Documentación PROA 09-10

Índice de contenido

Manual del usuario

JavaTrack esta compuesto por tres aplicaciones

El SBD o servidor de objetos RMI

Ejecución:
java -cp Escritorio/Proa/Netbeans/sBD/dist/sBD.jar:mysql-connector-java-5.1.11-bin.j
-Djava.security.policy=Escritorio/Proa/Netbeans/sBD/client.policy
-Djava.rmi.server.codebase=http://localhost/~luzem/sJavaTrack.jar sbd.ComputeEngine

Se indica el archivo de políticas de seguridad: client.policy
Las clases de los comandos deberán de estar en un servidor apache para que el servidor RMI resuelva las clases de los comandos que no conoce.

El sJavaTrack o servidor de sockets

Ejecución :
java -jar -Djava.security.policy=Escritorio/Proa/Netbeans/sJavaTrack/client.policy
Escritorio/Proa/Netbeans/sJavaTrack/dist/sJavaTrack.jar 12000

Se indica el archivo de politicas de seguridad: client.policy
Se indica el puerto donde debe de escuchar el servidor de sockets en este caso el 12000

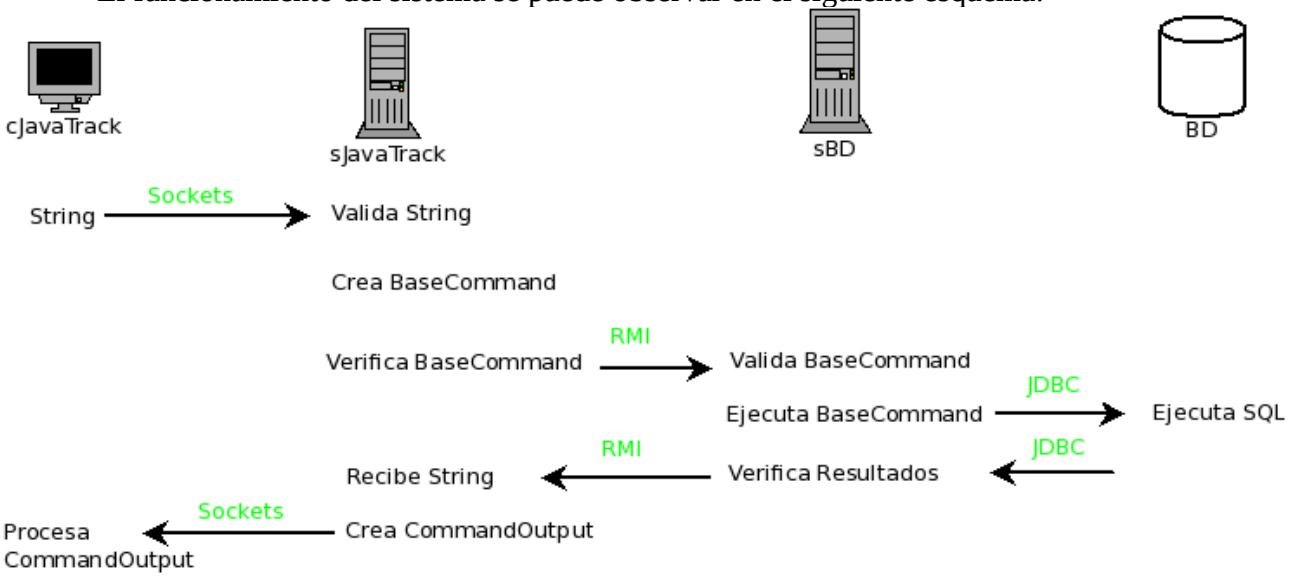
El cJavaTrack o cliente de Sockets
Ejecución: java -jar Escritorio/Proa/Netbeans/cJavaTrack/dist/cJavaTrack.jar localhost 1200

Se especifica el host y puerto del servidor de sockets

Para lista de comandos ver proa0910.pdf

Documentación Técnica

El funcionamiento del sistema se puede observar en el siguiente esquema:



Cada vez que un componente del sistema recibe información esta es verificada, en caso de error se salta el envió al siguiente componente y se genera una respuesta de error.

Comunicación cJavaTrack sJavaTrack

Se envia texto a traves de los sockets, sJavaTrack interpreta el texto y usando reflection crea un objeto que tiene el mismo nombre que el comando, todos estos objetos heredan de BaseCommand, llevan una implementación propia del metodo execute() y de su constructor modificando las siguiente variables.

- String help = una cadena de texto que contiene la ayuda
- boolean isUserCommand = booleano que indica si es un comando de usuario
- boolean isAdminCommand= booleano que indica si es un comando ejecutable por un administrador
- boolean toFile= indica si el resultado se debe guardar en un archivo
- String File= nombre del archivo donde se guardara el resultado

Los parametros del comando son bombeados en el Vector parameterList que deberan ser procesador y verificados en el metodo execute() el usuario y contraseña están introducidos en las variables userName y password .

Los métodos disponibles se pueden encontrar en la javadoc de BaseCommand en sBD.

Comunicación sJavaTrack sBD

sJavaTrack enviá el objeto BaseCommand a sBD y le manda ejecutar las funciones de verificación de permisos (remotePermisionsCheck(obtainedCommand)) y en caso de ser validos llama al método remoteExecute(obtainedCommand) utilizando un objeto ComputeEngine como un contenedor base para dotar a los BaseCommand de capacidades de ejecución remota.

Comunicación sBD BD

Se comunica a través de JDBC con el driver

Comunicación BD sBD

A través de JDBC con el driver

Comunicación sBD sJavaTrack

Las respuestas de los métodos execute son Strings que se utilizaran para mostrar en la consola del cliente se obtienen por comunicación a través de RMI.

Comunicación sJavaTrack cJavaTrack

el sJavaTrack crea un objeto CommandResponse con la cadena de texto de la ejecución remota y las variables que afectan a las operaciones de ficheros

Finalmente el cJavaTrack con la información de CommandResponse muestra el resultado por pantalla o lo almacena en un fichero.

Ampliación de Comandos

Se han implementado **todos** los comandos tras comprender la estructura la creación de un commando es trivial.

Código Fuente

SDB

```
/*
 * To change this template, choose Tools | Templates
 * and open the template in the editor.
 */

package sbd;

import java.rmi.Remote;
import java.rmi.RemoteException;


/**
 *
 * @author luzem
 */
public interface Compute extends Remote{

    /**
     * executes a base command
     * @param bc the base command
     * @return the base command execution
     * @throws RemoteException
     */
    public String execute(baseCommand bc) throws RemoteException;
    /**
     * check if the baseCommand has permisions to be executed
     * @param bc the baseCommand
     * @return true if the username has permisions to execute the command
     * @throws RemoteException
     */
    public boolean  hasPermisions(baseCommand bc) throws RemoteException;

    /**
     * Verifies the baseCommand login
     * @param bc the base command
     * @return true if the login has been verified
     * @throws RemoteException
     */
    public boolean  verifyLogin (baseCommand bc) throws RemoteException;
}

/*
 * To change this template, choose Tools | Templates
 * and open the template in the editor.
 */
package sbd;
```

```
import java.rmi.Naming;
import java.rmi.RemoteException;
import java.rmi.registry.LocateRegistry;
import java.rmi.server.UnicastRemoteObject;



/**
 *
 * @author luzem
 */
public class ComputeEngine extends UnicastRemoteObject implements Compute {

    /**
     * creates a new CompueEngine
     * @throws RemoteException
     */
    public ComputeEngine() throws RemoteException{
        super();
    }

    public boolean verifyLogin(baseCommand bc)
            {
            return bc.verifyLogin();
            }

    public String execute(baseCommand bc) {
        return bc.execute();
    }

    public boolean hasPermisions(baseCommand bc) {
        return bc.havePermisionstoExecuteCommand();
    }

    /**
     * launches a new sBD
     * @param args not used
     */
    public static void main(String[] args) {
        if (System.getSecurityManager() == null)
            {
            System.setSecurityManager(new SecurityManager());
            }
        try
            {
            LocateRegistry.createRegistry(1099);

            String name = "//localhost:1099/ComputeEngine";
            Compute engine = new ComputeEngine();
            Naming.rebind(name, engine);
```

```
            System.out.println("ComputeEngine bound");
            }
        catch (Exception e)
            {
            System.err.println("ComputeEngine exception:");
            e.printStackTrace();
            }
    }


}

/*
 * To change this template, choose Tools | Templates
 * and open the template in the editor.
 */
package sbd;

import java.io.Serializable;
import java.sql.Connection;
import java.sql.Statement;
import java.sql.ResultSet;
import java.sql.ResultSetMetaData;
import java.sql.DriverManager;
import java.sql.SQLException;
import java.util.Vector;

/**
 *
 * @author luzem
 */
public class baseCommand implements Serializable{

    protected Vector<String> parameterList;
    public String userName;
    public String password;
    public String Filename;
    protected String help;
    public boolean toFile;
    protected boolean isAdminCommand;
    protected boolean isUserCommand;

    /**
     * Creates a new baseCommand
     */
    public baseCommand()
    {
        parameterList=new Vector<String>();
        toFile=false;
        isAdminCommand=false;
        isUserCommand=false;
    }
```

```
/**
 * Executes the command
 * @return a String with the results
 */
public String execute() {
   return "";
}

/**
 * Check if the command need to be saved in a File
 * @return true if need be saved
 */
public boolean saveToFile()
   {
   return isToFile();
   }

/**
 * Add's a parameter in the command parameterList
 * @param parameter
 */
public void addParameter(String parameter) {
   parameterList.add(parameter);
}

/**
 * Get a conexión to the database
 * @return the database Connection
 */
protected Connection getConnection() {
   //TODO pharsear archivo para implementar independencia del driver
   Connection toret = null;
   try {
      toret = DriverManager.getConnection("jdbc:mysql://localhost/proa?", "root", "");
   } catch (SQLException ex) {
      System.out.println("SQLException: " + ex.getMessage());
      System.out.println("SQLState: " + ex.getSQLState());
      System.out.println("VendorError: " + ex.getErrorCode());
   }
   return toret;
}

/**
 * Get bug inserTode
 * @return the max code value in the bug table
 */
public int getBugMaxCodeValue() {
   int toret = 0;
   Connection con;
   con = getConnection();
```

```
    Statement stmt = null;
    ResultSet rs = null;
    try {
        stmt = con.createStatement();
        rs = stmt.executeQuery("select max(codigo) from bugs");
        rs.first();
        toret = rs.getInt(1);
        toret++;
        con.close();
    } catch (SQLException sqle) {
        System.out.println("SQLException: " + sqle.getMessage());
        System.out.println("SQLState: " + sqle.getSQLState());
        System.out.println("VendorError: " + sqle.getErrorCode());
    }
    return toret;
}

/**
 * Check if the username exist in the database
 * @param user the username
 * @return true if exist and it's enabled
 */
public boolean isValidUsername(String user) {
    boolean toret = false;
    Connection con;
    con = getConnection();
    Statement stmt = null;
    ResultSet rs = null;
    try {
        stmt = con.createStatement();
        rs = stmt.executeQuery("select * from usuarios where login='" + user + "'");
        rs.last();
        int count = rs.getRow();
        if (count > 0) {
            toret = true;
        }
        con.close();
    } catch (SQLException sqle) {
        System.out.println("SQLException: " + sqle.getMessage());
        System.out.println("SQLState: " + sqle.getSQLState());
        System.out.println("VendorError: " + sqle.getErrorCode());
    }
    return toret;
}

    /**
     * check if the username and password is valid
     * @return true if it's valid
     */
    public boolean verifyLogin()
    {
    return isValidUsernamePassword(userName,password);
```

```
        }


    /**
     * Check if a username and password is valid
     * @param user the username to verify
     * @param password the password to verify
     * @return true if it's a correct username and password
     */
    public boolean isValidUsernamePassword(String user, String password) {
        boolean toret = false;
        Connection con;
        con = getConnection();
        Statement stmt = null;
        ResultSet rs = null;
        try {
            stmt = con.createStatement();
            rs = stmt.executeQuery("select * from usuarios where login='" + user + "' and password='" +
password + "'");
            rs.last();
            int count = rs.getRow();
            if (count > 0) {
                toret = true;
            }
            con.close();
        } catch (SQLException sqle) {
            System.out.println("SQLException: " + sqle.getMessage());
            System.out.println("SQLState: " + sqle.getSQLState());
            System.out.println("VendorError: " + sqle.getErrorCode());
        }
        return toret;
    }

    /**
     * Check's if the username exist in the users table
     * @param user the username to check
     * @return true if exist
     */
    public boolean existUser(String user) {
        boolean toret = false;
        Connection con;
        Statement stmt = null;
        ResultSet rs = null;

        con = this.getConnection();
        try {
            stmt = con.createStatement();
            rs = stmt.executeQuery("select * from usuarios where login='" + user + "'");
            int count = 0;
            while (rs.next()) {
                count++;
            }
```

```
        if (count > 0) {
            toret = true;
        }
        con.close();
    } catch (SQLException sqle) {
        System.out.println("SQLException: " + sqle.getMessage());
        System.out.println("SQLState: " + sqle.getSQLState());
        System.out.println("VendorError: " + sqle.getErrorCode());
    }
    return toret;
}

/**
 * Executes a update SQL command
 * @param SQL the SQL instruccion
 * @return true if exit
 */
public boolean executeUpdate(String SQL)
    {
    boolean toret=false;
    try
        {
        Connection conn;
        conn=this.getConnection();
        Statement st=conn.createStatement();
        st.executeUpdate(SQL);
        st.close();
        conn.close();
        toret=true;
        }
    catch(SQLException sqle)
        {
        System.out.println("SQLException: " + sqle.getMessage());
        System.out.println("SQLState: " + sqle.getSQLState());
        System.out.println("VendorError: " + sqle.getErrorCode());
        }
    return toret;
    }

   /**
    * Executes a delete SQL command
    * @param SQL the SQL delete command
    * @return true if there's no problems in the execution
    */
   public boolean executeDelete(String SQL)
    {
    boolean toret=false;
    try
        {
        Connection conn;
        conn=this.getConnection();
        Statement st=conn.createStatement();
```

```
        st.execute(SQL);
        st.close();
        conn.close();
        toret=true;
        }
    catch(SQLException sqle)
        {
        System.out.println("SQLException: " + sqle.getMessage());
        System.out.println("SQLState: " + sqle.getSQLState());
        System.out.println("VendorError: " + sqle.getErrorCode());
        }
    return toret;
    }

/**
 * Exdecutes a insert SQL command
 * @param SQL the SQL inser command
 * @return a String with the error or exit
 */
public String executeInsert(String SQL)
    {
    String toret="";
    try
        {
        Connection conn;
        conn=this.getConnection();
        Statement st=conn.createStatement();
        st.execute(SQL);
        st.close();
        conn.close();
        toret="Register Inserted";
        }
    catch(SQLException sqle)
        {
        System.out.println("SQLException: " + sqle.getMessage());
        System.out.println("SQLState: " + sqle.getSQLState());
        System.out.println("VendorError: " + sqle.getErrorCode());
        toret="Error inserting in the database";

        }
    return toret;
    }

/**
 * Get's a select SQL query
 * @param SQL the SQL query
 * @return the result of the query
 */
public String getSelectQuery(String SQL)
    {
    String toret="";
    try{
```

```
    Connection conn;
    ResultSet rs;
    conn=this.getConnection();
    Statement st=conn.createStatement();
    st.executeQuery(SQL);
    rs=st.getResultSet();
    toret+=getResultSetMetaDataHeader(rs.getMetaData());
    toret+=getResultSetLines(rs);
    rs.close();
    st.close();
    conn.close();
    }catch(SQLException sqle)
        {
        System.out.println("SQLException: " + sqle.getMessage());
        System.out.println("SQLState: " + sqle.getSQLState());
        System.out.println("VendorError: " + sqle.getErrorCode());
        toret="Error querying database";
        }

    return toret;
    }

/**
 * Get a resultSet formated for print in the screen
 * @param rs the resultset for print
 * @return a String that represents the resultset
 */
public String getResultSetLines(ResultSet rs) {
    String toret = "";
    try {
        ResultSetMetaData rsmd;
        rs.first();
        rsmd = rs.getMetaData();
        do {
            for (int x = 1; x <= rsmd.getColumnCount(); x++) {
                toret += rs.getString(x) + "\t";
            }
            toret += "\n";
        } while (rs.next());
    } catch (SQLException sqle) {
        System.out.println("getResultSetLines");
        System.out.println("SQLException: " + sqle.getMessage());
        System.out.println("SQLState: " + sqle.getSQLState());
        System.out.println("VendorError: " + sqle.getErrorCode());
    }
    return toret;
}

/**
 * Get the resultSet fieldNames for screen print
 * @param rsmd the resultserMetaData
 * @return a String with the field's names
```

```
 */
public String getResultSetMetaDataHeader(ResultSetMetaData rsmd) {
   String toret = "";
   try {
      for (int x = 1; x <= rsmd.getColumnCount(); x++) {
         toret += rsmd.getColumnName(x) + "\t";
         System.out.print(rsmd.getColumnName(x) + "\t");
         System.out.println();
      }
   } catch (SQLException sqle) {
      System.out.println("SQLException: " + sqle.getMessage());
      System.out.println("SQLState: " + sqle.getSQLState());
      System.out.println("VendorError: " + sqle.getErrorCode());
   }
   toret += "\n";
   return toret;
}

/**
 * check if the bugCode is solved
 * @param bugCode the bugCode id
 * @return true if solved false otherwise
 */
public boolean isBugSolved(int bugCode)
   {
   boolean toret=false;
   //SELECT programador from bugs where codigo=bugCode
   try {
      Connection con;
      String bugStatus;
      con=this.getConnection();
      Statement st;
      ResultSet rs;
      st=con.createStatement();
      st.execute("SELECT estado from bugs where codigo="+bugCode);
      rs=st.getResultSet();
      if (!rs.first())
         {
         System.out.println("Invalid bug code");
         }
      else
         {
         bugStatus=rs.getString(1);
         System.out.println("bugStatus =="+bugStatus+"||");
         if (bugStatus.equals("RESUELTO"))
            {
            System.out.println("Cambiando valor de toret");
            toret=true;
            }
         }
      }catch (SQLException sqle) {
      System.out.println("SQLException: " + sqle.getMessage());
```

```
        System.out.println("SQLState: " + sqle.getSQLState());
        System.out.println("VendorError: " + sqle.getErrorCode());
    }
    return toret;
    }

/**
 * Check if the bugCode exist in the table
 * @param bugCode the bugcode to check
 * @return true if exist
 */
public boolean existBug(int bugCode)
    {
    boolean toret=false;
    try
        {
        String SQL="SELECT * FROM bugs where codigo="+bugCode;
        Connection con;
        con=this.getConnection();
        Statement st;
        ResultSet rs;
        st=con.createStatement();
        st.executeQuery(SQL);
        rs=st.getResultSet();
        if (rs.first())
            {
            toret=true;
            }
        }
    catch(SQLException sqle)
        {
        System.out.println("SQLException: " + sqle.getMessage());
        System.out.println("SQLState: " + sqle.getSQLState());
        System.out.println("VendorError: " + sqle.getErrorCode());
        }
    return toret;
    }

/**
 * Get the programmer asociated to a bug code
 * @param bugCode the bugCode ID
 * @return the user asociated to the bugCode
 */
public String getBugProgrammer(int bugCode)
    {
    String toret="";
    //SELECT programador from bugs where codigo=bugCode
    try {
        Connection con;
        con=this.getConnection();
        Statement st;
        ResultSet rs;
```

```
            st=con.createStatement();
            st.execute("SELECT programador from bugs where codigo="+bugCode);
            rs=st.getResultSet();
            if (!rs.first())
               {
               System.out.println("Invalid bug code");
               }
            else
               {
               toret=rs.getString(1);
               }
            }catch (SQLException sqle) {
            System.out.println("SQLException: " + sqle.getMessage());
            System.out.println("SQLState: " + sqle.getSQLState());
            System.out.println("VendorError: " + sqle.getErrorCode());
         }
         return toret;
         }

      /**
       * Check if a username is admin
       * @param user the username
       * @return true if the username is admin
       */
      public boolean isAdminUsername(String user) {
         boolean toret = false;
         Connection con;
         Statement stmt = null;
         ResultSet rs = null;

         con = this.getConnection();
         try {
            stmt = con.createStatement();
            rs = stmt.executeQuery("select * from usuarios where login='" + user + "'and
administrador=1");
            rs.last();
            int count = rs.getRow();
            if (count > 0) {
               toret = true;
            }
            con.close();
         } catch (SQLException sqle) {
            System.out.println("SQLException: " + sqle.getMessage());
            System.out.println("SQLState: " + sqle.getSQLState());
            System.out.println("VendorError: " + sqle.getErrorCode());
         }
         return toret;
      }

      /**
       * @return the user
       */
```

```
public String getUserName() {
    System.out.println("returning Username:"+userName);
    return userName;
}

/**
 * @param user the user to set
 */
public void setUserName(String user) {
    System.out.println("setting Username:"+user);
    this.userName = user;
}

/**
 * @return the pass
 */
public String getPassword() {
    System.out.println("returning Password:"+password);
    return password;
}

/**
 * @param pass the pass to set
 */
public void setPassword(String pass) {
    System.out.println("Setting password:"+pass);
    this.password = pass;
}

/**
 * return the help String associated to the command
 * @return help String associated to the command
 */
public String getHelp()
    {
    return help;
    }


/**
 * Check if the command has permisions to be exuted
 * @return true is have permissions
 */
public boolean havePermisionstoExecuteCommand()
    {
    boolean toret=false;
    if (isAdminUsername(userName) && isAdminCommand)
        {
        toret=true;
        }
    if ( !(isAdminUsername(userName)) && isUserCommand)
        {
```

```
            toret=true;
            }
        return toret;
        }

    /**
     * return a parameter from the Command
     * @param index the index of parameter
     * @return a String that contains the parameter
     */
    public String getParameterAt(int index)
        {
        return parameterList.elementAt(index);
        }

    /**
     * @return the toFile
     */
    public boolean isToFile() {
        return toFile;
    }

    /**
     * @return the Filename
     */
    public String getFilename() {
        return Filename;
    }
}
```

CjavaTrack

```
/*
 * To change this template, choose Tools | Templates
 * and open the template in the editor.
 */

package cjavatrack;


/**
 *
 * @author luzem
 */
public class Main {

    /**
     * Launches a new client
     * @param args the command line arguments
     */
    public static void main(String[] args) {
        String host="127.0.0.1";
```

```
        int port=12000;
        if (args.length==2)
            {
            new clientInterface(args[0],Integer.parseInt(args[1]));
            }
        else
            {
            System.out.println("usager cJavaTrack host port");
            }


    }

}

/*
 * To change this template, choose Tools | Templates
 * and open the template in the editor.
 */
package cjavatrack;

import java.net.*;
import java.io.*;
import sjavatrack.CommandOutput;

/**
 *
 * @author luzem
 */
public class clientInterface {

    /**
     *  skClient the socket user for the connections
     */
    Socket skClient = null;

    /**
     * Creates a new client interface
     * @param host host of the Socket server
     * @param port port where the server is listening
     */
    clientInterface(String host, int port) {
        String readData = null;
        CommandOutput co;
        try {
            skClient = new Socket(host, port);
            System.out.println("Socket Created");
            while (!skClient.isClosed()) {
                printPromt();
                readData = readKeyboard();
                if (readData.equals("quit"))
                    {
```

```
                skClient.close();
                System.exit(0);
                }
            else
                {
                sendData(readData);
                co=receiveCommandOutput();
                processCommandOutput(co);
                }
            //System.out.println("Response="+co.getResponse());
        }
    } catch (java.net.UnknownHostException uhe) {
        System.out.println(uhe.toString());
    } catch (java.io.IOException ioe) {
        System.out.println(ioe.toString());
    }
}


/**
 * Read's data from keyboard
 * @return the keyboard data reader
 */
private String readKeyboard() {
    String toret = null;
    try {
        InputStreamReader reader = new InputStreamReader(System.in);
        BufferedReader in = new BufferedReader(reader);
        toret = in.readLine();
    } catch (java.io.IOException ioe) {
        System.out.println(ioe.toString());
    }

    return toret;
}

/**
 * Sends text over the socket
 * @param data the String  for send
 */
private void sendData(String data) {
    try {
        OutputStream stream = skClient.getOutputStream();
        DataOutputStream flow = new DataOutputStream(stream);
        flow.writeUTF(data);
        //stream.close();
        //flow.close();
    } catch (java.io.IOException ioe) {
        System.out.println(ioe.toString());
    }
}
```

```
/**
 * Process a CommandOutput
 * @param co command to process
 */
private void processCommandOutput(CommandOutput co)
   {
   if (co.isToFile())
      {
      try {
         //Save to File
         System.out.println("Writing" + co.getResponse());
         FileOutputStream fos;
         fos = new FileOutputStream(co.getFilename());
         new PrintStream(fos).println(co.getResponse());
         fos.close();
         }
      catch (IOException ex)
         {
         System.out.println("Error writing a file");
         }
      }
   else
      {
      System.out.println(co.getResponse());
      }

   }

/**
 * Receives a CommandOutput from the server
 * @return the CommandOutput received
 */
private CommandOutput receiveCommandOutput()
   {
   CommandOutput co;
   co =new CommandOutput(false,null,"Error processing CommandOutput");
   try {
      InputStream is = skClient.getInputStream();
      ObjectInputStream ois=new ObjectInputStream(is);
      co=(CommandOutput) ois.readObject();
      //ois.close();
      //is.close();
      //processCommandOutput(co);
      }
   catch (ClassNotFoundException ex)
      {

      }
   catch (IOException ex)
      {
      }
```

```
        return co;
        }

    /**
     * Print's console Prompt
     */
    private void printPromt()
        {
        System.out.print(">");
        }

   /* private String receiveData() {
        String toret = null;
        try {
            InputStream aux = skClient.getInputStream();
            DataInputStream flow = new DataInputStream(aux);
            toret = flow.readUTF();
            //System.out.println(toret);
        } catch (java.io.IOException ioe) {
            //Server closes conecction
            System.exit(0);
        }
        return toret;
    }*/
}
```

sJavaTrack

```
/*
 * To change this template, choose Tools | Templates
 * and open the template in the editor.
 */

package sjavatrack;

import java.io.Serializable;

/**
 *
 * @author luzem
 */
public class CommandOutput implements Serializable{

    /**
     * indicates if the command output must be sabed in a file
     */
    private boolean toFile;
    /**
     * filename to save the command outut
     */
    private String filename;
    /**
```

```
 * command output
 */
private String response;
static final long serialVersionUID = 1;


/**
 * Generates a new CommandOutput
 * @param toFile
 * @param filename
 * @param response
 */
public CommandOutput(boolean toFile,String filename, String response)
    {
    this.toFile=toFile;
    this.filename=filename;
    this.response=response;
    }


/**
 * Returns a XML that contains the object values
 * @return a String that contains the XML
 */
public String toXML()
    {
    String toret="";
    toret+="<XML>";
        toret+="<COMMAND>";
            toret+="<TOFILE>";
            toret+=isToFile();
            toret+="</TOFILE>";
            toret+="<FILENAME>";
            toret+=getFilename();
            toret+="</FILENAME>";
            toret+="<RESPONSE>";
            toret+=getResponse();
            toret+="</RESPONSE>";
        toret+="</COMMAND>";
    toret+="</XML>";
    return toret;
    }

/**
 * @return the toFile
 */
public boolean isToFile() {
    return toFile;
}

/**
 * @return the filename
```

```
     */
    public String getFilename() {
        return filename;
    }

    /**
     * @return the response
     */
    public String getResponse() {
        return response;
    }

    /**
     * @param toFile the toFile to set
     */
    public void setToFile(boolean toFile) {
        this.toFile = toFile;
    }

    /**
     * @param filename the filename to set
     */
    public void setFilename(String filename) {
        this.filename = filename;
    }

    /**
     * @param response the response to set
     */
    public void setResponse(String response) {
        this.response = response;
    }
}

/*
 *
 * To change this template, choose Tools | Templates
 * and open the template in the editor.
 */

package sjavatrack;

import java.net.*;

/**
 *
 * @author luzem
 */
public class Main {

    /**
     * the socket used to listen connections
```

```
     */
    ServerSocket serverSocket = null;


    /**
     * Creates a new sJavaTrack instance
     * @param newPort the port were the server listen
     */
    Main(int newPort)
        {
        try
            {
            boolean listening=true;
            serverSocket = new ServerSocket(newPort);
            System.out.println("Socket Created \n");
            while (listening)
                {
                //serverSocket= new ServerSocket(port);
                System.out.println("Listening");
                //Creates a new thread for every incoming client conecction
                new socketThread(serverSocket.accept()).start();
                }
            }
        catch(java.net.UnknownHostException uhe)
            {
            System.out.println("Main Server\n");
            System.out.println(uhe.toString());
            }
        catch (java.io.IOException ioe)
            {
            System.out.println("Main Server \n");
            System.out.println(ioe.toString());
            }
        }

    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) {
        // TODO code application logic here
        if (args.length==1)
            {
            Main sever=new Main(Integer.parseInt(args[0]));
            }
        else
            {
            System.out.println("we need port number");
            }
    }
}

/*
```

```
 * To change this template, choose Tools | Templates
 * and open the template in the editor.
 */
package sjavatrack;

import java.util.StringTokenizer;
import java.net.*;
import java.io.*;
import java.rmi.Naming;
import java.util.logging.Level;
import java.util.logging.Logger;
import sbd.Compute;
import sbd.baseCommand;
/**
 *
 * @author luzem
 */
public class socketThread extends Thread {

    /**
     * the socket used to mantains the communication with the client
     */
    private Socket socket = null;

    /**
     * Creates a new socket thread
     * @param socket the socket connection tath listen
     */
    public socketThread(Socket socket) {
        super();
        this.socket = socket;
    }



    @Override
    public void run()
        {
        String username=null;
        String password=null;
        String clientMessage="";
        String receivedMessage="";
        StringTokenizer st;
        String commandtoExecute;
        Class comandClass;
        baseCommand obtainedCommand;
        CommandOutput co;
        co =new CommandOutput(false,null,"");
        System.out.println("Running a Thread");
        while (!socket.isClosed() )
            {
            receivedMessage = readClient();
```

```
System.out.println("Received message:"+receivedMessage);
if (!receivedMessage.equals(""))//case not whitespaces
   {
   st=new StringTokenizer(receivedMessage);
   commandtoExecute=st.nextToken();
   try
      {
      System.out.println("Command to Execute:"+commandtoExecute+":");
      if (commandtoExecute.equals("help"))//case help
         {
         System.out.print("Help command\n");
         if (st.hasMoreTokens())
            {
            comandClass = Class.forName("sjavatrack.commands." + st.nextToken() );
            }
         else
            {
            comandClass = Class.forName("sjavatrack.commands." + commandtoExecute);
            }
         }
      else
         {
         comandClass = Class.forName("sjavatrack.commands." + commandtoExecute);
         }
      obtainedCommand = (baseCommand) comandClass.newInstance();
      //Adding parameters
      while (st.hasMoreElements())
         {
         obtainedCommand.addParameter(st.nextToken());
         }
      obtainedCommand.setUserName(username);
      obtainedCommand.setPassword(password);
      if (commandtoExecute.equals("help"))
         {
         clientMessage=obtainedCommand.getHelp();
         }
      else
         {
         try {
            //System.out.println("Checking permissions");
            //System.out.println("Username ="+ username);
         if (remotePermisionsCheck(obtainedCommand))//Check permissions
            {
               //clientMessage=obtainedCommand.execute();//Execute
               clientMessage = remoteExecute(obtainedCommand);
               if (obtainedCommand.isToFile())
                  {
                  co.setToFile(true);
                  co.setFilename(obtainedCommand.getFilename());
                  }

            }
```

```
                else
                   {
                   clientMessage="ERROR invalid permisions";
                   }
                } catch (Exception ex) {
                      ex.printStackTrace();
                      closeSocket();
                      break;
                   }
                }
              if (commandtoExecute.equals("login"))
                {
                System.out.println("Initializeing login variables
log="+obtainedCommand.verifyLogin());
                try
                   {
                   if (clientMessage.equals("Login successful"))
                      {
                      System.out.println("Changuing login variables values");
                      System.out.println("Username:"+ obtainedCommand.getParameterAt(0));
                      System.out.println("Password:"+ obtainedCommand.getParameterAt(1));
                      username=obtainedCommand.getUserName();
                      password=obtainedCommand.getPassword();
                      }
                   }
                catch(Exception e)
                   {
                   System.out.println(e.toString());
                   closeSocket();
                   }
                }
             }
          catch (ClassNotFoundException cnfe)
                   {
                   clientMessage="Command not found use help for list";
                   }
                catch (InstantiationException ie)
                   {
                   clientMessage=ie.toString();
                   }
                catch (IllegalAccessException iae)
                   {
                   clientMessage=iae.toString();
                   }
           }
        co.setResponse(clientMessage);
        sendCommandOutput(co);
        //Reinitializa values
        co.setFilename("");
        co.setToFile(false);
        co.setResponse("");
        }
```

```
      System.out.println("Ending Thread");
    }


/**
 * Check permisions of the baseCommand
 * @param bc the base command
 * @return true if the user attached to the command works
 * @throws Exception
 */
private boolean remotePermisionsCheck(baseCommand bc) throws Exception
    {
    boolean toret=false;
     if (System.getSecurityManager() == null)
        {
        System.setSecurityManager(new SecurityManager());
        }
    String name = "ComputeEngine";
    Compute ce= (Compute) Naming.lookup("//localhost:1099/"+name);
    toret=ce.hasPermisions(bc);
    System.out.println("Permissions check gives "+toret);
    return toret;
    }

/**
 * Execute's a base command in the remote server
 * @param bc baseCommand to execute
 * @return the baseCommand result
 * @throws Exception
 */
    private String remoteExecute(baseCommand bc) throws Exception
    {
    String toret="";
     if (System.getSecurityManager() == null)
        {
        System.setSecurityManager(new SecurityManager());
        }

        String name = "ComputeEngine";
        Compute ce= (Compute) Naming.lookup("//localhost:1099/"+name);
        toret=ce.execute(bc);

    return toret;

    }

private boolean remoteVerify(baseCommand bc) throws Exception
    {
    boolean toret=false;
     if (System.getSecurityManager() == null)
        {
        System.setSecurityManager(new SecurityManager());
```

```
        }
        String name = "ComputeEngine";
        Compute ce= (Compute) Naming.lookup("//localhost:1099/"+name);
        toret=ce.verifyLogin(bc);

    return toret;

    }

/**
 * Read clients data
 * @return the clients data
 */
private String readClient() {
    String toret = null;
    try {
        InputStream aux = socket.getInputStream();
        DataInputStream flow = new DataInputStream(aux);
        toret = flow.readUTF();
        //System.out.println(toret);
        } catch (java.io.IOException ioe) {
        System.out.println("SocketThread readClient()");
        System.out.println(ioe.toString());
        closeSocket();
        toret = "";

    }
    return toret;
}

/**
 * close the thread socket
 */
private void closeSocket()
    {
    try{
        socket.close();
        }
    catch(java.io.IOException ioe)
        {
        System.out.println("Closing socket");
        }
    }

/**
 * Send a CommandOutput to the client
 * @param co the CommandOutput
 */
private void sendCommandOutput(CommandOutput co)
    {
    try {
        OutputStream os = socket.getOutputStream();
```

```
            ObjectOutputStream oos = new ObjectOutputStream(os);
            oos.writeObject(co);
            //oos.close();
            }
        catch (IOException ex)
            {
            Logger.getLogger(socketThread.class.getName()).log(Level.SEVERE, null, ex);
            }

        }


    private void sendMessage(String message) {
        try {
            OutputStream stream = socket.getOutputStream();
            DataOutputStream flow = new DataOutputStream(stream);
            flow.writeUTF(message);
        } catch (java.io.IOException ioe) {
            System.out.println(ioe.toString());
            this.interrupt();
        }
    }
}


/*
 * To change this template, choose Tools | Templates
 * and open the template in the editor.
 */

package sjavatrack.commands;

import sbd.baseCommand;
import java.sql.Statement;
import java.sql.Connection;
import java.sql.SQLException;

/**
 *
 * @author luzem
 */
public class add_bug extends baseCommand{

    /**
     * Aplication name
     */
    String aplication=null; //aplication name
    /**
     * bug description
     */
    String description=null; //description of the bug
    /**
```

```
 * developer login
 */
String developer=null; // developer name

static final long serialVersionUID = 2L;

/**
 * creates a new add_bug command
 */
public add_bug()
    {
    super();
    help="add_bug <aplicacion> <descripcion> <login_programador>";
    isUserCommand=true;
    }

/**
 * executes the add_bug command
 * @return a String with the command response
 */
@Override
public String  execute()
    {
    String toret="";
    processparameters();
    if (!validateParameters())
        {
        toret="Not enougth parameters";
        }
    else
        {
        if (isValidUsername(developer))
            {
            Connection con;
            Statement st;
            con=getConnection();
            String SQLInsert="";
            try{
            st=con.createStatement();
            SQLInsert="Insert into bugs values("+(getBugMaxCodeValue())+",";
            SQLInsert+="'"+aplication+"',";
            SQLInsert+="'"+description+"',";
            SQLInsert+="'"+userName+"',";
            SQLInsert+="'"+developer+"',";
            SQLInsert+="'"+"PENDIENTE"+"',";
            SQLInsert+="'"+"',";
            SQLInsert+="now()";
            SQLInsert+=")";
            System.out.println(SQLInsert);
            st.execute(SQLInsert);
            st.close();
            con.close();
```

```
            }catch(SQLException sqle)
               {
                System.out.println("SQLException: " + sqle.getMessage());
                System.out.println("SQLState: " + sqle.getSQLState());
                System.out.println("VendorError: " + sqle.getErrorCode());
               }

            }
          else
            {
            toret="Invalid Developer";
            }
          }
       return toret;
       }

    /**
     * Validates the parameters
     * @return true if valid parameters
     */
       private void processparameters()
          {
          for (int z=0;z<parameterList.size();z++)
            {
            System.out.println("Element at "+z+" value="+parameterList.elementAt(z) );
            }
          if (parameterList.size() < 2)
            {
            System.out.println("Not enougth parameters for add_Bug yau passed "+
parameterList.size());
            }
          else
            {
            aplication= this.parameterList.firstElement();
            description="";
            for(int x=1;x<parameterList.size()-1;x++)
              {
               description+=parameterList.elementAt(x);
               if (x!=(parameterList.size()-1))
                 {
                  description+=" ";
                 }
               }
            System.out.println("Description: "+ description);
            developer=parameterList.lastElement();
            }

          }

       /**
        * Validate the parameters
        * @return true if valid parameters
```

```
         */
        public boolean validateParameters()
            {
            boolean toret=true;
            if (aplication==null)
                {
                toret=false;
                }
            else if (description == null)
                {
                toret=false;
                }
            else if (developer==null)
                {
                toret=false;
                }
            return toret;
            }



}

/*
 * To change this template, choose Tools | Templates
 * and open the template in the editor.
 */

package sjavatrack.commands;

import sbd.baseCommand;
/**
 * @author luzem
 */
public class add_user extends baseCommand{

    /**
     * Username to insert
     */
    private String userNametoInsert;
    /**
     * password of the new user
     */
    private String passwordtoInsert;
    /**
     * indicates admin role
     */
    private int isAdmin;
    //
    static final long serialVersionUID = 2L;

    /**
```

```
     * Creates a new add_user command
     */
    public add_user()
        {
        super();
        help="add_user <login_usuario> <password_usuario> <¿es_administrador?>";
        isAdmin=0;
        isAdminCommand=true;
        }

    /**
     * executes the addUser command
     * @return a String with the response
     */
    @Override
    public String  execute()
        {
        String toret="";
        if(validateParameters())
            {
            if (!isValidUsername(userNametoInsert))
                {
                String SQL;
                SQL="insert into usuarios
values('"+userNametoInsert+"','"+passwordtoInsert+"',"+isAdmin+",0)";
                System.out.println("SQL to Execute=");
                System.out.println(SQL);
                toret=executeInsert(SQL);
                }
            else
                {
                toret="Error: Username exist";
                }
            }
        else
            {
            toret="Error: validating parameters ";
            }

        return toret;
        }

    /**
     * Validates the parameters
     * @return true if valid parameters
     */
    private boolean validateParameters()
        {
        boolean toret=false;
        if (!(parameterList.size()>3 ||parameterList.size()<2))
            {
            userNametoInsert=parameterList.elementAt(0);
```

```
            passwordtoInsert=parameterList.elementAt(1);
            if (parameterList.size()==3)
                {
                if (parameterList.elementAt(2).equals("1"))
                    {
                    isAdmin=1;
                    }
                }
            toret=true;
            }
        return toret;
        }
}

/*
 * To change this template, choose Tools | Templates
 * and open the template in the editor.
 */

package sjavatrack.commands;

import sbd.baseCommand;
/**
 *
 * @author luzem
 */
public class del_user extends baseCommand{

    String userNameToDelete;
    static final long serialVersionUID = 2L;
//
//    Elimina de la base de datos javatrack el usuario especificado.
//      Si existen incidencias cuyo autor es este usuario también deberán ser eliminados de la base de
datos.
//      Si existen incidencias  asignadas al programador que se va a eliminar, deberán quedar sin
asignar a nadie hasta que el
//    administrador la reasigne.

    /**
     * Creates a new del_user comman
     */
    public del_user()
        {
        super();
        help="del_user <login_usuario>";
        isAdminCommand=true;
        }

        /**
     * Executes the command
     * @return the response of execution
     */
```

```
    @Override
    public String  execute()
        {
        String toret="";
        if (processParameters())
        {
        if (isValidUsername(userNameToDelete))
            {
            String SQL;
            //Si existen incidencias  asignadas al programador que se va a eliminar, deberán quedar sin
asignar a nadie hasta que el administrador la reasigne.
            //SELECT * FROM bugs  where estado='PENDIENTE' and programador='admin'
            //Update bugs set programador=null where estado='PENDIENTE' and programador='admin'
            SQL="Update bugs set programador=null where estado='PENDIENTE' and programador=";
            SQL+="'";
            SQL+=userNameToDelete;
            SQL+="'";
            if (executeUpdate(SQL))
                {
                //Si existen incidencias cuyo autor es este usuario también deberán ser eliminados de la
base de datos.
                SQL="Delete from bugs where estado='RESUELTO' and autor='";
                SQL+=userNameToDelete;
                SQL+="'";
                if (executeDelete(SQL))
                    {
                    //Remove User
                    SQL="Delete from usuarios where login='";
                    SQL+=userNameToDelete;
                    SQL+="'";
                    if (executeDelete(SQL))
                        {
                        toret="User Deleted";
                        }
                    else
                        {
                        toret="ERROR:when removing User";
                        }
                    }
                else
                    {
                    toret="ERROR: while removing user reported bugs";
                    }
                }
            else
                {
                toret="ERROR: while freeing pendent bugs";
                }


            }
        else
```

```
            {
            toret="ERROR: Invalid username";
            }
        }
        else
            {
            toret="ERROR: Invalid parameters";
            }
        return toret;
        }


    /**
    * Validates the parameters
    * @return true if valid parameters
    */
        private boolean processParameters()
        {
        boolean toret=false;
        System.out.println("Parameter size "+parameterList.size());
            if (parameterList.size()==1)
                {
                userNameToDelete=parameterList.elementAt(0);
                toret=true;
                }
        return toret;
        }
}

/*
 * To change this template, choose Tools | Templates
 * and open the template in the editor.
 */

package sjavatrack.commands;

import sbd.baseCommand;

import java.lang.NumberFormatException;
/**
 *
 * @author luzem
 */
public class delete_bug extends baseCommand{
    /**
     * bigCode to delete
     */
    int bugCode;
    static final long serialVersionUID = 2L;

        /**
     * Creates a new delete_bug comman
```

```
 */
public delete_bug()
    {
    super();
    help="delete_bug bugID";
    isAdminCommand=true;
    }

    /**
 * Executes the command
 * @return the response of execution
 */
@Override
public String  execute()
    {
    String toret=null;
    if (processParameters())
        {
        if (existBug(bugCode))
            {
            String SQL="Delete from bugs where codigo="+bugCode;
            if (executeDelete(SQL))
                {
                toret="Bug Deleted";
                }
            else
                {
                toret="ERROR: Database error";
                }
            }
        else
            {
            toret="ERROR: Bug not exist";
            }
        }
    else
        {
        toret="ERROR: Invalid parameters";
        }
    return toret;
    }

/**
 * Validates the parameters
 * @return true if valid parameters
 */
private boolean processParameters()
    {
    boolean toret=false;
    System.out.println("Parameter size "+parameterList.size());
        if (parameterList.size()==1)
            {
```

```
            try
                {
                Integer castingInt;
                castingInt=new Integer(parameterList.elementAt(0));
                bugCode=castingInt.intValue();
                toret=true;
                }
            catch(NumberFormatException nfe)
                {
                toret=false;
                }
            }
        return toret;
        }
}

/*
 * To change this template, choose Tools | Templates
 * and open the template in the editor.
 */

package sjavatrack.commands;

import sbd.baseCommand;
/**
 *
 * @author luzem
 */
public class enable_user extends baseCommand{

    /**
     * username to enable
     */
    private String userNameToEnable;
    /**
     * indicates enabled status
     */
    private int    enabled;
    static final long serialVersionUID = 2L;

     /**
     * Creates a new enable_user command
     */
    public enable_user()
        {
        super();
        help="enable_user <username>";
        isAdminCommand=true;
        }

        /**
     * Executes the command
```

```
 * @return the response of execution
 */
@Override
public String  execute()
    {
    String toret=null;
    if (processParameters())
        {
        if (isValidUsername(userNameToEnable))
            {
            String SQL="Update usuarios set habilitado='" ;
            SQL+=enabled;
            SQL+="' where usuario='";
            SQL+=userNameToEnable;
            SQL+="'";
            if (executeUpdate(SQL))
                {
                toret="Update sucessulf";
                }
            else
                {
                toret="ERROR: updated database error";
                }
            }
        else
            {
            toret="ERROR: User Not exist";
            }

        }
    else
        {
        toret="ERROR: Invalid parameters";
        }
    return toret;
    }


 /**
 * Validates the parameters
 * @return true if valid parameters
 */
private boolean processParameters()
    {
    boolean toret=false;
    System.out.println("Parameter size "+parameterList.size());
        if (parameterList.size()==2)
            {
            try
                {
                userNameToEnable=parameterList.elementAt(0);
                Integer castingInt;
```

```
                castingInt=new Integer(parameterList.elementAt(1));
                enabled=castingInt.intValue();
                if (enabled>=0 && enabled<2)
                   {
                   toret=true;
                   }
                }
             catch(NumberFormatException nfe)
                {
                toret=false;
                }
             }
       return toret;
       }
}

/*
 * To change this template, choose Tools | Templates
 * and open the template in the editor.
 */

package sjavatrack.commands;

import sbd.baseCommand;
/**
 *
 * @author luzem
 */
public class help extends baseCommand{

   static final long serialVersionUID = 2L;
   /**
    * Creates a new help command
    */
   public help()
      {
      super();
      help="help command";
      isAdminCommand=true;
      isUserCommand=true;
      }



}

/*
 * To change this template, choose Tools | Templates
 * and open the template in the editor.
 */

package sjavatrack.commands;
```

```
import sbd.baseCommand;
import java.sql.Connection;
import java.sql.SQLException;
import java.sql.Statement;
import java.sql.ResultSet;
import java.sql.ResultSetMetaData;
import java.util.Date;
import java.util.StringTokenizer;
import java.util.Calendar;
/**
 *
 * @author luzem
 */
public class list_bugs extends baseCommand {

    /**
     * Developer to filter
     */
    private String developer=null;
    /**
     * first date to filter
     */
    private Calendar firstDate=null;
    /**
     * last date to filter
     */
    private Calendar lastDate=null;
    static final long serialVersionUID = 2L;
    /**
     * Creates a new list_bugs command
     */
    public list_bugs()
        {
        super();
        help="show bugs ";
        isAdminCommand=true;
        isUserCommand=true;
        }

        /**
     * Validates the parameters
     * @return true if valid parameters
     */
    private boolean processParameters()
        {
        boolean toret=true;
        System.out.println("Numero de parametros == "+parameterList.size());
        if (parameterList.size()>2)
            {
            toret= false;
            }
```

```
        else if(parameterList.size()==1)
           {
           System.out.println("Inicialiting developer var");
           developer=parameterList.firstElement();
           }
        else if (parameterList.size()==2)
           {
           //check dates
           if (!(stringToDate(true,parameterList.firstElement()) &&
stringToDate(false,parameterList.elementAt(1))))
              {
              toret=false;
              }
           }
        return toret;
        }

        /**
         * Converts a String to a date
         * @param first indicates if it's the first conversion
         * @param dataText te String to convers
         * @return true if the String was converted
         */
        private boolean stringToDate(boolean first,String dataText)
           {
           boolean toret=false;
           Calendar tempCal=Calendar.getInstance();
           StringTokenizer st=new StringTokenizer(dataText,"/");
           if (st.countTokens()==3)
              {
              int year;
              int month;
              int day;
              Integer intToCast;
              intToCast=new Integer(st.nextToken());
              day=intToCast.intValue();
              intToCast=new Integer(st.nextToken());
              month=intToCast.intValue();
              intToCast=new Integer(st.nextToken());
              year=intToCast.intValue();
              tempCal.set(Calendar.YEAR, year);
              tempCal.set(Calendar.MONTH, month-1);
              tempCal.set(Calendar.DATE, day);
              if (first)
                 {
                 firstDate=tempCal;
                 }
              else
                 {
                 lastDate=tempCal;
                 }
              toret=true;
```

```
            System.out.println(tempCal.toString());
            }
        return toret;
        }

        /**
* Executes the command
* @return the response of execution
*/

    @Override
    public String  execute()
        {
        boolean error=false;
        String toret="";
        if (processParameters())
            {
            String SQL="Select * from bugs ";
            if (developer!=null)
                {
                if (!existUser(developer))
                    {
                    System.out.println("Incorrect UserName to list");
                    toret="Incorrect UserName to list";
                    error=true;
                    }
                else
                    {
                    SQL+="where programador='"+developer+"'";
                    }
                }
            else if(firstDate!=null && lastDate!=null)
                {
                //SELECT * FROM proa.bugs WHERE fecha > '2010-01-30' AND fecha <'2010-02-02'
                SQL+="WHERE fecha>='";
                SQL+=firstDate.get(Calendar.YEAR);
                SQL+="-";
                SQL+=firstDate.get(Calendar.MONTH);
                SQL+="-";
                SQL+=firstDate.get(Calendar.YEAR);
                SQL+="' AND fecha<='";
                SQL+=lastDate.get(Calendar.YEAR);
                SQL+="-";
                SQL+=lastDate.get(Calendar.MONTH);
                SQL+="-";
                SQL+=lastDate.get(Calendar.YEAR);
                SQL+="'";
                }
            SQL+=" order by fecha asc";
            System.out.println(SQL);
            if (!error)
                {
```

```
                try{
                Connection conn= getConnection();
                Statement st= conn.createStatement();
                st.execute(SQL);
                ResultSet rs=st.getResultSet();
                if (rs==null)
                   {
                   toret="Error in SQL Query";
                   }
                else
                   {
                   rs.first();
                   ResultSetMetaData rsmd=rs.getMetaData();
                   toret+=getResultSetMetaDataHeader(rsmd);
                   toret+=getResultSetLines(rs);
                   }
                }catch(SQLException sqle)
                   {
                   System.out.println("SQLException: " + sqle.getMessage());
                   System.out.println("SQLState: " + sqle.getSQLState());
                   System.out.println("VendorError: " + sqle.getErrorCode());
                   }
                }
             }
          else
             {
             toret="Too much parameters";
             }
          return toret;
          }




}

/*
 * To change this template, choose Tools | Templates
 * and open the template in the editor.
 */

package sjavatrack.commands;

import sbd.baseCommand;
/**
 *
 * @author luzem
 */

//list_users
//     Produce un listado en el cliente de todos
```

```
//     los usuarios registrados en sistema que no son administradores.

public class list_users extends baseCommand {

    static final long serialVersionUID = 2L;

   /**
    * Creates a new list_users command
    */
    public list_users()
      {
      super();
      help="list users ";
      isAdminCommand=true;
      isUserCommand=true;
      }
    @Override
    public String  execute()
      {
      String toret="";
      if(processParameters())
          {
          String SQL="SELECT * FROM usuarios";
          if (!isAdminUsername(userName))
              {
              SQL+=" where administrador=0";
              }
          toret=getSelectQuery(SQL);
          }
      return toret;
      }
          /**
    * Validates the parameters
    * @return true if valid parameters
    */
    private boolean processParameters()
      {
      boolean toret=true;
      if (parameterList.size()>0)
          {
          toret=false;
          }
      return toret;
      }
}

/*
 * To change this template, choose Tools | Templates
 * and open the template in the editor.
 */

package sjavatrack.commands;
```

```
import sbd.baseCommand;
/**
 *
 * @author luzem
 */
public class login extends baseCommand{

    static final long serialVersionUID = 2L;
    /**
     * indicates if it's logged
     */
    private boolean loged;

     /**
     * Creates a new login command
     */
    public login()
        {
        super();
        help="Login's into the system";
        loged=false;
        }

    @Override
    public String  execute()
        {
        String toret=null;
        if (processParameters())
            {
            if (this.isValidUsernamePassword(userName, password))
                {
                 System.out.println("Login check correct");
                 System.out.println("Username:"+this.getUserName());
                 System.out.println("Password"+this.getPassword());
                 toret="Login successful";
                 loged=true;
                }
            else
                {
                toret="ERROR: Incorrect username or password";
                }
            }
        else
            {
            toret="ERROR: procesing parameters";
            }
        return toret;
        }
```

```
      /**
     * Validates the parameters
     * @return true if valid parameters
     */
    private boolean processParameters()
        {
        boolean toret=false;
        System.out.println("Number of parameters "+parameterList.size()+"\n");
            if (parameterList.size()==2)
                {
                userName=parameterList.elementAt(0);
                password=parameterList.elementAt(1);
                System.out.println("Moving from parameters to variables");
                System.out.println("Username:"+userName);
                System.out.println("Pass:"+password);
                toret=true;
                }
        return toret;
        }


}
/*
 * To change this template, choose Tools | Templates
 * and open the template in the editor.
 */

package sjavatrack.commands;

import sbd.baseCommand;
/**
 *
 * @author luzem
 */
public class reasing_bug extends baseCommand{

    /**
     * bugCode
     */
    private int bugCode;
    /**
     * new developer to assing
     */
    private String assignedDeveloper;
    static final long serialVersionUID = 2L;

    /**
     * Creates a new reasing_bug command
     */
```

```
   public reasing_bug()
       {
       super();
       help="reasing bugs ";
       isAdminCommand=true;
       }

      /**
    * Executes the command
    * @return the response of execution
    */
    @Override
    public String  execute()
       {
       String toret;
       if (processParameters())
          {
          if (existBug(bugCode))//Check if bug exist
             {
             if (isValidUsername(assignedDeveloper))//check developer
                {
                String SQL;
                SQL="Update bugs set programador='"+assignedDeveloper+"' where
codigo="+bugCode;
                if (super.executeUpdate(SQL))
                   {
                   toret="Bug assigned";
                   }
                else
                   {
                   toret="ERROR: assigning developer";
                   }
                }
             else
                {
                toret="ERROR: Invalid Developer";
                }
             }
          else
             {
             toret="ERROR: bug code not exist";
             }
          }
       else
          {
          toret="ERROR: Invalid parameters";
          }
       return toret;
       }


       /**
```

```
     * Validates the parameters
     * @return true if valid parameters
     */
    private boolean processParameters()
        {
        boolean toret=false;
        System.out.println("Parameter size "+parameterList.size());
            if (parameterList.size()==2)
                {
                try
                    {
                    Integer castingInt;
                    castingInt=new Integer(parameterList.elementAt(0));
                    bugCode=castingInt.intValue();
                    assignedDeveloper=parameterList.elementAt(1);
                    toret=true;
                    }
                catch(NumberFormatException nfe)
                    {
                    toret=false;
                    }
                }
        return toret;
        }

}
/*
 * To change this template, choose Tools | Templates
 * and open the template in the editor.
 */

package sjavatrack.commands;

import sbd.baseCommand;
import java.sql.Connection;
import java.sql.Statement;
import java.sql.SQLException;

/**
 *
 * @author luzem
 */

/*
 * resolve_bug <codigo_bug> <mensaje>
    Cambia el estado de la incidencia <codigo_bug> a “RESUELTO” añadiendo un <mensaje>
    explicativo de dicha resolución. Si la incidencia no estaba asignada al usuario que solicita esta
    operación, se le denegará.
*/
public class resolve_bug extends baseCommand{

    /**
```

```
     * bugCode
     */
    int bugCode;
    /**
     * message
     */
    String message;
    static final long serialVersionUID = 2L;

    /**
     * Creates a new resolve_bug command
     */
    public resolve_bug()
        {
        super();
        help="resolve_bug <codigo_bug> <mensaje>";
        isUserCommand=true;
        }

        /**
     * Executes the command
     * @return the response of execution
     */
     @Override
     public String  execute()
            {
            String toret=null;
            if (processParameters())
                {
                if (!isBugSolved(bugCode))
                    {
                    if(getBugProgrammer(bugCode).equals(userName))
                        {
                        try{
                            Connection conn;
                            Statement st;
                            String SQL;
                            conn=getConnection();
                            st=conn.createStatement();
                            SQL="Update bugs set estado='RESUELTO',
descripcion_resolucion="'+message+"'";
                            SQL+=" where codigo="+bugCode;
                            st.executeUpdate(SQL);
                            toret="Bug Updated";
                            st.close();
                            conn.close();
                            }catch(SQLException sqle)
                                {
                                System.out.println("SQLException: " + sqle.getMessage());
                                System.out.println("SQLState: " + sqle.getSQLState());
                                System.out.println("VendorError: " + sqle.getErrorCode());
                                }
```

```
                }
            else
                {
                if (getBugProgrammer(bugCode).equals(""))
                    {
                    toret="Invalid bug code";
                    }
                else
                    {
                    toret="Not Privilegies for edit this bug";
                    }
                }
              }
            else
                {
                toret="Solving a solved Bug ?????????";
                }
            }
        else
            {
            toret="Not enougth parameters";
            }
        return toret;
        }

     /**
* Validates the parameters
* @return true if valid parameters
*/
private boolean processParameters()
    {
    boolean toret=true;
    if (parameterList.size()<2)
        {
        return false;
        }
    else
        {
        System.out.println("parameterList ="+parameterList.firstElement());
        Integer forCasting;
        forCasting=new Integer(parameterList.firstElement());
        bugCode=forCasting.intValue();
        message="";
        for (int x=1;x<parameterList.size();x++)
            {
            message+=parameterList.elementAt(x);
            if (x!=parameterList.size())
                {
                message+=" ";
                }
            }
        }
```

```
        System.out.println("message:"+message);
        System.out.println("bugCode:"+bugCode);
        return toret;
        }



}
/*
 * To change this template, choose Tools | Templates
 * and open the template in the editor.
 */

package sjavatrack.commands;

import sbd.baseCommand;
/**
 *
 * @author luzem
 */
public class save_bugs extends baseCommand {

    static final long serialVersionUID = 2L;

        /**
     * Creates a new save_bugs command
     */
    public save_bugs()
        {
        super();
        help="Save bugs to a file";
        isUserCommand=true;
        isAdminCommand=true;
        this.toFile=true;
        }

    /**
     * Executes the command
     * @return the response of execution
     */
    @Override
    public String execute()
        {
        String toret;
        list_bugs listado=new list_bugs();
        listado.setPassword(this.getPassword());
        listado.setUserName(this.getUserName());

        for (int x=0;x<parameterList.size();x++)
            {
            this.addParameter(parameterList.elementAt(x));
            }
```

```
        toret=listado.execute();
        return toret;
        }

}
/*
 * To change this template, choose Tools | Templates
 * and open the template in the editor.
 */
package sjavatrack.commands;

import sbd.baseCommand;

/**
 *
 * @author luzem
 */

//set_pass <login_usuario> <new_password_usuario >
//    Cambia en la base de datos javatrack la password de acceso de un usuario existente en el
//    sistema.

public class set_pass extends baseCommand{

    private String userNametoEdit;
    private String newPassword;
    static final long serialVersionUID = 2L;

    /**
     * Creates a new set_pass command
     */
    public set_pass()
        {
        super();
        help="changue's password";
        isAdminCommand=true;
        }
    @Override
    public String  execute()
        {
        String toret="";
        if (processParameters())
            {
            if (isValidUsername(userNametoEdit))
                {
                String SQL="update usuarios set password='";
                SQL+=newPassword;
                SQL+="' where login='";
                SQL+=userNametoEdit;
                SQL+="'";
                if (executeUpdate(SQL))
                    {
```

```
                toret="User password changued";
                }
            else
                {
                toret="Error: Database Error changuing password";
                }
            }
        else
            {
            toret="Error: Username not exist";
            }

        }
    else
        {
        toret="Error: Parameter mistmach ";
        }
    return toret;
    }


  /**
   * Validates the parameters
   * @return true if valid parameters
   */
  private boolean processParameters()
    {
    boolean toret=false;
    System.out.println("Parameter size "+parameterList.size());
      if (parameterList.size()==2)
        {
        userNametoEdit=parameterList.elementAt(0);
        newPassword=parameterList.elementAt(1);
        toret=true;
        }
    return toret;
    }

}
```