|  |
| --- |
| **ENVIRONMENTAL CAR** |
| DTS Project |
|  |
|  |
|  |
| **Jose María González Gamito 45821905-V jose.gg88@gmail.com** |
| **Itziar Arriortua Peña 72407277-D i.arriortua@gmail.com** |
|  |

*Index*

* **Description** of the implementation strategies, problems and Solutions…*Pages 3-4*
* State transition diagram **requestServer** *...Page 5*

* State transition diagram **requestGridServer** *...Page 6*
* **User’s manual**
* Set-up process...Page 7
* Screenshots...Page 8-11
* Communication log...Page 12
* **Code** *...Page 13*

*Description of the implementation strategies, problems and Solutions*

* *Structure:*

The project has been divided into different packages that contain all the classes and resources needed by the application.

The packages we have are: domain, GUI, and DB inside of src Package, and data package that contains more packages inside. We are just going to have a look to the different parts.

* Domain

This Packaged contains all the classes related to the Business layer of our application, it means, the main functionality.

This Project consists on client/Server architecture, so we decided to divide this actual Packaged into other three:

* + Server: contains the classes of the daemon servers and the request servers
  + Util: it contains the class “SocketManager” used in the session’s lab, which are going to help us to communicate the client with the servers.
* DB

In this Packaged the only one class we can find is the database manager that is going to deal with every aspect related to the database.

* GUI:

It contains all the classes of the windows of the application, which are in fact the client of it.

* *Implementation*

Talking about the implementation, just begin with the servers:

The application has been implemented with two servers. The aim of the first server, called RequestServer, is to response to the request of the clients, it means, it’s going to provide answers to all the commands coming from the clients. On the other hand, the purpose of the second Server, GridRequestServer, is to provide only one service, although it has four commands.

The service this Server is going to provide is to obtain GPS coordinates of a picture that has been taken in the client, when the GPS system is switched off.

In addition to this two servers, other two has been implemented too, the daemon servers, whose purpose is to create different child servers, which are going to offer service to the clients.

The way they have been implemented is simple; they follow a state transition diagram (which will be represented in next section). Every time the server receives a request from the client, analyses the command, splitting the request line into different strings, to do an action or another. So, the program follows this abstract state transition diagram ,in order to allow only the correct request, so the user cannot for example try to switch on the gps while he’s seen a picture.

Furthermore, it has been said that there could be as many clients as we want, that’s why, a Server daemon has been implemented for each Server which attends the requests of the clients. This way, when a client ask for a service, the daemon Server catch that petition, but instead of process itself the request, it delegates in a “request Server” which is going to provide a response to the client. Summing up the aim of the daemon servers is to create “subservers” in order to attend the clients’ petitions.

According to the data management, as almost every application, ours works with data which could be stored in a text file, in an xml file or just in a database. We have decided to use an Access database, to stored and manipulate all the information referred to our application: vehicles, sensors, measurements… and all the relation among them.

This database is going to be used exclusive by the servers, clients are never going to know that there’s a database. It has been decided this way, because if we talk about a client/Server architecture, it doesn’t have sense that the client has access to the information it’s asking for to the Server. However, in our application we have followed this approach in part just to simplify things. It will be explained later in problems section.

Moving to the field of the database implementation, we decided to create a class which deals with all aspects of the information stored in it. This class is called DBM (Data Base Management). It has typical methods such as connect (constructor) and disconnect, and all the methods needed by the Server to obtain information.

To sum up, the GUI is the one who interacts with all these elements, providing a user-friendly interface to the user, in order to make easier the management of the car. All the windows has a reference to the Socket Manager, and they send the request and provides a proper feedback to the user. Of course, all the data shown in the GUI is not real, and its storage in our database, but also all the images in a folder of the project. That means that the picture shown in the GET\_PIC command is always the same, and so is the coords of the GPS.

However, we had a huge problem trying to show the picture in the GUI. The transmission of the image between the server and the GUI is kind of tricky. Our problem was that the Socket Manager got stuck by sending the bytes of the image, finally we figure out that it was caused by a loose of bytes, so the problem was the “while” which kept going until the data sent is the same as the size of the file, a condition which was never full-filed. Finally, we managed to do it by sending all the bytes on a time, instead of sending it by parts(as we supposed to do it according to the documentation found on the internet)

*State transition diagram requestServer*

CONNECT ip

[ip OK]

PASS password

[password OK]

USER username

[username OK]

USER username

[usernameWrong]

USER

[no username]

PASS

[No password]

PASS password

[passwordwrong]

CONNECT

[no ip]

CONNECT ip

Iipwrong]

LISTSENSOR

GPSON, GPSOFF

GET\_PIC

HISTORYLOG sensorId

ON sensorId

OFF sensroId

GET\_CURVALUEsensorId

RETURN

GET\_LOC

QUIT

*State transition diagram requestGridServer*

PASS password

[password OK]

USER username

[username OK]

USER username

[usernameWrong]

USER

[no username]

PASS

[No password]

PASS password

[passwordwrong]

GET\_COOR

QUIT

*User’s manual*

* Set-up process

In this section we will explain the steps any user has to follow, in order to deploy this application using the integrated development environment (IDE) “Eclipse”.

As we have mention in a previous section, we have two server and a client that can be launch as many times as we want to create different clients, so if we want to run the application, we have to launch first the servers, and finally the client.

These are the step to be taken:

1. Launch the GridServer class. This class can be found in the package: src/domain/server/GridServer.java
2. Launch the other server, CarServer which is in the route: src/domain/server/CarServer.java
3. Launch the client class. This class is in the GUI package, because in deed the client is an interface. The class that we must run is ClientGUI and it’s located in the following path: src/GUI/ClientGUI.java
4. As we use an access database we have to register it in the ODBC of windows. The database is in the path: data/data base/CarDB.mdb

* Screenshots

**Login Window**. This window certificates your user and password, and connects you with the car with matches with the IP address given. If the connection is correct, it leads you to:

****

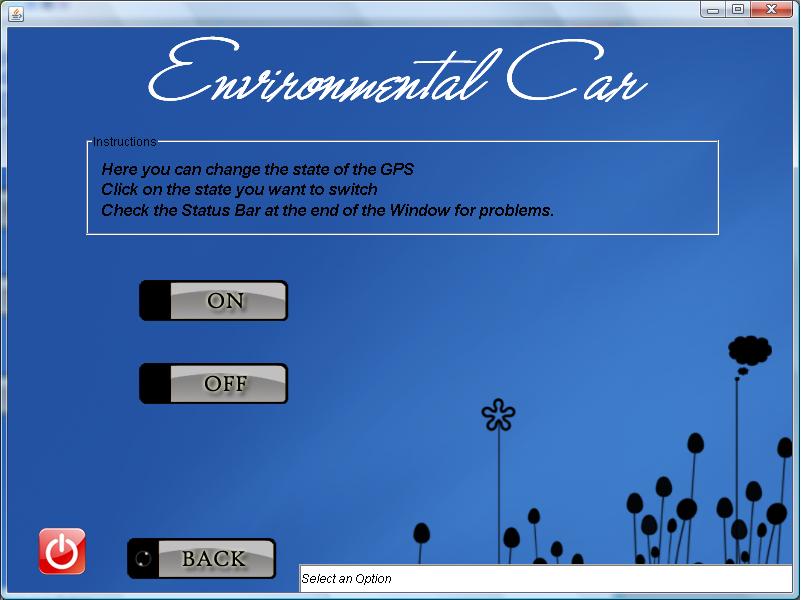
**Menu Window:** This window lets the user choose in between these three options. (1-GetPic; 2- Sensors;3-GPS)

**1-Get Pic Window**: Here you can take a picture, save it, and also get the cords of the picture.

**2-Sensor Window**. Here you can manage all the aspects concerning to the sensors. This is the initial window, that shows all the sensors, their description and state.

 If you click on “Cur Value” Button, you can get all the info associated to the selected item.

And also, y you click on ON/OFF Buttons, you can switch ON/OFF the selected item.

**3- GPS Window**. Here you can switch the GPS ON or OFF just clicking in the Button you wish.

* Communication log (client-requestServer-requestGridServer)
* **Client:** USER Itzi
  + **Server:** 201 OK Welcome Itzi
* **Client:** PASS 12345
  + **Server:** 202 OK Welcome to the system
* **Client:** CONNECT 127.0.0.1
  + **Server:** 214 OK Connection with the server established
* **Client:** LISTSENSOR
  + **Server:** 112 OK start of sensor list… 212 OK end of sensor list
* **Client:** ON 2
  + **Server:** 418 ERROR Sensor already activated
* **Client:** HISTORYLOG 3
  + **Server:** 113 OK Start of measurement list Ups, there aren't measurements for this sensor
* **Client:** GET\_CURVALUE 2
  + **Server:** 114 OK 10/4/2010; 19:49:51; 14º 34' 56.89''- 45º 56' 34.89''; 76%
* **Client:** RETURN
  + **Server:** Coming back to the main menu...
* **Client:** GPSOFF
  + **Server:** 420 ERROR GPS already deactivated
* **Client:** GET\_PIC
  + **Server:** 206 OK Loading image [96997 bytes]… photo transmitted
* **Client:** GET\_LOC
  + **Server:** USER Itzi
    - **Server2:** 201 OK Welcome Itzi
  + **Server:** PASS 12345
    - **Server2:** 202 OK Welcome to the system
  + **Server:** GET\_COOR 1
    - **Server2:** 114 Ok 14º 34' 56.89''- 45º 56' 34.89''
  + **Server:** QUIT
    - **Server2:** 222 Ok Bye
  + **Server:** 114 Ok 14º 34' 56.89''- 45º 56' 34.89''
* **Client:** QUIT
  + **Server:** 208 OK Bye

**Black:** requests from client to requestServer

**Red:** answer from requestServer to client

**Blue:** request from requestServer to requestGridServer

**Green:** answer from requestGridServer to requestServer

DATA BASE MANAGEMENT:

package DB;

import java.sql.Connection;

import java.sql.DriverManager;

import java.sql.ResultSet;

import java.sql.SQLException;

import java.sql.Statement;

import java.util.Vector;

public class DBM {

private Connection con;

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*//

public DBM()

{

//Driver load and Connection with the DataBase

try

{

Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");

con = DriverManager.getConnection("jdbc:odbc:CarDB","", "");

}

catch(Exception e)

{

System.out.println("Cannot load the JDBC-ODBC Driver");

}

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*//

public void disconnectDB()

{

try

{

this.con.close();

}

catch(SQLException se)

{

System.out.println("Cannot disconnect the Data base");

}

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*//

public boolean checkUser(String user)

{

Statement sentSQL = null;

String query = null;

ResultSet result = null;

boolean verifycation=false;

//Define the query

query = "SELECT \* FROM USER WHERE USER = '" + user + "'";

//Check user in the dataBase

try

{

sentSQL = con.createStatement();

result = sentSQL.executeQuery(query);

verifycation = result.next();

sentSQL.close();

}

catch (SQLException e)

{

// Auto-generated catch block

e.printStackTrace();

}

return verifycation;

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*//

public boolean checkPass(String user, String pass)

{

Statement sentSQL = null;

String query = null;

ResultSet rs = null;

boolean verifycation=false;

//Define the query

query = "SELECT \* FROM USER WHERE PASS = '" + pass + "' AND USER = '" +user + "'";

//Check user in the dataBase

try

{

sentSQL = con.createStatement();

rs = sentSQL.executeQuery(query);

verifycation=rs.next();

sentSQL.close();

}

catch (SQLException e)

{

// Auto-generated catch block

e.printStackTrace();

}

return verifycation;

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*//

public boolean checkSensorID(String sensorID)

{

//verifica si un sensorID existe, sin importarnos el coche en el que este alojado.

Statement sentSQL = null;

String query = null;

ResultSet rs = null;

boolean verifycation=false;

//Define the query

query = "SELECT \* FROM SENSOR WHERE SENSORID = '" + sensorID + "'";

//Check state in the dataBase

try

{

sentSQL = con.createStatement();

rs = sentSQL.executeQuery(query);

verifycation = rs.next();

sentSQL.close();

}

catch (SQLException e)

{

// Auto-generated catch block

e.printStackTrace();

}

return verifycation;

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*//

public boolean verifySensor(String sensorID, String ip)

{

//comprueba si un sensor id pretenece al coche en el que estamos conectados,

//para ello, primero verificamos que el sensorID exista sin importarnos a que coche pertenece.

//si existe, comprobamos que pertenece al coche al que estamos conectados

boolean result = false;

Statement sentSQL = null;

String query = null;

ResultSet rs = null;

try

{

boolean sensorIDExists = checkSensorID(sensorID);

if(sensorIDExists == false)

result = false;

else

{

sentSQL = con.createStatement();

query = "SELECT IP FROM VEHICLE WHERE VEHICLEID = (SELECT VEHICLEID FROM SENSOR WHERE SENSORID = '" + sensorID + "')";

rs = sentSQL.executeQuery(query);

rs.next();

if(ip.equals(rs.getString(1)))

result = true;

sentSQL.close();

}

}

catch (SQLException e)

{

// Auto-generated catch block

e.printStackTrace();

}

return result;

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*//

public void switchONSensor(String sensorID)

{

Statement sentSQL = null;

String query = null;

//Define the query

query = "UPDATE SENSOR SET STATE = '" + "ON" + "' WHERE SENSORID ='" + sensorID + "'";

try

{

sentSQL = con.createStatement();

sentSQL.executeUpdate(query);

sentSQL.close();

}

catch (SQLException e)

{

// Auto-generated catch block

e.printStackTrace();

}

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*//

public void switchOFFSensor(String sensorID)

{

Statement sentSQL = null;

String query = null;

//Define the query

query = "UPDATE SENSOR SET STATE = '" + "OFF" + "' WHERE SENSORID ='" + sensorID + "'";

try

{

sentSQL = con.createStatement();

sentSQL.executeUpdate(query);

sentSQL.close();

}

catch (SQLException e)

{

// Auto-generated catch block

e.printStackTrace();

}

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*//

public Vector<String> getListSensors(String ip)

{

Vector<String> listSensors=new Vector<String>();

Statement sentSQL = null;

String query = null;

ResultSet rs = null;

String id=null;

//Obtain the ID from IP

id=getId(ip);

//Define the query

query = "SELECT \* FROM SENSOR WHERE VEHICLEID='" + id + "'";

//Check user in the dataBase

try

{

sentSQL = con.createStatement();

rs = sentSQL.executeQuery(query);

//Convert resultSet into a Vector

while(rs.next())

{

listSensors.add(rs.getString("SENSORID") + ";;" + rs.getString("DESCRIPTION") + ";;" + rs.getString("STATE"));

}

sentSQL.close();

}

catch (SQLException e)

{

// Auto-generated catch block

e.printStackTrace();

}

return listSensors;

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*//

public Vector<String> getListMeasurements(String id)

{

Vector<String> listMeasurements=new Vector<String>();

Statement sentSQL = null;

String query = null;

ResultSet rs = null;

//Define the query

query = "SELECT \* FROM MEASUREMENT WHERE SENSORID='" + id + "'";

//Check user in the dataBase

try

{

sentSQL = con.createStatement();

rs = sentSQL.executeQuery(query);

//Convert resultSet into a Vector

while(rs.next())

{

listMeasurements.add(rs.getString("DATE") + ";" + rs.getString("TIME") + ";" + getCoord(rs.getString("COORD")) + ";" + rs.getString("VALUE"));

}

sentSQL.close();

}

catch (SQLException e)

{

// Auto-generated catch block

e.printStackTrace();

}

return listMeasurements;

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*//

private String getId(String ip)

{

Statement sentSQL = null;

String query = null;

ResultSet rs = null;

String id=null;

//Define the query

query = "SELECT \* FROM VEHICLE WHERE IP = '" + ip + "'";

//Check user in the dataBase

try

{

sentSQL = con.createStatement();

rs = sentSQL.executeQuery(query);

rs.next();

id=rs.getString(1);

sentSQL.close();

}

catch (SQLException e)

{

// Auto-generated catch block

e.printStackTrace();

}

return id;

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*//

public boolean getSensorState(String id)

{

String state;

boolean state2 = false;

Statement sentSQL = null;

String query = null;

ResultSet rs = null;

//Define the query

query = "SELECT STATE FROM SENSOR WHERE SENSORID='"+id+"'";

//Check user in the dataBase

try

{

sentSQL = con.createStatement();

rs = sentSQL.executeQuery(query);

rs.next();

state= rs.getString("STATE");

if(state.equals("ON"))

state2 = true;

else state2 = false;

sentSQL.close();

}

catch (SQLException e)

{

// Auto-generated catch block

e.printStackTrace();

}

return state2;

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*//

public boolean getGPSState(String ip)

{

String state = null;

boolean result = false;

Statement sentSQL = null;

String query = null;

ResultSet rs = null;

String id=null;

//Obtain the ID from IP

id=getId(ip);

//Define the query

query = "SELECT STATE FROM GPS WHERE VEHICLEID='"+id+"'";

//Check user in the dataBase

try

{

sentSQL = con.createStatement();

rs = sentSQL.executeQuery(query);

rs.next();

state = rs.getString(1);

if(state.equals("ON"))

result = true;

else result = false;

sentSQL.close();

}

catch (SQLException e)

{

// Auto-generated catch block

e.printStackTrace();

}

return result;

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*//

public void switchONGPS(String ip)

{

Statement sentSQL = null;

String query = null;

try

{

sentSQL = con.createStatement();

String vehicleID = getId(ip);

query = "UPDATE GPS SET STATE = '" + "ON" + "' WHERE VEHICLEID ='" + vehicleID + "'";

sentSQL.executeUpdate(query);

sentSQL.close();

}

catch (SQLException e)

{

// Auto-generated catch block

e.printStackTrace();

}

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*//

public void switchOFFGPS(String ip)

{

Statement sentSQL = null;

String query = null;

try

{

sentSQL = con.createStatement();

String vehicleID = getId(ip);

query = "UPDATE GPS SET STATE = '" + "OFF" + "' WHERE VEHICLEID ='" + vehicleID + "'";

sentSQL.executeUpdate(query);

sentSQL.close();

}

catch (SQLException e)

{

// Auto-generated catch block

e.printStackTrace();

}

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*//

public String getSensorValue(String sensorID)

{

String value = null;

Statement sentSQL = null;

String query = null;

ResultSet rs = null;

try

{

sentSQL = con.createStatement();

query = "select curValue from sensor where sensorID = '" + sensorID + "'";

rs = sentSQL.executeQuery(query);

rs.next();

value = rs.getString(1);

sentSQL.close();

}

catch (SQLException e)

{

// Auto-generated catch block

e.printStackTrace();

}

return value;

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*//

public String getSensorCoord(String sensorID)

{

String coord = null;

Statement sentSQL = null;

String query = null;

ResultSet rs = null;

try

{

sentSQL = con.createStatement();

query = "SELECT VAL FROM COORD WHERE COORD =(SELECT COORD FROM GPS WHERE GPSID = (SELECT GPSID FROM VEHICLE WHERE VEHICLEID = (SELECT VEHICLEID FROM SENSOR WHERE SENSORID = '" + sensorID + "')))";

rs = sentSQL.executeQuery(query);

rs.next();

coord = rs.getString(1);

sentSQL.close();

}

catch (SQLException e)

{

// Auto-generated catch block

e.printStackTrace();

}

return coord;

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*//

public String getCoord(String coorID)

{

Statement sentSQL = null;

String query = null;

ResultSet rs = null;

String coord = null;

try

{

sentSQL = con.createStatement();

query = "select val from coord where coord = '" + coorID + "'";

rs = sentSQL.executeQuery(query);

rs.next();

coord = rs.getString(1);

sentSQL.close();

}

catch (SQLException e)

{

// Auto-generated catch block

e.printStackTrace();

}

return coord;

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*//

public String getActualGrid(String ip)

{

Statement sentSQL = null;

String query = null;

ResultSet rs = null;

String grid = null;

try

{

sentSQL = con.createStatement();

query = "SELECT ACTUALGRID FROM VEHICLE WHERE IP = '" + ip + "'";

rs = sentSQL.executeQuery(query);

rs.next();

grid = rs.getString(1);

sentSQL.close();

}

catch (SQLException e)

{

// Auto-generated catch block

e.printStackTrace();

}

return grid;

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*//

public boolean verifyCell(String cellID)

{

Statement sentSQL = null;

String query = null;

ResultSet rs = null;

boolean result = false;

try

{

sentSQL = con.createStatement();

query = "SELECT \* FROM GRID WHERE GRIDID = '" + cellID + "'";

rs = sentSQL.executeQuery(query);

result = rs.next();

sentSQL.close();

}

catch (SQLException e)

{

// Auto-generated catch block

e.printStackTrace();

}

return result;

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*//

public String getGPSCoordCell(String cellID)

{

Statement sentSQL = null;

String query = null;

ResultSet rs = null;

String coord = null;

try

{

sentSQL = con.createStatement();

query = "SELECT COORD FROM GRID WHERE GRIDID = '" + cellID + "'";

rs = sentSQL.executeQuery(query);

rs.next();

coord = rs.getString(1);

coord = getCoord(coord);

sentSQL.close();

}

catch (SQLException e)

{

// Auto-generated catch block

e.printStackTrace();

}

return coord;

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*//

public String getGPSCoord(String ip)

{

String coord = null;

Statement sentSQL = null;

String query = null;

ResultSet rs = null;

try

{

sentSQL = con.createStatement();

query = "SELECT COORD FROM GPS WHERE VEHICLEID = (SELECT VEHICLEID FROM VEHICLE WHERE IP = '" + ip + "')";

rs = sentSQL.executeQuery(query);

rs.next();

coord = rs.getString(1);

coord = getCoord(coord);

sentSQL.close();

}

catch (SQLException e)

{

// Auto-generated catch block

e.printStackTrace();

}

return coord;

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*//

public boolean checkIP(String ip)

{

boolean validation = false;

Statement sentSQL = null;

String query = null;

ResultSet rs = null;

try

{

sentSQL = con.createStatement();

query = "SELECT \* FROM VEHICLE WHERE IP = '" + ip + "'";

rs = sentSQL.executeQuery(query);

validation = rs.next();

sentSQL.close();

}

catch (SQLException e)

{

// Auto-generated catch block

e.printStackTrace();

}

return validation;

}

}

CAR SERVER (Deamon)

package domain.server;

import java.net.\* ;

import domain.util.SocketManager;

public final class CarServer

{

public static void main(String argv[]) throws Exception

{

System.*out*.println("CarServer working");

// Set the port number.

int port = 3000;

ServerSocket wellcomeSocket = new ServerSocket(port);

while (true)

{

//Aceptar la nueva petición y crear el SocketManager para gestionar el Socket obtenido

SocketManager manager = new SocketManager(wellcomeSocket.accept());

//Crear un objeto HttpRequest para gestionar la petición

RequestServer rs = new RequestServer(manager);

//Crear un Thread para el objeto HttpRequest

Thread t = new Thread(rs);

System.*out*.println("I have created a requestServer child");

//Arrancar el Thread

t.start();

}

}

}

GRID SERVER (Deamon)

package domain.server;

import java.net.ServerSocket;

import domain.util.SocketManager;

public class GridServer {

/\*\*

\* @param args

\* @throws Exception

\*/

public static void main(String[] args) throws Exception {

System.*out*.println("GridServer working");

// Set the port number.

int port = 3001;//(new Integer(argv[0])).intValue();

ServerSocket wellcomeSocket = new ServerSocket(port);

while (true)

{

//Aceptar la nueva petición y crear el SocketManager para gestionar el Socket obtenido

SocketManager manager = new SocketManager(wellcomeSocket.accept());

//Crear un objeto HttpRequest para gestionar la petición

RequestGridServer rs = new RequestGridServer(manager);

//Crear un Thread para el objeto HttpRequest

Thread t = new Thread(rs);

System.*out*.println("I have created a requestGridServer child");

//Arrancar el Thread

t.start();

}

}

}

REQUEST GRID SERVER

package domain.server;

import java.util.NoSuchElementException;

import java.util.StringTokenizer;

import DB.DBM;

import domain.util.SocketManager;

final class RequestGridServer implements Runnable

{

private SocketManager sockManager;

private String user;

private int state = 0;

private DBM DBM = null;

//Constructor

public RequestGridServer(SocketManager sockMan) throws Exception

{

sockManager = sockMan;

this.DBM = new DBM();

}

// Implement the run() method of the Runnable interface.

public void run()

{

try

{

processRequest();

}

catch (Exception e)

{

System.out.println(e);

}

}

private void processRequest() throws Exception

{

while (state != 3)

{

String requestLine = sockManager.Leer();

System.out.println("RequestLine: " + requestLine);

//extract the command from the request line

StringTokenizer tokens = new StringTokenizer(requestLine);

String operation = tokens.nextToken();

//Grafo

if(operation.equals("QUIT"))

{

state = 3;

sockManager.Escribir("222 Ok Bye" + "\n");

}

else

{

switch(state)

{

case 0:

{

if(operation.equals("USER"))

{

try

{

user = tokens.nextToken();

boolean validation = DBM.checkUser(user);

if(validation == true)

{

sockManager.Escribir("201 OK Welcome " + user + "\n");

state = 1;

}

else

{

sockManager.Escribir("401 ERROR, user does not exist" + "\n");

}

}

catch(NoSuchElementException e)

{

sockManager.Escribir("404 ERROR, missing username parameter" + "\n");

}

}

else

sockManager.Escribir("Unknown command" + "\n");

break;

}

case 1:

{

if(operation.equals("PASS"))

{

try

{

String pass = tokens.nextToken();

boolean validation = DBM.checkPass(user,pass);

if(validation == true)

{

sockManager.Escribir("202 OK Welcome to the system" + "\n");

state = 2;

}

else

{

sockManager.Escribir("402 ERROR autentification error" + "\n" );

state = 0;

}

}

catch(NoSuchElementException e)

{

sockManager.Escribir("408 ERROR, missing password parameter" + "\n");

}

}

else

sockManager.Escribir("Unknown Command" + "\n");

break;

}

case 2:

{

if(operation.equals("GET\_COOR"))

{

try

{

String cellID = tokens.nextToken();

boolean verification = DBM.verifyCell(cellID);

if (verification == true)

{

//la celda existe

String coord = DBM.getGPSCoordCell(cellID);

sockManager.Escribir("114 Ok ;"+ coord + "\n");

}

else

{

sockManager.Escribir("417 ERROR Unknown cell" + "\n");

}

}

catch(NoSuchElementException e)

{

sockManager.Escribir("418 ERROR, missing cell\_id parameter" + "\n");

}

}

break;

}

}

}

}

// Close streams and socket.

sockManager.CerrarStreams();

sockManager.CerrarSocket();

}

}

REQUEST SERVER

package domain.server;

import java.io.File;

import java.io.FileInputStream;

import java.net.Socket;

import java.util.Calendar;

import java.util.NoSuchElementException;

import java.util.StringTokenizer;

import java.util.Vector;

import DB.DBM;

import domain.util.SocketManager;

final class RequestServer implements Runnable {

private SocketManager sockManager;

private String user;

private String pass;

private String ip;

private int state = 0;

private DBM DBM = null;

// Constructor

public RequestServer(SocketManager sockMan) throws Exception {

sockManager = sockMan;

this.DBM = new DBM();

}

// Implement the run() method of the Runnable interface.

public void run() {

try {

processRequest();

}

catch (Exception e) {

System.*out*.println(e);

}

}

private void processRequest() throws Exception {

while (state != 6 )

{

String requestLine = sockManager.Leer();

System.*out*.println("RequestLine: " + requestLine);

//Extract the command from the request line.

StringTokenizer tokens = new StringTokenizer(requestLine);

String operation = tokens.nextToken();

//Grafo

if(operation.equals("QUIT"))

{

state = 6;

sockManager.Escribir("208 OK Bye" + "\n");

}

else

{

switch (state)

{

case 0:

{

if(operation.equals("USER"))

{

try

{

user = tokens.nextToken();

boolean validation = DBM.checkUser(user);

if(validation == true)

{

sockManager.Escribir("201 OK Welcome " + user + '\n');

state = 1;

}

else

{

sockManager.Escribir("401 ERROR user does not exist" + '\n');

}

}

catch(NoSuchElementException e)

{

sockManager.Escribir("408 ERROR, missing username parameter" + "\n");

}

}

else

sockManager.Escribir("Unknown Command" + "\n");

break;

}

case 1:

{

if(operation.equals("PASS"))

{

try

{

String pass = tokens.nextToken();

this.pass = pass;

boolean validation = DBM.checkPass(user,pass);

if(validation == true)

{

sockManager.Escribir("202 OK Welcome to the system" + "\n");

state = 2;

}

else

{

sockManager.Escribir("402 ERROR autentification error" + "\n" );

state = 0;

}

}

catch(NoSuchElementException e)

{

sockManager.Escribir("408 ERROR, missing password parameter" + "\n");

}

}

else

sockManager.Escribir("Unknown Command" + "\n");

break;

}

case 2:

{

if(operation.equals("CONNECT"))

{

try

{

ip=tokens.nextToken();

//comprobar que existe un servidor con esa ip

boolean validation = DBM.checkIP(ip);

if(validation == true)

{

//hay un servidor con esa ip

sockManager.Escribir("214 OK Connection with the server stabished" + "\n");

state = 3;

}

else

{

//no hay ningun servidor con esa ip

sockManager.Escribir("444 ERROR, The server is not available o does not exist" + "\n");

//si esta mal que no me cambie el estado, que me mantenga el user y el pass

// y me deje introducir una nueva IP

}

}

catch(NoSuchElementException e)

{

sockManager.Escribir("408 ERROR, missing Server IP parameter" + "\n");

}

}

else

sockManager.Escribir("Unknown Command" + "\n");

break;

}

case 3:

{

if (operation.equals("LISTSENSOR"))

{

sockManager.Escribir("112 OK start of sensor list" + "\n");

Vector<String> listSensors=DBM.getListSensors(ip);

int size=listSensors.size();

sockManager.Escribir(size + "\n");

for(int i = 0; i<size; i++)

{

sockManager.Escribir(listSensors.elementAt(i) + "\n");

}

sockManager.Escribir("212 OK end of sensor list" + "\n");

state = 4;

}

if( operation.equals("GPSON"))

{

boolean GPSState = DBM.getGPSState(ip);

if(GPSState == true)

{

sockManager.Escribir("409 ERROR GPS already activated" + "\n");

}

else

{

DBM.switchONGPS(ip);

sockManager.Escribir("205 OK GPS activated" + "\n");

}

}

if( operation.equals("GPSOFF"))

{

boolean GPSState = DBM.getGPSState(ip);

if(GPSState == false)

{

sockManager.Escribir("420 ERROR GPS already deactivated" + "\n");

}

else

{

DBM.switchOFFGPS(ip);

sockManager.Escribir("206 OK GPS deactivated" + "\n");

}

}

if(operation.equals("GET\_PIC"))

{

/\*String fileName = "data/images/street/imagen1.jpg";

File originalPhoto = new File(fileName);

FileInputStream stream = new FileInputStream(originalPhoto);

byte[] buffer = new byte[1024];

long totalSize = originalPhoto.length();

sockManager.Escribir("206 OK Loading image [" + Long.toString(totalSize)+ "] bytes..." + "\n");

sockManager.Escribir(Long.toString(totalSize)+ "\n");

int read = 0;

for(read = stream.read(buffer); read !=-1; read = stream.read(buffer))

{

sockManager.EscribirBytes(buffer);

}

sockManager.Escribir("\n" + "Photo transmited..." + "\n");

state = 5;\*/

String fileName = "data/images/street/imagen1.jpg";

File originalPhoto = new File(fileName);

FileInputStream stream = new FileInputStream(originalPhoto);

byte[] buffer = new byte[(int)originalPhoto.length()];

sockManager.Escribir("206 OK Loading image [" + buffer.length+ " bytes]" + "\n");

long fileSize = originalPhoto.length();

sockManager.Escribir(fileSize + "\n");

stream.read(buffer);

sockManager.EscribirBytes(buffer);

sockManager.Escribir("photo transmited" + "\n");

state = 5;

}

if(!operation.equals("GET\_PIC") && !operation.equals("GPSON") && !operation.equals("GPSOFF") && !operation.equals("LISTSENSOR"))

{

sockManager.Escribir("Unknown Command" + "\n");

}

break;

}

case 4:

{

if (operation.equals("HISTORYLOG"))

{

try

{

String sensorID = tokens.nextToken();

boolean validation = DBM.verifySensor(sensorID, ip);

if (validation == true)

{

sockManager.Escribir("113 OK Start of measurement list" + "\n");

Vector<String> listMeasurements=DBM.getListMeasurements(sensorID);

int size=listMeasurements.size();

sockManager.Escribir(size + "\n");

if(size == 0)

{

sockManager.Escribir("Ups, there aren't measurements for this sensor" + "\n");

}

else

{

for(int i = 0; i < size; i++)

{

sockManager.Escribir(listMeasurements.elementAt(i) + "\n");

}

sockManager.Escribir("212 OK End of measurement list" + "\n");

}

}

else

{

sockManager.Escribir("414 ERROR unknown sensorID" + "\n");

}

}

catch(NoSuchElementException e)

{

sockManager.Escribir("408 ERROR, missing sensorID parameter" + "\n");

}

}

if (operation.equals("ON"))

{

try

{

String sensorID = tokens.nextToken();

boolean validation = DBM.verifySensor(sensorID, ip);

if( validation == true)

{

boolean sensorState = DBM.getSensorState(sensorID);

if(sensorState == true)

{

sockManager.Escribir("418 ERROR Sensor already activated" + "\n");

}

else

{

DBM.switchONSensor(sensorID);

sockManager.Escribir("203 OK Sensor activated" + "\n");

}

}

else

{

sockManager.Escribir("417 ERROR unknown sensorID" + "\n");

}

}

catch(NoSuchElementException e)

{

sockManager.Escribir("415 ERROR, missing sensor parameter" + "\n");

}

}

if (operation.equals("OFF"))

{

try

{

String sensorID = tokens.nextToken();

boolean validation = DBM.verifySensor(sensorID, ip);

if( validation == true)

{

boolean sensorState = DBM.getSensorState(sensorID);

if(sensorState == false)

{

sockManager.Escribir("419 ERROR Sensor already deactivated" + "\n");

}

else

{

DBM.switchOFFSensor(sensorID);

sockManager.Escribir("204 OK Sensor deactivated" + "\n");

}

}

else

{

sockManager.Escribir("417 ERROR unknown sensorID" + "\n");

}

}

catch(NoSuchElementException e)

{

sockManager.Escribir("420 ERROR, missing sensor parameter" + "\n");

}

}

if( operation.equals("GET\_CURVALUE"))

{

try

{

String sensorID = tokens.nextToken();

boolean validation = DBM.verifySensor(sensorID, ip);

if (validation == true)

{

//el sernsor pertenece al coche al que estamos conectados

boolean sensorState = DBM.getSensorState(sensorID);

if(sensorState == true)

{

//el gps esta encencido y podemos dar sus datos

//obetener los datos de la lectutra y enviarselos: DATE+TIME+COORDINATES+VALUE

String value = DBM.getSensorValue(sensorID);

String coord = DBM.getSensorCoord(sensorID);

//cogemos la fecha, esto es un poco rebuscado:

Calendar c = Calendar.*getInstance*();

String day = Integer.*toString*(c.get(Calendar.*DATE*));

String month = Integer.*toString*(c.get(Calendar.*MONTH*));

String year = Integer.*toString*(c.get(Calendar.*YEAR*));

//cogemos la hora actual:

String hour =Integer.*toString*(c.get(Calendar.*HOUR\_OF\_DAY*));

String minutes = Integer.*toString*(c.get(Calendar.*MINUTE*));

String seconds =Integer.*toString*( c.get(Calendar.*SECOND*));

sockManager.Escribir("114 OK; " + day + "/" + month + "/" + year + "; " + hour + ":" + minutes + ":" + seconds + "; " + coord + "; " + value + "\n");

}

else

{

//el gps esta apagado y no podemos dar sus datos

sockManager.Escribir("416 ERROR Sensor is not activated" + "\n");

}

}

else

{

//el sensor o no existe, o no pertenece al coche en el que estamos

sockManager.Escribir("414 ERROR Unknown sensorID" + "\n");

}

}

catch(NoSuchElementException e)

{

sockManager.Escribir("415 ERROR, missing sensor parameter" + "\n");

}

}

if( operation.equals("RETURN"))

{

state = 3;

sockManager.Escribir("Coming back to the main menu..." + "\n");

}

if(!operation.equals("HISTORYLOG") && !operation.equals("ON") && !operation.equals("OFF") && !operation.equals("GET\_CURVALUE") && !operation.equals("RETURN"))

{

sockManager.Escribir("Unknown Command" + "\n");

}

break;

}

case 5:

{

if(operation.equals("GET\_LOC"))

{

//tenemos que comprobar si el GPS esta activado, si lo esta devolveremos las coordenadas,

//si esta descativado llamamos al servidor Grid

boolean GPSState = DBM.getGPSState(ip);

if(GPSState == true)

{

//el GPS esta encendido

String coord = DBM.getGPSCoord(ip);

sockManager.Escribir("114 OK ;" + coord + "\n");

}

else

{

//el GPS esta apagado: Llamamos al Grid Server

Socket gridSocket = new Socket("127.0.0.1", 3001);

SocketManager manager = new SocketManager(gridSocket);

manager.Escribir("USER " + user + "\n");

System.*out*.println(manager.Leer());

manager.Escribir("PASS " + pass + "\n");

System.*out*.println(manager.Leer());

String cell = DBM.getActualGrid(ip);

manager.Escribir("GET\_COOR " + cell + "\n");

sockManager.Escribir(manager.Leer() + "\n");

manager.Escribir("QUIT" + "\n");

manager.Leer();

manager.CerrarSocket();

manager.CerrarStreams();

}

state = 3;

}

else

{

sockManager.Escribir("Unknown Command" + "\n");

}

break;

}

default:

{

System.*out*.println("I'm getting crazy!!!!!!");

}

}

}

}

// Close streams and socket.

sockManager.CerrarStreams();

sockManager.CerrarSocket();

DBM.disconnectDB();

}

}

SOCKET MANAGER

package domain.util;

import java.net.\*;

import java.io.\*;

public class SocketManager {

private Socket mySocket;

private DataOutputStream bufferEscritura;

private BufferedReader bufferLectura;

public SocketManager(Socket sock) throws IOException {

this.mySocket = sock;

InicializaStreams();

}

/\*\*

\*

\* @param address InetAddress

\* @param port int numero de puerto

\* @throws IOException

\*/

public SocketManager(InetAddress address, int port) throws IOException {

mySocket = new Socket(address, port);

InicializaStreams();

}

/\*\*

\*

\* @param host String nombre del servidor al que se conecta

\* @param port int puerto de conexion

\* @throws IOException

\*/

public SocketManager(String host, int port) throws IOException {

mySocket = new Socket(host, port);

InicializaStreams();

}

/\*\*

\* Inicialización de los bufferes de lectura y escritura del socket

\* @throws IOException

\*/

public void InicializaStreams() throws IOException {

bufferEscritura = new DataOutputStream(mySocket.getOutputStream());

bufferLectura = new BufferedReader(new InputStreamReader(mySocket.

getInputStream()));

}

public void CerrarStreams() throws IOException {

bufferEscritura.close();

bufferLectura.close();

}

public void CerrarSocket() throws IOException {

mySocket.close();

}

/\*\*

\*

\* @return String

\* @throws IOException

\*/

public String Leer() throws IOException {

return (bufferLectura.readLine());

}

public byte [] LeerBytes (int tam) throws IOException {

byte [] b = new byte [tam] ;

int bytesRead =0;

while (bytesRead<tam)

{

bytesRead += this.mySocket.getInputStream().read(b, bytesRead, ((tam-bytesRead)>=5000)?5000:tam-bytesRead);

}

return b;

}

public void Escribir(String contenido) throws IOException {

bufferEscritura.writeBytes(contenido);

}

public void Escribir(byte[] buffer, int bytes) throws IOException {

bufferEscritura.write(buffer, 0, bytes);

}

public void EscribirBytes(byte []buff) throws IOException {

this.mySocket.getOutputStream().write(buff);

}

}

CLIENT GUI

**package** GUI;

**import** java.awt.BorderLayout;

**import** java.awt.event.ActionEvent;

**import** java.io.IOException;

**import** java.net.Socket;

**import** javax.swing.AbstractAction;

**import** javax.swing.BorderFactory;

**import** javax.swing.BoxLayout;

**import** javax.swing.GroupLayout;

**import** javax.swing.ImageIcon;

**import** javax.swing.JButton;

**import** javax.swing.JComponent;

**import** javax.swing.JLabel;

**import** javax.swing.JPanel;

**import** javax.swing.JTextField;

**import** javax.swing.LayoutStyle;

**import** javax.swing.SwingUtilities;

**import** javax.swing.WindowConstants;

**import** javax.swing.border.BevelBorder;

**import** domain.util.SocketManager;

/\*\*

\* This code was edited or generated using CloudGarden's Jigloo

\* SWT/Swing GUI Builder, which is free for non-commercial

\* use. If Jigloo is being used commercially (ie, by a corporation,

\* company or business for any purpose whatever) then you

\* should purchase a license for each developer using Jigloo.

\* Please visit www.cloudgarden.com for details.

\* Use of Jigloo implies acceptance of these licensing terms.

\* A COMMERCIAL LICENSE HAS NOT BEEN PURCHASED FOR

\* THIS MACHINE, SO JIGLOO OR THIS CODE CANNOT BE USED

\* LEGALLY FOR ANY CORPORATE OR COMMERCIAL PURPOSE.

\*/

**public** **class** ClientGUI **extends** javax.swing.JFrame {

{

//Set Look & Feel

**try** {

javax.swing.UIManager.*setLookAndFeel*("com.sun.java.swing.plaf.motif.MotifLookAndFeel");

} **catch**(Exception e) {

e.printStackTrace();

}

}

/\*\*

\*

\*/

**private** **static** **final** **long** *serialVersionUID* = 1L;

**private** JLabel backgroundImage;

**private** JPanel backgroundPanel;

**private** String backgroundRoute="data/images/background.png";

**private** AbstractAction login;

**private** JButton conect;

**private** JLabel ipLabel;

**private** JLabel statusBar;

**private** AbstractAction connect;

**private** JTextField ipText;

**private** JTextField nameField;

**private** JLabel Password;

**private** JLabel jLabel1;

**private** JTextField password;

**private** JButton loginBUtton;

**private** JPanel arriba;

**private** AbstractAction abstractAction1;

**private** JButton jButton2;

**private** JPanel baseComponents;

**private** JPanel South;

**private** JPanel East;

**private** JPanel West;

**private** JPanel North;

**private** JPanel Center;

**private** **static** ClientGUI *inst*;

**private** **static** SocketManager *manager*;

/\*\*

\* Auto-generated main method to display this JFrame

\*/

**public** **static** **void** main(String[] args) {

SwingUtilities.*invokeLater*(**new** Runnable() {

**public** **void** run() {

**try** {

Socket clientSocket = **new** Socket("127.0.0.1", 3000);

*manager* = **new** SocketManager(clientSocket);

} **catch** (IOException e) {

// **TODO** Auto-generated catch block

e.printStackTrace();

}

*inst* = **new** ClientGUI();

*inst*.setLocationRelativeTo(**null**);

*inst*.setVisible(**true**);

}

});

}

**public** ClientGUI() {

**super**();

initGUI();

}

**private** **void** initGUI() {

**try** {

BoxLayout thisLayout = **new** BoxLayout(getContentPane(), javax.swing.BoxLayout.*X\_AXIS*);

getContentPane().setLayout(thisLayout);

setDefaultCloseOperation(WindowConstants.*DISPOSE\_ON\_CLOSE*);

{

backgroundPanel = **new** JPanel();

backgroundPanel.setLayout(**null**);

getContentPane().add(backgroundPanel);

{

backgroundImage = **new** JLabel();

backgroundPanel.add(backgroundImage);

backgroundImage.setText(" ");

backgroundImage.setIcon(**new** ImageIcon(backgroundRoute));

backgroundImage.setBounds(0, 0, 792, 566);

{

baseComponents = **new** JPanel();

BorderLayout baseComponentsLayout = **new** BorderLayout();

baseComponents.setLayout(baseComponentsLayout);

backgroundImage.add(baseComponents);

baseComponents.setBounds(0, 0, 792, 566);

baseComponents.setOpaque(**false**);

{

Center = **new** JPanel();

BoxLayout CenterLayout = **new** BoxLayout(Center, javax.swing.BoxLayout.*Y\_AXIS*);

Center.setLayout(CenterLayout);

baseComponents.add(Center, BorderLayout.*CENTER*);

Center.setOpaque(**false**);

Center.setPreferredSize(**new** java.awt.Dimension(772, 471));

Center.add(getArriba());

}

{

North = **new** JPanel();

GroupLayout NorthLayout = **new** GroupLayout((JComponent)North);

North.setLayout(NorthLayout);

baseComponents.add(North, BorderLayout.*NORTH*);

North.setPreferredSize(**new** java.awt.Dimension(792, 102));

North.setOpaque(**false**);

NorthLayout.setVerticalGroup(NorthLayout.createSequentialGroup());

NorthLayout.setHorizontalGroup(NorthLayout.createSequentialGroup());

}

{

West = **new** JPanel();

baseComponents.add(West, BorderLayout.*WEST*);

West.setOpaque(**false**);

West.setPreferredSize(**new** java.awt.Dimension(45, 382));

}

{

East = **new** JPanel();

baseComponents.add(East, BorderLayout.*EAST*);

East.setOpaque(**false**);

East.setPreferredSize(**new** java.awt.Dimension(48, 382));

}

{

South = **new** JPanel();

GroupLayout SouthLayout = **new** GroupLayout((JComponent)South);

South.setLayout(SouthLayout);

baseComponents.add(South, BorderLayout.*SOUTH*);

South.setOpaque(**false**);

South.setPreferredSize(**new** java.awt.Dimension(110, 82));

SouthLayout.setHorizontalGroup(SouthLayout.createSequentialGroup()

.addContainerGap(16, 16)

.addComponent(getJButton2(), GroupLayout.*PREFERRED\_SIZE*, 62, GroupLayout.*PREFERRED\_SIZE*)

.addGap(0, 51, Short.*MAX\_VALUE*)

.addComponent(getStatusBar(), GroupLayout.*PREFERRED\_SIZE*, 663, GroupLayout.*PREFERRED\_SIZE*));

SouthLayout.setVerticalGroup(SouthLayout.createSequentialGroup()

.addContainerGap(27, 27)

.addGroup(SouthLayout.createParallelGroup()

.addGroup(GroupLayout.Alignment.*LEADING*, SouthLayout.createSequentialGroup()

.addComponent(getJButton2(), GroupLayout.*PREFERRED\_SIZE*, 41, GroupLayout.*PREFERRED\_SIZE*)

.addGap(0, 14, Short.*MAX\_VALUE*))

.addGroup(GroupLayout.Alignment.*LEADING*, SouthLayout.createSequentialGroup()

.addGap(0, 27, Short.*MAX\_VALUE*)

.addComponent(getStatusBar(), GroupLayout.*PREFERRED\_SIZE*, 28, GroupLayout.*PREFERRED\_SIZE*))));

}

}

}

}

pack();

setSize(800, 600);

} **catch** (Exception e) {

e.printStackTrace();

}

}

**private** JButton getJButton2() {

**if**(jButton2 == **null**) {

jButton2 = **new** JButton();

jButton2.setAction(getAbstractAction1());

jButton2.setIcon(**new** ImageIcon("data/images/buttons/salir50x50.png"));

jButton2.setContentAreaFilled(**false**);

jButton2.setSize(50, 50);

jButton2.setBorder(BorderFactory.*createEmptyBorder*(0, 0, 0, 0));

}

**return** jButton2;

}

**private** AbstractAction getAbstractAction1() {

**if**(abstractAction1 == **null**) {

abstractAction1 = **new** AbstractAction("", **null**) {

/\*\*

\*

\*/

**private** **static** **final** **long** *serialVersionUID* = 7828558753203927061L;

**public** **void** actionPerformed(ActionEvent evt) {

**try** {

*manager*.Escribir("QUIT" + "\n");

System.*out*.println(*manager*.Leer());

} **catch** (IOException e) {

// **TODO** Auto-generated catch block

e.printStackTrace();

}

System.*exit*(0);

}

};

}

**return** abstractAction1;

}

**private** JPanel getArriba() {

**if**(arriba == **null**) {

arriba = **new** JPanel();

GroupLayout arribaLayout = **new** GroupLayout((JComponent)arriba);

arriba.setLayout(arribaLayout);

arriba.setPreferredSize(**new** java.awt.Dimension(699, 241));

arriba.setOpaque(**false**);

arribaLayout.setHorizontalGroup(arribaLayout.createSequentialGroup()

.addContainerGap(38, 38)

.addGroup(arribaLayout.createParallelGroup()

.addGroup(GroupLayout.Alignment.*LEADING*, arribaLayout.createSequentialGroup()

.addComponent(getPasswordx(), GroupLayout.*PREFERRED\_SIZE*, 130, GroupLayout.*PREFERRED\_SIZE*)

.addGap(6))

.addGroup(arribaLayout.createSequentialGroup()

.addGap(29)

.addGroup(arribaLayout.createParallelGroup()

.addComponent(getJLabel1(), GroupLayout.Alignment.*LEADING*, GroupLayout.*PREFERRED\_SIZE*, 107, GroupLayout.*PREFERRED\_SIZE*)

.addGroup(GroupLayout.Alignment.*LEADING*, arribaLayout.createSequentialGroup()

.addGap(31)

.addComponent(getIpLabel(), GroupLayout.*PREFERRED\_SIZE*, 55, GroupLayout.*PREFERRED\_SIZE*)

.addGap(21)))))

.addPreferredGap(LayoutStyle.ComponentPlacement.*RELATED*)

.addGroup(arribaLayout.createParallelGroup()

.addGroup(GroupLayout.Alignment.*LEADING*, arribaLayout.createSequentialGroup()

.addComponent(getNameField(), GroupLayout.*PREFERRED\_SIZE*, 200, GroupLayout.*PREFERRED\_SIZE*)

.addGap(0, 83, Short.*MAX\_VALUE*))

.addGroup(GroupLayout.Alignment.*LEADING*, arribaLayout.createSequentialGroup()

.addComponent(getPassword(), GroupLayout.*PREFERRED\_SIZE*, 200, GroupLayout.*PREFERRED\_SIZE*)

.addGap(0, 83, Short.*MAX\_VALUE*))

.addGroup(GroupLayout.Alignment.*LEADING*, arribaLayout.createSequentialGroup()

.addComponent(getIpText(), GroupLayout.*PREFERRED\_SIZE*, 200, GroupLayout.*PREFERRED\_SIZE*)

.addGap(0, 83, Short.*MAX\_VALUE*))

.addGroup(arribaLayout.createSequentialGroup()

.addGap(12)

.addGroup(arribaLayout.createParallelGroup()

.addGroup(arribaLayout.createSequentialGroup()

.addComponent(getConect(), GroupLayout.*PREFERRED\_SIZE*, 271, GroupLayout.*PREFERRED\_SIZE*)

.addGap(0, 0, Short.*MAX\_VALUE*))

.addGroup(GroupLayout.Alignment.*LEADING*, arribaLayout.createSequentialGroup()

.addGap(15)

.addComponent(getLoginBUtton(), GroupLayout.*PREFERRED\_SIZE*, 195, GroupLayout.*PREFERRED\_SIZE*)

.addGap(0, 61, Short.*MAX\_VALUE*)))))

.addContainerGap(236, 236));

arribaLayout.setVerticalGroup(arribaLayout.createSequentialGroup()

.addContainerGap(43, 43)

.addGroup(arribaLayout.createParallelGroup(GroupLayout.Alignment.*BASELINE*)

.addComponent(getNameField(), GroupLayout.Alignment.*BASELINE*, GroupLayout.*PREFERRED\_SIZE*, 36, GroupLayout.*PREFERRED\_SIZE*)

.addComponent(getJLabel1(), GroupLayout.Alignment.*BASELINE*, GroupLayout.*PREFERRED\_SIZE*, 29, GroupLayout.*PREFERRED\_SIZE*))

.addPreferredGap(LayoutStyle.ComponentPlacement.*RELATED*)

.addGroup(arribaLayout.createParallelGroup()

.addComponent(getPassword(), GroupLayout.Alignment.*LEADING*, GroupLayout.*PREFERRED\_SIZE*, 36, GroupLayout.*PREFERRED\_SIZE*)

.addGroup(GroupLayout.Alignment.*LEADING*, arribaLayout.createSequentialGroup()

.addComponent(getPasswordx(), GroupLayout.*PREFERRED\_SIZE*, 28, GroupLayout.*PREFERRED\_SIZE*)

.addGap(8)))

.addPreferredGap(LayoutStyle.ComponentPlacement.*RELATED*)

.addComponent(getLoginBUtton(), GroupLayout.*PREFERRED\_SIZE*, 63, GroupLayout.*PREFERRED\_SIZE*)

.addGap(38)

.addGroup(arribaLayout.createParallelGroup()

.addGroup(GroupLayout.Alignment.*LEADING*, arribaLayout.createSequentialGroup()

.addComponent(getIpLabel(), GroupLayout.*PREFERRED\_SIZE*, 26, GroupLayout.*PREFERRED\_SIZE*)

.addGap(9))

.addComponent(getIpText(), GroupLayout.Alignment.*LEADING*, GroupLayout.*PREFERRED\_SIZE*, 35, GroupLayout.*PREFERRED\_SIZE*))

.addPreferredGap(LayoutStyle.ComponentPlacement.*RELATED*)

.addComponent(getConect(), GroupLayout.*PREFERRED\_SIZE*, 107, GroupLayout.*PREFERRED\_SIZE*)

.addContainerGap(GroupLayout.*DEFAULT\_SIZE*, Short.*MAX\_VALUE*));

}

**return** arriba;

}

**private** JButton getLoginBUtton() {

**if**(loginBUtton == **null**) {

loginBUtton = **new** JButton();

loginBUtton.setBorder(BorderFactory.*createEmptyBorder*(0, 0, 0, 0));

loginBUtton.setSize(200, 54);

loginBUtton.setBorderPainted(**false**);

loginBUtton.setContentAreaFilled(**false**);

loginBUtton.setAction(getLogin());

loginBUtton.setIcon(**new** ImageIcon("data/images/buttons/buttonLogin.png"));

}

**return** loginBUtton;

}

**private** JTextField getPassword() {

**if**(password == **null**) {

password = **new** JTextField();

password.setFont(**new** java.awt.Font("High Tower Text",0,28));

password.setBorder(BorderFactory.*createBevelBorder*(BevelBorder.*LOWERED*));

password.setOpaque(**false**);

password.setText("12345");

}

**return** password;

}

**private** JLabel getJLabel1() {

**if**(jLabel1 == **null**) {

jLabel1 = **new** JLabel();

jLabel1.setText("Name");

jLabel1.setFont(**new** java.awt.Font("High Tower Text",1,28));

}

**return** jLabel1;

}

**private** JLabel getPasswordx() {

**if**(Password == **null**) {

Password = **new** JLabel();

Password.setText("Password");

Password.setFont(**new** java.awt.Font("High Tower Text",1,28));

}

**return** Password;

}

**private** JTextField getNameField() {

**if**(nameField == **null**) {

nameField = **new** JTextField();

nameField.setFont(**new** java.awt.Font("High Tower Text",0,28));

nameField.setOpaque(**false**);

nameField.setBorder(BorderFactory.*createBevelBorder*(BevelBorder.*LOWERED*));

nameField.setText("itzi");

}

**return** nameField;

}

**private** JLabel getIpLabel() {

**if**(ipLabel == **null**) {

ipLabel = **new** JLabel();

ipLabel.setText("IP");

ipLabel.setFont(**new** java.awt.Font("High Tower Text",1,28));

ipLabel.setEnabled(**false**);

}

**return** ipLabel;

}

**private** JTextField getIpText() {

**if**(ipText == **null**) {

ipText = **new** JTextField();

ipText.setFont(**new** java.awt.Font("High Tower Text",2,28));

ipText.setEnabled(**false**);

ipText.setBorder(BorderFactory.*createBevelBorder*(BevelBorder.*LOWERED*));

ipText.setOpaque(**false**);

ipText.setText("127.0.0.1");

}

**return** ipText;

}

**private** JButton getConect() {

**if**(conect == **null**) {

conect = **new** JButton();

conect.setText(" ");

conect.setBorderPainted(**false**);

conect.setContentAreaFilled(**false**);

conect.setIcon(**new** ImageIcon("data/images/buttons/buttonConect.png"));

conect.setAction(getConnect());

conect.setSize(200, 200);

}

**return** conect;

}

**private** AbstractAction getLogin() {

**if**(login == **null**) {

login = **new** AbstractAction("", **null**) {

/\*\*

\*

\*/

**private** **static** **final** **long** *serialVersionUID* = 1L;

**public** **void** actionPerformed(ActionEvent evt)

{

String user= nameField.getText();

**try** {

*manager*.Escribir("USER "+user+"\n");

String resultado = *manager*.Leer();

**if**(!resultado.contains("201"))

statusBar.setText(resultado);

**else**

{

*manager*.Escribir("PASS "+password.getText()+ "\n");

resultado = *manager*.Leer();

statusBar.setText(resultado);

**if**(resultado.contains("202"))

{

nameField.setEnabled(**false**);

password.setEnabled(**false**);

jLabel1.setEnabled(**false**);

Password.setEnabled(**false**);

loginBUtton.setVisible(**false**);

ipLabel.setEnabled(**true**);

ipText.setEnabled(**true**);

conect.setVisible(**true**);

}

}

} **catch** (IOException e) {

// **TODO** Auto-generated catch block

e.printStackTrace();

}

}

};

}

**return** login;

}

**private** AbstractAction getConnect() {

**if**(connect == **null**) {

connect = **new** AbstractAction(" ", **null**) {

/\*\*

\*

\*/

**private** **static** **final** **long** *serialVersionUID* = 1L;

**public** **void** actionPerformed(ActionEvent evt)

{

**try** {

*manager*.Escribir("CONNECT "+ipText.getText()+"\n");

String resultado = *manager*.Leer();

statusBar.setText(resultado);

**if**(resultado.contains("214"))

{

menuGUI menu = **new** menuGUI(*manager*);

menu.setGUI(menu);

menu.setVisible(**true**);

menu.setLocationRelativeTo(**null**);

*inst*.setVisible(**false**);

*inst*.dispose();

}

} **catch** (IOException e) {

// **TODO** Auto-generated catch block

e.printStackTrace();

}

}

};

}

**return** connect;

}

**private** JLabel getStatusBar() {

**if**(statusBar == **null**) {

statusBar = **new** JLabel();

statusBar.setText("Initializing...");

statusBar.setFont(**new** java.awt.Font("High Tower Text",2,16));

statusBar.setBackground(**new** java.awt.Color(255,255,255));

statusBar.setOpaque(**true**);

statusBar.setBorder(BorderFactory.*createBevelBorder*(BevelBorder.*LOWERED*));

conect.setContentAreaFilled(**false**);

conect.setIcon(**new** ImageIcon("data/images/buttons/buttonConect.png"));

}

**return** statusBar;

}

}

GETPIC GUI

**package** GUI;

**import** java.awt.event.ActionEvent;

**import** java.io.FileNotFoundException;

**import** java.io.FileOutputStream;

**import** java.io.IOException;

**import** javax.swing.AbstractAction;

**import** javax.swing.BorderFactory;

**import** javax.swing.BoxLayout;

**import** javax.swing.ImageIcon;

**import** javax.swing.JButton;

**import** javax.swing.JLabel;

**import** javax.swing.JPanel;

**import** javax.swing.JTextField;

**import** javax.swing.SwingUtilities;

**import** javax.swing.WindowConstants;

**import** javax.swing.border.BevelBorder;

**import** domain.util.SocketManager;

/\*\*

\* This code was edited or generated using CloudGarden's Jigloo

\* SWT/Swing GUI Builder, which is free for non-commercial

\* use. If Jigloo is being used commercially (ie, by a corporation,

\* company or business for any purpose whatever) then you

\* should purchase a license for each developer using Jigloo.

\* Please visit www.cloudgarden.com for details.

\* Use of Jigloo implies acceptance of these licensing terms.

\* A COMMERCIAL LICENSE HAS NOT BEEN PURCHASED FOR

\* THIS MACHINE, SO JIGLOO OR THIS CODE CANNOT BE USED

\* LEGALLY FOR ANY CORPORATE OR COMMERCIAL PURPOSE.

\*/

**public** **class** getPicGUI **extends** javax.swing.JFrame {

/\*\*

\*

\*/

**private** **static** **final** **long** *serialVersionUID* = 1L;

**private** JLabel background;

**private** JPanel center;

**private** JLabel picture;

**private** JButton getPic;

**private** JLabel statusBar;

**private** AbstractAction backAction;

**private** AbstractAction exitAction;

**private** JButton backButton;

**private** JTextField locField;

**private** AbstractAction saveImgAction;

**private** JButton saveImg;

**private** AbstractAction picAction;

**private** JButton exitButton;

**private** JPanel down;

**private** JPanel up;

**private** **static** getPicGUI *window*;

**private** SocketManager manager;

**private** **byte**[] buffer;

/\*\*

\* Auto-generated main method to display this JFrame

\*/

**public** **static** **void** main(String[] args) {

SwingUtilities.*invokeLater*(**new** Runnable() {

**public** **void** run() {

*window* = **new** getPicGUI();

*window*.setLocationRelativeTo(**null**);

*window*.setVisible(**true**);

}

});

}

**public** getPicGUI(SocketManager manager)

{

**super**();

initGUI();

**this**.manager=manager;

}

**public** getPicGUI() {

**super**();

initGUI();

}

@SuppressWarnings("static-access")

**public** **void** setGUI(getPicGUI window)

{

**this**.*window*=window;

}

**private** **void** initGUI() {

**try** {

setDefaultCloseOperation(WindowConstants.*DISPOSE\_ON\_CLOSE*);

getContentPane().setLayout(**null**);

{

background = **new** JLabel();

BoxLayout backgroundLayout = **new** BoxLayout(background, javax.swing.BoxLayout.*Y\_AXIS*);

background.setLayout(backgroundLayout);

getContentPane().add(background);

background.setBounds(0, 0, 792, 566);

background.setIcon(**new** ImageIcon("data/images/background.png"));

background.setOpaque(**true**);

{

up = **new** JPanel();

background.add(up);

up.setOpaque(**false**);

}

{

center = **new** JPanel();

background.add(center);

center.setPreferredSize(**new** java.awt.Dimension(792, 492));

center.setOpaque(**false**);

center.setLayout(**null**);

center.add(getPicture());

center.add(getGetPic());

center.add(getLocField());

center.add(getSaveImg());

}

{

down = **new** JPanel();

background.add(down);

down.setLayout(**null**);

down.setPreferredSize(**new** java.awt.Dimension(792, 79));

down.setOpaque(**false**);

{

exitButton = **new** JButton();

down.add(exitButton);

exitButton.setAction(getExitAction());

exitButton.setText(" ");

exitButton.setIcon(**new** ImageIcon("data/images/buttons/salir50x50.png"));

exitButton.setContentAreaFilled(**false**);

exitButton.setBorder(BorderFactory.*createEmptyBorder*(0, 0, 0, 0));

exitButton.setOpaque(**false**);

exitButton.setBounds(17, 0, 80, 67);

}

{

backButton = **new** JButton();

down.add(backButton);

down.add(getStatusBar());

backButton.setAction(getBackAction());

backButton.setText(" ");

backButton.setBounds(122, 11, 157, 45);

backButton.setBorder(BorderFactory.*createEmptyBorder*(0, 0, 0, 0));

backButton.setOpaque(**false**);

backButton.setIcon(**new** ImageIcon("data/images/buttons/backButton.png"));

backButton.setContentAreaFilled(**false**);

}

}

}

pack();

**this**.setSize(800, 600);

} **catch** (Exception e) {

e.printStackTrace();

}

}

**private** AbstractAction getExitAction() {

**if**(exitAction == **null**) {

exitAction = **new** AbstractAction(" ", **null**) {

/\*\*

\*

\*/

**private** **static** **final** **long** *serialVersionUID* = 1L;

**public** **void** actionPerformed(ActionEvent evt) {

**try** {

manager.Escribir("QUIT" + "\n");

System.*out*.println(manager.Leer());

} **catch** (IOException e) {

// **TODO** Auto-generated catch block

e.printStackTrace();

}

System.*exit*(0);

}

};

}

**return** exitAction;

}

**private** AbstractAction getBackAction() {

**if**(backAction == **null**) {

backAction = **new** AbstractAction(" ", **null**) {

/\*\*

\*

\*/

**private** **static** **final** **long** *serialVersionUID* = 1L;

**public** **void** actionPerformed(ActionEvent evt) {

menuGUI menu = **new** menuGUI(manager);

menu.setVisible(**true**);

menu.setGUI(menu);

menu.setLocationRelativeTo(**null**);

*window*.setVisible(**false**);

*window*.dispose();

}

};

}

**return** backAction;

}

**private** JLabel getStatusBar() {

**if**(statusBar == **null**) {

statusBar = **new** JLabel();

statusBar.setText(" ");

statusBar.setBounds(304, 50, 488, 26);

statusBar.setBackground(**new** java.awt.Color(255,255,255));

statusBar.setBorder(BorderFactory.*createBevelBorder*(BevelBorder.*LOWERED*));

statusBar.setOpaque(**true**);

statusBar.setFont(**new** java.awt.Font("Dialog",2,12));

}

**return** statusBar;

}

**private** JLabel getPicture() {

**if**(picture == **null**) {

picture = **new** JLabel();

picture.setText(" ");

picture.setBounds(28, 92, 500, 375);

picture.setBorder(BorderFactory.*createEtchedBorder*(BevelBorder.*LOWERED*));

}

**return** picture;

}

**private** JButton getGetPic() {

**if**(getPic == **null**) {

getPic = **new** JButton();

getPic.setBounds(566, 119, 138, 46);

getPic.setAction(getPicAction());

getPic.setOpaque(**false**);

getPic.setBorder(BorderFactory.*createEmptyBorder*(0, 0, 0, 0));

getPic.setIcon(**new** ImageIcon("data/images/buttons/getImg.png"));

getPic.setContentAreaFilled(**false**);

}

**return** getPic;

}

**private** JTextField getLocField() {

**if**(locField == **null**) {

locField = **new** JTextField();

locField.setBounds(546, 243, 214, 43);

}

**return** locField;

}

**private** AbstractAction getPicAction() {

**if**(picAction == **null**) {

picAction = **new** AbstractAction(" ", **null**) {

/\*\*

\*

\*/

**private** **static** **final** **long** *serialVersionUID* = 1L;

**public** **void** actionPerformed(ActionEvent evt) {

statusBar.setText("");

**try** {

/\*manager.Escribir("GET\_PIC" + "\n");

System.out.println(manager.Leer());

int totalSize = Integer.parseInt(manager.Leer());

buffer = new byte[totalSize];

int read = 0;

while(read < totalSize)

{

int bytesToRead = 0;

if(((totalSize-read)/1024) >= 1)

bytesToRead = 1024;

else

bytesToRead = totalSize - read;

byte[] subBuffer = manager.LeerBytes(bytesToRead);//bytes

for(int i = 0; i < subBuffer.length; i++)

{

buffer[read] = subBuffer[i];

read++;

}

}

manager.Leer();

System.out.println(manager.Leer());//transmited

picture.setIcon(new ImageIcon(buffer));\*/

manager.Escribir("GET\_PIC" + "\n");

System.*out*.println(manager.Leer());

**int** tam =Integer.*parseInt*(manager.Leer());

buffer = manager.LeerBytes(tam);

System.*out*.println(manager.Leer());

picture.setIcon(**new** ImageIcon(buffer));

manager.Escribir("GET\_LOC" + "\n");

String response=manager.Leer();

statusBar.setText(response);

**if**(response.contains("114"))

{

String[] location = response.split(";");

locField.setText(location[1]);

}

} **catch** (IOException e) {

// **TODO** Auto-generated catch block

e.printStackTrace();

}

}

};

}

**return** picAction;

}

**private** JButton getSaveImg() {

**if**(saveImg == **null**) {

saveImg = **new** JButton();

saveImg.setText(" ");

saveImg.setBounds(566, 184, 138, 47);

saveImg.setAction(getSaveImgAction());

saveImg.setOpaque(**false**);

saveImg.setBorder(BorderFactory.*createEmptyBorder*(0, 0, 0, 0));

saveImg.setIcon(**new** ImageIcon("data/images/buttons/saveImg.png"));

saveImg.setContentAreaFilled(**false**);

}

**return** saveImg;

}

**private** AbstractAction getSaveImgAction() {

**if**(saveImgAction == **null**) {

saveImgAction = **new** AbstractAction("", **null**) {

/\*\*

\*

\*/

**private** **static** **final** **long** *serialVersionUID* = 1L;

**public** **void** actionPerformed(ActionEvent evt) {

statusBar.setText("");

FileOutputStream fos;

**try** {

fos = **new** FileOutputStream("data/ImageTaken.jpg");

**try** {

fos.write(buffer);

statusBar.setText("Image successfully saved in your data folder");

} **catch** (IOException e) {

// **TODO** Auto-generated catch block

e.printStackTrace();

}

} **catch** (FileNotFoundException e) {

// **TODO** Auto-generated catch block

e.printStackTrace();

}

}

};

}

**return** saveImgAction;

}

}

GPS GUI

**package** GUI;

**import** java.awt.BorderLayout;

**import** java.awt.event.ActionEvent;

**import** java.io.IOException;

**import** javax.swing.AbstractAction;

**import** javax.swing.BorderFactory;

**import** javax.swing.BoxLayout;

**import** javax.swing.ImageIcon;

**import** javax.swing.JButton;

**import** javax.swing.JLabel;

**import** javax.swing.JPanel;

**import** javax.swing.SwingUtilities;

**import** javax.swing.WindowConstants;

**import** javax.swing.border.BevelBorder;

**import** domain.util.SocketManager;

/\*\*

\* This code was edited or generated using CloudGarden's Jigloo

\* SWT/Swing GUI Builder, which is free for non-commercial

\* use. If Jigloo is being used commercially (ie, by a corporation,

\* company or business for any purpose whatever) then you

\* should purchase a license for each developer using Jigloo.

\* Please visit www.cloudgarden.com for details.

\* Use of Jigloo implies acceptance of these licensing terms.

\* A COMMERCIAL LICENSE HAS NOT BEEN PURCHASED FOR

\* THIS MACHINE, SO JIGLOO OR THIS CODE CANNOT BE USED

\* LEGALLY FOR ANY CORPORATE OR COMMERCIAL PURPOSE.

\*/

**public** **class** gpsGUI **extends** javax.swing.JFrame {

/\*\*

\*

\*/

**private** **static** **final** **long** *serialVersionUID* = 1L;

**private** JLabel background;

**private** JPanel center;

**private** JPanel east;

**private** JPanel west;

**private** JPanel centercenter;

**private** JLabel statusBar;

**private** AbstractAction backAction;

**private** AbstractAction exitAction;

**private** JButton backButton;

**private** AbstractAction ONButton;

**private** JButton OFF;

**private** JButton ON;

**private** JLabel jLabel2;

**private** JLabel jLabel1;

**private** JLabel instructions;

**private** JPanel jPanel2;

**private** JPanel center3;

**private** JPanel jPanel1;

**private** JPanel north;

**private** JButton exitButton;

**private** JPanel down;

**private** JPanel up;

**private** **static** gpsGUI *window*;

**private** SocketManager manager;

**private** AbstractAction offButton;

/\*\*

\* Auto-generated main method to display this JFrame

\*/

**public** **static** **void** main(String[] args) {

SwingUtilities.*invokeLater*(**new** Runnable() {

**public** **void** run() {

*window* = **new** gpsGUI();

*window*.setLocationRelativeTo(**null**);

*window*.setVisible(**true**);

}

});

}

**public** gpsGUI() {

**super**();

initGUI();

}

**public** gpsGUI(SocketManager manager)

{

**super**();

initGUI();

**this**.manager=manager;

}

@SuppressWarnings("static-access")

**public** **void** setGUI(gpsGUI window)

{

**this**.*window*=window;

}

**private** **void** initGUI() {

**try** {

setDefaultCloseOperation(WindowConstants.*DISPOSE\_ON\_CLOSE*);

getContentPane().setLayout(**null**);

{

background = **new** JLabel();

BoxLayout backgroundLayout = **new** BoxLayout(background, javax.swing.BoxLayout.*Y\_AXIS*);

background.setLayout(backgroundLayout);

getContentPane().add(background);

background.setText(" ");

background.setBounds(0, 0, 792, 566);

background.setIcon(**new** ImageIcon("data/images/background.png"));

{

up = **new** JPanel();

background.add(up);

up.setPreferredSize(**new** java.awt.Dimension(792, 94));

up.setOpaque(**false**);

}

{

center = **new** JPanel();

BorderLayout centerLayout = **new** BorderLayout();

center.setLayout(centerLayout);

background.add(center);

center.setPreferredSize(**new** java.awt.Dimension(792, 352));

center.setOpaque(**false**);

center.add(getJPanel1(), BorderLayout.*CENTER*);

center.add(getWest(), BorderLayout.*WEST*);

center.add(getEast(), BorderLayout.*EAST*);

center.add(getNorth(), BorderLayout.*NORTH*);

}

{

down = **new** JPanel();

background.add(down);

down.setPreferredSize(**new** java.awt.Dimension(792, 88));

down.setLayout(**null**);

down.setOpaque(**false**);

{

exitButton = **new** JButton();

down.add(exitButton);

exitButton.setAction(getExitAction());

exitButton.setText(" ");

exitButton.setBounds(12, 6, 91, 80);

exitButton.setBorder(BorderFactory.*createEmptyBorder*(0, 0, 0, 0));

exitButton.setOpaque(**false**);

exitButton.setIcon(**new** ImageIcon("data/images/buttons/salir50x50.png"));

exitButton.setContentAreaFilled(**false**);

}

{

backButton = **new** JButton();

down.add(backButton);

down.add(getStatusBar());

backButton.setAction(getBackAction());

backButton.setText(" ");

backButton.setBounds(115, 20, 164, 66);

backButton.setOpaque(**false**);

backButton.setBorder(BorderFactory.*createEmptyBorder*(0, 0, 0, 0));

backButton.setIcon(**new** ImageIcon("data/images/buttons/backButton.png"));

backButton.setContentAreaFilled(**false**);

}

}

}

pack();

**this**.setSize(800, 600);

} **catch** (Exception e) {

e.printStackTrace();

}

}

**private** AbstractAction getExitAction() {

**if**(exitAction == **null**) {

exitAction = **new** AbstractAction("ONButton", **null**) {

/\*\*

\*

\*/

**private** **static** **final** **long** *serialVersionUID* = 1L;

**public** **void** actionPerformed(ActionEvent evt) {

**try** {

manager.Escribir("QUIT" + "\n");

System.*out*.println(manager.Leer());

} **catch** (IOException e) {

// **TODO** Auto-generated catch block

e.printStackTrace();

}

System.*exit*(0);

}

};

}

**return** exitAction;

}

**private** AbstractAction getBackAction() {

**if**(backAction == **null**) {

backAction = **new** AbstractAction(" ", **null**) {

/\*\*

\*

\*/

**private** **static** **final** **long** *serialVersionUID* = 1L;

**public** **void** actionPerformed(ActionEvent evt) {

menuGUI menu = **new** menuGUI(manager);

menu.setVisible(**true**);

menu.setGUI(menu);

menu.setLocationRelativeTo(**null**);

*window*.setVisible(**false**);

*window*.dispose();

}

};

}

**return** backAction;

}

**private** JLabel getStatusBar() {

**if**(statusBar == **null**) {

statusBar = **new** JLabel();

statusBar.setText("Select an Option");

statusBar.setBounds(291, 59, 501, 29);

statusBar.setOpaque(**true**);

statusBar.setFont(**new** java.awt.Font("Dialog",2,12));

statusBar.setBackground(**new** java.awt.Color(255,255,255));

statusBar.setBorder(BorderFactory.*createBevelBorder*(BevelBorder.*LOWERED*));

}

**return** statusBar;

}

**private** JPanel getJPanel1() {

**if**(centercenter == **null**) {

centercenter = **new** JPanel();

BoxLayout centercenterLayout = **new** BoxLayout(centercenter, javax.swing.BoxLayout.*Y\_AXIS*);

centercenter.setLayout(centercenterLayout);

centercenter.setPreferredSize(**new** java.awt.Dimension(772, 267));

centercenter.setOpaque(**false**);

centercenter.add(getJPanel1x());

centercenter.add(getJPanel2());

centercenter.add(getCenter3());

}

**return** centercenter;

}

**private** JPanel getWest() {

**if**(west == **null**) {

west = **new** JPanel();

west.setPreferredSize(**new** java.awt.Dimension(76, 373));

west.setOpaque(**false**);

}

**return** west;

}

**private** JPanel getEast() {

**if**(east == **null**) {

east = **new** JPanel();

east.setPreferredSize(**new** java.awt.Dimension(79, 373));

east.setOpaque(**false**);

}

**return** east;

}

**private** JPanel getNorth() {

**if**(north == **null**) {

north = **new** JPanel();

north.setOpaque(**false**);

}

**return** north;

}

**private** JPanel getJPanel1x() {

**if**(jPanel1 == **null**) {

jPanel1 = **new** JPanel();

jPanel1.setPreferredSize(**new** java.awt.Dimension(637, 106));

jPanel1.setBorder(BorderFactory.*createTitledBorder*("Instructions"));

jPanel1.setLayout(**null**);

jPanel1.setOpaque(**false**);

jPanel1.add(getInstructions());

jPanel1.add(getJLabel1());

jPanel1.add(getJLabel2());

}

**return** jPanel1;

}

**private** JPanel getCenter3() {

**if**(center3 == **null**) {

center3 = **new** JPanel();

center3.setPreferredSize(**new** java.awt.Dimension(637, 46));

center3.setOpaque(**false**);

}

**return** center3;

}

**private** JPanel getJPanel2() {

**if**(jPanel2 == **null**) {

jPanel2 = **new** JPanel();

jPanel2.setPreferredSize(**new** java.awt.Dimension(637, 222));

jPanel2.setOpaque(**false**);

jPanel2.setLayout(**null**);

jPanel2.add(getON());

jPanel2.add(getJButton1());

}

**return** jPanel2;

}

**private** JLabel getInstructions() {

**if**(instructions == **null**) {

instructions = **new** JLabel();

instructions.setText("Here you can change the state of the GPS ");

instructions.setBounds(17, 26, 596, 20);

instructions.setFont(**new** java.awt.Font("Dialog",3,16));

}

**return** instructions;

}

**private** JLabel getJLabel1() {

**if**(jLabel1 == **null**) {

jLabel1 = **new** JLabel();

jLabel1.setText("Click on the state you want to switch");

jLabel1.setFont(**new** java.awt.Font("Dialog",3,16));

jLabel1.setBounds(17, 46, 596, 21);

}

**return** jLabel1;

}

**private** JLabel getJLabel2() {

**if**(jLabel2 == **null**) {

jLabel2 = **new** JLabel();

jLabel2.setText("Check the Status Bar at the end of the Window for problems.");

jLabel2.setFont(**new** java.awt.Font("Dialog",3,16));

jLabel2.setBounds(17, 67, 596, 21);

}

**return** jLabel2;

}

**private** JButton getON() {

**if**(ON == **null**) {

ON = **new** JButton();

ON.setText(" ");

ON.setAction(getONButton());

ON.setBounds(55, 30, 157, 66);

ON.setBorder(BorderFactory.*createEmptyBorder*(0, 0, 0, 0));

ON.setOpaque(**false**);

ON.setIcon(**new** ImageIcon("data/images/buttons/onOnlyButton.png"));

ON.setContentAreaFilled(**false**);

}

**return** ON;

}

**private** JButton getJButton1() {

**if**(OFF == **null**) {

OFF = **new** JButton();

OFF.setText(" ");

OFF.setAction(getOffButton());

OFF.setBounds(55, 113, 157, 66);

OFF.setBorder(BorderFactory.*createEmptyBorder*(0, 0, 0, 0));

OFF.setOpaque(**false**);

OFF.setIcon(**new** ImageIcon("data/images/buttons/offOnlyButton.png"));

OFF.setContentAreaFilled(**false**);

}

**return** OFF;

}

**private** AbstractAction getONButton() {

**if**(ONButton == **null**) {

ONButton = **new** AbstractAction(" ", **null**) {

/\*\*

\*

\*/

**private** **static** **final** **long** *serialVersionUID* = 1L;

**public** **void** actionPerformed(ActionEvent evt)

{

**try** {

manager.Escribir("GPSON" + "\n");

String resultado = manager.Leer();

statusBar.setText(resultado);

} **catch** (IOException e) {

// **TODO** Auto-generated catch block

e.printStackTrace();

}

}

};

}

**return** ONButton;

}

**private** AbstractAction getOffButton() {

**if**(offButton == **null**) {

offButton = **new** AbstractAction(" ", **null**) {

/\*\*

\*

\*/

**private** **static** **final** **long** *serialVersionUID* = 1L;

**public** **void** actionPerformed(ActionEvent evt) {

**try** {

manager.Escribir("GPSOFF" + "\n");

String resultado = manager.Leer();

statusBar.setText(resultado);

} **catch** (IOException e) {

// **TODO** Auto-generated catch block

e.printStackTrace();

}

}

};

}

**return** offButton;

}

}

MENU GUI

**package** GUI;

**import** java.awt.event.ActionEvent;

**import** java.io.IOException;

**import** javax.swing.AbstractAction;

**import** javax.swing.BorderFactory;

**import** javax.swing.BoxLayout;

**import** javax.swing.GroupLayout;

**import** javax.swing.ImageIcon;

**import** javax.swing.JButton;

**import** javax.swing.JComponent;

**import** javax.swing.JLabel;

**import** javax.swing.JPanel;

**import** javax.swing.SwingUtilities;

**import** javax.swing.WindowConstants;

**import** javax.swing.border.BevelBorder;

**import** domain.util.SocketManager;

/\*\*

\* This code was edited or generated using CloudGarden's Jigloo

\* SWT/Swing GUI Builder, which is free for non-commercial

\* use. If Jigloo is being used commercially (ie, by a corporation,

\* company or business for any purpose whatever) then you

\* should purchase a license for each developer using Jigloo.

\* Please visit www.cloudgarden.com for details.

\* Use of Jigloo implies acceptance of these licensing terms.

\* A COMMERCIAL LICENSE HAS NOT BEEN PURCHASED FOR

\* THIS MACHINE, SO JIGLOO OR THIS CODE CANNOT BE USED

\* LEGALLY FOR ANY CORPORATE OR COMMERCIAL PURPOSE.

\*/

**public** **class** menuGUI **extends** javax.swing.JFrame {

/\*\*

\*

\*/

**private** **static** **final** **long** *serialVersionUID* = 1L;

**private** JLabel background;

**private** JPanel center;

**private** AbstractAction gpsAction;

**private** JButton exitButton;

**private** AbstractAction actionExit;

**private** JButton getPicButton;

**private** JButton gpsButton;

**private** AbstractAction sensorsAction;

**private** AbstractAction picAction;

**private** JButton sensorsButton;

**private** JLabel statusBar;

**private** JButton exit;

**private** JPanel down;

**private** JPanel up;

**private** JPanel base;

**private** **static** menuGUI *window*;

**private** SocketManager manager;

/\*\*

\* Auto-generated main method to display this JFrame

\*/

**public** **static** **void** main(String[] args) {

SwingUtilities.*invokeLater*(**new** Runnable() {

**public** **void** run() {

*window* = **new** menuGUI();

*window*.setLocationRelativeTo(**null**);

*window*.setVisible(**true**);

}

});

}

**public** menuGUI() {

**super**();

initGUI();

}

**public** menuGUI(SocketManager manager)

{

**super**();

initGUI();

**this**.manager=manager;

}

@SuppressWarnings("static-access")

**public** **void** setGUI(menuGUI window)

{

**this**.*window*=window;

}

**private** **void** initGUI() {

**try** {

setDefaultCloseOperation(WindowConstants.*DISPOSE\_ON\_CLOSE*);

getContentPane().setLayout(**null**);

{

background = **new** JLabel();

getContentPane().add(background);

background.setText(" ");

background.setBounds(0, 0, 792, 566);

background.setIcon(**new** ImageIcon("data/images/background.png"));

{

base = **new** JPanel();

BoxLayout baseLayout = **new** BoxLayout(base, javax.swing.BoxLayout.*Y\_AXIS*);

base.setLayout(baseLayout);

background.add(base);

base.setBounds(0, 0, 792, 566);

base.setOpaque(**false**);

{

up = **new** JPanel();

base.add(up);

up.setOpaque(**false**);

}

{

center = **new** JPanel();

GroupLayout centerLayout = **new** GroupLayout((JComponent)center);

center.setLayout(centerLayout);

base.add(center);

center.setPreferredSize(**new** java.awt.Dimension(792, 440));

center.setOpaque(**false**);

{

getPicButton = **new** JButton();

getPicButton.setAction(getPicAction());

getPicButton.setText(" ");

getPicButton.setOpaque(**false**);

getPicButton.setBorder(BorderFactory.*createEmptyBorder*(0, 0, 0, 0));

getPicButton.setIcon(**new** ImageIcon("data/images/buttons/getPicButton.png"));

getPicButton.setContentAreaFilled(**false**);

}

{

sensorsButton = **new** JButton();

sensorsButton.setAction(getSensorsAction());

sensorsButton.setText(" ");

sensorsButton.setBorder(BorderFactory.*createEmptyBorder*(0, 0, 0, 0));

sensorsButton.setOpaque(**false**);

sensorsButton.setIcon(**new** ImageIcon("data/images/buttons/sensorsButton.png"));

sensorsButton.setContentAreaFilled(**false**);

}

{

gpsButton = **new** JButton();

gpsButton.setAction(getGpsAction());

gpsButton.setText(" ");

gpsButton.setBorder(BorderFactory.*createEmptyBorder*(0, 0, 0, 0));

gpsButton.setOpaque(**false**);

gpsButton.setIcon(**new** ImageIcon("data/images/buttons/gps.png"));

gpsButton.setContentAreaFilled(**false**);

}

centerLayout.setHorizontalGroup(centerLayout.createSequentialGroup()

.addContainerGap(116, 116)

.addGroup(centerLayout.createParallelGroup()

.addGroup(centerLayout.createSequentialGroup()

.addComponent(getPicButton, GroupLayout.*PREFERRED\_SIZE*, 172, GroupLayout.*PREFERRED\_SIZE*)

.addGap(0, 0, Short.*MAX\_VALUE*))

.addComponent(sensorsButton, GroupLayout.Alignment.*LEADING*, 0, 172, Short.*MAX\_VALUE*)

.addGroup(centerLayout.createSequentialGroup()

.addGap(0, 0, Short.*MAX\_VALUE*)

.addComponent(gpsButton, GroupLayout.*PREFERRED\_SIZE*, 172, GroupLayout.*PREFERRED\_SIZE*)))

.addContainerGap(504, 504));

centerLayout.setVerticalGroup(centerLayout.createSequentialGroup()

.addContainerGap(113, 113)

.addComponent(getPicButton, GroupLayout.*PREFERRED\_SIZE*, 51, GroupLayout.*PREFERRED\_SIZE*)

.addGap(37)

.addComponent(sensorsButton, GroupLayout.*PREFERRED\_SIZE*, 49, GroupLayout.*PREFERRED\_SIZE*)

.addGap(50)

.addComponent(gpsButton, GroupLayout.*PREFERRED\_SIZE*, 49, GroupLayout.*PREFERRED\_SIZE*)

.addContainerGap(92, Short.*MAX\_VALUE*));

}

{

down = **new** JPanel();

down.setLayout(**null**);

base.add(down);

down.setPreferredSize(**new** java.awt.Dimension(445, 92));

down.setOpaque(**false**);

{

exit = **new** JButton();

down.add(exit);

exit.setAction(getActionExit()); exit.setBounds(34, 21, 113, 61);

exit.setContentAreaFilled(**false**);

exit.setBorder(BorderFactory.*createEmptyBorder*(0, 0, 0, 0));

exit.setIcon(**new** ImageIcon("data/images/buttons/salir50x50.png"));

exit.setOpaque(**false**);

exit.setVisible(**false**);

}

{

statusBar = **new** JLabel();

down.add(statusBar);

down.add(getExitButton());

statusBar.setText("Select an Option");

statusBar.setBounds(170, 70, 622, 30);

statusBar.setBackground(**new** java.awt.Color(255,255,255));

statusBar.setBorder(BorderFactory.*createBevelBorder*(BevelBorder.*LOWERED*));

statusBar.setOpaque(**true**);

statusBar.setFont(**new** java.awt.Font("Dialog",2,12));

}

}

}

}

pack();

**this**.setSize(800, 600);

} **catch** (Exception e) {

e.printStackTrace();

}

}

**private** AbstractAction getActionExit() {

**if**(actionExit == **null**) {

actionExit = **new** AbstractAction(" ", **null**) {

/\*\*

\*

\*/

**private** **static** **final** **long** *serialVersionUID* = 1L;

**public** **void** actionPerformed(ActionEvent evt) {

**try** {

manager.Escribir("QUIT" + "\n");

System.*out*.println(manager.Leer());

} **catch** (IOException e) {

// **TODO** Auto-generated catch block

e.printStackTrace();

}

System.*exit*(0);

}

};

}

**return** actionExit;

}

**private** JButton getExitButton() {

**if**(exitButton == **null**) {

exitButton = **new** JButton();

exitButton.setAction(getActionExit());

exitButton.setText(" ");

exitButton.setContentAreaFilled(**false**);

exitButton.setIcon(**new** ImageIcon("data/images/buttons/salir50x50.png"));

exitButton.setOpaque(**false**);

exitButton.setBorder(BorderFactory.*createEmptyBorder*(0, 0, 0, 0));

exitButton.setBounds(0, 0, 118, 70);

}

**return** exitButton;

}

**private** AbstractAction getGpsAction() {

**if**(gpsAction == **null**) {

gpsAction = **new** AbstractAction(" ", **null**) {

/\*\*

\*

\*/

**private** **static** **final** **long** *serialVersionUID* = 1L;

**public** **void** actionPerformed(ActionEvent evt) {

gpsGUI gps= **new** gpsGUI(manager);

gps.setGUI(gps);

gps.setVisible(**true**);

gps.setLocationRelativeTo(**null**);

*window*.setVisible(**false**);

*window*.dispose();

}

};

}

**return** gpsAction;

}

**private** AbstractAction getPicAction() {

**if**(picAction == **null**) {

picAction = **new** AbstractAction(" ", **null**) {

/\*\*

\*

\*/

**private** **static** **final** **long** *serialVersionUID* = 1L;

**public** **void** actionPerformed(ActionEvent evt) {

getPicGUI getPic= **new** getPicGUI(manager);

getPic.setGUI(getPic);

getPic.setVisible(**true**);

getPic.setLocationRelativeTo(**null**);

*window*.setVisible(**false**);

*window*.dispose();

}

};

}

**return** picAction;

}

**private** AbstractAction getSensorsAction() {

**if**(sensorsAction == **null**) {

sensorsAction = **new** AbstractAction(" ", **null**) {

/\*\*

\*

\*/

**private** **static** **final** **long** *serialVersionUID* = 1L;

**public** **void** actionPerformed(ActionEvent evt) {

sensorsGUI sensor= **new** sensorsGUI(manager);

sensor.setGUI(sensor);

sensor.setVisible(**true**);

sensor.setLocationRelativeTo(**null**);

*window*.setVisible(**false**);

*window*.dispose();

}

};

}

**return** sensorsAction;

}

}

SENSORS GUI

**package** GUI;

**import** java.awt.event.ActionEvent;

**import** java.io.IOException;

**import** javax.swing.AbstractAction;

**import** javax.swing.BorderFactory;

**import** javax.swing.BoxLayout;

**import** javax.swing.DebugGraphics;

**import** javax.swing.GroupLayout;

**import** javax.swing.ImageIcon;

**import** javax.swing.JButton;

**import** javax.swing.JComponent;

**import** javax.swing.JLabel;

**import** javax.swing.JPanel;

**import** javax.swing.JScrollPane;

**import** javax.swing.JTable;

**import** javax.swing.JTextField;

**import** javax.swing.SwingUtilities;

**import** javax.swing.WindowConstants;

**import** javax.swing.border.BevelBorder;

**import** javax.swing.border.LineBorder;

**import** javax.swing.table.DefaultTableModel;

**import** javax.swing.table.TableModel;

**import** domain.util.SocketManager;

/\*\*

\* This code was edited or generated using CloudGarden's Jigloo

\* SWT/Swing GUI Builder, which is free for non-commercial

\* use. If Jigloo is being used commercially (ie, by a corporation,

\* company or business for any purpose whatever) then you

\* should purchase a license for each developer using Jigloo.

\* Please visit www.cloudgarden.com for details.

\* Use of Jigloo implies acceptance of these licensing terms.

\* A COMMERCIAL LICENSE HAS NOT BEEN PURCHASED FOR

\* THIS MACHINE, SO JIGLOO OR THIS CODE CANNOT BE USED

\* LEGALLY FOR ANY CORPORATE OR COMMERCIAL PURPOSE.

\*/

**public** **class** sensorsGUI **extends** javax.swing.JFrame {

/\*\*

\*

\*/

**private** **static** **final** **long** *serialVersionUID* = 1L;

**private** JLabel background;

**private** JPanel center;

**private** JPanel two;

**private** JPanel one;

**private** JLabel statusBar;

**private** AbstractAction backAction;

**private** AbstractAction exitAction;

**private** JButton backButton;

**private** JScrollPane jScrollPane1;

**private** JTextField valueField;

**private** JTextField coordField;

**private** AbstractAction currentValue;

**private** JTextField horaField;

**private** JButton jButton2;

**private** JButton jButton1;

**private** JTable table;

**private** JPanel left;

**private** JPanel three;

**private** JButton exitButton;

**private** JPanel down;

**private** JPanel up;

**private** **static** sensorsGUI *window*;

**private** SocketManager manager;

**private** JLabel jLabel5;

**private** JLabel jLabel4;

**private** JLabel dialog;

**private** JPanel jPanel2;

**private** JPanel jPanel1;

**private** JPanel RIGHT;

**private** JPanel LEFT;

**private** JButton jButton3;

**private** AbstractAction OFF;

**private** AbstractAction ON;

**private** JButton OFFButton;

**private** JButton ONButton;

**private** AbstractAction history;

**private** JTable table2;

**private** JScrollPane jScrollPane2;

**private** JLabel jLabel3;

**private** JTextField dateField;

**private** JLabel jLabel2;

**private** JLabel jLabel1;

**private** JLabel Hora;

/\*\*

\* Auto-generated main method to display this JFrame

\*/

**public** **static** **void** main(String[] args) {

SwingUtilities.*invokeLater*(**new** Runnable() {

**public** **void** run() {

*window* = **new** sensorsGUI();

*window*.setLocationRelativeTo(**null**);

*window*.setVisible(**true**);

}

});

}

**public** sensorsGUI() {

**super**();

initGUI();

}

**public** sensorsGUI(SocketManager manager)

{

**super**();

initGUI();

**this**.manager=manager;

String result;

DefaultTableModel tablareh = (DefaultTableModel)**this**.table.getModel();

**try** {

manager.Escribir("LISTSENSOR" + "\n");

System.*out*.println(manager.Leer());

**int** size = Integer.*parseInt*(manager.Leer());

tablareh.setNumRows(size);

**for**(**int** i=0;i<size;i++)

{

result = manager.Leer();

String[] field = result.split(";;");

**for** (**int** x=0; x<field.length; x++)

{

tablareh.setValueAt(field[x],i,x);

}

}

statusBar.setText(manager.Leer());

} **catch** (IOException e) {

// **TODO** Auto-generated catch block

e.printStackTrace();

}

}

**private** **void** cleanTable()

{

DefaultTableModel tablareh = (DefaultTableModel)table2.getModel();

**int** rows=tablareh.getRowCount();

**for** (**int** i=0;i<rows;i++)

{

**for** (**int** x=0;x<=3;x++)

{

tablareh.setValueAt("", i, x);

}

}

}

@SuppressWarnings("static-access")

**public** **void** setGUI(sensorsGUI window)

{

**this**.*window*=window;

}

**private** **void** initGUI() {

**try** {

setDefaultCloseOperation(WindowConstants.*DISPOSE\_ON\_CLOSE*);

getContentPane().setLayout(**null**);

{

background = **new** JLabel();

BoxLayout backgroundLayout = **new** BoxLayout(background, javax.swing.BoxLayout.*Y\_AXIS*);

background.setLayout(backgroundLayout);

getContentPane().add(background);

background.setText(" ");

background.setBounds(0, 0, 792, 566);

background.setIcon(**new** ImageIcon("data/images/background.png"));

{

up = **new** JPanel();

background.add(up);

up.setOpaque(**false**);

up.setPreferredSize(**new** java.awt.Dimension(792, 109));

}

{

center = **new** JPanel();

BoxLayout centerLayout = **new** BoxLayout(center, javax.swing.BoxLayout.*Y\_AXIS*);

center.setLayout(centerLayout);

background.add(center);

center.setPreferredSize(**new** java.awt.Dimension(792, 463));

center.setOpaque(**false**);

center.add(getOne());

center.add(getTwo());

center.add(getThree());

}

{

down = **new** JPanel();

background.add(down);

down.setLayout(**null**);

down.setOpaque(**false**);

down.setPreferredSize(**new** java.awt.Dimension(792, 63));

{

exitButton = **new** JButton();

down.add(getStatusBar());

down.add(exitButton);

exitButton.setAction(getExitAction());

exitButton.setText(" ");

exitButton.setBounds(12, -9, 77, 64);

exitButton.setOpaque(**false**);

exitButton.setBorder(BorderFactory.*createEmptyBorder*(0, 0, 0, 0));

exitButton.setIcon(**new** ImageIcon("data/images/buttons/salir50x50.png"));

exitButton.setContentAreaFilled(**false**);

}

{

backButton = **new** JButton();

down.add(backButton);

backButton.setText(" ");

backButton.setAction(getBackAction());

backButton.setBorder(BorderFactory.*createEmptyBorder*(0, 0, 0, 0));

backButton.setOpaque(**false**);

backButton.setIcon(**new** ImageIcon("data/images/buttons/backButton.png"));

backButton.setContentAreaFilled(**false**);

backButton.setBounds(122, -9, 216, 65);

}

}

}

pack();

**this**.setSize(800, 600);

} **catch** (Exception e) {

e.printStackTrace();

}

}

**private** AbstractAction getExitAction() {

**if**(exitAction == **null**) {

exitAction = **new** AbstractAction(" ", **null**) {

/\*\*

\*

\*/

**private** **static** **final** **long** *serialVersionUID* = 1L;

**public** **void** actionPerformed(ActionEvent evt) {

**try** {

manager.Escribir("QUIT" + "\n");

System.*out*.println(manager.Leer());

} **catch** (IOException e) {

// **TODO** Auto-generated catch block

e.printStackTrace();

}

System.*exit*(0);

}

};

}

**return** exitAction;

}

**private** AbstractAction getBackAction() {

**if**(backAction == **null**) {

backAction = **new** AbstractAction(" ", **null**) {

/\*\*

\*

\*/

**private** **static** **final** **long** *serialVersionUID* = 1L;

**public** **void** actionPerformed(ActionEvent evt) {

**try** {

manager.Escribir("RETURN" + "\n");

System.*out*.println(manager.Leer());

} **catch** (IOException e) {

// **TODO** Auto-generated catch block

e.printStackTrace();

}

menuGUI menu = **new** menuGUI(manager);

menu.setVisible(**true**);

menu.setGUI(menu);

menu.setLocationRelativeTo(**null**);

*window*.setVisible(**false**);

*window*.dispose();

}

};

}

**return** backAction;

}

**private** JLabel getStatusBar() {

**if**(statusBar == **null**) {

statusBar = **new** JLabel();

statusBar.setText(" ");

statusBar.setBounds(371, 30, 419, 25);

statusBar.setBackground(**new** java.awt.Color(255,255,255));

statusBar.setBorder(BorderFactory.*createBevelBorder*(BevelBorder.*LOWERED*));

statusBar.setOpaque(**true**);

statusBar.setFont(**new** java.awt.Font("Dialog",2,11));

}

**return** statusBar;

}

**private** JPanel getOne() {

**if**(one == **null**) {

one = **new** JPanel();

BoxLayout oneLayout = **new** BoxLayout(one, javax.swing.BoxLayout.*X\_AXIS*);

one.setLayout(oneLayout);

one.setPreferredSize(**new** java.awt.Dimension(792, 235));

one.setOpaque(**false**);

one.add(getLeft());

}

**return** one;

}

**private** JPanel getTwo() {

**if**(two == **null**) {

two = **new** JPanel();

BoxLayout twoLayout = **new** BoxLayout(two, javax.swing.BoxLayout.*X\_AXIS*);

two.setPreferredSize(**new** java.awt.Dimension(792, 125));

two.setOpaque(**false**);

two.setLayout(twoLayout);

two.add(getLEFT());

two.add(getRIGHT());

}

**return** two;

}

**private** JPanel getThree() {

**if**(three == **null**) {

three = **new** JPanel();

GroupLayout threeLayout = **new** GroupLayout((JComponent)three);

three.setLayout(threeLayout);

three.setOpaque(**false**);

three.setPreferredSize(**new** java.awt.Dimension(792, 10));

threeLayout.setVerticalGroup(threeLayout.createParallelGroup(GroupLayout.Alignment.*BASELINE*)

.addComponent(getJButton3(), GroupLayout.Alignment.*BASELINE*, GroupLayout.*PREFERRED\_SIZE*, GroupLayout.*PREFERRED\_SIZE*, GroupLayout.*PREFERRED\_SIZE*));

threeLayout.setHorizontalGroup(threeLayout.createSequentialGroup()

.addContainerGap(138, 138)

.addComponent(getJButton3(), GroupLayout.*PREFERRED\_SIZE*, GroupLayout.*PREFERRED\_SIZE*, GroupLayout.*PREFERRED\_SIZE*)

.addContainerGap(389, Short.*MAX\_VALUE*));

}

**return** three;

}

**private** JPanel getLeft() {

**if**(left == **null**) {

left = **new** JPanel();

left.setPreferredSize(**new** java.awt.Dimension(639, 190));

left.setLayout(**null**);

left.setOpaque(**false**);

left.add(getJScrollPane1());

left.add(getJTextField1());

left.add(getJTextField1x());

left.add(getJTextField2());

left.add(getHora());

left.add(getJLabel1());

left.add(getJLabel2());

left.add(getJTextField3());

left.add(getJLabel3());

left.add(getONButton());

left.add(getOFFButton());

left.add(getJButton1());

}

**return** left;

}

**private** JScrollPane getJScrollPane1() {

**if**(jScrollPane1 == **null**) {

jScrollPane1 = **new** JScrollPane();

jScrollPane1.setBounds(39, 49, 470, 73);

jScrollPane1.setFont(**new** java.awt.Font("High Tower Text",1,12));

jScrollPane1.setDebugGraphicsOptions(DebugGraphics.*BUFFERED\_OPTION*);

jScrollPane1.setBorder(BorderFactory.*createMatteBorder*(1, 1, 1, 1, **new** java.awt.Color(0,0,0)));

jScrollPane1.setViewportView(getTable());

}

**return** jScrollPane1;

}

**private** JTable getTable() {

**if**(table == **null**) {

TableModel tableModel =

**new** DefaultTableModel(

**new** String[][] { { "", "" }, { "", "" } },

**new** String[] { "ID", "Descripcion","State"});

table = **new** JTable();

table.setModel(tableModel);

table.setFont(**new** java.awt.Font("Dialog",0,14));

}

**return** table;

}

**private** JButton getJButton1() {

**if**(jButton1 == **null**) {

jButton1 = **new** JButton();

jButton1.setText(" ");

jButton1.setBounds(574, 89, 144, 54);

jButton1.setAction(getCurrentValue());

jButton1.setIcon(**new** ImageIcon("data/images/buttons/curValue.png"));

jButton1.setContentAreaFilled(**false**);

jButton1.setBorder(BorderFactory.*createEmptyBorder*(0, 0, 0, 0));

jButton1.setOpaque(**false**);

}

**return** jButton1;

}

**private** JButton getJButton2() {

**if**(jButton2 == **null**) {

jButton2 = **new** JButton();

jButton2.setText(" ");

jButton2.setBounds(133, 5, 193, 47);

jButton2.setAction(getHistory());

jButton2.setIcon(**new** ImageIcon("data/images/buttons/history.png"));

jButton2.setContentAreaFilled(**false**);

jButton2.setOpaque(**false**);

jButton2.setBorder(BorderFactory.*createEmptyBorder*(0, 0, 0, 0));

}

**return** jButton2;

}

**private** JTextField getJTextField1() {

**if**(horaField == **null**) {

horaField = **new** JTextField();

horaField.setBounds(271, 165, 82, 26);

}

**return** horaField;

}

**private** AbstractAction getCurrentValue() {

**if**(currentValue == **null**) {

currentValue = **new** AbstractAction("", **null**) {

/\*\*

\*

\*/

**private** **static** **final** **long** *serialVersionUID* = 1L;

**public** **void** actionPerformed(ActionEvent evt) {

statusBar.setText("");

**try** {

**int** row= table.getSelectedRow();

String id;

**try**

{

id = (String) table.getValueAt(row,0);

}**catch**(Exception e)

{

id="";

}

manager.Escribir("GET\_CURVALUE "+id+"" + "\n");

String currentValue=manager.Leer();

String[] values=currentValue.split(";");

**if** (values[0].contains("114"))

{

statusBar.setText(values[0]);

dateField.setText(values[1]);

horaField.setText(values[2]);

coordField.setText(values[3]);

valueField.setText(values[4]);

}

**else**

{

statusBar.setText(currentValue);

}

} **catch** (IOException e) {

// **TODO** Auto-generated catch block

e.printStackTrace();

}

}

};

}

**return** currentValue;

}

**private** JTextField getJTextField1x() {

**if**(coordField == **null**) {

coordField = **new** JTextField();

coordField.setBounds(426, 165, 191, 27);

}

**return** coordField;

}

**private** JTextField getJTextField2() {

**if**(valueField == **null**) {

valueField = **new** JTextField();

valueField.setBounds(685, 165, 95, 27);

}

**return** valueField;

}

**private** JLabel getHora() {

**if**(Hora == **null**) {

Hora = **new** JLabel();

Hora.setText("Date");

Hora.setBounds(39, 164, 49, 30);

Hora.setFont(**new** java.awt.Font("Dialog",1,18));

}

**return** Hora;

}

**private** JLabel getJLabel1() {

**if**(jLabel1 == **null**) {

jLabel1 = **new** JLabel();

jLabel1.setText("Value");

jLabel1.setFont(**new** java.awt.Font("Dialog",1,18));

jLabel1.setBounds(629, 167, 50, 25);

}

**return** jLabel1;

}

**private** JLabel getJLabel2() {

**if**(jLabel2 == **null**) {

jLabel2 = **new** JLabel();

jLabel2.setText("Coord");

jLabel2.setFont(**new** java.awt.Font("Dialog",1,18));

jLabel2.setBounds(365, 164, 55, 30);

}

**return** jLabel2;

}

**private** JTextField getJTextField3() {

**if**(dateField == **null**) {

dateField = **new** JTextField();

dateField.setBounds(88, 166, 82, 25);

}

**return** dateField;

}

**private** JLabel getJLabel3() {

**if**(jLabel3 == **null**) {

jLabel3 = **new** JLabel();

jLabel3.setText("Hour");

jLabel3.setFont(**new** java.awt.Font("Dialog",1,18));

jLabel3.setBounds(216, 167, 49, 24);

}

**return** jLabel3;

}

**private** JScrollPane getJScrollPane2() {

**if**(jScrollPane2 == **null**) {

jScrollPane2 = **new** JScrollPane();

jScrollPane2.setBounds(-74, 0, 411, 146);

jScrollPane2.setViewportView(getJTable1());

}

**return** jScrollPane2;

}

**private** JTable getJTable1() {

**if**(table2 == **null**) {

TableModel jTable1Model =

**new** DefaultTableModel(

**new** String[][] { { "", "" }, { "", "" } },

**new** String[] { "Date", "Hour","Coord","Value" });

table2 = **new** JTable();

table2.setModel(jTable1Model);

}

**return** table2;

}

**private** AbstractAction getHistory() {

**if**(history == **null**) {

history = **new** AbstractAction(" ", **null**) {

/\*\*

\*

\*/

**private** **static** **final** **long** *serialVersionUID* = 1L;

**public** **void** actionPerformed(ActionEvent evt) {

statusBar.setText("");

cleanTable();

String resultado;

**try** {

**int** row= table.getSelectedRow();

String id;

**try**

{

id = (String) table.getValueAt(row,0);

}**catch**(Exception e)

{

id="";

}

manager.Escribir("HISTORYLOG "+id+"" + "\n");

String response=manager.Leer();

System.*out*.println("Response: "+response);

**if**(response.contains("113"))

{

**int** size2 = Integer.*parseInt*(manager.Leer());

System.*out*.println("Size: "+size2);

**if**(size2==0)

{

statusBar.setText(manager.Leer());

}

**else**

{

DefaultTableModel tablareh = (DefaultTableModel)table2.getModel();

tablareh.setNumRows(size2);

**for**(**int** i=0;i<size2;i++)

{

resultado = manager.Leer();

String[] values=resultado.split(";");

tablareh.setNumRows(size2);

**for** (**int** x=0; x<values.length; x++)

{

tablareh.setValueAt(values[x],i,x);

}

}

statusBar.setText(manager.Leer());

}

}

**else**

{

statusBar.setText(response);

}

} **catch** (IOException e) {

// **TODO** Auto-generated catch block

e.printStackTrace();

}

}

};

}

**return** history;

}

**private** JButton getONButton() {

**if**(ONButton == **null**) {

ONButton = **new** JButton();

ONButton.setText(" ");

ONButton.setBounds(512, 40, 147, 61);

ONButton.setAction(getON());

ONButton.setIcon(**new** ImageIcon("data/images/buttons/onSensor.png"));

ONButton.setContentAreaFilled(**false**);

ONButton.setBorder(BorderFactory.*createEmptyBorder*(0, 0, 0, 0));

ONButton.setOpaque(**false**);

}

**return** ONButton;

}

**private** JButton getOFFButton() {

**if**(OFFButton == **null**) {

OFFButton = **new** JButton();

OFFButton.setBounds(647, 45, 146, 55);

OFFButton.setAction(getOFF());

OFFButton.setIcon(**new** ImageIcon("data/images/buttons/offSensor.png"));

OFFButton.setContentAreaFilled(**false**);

OFFButton.setBorder(BorderFactory.*createEmptyBorder*(0, 0, 0, 0));

OFFButton.setOpaque(**false**);

}

**return** OFFButton;

}

**private** AbstractAction getON() {

**if**(ON == **null**) {

ON = **new** AbstractAction(" ", **null**) {

/\*\*

\*

\*/

**private** **static** **final** **long** *serialVersionUID* = 1L;

**public** **void** actionPerformed(ActionEvent evt) {

statusBar.setText("");

**try** {

**int** row= table.getSelectedRow();

String id;

**try**

{

id = (String) table.getValueAt(row,0);

}**catch**(Exception e)

{

id="";

}

manager.Escribir("ON "+id+"" + "\n");

String response=manager.Leer();

**if**(response.contains("203"))

{

table.setValueAt("ON", row, 2);

}

statusBar.setText(response);

} **catch** (IOException e) {

// **TODO** Auto-generated catch block

e.printStackTrace();

}

}

};

}

**return** ON;

}

**private** AbstractAction getOFF() {

**if**(OFF == **null**) {

OFF = **new** AbstractAction(" ", **null**) {

/\*\*

\*

\*/

**private** **static** **final** **long** *serialVersionUID* = 1L;

**public** **void** actionPerformed(ActionEvent evt) {

statusBar.setText("");

**try** {

**int** row= table.getSelectedRow();

String id;

**try**

{

id = (String) table.getValueAt(row,0);

}**catch**(Exception e)

{

id="";

}

manager.Escribir("OFF "+id+"" + "\n");

String response=manager.Leer();

**if**(response.contains("204"))

{

table.setValueAt("OFF", row, 2);

}

statusBar.setText(response);

} **catch** (IOException e) {

// **TODO** Auto-generated catch block

e.printStackTrace();

}

}

};

}

**return** OFF;

}

**private** JButton getJButton3() {

**if**(jButton3 == **null**) {

jButton3 = **new** JButton();

jButton3.setText("jButton3");

jButton3.setOpaque(**false**);

jButton3.setEnabled(**false**);

jButton3.setVisible(**false**);

}

**return** jButton3;

}

**private** JPanel getLEFT() {

**if**(LEFT == **null**) {

LEFT = **new** JPanel();

BoxLayout LEFTLayout = **new** BoxLayout(LEFT, javax.swing.BoxLayout.*Y\_AXIS*);

LEFT.setPreferredSize(**new** java.awt.Dimension(334, 146));

LEFT.setLayout(LEFTLayout);

LEFT.setOpaque(**false**);

LEFT.add(getJPanel1());

LEFT.add(getJPanel2());

}

**return** LEFT;

}

**private** JPanel getRIGHT() {

**if**(RIGHT == **null**) {

RIGHT = **new** JPanel();

RIGHT.setOpaque(**false**);

RIGHT.add(getJScrollPane2());

}

**return** RIGHT;

}

**private** JPanel getJPanel1() {

**if**(jPanel1 == **null**) {

jPanel1 = **new** JPanel();

jPanel1.setOpaque(**false**);

jPanel1.setPreferredSize(**new** java.awt.Dimension(331, 98));

jPanel1.setLayout(**null**);

jPanel1.setBorder(**new** LineBorder(**new** java.awt.Color(0,0,0), 1, **false**));

jPanel1.add(getDialog());

jPanel1.add(getJLabel4());

jPanel1.add(getJLabel5());

}

**return** jPanel1;

}

**private** JPanel getJPanel2() {

**if**(jPanel2 == **null**) {

jPanel2 = **new** JPanel();

jPanel2.setOpaque(**false**);

jPanel2.setPreferredSize(**new** java.awt.Dimension(331, 59));

jPanel2.setLayout(**null**);

jPanel2.add(getJButton2());

}

**return** jPanel2;

}

**private** JLabel getDialog() {

**if**(dialog == **null**) {

dialog = **new** JLabel();

dialog.setText("Choose a sensor in the table above");

dialog.setFont(**new** java.awt.Font("Dialog",3,16));

dialog.setBounds(21, 6, 283, 19);

}

**return** dialog;

}

**private** JLabel getJLabel4() {

**if**(jLabel4 == **null**) {

jLabel4 = **new** JLabel();

jLabel4.setText("Then, choose an action to perform");

jLabel4.setFont(**new** java.awt.Font("Dialog",3,16));

jLabel4.setBounds(21, 31, 283, 20);

}

**return** jLabel4;

}

**private** JLabel getJLabel5() {

**if**(jLabel5 == **null**) {

jLabel5 = **new** JLabel();

jLabel5.setText("Check the StatusBar for errors");

jLabel5.setFont(**new** java.awt.Font("Dialog",3,16));

jLabel5.setBounds(21, 57, 283, 21);

}

**return** jLabel5;

}

}