Implementazione JAVA

Traffic monitor

Ingegneria del software 2018/2019

Gruppo 6:
Mazzoleni Sara
Megler Federico
Paone Angelo

Il progetto è stato realizzato tramite l'utilizzo del programma *Eclipse,* il tool *WindowBuilder* per l'interfaccia grafica e *MySQLWorkbench* per la gestione del database.

Struttura del progetto

Il progetto e le classi java sono stati organizzati nei seguenti pacchetti:

- **client**: contiene tutte le classi per l'utilizzazione del programma;
- **server**: contiene tutte le classi relative ai gestori, tra cui quello per la connessione al database;
- **shared**: contiene tutte le interfacce relative ai vari gestori, e quindi i vari metodi che vengono passati tramite RMI.
- **centralina:** contiene tutte le classi che descrivono il comportamento delle centraline stradali;
- **app mobile:** contiene tutte le classi utili per lo scambio di informazioni tra appMobile e server, quali invio di segnalazioni e ricezione di notifiche.

Architettura RMI

Il programma è basato su un'architettura RMI, che rende possibile la comunicazione tra oggetti remoti (cioè non per forza localizzati sulla stessa macchina, come però sono nel nostro caso per motivi di praticità) attraverso l'invocazione di metodi tra gli oggetti stessi.

Il **server** istanzia nel suo main gli oggetti gestore (tutti singleton, ognuno implementa la relativa interfaccia), crea un registro sulla porta specificata, in cui verranno salvate le informazioni per la connessione per poi collegare i vari gestori istanziati al rispettivo nome.
Il **client** si collega al registro (tramite la stessa porta) e ricava i vari oggetti gestore effettuando un cast con la rispettiva interfaccia, nella quale sono dichiarati i metodi alla quale il client avrà accesso.
A questo punto il client lancia l'interfaccia utente.

Server

```
Naming.rebind("rmi://localhost:12345/DIRETTIVE",
        (IntAggiornamento)direttive);
Naming.rebind("rmi://localhost:12345/ACCESSO",
        (IntACC)accesso);
Naming.rebind("rmi://localhost:12345/SEGNALAZIONI",
        (IntSegnalazioni)segnalazione);
Naming.rebind("rmi://localhost:12345/MAPPA",
        (IntMappa)mappa);
Naming.rebind("rmi://localhost:12345/DATISTRADALI",
        (IntCentralina)datistradali);
```

```
GestoreDirettive direttive =
        GestoreDirettive.getIstance();
GestoreAccessoSistema accesso =
        GestoreAccessoSistema.getIstance();
GestoreSegnalazioni segnalazione =
        GestoreSegnalazioni.getIstance();
GestoreDatiStradali datistradali =
        GestoreDatiStradali.getIstance();
CreatoreMappa mappa =
        CreatoreMappa.getistance();
```

Client

```
IntAggiornamento server = (IntAggiornamento)
        Naming.lookup("rmi://localhost:12345/DIRETTIVE");
```

```
IntACC server = (IntACC)
        Naming.lookup("rmi://localhost:12345/ACCESSO");
```

```
IntSegnalazioni server = (IntSegnalazioni)
        Naming.lookup("rmi://localhost:12345/SEGNALAZIONI");
```

```
IntMappa server = (IntMappa)
        Naming.lookup("rmi://localhost:12345/MAPPA");
```

```
IntCentralina creatore = (IntCentralina)
        Naming.lookup("rmi://localhost:12345/DATISTRADALI");
```

Classe Mappa

Per la mappa abbiamo deciso di implementare un grafo basato su una struttura dati di tipo HashMap, la quale prende come chiavi le coordinate dei nodi (gli incroci stradali) e come valori i nodi stessi.

Ogni nodo ha quindi come attributi le coordinate sopracitate e una lista di oggetti strada, le quali sono gli archi che connettono due nodi, e sono descritte quindi dalla coordinate di inizio e di fine. Le strade sono inoltre caratterizzate da uno *stato* che ne indica le condizioni di traffico, da una *lunghezza*, pari alla distanza tra i due nodi, e da un *nome.*

La mappa viene costruita attraverso il metodo *add(),* il quale preleva le coordinate dei nodi, il nome e la lunghezza della via, l'id di un'eventuale centralina associata dal database, e costruisce il grafo stesso.

```
public void add (Nodo node1, Nodo node2, int distanza, String nomevia, String idcentralina) {
    if(!nodi.containsKey(node1.getCoordinate()))
        nodi.put(node1.getCoordinate(), node1);
    //controllo se è già presente la chiave nel caso l'aggiungo nella hashmap
    if(!nodi.containsKey(node2.getCoordinate()))
        nodi.put(node2.getCoordinate(), node2);
      //creo un nuovo oggetto strada e lo aggiungo alla lista dei nodi
    Strada tratto=new Strada(node1,node2,nomevia,distanza,idcentralina);
        nodi.get(node1.getCoordinate()).getLista().add(tratto);
        nodi.get(node2.getCoordinate()).getLista().add(tratto);
}
```

Per l'identificazione di una posizione di un'app, ad esempio, si ricorre quindi al metodo *riferimento()*, il quale, prese le coordinate della posizione interessa e quelle di un nodo qualsiasi ne calcola la distanza per indentificare il nodo più vicino. Questo metodo è utilizzato anche dal metodo *action()* per identificare il luogo di situazioni di traffico vicine all'utente.

```
public String action(String coordinate) {
  String rif = riferimento(coordinate);
  HashMap<String,String> strade =new HashMap<String,String>();
  int distanza=0;
  controllo (strade,distanza,rif) ;
  Iterator<String> i=strade.keySet().iterator();
  String l[];
  String s = new String();
  while(i.hasNext()) {
      String tep=i.next();
      l= strade.get(tep).split("\\ ");
      s += "Traffico in "+tep+" con tipo traffico "+l[0]
              +" a circa "+l[1]+" metri dal prossimo incrocio.\n\n";
  }
  return s;
}
```

Gestori

Creatore Mappa

La classe CreatoreMappa implementa l'interfaccia *IntMappa.*

Tramite il metodo *generaLinee()* va a prendere le coordinate relative agli oggetti strada, la situazione del traffico e l'eventuale presenza di centraline su di esse e le fornisce al client.

```
@Override
public Set<String> generaLinee() throws RemoteException, SQLException{
    Set<String> linee = new HashSet<String>();
    Mappa mappa = Mappa.getIstance();
    Iterator<String> it = mappa.getNodi().keySet().iterator();
    Iterator<Strada> i;
    Nodo n;
    Strada s;
    while(it.hasNext()) {
        n = mappa.getNodi().get(it.next());
        i = n.getLista().iterator();
        while(i.hasNext()) {
            s = i.next();
            String stringa;
            if(s.getId_centralina() != null)
                stringa = new String(s.getNode1().getCoordinate()+","+s.getNode2().getCoordinate()
                        +","+s.getStato()+","+s.getVia()+","+"1");
            else
                stringa = new String(s.getNode1().getCoordinate()+","+s.getNode2().getCoordinate()
                        +","+s.getStato()+","+s.getVia()+","+"0");
            linee.add(stringa);
        }
    }
    return linee;
}
```

Gestore AccessoDatabase

Per salvare i dati abbiamo creato un database MySQL, al quale accediamo tramite la classe *GestoreAccessoDatabase*, la quale contiene i parametri per la connessione (URL, USER e PASS).

Ogni metodo che deve interagire con il database richiamerà la *getConnection()*, che restituisce la connessione al DB, e una volta terminate le sue funzionalità chiude la connessione, tramite *close().*

Per l'invio della query viene creato uno statement tramite *createStatement()* per poi eseguire statement.executeUpdate(query).

Ad esempio:

```
public boolean modificaPassword(String nickname, String password) {
    try {
        Connection conn = DriverManager.getConnection(URL,USER,PASS);
        Statement statement = conn.createStatement();
        String query = new String("UPDATE utente SET password = '" + password
                + "' WHERE nickname = '" + nickname + "';");
        statement.executeUpdate(query);
        conn.close();
        return true;
    }
    catch(Exception e) {
        e.printStackTrace();
        return false;
    }
}
```

Gestore AccessoSistema

La classe GestoreAccessoSistema implementa l'interfaccia *IntACC.*
Si occupa di prelevare i dati inseriti al momento del **login** e del **signup,** azioni rese disponibili a *PaginaUtente* del client, e di confrontarli con quelli contenuti nel database.

In particolare, al momento della registrazione, si occupa di verificare che le credenziali inserite siano valide, cioè che l'utente non sia già esistente e che le due password corrispondano, oltre che ad essere della giusta lunghezza(8-16); per poi inviarle al database.

Al momento dell'accesso, si occupa di verificare che il nomeutente e la password corrispondano a quelle inserite nel database.

Questa classe permette inoltre di modificare la propria password e di cancellare il proprio profilo.

Metodi d'esempio:

```
@Override
public void aggiungiUtente(String nome, String cognome, String nickname, String password, int isAdmin)
        throws RemoteException {
    // TODO Auto-generated method stub
    GestoreAccessoDatabase.getIstance().setCredenziali(nome, cognome, nickname, password, isAdmin);
}

@Override
public int esisteUtente(String nickname) throws RemoteException {
    // TODO Auto-generated method stub
    return GestoreAccessoDatabase.getIstance().controlloNick(nickname);
}
@Override
public boolean credValide(String nickname, String password) throws RemoteException {
    String[] vettore;
    vettore = GestoreAccessoDatabase.getIstance().getCredenziali(nickname);
    if(vettore == null) {
        System.out.println(nickname + password);
        return false;
    }
    if(vettore[3].equals(password)) {
        return true;
    }
    return false;
}
```

Gestore DatiStradali

La classe GestoreDatiStradali implementa l'interfaccia *IntCentralina.*
Al momento dell'avvio, si occupa di prelevare tramite *ottieniCentraline()* le informazioni relative alle centraline stradali **(idcentralina; limite di velocità; soglia critica veicoli; inizio, fine e nome della** via in cui si trova) da cui andrà a prelevare i dati stradali da esse raccolti, tramite *salvaDatiStradali(),* per inviarli al database.

```
@Override
public void salvaDatiStradali(int id_centralina, String inizio, String fine, int velocita_media, int n_veicoli)
        throws RemoteException {
    DateTimeFormatter format = DateTimeFormatter.ofPattern("yyyy/MM/dd HH:mm:ss");
    LocalDateTime ora = LocalDateTime.now();
    ResultSet ris = GestoreAccessoDatabase.getIstance().ottieniDatiCentralina(id_centralina);
    int limite_vel = 0;
    int lim_auto = 0;
    char tipo;
    try {
        while(ris.next()) {
            limite_vel = ris.getInt("limite_velocita");
            lim_auto = ris.getInt("limite_auto");
        }
        tipo = computaTraffico(n_veicoli, velocita_media,limite_vel, lim_auto);
        GestoreAccessoDatabase.getIstance().salvaDatiCentralina(id_centralina, inizio,
                fine, velocita_media, n_veicoli, format.format(ora),
                tipo);
        Mappa.getIstance().setStatoVia(inizio, fine, tipo);
    } catch (SQLException e1) {

        e1.printStackTrace();
    }
}
```

Questi dati verranno qui analizzati per stabilire a quali condizione di traffico essi corrispondono (**traffico intenso, moderato o regolare)** tramite *computaTraffico*().

```
public char computaTraffico(int veicoli_rilev, int vel_rilev, int limite_velocita, int n_limite_veicoli) {
    if(n_limite_veicoli/2 < veicoli_rilev) {
        if(limite_velocita/2 > vel_rilev) {
            return 'I';
        }
        else if(limite_velocita*0.8 > vel_rilev) {
            return 'M';
        }
    }
    return 'R';
}
```

Gestore Direttive

La classe GestoreDirettive implementa l'interfaccia *IntDirettive.*
Essa si occupa di gestire gli aggiornamenti sulle centraline inseriti dall'admin, in particolare è possibile:
-creare una centralina,
-modificare i parametri (limite velocità e veicoli) di una centralina,
-rimuovere una centralina.

In particolare abbiamo due metodi *creaCentralina(),* il primo, che riceve come parametro anche l'idcentralina, si occupa di mandare le informazioni sulla nuova centralina al database. Il secondo si occupa invece di ricevere queste informazioni e di avviare per la nuova centralina un nuovo thread.

```
public boolean creaCentralina( String nome_via,String inizio, String fine, int n_auto_limite, int velocita_limite)
        throws MalformedURLException, RemoteException, NotBoundException {
    IntDirettiva server = (IntDirettiva) Naming.lookup("rmi://localhost:12344/CENTRALINA");
    Nodo node=Mappa.getIstance().getNodi().get(inizio);
    Iterator<Strada> it = node.getLista().iterator();
    Strada tratto;
    int idcentralina;
    while(it.hasNext()) {
        tratto=it.next();
        if(tratto.getNode2().getCoordinate().equals(fine)) {
            if(tratto.getId_centralina()!=null)
                return false;
            break;
        }
    }
    try {
        idcentralina = GestoreAccessoDatabase.getIstance().ottieniMaxIDCentralina();
        server.nuovaCentralina(nome_via,inizio,fine,idcentralina,n_auto_limite,velocita_limite);
        GestoreAccessoDatabase.getIstance().salvaNuovaCentralina(idcentralina, inizio, fine, velocita_limite, n_auto_limite,
                nome_via);
        GestoreAccessoDatabase.getIstance().aggiornaStrada(idcentralina, inizio, fine);
           it = node.getLista().iterator();
            while(it.hasNext()) {
                tratto=it.next();
                if(tratto.getNode2().getCoordinate().equals(fine)) {
                    tratto.setId_centralina(Integer.toString(idcentralina));
                }
            }
        return true;
    }
```

```
    public void creaCentralina(int idCentralina, String nome_via,String inizio, String fine, int n_auto_limite,
            int velocita_limite) throws MalformedURLException, RemoteException, NotBoundException {
        IntDirettiva server = (IntDirettiva) Naming.lookup("rmi://localhost:12344/CENTRALINA");
        try {
            server.nuovaCentralina(nome_via,inizio,fine,idCentralina,n_auto_limite,velocita_limite);
        }
        catch(RemoteException e) {
            e.printStackTrace();
        }
    }
}
```

Gestore Segnalazioni

La classe GestoreSegnalazioni implementa l'interfaccia *IntSegnalazioni.*
Tramite il metodo *riceviNodoVicino(),* che riceve in ingresso la posizione fornitagli dall'AppMobile, la classe è in grado di identificare il nodo (incrocio stradale) a cui l'utente si trova più vicino.

Fatto ciò, l'utente è in grado di inviare una segnalazione, tramite *inviaSegnalazione(),* a cui accede tramite l'interfaccia dell'AppMobile, dove potrà scegliere tra tre diverse opzioni di traffico.
La scelta e la posizione ad essa associata verranno quindi qui mandate al database.

```
public boolean inviaSegnalazione(char tipo_traffico, String posizione) throws RemoteException{
    Mappa.getIstance().segnalazione(posizione, tipo_traffico);
    return true;
}
```

Il metodo *notifica(),* richiama il metodo *action()* di mappa, e ricevuta a posizione dell'app, si occupa di inviarle eventuali notifiche su situazioni di traffico particolari.

```
public String notifica(String posizione) {
    return Mappa.getIstance().action(posizione);
}
```

Testing

Abbiamo effettuato alcuni test con JUnit per provare il funzionamento del programma.

```
class GestoreAccessoSistemaTest {

    @Test
    void testEsisteUtente() throws RemoteException {
        GestoreAccessoSistema gestore = GestoreAccessoSistema.getIstance();
        assertEquals(1,gestore.esisteUtente("fedemeg"));
        assertEquals(0, gestore.esisteUtente("nonesiste"));
    }

    @Test
    void testCredValide() throws RemoteException {
        GestoreAccessoSistema gestore = GestoreAccessoSistema.getIstance();
        assertEquals(true,gestore.credValide("fedemeg", "federico"));
        assertEquals(false,gestore.credValide("fedemeg", "sbagliata"));
    }

}
```

I test effettuati sono:

- **GestoreAccessoSistemaTest**: testa la corretta elaborazione delle credenziali d'accesso al sistema, in particolare se l'utente inserito esiste realmente e se la password è corretta.
- **GestoreAccessoDatabaseTest:** testa l'effettiva modifica della password, la corretta ricezione delle credenziali accessibili dal profilo, l'effettiva modifica degli attributi propri di una centralina e la corretta identificazione degli id della centralina.
- **TestMappa**: testa la corretta identificazione del nodo più vicino alle coordinate inserite.
- **GestoreDatiStradaliTest**: testa l'identificazione della corretta situazione di traffico (intenso, moderato, regolare) a seconda dei dati stradali inseriti.