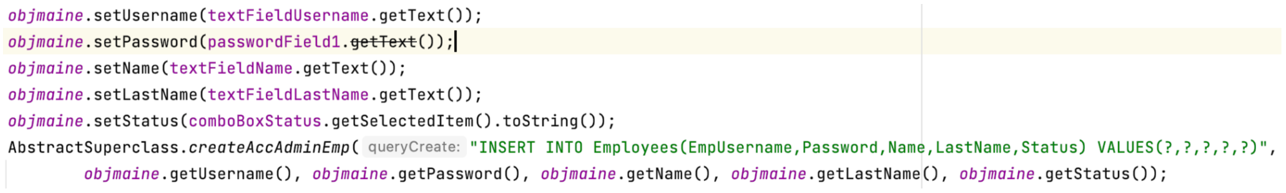
Appendix 1:

<u>Creating an employee account sample query</u>

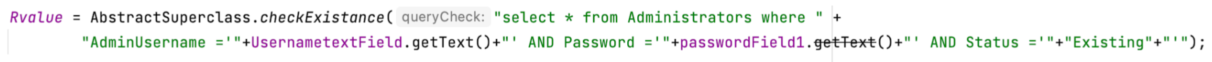


```
objmaine.setUsername(textFieldUsername.getText());
objmaine.setPassword(passwordField1.~~getText~~());
objmaine.setName(textFieldName.getText());
objmaine.setLastName(textFieldLastName.getText());
objmaine.setStatus(comboBoxStatus.getSelectedItem().toString());
AbstractSuperclass.createAccAdminEmp( queryCreate: "INSERT INTO Employees(EmpUsername,Password,Name,LastName,Status) VALUES(?,?,?,?,?)",
        objmaine.getUsername(), objmaine.getPassword(), objmaine.getName(), objmaine.getLastName(), objmaine.getStatus());
```

Appendix 2:

<u>Checking the existence of accounts query</u>

(Sample code used in the login interface)

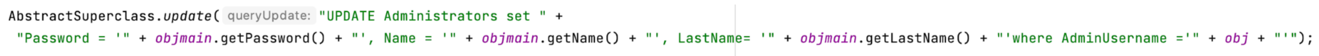


```
Rvalue = AbstractSuperclass.checkExistance( queryCheck: "select * from Administrators where " +
        "AdminUsername ='"+UsernametextField.getText()+"' AND Password ='"+passwordField1.~~getText~~()+"' AND Status ='"+"Existing"+"'");
```

Appendix 3:

<u>Update of account and task information</u>

(Sample code used in the Admin_accounts interface)

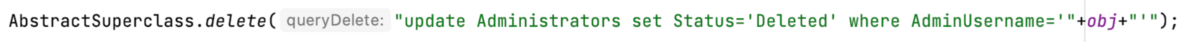


```
AbstractSuperclass.update( queryUpdate: "UPDATE Administrators set " +
 "Password = '" + objmain.getPassword() + "', Name = '" + objmain.getName() + "', LastName= '" + objmain.getLastName() + "'where AdminUsername ='" + obj + "'");
```

Appendix 4:

<u>Logical removal of an account or task</u>

(Sample code used in the Admin_accounts interface)



```
AbstractSuperclass.delete( queryDelete: "update Administrators set Status='Deleted' where AdminUsername='"+obj+"'");
```

<u>Appendix 5:</u>

<u>First interview with client of possible improvements</u>

Me: So can you tell me about an issue that you are currently facing in the company? Or even an operation which you would like to digitalize the processes?

Client: Yes obviously. As you already know, UNIGIS is a private limited company that provides software (TMS platforms) to many different companies. Therefore we receive hundreds of requirements monthly from each client of possible improvements for our software or even of more functionalities they would like to add. We're currently using email as our intermediary of communication between our employees and the clients which is very inefficient. We receive tons of mails with requirements and its very unorganized. Sometimes even clients get angry because we aren't able to answer them due to this lack of organization. Therefore what we need is an easy-to-use system where clients, employees and administrators could login to the system and perform their corresponding tasks. The system should grant different access to the three types of users.

Me: Perfect but who woul you like to create the accounts? All the users by themselves or how?

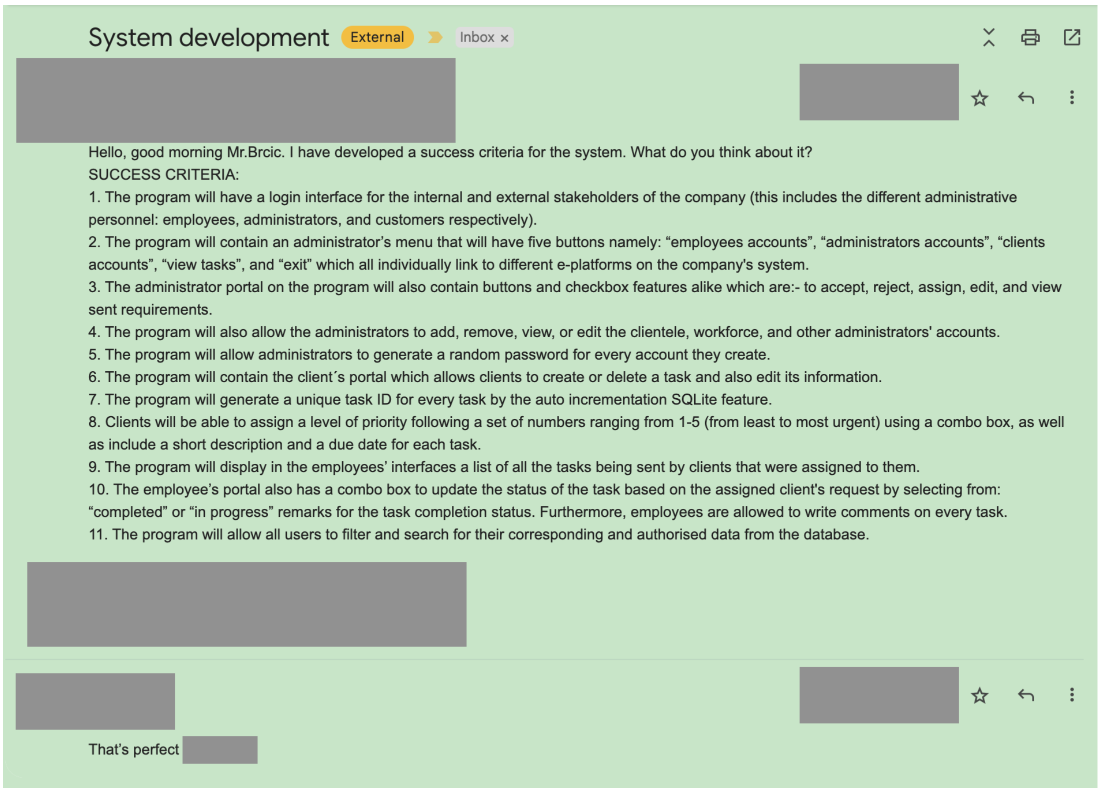
Client: No no, I would like the system to be in the control of administrators. This means that they should be the ones responsible for creating, editing and deleting all the accounts of every type. Clients and employees will have to be informed through an email maybe of their username and password to access their corresponding portal.

Me: Ok and how would you like the flow of tasks to be?

Client: The clients should be able to create tasks with a priority assigned, a due date, details and a comment. After the task is created, an administrator should accept or reject the task and the client should be able to receive a notification according to decision made. If the task is accepted, the administrator should assign the task to the employee they want to finish the task.

Me: Perfect, I will contact you again after I develop a success criteria.

Appendix 6:
Email exchange with client



[redacted] [redacted]

Hello, good morning Mr.Brcic. I have developed a success criteria for the system. What do you think about it?
SUCCESS CRITERIA:
1. The program will have a login interface for the internal and external stakeholders of the company (this includes the different administrative personnel: employees, administrators, and customers respectively).
2. The program will contain an administrator's menu that will have five buttons namely: "employees accounts", "administrators accounts", "clients accounts", "view tasks", and "exit" which all individually link to different e-platforms on the company's system.
3. The administrator portal on the program will also contain buttons and checkbox features alike which are:- to accept, reject, assign, edit, and view sent requirements.
4. The program will also allow the administrators to add, remove, view, or edit the clientele, workforce, and other administrators' accounts.
5. The program will allow administrators to generate a random password for every account they create.
6. The program will contain the client´s portal which allows clients to create or delete a task and also edit its information.
7. The program will generate a unique task ID for every task by the auto incrementation SQLite feature.
8. Clients will be able to assign a level of priority following a set of numbers ranging from 1-5 (from least to most urgent) using a combo box, as well as include a short description and a due date for each task.
9. The program will display in the employees' interfaces a list of all the tasks being sent by clients that were assigned to them.
10. The employee's portal also has a combo box to update the status of the task based on the assigned client's request by selecting from: "completed" or "in progress" remarks for the task completion status. Furthermore, employees are allowed to write comments on every task.
11. The program will allow all users to filter and search for their corresponding and authorised data from the database.

[redacted]

[redacted] [redacted]

That's perfect [redacted]

Appendix 7:
Second interview with client of possible improvements

Me: How do you feel about the system?

Client: I am delighted with the system, but there are some functionalities that I would like to add in the future, and also some improvements in the design to enhance the usability of the program.

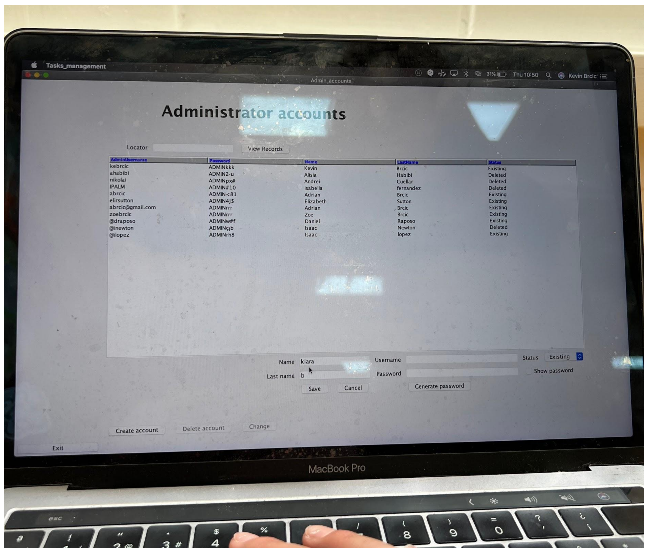
Me: Perfect, what would you like to change or add in future to the system?

Client: I would like the system to include the colors that represent UNIGIS which are red, black, and white to make the system more relatable to the firm. Furthermore, I want the program to have colors assigned to important buttons such as red for the button to delete. I think these will make the system more efficient and improve its usability because the users will not waste time finding the adequate button. Or even icons and images could be added to replace the description of the buttons.

Me: Sounds good, would you also like to change the colors in the JTables displayed? For instance, instead of only having a number to represent the priority of each task, we could also have a color assigned to it. What do you think of this?

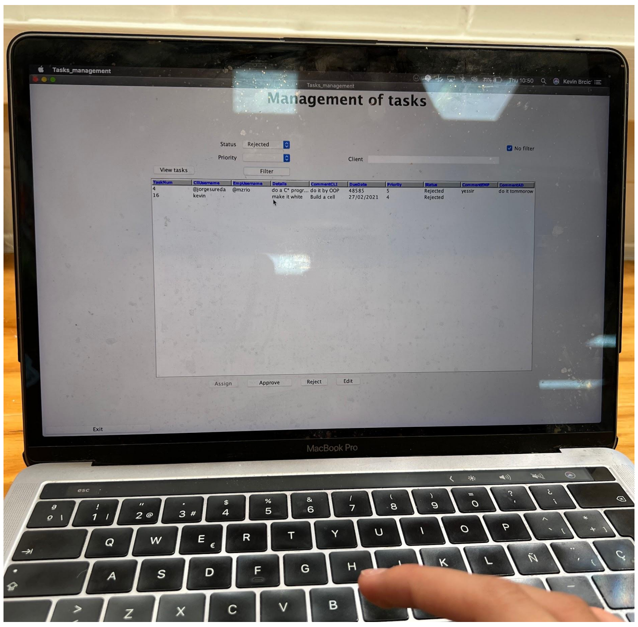
Client: Yes, 100%, that would make the program even more visual and improve its usability as well. Now, getting into future functionalities, I would like administrators to have a section in the menu of statistical analysis. This means an interface that would provide analysis with graphs and tables of the status and assignation of the tasks.
Furthermore, I would also like employees to get notified by email once a task is within five days of the due date. This will ensure that all the employees are aware of the due dates of their tasks.

<u>Appendix 8:</u>
<u>Client testing the system</u>



AdminUsername	Password	Name	LastName	Status
kebrcic	ADMINkkk	Kevin	Brcic	Existing
ahabibi	ADMIN2-u	Alisia	Habibi	Deleted
nikolai	ADMINpx#	Andrei	Cuellar	Deleted
IPALM	ADMIN#10	isabella	fernandez	Deleted
abrcic	ADMIN<81	Adrian	Brcic	Existing
elirsutton	ADMIN4j$	Elizabeth	Sutton	Existing
abrcic@gmail.com	ADMINrrr	Adrian	Brcic	Existing
zoebrcic	ADMINrrr	Zoe	Brcic	Existing
@draposo	ADMINw#f	Daniel	Raposo	Existing
@inewton	ADMINçjb	Isaac	Newton	Deleted
@ilopez	ADMINrh8	Isaac	lopez	Existing

Name: kiara
Last name: b
Username:
Password:
Status: Existing



Management of tasks

Status: Rejected
Priority:
Client:
[✓] No filter

TaskNum	CliUsername	EmpUsername	Details	CommentCLI	DueDate	Priority	Status	CommentEMP	CommentAD
4	@jorgesureda	@mzrio	do a C* progr...	do it by OOP	48585	5	Rejected	yessir	do it tommorow
16	kevin		make it white	Build a cell	27/02/2021	4	Rejected		

<u>Appendix 9:</u>
<u>Source Code</u>

<u>CLASS: AbstractSuperclass</u>

```java
package TaskOrganiser;

import com.toedter.calendar.JDateChooser;
import org.jetbrains.annotations.NotNull;

import javax.swing.*;
import javax.swing.event.ListSelectionListener;
import javax.swing.table.JTableHeader;
import javax.swing.table.TableModel;
import javax.swing.table.TableRowSorter;
import java.awt.*;
import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.text.SimpleDateFormat;
import java.util.Calendar;
import java.util.Random;

/*
Calendar functionality from: Words, G., Idea, W., asktejas, B., Flojea, M.,
freaks, U., 问说网, J., Ethics, I., softwareengineeringcodefoethic, S.,
Website, J., Aplikasi, D., JCalender &#8211; Tech Blicks | Tips, V., JAVA,
K. and java.lang.IllegalStateException: Attempt to mutate in notification
(JDateChooser, J. -. J. D. D. -. P.
Words, Google et al. "Jcalendar - Toedter.Com". Toedter.Com, 2023,
https://toedter.com/jcalendar/. Accessed 20 Sept 2023.
```

```
Downloaded from:
http://www.java2s.com/Code/Jar/j/Downloadjcalendar14jar.htm
*/

public abstract class AbstractSuperclass extends JFrame implements
ListSelectionListener {
   static int value;
   static Connection connection = null;
   public static String USERNAME;


   public static void update(String queryUpdate) {
       /*parameterized method to enhance standardization of update of
records.
       Many classes can use the update method by evoking it and sending
their own query as an argument of the function
        */
       connection = SQLite_Connection.dbConnector();
       try {
           //Prepared statement used to execute the query
           PreparedStatement pstt =
connection.prepareStatement(queryUpdate);
           pstt.execute();
           //message to the user to indicate that the operation was
successful
           JOptionPane.showMessageDialog(null, "Record has been updated
successfully");
           pstt.close();

       } catch (Exception e4) {
           e4.printStackTrace();

       }

   }

   public static void delete(String queryDelete) {
       connection = SQLite_Connection.dbConnector();
       //making sure the user wants to delete through a YES_NO_OPTION
       int action = JOptionPane.showConfirmDialog(null, "Are you sure you
want to delete", "Delete", JOptionPane.YES_NO_OPTION);
       //If the "yes" option is selected...
       if (action == 0) {
           try {
               PreparedStatement pst4 =
connection.prepareStatement(queryDelete);
               pst4.execute();
               //Notify the user that the account has been deleted
               JOptionPane.showMessageDialog(null, "Account is now
'deleted'");
               pst4.close();
           } catch (Exception e2) {
           }
       }
   }

   public static void createAccAdminEmp(String queryCreate, String
TXTusername, String TXTpassword, String TXTname, String TXTlastname, String
TXTstatus ) {
       //parameters in the method to enable reusability of this code.
Hence, the algorithm is standardized and was coded in general terms.
       connection = SQLite_Connection.dbConnector();
       try{
       //inserting into the Administrator table unknown values initially
into the fields that are being stated
       PreparedStatement pst = connection.prepareStatement(queryCreate);
       //Only able to execute command (query) with a prepared statement.
The query is ready for execution
       //Therefore, the prepared statement is running the query

       pst.setString(1, TXTusername);
       //Grab what the user has inputted in the fields and send it to the
database in the correct column index, 1 being the first one.
       pst.setString(2, TXTpassword);
       pst.setString(3, TXTname);
       pst.setString(4, TXTlastname);
       pst.setString(5, TXTstatus);

       pst.execute();
       JOptionPane.showMessageDialog(null, "Account has been successfully
created");
       pst.close();

       }
       catch(Exception e1){

       }

   }

   public static void createAccCli(String queryCreate, String TXTusername,
String TXTpassword, String TXTCname, String TXTstatus ) {
       //The connection with the database needs to be established.
       connection = SQLite_Connection.dbConnector();
       try{
           //inserting into the Administrator table unknown values
initially into the fields that are being stated.
           //This is written on the query.
           PreparedStatement pst =
connection.prepareStatement(queryCreate);
           /*Only able to execute command (query) with a prepared
statement. The query is ready for execution
           Therefore the prepared statement is running the query
            */
           pst.setString(1, TXTusername);
           //Temporarily store what the user has inputted in the fields and
send it to the database.
           pst.setString(2, TXTpassword);
           pst.setString(3, TXTCname);
           pst.setString(4, TXTstatus);
           pst.execute();
           JOptionPane.showMessageDialog(null, "Account has been
successfully created");
           pst.close();

       }
       catch(Exception e1){

       }

   }

   public static @NotNull String generatePassword(String prefix) {

       //creating a string of all characters
       String gen =
"abcdefghijklmnopqrstuvwxyz123456789!·$%/()=¿?*^¨ç><'¡#";
       //creating a random string builder
       StringBuilder code = new StringBuilder();
       StringBuilder codef = new StringBuilder();
       // create an object of the random class
       Random random = new Random();
       // create a variable to specify the length of the randomly picked
       int length = 3;
       // for loop to loop through the selected characters and keep
generating different strings
       // looping from 1-38, not through the characters.
       // Getting a random index and then use the random index to choose
the character from gen
       for (int i = 0; i < length; i++) {
           // generate random index number
           int index = random.nextInt(gen.length());
           //generating character from string by specified index
           char randomChar = gen.charAt(index);
           // append the character to a string builder
           code.append(randomChar);
           //Join the prefix string to the randomly selected character to
have the password
       }
       codef.append(prefix + code);

       //Turning the final code into a string using the toString method
       String FinalRandomString = codef.toString();
       return FinalRandomString;

   }
```

```
   public static void UseLocator(@NotNull String Locatortxt, TableModel
tMod, @NotNull JTable x  ){
       //Parameters to allow all interfaces to make use of it by sending
their JTable, TableModel and the text of the locator.
       //TableRowSorter to implement the RowSorter for a TableModel
       TableRowSorter<TableModel> sorter = new
TableRowSorter<TableModel>(tMod);
       //sorting the JTable
       x.setRowSorter(sorter);
       //.trim() eliminates spaces in the text on the textField entered
       if (Locatortxt.trim().length() ==0){
           sorter.setRowFilter(null);
           //nothing on the textfield means no filter
       }
       else {
           sorter.setRowFilter(RowFilter.regexFilter(Locatortxt));
           //filtering takes place according to what has been written by
the user in the locatorTextField
       }
   }

   public static Integer checkExistance(String queryCheck){
       connection = SQLite_Connection.dbConnector();
       try {
           PreparedStatement pstt2 =
connection.prepareStatement(queryCheck);
           /*A ResultSet is necessary to represent the database results
after the execution of a query
           by the PreparedStatement. The ResultSet object brings data from
the database that satisfies the query.
            */
           ResultSet r2 = pstt2.executeQuery();
           int counting = 0;
           /*A count needs to be established to all the rows selected from
the database that satisfy the query.
           If the count is 1, then there is a record that already exists
with the requirements from the query.
           So existance is true. If the count is 0 then existance is false.
            */
           value = 0;
           while(r2.next()) {
               counting += 1;
           }
           if(counting == 1){
              value = 1;
              return value;
           }
           else if(counting == 0){
               value = 0;
               return value;
           }
           r2.close();
           pstt2.close();
```

```
        }
        catch(Exception e1){
            e1.printStackTrace();}
        return value;
    }

    public static String Calendar( JDateChooser dchooser ){
            SimpleDateFormat sdf = new SimpleDateFormat("dd/MM/yyyy");
            String dt = sdf.format(dchooser.getDate());
            return dt;

    }

    public static void showColumn(JTable table){
        JTableHeader th = table.getTableHeader();
        th.setForeground(Color.BLUE);
        th.setBackground(Color.LIGHT_GRAY);
        th.setFont(new Font("Tahoma",Font.BOLD, 10));
    }

}
```

CLASS: MainAdmin_accounts

```
package TaskOrganiser;

public class MainAdmin_accounts {
    private String Username;
    private String Password;
    private String Name;
    private String LastName;
    private String Status;

    public String getUsername (){
        return Username;
    }
    public void setUsername(String Username){
        this.Username= Username;
    }

    public String getPassword (){
        return Password;
    }
    public void setPassword(String Password){
        this.Password= Password;
    }

    public String getName (){
        return Name;
    }
    public void setName(String Name){
        this.Name= Name;
    }

    public String getLastName (){
        return LastName;
    }
    public void setLastName(String LastName){
        this.LastName= LastName;
    }

    public String getStatus (){
        return Status;
    }
    public void setStatus(String Status){
        this.Status= Status;
    }

}
```

CLASS: MainClient_accounts

```
package TaskOrganiser;

public class MainClient_accounts {
    private String Username;
    private String Password;
    private String CName;
    private String Status;

    public String getUsername (){
        return Username;
    }
    public void setUsername(String Username){
        this.Username= Username;
    }

    public String getPassword (){
        return Password;
    }
    public void setPassword(String Password){
        this.Password= Password;
    }

    public String getCName (){
        return CName;
    }
    public void setCName(String CName){
        this.CName= CName;
    }

    public String getStatus (){
        return Status;
    }
```

```
   public void setStatus(String Status){
       this.Status= Status;
   }
}
```

CLASS: MainClient_portal

```
package TaskOrganiser;

public class MainClient_portal {
   private String commentCLI;
   private String details;
   private String dueDate;
   private String priority;
   private String Tstatus;

   public String getCommentCLI (){
       return commentCLI;
   }
   public void setCommentCLI(String commentCLI){
       this.commentCLI= commentCLI;
   }

   public String getDetails (){
       return details;
   }
   public void setDetails(String details){
       this.details= details;
   }

   public String getDueDate (){
       return dueDate;
   }
   public void setDueDate(String dueDate){
       this.dueDate= dueDate;
   }

   public String getPriority(){
       return priority;
   }
   public void setPriority(String priority){
       this.priority=priority;
   }
   public String getTstatus(){return Tstatus;}

   public void setTstatus(String Tstatus){
       this.Tstatus=Tstatus;
   }

}
```

CLASS: MainEmployee_accounts

```java
package TaskOrganiser;

public class MainEmployee_accounts {
   private String Username;
   private String Password;
   private String Name;
   private String LastName;
   private String Status;

   public String getUsername (){
       return Username;
   }
   public void setUsername(String Username){
       this.Username= Username;
   }

   public String getPassword (){
       return Password;
   }
   public void setPassword(String Password){
       this.Password= Password;
   }

   public String getName (){
       return Name;
   }
   public void setName(String Name){
       this.Name= Name;
   }

   public String getLastName (){
       return LastName;
   }
   public void setLastName(String LastName){
       this.LastName= LastName;
   }

   public String getStatus (){
       return Status;
   }
   public void setStatus(String Status){
       this.Status= Status;
   }
}
```

CLASS: MainEmployee_portal

```java
package TaskOrganiser;

public class MainEmployee_portal {
   private String commentEMP;
   private String Tstatus;
   public String getCommentEMP (){
       return commentEMP;
   }
   public void setCommentEMP(String commentEMP){
       this.commentEMP= commentEMP;
   }
   public String getTstatus(){return Tstatus;}

   public void setTstatus(String Tstatus){
       this.Tstatus=Tstatus;
   }




}
```

CLASS: MainTasks_management

```
public class MainTasks_management {
   private String Tstatus;
   private String commentAD;
   private String Eusername;

   public String getTstatus (){
       return Tstatus;
   }
   public void setTstatus(String Tstatus){
       this.Tstatus= Tstatus;
   }

   public String getCommentAD (){
       return commentAD;
   }
   public void setCommentAD(String commentAD){
       this.commentAD= commentAD;
   }

   public String getEusername (){
       return Eusername;
   }
   public void setEusername(String Eusername){
       this.Eusername= Eusername;
   }



}
```

CLASS: SQLite_Connection

```
package TaskOrganiser;
import java.sql.*;
import javax.swing.*;

public class SQLite_Connection {
   Connection conn = null;
   public static Connection dbConnector(){
       try{
           Class.forName("org.sqlite.JDBC");
           Connection conn =
DriverManager.getConnection("jdbc:sqlite:/Users/KevinBrcic/Downloads/Comput
er_Science_IA/src/TaskOrganiser/Task_Organiser.sqlite");
           //JOptionPane.showMessageDialog(null, "Connection is successful
with Database!");
           return conn;
       }

       catch(Exception e){
           JOptionPane.showMessageDialog(null, e);
           return null;
       }
   }

}
```

CLASS: Admin_accounts

```
package TaskOrganiser;

import net.proteanit.sql.DbUtils;

import javax.swing.*;
import javax.swing.event.ListSelectionEvent;
import javax.swing.table.JTableHeader;
import javax.swing.table.TableModel;
import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.event.KeyAdapter;
import java.awt.event.KeyEvent;
import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.util.Objects;


public class Admin_accounts extends AbstractSuperclass{
   public JPanel Admin_accounts;
   private JTextField textFieldLocator;
   private JTable AdminTable;
   private JTextField textFieldName;
   private JTextField textFieldLastName;
   private JButton generatePasswordButton;
   private JButton saveButton;
   private JButton cancelButton;
```

```
    private JButton changeButton;
    private JButton createAccountButton;
    private JButton deleteAccountButton;
    private JPasswordField passwordField1;
    private JComboBox comboBoxStatus;
    private JButton viewRecordsButton;
    private JLabel LN;
    private JLabel Na;
    private JLabel Pass;
    private JLabel Sta;
    private JCheckBox showPasswordCheckBox;
    private JTextField textFieldUsername;
    private JLabel UserN;
    private JButton updateButton;
    private JButton exitButton;
    Connection connection = null;
    static JFrame aaccFrame = new JFrame("Admin_accounts");
    static Object obj;
    static String status;
    static Admin_accounts newobj = new Admin_accounts();
    static MainAdmin_accounts objmain = new MainAdmin_accounts();


    public Admin_accounts() {

        status =
Objects.requireNonNull(comboBoxStatus.getSelectedItem()).toString();
        connection = SQLite_Connection.dbConnector();
        textFieldName.setVisible(false);
        textFieldLastName.setVisible(false);
        textFieldUsername.setVisible(false);
        comboBoxStatus.setVisible(false);
        passwordField1.setVisible(false);
        saveButton.setVisible(false);
        cancelButton.setVisible(false);
        generatePasswordButton.setVisible(false);
        showPasswordCheckBox.setVisible(false);
        updateButton.setVisible(false);
        Na.setVisible(false);
        Sta.setVisible(false);
        LN.setVisible(false);
        Pass.setVisible(false);
        UserN.setVisible(false);
        deleteAccountButton.setEnabled(false);
        changeButton.setEnabled(false);
        ListSelectionModel selectionModel = AdminTable.getSelectionModel();
        TableModel model = AdminTable.getModel();
        //Add listener to this table in this class
        selectionModel.addListSelectionListener(this);
        //This keyword means that the method is only happening in this
specific class
        textFieldLocator.addKeyListener(new KeyAdapter() {
            @Override
            public void keyReleased(KeyEvent e) {
                super.keyReleased(e);
                try{
                        
AbstractSuperclass.UseLocator(textFieldLocator.getText(),
AdminTable.getModel(), AdminTable );
                }
                catch(Exception e2){
                    e2.printStackTrace();
                }
            }
        });
        viewRecordsButton.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                try{
                    String query1 = "select * from Administrators";
                    PreparedStatement pst =
connection.prepareStatement(query1);
                    ResultSet rs = pst.executeQuery();
                    AdminTable.setModel(DbUtils.resultSetToTableModel(rs));
                    AbstractSuperclass.showColumn(AdminTable);
                    pst.close();
                    rs.close();
                }
                catch(Exception e2){
                    e2.printStackTrace();}
            }
        });
        createAccountButton.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                textFieldName.setVisible(true);
                textFieldLastName.setVisible(true);
                textFieldUsername.setVisible(true);
                comboBoxStatus.setVisible(true);
                passwordField1.setVisible(true);
                saveButton.setVisible(true);
                cancelButton.setVisible(true);
                generatePasswordButton.setVisible(true);
                showPasswordCheckBox.setVisible(true);
                Na.setVisible(true);
                Sta.setVisible(true);
                LN.setVisible(true);
                Pass.setVisible(true);
                UserN.setVisible(true);
            }
        });
        cancelButton.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                textFieldName.setText("");
                textFieldLastName.setText("");
                passwordField1.setText("");
                textFieldUsername.setText("");
                textFieldName.setVisible(false);
                textFieldLastName.setVisible(false);
                textFieldUsername.setVisible(false);
                comboBoxStatus.setVisible(false);
                passwordField1.setVisible(false);
                saveButton.setVisible(false);
                cancelButton.setVisible(false);
                generatePasswordButton.setVisible(false);
                showPasswordCheckBox.setVisible(false);
                updateButton.setVisible(false);
                Na.setVisible(false);
                Sta.setVisible(false);
                LN.setVisible(false);
                Pass.setVisible(false);
                UserN.setVisible(false);
            }
        });
        generatePasswordButton.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                String pa =AbstractSuperclass.generatePassword("ADMIN");
                passwordField1.setText(pa);
            }
        });
        showPasswordCheckBox.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                if (showPasswordCheckBox.isSelected()){
                    // echoe the character and 0 means that it will show
everything and start echoing characters
                    // from the beginning (from index position 0)
                    passwordField1.setEchoChar((char)0);
                }
                else{
                    //If the checkbox is not selected, the password will
hide again with *
                    passwordField1.setEchoChar('*');
                }
            }
        });
        saveButton.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                try{
                    //Checking if any of the fields is empty.
                    if ((textFieldUsername.getText().equals(""))||
(textFieldName.getText().equals(""))||(textFieldLastName.getText().equals("
"))||
                            (passwordField1.getText().equals(""))){
                        JOptionPane.showMessageDialog(null, "You have left
one of the fields empty");
```

```
                   }
                   //Checking that the password contains the prefix for the
type of account being created. In this case "ADMIN".
                   String passwordcheck = passwordField1.getText();
                   if(!passwordcheck.contains("ADMIN")){
                       JOptionPane.showMessageDialog(null, "The password
needs to have 'ADMIN' prefix ");
                   }
                   //Validating that the username is unique and doesn't
exist in the database table where the information will be inserted. The
username is the primary key.
                   Integer valuer =
AbstractSuperclass.checkExistance("select * from Administrators where
AdminUsername ='"+textFieldUsername.getText()+"'");
                   if (valuer == 1){
                       JOptionPane.showMessageDialog(null, "The username
exists");
                   }
                   //Checking the username contains the character '@'
                   if ((!textFieldUsername.getText().contains("@"))){
                       JOptionPane.showMessageDialog(null, "The character
'@' is required in the username");
                   }

                   if ((valuer ==
1)||(textFieldUsername.getText().equals(""))||
(textFieldName.getText().equals(""))||

(textFieldLastName.getText().equals(""))||(passwordField1.getText().equals(
""))||(!passwordcheck.contains("ADMIN"))||
                           (!textFieldUsername.getText().contains("@"))){
                       JOptionPane.showMessageDialog(null, "Invalid");
                   }
                   else {
                       objmain.setUsername(textFieldUsername.getText());
                       objmain.setPassword(passwordField1.getText());
                       objmain.setName(textFieldName.getText());
                       objmain.setLastName(textFieldLastName.getText());

objmain.setStatus(comboBoxStatus.getSelectedItem().toString());
                       AbstractSuperclass.createAccAdminEmp("INSERT INTO
Administrators(AdminUsername,Password,Name,LastName,Status)
VALUES(?,?,?,?,?)",
                               objmain.getUsername(),
objmain.getPassword(), objmain.getName(), objmain.getLastName(),
objmain.getStatus());
                   }
               }
               catch (Exception e2){
                   e2.printStackTrace();
               }
           }
       });
```

```
        deleteAccountButton.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {

                AbstractSuperclass.delete("update Administrators set
Status='Deleted' where AdminUsername='"+obj+"'");
            }
        });

        changeButton.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                textFieldUsername.setVisible(false);
                comboBoxStatus.setVisible(false);
                Sta.setVisible(false);
                UserN.setVisible(false);
                saveButton.setVisible(false);
                textFieldName.setVisible(true);
                textFieldLastName.setVisible(true);
                passwordField1.setVisible(true);
                cancelButton.setVisible(true);
                generatePasswordButton.setVisible(true);
                showPasswordCheckBox.setVisible(true);
                Na.setVisible(true);
                LN.setVisible(true);
                Pass.setVisible(true);
                updateButton.setVisible(true);
            }
        });


        updateButton.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                try {
                    if ((textFieldUsername.getText().equals("")) ||
(textFieldName.getText().equals(" ")) ||
(textFieldLastName.getText().equals(" ")) ||
(passwordField1.getText().equals(" "))) {
                        JOptionPane.showMessageDialog(null, "You have left
one of the fields empty");
                    }
                    String passwordcheck = passwordField1.getText();
                    if (!passwordcheck.contains("ADMIN")) {
                        JOptionPane.showMessageDialog(null, "The password
needs to have 'ADMIN' prefix ");
                    } else {
                        objmain.setPassword(passwordField1.getText());
                        objmain.setName(textFieldName.getText());
                        objmain.setLastName(textFieldLastName.getText());
                        AbstractSuperclass.update("UPDATE Administrators set
Password = '" + objmain.getPassword() + "', Name = '" + objmain.getName() +
"', LastName= '" + objmain.getLastName() + "'where AdminUsername ='" + obj
+ "'");
                    }
                }catch(Exception e7){e7.printStackTrace();}


            }
        });


        exitButton.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {

                JFrame frame = new JFrame("Menu");
                frame.setContentPane(new Menu().Menu);
                frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
                frame.pack();
                frame.setVisible(true);
                aaccFrame.dispose();

            }
        });
    }

    @Override
    public void valueChanged(ListSelectionEvent e) {
        if(!e.getValueIsAdjusting()){ //return true if the selection is
still
            deleteAccountButton.setEnabled(true);
            changeButton.setEnabled(true);
            int SelectedRow = AdminTable.getSelectedRow(); //getSelectedRow
returns an integer of the row selected
            if(SelectedRow >=0){ //selected row ha to be >=0 because it
means you are selecting something from the table, a row.
                TableModel model = AdminTable.getModel();
                //getting selected row and then selected column
                obj = model.getValueAt(SelectedRow, 0); //getting the value
of the row selected and the column that we want.
                // Then storing this data in obj.
                textFieldUsername.setText(obj == null ? "": obj.toString());
                Object obj1 = model.getValueAt(SelectedRow, 1);
                passwordField1.setText(obj1 == null ? "":obj1.toString());
                Object obj2 = model.getValueAt(SelectedRow, 2);
                textFieldName.setText(obj2 == null ? "":obj2.toString());
                Object obj3 = model.getValueAt(SelectedRow, 3);
                textFieldLastName.setText(obj3 == null ?
"":obj3.toString());
                Object obj4 = model.getValueAt(SelectedRow, 4);
                if(obj4.equals("Deleted")){
                    deleteAccountButton.setEnabled(false);
                }
                else{
```

```
                        deleteAccountButton.setEnabled(true);
                    }
                }
            }

    }

    public static void main(String[] args) {
        aaccFrame.setContentPane(new Admin_accounts().Admin_accounts);
        aaccFrame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        aaccFrame.pack();
        aaccFrame.setVisible(true);
    }


}
```

CLASS: Clients_accounts

```
package TaskOrganiser;

import net.proteanit.sql.DbUtils;

import javax.swing.*;
import javax.swing.event.ListSelectionEvent;
import javax.swing.event.ListSelectionListener;
import javax.swing.table.JTableHeader;
import javax.swing.table.TableModel;
import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.event.KeyAdapter;
import java.awt.event.KeyEvent;
import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.util.Objects;

public class Client_accounts extends AbstractSuperclass{
    public JPanel Client_accounts;
    private JTextField textFieldLocator;
    private JTable CliTable;
    private JTextField textFieldCName;
    private JButton generatePasswordButton;
    private JButton saveButton;
    private JButton cancelButton;
    private JButton changeButton;
    private JButton createAccountButton;
    private JButton deleteAccountButton;
    private JPasswordField passwordField1;
    private JComboBox comboBoxStatus;
    private JButton viewRecordsButton;
    private JLabel C_Na;
    private JLabel Pass;
```

```java
    private JLabel Sta;
    private JCheckBox showPasswordCheckBox;
    private JTextField textFieldUsername;
    private JLabel UserN;
    private JButton updateButton;
    private JButton exitButton;
    Connection connection = null;
    static JFrame CaccFrame = new JFrame("Client_accounts");
    static Object obj;
    static String status;
    static Client_accounts newobj = new Client_accounts();
    static MainClient_accounts objmaine = new MainClient_accounts();

    public Client_accounts() {
        status =
Objects.requireNonNull(comboBoxStatus.getSelectedItem()).toString();
        connection = SQLite_Connection.dbConnector();
        textFieldCName.setVisible(false);
        textFieldUsername.setVisible(false);
        comboBoxStatus.setVisible(false);
        passwordField1.setVisible(false);
        saveButton.setVisible(false);
        cancelButton.setVisible(false);
        generatePasswordButton.setVisible(false);
        showPasswordCheckBox.setVisible(false);
        updateButton.setVisible(false);
        C_Na.setVisible(false);
        Sta.setVisible(false);
        Pass.setVisible(false);
        UserN.setVisible(false);
        deleteAccountButton.setEnabled(false);
        changeButton.setEnabled(false);
        ListSelectionModel selectionModel = CliTable.getSelectionModel();
        //Add listener to this table in this class
        selectionModel.addListSelectionListener( this);
        textFieldLocator.addKeyListener(new KeyAdapter() {
            @Override
            public void keyReleased(KeyEvent e) {
                super.keyReleased(e);
                try{

AbstractSuperclass.UseLocator(textFieldLocator.getText(),
CliTable.getModel(), CliTable);
                }
                catch(Exception e2){
                    e2.printStackTrace();
                }
            }
        });
        viewRecordsButton.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                try{
                    String query1 = "select * from Clients";
                    PreparedStatement pst =
connection.prepareStatement(query1);
                    ResultSet rs = pst.executeQuery();
                    CliTable.setModel(DbUtils.resultSetToTableModel(rs));
                    AbstractSuperclass.showColumn(CliTable);
                    pst.close();
                    rs.close();
                }
                catch(Exception e2){
                    e2.printStackTrace();}
            }
        });
        createAccountButton.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                textFieldCName.setVisible(true);
                textFieldUsername.setVisible(true);
                comboBoxStatus.setVisible(true);
                passwordField1.setVisible(true);
                saveButton.setVisible(true);
                cancelButton.setVisible(true);
                generatePasswordButton.setVisible(true);
                showPasswordCheckBox.setVisible(true);
                C_Na.setVisible(true);
                Sta.setVisible(true);
                Pass.setVisible(true);
                UserN.setVisible(true);
            }
        });
        cancelButton.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                textFieldCName.setText("");
                passwordField1.setText("");
                textFieldUsername.setText("");
                textFieldCName.setVisible(false);
                textFieldUsername.setVisible(false);
                comboBoxStatus.setVisible(false);
                passwordField1.setVisible(false);
                saveButton.setVisible(false);
                cancelButton.setVisible(false);
                generatePasswordButton.setVisible(false);
                showPasswordCheckBox.setVisible(false);
                updateButton.setVisible(false);
                C_Na.setVisible(false);
                Sta.setVisible(false);
                Pass.setVisible(false);
                UserN.setVisible(false);
            }
        });
        generatePasswordButton.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                String pa =AbstractSuperclass.generatePassword("CLI");
                passwordField1.setText(pa);
            }
        });
        showPasswordCheckBox.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                if (showPasswordCheckBox.isSelected()){
                    // echoe the character and 0 means that it will show
everything and start echoing characters
                    // from the beginning (from index position 0)
                    passwordField1.setEchoChar((char)0);
                }
                else{
                    //If the checkbox is not selected, the password will
hide again with *
                    passwordField1.setEchoChar('*');
                }
            }
        });
        saveButton.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                try{
                    //Checking if any of the fields is empty.
                    if ((textFieldUsername.getText().equals(""))||
(textFieldCName.getText().equals(""))||
(passwordField1.getText().equals(""))){
                        JOptionPane.showMessageDialog(null, "You have left
one of the fields empty");
                    }
                    //Checking that the password contains the prefix for the
type of account being created. In this case "CLI".
                    String passwordcheck = passwordField1.getText();
                    if(!passwordcheck.contains("CLI")){
                        JOptionPane.showMessageDialog(null, "The password
needs to have 'CLI' prefix ");
                    }
                    //Validating that the username is unique and doesn't
exist in the database table where the information will be inserted. The
username is the primary key.
                    Integer valuer =
AbstractSuperclass.checkExistance("select * from Clients where CliUsername
='"+textFieldUsername.getText()+"'");
                    if (valuer == 1){
                        JOptionPane.showMessageDialog(null, "The username
already exists");
                    }
                    //Checking the username contains the character '@'
                    if ((!textFieldUsername.getText().contains("@"))){
                        JOptionPane.showMessageDialog(null, "The character
'@' is required in the username");
```

```
                    }
                    if ((valuer ==
1)||(textFieldUsername.getText().equals(""))||
(textFieldCName.getText().equals(""))||

(passwordField1.getText().equals(""))||(!passwordcheck.contains("CLI"))||(!
textFieldUsername.getText().contains("@"))){
                        JOptionPane.showMessageDialog(null, "Invalid");
                    }
                    else{
                        objmaine.setUsername(textFieldUsername.getText());
                        objmaine.setPassword(passwordField1.getText());
                        objmaine.setCName(textFieldCName.getText());

objmaine.setStatus(comboBoxStatus.getSelectedItem().toString());
                        AbstractSuperclass.createAccCli("INSERT INTO
Clients(CliUsername,Password,Company,Status) VALUES(?,?,?,?)",
objmaine.getUsername(), objmaine.getPassword(), objmaine.getCName(),
objmaine.getStatus());

                        //AbstractSuperclass.createAccAdminEmp("INSERT INTO
Administrators(AdminUsername,Password,Name,LastName,Status)
VALUES(?,?,?,?,?)", textFieldUsername.getText(), passwordField1.getText(),
textFieldName.getText(), textFieldLastName.getText(), status);
                    }
                }
                catch (Exception e2){
                    e2.printStackTrace();
                }
            }
        });
        deleteAccountButton.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                AbstractSuperclass.delete("update Clients set
Status='Deleted' where CliUsername='"+obj+"'");
            }
        });

        changeButton.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                textFieldUsername.setVisible(false);
                comboBoxStatus.setVisible(false);
                Sta.setVisible(false);
                UserN.setVisible(false);
                saveButton.setVisible(false);
                textFieldCName.setVisible(true);
                passwordField1.setVisible(true);
                cancelButton.setVisible(true);
                generatePasswordButton.setVisible(true);
                showPasswordCheckBox.setVisible(true);
                C_Na.setVisible(true);
                Pass.setVisible(true);
                updateButton.setVisible(true);
            }
        });


        updateButton.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                try {
                    if ((textFieldUsername.getText().equals("")) ||
(textFieldCName.getText().equals(" ")) ||
(passwordField1.getText().equals(" "))) {
                        JOptionPane.showMessageDialog(null, "You have left
one of the fields empty");
                    }
                    String passwordcheck = passwordField1.getText();
                    if (!passwordcheck.contains("CLI")) {
                        JOptionPane.showMessageDialog(null, "The password
needs to have 'CLI' prefix ");
                    } else {
                        objmaine.setPassword(passwordField1.getText());
                        objmaine.setCName(textFieldCName.getText());
                        AbstractSuperclass.update("UPDATE Clients set
Password = '" + objmaine.getPassword() + "', Company = '" +
objmaine.getCName() +  "'where CliUsername ='" + obj + "'");
                    }
                }catch(Exception e7){e7.printStackTrace();}


            }
        });


        exitButton.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {

                JFrame frame = new JFrame("Menu");
                frame.setContentPane(new Menu().Menu);
                frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
                frame.pack();
                frame.setVisible(true);
                CaccFrame.dispose();

            }
        });


    }

    @Override
    public void valueChanged(ListSelectionEvent e) {
        if(!e.getValueIsAdjusting()){ //return true if the selection is
still
            deleteAccountButton.setEnabled(true);
            changeButton.setEnabled(true);
            int SelectedRow = CliTable.getSelectedRow(); //getSelectedRow
returns an integer of the row selected
            if(SelectedRow >=0){ //selected row ha to be >=0 because it
means you are selecting something from the table, a row.
                TableModel model = CliTable.getModel();
                //getting selected row and then selected column
                obj = model.getValueAt(SelectedRow, 0); //getting the value
of the row selected and the column that we want. Then storing this data in
obj1.
                textFieldUsername.setText(obj == null ? "": obj.toString());
                Object obj1 = model.getValueAt(SelectedRow, 1);
                passwordField1.setText(obj1 == null ? "":obj1.toString());
                Object obj2 = model.getValueAt(SelectedRow, 2);
                textFieldCName.setText(obj2 == null ? "":obj2.toString());
                Object obj3 = model.getValueAt(SelectedRow, 3);
                if(obj3.equals("Deleted")){
                    deleteAccountButton.setEnabled(false);
                }
                else{
                    deleteAccountButton.setEnabled(true);
                }

            }
        }
    }

    public static void main(String[] args) {
        CaccFrame.setContentPane(new Client_accounts().Client_accounts);
        CaccFrame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        CaccFrame.pack();
        CaccFrame.setVisible(true);
    }


}
```

CLASS: Client_portal

```
package TaskOrganiser;

import com.toedter.calendar.JDateChooser;
import net.proteanit.sql.DbUtils;

import javax.swing.*;
import javax.swing.event.ListSelectionEvent;
import javax.swing.table.TableModel;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.sql.Connection;
```

```java
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.text.SimpleDateFormat;
import java.util.ArrayList;
import java.util.Calendar;
import java.util.Hashtable;
import java.util.Objects;

public class Client_portal extends AbstractSuperclass {
   private JComboBox StatusCbox;
   private JComboBox PriorityCbox;
   private JButton filterButton;
   private JTable tableTasks;
   private JTextArea textAreaCommentCli;
   private JComboBox comboBoxnewPriority;
   private JButton saveButton;
   private JButton cancelButton;
   private JLabel DD;
   private JLabel Comm;
   private JLabel Priority;
   private JButton viewButton;
   public JPanel Client_portal;
   private JButton exitButton;
   private JTextField textFieldDetails;
   private JButton createTaskButton;
   private JButton deleteTaskButton;
   private JButton editTaskButton;
   private JLabel detailsLA;
   private JButton updateButton;
   private JPanel calendar;
   private JPanel FilCalen;
   private JCheckBox checkBoxdate;
   private JLabel FDD;
   Connection connection = null;
   static JFrame clientFrame = new JFrame("Client_portal");
   static Object obj;
   static Object obj3;
   static Object obj4;
   static String status;
   static MainClient_portal objmaine = new MainClient_portal();
   Calendar calen = Calendar.getInstance();
   JDateChooser datechos = new JDateChooser(calen.getTime());
   static String prefix;
   static String dateYN;
   static String filter_query;
   Hashtable<String, String> filterQuerys = new Hashtable<>();
   ArrayList<String> filters_chosen = new ArrayList<String>(5);
   Calendar FiltCalen = Calendar.getInstance();
   JDateChooser fdatechos = new JDateChooser(FiltCalen.getTime());


   public Client_portal() {
       connection = SQLite_Connection.dbConnector();
```

```
        ListSelectionModel selectionModel = tableTasks.getSelectionModel();
        //Add listener to this table in this class
        selectionModel.addListSelectionListener(this);

        //formatting the date
        datechos.setDateFormatString("dd/MM/yyyy");
        fdatechos.setDateFormatString("dd/MM/yyyy");
        //displaying chosen date on panel
        calendar.add(datechos);
        FilCalen.add(fdatechos);


        calendar.setVisible(false);
        DD.setVisible(false);
        Comm.setVisible(false);
        textAreaCommentCli.setVisible(false);
        comboBoxnewPriority.setVisible(false);
        Priority.setVisible(false);
        textFieldDetails.setVisible(false);
        detailsLA.setVisible(false);
        saveButton.setVisible(false);
        cancelButton.setVisible(false);
        updateButton.setVisible(false);



        exitButton.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {

                JFrame frame = new JFrame("Login");
                frame.setContentPane(new Login().Login);
                frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
                frame.pack();
                frame.setVisible(true);
                clientFrame.dispose();
            }
        });
        viewButton.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                try{
                    String query1 = "select
TaskNum,Details,CommentCLI,DueDate,Priority,Status,CommentEMP from Tasks
where CliUsername = '"+USERNAME+"'";
                    PreparedStatement pst =
connection.prepareStatement(query1);
                    ResultSet rs = pst.executeQuery();
                    tableTasks.setModel(DbUtils.resultSetToTableModel(rs));
                    AbstractSuperclass.showColumn(tableTasks);
                    pst.close();
                    rs.close();
                }
                catch(Exception e2){
                    e2.printStackTrace();}
            }
        });
        filterButton.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                try{
                    if(checkBoxdate.isSelected()){
                        //Due date filter not used
                        dateYN = "";
                    }
                    else{
                        //Obtaining the date chosen
                        dateYN = AbstractSuperclass.Calendar(fdatechos);
                    }
                    //Standard query that all filters require to select the
correct data from the database
                    prefix = "select
TaskNum,Details,CommentCLI,DueDate,Priority,Status,CommentEMP from Tasks
where CliUsername='"+USERNAME+"'";
                    //HashTable that defines a name to each filter and its
corresponding query that will append the "prefix" if the filter is
selected.
                    filterQuerys.put("Stat", " AND Status
='"+StatusCbox.getSelectedItem().toString()+"'");
                    filterQuerys.put("Prior", "AND Priority ='"+
PriorityCbox.getSelectedItem().toString()+"'");
                    filterQuerys.put("DueDate", "AND DueDate
='"+dateYN+"'");
                    //series of conditions to check the filters being used
and append them to an array list in an order.
                    if
(!StatusCbox.getSelectedItem().toString().equals("")){
                        filters_chosen.add(0, "Stat");
                        if
(!PriorityCbox.getSelectedItem().toString().equals("")){
                            filters_chosen.add(1, "Prior");
                            if (!dateYN.equals("")){
                                filters_chosen.add(2, "DueDate");
                            }
                        }
                        if (!dateYN.equals("")){
                            filters_chosen.add(1, "DueDate");

                        }
                    }
                    else if
(!PriorityCbox.getSelectedItem().toString().equals("")){
                        filters_chosen.add(0, "Prior");
                        if
(!StatusCbox.getSelectedItem().toString().equals("")){
                            filters_chosen.add(1, "Stat");
```

```
                           if (!dateYN.equals("")){
                               filters_chosen.add(2, "DueDate");
                           }
                       }
                       if (!dateYN.equals("")){
                           filters_chosen.add(1, "DueDate");
                       }
                   }
                   else {
                       filters_chosen.add(0, "DueDate");
                       if
(!StatusCbox.getSelectedItem().toString().equals("")) {
                           filters_chosen.add(1, "Stat");
                           if
(!PriorityCbox.getSelectedItem().toString().equals("")){
                               filters_chosen.add(2, "Prior");}
                       }
                       if
(!PriorityCbox.getSelectedItem().toString().equals("")){
                           filters_chosen.add(1, "Prior");
                       }

                   }
                   String filter_query1 = "";
                   //Looping through the array that has the "keys" of the
filters used
                   for(int x =0;x<filters_chosen.size();x++){
                       //Accesing the "value" of the "keys" of the filters
which are querys.
                       String info =
filterQuerys.get(filters_chosen.get(x));
                       //Joining the "values" which are querys of the
filters selected.
                       filter_query1 = filter_query1  +" "+ info+ " ";
                   }
                   //Adding the prefix query to the query constructed by
the filters selected.
                   filter_query =  prefix + " "+ filter_query1 ;
                   //Executing the query
                   PreparedStatement pst =
connection.prepareStatement(filter_query);
                   ResultSet rs = pst.executeQuery();
                   tableTasks.setModel(DbUtils.resultSetToTableModel(rs));
                   AbstractSuperclass.showColumn(tableTasks);
                   pst.close();
                   rs.close();
                   filters_chosen.clear();
               }
               catch(Exception e2){
                   e2.printStackTrace();}

           }
       });
```

```
        createTaskButton.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                calendar.setVisible(true);
                DD.setVisible(true);
                Comm.setVisible(true);
                textAreaCommentCli.setVisible(true);
                comboBoxnewPriority.setVisible(true);
                Priority.setVisible(true);
                textFieldDetails.setVisible(true);
                detailsLA.setVisible(true);
                saveButton.setVisible(true);
                cancelButton.setVisible(true);
                updateButton.setVisible(false);
                deleteTaskButton.setEnabled(false);
                editTaskButton.setEnabled(false);
            }
        });

        deleteTaskButton.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                AbstractSuperclass.update("UPDATE Tasks set Status =
'"+"Deleted"+"' where TaskNum ='"+obj+"'");
            }
        });
        saveButton.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                try{

if((textAreaCommentCli.getText().equals(""))||(textFieldDetails.getText().e
quals(""))){
                        JOptionPane.showMessageDialog(null, "There are
fields empty");
                    }
                    else{
                        //inserting into the Administrator table unknown
values initially into the fields that are being stated
                        String query = "INSERT INTO
Tasks(TaskNum,CliUsername,EmpUsername,Details,CommentCLI,DueDate,Priority,S
tatus,CommentEMP,CommentAD) VALUES(?,?,?,?,?,?,?,?,?,?)";
                        PreparedStatement pst =
connection.prepareStatement(query);
                        //Only able to execute command (query) with a
prepared statement. The query is ready for execution
                        //Therefore the prepared statement is running the
query
                        objmaine.setDetails(textFieldDetails.getText());

objmaine.setCommentCLI(textAreaCommentCli.getText());

objmaine.setDueDate(AbstractSuperclass.Calendar(datechos));
objmaine.setPriority(comboBoxnewPriority.getSelectedItem().toString());
                        objmaine.setTstatus("Pending");


                        pst.setInt(3, 1);
                        pst.setString(2, USERNAME);
                        pst.setString(3, null);
                        //Grab what the user has inputted in the textfields
and send it to the database
                        pst.setString(4, objmaine.getDetails());
                        pst.setString(5, objmaine.getCommentCLI());
                        pst.setString(6, objmaine.getDueDate());
                        pst.setString(7, objmaine.getPriority());
                        pst.setString(8, objmaine.getTstatus());
                        pst.setString(9, null);
                        pst.setString(10, null);

                        pst.execute();
                        JOptionPane.showMessageDialog(null, "Task has been
successfully created");
                        pst.close();}

                }
                catch(Exception e1){

                }

            }
        });

        updateButton.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                try{

if((textAreaCommentCli.getText().equals(""))||(textFieldDetails.getText().e
quals(""))){
                        JOptionPane.showMessageDialog(null, "There are
fields empty");
                    }
                    else {
                        Integer ConfMess =
JOptionPane.showConfirmDialog(null, "Are you sure you want to update?",
"Update", JOptionPane.YES_NO_OPTION);
                        if (ConfMess== 0) {
                            objmaine.setDetails(textFieldDetails.getText());

objmaine.setCommentCLI(textAreaCommentCli.getText());

objmaine.setDueDate(AbstractSuperclass.Calendar(datechos));
                            AbstractSuperclass.update("UPDATE Tasks set
CommentCLI = '" + objmaine.getCommentCLI() + "', Details = '" +
objmaine.getDetails() + "', DueDate= '" + objmaine.getDueDate() + "'where
TaskNum ='" + obj + "'");
                        }
                    }
                }
                catch(Exception e1){e1.printStackTrace();}
            }
        });
        editTaskButton.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                calendar.setVisible(true);
                DD.setVisible(true);
                Comm.setVisible(true);
                textAreaCommentCli.setVisible(true);
                comboBoxnewPriority.setVisible(false);
                Priority.setVisible(false);
                textFieldDetails.setVisible(true);
                detailsLA.setVisible(true);
                saveButton.setVisible(false);
                cancelButton.setVisible(true);
                updateButton.setVisible(true);
                createTaskButton.setEnabled(false);
                deleteTaskButton.setEnabled(false);


            }
        });
        cancelButton.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                calendar.setVisible(false);
                DD.setVisible(false);
                Comm.setVisible(false);
                textAreaCommentCli.setVisible(false);
                comboBoxnewPriority.setVisible(false);
                Priority.setVisible(false);
                textFieldDetails.setVisible(false);
                detailsLA.setVisible(false);
                saveButton.setVisible(false);
                cancelButton.setVisible(false);
                updateButton.setVisible(false);
                editTaskButton.setEnabled(true);
                createTaskButton.setEnabled(true);

            }
        });
        checkBoxdate.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                if(checkBoxdate.isSelected()){
                    FilCalen.setVisible(false);
                    FDD.setVisible(false);
                    dateYN = "";
                }
                else{
                    FilCalen.setVisible(true);
                    FDD.setVisible(true);
                }
            }
        });
    }

    @Override
    public void valueChanged(ListSelectionEvent e) {
        if(!e.getValueIsAdjusting()){ //return true if the selection is
still
            int SelectedRow = tableTasks.getSelectedRow(); //getSelectedRow
returns an integer of the row selected
            if(SelectedRow >=0){ //selected row ha to be >=0 because it
means you are selecting something from the table, a row.
                TableModel model = tableTasks.getModel();
                //getting selected row and then selected column
                obj = model.getValueAt(SelectedRow, 0); //getting the value
of the row selected and the column that we want. Then storing this data in
obj1.
                Object obj2 = model.getValueAt(SelectedRow, 1);
                textFieldDetails.setText(obj2 == null ? "":obj2.toString());
                Object obj5 = model.getValueAt(SelectedRow, 2 );
                textAreaCommentCli.setText(obj5 == null ?
"":obj5.toString());
                Object obj3 = model.getValueAt(SelectedRow, 5);

                if (!obj3.equals("Pending")){
                    deleteTaskButton.setEnabled(false);
                }
                else{
                    deleteTaskButton.setEnabled(true);
                }
                if (obj3.equals("Deleted")){
                    editTaskButton.setEnabled(false);
                }
                else{editTaskButton.setEnabled(true);}

            }
        }
    }

    public static void main(String[] args) {
        clientFrame.setContentPane(new Client_portal().Client_portal);
        clientFrame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        clientFrame.pack();
        clientFrame.setVisible(true);
    }
}
```

CLASS: Employee_accounts

```
package TaskOrganiser;

import net.proteanit.sql.DbUtils;

import javax.swing.*;
import javax.swing.event.ListSelectionEvent;
import javax.swing.table.JTableHeader;
import javax.swing.table.TableModel;
import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.event.KeyAdapter;
import java.awt.event.KeyEvent;
import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.util.Objects;

public class Employee_accounts extends AbstractSuperclass {
   public JPanel Employee_accounts;
   private JTextField textFieldLocator;
   private JTable EmpTable;
   private JTextField textFieldName;
   private JTextField textFieldLastName;
   private JButton generatePasswordButton;
   private JButton saveButton;
   private JButton cancelButton;
   private JButton changeButton;
   private JButton createAccountButton;
   private JButton deleteAccountButton;
   private JPasswordField passwordField1;
   private JComboBox comboBoxStatus;
   private JButton viewRecordsButton;
   private JLabel LN;
   private JLabel Na;
   private JLabel Pass;
   private JLabel Sta;
   private JCheckBox showPasswordCheckBox;
   private JTextField textFieldUsername;
   private JLabel UserN;
   private JButton updateButton;
   private JButton exitButton;
   Connection connection = null;
   static JFrame EaccFrame = new JFrame("Employee_accounts");
   static Object obj;
   static String status;
   static Employee_accounts newobj = new Employee_accounts();
   static MainEmployee_accounts objmaine = new MainEmployee_accounts();

   public Employee_accounts() {
       status =
Objects.requireNonNull(comboBoxStatus.getSelectedItem()).toString();
       connection = SQLite_Connection.dbConnector();
       textFieldName.setVisible(false);
       textFieldLastName.setVisible(false);
       textFieldUsername.setVisible(false);
       comboBoxStatus.setVisible(false);
       passwordField1.setVisible(false);
       saveButton.setVisible(false);
       cancelButton.setVisible(false);
       generatePasswordButton.setVisible(false);
       showPasswordCheckBox.setVisible(false);
       updateButton.setVisible(false);
       Na.setVisible(false);
       Sta.setVisible(false);
       LN.setVisible(false);
       Pass.setVisible(false);
       UserN.setVisible(false);
       deleteAccountButton.setEnabled(false);
       changeButton.setEnabled(false);
       ListSelectionModel selectionModel = EmpTable.getSelectionModel();
       //Add listener to this table in this class
       selectionModel.addListSelectionListener(this);
       textFieldLocator.addKeyListener(new KeyAdapter() {
           @Override
           public void keyReleased(KeyEvent e) {
               super.keyReleased(e);
               try{

AbstractSuperclass.UseLocator(textFieldLocator.getText(),
EmpTable.getModel(), EmpTable);
               }
               catch(Exception e2){
                   e2.printStackTrace();
               }
           }
       });
       viewRecordsButton.addActionListener(new ActionListener() {
           @Override
           public void actionPerformed(ActionEvent e) {
               try{
                   String query1 = "select * from Employees";
                   PreparedStatement pst =
connection.prepareStatement(query1);
                   ResultSet rs = pst.executeQuery();
                   EmpTable.setModel(DbUtils.resultSetToTableModel(rs));
                   AbstractSuperclass.showColumn(EmpTable);
                   pst.close();
                   rs.close();
               }
               catch(Exception e2){
                   e2.printStackTrace();}
           }
```

```java
        });
        createAccountButton.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                textFieldName.setVisible(true);
                textFieldLastName.setVisible(true);
                textFieldUsername.setVisible(true);
                comboBoxStatus.setVisible(true);
                passwordField1.setVisible(true);
                saveButton.setVisible(true);
                cancelButton.setVisible(true);
                generatePasswordButton.setVisible(true);
                showPasswordCheckBox.setVisible(true);
                Na.setVisible(true);
                Sta.setVisible(true);
                LN.setVisible(true);
                Pass.setVisible(true);
                UserN.setVisible(true);
            }
        });
        cancelButton.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                textFieldName.setText("");
                textFieldLastName.setText("");
                passwordField1.setText("");
                textFieldUsername.setText("");
                textFieldName.setVisible(false);
                textFieldLastName.setVisible(false);
                textFieldUsername.setVisible(false);
                comboBoxStatus.setVisible(false);
                passwordField1.setVisible(false);
                saveButton.setVisible(false);
                cancelButton.setVisible(false);
                generatePasswordButton.setVisible(false);
                showPasswordCheckBox.setVisible(false);
                updateButton.setVisible(false);
                Na.setVisible(false);
                Sta.setVisible(false);
                LN.setVisible(false);
                Pass.setVisible(false);
                UserN.setVisible(false);
            }
        });
        generatePasswordButton.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                String pa =AbstractSuperclass.generatePassword("EMP");
                passwordField1.setText(pa);
            }
        });
        showPasswordCheckBox.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                if (showPasswordCheckBox.isSelected()){
                    // echoe the character and 0 means that it will show
everything and start echoing characters
                    // from the beginning (from index position 0)
                    passwordField1.setEchoChar((char)0);
                }
                else{
                    //If the checkbox is not selected, the password will
hide again with *
                    passwordField1.setEchoChar('*');
                }
            }
        });
        saveButton.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                try{
                    if ((textFieldUsername.getText().equals(""))||
(textFieldName.getText().equals(""))||(textFieldLastName.getText().equals("
"))|| (passwordField1.getText().equals(""))){
                        JOptionPane.showMessageDialog(null, "You have left
one of the fields empty");
                    }
                    String passwordcheck = passwordField1.getText();
                    if(!passwordcheck.contains("EMP")){
                        JOptionPane.showMessageDialog(null, "The password
needs to have 'EMP' prefix ");
                    }
                    Integer valuer =
AbstractSuperclass.checkExistance("select * from Employees where
EmpUsername ='"+textFieldUsername.getText()+"'");
                    if (valuer == 1){
                        JOptionPane.showMessageDialog(null, "The username
already exists");
                    }
                    if ((!textFieldUsername.getText().contains("@"))){
                        JOptionPane.showMessageDialog(null, "The character
'@' is required in the username");
                    }

                    if ((valuer ==
1)||(textFieldUsername.getText().equals(""))||
(textFieldName.getText().equals(""))||(textFieldLastName.getText().equals("
"))||

(passwordField1.getText().equals(""))||(!passwordcheck.contains("EMP"))||(!
textFieldUsername.getText().contains("@"))){
                        JOptionPane.showMessageDialog(null, "Invalid");
                    }
                    else{
                        objmaine.setUsername(textFieldUsername.getText());
                        objmaine.setPassword(passwordField1.getText());
                        objmaine.setName(textFieldName.getText());
                        objmaine.setLastName(textFieldLastName.getText());

objmaine.setStatus(comboBoxStatus.getSelectedItem().toString());
                        AbstractSuperclass.createAccAdminEmp("INSERT INTO
Employees(EmpUsername,Password,Name,LastName,Status) VALUES(?,?,?,?,?)",
objmaine.getUsername(), objmaine.getPassword(), objmaine.getName(),
objmaine.getLastName(), objmaine.getStatus());

                        //AbstractSuperclass.createAccAdminEmp("INSERT INTO
Administrators(AdminUsername,Password,Name,LastName,Status)
VALUES(?,?,?,?,?)", textFieldUsername.getText(), passwordField1.getText(),
textFieldName.getText(), textFieldLastName.getText(), status);
                    }
                }
                catch (Exception e2){
                    e2.printStackTrace();
                }
            }
        });
        deleteAccountButton.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                AbstractSuperclass.delete("update Employees set
Status='Deleted' where EmpUsername='"+obj+"'");
            }
        });

        changeButton.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                textFieldUsername.setVisible(false);
                comboBoxStatus.setVisible(false);
                Sta.setVisible(false);
                UserN.setVisible(false);
                saveButton.setVisible(false);
                textFieldName.setVisible(true);
                textFieldLastName.setVisible(true);
                passwordField1.setVisible(true);
                cancelButton.setVisible(true);
                generatePasswordButton.setVisible(true);
                showPasswordCheckBox.setVisible(true);
                Na.setVisible(true);
                LN.setVisible(true);
                Pass.setVisible(true);
                updateButton.setVisible(true);
            }
        });


        updateButton.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                try {
                    if ((textFieldUsername.getText().equals("")) ||
(textFieldName.getText().equals(" ")) ||
(textFieldLastName.getText().equals(" ")) ||
(passwordField1.getText().equals(" "))) {
                        JOptionPane.showMessageDialog(null, "You have left
one of the fields empty");
                    }
                    String passwordcheck = passwordField1.getText();
                    if (!passwordcheck.contains("EMP")) {
                        JOptionPane.showMessageDialog(null, "The password
needs to have 'EMP' prefix ");
                    } else {
                        objmaine.setPassword(passwordField1.getText());
                        objmaine.setName(textFieldName.getText());
                        objmaine.setLastName(textFieldLastName.getText());
                        AbstractSuperclass.update("UPDATE Employees set
Password = '" + objmaine.getPassword() + "', Name = '" + objmaine.getName()
+ "', LastName= '" + objmaine.getLastName() + "'where EmpUsername ='" + obj
+ "'");
                    }
                }catch(Exception e7){e7.printStackTrace();}


            }
        });


        exitButton.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {

                JFrame frame = new JFrame("Menu");
                frame.setContentPane(new Menu().Menu);
                frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
                frame.pack();
                frame.setVisible(true);
                EaccFrame.dispose();

            }
        });


    }

    @Override
    public void valueChanged(ListSelectionEvent e) {
        if(!e.getValueIsAdjusting()){ //return true if the selection is
still
            deleteAccountButton.setEnabled(true);
            changeButton.setEnabled(true);
            int SelectedRow = EmpTable.getSelectedRow(); //getSelectedRow
returns an integer of the row selected
```

```
            if(SelectedRow >=0){ //selected row ha to be >=0 because it
means you are selecting something from the table, a row.
                TableModel model = EmpTable.getModel();
                //getting selected row and then selected column
                obj = model.getValueAt(SelectedRow, 0); //getting the value
of the row selected and the column that we want. Then storing this data in
obj1.
                textFieldUsername.setText(obj == null ? "": obj.toString());
                Object obj1 = model.getValueAt(SelectedRow, 1);
                passwordField1.setText(obj1 == null ? "":obj1.toString());
                Object obj2 = model.getValueAt(SelectedRow, 2);
                textFieldName.setText(obj2 == null ? "":obj2.toString());
                Object obj3 = model.getValueAt(SelectedRow, 3);
                textFieldLastName.setText(obj3 == null ?
"":obj3.toString());
                Object obj4 = model.getValueAt(SelectedRow, 4);
                if(obj4.equals("Deleted")){
                    deleteAccountButton.setEnabled(false);
                }
                else{
                    deleteAccountButton.setEnabled(true);
                }

            }
        }
    }

    public static void main(String[] args) {
        EaccFrame.setContentPane(new Employee_accounts().Employee_accounts);
        EaccFrame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        EaccFrame.pack();
        EaccFrame.setVisible(true);
    }
}
```

CLASS: Employee_portal

```
package TaskOrganiser;

import com.toedter.calendar.JDateChooser;
import net.proteanit.sql.DbUtils;

import javax.swing.*;
import javax.swing.event.ListSelectionEvent;
import javax.swing.table.TableModel;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.util.Calendar;
import java.util.Hashtable;
import java.util.*;
```

```java
public class Employee_portal extends AbstractSuperclass {
   private JComboBox StatusCbox;
   private JComboBox PriorityCbox;
   private JButton filterButton;
   private JTable tableTasks;
   private JTextArea textAreaCommentEmp;
   private JButton saveButton;
   private JButton cancelButton;
   private JLabel StatusL;
   private JLabel Comm;
   private JButton viewButton;
   public JPanel Employee_portal;
   private JButton exitButton;
   private JButton editTaskButton;
   private JTextField textFieldClient;
   private JComboBox comboBoxnewStatus;
   private JPanel calendar;
   private JCheckBox checkBoxdate;
   private JLabel FDD;
   Connection connection = null;
   static JFrame employeeFrame = new JFrame("Employee_portal");
   static Object obj;
   //static Employee_portal newobj = new Employee_portal();
   static MainEmployee_portal objmaine = new MainEmployee_portal();
   static String prefix;
   static String dateYN;
   static String filter_query;
   Calendar calen = Calendar.getInstance();
   JDateChooser datechos = new JDateChooser(calen.getTime());
   Hashtable<String, String> filterQuerys = new Hashtable<>();
   ArrayList<String> filters_chosen = new ArrayList<String>(5);



   public Employee_portal() {
       connection = SQLite_Connection.dbConnector();
       ListSelectionModel selectionModel = tableTasks.getSelectionModel();
       //Add listener to this table in this class
       selectionModel.addListSelectionListener(this);
       textAreaCommentEmp.setVisible(false);
       Comm.setVisible(false);
       StatusL.setVisible(false);
       comboBoxnewStatus.setVisible(false);
       saveButton.setVisible(false);
       cancelButton.setVisible(false);
       datechos.setDateFormatString("dd/MM/yyyy");
       calendar.add(datechos);
```

```
        exitButton.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {

                JFrame frame = new JFrame("Login");
                frame.setContentPane(new Login().Login);
                frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
                frame.pack();
                frame.setVisible(true);
                employeeFrame.dispose();
            }
        });
        editTaskButton.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                textAreaCommentEmp.setVisible(true);
                Comm.setVisible(true);
                StatusL.setVisible(true);
                comboBoxnewStatus.setVisible(true);
                saveButton.setVisible(true);
                cancelButton.setVisible(true);
            }
        });
        cancelButton.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                textAreaCommentEmp.setVisible(false);
                Comm.setVisible(false);
                StatusL.setVisible(false);
                comboBoxnewStatus.setVisible(false);
                saveButton.setVisible(false);
                cancelButton.setVisible(false);
            }
        });
        viewButton.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                try{
                    String query1 = "select
TaskNum,CliUsername,Details,CommentCLI,DueDate,Priority,Status,CommentEMP,C
ommentAD from Tasks where EmpUsername = '"+USERNAME+"'";
                    PreparedStatement pst =
connection.prepareStatement(query1);
                    ResultSet rs = pst.executeQuery();
                    tableTasks.setModel(DbUtils.resultSetToTableModel(rs));
                    AbstractSuperclass.showColumn(tableTasks);
                    pst.close();
                    rs.close();
                }
                catch(Exception e2){
                    e2.printStackTrace();}
            }
        });
        saveButton.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                try{
                    if((textAreaCommentEmp.getText().equals(""))){
                        JOptionPane.showMessageDialog(null, "The comment 
area needs to be filled");
                    }
                    else {
                        Integer ConfMess = 
JOptionPane.showConfirmDialog(null, "Are you sure you want to update?", 
"Update", JOptionPane.YES_NO_OPTION);
                        if (ConfMess== 0) {

objmaine.setCommentEMP(textAreaCommentEmp.getText());

objmaine.setTstatus(comboBoxnewStatus.getSelectedItem().toString());
                            AbstractSuperclass.update("UPDATE Tasks set 
CommentEMP = '" + objmaine.getCommentEMP()+ "', Status = '" + 
objmaine.getTstatus() + "'where TaskNum ='" + obj + "'");
                        }
                    }
                }
                catch(Exception e1){e1.printStackTrace();}
            }
        });
        filterButton.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                try{
                    if(checkBoxdate.isSelected()){
                        dateYN = "";
                    }
                    else{
                        dateYN = AbstractSuperclass.Calendar(datechos);
                    }
                    prefix = "select 
TaskNum,CliUsername,Details,CommentCLI,DueDate,Priority,Status,CommentEMP,C
ommentAD from Tasks where EmpUsername='"+USERNAME+"'";
                    filterQuerys.put("Stat", " AND Status 
='"+StatusCbox.getSelectedItem().toString()+"'");
                    filterQuerys.put("Prior", "AND Priority ='"+ 
PriorityCbox.getSelectedItem().toString()+"'");
                    filterQuerys.put("Cli", "AND CliUsername = 
'"+textFieldClient.getText()+"'");
                    filterQuerys.put("DueDate", "AND DueDate 
='"+dateYN+"'");
                    if 
(!StatusCbox.getSelectedItem().toString().equals("")){
                        filters_chosen.add(0, "Stat");
                        if 
(!PriorityCbox.getSelectedItem().toString().equals("")){
```

```
                                filters_chosen.add(1, "Prior");
                                if (!dateYN.equals("")){
                                    filters_chosen.add(2, "DueDate");
                                    if (!textFieldClient.getText().equals("")){
                                        filters_chosen.add(3, "Cli");
                                    }
                                }
                                else if (!textFieldClient.getText().equals("")){
                                    filters_chosen.add(2, "Cli");
                                }
                            }
                            else if (!dateYN.equals("")){
                                filters_chosen.add(1, "DueDate");
                                if (!textFieldClient.getText().equals("")){
                                    filters_chosen.add(2, "Cli");
                                }
                            }
                            else if (!textFieldClient.getText().equals("")){
                                filters_chosen.add(1, "Cli");}
                        }
                        else if
(!PriorityCbox.getSelectedItem().toString().equals("")){
                            filters_chosen.add(0, "Prior");
                            if (!dateYN.equals("")){
                                filters_chosen.add(1, "DueDate");
                                if (!textFieldClient.getText().equals("")){
                                    filters_chosen.add(2, "Cli");
                                    }
                                }

                            else if (!textFieldClient.getText().equals("")) {
                                filters_chosen.add(1, "Cli");
                            }
                        }
                       else if (!dateYN.equals("")){
                            filters_chosen.add(0, "DueDate");
                            if (!textFieldClient.getText().equals("")){
                                filters_chosen.add(1, "Cli");
                            }

                        }
                       else if(!textFieldClient.getText().equals("")){
                            filters_chosen.add(0, "Cli");

                        }
                        String filter_query1 = "";
                        for(int x =0;x<filters_chosen.size();x++){
                            String info =
filterQuerys.get(filters_chosen.get(x));
                            filter_query1 = filter_query1  +" "+ info+ " ";
                        }
                        filter_query =  prefix + " "+ filter_query1 ;
```

```
                    PreparedStatement pst =
connection.prepareStatement(filter_query);
                    ResultSet rs = pst.executeQuery();
                    tableTasks.setModel(DbUtils.resultSetToTableModel(rs));
                    pst.close();
                    rs.close();
                    filters_chosen.clear();


                    //String query1 = "select
TaskNum,CliUsername,Details,CommentCLI,DueDate,Priority,Status,CommentEMP,C
ommentAD from Tasks where Priority ='"+
PriorityCbox.getSelectedItem().toString()+"'AND Status
='"+StatusCbox.getSelectedItem().toString()+"'AND DueDate
='"+AbstractSuperclass.Calendar(datechos)+"'AND EmpUsername='"+USERNAME+"'"
;
                    //PreparedStatement pst =
connection.prepareStatement(query1);
                    //ResultSet rs = pst.executeQuery();

//tableTasks.setModel(DbUtils.resultSetToTableModel(rs));
                    //newobj.show_column();
                    //pst.close();
                    //rs.close();
                }
                catch(Exception e2){
                    e2.printStackTrace();}
            }
        });
        checkBoxdate.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                if(checkBoxdate.isSelected()){
                    calendar.setVisible(false);
                    FDD.setVisible(false);
                    dateYN = "";
                }
                else{
                    calendar.setVisible(true);
                    FDD.setVisible(true);
                }

            }
        });
    }
    @Override
    public void valueChanged(ListSelectionEvent e) {
        if(!e.getValueIsAdjusting()){ //return true if the selection is
still
            int SelectedRow = tableTasks.getSelectedRow(); //getSelectedRow
returns an integer of the row selected
            if(SelectedRow >=0) { //selected row ha to be >=0 because it
means you are selecting something from the table, a row.
```

```
                TableModel model = tableTasks.getModel();
                //getting selected row and then selected column
                obj = model.getValueAt(SelectedRow, 0); //getting the value
of the row selected and the column that we want. Then storing this data in
obj1.
                Object obj1 = model.getValueAt(SelectedRow, 7);
                textAreaCommentEmp.setText(obj1 == null ? "" :
obj1.toString());
                Object obj2 = model.getValueAt(SelectedRow, 6);
                if(obj2.equals("Deleted")){
                    editTaskButton.setEnabled(false);
                }
                else{
                    editTaskButton.setEnabled(true);
                }

            }
        }
    }


    public static void main(String[] args) {
        employeeFrame.setContentPane(new Employee_portal().Employee_portal);
        employeeFrame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        employeeFrame.pack();
        employeeFrame.setVisible(true);
    }

}
```

<u>CLASS: Login</u>

```
package TaskOrganiser;

import javax.swing.*;
import javax.swing.event.ListSelectionEvent;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.sql.Connection;

public class Login extends AbstractSuperclass {
    //CREATE ENCAPSULATED CLASS
    public JPanel Login;
    private JComboBox RolecomboBox1;
    private JPasswordField passwordField1;
    private JTextField UsernametextField;
    private JButton submitButton;
    private JCheckBox showPasswordCheckBox;
    static Integer Rvalue;
    static JFrame framelogin = new JFrame("Login");
    static String role;

    Connection connection = null;
    public Login() {
        connection= SQLite_Connection.dbConnector();
        showPasswordCheckBox.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                if (showPasswordCheckBox.isSelected()){
                    // echoe the character and 0 means that it will show
everything and start echoing characters
                    // from the beginning (from index position 0)
                    passwordField1.setEchoChar((char)0);
                }
                else{
                    //If the checkbox is not selected, the password will
hide again with *
                    passwordField1.setEchoChar('*');
                }

            }
        });

        submitButton.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                try{
                    role = RolecomboBox1.getSelectedItem().toString();
                    if
(UsernametextField.getText().equals("")||(passwordField1.getText().equals("
"))){
                        JOptionPane.showMessageDialog(null,"You have left
field/s empty");
                    }
                    else {
                        if (role.equals("Administrator")) {
                            Rvalue =
AbstractSuperclass.checkExistance("select * from Administrators where
AdminUsername ='"+UsernametextField.getText()+"' AND Password
='"+passwordField1.getText()+"' AND Status ='"+"Existing"+"'");

                            if (Rvalue == 1) {
                                JOptionPane.showMessageDialog(null, "Valid
fields ");

                                framelogin.dispose();
                                JFrame frame = new JFrame("Menu");
                                frame.setContentPane(new Menu().Menu);

frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
                                frame.pack();
                                frame.setVisible(true);
```

```
                              }else if (Rvalue == 0) {
                                  JOptionPane.showMessageDialog(null, "Invalid
fields ");

                              }
                          }
                          else if (role.equals("Employee")) {
                              Rvalue =
AbstractSuperclass.checkExistance("select * from Employees where
EmpUsername ='"+UsernametextField.getText()+"' AND Password
='"+passwordField1.getText()+"'AND Status ='"+"Existing"+"'");
                              if (Rvalue == 1) {
                                  //SHOW CORRESPONDING PORTAL
                                  USERNAME = UsernametextField.getText();
                                  JOptionPane.showMessageDialog(null, "Valid
fields");
                                  framelogin.dispose();
                                  JFrame frame = new
JFrame("Employee_portal");
                                  frame.setContentPane(new
Employee_portal().Employee_portal);

frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
                                  frame.pack();
                                  frame.setVisible(true);
                              }else if (Rvalue == 0) {
                                  JOptionPane.showMessageDialog(null, "Invalid
fields ");

                              }
                          }
                          else if (role.equals("Client")) {
                              Rvalue =
AbstractSuperclass.checkExistance("select * from Clients where CliUsername
='"+UsernametextField.getText()+"' AND Password
='"+passwordField1.getText()+"'AND Status ='"+"Existing"+"'");
                              if (Rvalue == 1) {
                                  JOptionPane.showMessageDialog(null, "Valid
fields");
                                  USERNAME = UsernametextField.getText();
                                  framelogin.dispose();
                                  JFrame frame = new JFrame("Client_portal");
                                  frame.setContentPane(new
Client_portal().Client_portal);

frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
                                  frame.pack();
                                  frame.setVisible(true);


                              }else if (Rvalue == 0) {
                                  JOptionPane.showMessageDialog(null, "Invalid
fields");
```

```
                            }
                        }
                    }

                }
                catch (Exception e1){
                    e1.printStackTrace();
                }
            }
        });
    }

    public static void main(String[] args) {
        framelogin.setContentPane(new Login().Login);
        framelogin.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        framelogin.pack();
        framelogin.setVisible(true);

    }

    @Override
    public void valueChanged(ListSelectionEvent e) {

    }
}
```

CLASS: Menu

```
package TaskOrganiser;

import javax.swing.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

public class Menu {
    public JPanel Menu;
    private JButton clientsAccountsButton;
    private JButton employeesAccountsButton;
    private JButton administratorsAccountsButton;
    private JButton viewTasksButton;
    private JButton exitButton;
    static JFrame menuFrame = new JFrame("Menu");

    public Menu() {
        administratorsAccountsButton.addActionListener(new ActionListener()
{
            @Override
            public void actionPerformed(ActionEvent e) {
                menuFrame.dispose();
                JFrame frame = new JFrame("Admin_accounts");
                frame.setContentPane(new Admin_accounts().Admin_accounts);
                frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
                frame.pack();
                frame.setVisible(true);

            }
        });
        clientsAccountsButton.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                menuFrame.dispose();
                JFrame frame = new JFrame("Client_accounts");
                frame.setContentPane(new Client_accounts().Client_accounts);
                frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
                frame.pack();
                frame.setVisible(true);
            }
        });
        employeesAccountsButton.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                menuFrame.dispose();
                JFrame frame = new JFrame("Employee_accounts");
                frame.setContentPane(new
Employee_accounts().Employee_accounts);
                frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
                frame.pack();
                frame.setVisible(true);
            }
        });
        viewTasksButton.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                menuFrame.dispose();
                JFrame frame = new JFrame("Tasks_management");
                frame.setContentPane(new
Tasks_management().Tasks_management);
                frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
                frame.pack();
                frame.setVisible(true);
            }
        });
        exitButton.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                menuFrame.dispose();
                JFrame frame = new JFrame("Login");
                frame.setContentPane(new Login().Login);
                frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
                frame.pack();
                frame.setVisible(true);
            }
        });
    }
```

```
   public static void main(String[] args) {
       menuFrame.setContentPane(new Menu().Menu);
       menuFrame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
       menuFrame.pack();
       menuFrame.setVisible(true);

   }
}
```

CLASS: Tasks_management

```
package TaskOrganiser;

import com.toedter.calendar.JDateChooser;
import net.proteanit.sql.DbUtils;

import javax.swing.*;
import javax.swing.event.ListSelectionEvent;
import javax.swing.table.TableModel;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.util.ArrayList;
import java.util.Calendar;
import java.util.Hashtable;

public class Tasks_management extends AbstractSuperclass{
   private JComboBox StatusCbox;
   private JComboBox PriorityCbox;
   private JButton filterButton;
   private JTextField textFieldClient;
   private JTable tableTasks;
   private JButton assignButton;
   private JButton approveButton;
   private JButton editButton;
   private JTextField textFieldEusername;
   private JTextArea textAreaCommentAdmin;
   private JComboBox comboBoxnewStatus;
   private JButton saveButton;
   private JButton cancelButton;
   private JLabel Eu;
   private JLabel Comm;
   private JLabel Stat;
   private JButton rejectButton;
   private JButton viewButton;
```

```
    public JPanel Tasks_management;
    private JButton exitButton;
    private JPanel calendar;
    private JCheckBox CheckBoxdate;
    private JLabel FDD;
    Connection connection = null;
    static JFrame tasksFrame = new JFrame("Tasks_management");
    static Object obj;
    static Object obj3;
    static Object obj4;
    static String prefix;
    static String dateYN;
    static String filter_query;
    static Tasks_management newobj = new Tasks_management();
    static MainTasks_management objmaine = new MainTasks_management();
    Calendar calen = Calendar.getInstance();
    JDateChooser datechos = new JDateChooser(calen.getTime());
    Hashtable<String, String> filterQuerys = new Hashtable<>();
    ArrayList<String> filters_chosen = new ArrayList<String>(5);

    public Tasks_management() {
        Eu.setVisible(false);
        Comm.setVisible(false);
        Stat.setVisible(false);
        textFieldEusername.setVisible(false);
        textAreaCommentAdmin.setVisible(false);
        comboBoxnewStatus.setVisible(false);
        saveButton.setVisible(false);
        cancelButton.setVisible(false);
        assignButton.setEnabled(false);
        connection = SQLite_Connection.dbConnector();
        ListSelectionModel selectionModel = tableTasks.getSelectionModel();
        //Add listener to this table in this class
        selectionModel.addListSelectionListener(this);
        datechos.setDateFormatString("dd/MM/yyyy");
        calendar.add(datechos);

        filterButton.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                try{
                    if(CheckBoxdate.isSelected()){
                        dateYN = "";
                    }
                    else{
                        dateYN = AbstractSuperclass.Calendar(datechos);
                    }
                    prefix = "select * from Tasks where TaskNum !=
'"+"-1"+"'";
                    filterQuerys.put("Stat", " AND Status
='"+StatusCbox.getSelectedItem().toString()+"'");
                    filterQuerys.put("Prior", " AND Priority ='"+
PriorityCbox.getSelectedItem().toString()+"'");
```

```
                    filterQuerys.put("Cli", "AND CliUsername =
'"+textFieldClient.getText()+"'");
                    filterQuerys.put("DueDate", " AND DueDate
='"+dateYN+"'");
                    if
(!StatusCbox.getSelectedItem().toString().equals("")){
                        filters_chosen.add(0, "Stat");
                        if
(!PriorityCbox.getSelectedItem().toString().equals("")){
                            filters_chosen.add(1, "Prior");
                            if (!dateYN.equals("")){
                                filters_chosen.add(2, "DueDate");
                                if (!textFieldClient.getText().equals("")){
                                    filters_chosen.add(3, "Cli");
                                }
                            }
                            else if (!textFieldClient.getText().equals("")){
                                filters_chosen.add(2, "Cli");
                            }
                        }
                        else if (!dateYN.equals("")){
                            filters_chosen.add(1, "DueDate");
                            if (!textFieldClient.getText().equals("")){
                                filters_chosen.add(2, "Cli");
                            }
                        }
                        else if (!textFieldClient.getText().equals("")){
                            filters_chosen.add(1, "Cli");}
                    }
                    else if
(!PriorityCbox.getSelectedItem().toString().equals("")){
                        filters_chosen.add(0, "Prior");
                        if (!dateYN.equals("")){
                            filters_chosen.add(1, "DueDate");
                            if (!textFieldClient.getText().equals("")){
                                filters_chosen.add(2, "Cli");
                            }
                        }

                        else if (!textFieldClient.getText().equals("")) {
                            filters_chosen.add(1, "Cli");
                        }
                    }
                    else if (!dateYN.equals("")){
                        filters_chosen.add(0, "DueDate");
                        if (!textFieldClient.getText().equals("")){
                            filters_chosen.add(1, "Cli");
                        }

                    }
                    else if(!textFieldClient.getText().equals("")){
                        filters_chosen.add(0, "Cli");
```

```
                   }
                   String filter_query1 = "";
                   for (int x = 0; x < filters_chosen.size(); x++) {
                           String info =
filterQuerys.get(filters_chosen.get(x));
                           filter_query1 = filter_query1 + " " + info + " "
;
                       }
                   filter_query =  prefix + " "+ filter_query1 ;
                   PreparedStatement pst =
connection.prepareStatement(filter_query);
                   ResultSet rs = pst.executeQuery();
                   tableTasks.setModel(DbUtils.resultSetToTableModel(rs));
                   AbstractSuperclass.showColumn(tableTasks);
                   pst.close();
                   rs.close();
                   filters_chosen.clear();
               }
               catch(Exception e2){
                   e2.printStackTrace();}

           }
       });
       viewButton.addActionListener(new ActionListener() {
           @Override
           public void actionPerformed(ActionEvent e) {
               try{
                   String query1 = "select * from Tasks ";
                   PreparedStatement pst =
connection.prepareStatement(query1);
                   ResultSet rs = pst.executeQuery();
                   tableTasks.setModel(DbUtils.resultSetToTableModel(rs));
                   AbstractSuperclass.showColumn(tableTasks);
                   pst.close();
                   rs.close();
               }
               catch(Exception e2){
                   e2.printStackTrace();}
           }
       });
       approveButton.addActionListener(new ActionListener() {
           @Override
           public void actionPerformed(ActionEvent e) {
               objmaine.setTstatus("Approved");
               AbstractSuperclass.update("UPDATE Tasks set Status =
'"+objmaine.getTstatus()+"' where TaskNum ='"+obj+"'");
           }
       });

       rejectButton.addActionListener(new ActionListener() {
           @Override
           public void actionPerformed(ActionEvent e) {
               objmaine.setTstatus("Rejected");
```

```
                AbstractSuperclass.update("UPDATE Tasks set Status =
'"+objmaine.getTstatus()+"' where TaskNum ='"+obj+"'");

            }
        });
        assignButton.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                Eu.setVisible(true);
                Comm.setVisible(true);
                textFieldEusername.setVisible(true);
                textAreaCommentAdmin.setVisible(true);
                saveButton.setVisible(true);
                cancelButton.setVisible(true);

                //approveButton.setEnabled(false);
                //rejectButton.setEnabled(false);
                //editButton.setEnabled(false);
                // assignButton.setEnabled(false);
                //  viewButton.setEnabled(false);
            }
        });
        editButton.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                Eu.setVisible(true);
                Comm.setVisible(true);
                Stat.setVisible(true);
                textFieldEusername.setVisible(true);
                textAreaCommentAdmin.setVisible(true);
                comboBoxnewStatus.setVisible(true);
                saveButton.setVisible(true);
                cancelButton.setVisible(true);

                // approveButton.setEnabled(false);
                //rejectButton.setEnabled(false);
                //editButton.setEnabled(false);
                //assignButton.setEnabled(false);
                //viewButton.setEnabled(false);


            }
        });
        saveButton.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                try{
                    //validation

if((textAreaCommentAdmin.getText().equals(""))||(textFieldEusername.getText
().equals(""))){
                        JOptionPane.showMessageDialog(null, "There are
fields empty");
```

```
                    }
                    Integer valuer =
AbstractSuperclass.checkExistance("select * from Employees where
EmpUsername ='"+textFieldEusername.getText()+"' AND Status
='"+"Existing"+"'");
                    if (valuer ==0 ){
                        JOptionPane.showMessageDialog(null, "The employee
username does not exist");
                    }

                    if ((valuer ==
0)||(textAreaCommentAdmin.getText().equals(""))||
(textFieldEusername.getText().equals(""))){
                        JOptionPane.showMessageDialog(null, "Invalid");
                    }

                    else {
                        Integer ConfMess =
JOptionPane.showConfirmDialog(null, "Are you sure you want to save?",
"Save", JOptionPane.YES_NO_OPTION);
                        if (ConfMess== 0) {

objmaine.setTstatus(comboBoxnewStatus.getSelectedItem().toString());

objmaine.setCommentAD(textAreaCommentAdmin.getText());

objmaine.setEusername(textFieldEusername.getText());
                            AbstractSuperclass.update("UPDATE Tasks set
Status = '" + objmaine.getTstatus() + "', EmpUsername = '" +
objmaine.getEusername() + "', CommentAD= '" + objmaine.getCommentAD() +
"'where TaskNum ='" + obj + "'");
                            assignButton.setEnabled(false);
                        }
                    }
                }
                catch(Exception e1){e1.printStackTrace();}
            }
        });
        cancelButton.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                saveButton.setVisible(false);
                Eu.setVisible(false);
                Comm.setVisible(false);
                Stat.setVisible(false);
                textFieldEusername.setVisible(false);
                textAreaCommentAdmin.setVisible(false);
                comboBoxnewStatus.setVisible(false);
                cancelButton.setVisible(false);


            }
        });
        exitButton.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {

                JFrame frame = new JFrame("Menu");
                frame.setContentPane(new Menu().Menu);
                frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
                frame.pack();
                frame.setVisible(true);
                tasksFrame.dispose();

            }
        });
        CheckBoxdate.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                if(CheckBoxdate.isSelected()){
                    calendar.setVisible(false);
                    FDD.setVisible(false);
                    dateYN = "";
                }
                else{
                    calendar.setVisible(true);
                    FDD.setVisible(true);
                }
            }
        });
    }

    @Override
    public void valueChanged(ListSelectionEvent e) {
        if(!e.getValueIsAdjusting()){ //return true if the selection is
still
            int SelectedRow = tableTasks.getSelectedRow(); //getSelectedRow
returns an integer of the row selected
            if(SelectedRow >=0){ //selected row ha to be >=0 because it
means you are selecting something from the table, a row.
                TableModel model = tableTasks.getModel();
                //getting selected row and then selected column
                obj = model.getValueAt(SelectedRow, 0); //getting the value
of the row selected and the column that we want. Then storing this data in
obj1.
                Object obj1 = model.getValueAt(SelectedRow, 2);
                textFieldEusername.setText(obj1 == null ?
"":obj1.toString());
                Object obj2 = model.getValueAt(SelectedRow, 9);
                textAreaCommentAdmin.setText(obj2 == null ?
"":obj2.toString());
                obj3 = model.getValueAt(SelectedRow, 7);
                obj4 = model.getValueAt(SelectedRow, 2);
                if ((!obj3.equals("Pending"))){
                    approveButton.setEnabled(false);
                    if (obj3.equals("Approved")){
                        rejectButton.setEnabled(true);
                    }
                    else{rejectButton.setEnabled(false);}

if((obj4==null)&(!obj3.equals("Deleted"))&(!obj3.equals("Pending"))&(!obj3.
equals("Rejected"))){
                        assignButton.setEnabled(true);
                    }
                    else{assignButton.setEnabled(false);}
                }
                else{
                    approveButton.setEnabled(true);
                    rejectButton.setEnabled(true);
                }



            }
        }
    }

    public static void main(String[] args) {
        //tasksFrame is an object of the frame
        tasksFrame.setContentPane(new
Tasks_management().Tasks_management);//name of class and then name of panel
        tasksFrame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        tasksFrame.pack();
        tasksFrame.setVisible(true);
    }

}
```