {
        System.out.println("Log dati di simulazione: " + dati);
    }
}
```

3.5 ServerSimulazione

Mappa dei Dispositivi Telepass:

- Dichiarazione di una mappa per tenere traccia degli stati dei dispositivi Telepass. La chiave è la targa del veicolo associata al dispositivo, e il valore booleano indica se il dispositivo è presente (true) o assente (false).

```
12      private static Map<String, Boolean> telepassDevices = new HashMap<>();
```

Metodo per ottenere la Mappa dei Dispositivi Telepass:

- Metodo che restituisce la mappa dei dispositivi Telepass. Questo metodo fornisce un'interfaccia per accedere e modificare lo stato dei dispositivi Telepass.

```
15      public static Map<String, Boolean> getTelepassDevices() {
16          return telepassDevices;
```

Metodo Principale per Avviare il Server di Simulazione:

- Nel metodo principale, viene creato un Serversocket sulla porta 8000 per accettare connessioni dai client di simulazione. Il server rimane in un loop continuo, accettando ogni connessione in ingresso e gestendo la simulazione del client in un thread separato (ServerSimulazioneHandler). Il thread è avviato per gestire la simulazione del client

```
19  // Metodo principale per avviare il server di simulazione
20  public static void main(String[] args) {
21      try {
22          // Creazione del ServerSocket sulla porta 8000 per accettare connessioni dai client di simulazione
23          ServerSocket serverSocket = new ServerSocket( port: 8000);
24
25          while (true) {
26              // Accettazione della connessione in ingresso da parte di un client
27              Socket clientSocket = serverSocket.accept();
28
29              // Gestione della simulazione del client in un thread separato, passando anche la mappa dei dispositivi Telepass
30              Thread thread = new Thread(new ServerSimulazioneHandler(clientSocket, telepassDevices));
31              thread.start(); // Avvio del thread per gestire la simulazione del client
32          }
33      } catch (IOException e) {
34          e.printStackTrace(); // Gestione delle eccezioni di I/O
35      }
```

3.6 ServerSimulazioneHandler

Costruttore:

- Il costruttore inizializza il gestore con il socket del client, la mappa dei dispositivi Telepass ottenuta dal ServerSimulazione, e uno scanner per la lettura dell'input.

```
18      public ServerSimulazioneHandler(Socket clientSocket, Map<String, Boolean> telepassDevices) {
19          this.clientSocket = clientSocket;
20          this.scanner = new Scanner(System.in);
21          this.telepassDevices = ServerSimulazione.getTelepassDevices();
```

Metodo run():

- Il metodo run() gestisce il flusso di esecuzione del thread. Legge continuamente i dati di simulazione inviati dal client e, per ciascun evento, esegue la simulazione e invia i risultati al server centrale. Il loop continua fino a quando il client chiude la connessione.

```
22      @Override
23      public void run() {
24          try {
25              // Inizializzazione del BufferedReader per leggere i dati inviati dal client
26              BufferedReader reader = new BufferedReader(new InputStreamReader(clientSocket.getInputStream()));
27
28              String simulationData;
29              // Ciclo di lettura dei dati inviati dal client
30              while ((simulationData = reader.readLine()) != null) {
31                  // Esecuzione della simulazione e invio dei risultati al server centrale
32                  String risultatiSimulazione = eseguiSimulazione(simulationData);
33                  inviaDatiAlServerCentrale(risultatiSimulazione);
34              }
35
36          } catch (IOException e) {
37              // Gestione delle eccezioni durante la lettura dei dati o la chiusura del socket
38              System.err.println("Errore durante la lettura dei dati dal client: " + e.getMessage());
39          } finally {
40              try {
41                  // Chiusura del socket del client
42                  clientSocket.close();
43              } catch (IOException e) {
44                  System.err.println("Errore durante la chiusura del socket: " + e.getMessage());
45              }
46          }
47      }
```

Metodo eseguiSimulazione(String simulationData):

- Il metodo eseguiSimulazione interpreta e gestisce l'evento di simulazione. Estrae il tipo di evento e la targa dalla stringa di simulazione, quindi esegue le azioni corrispondenti, come simulare l'ingresso o l'uscita al casello, l'inserimento o la rimozione del dispositivo Telepass

```
                    // Rimozione effettiva del dispositivo Telepass
                    System.out.println("Dispositivo Telepass rimosso per la macchina con targa " + targa);
                    System.out.println("Rimozione del Telepass avvenuta.");
                    telepassDevices.put(targa, false);
                }
            } else {
                // Il dispositivo Telepass non esiste
                System.out.println("Il dispositivo Telepass " + targa + " non esiste.");
                inviaDatiAlServerCentrale( datiSimulazione: "Non esiste nessun dispositivo per la macchina con targa:" + targa);
                System.out.println("Rimozione del Telepass annullata.");
            }
        }
        // Composizione e restituzione dei risultati della simulazione
        return tipoEvento + ":" + targa;
    }
    // Metodo per eseguire la simulazione in base ai dati ricevuti
    1 usage
    private String eseguiSimulazione(String simulationData) {
        // Suddivisione dei dati in tipo di evento e targa
        String[] parts = simulationData.split( regex: ":");
        String tipoEvento = parts[0];
        String targa = parts[1];

        // Gestione dei diversi tipi di evento
        if (tipoEvento.equals("INGRESSO")) {
            // Simulazione di ingresso al casello
            System.out.println("Macchina con targa " + targa + " è entrata al casello.");
            boolean hasTelepass = telepassDevices.getOrDefault(targa, defaultValue: false);
            if (!hasTelepass) {
                // Avvio di un timer per simulare l'uscita dopo 5 secondi
                Timer timer = new Timer();
                timer.schedule(new TimerTask() {
                    @Override
                    public void run() {
                        System.out.println("Macchina con targa " + targa + " è uscita dal casello.");
                        inviaDatiAlServerCentrale( datiSimulazione: "USCITA:" + targa);
                    }
                }, delay: 5000);
            }
        } else if (tipoEvento.equals("INSERISCI_TELEPASS")) {
            // Simulazione di inserimento del dispositivo Telepass
            System.out.println("Dispositivo Telepass inserito per la macchina con targa " + targa);
            telepassDevices.put(targa, true);
        } else if (tipoEvento.equals("RIMUOVI_TELEPASS")) {
            // Simulazione di richiesta di rimozione del dispositivo Telepass
            System.out.println("Richiesta di rimozione del dispositivo Telepass per la macchina con targa " + targa);
            if (telepassDevices.containsKey(targa)) {
                boolean hasTelepass = telepassDevices.get(targa);
                if (hasTelepass) {
```

Metodo inviaDatiAlServerCentrale:

- Il metodo inviaDatiAlServerCentrale invia i dati di simulazione al server centrale (ServerAccesso) tramite una connessione socket. Questo meccanismo consente di comunicare i risultati della simulazione al sistema principale.

```
98      // Metodo per inviare i risultati della simulazione al server centrale
        3 usages
99      private void inviaDatiAlServerCentrale(String datiSimulazione) {
100         try (Socket socket = new Socket( host: "localhost", port: 54321);
101              PrintWriter out = new PrintWriter(socket.getOutputStream(), autoFlush: true)) {
102             // Invio dei dati al server centrale
103             out.println("DATI_SIMULAZIONE:" + datiSimulazione);
104             System.out.println("Dati inviati al ServerAccesso con successo.");
105         } catch (IOException e) {
106             // Gestione delle eccezioni durante l'invio dei dati al server centrale
107             System.err.println("Errore nell'invio dei dati al ServerAccesso: " + e.getMessage());
108         }
109     }
110 }
```

Capitolo 4
Istruzioni per l'esecuzione

4.1 Login

- Questa interazione consente all'utente di specificare il proprio ruolo, che determinerà le operazioni successive.

```
/Users/lorenzogaudio/Library/Java/JavaVirtualMachines/openjdk-21/Contents/Home/bin/java ...
Inserisci il tipo di accesso (amministratore/utente):
```

4.2 Menù utente

- Se si effettua il login come utente viene mostrato il menù dedicato con le due opzioni a disposizione dell'utente.

```
Inserisci il tipo di accesso (amministratore/utente):
utente
Risposta dal server: Accesso utente confermato
Menu Utente:
1. Simulazione - Entrare al casello
2. Esci
Inserisci la tua scelta:
```

4.3 Simulazione accesso al casello

- Se l'utente sceglie l'opzione "1. Simulazione – Entrare al casello" viene richiesto l'inserimento della targa del veicolo. Inserita la targa i dati vengono inviati al server Simulazione che li invia successivamente al server Accesso.

```
Menu Utente:
1. Simulazione - Entrare al casello
2. Esci
Inserisci la tua scelta: 1
Inserisci la targa: DY247MP
Dati di ingresso inviati con successo
```

- Visualizzazione del ServerSimulazione dopo l’inserimento della targa.

```
Macchina con targa DY247MP è entrata al casello.
Dati inviati al ServerAccesso con successo.
Macchina con targa DY247MP è uscita dal casello.
Dati inviati al ServerAccesso con successo.
```

- Visualizzazione del ServerAccesso dopo l’inserimento della targa.

```
Accesso utente autorizzato.
Ricevuti dati di simulazione: INGRESSO:DY247MP
Log dati di simulazione: INGRESSO:DY247MP
Ricevuti dati di simulazione: USCITA:DY247MP
Log dati di simulazione: USCITA:DY247MP
```

4.4 Menù amministratore

- Se si effettua il login come amministratore viene mostrato il menù dedicato con le due opzioni a disposizione dell’admin.

```
Inserisci il tipo di accesso (amministratore/utente):
amministratore
Risposta dal server: Accesso amministratore confermato
Menu Amministratore:
1. Inserire Telepass per una macchina
2. Rimuovere Telepass per una macchina
3. Esci
Inserisci la tua scelta:
```

4.5 Inserimento-Revoca Telepass

- Se l'admin sceglie di inserire o rimuovere un dispositivo Telepass dovrà inserire come richiesto la targa del veicolo, dopodiché la notifica di sistema avvertirà che l'operazione è andata a buon fine.

```
Menu Amministratore:
1. Inserire Telepass per una macchina
2. Rimuovere Telepass per una macchina
3. Esci
Inserisci la tua scelta: 1
Inserisci la targa del dispositivo Telepass da aggiungere: DY247MP
Targa aggiunta al dispositivo telepass!
```

- Visualizzazione del ServerSimulazione dopo l'inserimento della targa del nuovo dispositivo.

```
Dispositivo Telepass inserito per la macchina con targa DY247MP
Dati inviati al ServerAccesso con successo.
```

- Visualizzazione del ServerAccesso dopo l'inserimento della targa del nuovo dispositivo.

```
Accesso amministratore autorizzato.
Ricevuti dati di simulazione: INSERISCI_TELEPASS:DY247MP
Log dati di simulazione: INSERISCI_TELEPASS:DY247MP
```

4.6 Operazione "Esci"

- Per l'amministratore come per l'utente se viene selezionata dal menù l'opzione "Esci" il meno viene chiuso e il processo termina.

```
Menu Amministratore:
1. Inserire Telepass per una macchina
2. Rimuovere Telepass per una macchina
3. Esci
Inserisci la tua scelta: 3

Process finished with exit code 0
```