

```
1 package il.ac.hit.CashFlowManagement;
2
3 import il.ac.hit.CashFlowManagement.viewmodel.IViewModel;
4 import il.ac.hit.CashFlowManagement.viewmodel.ManagementViewModel;
5
6 /**
7  * The program class
8  * @author Moshiko Davila 308459841
9  * @author Sagi Sela 201325560
10 * @version March 24 , 2020.
11 */
12 public class Program {
13     public static void main(String[] args)
14     {
15         IViewModel viewModel = ManagementViewModel.getInstance();
16
17         viewModel.startProgram();
18     }
19 }
```

```
1 package il.ac.hit.CashFlowManagement.view;
2
3 import org.jetbrains.annotations.NotNull;
4
5 import java.awt.*;
6
7 /**
8  * this interface define the operations that each Form/Frame must
9  * implement
10 */
10 public interface IView {
11
12     /**
13      * Show dialog of the form
14      */
15     void showForm();
16
17     /**
18      * Close the dialog of the form
19      * @param form the form who should be closed, can't be null
20      */
21     default void closeForm(@NotNull Window form){
22         form.setVisible(false);
23         form.dispose();
24     }
25
26     /**
27      * set the form view model to the ManagementViewModel instance
28      */
29     void setViewModel();
30 }
31
```

```

1 package il.ac.hit.CashFlowManagement.view;
2
3 import il.ac.hit.CashFlowManagement.exception.FormCastingException;
4 import il.ac.hit.CashFlowManagement.viewmodel.IViewModel;
5 import il.ac.hit.CashFlowManagement.viewmodel.ManagementViewModel;
6 import il.ac.hit.CashFlowManagement.viewmodel.TableData;
7 import org.apache.log4j.Logger;
8
9 import javax.swing.*;
10 import java.awt.*;
11 import java.awt.event.ActionEvent;
12 import java.awt.event.ActionListener;
13
14 /**
15  * The main form class
16 */
17 public class MainForm extends JFrame implements ActionListener, IView
18 {
19     private static Logger logger = Logger.getLogger(MainForm.class);
20     private IViewModel viewModel;
21
22     //creating dates scrollable strings
23     private String[] months = {"", "Jan", "Feb", "Mar", "Apr", "May",
24         , "Jun", "Jul", "Aug", "Sept", "Oct", "Nov", "Dec"};
25     private String[] days = {"", "1", "2", "3", "4", "5",
26         "6", "7", "8", "9", "10",
27         "11", "12", "13", "14", "15",
28         "16", "17", "18", "19", "20",
29         "21", "22", "23", "24", "25",
30         "26", "27", "28", "29", "30",
31         "31"};
32     private String years[]
33         = {"", "1995", "1996", "1997", "1998",
34             "1999", "2000", "2001", "2002",
35             "2003", "2004", "2005", "2006",
36             "2007", "2008", "2009", "2010",
37             "2011", "2012", "2013", "2014",
38             "2015", "2016", "2017", "2018",
39             "2019", "2020"};
40
41     //creating gui components
42     private Container container = getContentPane();
43     private JComboBox monthsComboBox = new JComboBox(months);

```

```

43     private JComboBox daysComboBox = new JComboBox(days);
44     private JComboBox yearsComboBox = new JComboBox(years);
45     private JComboBox classificationComboBox = new JComboBox(
46         classification);
46     private DesignedJLabel titleLabel = new DesignedJLabel("Main Form");
47     private DesignedJLabel newExpenseLabel = new DesignedJLabel("New
48     Expense:".toUpperCase());
48     private DesignedJLabel dateLabel = new DesignedJLabel("DATE");
49     private DesignedJLabel classificationLabel = new DesignedJLabel(
49         "CLASSIFICATION");
50     private DesignedJLabel detailLabel = new DesignedJLabel("DETAIL");
50 );
51     private DesignedJLabel costLabel = new DesignedJLabel("COST");
52     private JButton addBtn = new DesignedJButton("Add");
53     private JTextField detailTextField = new JTextField();
54     private JTextField costTextField = new JTextField();
55     private JButton refreshTable = new DesignedJButton("Refresh Table");
56     JScrollPane tableScrollPane = new JScrollPane();
57     private TableData tableDataForm;
58
59 /**
60 * MainForm C'tor
61 */
62 public MainForm() {
63     setLayoutManager();
64     setLocationAndSize();
65     addComponentsToContainer();
66     addActionEvent();
67
68     logger.info("MainForm was create successfully!!!");
69 }
70
71 public JComboBox getMonthsComboBox() {
72     return monthsComboBox;
73 }
74
75 public JComboBox getDaysComboBox() {
76     return daysComboBox;
77 }
78
79 public JComboBox getYearsComboBox() {
80     return yearsComboBox;
81 }
82

```

```

83     public JComboBox getClassificationComboBox() {
84         return classificationComboBox;
85     }
86
87     public JTextField getDetailTextField() {
88         return detailTextField;
89     }
90
91     public JTextField getCostTextField() {
92         return costTextField;
93     }
94
95     /**
96      * @see IView {@Link #showForm()}
97      */
98     public void showForm()
99     {
100         tableDataForm = new TableData();
101         setTitle("Main Form");
102         setVisible(true);
103         setBounds(300, 90, 1100, 800);
104         setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
105         setResizable(false);
106         setTitle("Logged in as " + LoginForm.username);
107         refreshTable();
108         logger.info(MainForm.class+" was shown..");
109
110         //URL image
111         JLabel ImageLogo;
112         ImageLogo = new JLabel(new ImageIcon("Cash_Management.png"
113         ));
113         ImageLogo.setBounds(0, 0, 1100, 800);
114         container.add(ImageLogo);
115         logger.info("Create Image was Successes!!!");
116
117     }
118
119     /**
120      * @see IView {@Link #setViewModel()}
121      */
122     @Override
123     public void setViewModel(){
124         viewModel = ManagementViewModel.getInstance();
125     }
126
127     public String[] getMonths(){

```

```
128         return months;
129     }
130
131     public void setLayoutManager() {
132         container.setLayout(null);
133     }
134
135     public void setLocationAndSize()
136     {
137         //set positioning, fonts and colors of gui components
138         titleLabel.setBounds(500, 30, 200, 40);
139         titleLabel.setFont(new Font("Verdana", Font.BOLD, 15));
140         newExpenseLabel.setBounds(50, 120, 200, 40);
141         dateLabel.setBounds(50, 150, 200, 40);
142         daysComboBox.setBounds(100, 160, 60, 20);
143         monthsComboBox.setBounds(170, 160, 60, 20);
144         yearsComboBox.setBounds(240, 160, 60, 20);
145         classificationLabel.setBounds(310, 150, 200, 40);
146         classificationComboBox.setBounds(410, 160, 100, 20);
147         detailLabel.setBounds(520, 150, 200, 40);
148         detailTextField.setBounds(570, 160, 200, 20);
149         costLabel.setBounds(780, 150, 200, 40);
150         costTextField.setBounds(830, 160, 100, 20);
151         addBtn.setBounds(950, 160, 60, 20);
152         refreshTable.setBounds(450, 250, 200, 20);
153     }
154
155     public void addComponentsToContainer()
156     {
157         //fill form container with gui components
158         container.add(titleLabel);
159         container.add(newExpenseLabel);
160         container.add(dateLabel);
161         container.add(classificationLabel);
162         container.add(detailLabel);
163         container.add(costLabel);
164         container.add(monthsComboBox);
165         container.add(daysComboBox);
166         container.add(yearsComboBox);
167         container.add(classificationComboBox);
168         container.add(detailTextField);
169         container.add(costTextField);
170         container.add(addBtn);
171         container.add(refreshTable);
172     }
173 }
```

```
174     public void addActionEvent() {
175         //adding events listeners
176         addBtn.addActionListener(this);
177         refreshTable.addActionListener(this);
178     }
179
180
181     @Override
182     public void actionPerformed(ActionEvent e) {
183         //handling events listeners
184         if (e.getSource() == addBtn) {
185             SwingUtilities.invokeLater(new Runnable() {
186                 @Override
187                 public void run() {
188                     try {
189                         viewModel.addNewExpense();
190                         refreshTable();
191                     } catch (FormCastingException ex) {
192                         ex.printStackTrace();
193                     }
194                 }
195             });
196         }
197         else if (e.getSource() == refreshTable) {
198             refreshTable();
199         }
200     }
201
202     private void refreshTable(){
203         container.remove(tableJScrollPane);
204         tableJScrollPane = tableDataForm.getUpdatedTableJScrollPane
205         ();
206         tableJScrollPane.setBounds(50, 320, 900, 350);
207         container.add(tableJScrollPane, SwingConstants.CENTER);
208     }
209
210
211
```

```

1 package il.ac.hit.CashFlowManagement.view;
2
3 import il.ac.hit.CashFlowManagement.viewmodel.IViewModel;
4 import il.ac.hit.CashFlowManagement.viewmodel.ManagementViewModel;
5 import org.apache.log4j.Logger;
6
7 import javax.swing.*;
8 import java.awt.*;
9 import java.awt.event.ActionEvent;
10 import java.awt.event.ActionListener;
11
12 /**
13 * The Login form class
14 */
15 public class LoginForm extends JFrame implements ActionListener,
16 IView {
16     private static Logger logger = Logger.getLogger(LoginForm.class);
17     public static String username;
18     private IViewModel viewModel;
19     //creating gui components
20     private Container container = getContentPane();
21     private DesignedJLabel titleLabel = new DesignedJLabel("Login");
22     private DesignedJLabel userLabel = new DesignedJLabel("USERNAME")
23 );
23     private DesignedJLabel passwordLabel = new DesignedJLabel("PASSWORD");
24     private JTextField userTextField = new JTextField();
25     private JPasswordField passwordField = new JPasswordField();
26     private DesignedJButton loginButton = new DesignedJButton("LOGIN")
27 );
27     private DesignedJButton resetButton = new DesignedJButton("RESET")
28 );
28     private DesignedJButton registerButton = new DesignedJButton("REGISTER");
29     private JCheckBox showPassword = new JCheckBox("Show Password");
30     private JLabel lblBackground;
31
32 /**
33 * LoginForm C'tor
34 */
35 public LoginForm()
36 {
37     //Calling setLayoutManger() method inside the constructor.
38     setLayoutManager();
39     setLocationAndSize();
40     addComponentsToContainer();

```

```

41         addActionEvent();
42         logger.info("Login form was create successfully!!");
43     }
44
45     /**
46      * @see IView {@Link #showForm()}
47      */
48     public void showForm()
49     {
50         setTitle("Login Form");
51         setVisible(true);
52         setBounds(10, 10, 400, 600);
53         setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
54         setResizable(false);
55         setLocationRelativeTo(null);
56         logger.info(LoginForm.class+" was shown..");
57
58         //URL image
59         JLabel ImageLogo;
60         ImageLogo = new JLabel(new ImageIcon("cash15.png"));
61         ImageLogo.setBounds(10, 10, 400, 600);
62         container.add(ImageLogo);
63         logger.info("Create Image was Successes!!");
64     }
65
66     /**
67      * @see IView {@Link #setViewModel()}
68      */
69     @Override
70     public void setViewModel(){
71         viewModel = ManagementViewModel.getInstance();
72     }
73
74
75     public void setLayoutManager() {
76         //Setting Layout manager of Container to null
77         container.setLayout(null);
78     }
79
80     public void setLocationAndSize() {
81         //set positioning, fonts and colors of gui components
82         titleLabel.setBounds(170, 30, 200, 40);
83         titleLabel.setFont(new Font(" SERIF", Font.BOLD, 20));
84         userLabel.setBounds(50, 150, 100, 30);
85         passwordLabel.setBounds(50, 220, 100, 30);
86         userTextField.setBounds(150, 150, 150, 30);

```

```

87         passwordField.setBounds(150, 220, 150, 30);
88         showPassword.setBounds(150, 250, 150, 30);
89         showPassword.setBackground(Color.white);
90         loginButton.setBounds(50, 300, 100, 30);
91         resetButton.setBounds(200, 300, 100, 30);
92         registerButton.setBounds(130, 350, 100, 30);
93     }
94
95     public void addComponentsToContainer() {
96         //fill form container with gui components
97         container.add(titleLabel);
98         container.add(userLabel);
99         container.add(passwordLabel);
100        container.add(userTextField);
101        container.add(passwordField);
102        container.add(showPassword);
103        container.add(loginButton);
104        container.add(resetButton);
105        container.add(registerButton);
106    }
107
108    public void addActionEvent() {
109        //adding events listeners
110        loginButton.addActionListener(this);
111        resetButton.addActionListener(this);
112        registerButton.addActionListener(this);
113        showPassword.addActionListener(this);
114    }
115
116
117    @Override
118    public void actionPerformed(ActionEvent e) {
119        //handling events listeners
120        if (e.getSource() == loginButton)
121        {
122            String userText;
123            String pwdText;
124            userText = userTextField.getText();
125            pwdText = passwordField.getText();
126            if (viewModel.verifyUser(userText, pwdText)) {
127                username = userText;
128                viewModel.showMainForm();
129                closeForm(this);
130            } else {
131                JOptionPane.showMessageDialog(this, "Invalid
Username or Password");

```

```
132         }
133     }
134     if (e.getSource() == registerButton)
135     {
136         viewModel.showRegisterForm();
137         closeForm(this);
138     }
139     if (e.getSource() == resetButton) {
140         userTextField.setText("");
141         passwordField.setText("");
142     }
143     if (e.getSource() == showPassword) {
144         if (showPassword.isSelected()) {
145             passwordField.setEchoChar((char) 0);
146         } else {
147             passwordField.setEchoChar('*');
148         }
149     }
150 }
151 }
152 }
```

```

1 package il.ac.hit.CashFlowManagement.view;
2
3 import il.ac.hit.CashFlowManagement.viewmodel.IViewModel;
4 import il.ac.hit.CashFlowManagement.viewmodel.ManagementViewModel;
5 import org.apache.log4j.Logger;
6
7 import javax.swing.*;
8 import java.awt.*;
9 import java.awt.event.ActionEvent;
10 import java.awt.event.ActionListener;
11
12
13
14 /**
15  * The register form class
16 */
17
18 public class RegisterForm extends JFrame implements ActionListener,
19 IView
20 {
21     private static Logger logger = Logger.getLogger(RegisterForm.
22         class);
23     private IViewModel viewModel;
24     //creating gui components
25     private Container container = getContentPane();
26     private String[] country={"","","INDIA","AMERICA","AUSTRALIA",
27     "PHILIPPINES","SPAIN","ISRAEL"};
28     private DesignedJLabel titleLabel = new DesignedJLabel("Registration Form");
29     private DesignedJLabel userLabel=new DesignedJLabel("USERNAME");
30     private DesignedJLabel passwordLabel=new DesignedJLabel("PASSWORD");
31     private DesignedJLabel confirmPasswordLabel=new DesignedJLabel("CONFIRM PASSWORD");
32     private DesignedJLabel countryLabel=new DesignedJLabel("SELECT COUNTRY");
33     private DesignedJLabel genderLabel=new DesignedJLabel("GENDER");
34     private JCheckBox maleGender=new JCheckBox("MALE");
35     private JCheckBox femaleGender=new JCheckBox("FEMALE");
36     private ButtonGroup bg=new ButtonGroup();
37     private JComboBox countryComboBox=new JComboBox(country);
38     private DesignedJButton submitButton=new DesignedJButton("SUBMIT");
39 );
40     private DesignedJButton resetButton=new DesignedJButton("RESET");
41     private JTextField userTextField = new JTextField();
42     private JPasswordField passwordField=new JPasswordField();

```

```

39     private JPasswordField confirmPasswordField=new JPasswordField();
40
41     /**
42      * RegisterForm C'tor
43      */
44     public RegisterForm() {
45         //Calling setLayoutManger() method inside the constructor.
46         setLayoutManager();
47         setLocationAndSize();
48         addComponentsToContainer();
49         addActionEvent();
50         logger.info("Register form was create successfully");
51     }
52
53     /**
54      * @see IView {@Link #showForm()}
55      */
56     public void showForm()
57     {
58         setTitle("Register Form");
59         setVisible(true);
60         getContentPane().setLayout(null);
61         setBounds(40, 40, 500, 600);
62         // getContentPane().setBackground(Color.LIGHT_GRAY);
63         setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
64         setResizable(false);
65         logger.info(RegisterForm.class+" was shown..");
66
67         //URL image
68         JLabel ImageLogo;
69         ImageLogo = new JLabel(new ImageIcon("cash.jpg"));
70         ImageLogo.setBounds(0, 0, 500, 600);
71         container.add(ImageLogo);
72         logger.info("Create Image was Successes!!!");
73     }
74
75     /**
76      * @see IView {@Link #setViewModel()}
77      */
78     @Override
79     public void setViewModel(){
80         viewModel = ManagementViewModel.getInstance();
81     }
82
83     public void setLayoutManager() {
84         //Setting Layout manager of Container to null

```

```
85         container.setLayout(null);
86     }
87
88     public void setLocationAndSize() {
89         //set positioning, fonts and colors of gui components
90         titleLabel.setBounds(170, 30, 200, 40);
91         titleLabel.setFont(new Font(" SERIF", Font.BOLD, 20));
92         userLabel.setBounds(100, 150, 200, 40);
93         passwordLabel.setBounds(100, 200, 200, 40);
94         confirmPasswordLabel.setBounds(100, 250, 200, 40);
95         countryLabel.setBounds(100, 300, 200, 40);
96         genderLabel.setBounds(100, 400, 200, 40);
97         userTextField.setBounds(250,150,150,30);
98         passwordField.setBounds(250,200,150,30);
99         confirmPasswordField.setBounds(250,250,150,30);
100        maleGender.setBounds(250,400,100,40);
101        maleGender.setBackground(Color.LIGHT_GRAY);
102        femaleGender.setBounds(350,400,100,40);
103        countryComboBox.setBounds(250,300,120,40);
104        countryComboBox.setBackground(Color.WHITE);
105        submitButton.setBounds(100,500,120,40);
106        resetButton.setBounds(250,500,120,40);
107    }
108
109
110    public void addComponentsToContainer() {
111        //fill form container with gui components
112        container.add(titleLabel);
113        container.add(userLabel);
114        container.add(passwordLabel);
115        container.add(confirmPasswordLabel);
116        container.add(countryLabel);
117        container.add(genderLabel);
118        container.add(userTextField);
119        container.add(passwordField);
120        container.add(confirmPasswordField);
121        container.add(maleGender);
122        container.add(femaleGender);
123        bg.add(maleGender);
124        bg.add(femaleGender);
125        container.add(countryComboBox);
126        container.add(submitButton);
127        container.add(resetButton);
128    }
129
130    public void addActionEvent() {
```

```

131         //adding events listeners
132         submitButton.addActionListener(this);
133         resetButton.addActionListener(this);
134     }
135
136     @Override
137     public void actionPerformed(ActionEvent e) {
138         //handling events listeners
139         if (e.getSource() == submitButton) {
140             String userName, pwd, cPwd, country, gender;
141             boolean correctInformation;
142
143             userName = userTextField.getText();
144             pwd = passwordField.getText();
145             cPwd = confirmPasswordField.getText();
146             country = countryComboBox.getSelectedItem().toString();
147             gender = maleGender.isSelected() ? "Male": "Female";
148             correctInformation = pwd.length() >= 4 && pwd.
149             equalsIgnoreCase(cPwd) && !userName.equalsIgnoreCase("") && userName
150             .chars().allMatch(Character::isLetter) && (maleGender.isSelected
151             () || femaleGender.isSelected()) && country != "";
152             if (correctInformation) {
153                 if(ManagementViewModel.getInstance().register(
154                     userName, pwd, country, gender)){
155                     if (e.getSource() == submitButton) {
156                         JOptionPane.showMessageDialog(null, "Data
157                         Registered Successfully");
158                         viewModel.showLoginForm();
159                         closeForm(this);
160                     }
161                     else if(userName.equalsIgnoreCase("") || !userName.chars
162                     ().allMatch(Character::isLetter)){
163                         if(userName.equalsIgnoreCase("")){
164                             JOptionPane.showMessageDialog(null, "Username
165                             can't be empty");
166                         else
167                             JOptionPane.showMessageDialog(null, "UserName
168                             must contain only letters");
169                     }
170                 }
171             }
172         }
173     }

```

```
168         }
169         else if(!maleGender.isSelected() || femaleGender.
170     isSelected()))){
170             JOptionPane.showMessageDialog(null, "Please select
171             gender");
171         }
172         else if(country != ""){
173             JOptionPane.showMessageDialog(null, "Please select
174             country");
174         }
175         else {
176             if(pwd.length() < 4) {
177                 JOptionPane.showMessageDialog(null, "Password
178             must include minimum 4 characters");
178         }
179         else {
180             JOptionPane.showMessageDialog(null, "Password
181             did not match");
181         }
182     }
183 }
184 else if (e.getSource() == resetButton) {
185     userTextField.setText("");
186     countryComboBox.setSelectedItem("");
187     passwordField.setText("");
188     confirmPasswordField.setText("");
189 }
190 }
191 }
192
193
194
195
```

```
1 package il.ac.hit.CashFlowManagement.view;
2
3 import javax.swing.*;
4 /**
5  * The Designed JLabel class
6 */
7 public class DesignedJLable extends JLabel {
8     public DesignedJLable (String text){
9         this.setText(text);
10        this.setBounds(50, 50,100, 30);
11 //        setOpaque(true);
12 //        this.setBackground(Color.black);
13 //        this.setFont(new Font("Verdana", Font.BOLD, 12));
14 //        this.setForeground(Color.black);
15    }
16 }
17
```

```
1 package il.ac.hit.CashFlowManagement.view;
2
3 import javax.swing.*;
4 /**
5  * The Designed JButton class
6 */
7
8 public class DesignedJButton extends JButton {
9     public DesignedJButton (String text){
10         this.setText(text);
11         this.setSize(100, 50);
12 //         this.setBackground(Color.cyan);
13 //         this.setFont(new Font("Verdana", Font.BOLD, 12));
14 //         this.setForeground(Color.black);
15     }
16 }
17
```

```
1 package il.ac.hit.CashFlowManagement.model;
2
3 import org.jetbrains.annotations.NotNull;
4
5 import java.text.ParseException;
6 import java.text.SimpleDateFormat;
7 import java.util.Date;
8
9 /**
10  * The Expense class
11 */
12 public class Expense
13 {
14     private Date date;
15     private String classification, details;
16     private double cost;
17     private SimpleDateFormat simpleDateFormat = new SimpleDateFormat(
18         "dd/MM/yyyy");
19
20     /**
21      * Expense C'tor
22      * @param iDateStr Date of new expense, can't be null
23      * @param iClassification Classification of new expense, can't be
24      * null
25      * @param iDetails New expense details, can't be null
26      * @param iCost New expense cost, can't be null
27      */
28     public Expense(@NotNull String iDateStr, @NotNull String
29         iClassification, @NotNull String iDetails, @NotNull double iCost) {
30         try{
31             setDate(simpleDateFormat.parse(iDateStr));
32             setClassification(iClassification);
33             setDetails(iDetails);
34             setCost(iCost);
35         } catch(ParseException pe) {
36             throw new IllegalArgumentException(pe);
37         }
38     }
39
40     public String getClassification() {
41         return classification;
42     }
43     public void setClassification(String classification) {
44         this.classification = classification;
45     }
```

```
44
45     public String getDetails() {
46         return details;
47     }
48
49     public void setDetails(String details) {
50         this.details = details;
51     }
52
53     public double getCost() {
54         return cost;
55     }
56
57     public void setCost(double cost) {
58         this.cost = cost;
59     }
60
61     public Date getDate() {
62         return date;
63     }
64
65     public void setDate(Date date) {
66         this.date = date;
67     }
68
69     public void setDate(String i_Date) {
70         try {
71             setDate(simpleDateFormat.parse(i_Date));
72         } catch (ParseException e) {
73             e.printStackTrace();
74         }
75     }
76 }
77
```

```
1 package il.ac.hit.CashFlowManagement.model;
2
3 import il.ac.hit.CashFlowManagement.exception.InsertToDBException;
4 import il.ac.hit.CashFlowManagement.exception.
5 JDBC DataBaseLogicException;
6 import org.apache.log4j.Logger;
7 import org.jetbrains.annotations.NotNull;
8
9
10
11 /**
12  * The User Logic class
13 */
14 public class UserLogic implements IUserLogic
15 {
16     private static Logger logger;
17     private JDBC DataBaseLogic DataBaseLogic;
18     private final String nameOfTable = "Registered_Users";
19
20     /**
21      * UserLogic C'tor
22      */
23     public UserLogic()
24     {
25         setLogger(Logger.getLogger(UserLogic.class));
26         setDataBaseLogic(JDBC DataBaseLogic.getInstance());
27         try {
28             DataBaseLogic.createTableIfNotExist(
29                 nameOfTable,
30                 "UserName",
31                 "varchar(300)",
32                 "Password",
33                 "varchar(300)",
34                 "Country",
35                 "varchar(300)",
36                 "Gander",
37                 "varchar(300)"
38             );
39             logger.info("UserLogic was create successfully!!!");
40         } catch (JDBC DataBaseLogicException e) {
41             logger.info("Failed to create UserLogic");
42             e.printStackTrace();
43         }
44     }
45 }
```

```

46     /**
47      * see IUserLogic {@Link #addUser(String iUserName, String
48      * iPassword, String iCountry, String iGander)}
49
50      public void addUser(@NotNull String iUserName, @NotNull String
51      * iPassword, @NotNull String iCountry, @NotNull String iGander) {
52          String parameters =
53          " (" +
54          """ + iUserName + """
55          + "," + """ + iPassword + """
56          + "," + """ + iCountry + """
57          + "," + """ + iGander + """
58          + ")"
59
60          try {
61              DataBaseLogic.insertValue(
62                  nameOfTable,
63                  parameters);
64              logger.info("successfully registered new user in DB");
65          } catch (InsertToDBException e) {
66              logger.info("failed to register new user in DB");
67              e.printStackTrace();
68          }
69
70      /**
71      * see IUserLogic {@Link #checkIfExist(String)}
72      */
73      public boolean checkIfExist(@NotNull String iUserName) {
74          boolean valToReturn = false;
75
76          try {
77              ResultSet rs = DataBaseLogic.query("select * from " +
78          nameOfTable + " where " + nameOfTable + ".USERNAME ='" + iUserName +
79          "'");
80
81              if (rs.next()) {
82                  valToReturn = true;
83                  logger.info("user exists in DB");
84              }
85          } else{
86              logger.info("user doesn't exists in DB");
87          }
88      } catch (JDBCDataBaseLogicException e) {
89          logger.info("checkIfExist method failed");

```

```

88             e.printStackTrace();
89     } finally {
90         return valToReturn;
91     }
92 }
93
94 /**
95  * @see IUserLogic {@Link #checkUserPassword(String iUserName,
96  String iPassword)}
97 */
98 public boolean checkUserPassword(@NotNull String iUserName, @
99 NotNull String iPassword) {
100     boolean valToReturn = false;
101
102     try {
103         ResultSet rs = DataBaseLogic.query("select * from "+
104 nameOfTable + " where "+ nameOfTable +".USERNAME ='" +iUserName+ "' and
105         "+
106             nameOfTable +".PASSWORD ='" +iPassword+ "'");
107
108         if(rs.next())
109         {
110             valToReturn = true;
111             logger.info("login information correct");
112         }
113         else{
114             logger.info("login information is not correct or the
115 user don't exist");
116         }
117     } catch (JDBCDataBaseLogicException e)
118     {
119         logger.info("failed to validate user login information
120 with UserLogic.checkUserPassword method");
121         e.printStackTrace();
122     }
123     finally {
124         return valToReturn;
125     }
126
127     public static void setLogger(Logger logger) {
128         UserLogic.logger = logger;
129     }
130
131     public void set DataBaseLogic(JDBCDataBaseLogic DataBaseLogic) {
132         this.dataBaseLogic = DataBaseLogic;

```

```
128      }  
129 }
```

```
1 package il.ac.hit.CashFlowManagement.model;
2
3 import org.jetbrains.annotations.NotNull;
4
5 /**
6  * this interface define the operations that the UserLogic must
7  * implement
8 */
9 public interface IUserLogic {
10    /**
11     * check if user exist in the Registered_Users table
12     * @param iUserName the new user username, can't be null
13     * @return true either false if the username exists in the DB
14     */
15    boolean checkIfExist(@NotNull String iUserName);
16
17    /**
18     * validate user login information (username and password match
19     * ) in the Registered_Users table
20     * @param iUserName the username of the user who want to login,
21     * can't be null
22     * @param iPassword the password of the user who want to login,
23     * can't be null
24     * @return true either false if the login information is correct
25     */
26    boolean checkUserPassword(@NotNull String iUserName, @NotNull
27      String iPassword);
28
29    /**
30     * add user to the Registered_Users table in the DB
31     * @param iUserName the new user username, can't be null
32     * @param iPassword the new user password, can't be null
33     * @param iCountry the new user country, can't be null
34     * @param iGander the new user gander, can't be null
35     */
36    void addUser(@NotNull String iUserName, @NotNull String iPassword
37      , @NotNull String iCountry, @NotNull String iGander);
38 }
39 }
```

```

1 package il.ac.hit.CashFlowManagement.model;
2
3 import il.ac.hit.CashFlowManagement.exception.JDBCDataBaseException;
4 import org.apache.log4j.Logger;
5
6 import java.sql.Connection;
7 import java.sql.DriverManager;
8 import java.sql.SQLException;
9 import java.sql.Statement;
10
11
12
13 /**
14  * The DataBase class
15 */
16 public class JDBCDataBase
17 {
18     private static JDBCDataBase instance = null;
19     private static Object lock = new Object();
20     private static Logger logger;
21     private final String driver = "org.apache.derby.jdbc.
EmbeddedDriver";
22     private final String protocol = "jdbc:derby:CashFlowManagementDB;
create=true";
23     private boolean isConnected;
24     private Connection connection = null;
25     private Statement statement = null;
26
27     private JDBCDataBase() throws JDBCDataBaseException {
28         try {
29             setLogger(Logger.getLogger(JDBCDataBase.class));
30             setConnected(false);
31             init();
32             logger.info("DataBase was create successfully!!!");
33         } catch (JDBCDataBaseException e){
34             logger.info("DataBase creation failed!!!");
35             throw new JDBCDataBaseException("DataBase creation
problem " + e.getMessage(), e);
36         }
37     }
38
39 /**
40  * This method Describes Connection to DataBase
41  * @return Connection to DataBase , m_IsConnected
42  */
43     private boolean init() throws JDBCDataBaseException {

```

```
44     try {
45         Class.forName(driver);
46         setConnection(DriverManager.getConnection(protocol));
47         setStatement(connection.createStatement());
48         setConnected(true);
49     } catch (SQLException e){
50         throw new JDBC DataBaseException("Connection problem" , e
51 );
52     } catch (ClassNotFoundException ex){
53         throw new JDBC DataBaseException("Driver Class not found"
54 , ex);
55     }
56 }
57 }
58 /**
59 * check if there is a singleton instance of JDBC DataBase and
60 create it's if needed
61 * @return instance of DataBase
62 * @throws JDBC DataBaseException if failed to get create DB
63 instance
64 */
65 public static JDBC DataBase getInstance() throws
66 JDBC DataBaseException
67 {
68     if(instance == null)
69     {
70         synchronized (lock)
71         {
72             if (instance == null)
73             {
74                 instance = new JDBC DataBase();
75             }
76         }
77     }
78     return instance;
79 }
80 public Statement getStatement()
81 {
82     return statement;
83 }
84
```

```
85     public Connection getConnection()
86     {
87         return connection;
88     }
89
90     public static void setLock(Object lock) {
91         JDBCDataBase.lock = lock;
92     }
93
94     public static void setLogger(Logger logger) {
95         JDBCDataBase.logger = logger;
96     }
97
98     public void setConnected(boolean connected) {
99         isConnected = connected;
100    }
101
102    public void setConnection(Connection connection) {
103        this.connection = connection;
104    }
105
106    public void setStatement(Statement statement) {
107        this.statement = statement;
108    }
109 }
110
```

```
1 package il.ac.hit.CashFlowManagement.model;
2
3 import il.ac.hit.CashFlowManagement.exception.
4 GetAllUserExpensesException;
5 import il.ac.hit.CashFlowManagement.exception.InsertToDBException;
6 import il.ac.hit.CashFlowManagement.exception.JDBCDataBaseLogicException;
7 import il.ac.hit.CashFlowManagement.view.LoginForm;
8 import org.apache.log4j.Logger;
9 import org.jetbrains.annotations.NotNull;
10
11 import java.sql.ResultSet;
12 import java.text.SimpleDateFormat;
13
14 /**
15  * The Expenses Logic class
16 */
17 public class ExpensesLogic implements IExpenseHandle
18 {
19     private static Logger logger;
20     private final String nameOfTable;
21     private JDBCDataBaseLogic DataBase;
22     private SimpleDateFormat formatter;
23
24     /**
25      * ExpensesLogic C'tor
26     */
27     public ExpensesLogic() {
28         setLogger(Logger.getLogger(ExpensesLogic.class));
29         nameOfTable = LoginForm.username.toUpperCase() + "_Expenses".
30         toUpperCase();
31         setFormatter(new SimpleDateFormat("dd/MM/yyyy"));
32         set DataBase(JDBCDataBaseLogic.getInstance());
33         try
34         {
35             DataBase.createTableIfNotExist(
36                 nameOfTable,
37                 "Date",
38                 "varchar(300)",
39                 "Classification",
40                 "varchar(300)",
41                 "Details",
42                 "varchar(300)",
43                 "Cost",
44                 "varchar(300)");
45             logger.info("ExpensesLogic was create successfully!!!");
46         }
47     }
48 }
```

```

44         } catch (JDBCDataBaseLogicException e){
45             logger.info(e.getMessage());
46             e.printStackTrace();
47         }
48     }
49
50 /**
51 * @see IExpenseHandle {@Link #addExpense(Expense)}
52 */
53 public void addExpense(@NotNull Expense expense) {
54     try {
55         String date = formatter.format(expense.getDate());
56         String parameters =
57             """ + date + """ + ","
58             + """ + expense.getClassification() + """
59             + """ + expense.getDetails() + """ + ","
60             + """ + Double.toString(expense.getCost
61             ()) + """;
62
63         DataBase.insertValue(nameOfTable, parameters);
64         logger.info("ExpensesLogic.addExpense finished
65 successfully");
66     } catch (InsertToDBException e) {
67         logger.info("ExpensesLogic.addExpense failed");
68         e.printStackTrace();
69     }
70 /**
71 * @see ExpensesLogic {@Link #getAllUserExpenses()}
72 */
73 public ResultSet getAllUserExpenses() throws
74 GetAllUserExpensesException {
75     try {
76         return DataBase.query("select * from " + nameOfTable);
77     } catch (JDBCDataBaseLogicException e){
78         throw new GetAllUserExpensesException("failed to get all
79 users expenses", e);
80     }
81 }
82
83 public static void setLogger(Logger logger) {
84     ExpensesLogic.logger = logger;
85 }
86
87 }
88
89 }
90
91 }
92
93 }
94
95 }
96
97 }
98
99 }
100
101 }
102
103 }
104
105 }
106
107 }
108
109 }
110
111 }
112
113 }
114
115 }
116
117 }
118
119 }
120
121 }
122
123 }
124
125 }
126
127 }
128
129 }
130
131 }
132
133 }
134
135 }
136
137 }
138
139 }
140
141 }
142
143 }
144
145 }
146
147 }
148
149 }
150
151 }
152
153 }
154
155 }
156
157 }
158
159 }
160
161 }
162
163 }
164
165 }
166
167 }
168
169 }
170
171 }
172
173 }
174
175 }
176
177 }
178
179 }
180
181 }
182
183 }
184
185 }
186
187 }
188
189 }
190
191 }
192
193 }
194
195 }
196
197 }
198
199 }
200
201 }
202
203 }
204
205 }
206
207 }
208
209 }
210
211 }
212
213 }
214
215 }
216
217 }
218
219 }
220
221 }
222
223 }
224
225 }
226
227 }
228
229 }
230
231 }
232
233 }
234
235 }
236
237 }
238
239 }
240
241 }
242
243 }
244
245 }
246
247 }
248
249 }
250
251 }
252
253 }
254
255 }
256
257 }
258
259 }
259
260 }
261
262 }
263
264 }
265
266 }
267
268 }
269
270 }
271
272 }
273
274 }
275
276 }
277
278 }
279
279
280 }
281
282 }
283
284 }
285
286 }
287
288 }
289
289
290 }
291
292 }
293
294 }
295
296 }
297
298 }
299
299
300 }
301
302 }
303
304 }
305
306 }
307
308 }
309
309
310 }
311
312 }
313
314 }
315
316 }
317
318 }
319
319
320 }
321
322 }
323
324 }
325
326 }
327
328 }
329
329
330 }
331
332 }
333
334 }
335
336 }
337
338 }
339
339
340 }
341
342 }
343
344 }
345
346 }
347
348 }
349
349
350 }
351
352 }
353
354 }
355
356 }
357
358 }
359
359
360 }
361
362 }
363
364 }
365
366 }
367
368 }
369
369
370 }
371
372 }
373
374 }
375
376 }
377
378 }
379
379
380 }
381
382 }
383
384 }
385
386 }
387
388 }
389
389
390 }
391
392 }
393
394 }
395
396 }
397
398 }
399
399
400 }
401
402 }
403
404 }
405
406 }
407
408 }
409
409
410 }
411
412 }
413
414 }
415
416 }
417
418 }
419
419
420 }
421
422 }
423
424 }
425
426 }
427
428 }
429
429
430 }
431
432 }
433
434 }
435
436 }
437
438 }
439
439
440 }
441
442 }
443
444 }
445
446 }
447
448 }
449
449
450 }
451
452 }
453
454 }
455
456 }
457
458 }
459
459
460 }
461
462 }
463
464 }
465
466 }
467
468 }
469
469
470 }
471
472 }
473
474 }
475
476 }
477
478 }
479
479
480 }
481
482 }
483
484 }
485
486 }
487
488 }
489
489
490 }
491
492 }
493
494 }
495
496 }
497
498 }
499
499
500 }
501
502 }
503
504 }
505
506 }
507
508 }
509
509
510 }
511
512 }
513
514 }
515
516 }
517
518 }
519
519
520 }
521
522 }
523
524 }
525
526 }
527
528 }
529
529
530 }
531
532 }
533
534 }
535
536 }
537
538 }
539
539
540 }
541
542 }
543
544 }
545
546 }
547
548 }
549
549
550 }
551
552 }
553
554 }
555
556 }
557
558 }
559
559
560 }
561
562 }
563
564 }
565
566 }
567
568 }
569
569
570 }
571
572 }
573
574 }
575
576 }
577
578 }
579
579
580 }
581
582 }
583
584 }
585
586 }
587
588 }
589
589
590 }
591
592 }
593
594 }
595
596 }
597
598 }
599
599
600 }
601
602 }
603
604 }
605
606 }
607
608 }
609
609
610 }
611
612 }
613
614 }
615
616 }
617
618 }
619
619
620 }
621
622 }
623
624 }
625
626 }
627
628 }
629
629
630 }
631
632 }
633
634 }
635
636 }
637
638 }
639
639
640 }
641
642 }
643
644 }
645
646 }
647
648 }
649
649
650 }
651
652 }
653
654 }
655
656 }
657
658 }
659
659
660 }
661
662 }
663
664 }
665
666 }
667
668 }
669
669
670 }
671
672 }
673
674 }
675
676 }
677
678 }
679
679
680 }
681
682 }
683
684 }
685
686 }
687
688 }
689
689
690 }
691
692 }
693
694 }
695
696 }
697
698 }
699
699
700 }
701
702 }
703
704 }
705
706 }
707
708 }
709
709
710 }
711
712 }
713
714 }
715
716 }
717
718 }
719
719
720 }
721
722 }
723
724 }
725
726 }
727
728 }
729
729
730 }
731
732 }
733
734 }
735
736 }
737
738 }
739
739
740 }
741
742 }
743
744 }
745
746 }
747
748 }
749
749
750 }
751
752 }
753
754 }
755
756 }
757
758 }
759
759
760 }
761
762 }
763
764 }
765
766 }
767
768 }
769
769
770 }
771
772 }
773
774 }
775
776 }
777
778 }
779
779
780 }
781
782 }
783
784 }
785
786 }
787
788 }
789
789
790 }
791
792 }
793
794 }
795
796 }
797
798 }
799
799
800 }
801
802 }
803
804 }
805
806 }
807
808 }
809
809
810 }
811
812 }
813
814 }
815
816 }
817
818 }
819
819
820 }
821
822 }
823
824 }
825
826 }
827
828 }
829
829
830 }
831
832 }
833
834 }
835
836 }
837
838 }
839
839
840 }
841
842 }
843
844 }
845
846 }
847
848 }
849
849
850 }
851
852 }
853
854 }
855
856 }
857
858 }
859
859
860 }
861
862 }
863
864 }
865
866 }
867
868 }
869
869
870 }
871
872 }
873
874 }
875
876 }
877
878 }
879
879
880 }
881
882 }
883
884 }
885
886 }
887
888 }
889
889
890 }
891
892 }
893
894 }
895
896 }
897
898 }
899
899
900 }
901
902 }
903
904 }
905
906 }
907
908 }
909
909
910 }
911
912 }
913
914 }
915
916 }
917
918 }
919
919
920 }
921
922 }
923
924 }
925
926 }
927
928 }
929
929
930 }
931
932 }
933
934 }
935
936 }
937
938 }
939
939
940 }
941
942 }
943
944 }
945
946 }
947
948 }
949
949
950 }
951
952 }
953
954 }
955
956 }
957
958 }
959
959
960 }
961
962 }
963
964 }
965
966 }
967
968 }
969
969
970 }
971
972 }
973
974 }
975
976 }
977
978 }
979
979
980 }
981
982 }
983
984 }
985
986 }
987
988 }
989
989
990 }
991
992 }
993
994 }
995
996 }
997
998 }
999
999
1000 }
1001
1002 }
1003
1004 }
1005
1006 }
1007
1008 }
1009
1009
1010 }
1011
1012 }
1013
1014 }
1015
1016 }
1017
1018 }
1019
1019
1020 }
1021
1022 }
1023
1024 }
1025
1026 }
1027
1028 }
1029
1029
1030 }
1031
1032 }
1033
1034 }
1035
1036 }
1037
1038 }
1039
1039
1040 }
1041
1042 }
1043
1044 }
1045
1046 }
1047
1048 }
1049
1049
1050 }
1051
1052 }
1053
1054 }
1055
1056 }
1057
1058 }
1059
1059
1060 }
1061
1062 }
1063
1064 }
1065
1066 }
1067
1068 }
1069
1069
1070 }
1071
1072 }
1073
1074 }
1075
1076 }
1077
1078 }
1079
1079
1080 }
1081
1082 }
1083
1084 }
1085
1086 }
1087
1088 }
1089
1089
1090 }
1091
1092 }
1093
1094 }
1095
1096 }
1097
1098 }
1099
1099
1100 }
1101
1102 }
1103
1104 }
1105
1106 }
1107
1108 }
1109
1109
1110 }
1111
1112 }
1113
1114 }
1115
1116 }
1117
1118 }
1119
1119
1120 }
1121
1122 }
1123
1124 }
1125
1126 }
1127
1128 }
1129
1129
1130 }
1131
1132 }
1133
1134 }
1135
1136 }
1137
1138 }
1139
1139
1140 }
1141
1142 }
1143
1144 }
1145
1146 }
1147
1148 }
1149
1149
1150 }
1151
1152 }
1153
1154 }
1155
1156 }
1157
1158 }
1159
1159
1160 }
1161
1162 }
1163
1164 }
1165
1166 }
1167
1168 }
1169
1169
1170 }
1171
1172 }
1173
1174 }
1175
1176 }
1177
1178 }
1179
1179
1180 }
1181
1182 }
1183
1184 }
1185
1186 }
1187
1188 }
1189
1189
1190 }
1191
1192 }
1193
1194 }
1195
1196 }
1197
1198 }
1199
1199
1200 }
1201
1202 }
1203
1204 }
1205
1206 }
1207
1208 }
1209
1209
1210 }
1211
1212 }
1213
1214 }
1215
1216 }
1217
1218 }
1219
1219
1220 }
1221
1222 }
1223
1224 }
1225
1226 }
1227
1228 }
1229
1229
1230 }
1231
1232 }
1233
1234 }
1235
1236 }
1237
1238 }
1239
1239
1240 }
1241
1242 }
1243
1244 }
1245
1246 }
1247
1248 }
1249
1249
1250 }
1251
1252 }
1253
1254 }
1255
1256 }
1257
1258 }
1259
1259
1260 }
1261
1262 }
1263
1264 }
1265
1266 }
1267
1268 }
1269
1269
1270 }
1271
1272 }
1273
1274 }
1275
1276 }
1277
1278 }
1279
1279
1280 }
1281
1282 }
1283
1284 }
1285
1286 }
1287
1288 }
1289
1289
1290 }
1291
1292 }
1293
1294 }
1295
1296 }
1297
1298 }
1299
1299
1300 }
1301
1302 }
1303
1304 }
1305
1306 }
1307
1308 }
1309
1309
1310 }
1311
1312 }
1313
1314 }
1315
1316 }
1317
1318 }
1319
1319
1320 }
1321
1322 }
1323
1324 }
1325
1326 }
1327
1328 }
1329
1329
1330 }
1331
1332 }
1333
1334 }
1335
1336 }
1337
1338 }
1339
1339
1340 }
1341
1342 }
1343
1344 }
1345
1346 }
1347
1348 }
1349
1349
1350 }
1351
1352 }
1353
1354 }
1355
1356 }
1357
1358 }
1359
1359
1360 }
1361
1362 }
1363
1364 }
1365
1366 }
1367
1368 }
1369
1369
1370 }
1371
1372 }
1373
1374 }
1375
1376 }
1377
1378 }
1379
1379
1380 }
1381
1382 }
1383
1384 }
1385
1386 }
1387
1388 }
1389
1389
1390 }
1391
1392 }
1393
1394 }
1395
1396 }
1397
1398 }
1399
1399
1400 }
1401
1402 }
1403
1404 }
1405
1406 }
1407
1408 }
1409
1409
1410 }
1411
1412 }
1413
1414 }
1415
1416 }
1417
1418 }
1419
1419
1420 }
1421
1422 }
1423
1424 }
1425
1426 }
1427
1428 }
1429
1429
1430 }
1431
1432 }
1433
1434 }
1435
1436 }
1437
1438 }
1439
1439
1440 }
1441
1442 }
1443
1444 }
1445
1446 }
1447
1448 }
1449
1449
1450 }
1451
1452 }
1453
1454 }
1455
1456 }
1457
1458 }
1459
1459
1460 }
1461
1462 }
1463
1464 }
1465
1466 }
1467
1468 }
1469
1469
1470 }
1471
1472 }
1473
1474 }
1475
1476 }
1477
1478 }
1479
1479
1480 }
1481
1482 }
1483
1484 }
1485
1486 }
1487
1488 }
1489
1489
1490 }
1491
1492 }
1493
1494 }
1495
1496 }
1497
1498 }
1499
1499
1500 }
1501
1502 }
1503
1504 }
1505
1506 }
1507
1508 }
1509
1509
1510 }
1511
1512 }
1513
1514 }
1515
1516 }
1517
1518 }
1519
1519
1520 }
1521
1522 }
1523
1524 }
1525
1526 }
1527
1528 }
1529
1529
1530 }
1531
1532 }
1533
1534 }
1535
1536 }
1537
1538 }
1539
1539
1540 }
1541
1542 }
1543
1544 }
1545
1546 }
1547
1548 }
1549
1549
1550 }
1551
1552 }
1553
1554 }
1555
1556 }
1557
1558 }
1559
1559
1560 }
1561
1562 }
1563
1564 }
1565
1566 }
1567
1568 }
1569
1569
1570 }
1571
1572 }
1573
1574 }
1575
1576 }
1577
1578 }
1579
1579
1580 }
1581
1582 }
1583
1584 }
1585
1586 }
1587
1588 }
1589
1589
1590 }
1591
1592 }
1593
1594 }
1595
1596 }
1597
1598 }
1599
1599
1600 }
1601
1602 }
1603
1604 }
1605
1606 }
1607
1608 }
1609
1609
1610 }
1611
1612 }
1613
1614 }
1615
1616 }
1617
1618 }
1619
1619
1620 }
1621
1622 }
1623
1624 }
1625
1626 }
1627
1628 }
1629
1629
1630 }
1631
1632 }
1633
1634 }
1635
1636 }
1637
1638 }
1639
1639
1640 }
1641
1642 }
1643
1644 }
1645
1646 }
1647
1648 }
1649
1649
1650 }
1651
1652 }
1653
1654 }
1655
1656 }
1657
1658 }
1659
1659
1660 }
1661
1662 }
1663
1664 }
1665
1666 }
1667
1668 }
1669
1669
1670 }
1671
1672 }
1673
1674 }
1675
1676 }
1677
1678 }
1679
1679
1680 }
1681
1682 }
1683
1684 }
1685
1686 }
1687
1688 }
1689
1689
1690 }
1691
1692 }
1693
1694 }
1695
1696 }
1697
1698 }
1699
1699
1700 }
1701
1702 }
1703
1704 }
1705
1706 }
1707
1708 }
1709
1709
1710 }
1711
1712 }
1713
1714 }
1715
1716 }
1717
1718 }
1719
1719
1720 }
1721
1722 }
1723
1724 }
1725
1726 }
1727
1728 }
1729
1729
1730 }
1731
1732 }
1733
1734 }
1735
1736 }
1737
1738 }
1739
1739
1740 }
1741
1742 }
1743
1744 }
1745
1746 }
1747
1748 }
1749
1749
1750 }
1751
1752 }
1753
1754 }
1755
1756 }
1757
1758 }
1759
1759
1760 }
1761
1762 }
1763
1764 }
1765
1766 }
1767
1768 }
1769
1769
1770 }
1771
1772 }
1773
1774 }
1775
1776 }
1777
1778 }
1779
1779
1780 }
1781
1782 }
1783
1784 }
1785
1786 }
1787
1788 }
1789
1789
1790 }
1791
1792 }
1793
1794 }
1795
1796 }
1797
1798 }
1799
1799
1800 }
1801
1802 }
1803
1804 }
1805
1806 }
1807
1808 }
1809
1809
1810 }
1811
1812 }
1813
1814 }
1815
1816 }
1817
1818 }
1819
1819
1820 }
1821
1822 }
1823
1824 }
1825
1826 }
1827
1828 }
1829
1829
1830 }
1831
1832 }
1833
1834 }
1835
1836 }
1837
1838 }
1839
1839
1840 }
1841
1842 }
1843
1844 }
1845
1846 }
1847
1848 }
1849
1849
1850 }
1851
1852 }
1853
1854 }
1855
1856 }
1857
1858 }
1859
1859
1860 }
1861
1862 }
1863
1864 }
1865
1866 }
1867
1868 }
1869
1869
1870 }
1871
1872 }
1873
1874 }
1875
1876 }
1877
1878 }
1879
1879
1880 }
1881
1882 }
1883
1884 }
1885
1886 }
1887
1888 }
1889
1889
1890 }
1891
1892 }
1893
1894 }
1895
1896 }
1897
1898 }
1899
1899
1900 }
1901
1902 }
1903
1904 }
1905
1906 }
1907
1908 }
1909
1909
1910 }
1911
1912 }
1913
1914 }
1915
1916 }
1917
1918 }
1919
1919
1920 }
1921
1922 }
1923
1924 }
1925
1926 }
1927
1928 }
1929
1929
1930 }
1931
1932 }
1933
1934 }
1935
1936 }
1937
1938 }
1939
1939
1940 }
1941
1942 }
1943
1944 }
1945
1946 }
1947
1948 }
1949
1949
1950 }
1951
1952 }
1953
1954 }
1955
1956 }
1957
1958 }
1959
1959
1960 }
1961
1962 }
1963
1964 }
1965
1966 }
1967
1968 }
1969
1969
1970 }
1971
1972 }
1973
1974 }
1975
1976 }
1977
1978 }
1979
1979
1980 }
1981
1982 }
1983
1984 }
1985
1986 }
1987
1988 }
1989
1989
1990 }
1991
1992 }
1993
1994 }
1995
1996 }
1997
1998 }
1999
1999
2000 }
2001
2002 }
2003
2004 }
2005
2006 }
2007
2008 }
2009
2009
2010 }
2011
2012 }
2013
2014 }
2015
2016 }
2017
2018 }
2019
2019
2020 }
2021
2022 }
2023
2024 }
2025
2026 }
2027
2028 }
2029
2029
2030 }
2031
2032 }
2033
2034 }
2035
2036 }
2037
2038 }
2039
2039
2040 }
2041
2042 }
2043
2044 }
2045
2046 }
2047
2048 }
2049
2049
2050 }
2051
2052 }
2053
2054 }
2055
2056 }
2057
2058 }
2059
2059
2060 }
2061
2062 }
2063
2064 }
2065
2066 }
2067
2068 }
2069
2069
2070 }
2071
2072 }
2073
2074 }
2075
2076 }
2077
2078 }
2079
2079
2080 }
2081
2082 }
2083
2084 }
2085
2086 }
2087
2088 }
2089
2089
2090 }
2091
2092 }
2093
2094 }
2095
2096 }
2097
2098 }
2099
2099
2100 }
2101
2102 }
2103
2104 }
2105
2106 }
2107
2108 }
2109
2109
2110 }
2111
2112 }
2113
2114 }
2115
2116 }
2117
2118 }
2119
2119
2120 }
2121
2122 }
2123
2124 }
2125
2126 }
2127
2128 }
2129
2129
2130 }
2131
2132 }
2133
2134 }
2135
2136 }
2137
2138 }
2139
2139
2140 }
2141
2142 }
2143
2144 }
2145
2146 }
2147
2148 }
2149
2149
2150 }
2151
2152 }
2153
2154 }
2155
2156 }
2157
2158 }
2159
2159
2160 }
2161
2162 }
2163
2164 }
2165
2166 }
2167
2168 }
2169
2169
2170 }
2171
2172 }
2173
2174 }
2175
2176 }
2177
2178 }
2179
2179
2180 }
2181
2182 }
2183
2184 }
2185
2186 }
2187
2188 }
2189
2189
2190 }
2191
2192 }
2193
2194 }
2195
2196 }
2197
2198 }
2199
2199
2200 }
2201
2202 }
2203
2204 }
2205
2206 }
2207
2208 }
2209
2209
2210 }
2211
2212 }
2213
2214 }
2215
2216 }
2217
2218 }
2219
2219
2220 }
2221
2222 }
2223
2224 }
2225
2226 }
2227
2228 }
2229
2229
2230 }
2231
2232 }
2233
2234 }
2235
2236 }
2237
223
```

```
85     public void set DataBase(JDBCDataBaseLogic DataBase) {  
86         this.dataBase = DataBase;  
87     }  
88  
89     public void setFormatter(SimpleDateFormat formatter) {  
90         this.formatter = formatter;  
91     }  
92 }  
93  
94
```

```
1 package il.ac.hit.CashFlowManagement.model;
2
3 import il.ac.hit.CashFlowManagement.exception.
4 GetAllUserExpensesException;
5 import il.ac.hit.CashFlowManagement.exception.InsertToDBException;
6 import org.jetbrains.annotations.NotNull;
7
8 import java.sql.ResultSet;
9
10 /**
11  * this interface define the operations each ExpensesLogic must
12  * implement
13 */
14 public interface IExpenseHandle {
15     /**
16      * add an expense to the user expenses in the DB
17      * @param expense full expense information, can't be null
18      * @throws InsertToDBException throws exception if failed to
19      * insert the new expenses to the DB
20      */
21     void addExpense(@NotNull Expense expense) throws
22     InsertToDBException;
23
24     /**
25      * get all user expenses from the DB
26      * @return ResultSet object that contains full information about
27      * all user expenses from DB
28      * @throws GetAllUserExpensesException throws exception if failed
29      * to get all users expenses
30      */
31     ResultSet getAllUserExpenses() throws GetAllUserExpensesException
32 ;
33 }
```

```
1 package il.ac.hit.CashFlowManagement.model;
2
3 import il.ac.hit.CashFlowManagement.exception.InsertToDBException;
4 import il.ac.hit.CashFlowManagement.exception.JDBCDataBaseException;
5 import il.ac.hit.CashFlowManagement.exception.
JDBCDataBaseLogicException;
6 import org.apache.log4j.Logger;
7 import org.jetbrains.annotations.NotNull;
8
9 import java.sql.DatabaseMetaData;
10 import java.sql.ResultSet;
11 import java.sql.SQLException;
12 import java.sql.Statement;
13
14 /**
15 * The Data Base Logic class
16 */
17 public class JDBCDataBaseLogic implements IJDBCDataBaseLogic {
18     private static JDBCDataBaseLogic instance = null;
19     private static Object lock = new Object();
20     private static Logger logger;
21     private DataBase DataBase = null;
22     private Statement statement = null;
23     private DatabaseMetaData metaData = null;
24
25     private JDBCDataBaseLogic() {
26         try {
27             setLogger(Logger.getLogger(JDBCDataBaseLogic.class));
28             setDataBase(DataBase.getInstance());
29             setMetaData(DataBase.getConnection().getMetaData());
30             setStatement(DataBase.getStatement());
31             logger.info("DataBaseLogic was create successfully!!!");
32         } catch (SQLException | JDBCDataBaseException e) {
33             e.printStackTrace();
34         }
35     }
36
37 /**
38 * check if there is a singleton instance of DataBaseLogic and
39 * create it's if needed
40 * @return instance of DataBaseLogic
41 */
42     public static JDBCDataBaseLogic getInstance() {
43         if (instance == null) {
44             synchronized (lock) {
45                 if (instance == null) {
```

```

45                     instance = new JDBCDataBaseLogic();
46                 }
47             }
48         }
49
50     return instance;
51 }
52
53 /**
54  * if the table is not exists in the DB the table would be
55  * created
56  * @see IJDBCDataBaseLogic {@Link #createTableIfNotExist(String,
57  String...)}
58  */
59
60 public boolean createTableIfNotExist(@NotNull String iNameOfTable
61 , @NotNull String... iParameters) throws JDBCDataBaseLogicException {
62     boolean newTableCreated = false;
63
64     if (iParameters.length % 2 != 0) {
65         logger.info("Create Table was failed, params is not even"
66     );
67         throw new JDBCDataBaseLogicException("Please enter the
68 parameters as follows: Param1Name ,Param1Type, Param2Name ,Param2Type
69 .... , ParamName ,ParamType");
70     }
71
72     iNameOfTable = iNameOfTable.toUpperCase();
73     String str = stringsConcat(iParameters);
74     try {
75         ResultSet rs = metaData.getTables(null, "APP",
76 iNameOfTable, null);
77         if (!rs.next()) {
78             String sqlStatement = "create table " + iNameOfTable
79             + str;
80             statement.execute(sqlStatement);
81             newTableCreated = true;
82             logger.info("New table created successfully, new
83 table name is " + iNameOfTable);
84         }
85         else{
86             newTableCreated = false;
87             logger.info(iNameOfTable + " table already exist and
88 therefor new table wasn't created");
89         }
90     } catch (SQLException e) {
91         logger.info("New table creation failed");
92     }
93 }
```

```

81             throw new JDBCDataBaseLogicException(e.getMessage());
82         }
83     } finally {
84
85         return newTableCreated;
86     }
87 }
88
89 /**
90 * @see IJDBCDataBaseLogic {@Link #insertValue(String, String)}
91 */
92 public void insertValue(@NotNull String iNameOfTable, @NotNull
String iParameters) throws InsertToDBException {
93     String sqlStatement = "insert into " + iNameOfTable.
toUpperCase() + " values (" + iParameters + ")";
94
95     try {
96         statement.executeUpdate(sqlStatement);
97         logger.info("Parameters has been successfully inserted
to " + iNameOfTable);
98     } catch (SQLException e)
99     {
100         logger.info("Parameters insertion to " + iNameOfTable +
" failed");
101         throw new InsertToDBException("Parameters insertion to "
+ iNameOfTable + " failed", e);
102     }
103 }
104
105 /**
106 * @see IJDBCDataBaseLogic {@Link #query(String)}
107 */
108 @NotNull
109 public ResultSet query(@NotNull String iQuery) throws
JDBCDataBaseLogicException{
110     ResultSet rs = null;
111     try {
112         rs = statement.executeQuery(iQuery);
113     } catch (SQLException e) {
114         throw new JDBCDataBaseLogicException("query execution
failed " + e.getMessage(), e);
115     }
116
117     return rs;
118 }
119

```

```

120     /**
121      * see IJDBCDataBaseLogic {@Link #removeTable(String)}
122      */
123      @NotNull
124      public void removeTable(@NotNull String iNameOfTable) throws
JDBCDataBaseLogicException {
125          try {
126              statement.execute("DROP TABLE "+iNameOfTable);
127              logger.info(iNameOfTable + " successfully removed from
DB");
128          } catch (SQLException e) {
129              logger.info("failed to remove" + iNameOfTable + " from
DB");
130              throw new JDBCDataBaseLogicException("failed to remove "
+ iNameOfTable + " from DB");
131          }
132      }
133
134      /**
135         * get an array or a sequence of arguments and build a string
for the SQL query from it
136         * param iString an array or a sequence of arguments, can't be
null
137         * return string for the SQL query
138     */
139     @NotNull
140     private String stringsConcat(@NotNull String... iString) {
141         StringBuffer stringBuffer = new StringBuffer();
142         int length = iString.length - 1;
143
144         stringBuffer.append("(");
145         for (int i = 0; i < length; i += 2) {
146             stringBuffer.append(iString[i]);
147             stringBuffer.append(" ");
148             stringBuffer.append(iString[i + 1]);
149             stringBuffer.append(", ");
150         }
151         stringBuffer.setLength(stringBuffer.length() - 2);
152         stringBuffer.append(")");
153
154         return stringBuffer.toString();
155     }
156
157     /**
158         * setter of a Locking mechanism for a singleton instance of
DataBaseLogic

```

```
159     * @param Lock static created object
160     */
161     public static void setLock(Object lock) {
162         JDBCDataBaseLogic.lock = lock;
163     }
164
165     public void set DataBase(JDBCDataBase DataBase) {
166         this.dataBase = DataBase;
167     }
168
169     public void set Statement(Statement statement) {
170         this.statement = statement;
171     }
172
173     public void set MetaData(DatabaseMetaData metaData) {
174         this.metaData = metaData;
175     }
176
177     public static void set Logger(Logger logger) {
178         JDBCDataBaseLogic.logger = logger;
179     }
180 }
```

```
1 package il.ac.hit.CashFlowManagement.model;
2
3 import il.ac.hit.CashFlowManagement.exception.InsertToDBException;
4 import il.ac.hit.CashFlowManagement.exception.JDBCDataBaseLogicException;
5 import org.jetbrains.annotations.NotNull;
6
7 import java.sql.ResultSet;
8
9 /**
10  * this interface define the operations that the JDBCDataBaseLogic
11  * must implement
12 */
13 public interface IJDBCDataBaseLogic<T>
14 {
15     /**
16      * param nameOfTable name of the table
17      * param parameters parameter to Create
18      * return true if a new table created, false if the table is
19      * already exists
20      * throws JDBCDataBaseLogicException if failed to create table
21      * because params is not even or SQLException
22      */
23     boolean createTableIfNotExist(@NotNull String nameOfTable, @
24         NotNull String... parameters) throws JDBCDataBaseLogicException;
25
26     /**
27      * Remove table from DB query (DROP TABLE SQL query)
28      * param nameOfTable table name to remove, can't be null
29      * throws JDBCDataBaseLogicException if failed to remove table
30      * because of SQLException
31      */
32     void removeTable(@NotNull String nameOfTable) throws
33     JDBCDataBaseLogicException;
34
35     /**
36      * Insert a values into a table in the DB
37      * param nameOfTable name of the table, can't be null
38      * param parameters row parameters to insert, can't be null
39      * throws InsertToDBException if failed to insert value to table
40      * because of SQLException
41      */
42     void insertValue(@NotNull String nameOfTable, String parameters)
43     throws InsertToDBException;
44
45     /**
46      *
47      */
48 }
```

```
38     * Execute DB Query
39     * param query query to be executed, can't be null
40     * return ResultSet object that contains the results of the SQL
41       query executing
42     * throws JDBC DataBaseLogicException if failed to execute query
43       because of SQLException
43     */
43     ResultSet query(@NotNull String query) throws
43       JDBC DataBaseLogicException;
44 }
```

```
1 package il.ac.hit.CashFlowManagement.exception;  
2  
3 /**  
4  * Exception for InsertToDB query  
5 */  
6 public class InsertToDBException extends Exception {  
7  
8     /**  
9      * Parameterless C'tor  
10     */  
11    public InsertToDBException() {  
12        super();  
13    }  
14  
15    /**  
16     * C'tor  
17     * @param message message that represent the exception  
18     * @param cause warp the throwable  
19     */  
20    public InsertToDBException(String message, Throwable cause) {  
21        super(message, cause);  
22    }  
23  
24    /**  
25     * C'tor  
26     * @param message message that represent the exception  
27     */  
28    public InsertToDBException(String message) {  
29        super(message);  
30    }  
31 }
```

```
1 package il.ac.hit.CashFlowManagement.exception;  
2  
3 /**  
4  * Exception for FormCasting  
5 */  
6 public class FormCastingException extends Exception {  
7  
8     /**  
9      * Parameterless C'tor  
10     */  
11    public FormCastingException() {  
12        super();  
13    }  
14  
15    /**  
16     * C'tor  
17     * @param message message that represent the exception  
18     * @param cause warp the throwable  
19     */  
20    public FormCastingException(String message, Throwable cause) {  
21        super(message, cause);  
22    }  
23  
24    /**  
25     * C'tor  
26     * @param message message that represent the exception  
27     */  
28    public FormCastingException(String message) {  
29        super(message);  
30    }  
31 }
```

```
1 package il.ac.hit.CashFlowManagement.exception;
2
3 /**
4  * Exception for JDBC DataBase
5 */
6 public class JDBCDataBaseException extends Exception {
7     /**
8      * Parameterless C'tor
9      */
10    public JDBCDataBaseException() {
11        super();
12    }
13
14    /**
15     * C'tor
16     * @param message message that represent the exception
17     * @param cause warp the throwable
18     */
19    public JDBCDataBaseException(String message, Throwable cause) {
20        super(message, cause);
21    }
22
23    /**
24     * C'tor
25     * @param message message that represent the exception
26     */
27    public JDBCDataBaseException(String message) {
28        super(message);
29    }
30 }
```

```
1 package il.ac.hit.CashFlowManagement.exception;  
2  
3 /**  
4  * Exception for JDBC DataBase Logic  
5 */  
6 public class JDBCDataBaseLogicException extends Exception{  
7  
8     /**  
9      * Parameterless C'tor  
10     */  
11    public JDBCDataBaseLogicException() {  
12        super();  
13    }  
14  
15    /**  
16     * C'tor  
17     * @param message message that represent the exception  
18     * @param cause warp the throwable  
19     */  
20    public JDBCDataBaseLogicException(String message, Throwable cause  
) {  
21        super(message, cause);  
22    }  
23  
24    /**  
25     * C'tor  
26     * @param message message that represent the exception  
27     */  
28    public JDBCDataBaseLogicException(String message) {  
29        super(message);  
30    }  
31 }  
32
```

```
1 package il.ac.hit.CashFlowManagement.exception;  
2  
3 /**  
4  * Exception for getAllUserExpenses query  
5 */  
6 public class GetAllUserExpensesException extends Exception {  
7  
8     /**  
9      * Parameterless C'tor  
10     */  
11    public GetAllUserExpensesException() {  
12        super();  
13    }  
14  
15    /**  
16     * C'tor  
17     * @param message message that represent the exception  
18     * @param cause warp the throwable  
19     */  
20    public GetAllUserExpensesException(String message, Throwable  
cause) {  
21        super(message, cause);  
22    }  
23  
24    /**  
25     * C'tor  
26     * @param message message that represent the exception  
27     */  
28    public GetAllUserExpensesException(String message) {  
29        super(message);  
30    }  
31 }
```

```
1 package il.ac.hit.CashFlowManagement.viewmodel;
2 import il.ac.hit.CashFlowManagement.exception.
3 GetAllUserExpensesException;
4 import org.apache.log4j.Logger;
5
6 import javax.swing.*;
7 import javax.swing.table.DefaultTableModel;
8 import javax.swing.table.TableModel;
9 import javax.swing.table.TableRowSorter;
10 import java.sql.ResultSet;
11 import java.sql.SQLException;
12 import java.time.LocalDate;
13 import java.time.format.DateTimeFormatter;
14 import java.util.Comparator;
15 import java.util.Locale;
16 /**
17 * The Table Data class
18 */
19
20 public class TableData
21 {
22     private static Logger logger;
23     private IViewModel viewModel;
24     private final String[] columnNames = { "Date", "Classification",
25                                         "Details", "Cost"};
26     //creating gui components
27     private JTable dataTable;
28     private JScrollPane tableScrollPane;
29     private DefaultTableModel model;
30     private JButton back = new JButton("Back");
31
32     /**
33      * TableData C'tor
34     */
35     public TableData() {
36         init();
37     }
38
39     /**
40      * initialize table view and sorters logic
41     */
42     private void init() {
43         setLogger();
44         setViewModel();
```

```

45      dataTable = new JTable(new DefaultTableModel(null,
46          columnNames));
47      dataTable.setAutoCreateRowSorter(true);
48      dataTable.setAutoResizeMode(JTable.AUTO_RESIZE_OFF);
49      dataTable.setAutoscrolls(true);
50      dataTable.setAutoResizeMode(JTable.AUTO_RESIZE_ALL_COLUMNS);
51      tableJScrollPane = new JScrollPane(dataTable);
52      model = (DefaultTableModel) dataTable.getModel();
53      TableRowSorter<TableModel> sorter = new TableRowSorter<>(
54          dataTable.getModel());
55      sorter.setComparator(dataTable.getColumnCount() -1 , new
56      Comparator<String>() {
57          @Override
58          public int compare(String cost1, String cost2) {
59              Double cost1d, cost2d;
60
61              cost1d = Double.parseDouble(cost1);
62              cost2d = Double.parseDouble(cost2);
63
64              return cost1d.compareTo(cost2d);
65          }
66      });
67      sorter.setComparator(0 , new Comparator<String>() {
68          @Override
69          public int compare(String date1, String date2) {
70              LocalDate date1LD, date2LD;
71              DateTimeFormatter formatter = DateTimeFormatter.
72                  ofPattern("dd/MM/yyyy", Locale.getDefault());
73
74              date1LD = LocalDate.parse(date1, formatter);
75              date2LD = LocalDate.parse(date2, formatter);
76
77              return date1LD.compareTo(date2LD);
78          }
79      });
80      dataTable.setRowSorter(sorter);
81      dataTable.getRowSorter().toggleSortOrder(0);
82      dataTable.getRowSorter().toggleSortOrder(0);
83
84      /**
85      * get all expenses data from DB
86      */
87      private void getData() {
88          Runnable run = () -> {
89              // variables to contain all the information the SQL query

```

```

86     return from DB
87         ResultSet rs;
88         String date;
89         String cost;
90         String classification;
91         String details;
92
93         logger.info("TableData.getData started");
94         model.setRowCount(0);
95         try
96     {
97             rs = viewModel.getAllExpenses();
98             while (rs.next()) {
99                 // final variables which would be used in the
100                GUI creation task (GUI thread) who updates the table information
101                String finalClassification, finalDetails,
102                finalDate, finalCost;
103
104                date = rs.getString("Date");
105                LocalDate localDate = LocalDate.parse(date,
106                    DateTimeFormatter.ofPattern("dd/MM/yyyy", Locale.getDefault()));
107                cost = rs.getString("Cost");
108                classification = rs.getString("Classification");
109                details = rs.getString("Details");
110                finalDate = localDate.format(DateTimeFormatter.
111                    ofPattern("dd/MM/yyyy"));
112                finalCost = cost;
113                finalClassification = classification;
114                finalDetails = details;
115
116                SwingUtilities.invokeLater(new Runnable() {
117                    @Override
118                    public void run() {
119                        Object[] row = new Object[]{finalDate,
120                            finalClassification, finalDetails, finalCost};
121                        model.insertRow(dataTable.getRowCount
122                            (), row);
123                    }
124                });
125            }
126            logger.info("TableData.getData ended successfully");
127        } catch (SQLException | GetAllUserExpensesException e)
128        {
129            logger.info("TableData.getData failed because" + e.
130            getMessage());
131            e.printStackTrace();
132        }
133    }
134}

```

```
125         }
126     };
127
128     SwingUtilities.invokeLater(run);
129 }
130
131 /**
132  * refresh table data from DB
133 */
134 private void refreshTable() {
135     getData();
136 }
137
138 public JScrollPane getUpdatedTableJScrollPane() {
139     refreshTable();
140     return tableJScrollPane;
141 }
142
143 private static void setLogger() {
144     TableData.logger = Logger.getLogger(TableData.class);
145 }
146
147 private void setViewModel(){
148     viewModel = ManagementViewModel.getInstance();
149 }
150 }
151
```

```

1 package il.ac.hit.CashFlowManagement.viewmodel;
2
3 import il.ac.hit.CashFlowManagement.exception.FormCastingException;
4 import il.ac.hit.CashFlowManagement.exception.
5 GetAllUserExpensesException;
6 import org.jetbrains.annotations.NotNull;
7
8
9 /**
10 * this interface define the operations that each Form/Frame must
11 * implement
12 */
13 public interface IViewModel {
14
15     /**
16      * add viewModel to all forms and Show the dialog of the
17      LoginForm
18      */
19     void startProgram();
20
21     /**
22      * The method verifies that the Login information (username and
23      password) is correct
24      * @param iUserName the username of the user who want to login,
25      can't be null
26      * @param iPassword the password of the user who want to login,
27      can't be null
28      * @return true either false if the Login information correct
29      */
30     boolean verifyUser(@NotNull String iUserName, @NotNull String
31     iPassword);
32
33     /**
34      * The method check if the user exist and if not created a new
35      user
36      * @param iUserName the new user username, can't be null
37      * @param iPassword the new user password, can't be null
38      * @param iCountry the new user country, can't be null
39      * @param iGander the new user gander, can't be null
40      * @return true if new user registered or false if the user
41      already exists
42      */
43     boolean register(@NotNull String iUserName, @NotNull String
44     iPassword, @NotNull String iCountry, @NotNull String iGander);
45
46

```

```
37     /**
38      * Add a new expense to the data base
39      * @throws FormCastingException if failed to add new expense
40      * because of IForm to MainForm casting problem or parse from cost which
41      * is not double
42      */
43     void addNewExpense() throws FormCastingException;
44
45     /**
46      * Show the dialog of the MainForm
47      */
48     void showMainForm();
49
50     /**
51      * Show the dialog of the RegisterForm
52      */
53     void showRegisterForm();
54
55     /**
56      * Show the dialog of the LoginForm
57      */
58     void showLoginForm();
59
60     /**
61      * get all rhe expenses of a specific user
62      * @return Result set that represent the table with all the
63      * expenses of a specific user
64      * @throws GetAllUserExpensesException if there is a problem in
65      * getting the information
66     */
67     ResultSet getAllExpenses() throws GetAllUserExpensesException;
68 }
```

```

1 package il.ac.hit.CashFlowManagement.viewmodel;
2
3 import il.ac.hit.CashFlowManagement.exception.FormCastingException;
4 import il.ac.hit.CashFlowManagement.exception.
5     GetAllUserExpensesException;
6 import il.ac.hit.CashFlowManagement.model.Expense;
7 import il.ac.hit.CashFlowManagement.model.ExpensesLogic;
8 import il.ac.hit.CashFlowManagement.model.UserLogic;
9 import il.ac.hit.CashFlowManagement.view.IView;
10 import il.ac.hit.CashFlowManagement.view.LoginForm;
11 import il.ac.hit.CashFlowManagement.view.MainForm;
12 import il.ac.hit.CashFlowManagement.view.RegisterForm;
13 import org.apache.log4j.BasicConfigurator;
14 import org.apache.log4j.Logger;
15 import org.jetbrains.annotations.NotNull;
16
17 import javax.swing.*;
18 import java.sql.ResultSet;
19 import java.util.Arrays;
20
21 /**
22 * The Management View Model class
23 */
24 public class ManagementViewModel implements IViewModel{
25     private static Logger logger;
26     private static ManagementViewModel instance = null;
27     private static Object lock = new Object();
28     private IView loginForm, mainForm, registerForm;
29     private ExpensesLogic expensesLogic;
30     private UserLogic userLogic;
31     private String newExpenseDay, newExpenseMonth, newExpenseYear,
32     newExpenseClassification, newExpenseDetail, newExpenseCost;
33
34     /**
35      * ManagementViewModel C'tor
36     */
37     private ManagementViewModel() {
38         BasicConfigurator.configure();
39         setLogger(Logger.getLogger(ManagementViewModel.class));
40         setMainForm(new MainForm());
41         setLoginForm(new LoginForm());
42         setRegisterForm(new RegisterForm());
43         setUserLogic(new UserLogic());
44         logger.info("ManagementViewModel was created successfully");
45     }

```

```

45
46     /**
47      * check if there is a singleton instance of ManagementViewModel
48      * and create it's if needed
49      */
50     public static ManagementViewModel getInstance() {
51         if (instance == null) {
52             synchronized (lock) {
53                 if (instance == null) {
54                     instance = new ManagementViewModel();
55                 }
56             }
57         }
58
59         return instance;
60     }
61
62     /**
63      * @see IViewModel {@Link #startProgram()}
64      */
65     public void startProgram() {
66         logger.info("ManagementViewModel.startProgram called");
67         loginForm.setViewModel();
68         registerForm.setViewModel();
69         mainForm.setViewModel();
70
71         SwingUtilities.invokeLater(new Runnable() {
72             @Override
73             public void run() {
74                 loginForm.showForm();
75             }
76         });
77     }
78
79     /**
80      * @see IViewModel {@Link #showLoginForm()}
81      */
82     public void showLoginForm() {
83         logger.info("ManagementViewModel.LoginForm called");
84         SwingUtilities.invokeLater(new Runnable() {
85             @Override
86             public void run() {
87                 loginForm.showForm();
88             }
89         });

```

```
90     }
91
92     /**
93      * see IViewModel {@Link #showRegisterForm()}
94     */
95     public void showRegisterForm() {
96         logger.info("ManagementViewModel.showRegisterForm called");
97         SwingUtilities.invokeLater(new Runnable() {
98             @Override
99             public void run() {
100                 registerForm.showForm();
101             }
102         });
103     }
104
105    /**
106     * see IViewModel {@Link #showMainForm()}
107    */
108    public void showMainForm(){
109        logger.info("ManagementViewModel.showMainForm called");
110        SwingUtilities.invokeLater(new Runnable() {
111            @Override
112            public void run() {
113                setExpensesLogic(new ExpensesLogic());
114                mainForm.showForm();
115            }
116        });
117    }
118
119    /**
120     * see IViewModel {@Link #verifyUser(String, String)}
121    */
122    public boolean verifyUser(@NotNull String iUserName, @NotNull
123     String iPassword) {
124        Boolean valueToReturn = false;
125
126        logger.info("ManagementViewModel.verifyUser started");
127        if(getUserLogic().checkUserPassword(iUserName, iPassword))
128        {
129            valueToReturn = true;
130        }
131
132        logger.info("ManagementViewModel.verifyUser ended");
133
134        return valueToReturn;
135    }
```

```

135
136     /**
137      * see IViewModel {@Link #register(String, String, String,
138      * String)}
139
140     public boolean register(@NotNull String iUserName, @NotNull
141     String iPassword, @NotNull String iCountry, @NotNull String iGander
142     ) {
143         boolean newUserRegister = false;
144
145         logger.info("ManagementViewModel.register started");
146         if (!getUserLogic().checkIfExist(iUserName)) {
147             getUserLogic().addUser(iUserName, iPassword, iCountry,
148             iGander);
149             logger.info("new user register successfully");
150             newUserRegister = true;
151         }
152         else{
153             logger.info("new user registering failed since the user
154             " + iUserName + " is already exist");
155         }
156
157         return newUserRegister;
158     }
159
160     /**
161      * see IViewModel {@Link #getAllExpenses()}
162      */
163
164     public ResultSet getAllExpenses() throws
165     GetAllUserExpensesException {
166         return expensesLogic.getAllUserExpenses();
167     }
168
169     /**
170      * The method updates the new expense data members from the
MainForm
171      * @throws FormCastingException if the casting failed
172      */
173
174     private void getUpdatedNewExpenseInformation() throws
175     FormCastingException {
176         logger.info("ManagementViewModel.
177         getUpdatedNewExpenseInformation started");
178         try{
179             MainForm form = (MainForm) mainForm;
180             setNewExpenseDay(form.getDaysComboBox().getSelectedItem
181             ().toString());

```

```

171         setNewExpenseMonth(form.getMonthsComboBox() .
172             getSelectedItem().toString());
173         setNewExpenseYear(form.getYearsComboBox() .
174             getSelectedItem().toString());
175         setNewExpenseClassification(form.
176             getClassificationComboBox().getSelectedItem().toString());
177         setNewExpenseDetail(form.getDetailTextField().getText
178             ());
179         setNewExpenseCost(form.getCostTextField().getText());
180     } catch (ClassCastException e){
181         logger.info("casting from IForm to MainForm didn't
182 succeed, therefor the expense data members update failed");
183         throw new FormCastingException("casting from IForm to
184 MainForm didn't succeed, therefor the expense data members update
185 failed", e);
186     }
187 }
188 /**
189 * The method verifies a new expense if all the parameters is
190 correct
191 */
192 private boolean correctInfoForNewExpense() throws
193 FormCastingException {
194     String day, month, year, classification, detail, cost;
195     boolean correctInformation;
196
197     logger.info("ManagementViewModel.correctInfoForNewExpense
198 started");
199     getUpdatedNewExpenseInformation();
200     day = getNewExpenseDay();
201     month = getNewExpenseMonth();
202     year = getNewExpenseYear();
203     classification = getNewExpenseClassification();
204     detail = getNewExpenseDetail();
205     cost = getNewExpenseCost();
206
207     correctInformation = !day.equalsIgnoreCase("") && !month.
208         equalsIgnoreCase("") && !year.equalsIgnoreCase("")
209             && !classification.equalsIgnoreCase("") && !detail.
210             equalsIgnoreCase("") && !cost.equalsIgnoreCase(""));
211
212     try{
213         if (!correctInformation)
214         {
215             StringBuilder message = new StringBuilder("Please ")
216         );

```

```

204
205             if(day.equalsIgnoreCase("")) message.append("select
206                 day");
207             else if(month.equalsIgnoreCase("")) message.append("select month");
208             else if(year.equalsIgnoreCase("")) message.append("select year");
209             else if(classification.equalsIgnoreCase("")) message.append("select classification");
210             else if(detail.equalsIgnoreCase("")) message.append("add some detail");
211             else if(cost.equalsIgnoreCase("")) message.append("fill cost");
212         JOptionPane.showMessageDialog(null, message);
213     }
214     Double.parseDouble(cost);
215 } catch (NumberFormatException e){
216     JOptionPane.showMessageDialog(null, "Cost must be only
217     number");
218 }
219 finally {
220     logger.info("ManagementViewModel.
221     correctInfoForNewExpense ended successfully");
222     return correctInformation;
223 }
224
225 /**
226 * see IViewModel {@Link #addNewExpense()}
227 */
228 public void addNewExpense() throws FormCastingException {
229     try {
230         MainForm form = (MainForm) mainForm;
231
232         if (ManagementViewModel.getInstance().
233             correctInfoForNewExpense()) {
234             double cost = Double.parseDouble(getNewExpenseCost
235             ());
236             String dateStr = getNewExpenseDay() + "/" + Arrays.
237             asList(form.getMonths()).indexOf(getNewExpenseMonth()) + "/" +
238             getNewExpenseYear();
239
240             Expense expense = new Expense(dateStr,
241             getNewExpenseClassification(), getNewExpenseDetail(), cost);

```

```
237         expensesLogic.addExpense(expense);
238     }
239     logger.info("ManagementViewModel.addNewExpense ended
240 successfully, new expense added");
240     } catch (ClassCastException e){
241         throw new FormCastingException("ManagementViewModel.
241 addNewExpense failed, IForm to MainForm casting problem",e);
242     }
243     catch (NumberFormatException ex){
244         throw new FormCastingException("ManagementViewModel.
244 addNewExpense failed, parse from cost which is not double", ex);
245     }
246 }
247
248 public void setLoginForm(LoginForm loginForm) {
249     this.loginForm = loginForm;
250 }
251
252 public void setMainForm(MainForm mainForm) {
253     this.mainForm = mainForm;
254 }
255
256 public void setRegisterForm(RegisterForm registerForm) {
257     this.registerForm = registerForm;
258 }
259
260 public void setUserLogic(UserLogic userLogic) {
261     this.userLogic = userLogic;
262 }
263
264 public static void setLogger(Logger logger) {
265     ManagementViewModel.logger = logger;
266 }
267
268 public static void setInstance(ManagementViewModel instance) {
269     ManagementViewModel.instance = instance;
270 }
271
272 public static void setLock(Object lock) {
273     ManagementViewModel.lock = lock;
274 }
275
276 public void setExpensesLogic(ExpensesLogic expensesLogic) {
277     this.expensesLogic = expensesLogic;
278 }
279
```

```
280     public UserLogic getUserLogic() {
281         return userLogic;
282     }
283
284     public String getNewExpenseDay() {
285         return newExpenseDay;
286     }
287
288     public void setNewExpenseDay(String newExpenseDay) {
289         this.newExpenseDay = newExpenseDay;
290     }
291
292     public String getNewExpenseMonth() {
293         return newExpenseMonth;
294     }
295
296     public void setNewExpenseMonth(String newExpenseMonth) {
297         this.newExpenseMonth = newExpenseMonth;
298     }
299
300     public String getNewExpenseYear() {
301         return newExpenseYear;
302     }
303
304     public void setNewExpenseYear(String newExpenseYear) {
305         this.newExpenseYear = newExpenseYear;
306     }
307
308     public String getNewExpenseClassification() {
309         return newExpenseClassification;
310     }
311
312     public void setNewExpenseClassification(String
newExpenseClassification) {
313         this.newExpenseClassification = newExpenseClassification;
314     }
315
316     public String getNewExpenseDetail() {
317         return newExpenseDetail;
318     }
319
320     public void setNewExpenseDetail(String newExpenseDetail) {
321         this.newExpenseDetail = newExpenseDetail;
322     }
323
324     public String getNewExpenseCost() {
```

```
325         return newExpenseCost;
326     }
327
328     public void setNewExpenseCost(String newExpenseCost) {
329         this.newExpenseCost = newExpenseCost;
330     }
331 }
```