File - C:\Users\sagis\IdeaProjects\CashFlowManagement\src\il\ac\hit\CashFlowManagement\Program.java

```java
package il.ac.hit.CashFlowManagement;

import il.ac.hit.CashFlowManagement.viewmodel.IViewModel;
import il.ac.hit.CashFlowManagement.viewmodel.ManagementViewModel;

/**
 * The program class
 * @author Moshiko Davila 308459841
 * @author Sagi Sela 201325560
 * @version March 24 , 2020.
 */
public class Program {
    public static void main(String[] args)
    {
        IViewModel viewModel = ManagementViewModel.getInstance();

        viewModel.startProgram();
    }
}
```

```java
package il.ac.hit.CashFlowManagement.view;

import org.jetbrains.annotations.NotNull;

import java.awt.*;

/**
 * this interface define the operations that each Form/Frame must implement
 */
public interface IView {

    /**
     * Show dialog of the form
     */
    void showForm();

    /**
     * Close the dialog of the form
     * @param form the form who should be closed, can't be null
     */
    default void closeForm(@NotNull Window form){
        form.setVisible(false);
        form.dispose();
    }

    /**
     * set the form view model to the ManagementViewModel instance
     */
    void setViewModel();
}

```

```java
package il.ac.hit.CashFlowManagement.view;

import il.ac.hit.CashFlowManagement.exception.FormCastingException;
import il.ac.hit.CashFlowManagement.viewmodel.IViewModel;
import il.ac.hit.CashFlowManagement.viewmodel.ManagementViewModel;
import il.ac.hit.CashFlowManagement.viewmodel.TableData;
import org.apache.log4j.Logger;

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

/**
 * The main form class
 */
public class MainForm extends JFrame implements ActionListener, IView {
    private static Logger logger = Logger.getLogger(MainForm.class);
    private IViewModel viewModel;

    //creating dates scrollable strings
    private String[] months = {"", "Jan", "Feb", "Mar", "Apr", "May", "Jun", "Jul", "Aug", "Sept", "Oct", "Nov", "Dec"};
    private String[] days = {"", "1", "2", "3", "4", "5",
            "6", "7", "8", "9", "10",
            "11", "12", "13", "14", "15",
            "16", "17", "18", "19", "20",
            "21", "22", "23", "24", "25",
            "26", "27", "28", "29", "30",
            "31"};
    private String years[]
            = {"", "1995", "1996", "1997", "1998",
            "1999", "2000", "2001", "2002",
            "2003", "2004", "2005", "2006",
            "2007", "2008", "2009", "2010",
            "2011", "2012", "2013", "2014",
            "2015", "2016", "2017", "2018",
            "2019", "2020"};
    private String classification[] = {"", "Assets", "Clothing", "Culture", "Education", "Environment", "Entertainment", "Food", "Footwear", "Health, Absorb.& Religion"};

    //creating gui components
    private Container container = getContentPane();
    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(
    classification);
46      private DesignedJLable titleLabel = new DesignedJLable("Main Form
    ");
47      private DesignedJLable newExpenseLabel = new DesignedJLable("New
    Expense:".toUpperCase());
48      private DesignedJLable dateLabel = new DesignedJLable("DATE");
49      private DesignedJLable classificationLabel = new DesignedJLable("
    CLASSIFICATION");
50      private DesignedJLable detailLabel = new DesignedJLable("DETAIL"
    );
51      private DesignedJLable costLabel = new DesignedJLable("COST");
52      private DesignedJButton addBtn = new DesignedJButton("Add");
53      private JTextField detailTextField = new JTextField();
54      private JTextField costTextField = new JTextField();
55      private DesignedJButton refreshTable = new DesignedJButton("
    Refresh Table");
56      private JScrollPane tableJScrollPane = 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          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         tableJScrollPane.setBounds(50, 320, 900, 350);
206         container.add(tableJScrollPane, SwingConstants.CENTER);
207     }
208 }
209
210
211
```

```java
package il.ac.hit.CashFlowManagement.view;

import il.ac.hit.CashFlowManagement.viewmodel.IViewModel;
import il.ac.hit.CashFlowManagement.viewmodel.ManagementViewModel;
import org.apache.log4j.Logger;

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

/**
 * The Login form class
 */
public class LoginForm extends JFrame implements ActionListener, IView {
    private static Logger logger = Logger.getLogger(LoginForm.class);
    public static String username;
    private IViewModel viewModel;
    //creating gui components
    private Container container = getContentPane();
    private DesignedJLable titleLabel = new  DesignedJLable("Login");
    private DesignedJLable userLabel = new DesignedJLable("USERNAME");
    private DesignedJLable passwordLabel = new DesignedJLable("PASSWORD");
    private JTextField userTextField = new JTextField();
    private JPasswordField passwordField = new JPasswordField();
    private DesignedJButton loginButton = new DesignedJButton("LOGIN");
    private DesignedJButton resetButton = new DesignedJButton("RESET");
    private DesignedJButton registerButton = new DesignedJButton("REGISTER");
    private JCheckBox showPassword = new JCheckBox("Show Password");
    private JLabel lblBackground;

    /**
     * LoginForm C'tor
     */
    public LoginForm()
    {
        //Calling setLayoutManger() method inside the constructor.
        setLayoutManager();
        setLocationAndSize();
        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
```

```java
package il.ac.hit.CashFlowManagement.view;

import il.ac.hit.CashFlowManagement.viewmodel.IViewModel;
import il.ac.hit.CashFlowManagement.viewmodel.ManagementViewModel;
import org.apache.log4j.Logger;

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



/**
 * The register form class
 */

public class RegisterForm extends JFrame implements ActionListener, IView
{
    private static Logger logger = Logger.getLogger(RegisterForm.class);
    private IViewModel viewModel;
    //creating gui components
    private Container container = getContentPane();
    private String[] country={"","INDIA","AMERICA","AUSTRALIA","PHILIPPINES","SPAIN","ISRAEL" };
    private DesignedJLable titleLabel = new  DesignedJLable("Registration Form");
    private DesignedJLable userLabel=new  DesignedJLable("USERNAME");
    private DesignedJLable passwordLabel=new  DesignedJLable("PASSWORD");
    private DesignedJLable confirmPasswordLabel=new DesignedJLable("CONFIRM PASSWORD");
    private DesignedJLable countryLabel=new  DesignedJLable("SELECT COUNTRY");
    private DesignedJLable genderLabel=new  DesignedJLable("GENDER");
    private JCheckBox maleGender=new JCheckBox("MALE");
    private JCheckBox femaleGender=new JCheckBox("FEMALE");
    private ButtonGroup bg=new ButtonGroup();
    private JComboBox countryComboBox=new JComboBox(country);
    private DesignedJButton submitButton=new DesignedJButton("SUBMIT");
    private DesignedJButton resetButton=new DesignedJButton("RESET");
    private JTextField userTextField = new JTextField();
    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.
    equalsIgnoreCase(cPwd) && !userName.equalsIgnoreCase("") && userName
    .chars().allMatch(Character::isLetter) && (maleGender.isSelected
    () || femaleGender.isSelected()) && country != "";
149             if (correctInformation) {
150                 if(ManagementViewModel.getInstance().register(
    userName, pwd, country, gender)){
151                     if (e.getSource() == submitButton) {
152                         JOptionPane.showMessageDialog(null, "Data
    Registered Successfully");
153                         viewModel.showLoginForm();
154                         closeForm(this);
155                     }
156                 }
157                 else{
158                     JOptionPane.showMessageDialog(null, "This
    username is already taken");
159                 }
160             }
161             else if(userName.equalsIgnoreCase("") || !userName.chars
    ().allMatch(Character::isLetter)){
162                 if(userName.equalsIgnoreCase("")){
163                     JOptionPane.showMessageDialog(null, "Username
    can't be empty");
164                 }
165                 else{
166                     JOptionPane.showMessageDialog(null, "UserName
    must contain only letters");
167                 }
```

```
168             }
169             else if(!(maleGender.isSelected() || femaleGender.
    isSelected())){
170                 JOptionPane.showMessageDialog(null, "Please select
    gender");
171             }
172             else if(!(country != "")){
173                 JOptionPane.showMessageDialog(null, "Please select
    country");
174             }
175             else {
176                 if(pwd.length() < 4) {
177                     JOptionPane.showMessageDialog(null, "Password
    must include minimum 4 characters");
178                 }
179                 else {
180                     JOptionPane.showMessageDialog(null, "Password
    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
```

```java
package il.ac.hit.CashFlowManagement.view;

import javax.swing.*;
/**
 * The  Designed JLable class
 */
public class DesignedJLable extends JLabel {
    public DesignedJLable (String text){
        this.setText(text);
        this.setBounds(50, 50,100, 30);
//        setOpaque(true);
//        this.setBackground(Color.black);
//        this.setFont(new Font("Verdana", Font.BOLD, 12));
//        this.setForeground(Color.black);
    }
}

```

```java
package il.ac.hit.CashFlowManagement.view;

import javax.swing.*;
/**
 * The  Designed JButton class
 */

public class DesignedJButton extends JButton {
    public DesignedJButton (String text){
        this.setText(text);
        this.setSize(100, 50);
//        this.setBackground(Color.cyan);
//        this.setFont(new Font("Verdana", Font.BOLD, 12));
//        this.setForeground(Color.black);
    }
}

```

```java
package il.ac.hit.CashFlowManagement.model;

import org.jetbrains.annotations.NotNull;

import java.text.ParseException;
import java.text.SimpleDateFormat;
import java.util.Date;

/**
 *  The Expense class
 */
public class Expense
{
    private Date date;
    private String classification, details;
    private double cost;
    private SimpleDateFormat simpleDateFormat = new SimpleDateFormat("dd/MM/yyyy");

    /**
     * Expense C'tor
     * @param iDateStr Date of new expense, can't be null
     * @param iClassification Classification of new expense, can't be null
     * @param iDetails New expense details, can't be null
     * @param iCost New expense cost, can't be null
     */
    public Expense(@NotNull String iDateStr, @NotNull String iClassification, @NotNull String iDetails, @NotNull double iCost) {
        try{
            setDate(simpleDateFormat.parse(iDateStr));
            setClassification(iClassification);
            setDetails(iDetails);
            setCost(iCost);
        } catch(ParseException pe) {
            throw new IllegalArgumentException(pe);
        }
    }

    public String getClassification() {
        return classification;
    }

    public void setClassification(String classification) {
        this.classification = classification;
    }
```

```
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
```

```java
package il.ac.hit.CashFlowManagement.model;

import il.ac.hit.CashFlowManagement.exception.InsertToDBException;
import il.ac.hit.CashFlowManagement.exception.
JDBCDataBaseLogicException;
import org.apache.log4j.Logger;
import org.jetbrains.annotations.NotNull;

import java.sql.ResultSet;


/**
 * The  User Logic class
 */
public class UserLogic implements IUserLogic
{
    private static Logger logger;
    private JDBCDataBaseLogic dataBaseLogic;
    private final String nameOfTable = "Registered_Users";

    /**
     * UserLogic C'tor
     */
    public UserLogic()
    {
        setLogger(Logger.getLogger(UserLogic.class));
        setDataBaseLogic(JDBCDataBaseLogic.getInstance());
        try {
            dataBaseLogic.createTableIfNotExist(
                    nameOfTable,
                    "UserName",
                    "varchar(300)",
                    "Password",
                    "varchar(300)",
                    "Country",
                    "varchar(300)",
                    "Gander",
                    "varchar(300)"
            );
            logger.info("UserLogic was create successfully!!");
        } catch (JDBCDataBaseLogicException e) {
            logger.info("failed to create UserLogic");
            e.printStackTrace();
        }
    }

```

```
46      /**
47       * @see IUserLogic {@link #addUser(String iUserName, String
    iPassword, String iCountry, String iGander)}
48       */
49      public void addUser(@NotNull String iUserName, @NotNull String
    iPassword, @NotNull String iCountry, @NotNull String iGander) {
50          String parameters =
51  //                  "(" +
52                  "'" + iUserName + "'"
53                  + "," + "'" + iPassword + "'"
54                  + "," + "'" + iCountry + "'"
55                  + "," + "'" + iGander + "'"
56  //                + ")"
57                  ;
58
59          try {
60              dataBaseLogic.insertValue(
61                      nameOfTable,
62                      parameters);
63              logger.info("successfully registered new user in DB");
64          } catch (InsertToDBException e) {
65              logger.info("failed to register new user in DB");
66              e.printStackTrace();
67          }
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 " +
    nameOfTable + " where " + nameOfTable + ".USERNAME ='" + iUserName +
    "'");
78
79              if (rs.next()) {
80                  valToReturn = true;
81                  logger.info("user exists in DB");
82              }
83              else{
84                  logger.info("user doesn't exists in DB");
85              }
86          } catch (JDBCDataBaseLogicException e) {
87              logger.info("checkIfExist method failed");
```

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

```
128     }
129 }
```

```java
package il.ac.hit.CashFlowManagement.model;

import org.jetbrains.annotations.NotNull;

/**
 * this interface define the operations that the UserLogic must implement
 */
public interface IUserLogic {
    /**
     * check if user exist in the Registered_Users table
     * @param iUserName the new user username, can't be null
     * @return true either false if the username exists in the DB
     */
    boolean checkIfExist(@NotNull String iUserName);

    /** validate user login information (username and password match) in the Registered_Users table
     * @param iUserName the username of the user who want to login, can't be null
     * @param iPassword the password of the user who want to login, can't be null
     * @return true either false if the login information is correct
     */
    boolean checkUserPassword(@NotNull String iUserName, @NotNull String iPassword);

    /** add user to the Registered_Users table in the DB
     * @param iUserName the new user username, can't be null
     * @param iPassword the new user password, can't be null
     * @param iCountry the new user country, can't be null
     * @param iGander the new user gander, can't be null
     */
    void addUser(@NotNull String iUserName, @NotNull String iPassword, @NotNull String iCountry, @NotNull String iGander);
}

```

```java
package il.ac.hit.CashFlowManagement.model;

import il.ac.hit.CashFlowManagement.exception.JDBCDataBaseException;
import org.apache.log4j.Logger;

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;
import java.sql.Statement;



/**
 * The DataBase class
 */
public class JDBCDataBase
{
    private static JDBCDataBase instance = null;
    private static Object lock = new Object();
    private static Logger logger;
    private final String driver = "org.apache.derby.jdbc.EmbeddedDriver";
    private final String protocol = "jdbc:derby:CashFlowManagementDB;create=true";
    private boolean isConnected;
    private Connection connection = null;
    private Statement statement = null;

    private JDBCDataBase() throws JDBCDataBaseException {
        try {
            setLogger(Logger.getLogger(JDBCDataBase.class));
            setConnected(false);
            init();
            logger.info("DataBase was create successfully!!");
        } catch (JDBCDataBaseException e){
            logger.info("DataBase creation failed!!");
            throw new JDBCDataBaseException("DataBase creation problem " + e.getMessage(), e);
        }
    }

    /**
     * This method Describes Connection to DataBase
     * @return Connection to DataBase , m_IsConnected
     */
    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 JDBCDataBaseException("Connection problem" , e
    );
51          } catch (ClassNotFoundException ex){
52              throw new JDBCDataBaseException("Driver Class not found"
     , ex);
53          }
54          finally {
55              return isConnected;
56          }
57      }
58
59      /**
60       * check if there is a singleton instance of JDBCDataBase and
    create it's if needed
61       * @return instance of DataBase
62       * @throws JDBCDataBaseException if failed to get create DB
    instance
63       */
64      public static JDBCDataBase getInstance() throws
    JDBCDataBaseException
65      {
66          if(instance == null)
67          {
68              synchronized (lock)
69              {
70                  if (instance == null)
71                  {
72                      instance = new JDBCDataBase();
73                  }
74              }
75          }
76
77          return instance;
78      }
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
```

```java
package il.ac.hit.CashFlowManagement.model;

import il.ac.hit.CashFlowManagement.exception.
GetAllUserExpensesException;
import il.ac.hit.CashFlowManagement.exception.InsertToDBException;
import il.ac.hit.CashFlowManagement.exception.
JDBCDataBaseLogicException;
import il.ac.hit.CashFlowManagement.view.LoginForm;
import org.apache.log4j.Logger;
import org.jetbrains.annotations.NotNull;

import java.sql.ResultSet;
import java.text.SimpleDateFormat;

/**
 * The  Expenses Logic class
 */
public class ExpensesLogic implements IExpenseHandle
{
    private static Logger logger;
    private final String nameOfTable;
    private JDBCDataBaseLogic dataBase;
    private SimpleDateFormat formatter;

    /**
     * ExpensesLogic C'tor
     */
    public ExpensesLogic() {
        setLogger(Logger.getLogger(ExpensesLogic.class));
        nameOfTable = LoginForm.username.toUpperCase() + "_Expenses".
toUpperCase();
        setFormatter(new SimpleDateFormat("dd/MM/yyyy"));
        setDataBase(JDBCDataBaseLogic.getInstance());
        try
        {
            dataBase.createTableIfNotExist(
                    nameOfTable,
                    "Date",
                    "varchar(300)",
                    "Classification",
                    "varchar(300)",
                    "Details",
                    "varchar(300)",
                    "Cost",
                    "varchar(300)");
            logger.info("ExpensesLogic was create successfully!!!");
```

```
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              dataBase.insertValue(nameOfTable, parameters);
63              logger.info("ExpensesLogic.addExpense finished
    successfully");
64          } catch (InsertToDBException e) {
65              logger.info("ExpensesLogic.addExpense failed");
66              e.printStackTrace();
67          }
68      }
69
70      /**
71       * @see ExpensesLogic {@Link #getAllUserExpenses()}
72       */
73      public ResultSet getAllUserExpenses() throws
    GetAllUserExpensesException {
74          try {
75              return dataBase.query("select * from " + nameOfTable);
76          } catch (JDBCDataBaseLogicException e){
77              throw new GetAllUserExpensesException("failed to get all
    users expenses", e);
78          }
79      }
80
81      public static void setLogger(Logger logger) {
82          ExpensesLogic.logger = logger;
83      }
84
```

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

```java
package il.ac.hit.CashFlowManagement.model;

import il.ac.hit.CashFlowManagement.exception.GetAllUserExpensesException;
import il.ac.hit.CashFlowManagement.exception.InsertToDBException;
import org.jetbrains.annotations.NotNull;

import java.sql.ResultSet;

/**
 * this interface define the operations each ExpensesLogic must implement
 */
public interface IExpenseHandle {
    /**
     * add an expense to the user expenses in the DB
     * @param expense full expense information, can't be null
     * @throws InsertToDBException throws exception if failed to insert the new expenses to the DB
     */
    void addExpense(@NotNull Expense expense) throws InsertToDBException;

    /**
     * get all user expenses from the the DB
     * @return ResultSet object that contains full information about all user expenses from DB
     * @throws GetAllUserExpensesException throws exception if failed to get all users expenses
     */
    ResultSet getAllUserExpenses() throws GetAllUserExpensesException;
}
```

```java
package il.ac.hit.CashFlowManagement.model;

import il.ac.hit.CashFlowManagement.exception.InsertToDBException;
import il.ac.hit.CashFlowManagement.exception.JDBCDataBaseException;
import il.ac.hit.CashFlowManagement.exception.
JDBCDataBaseLogicException;
import org.apache.log4j.Logger;
import org.jetbrains.annotations.NotNull;

import java.sql.DatabaseMetaData;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;

/**
 * The  Data Base Logic class
 */
public class JDBCDataBaseLogic implements IJDBCDataBaseLogic {
    private static JDBCDataBaseLogic instance = null;
    private static Object lock = new Object();
    private static Logger logger;
    private JDBCDataBase dataBase = null;
    private Statement statement = null;
    private DatabaseMetaData metaData = null;

    private JDBCDataBaseLogic() {
        try {
            setLogger(Logger.getLogger(JDBCDataBaseLogic.class));
            setDataBase(JDBCDataBase.getInstance());
            setMetaData(dataBase.getConnection().getMetaData());
            setStatement(dataBase.getStatement());
            logger.info("DataBaseLogic was create successfully!!!");
        } catch (SQLException | JDBCDataBaseException e) {
            e.printStackTrace();
        }
    }

    /**
     * check if there is a singleton instance of DataBaseLogic and
create it's if needed
     * @return instance of DataBaseLogic
     */
    public static JDBCDataBaseLogic getInstance() {
        if (instance == null) {
            synchronized (lock) {
                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
   created
55       * @see IJDBCDataBaseLogic {@Link #createTableIfNotExist(String,
   String...)}
56       */
57      public boolean createTableIfNotExist(@NotNull String iNameOfTable
   , @NotNull String... iParameters) throws JDBCDataBaseLogicException {
58          boolean newTableCreated = false;
59
60          if (iParameters.length % 2 != 0) {
61              logger.info("Create Table was failed, params is not even"
   );
62              throw new JDBCDataBaseLogicException("Please enter the
   parameters as follows: Param1Name ,Param1Type, Param2Name ,Param2Type
    .... , ParamName ,ParamType");
63          }
64
65          iNameOfTable = iNameOfTable.toUpperCase();
66          String str = stringsConcat(iParameters);
67          try {
68              ResultSet rs = metaData.getTables(null, "APP",
   iNameOfTable, null);
69              if (!rs.next()) {
70                  String sqlStatement = "create table " + iNameOfTable
    + str;
71                  statement.execute(sqlStatement);
72                  newTableCreated = true;
73                  logger.info("New table created successfully, new
   table name is " + iNameOfTable);
74              }
75              else{
76                  newTableCreated = false;
77                  logger.info(iNameOfTable + " table already exist and
   therefor new table wasn't created");
78              }
79          } catch (SQLException e) {
80              logger.info("New table creation failed");
```

```
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 setDataBase(JDBCDataBase dataBase) {
166         this.dataBase = dataBase;
167     }
168
169     public void setStatement(Statement statement) {
170         this.statement = statement;
171     }
172
173     public void setMetaData(DatabaseMetaData metaData) {
174         this.metaData = metaData;
175     }
176
177     public static void setLogger(Logger logger) {
178         JDBCDataBaseLogic.logger = logger;
179     }
180 }
```

```java
package il.ac.hit.CashFlowManagement.model;

import il.ac.hit.CashFlowManagement.exception.InsertToDBException;
import il.ac.hit.CashFlowManagement.exception.
JDBCDataBaseLogicException;
import org.jetbrains.annotations.NotNull;

import java.sql.ResultSet;

/**
 * this interface define the operations that the JDBCDataBaseLogic
must implement
 */
public interface IJDBCDataBaseLogic<T>
{
    /**
     * @param nameOfTable name of the table
     * @param parameters parameter to Create
     * @return true if a new table created, false if the table is
already exists
     * @throws JDBCDataBaseLogicException if failed to create table
because params is not even or SQLException
     */
    boolean createTableIfNotExist(@NotNull String nameOfTable, @
NotNull String... parameters) throws JDBCDataBaseLogicException;

    /**
     * Remove table from DB query (DROP TABLE SQL query)
     * @param nameOfTable table name to remove, can't be null
     * @throws JDBCDataBaseLogicException if failed to remove table
because of SQLException
     */
    void removeTable(@NotNull String nameOfTable) throws
JDBCDataBaseLogicException;

    /**
     * Insert a values into a table in the DB
     * @param nameOfTable name of the table, can't be null
     * @param parameters row parameters to insert, can't be null
     * @throws InsertToDBException if failed to insert value to table
 because of SQLException
     */
    void insertValue(@NotNull String nameOfTable, String parameters)
throws InsertToDBException;

    /**
```

```
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
    query executing
41      * @throws JDBCDataBaseLogicException if failed to execute query
    because of SQLException
42      */
43     ResultSet query(@NotNull String query) throws
    JDBCDataBaseLogicException;
44 }
45
46
47
```

```java
package il.ac.hit.CashFlowManagement.exception;

/**
 * Exception for InsertToDB query
 */
public class InsertToDBException extends Exception {

    /**
     * Parameterless C'tor
     */
    public InsertToDBException() {
        super();
    }

    /**
     * C'tor
     * @param message message that represent the exception
     * @param cause warp the throwable
     */
    public InsertToDBException(String message, Throwable cause) {
        super(message, cause);
    }

    /**
     * C'tor
     * @param message message that represent the exception
     */
    public InsertToDBException(String message) {
        super(message);
    }
}
```

```java
package il.ac.hit.CashFlowManagement.exception;

/**
 * Exception for FormCasting
 */
public class FormCastingException extends Exception {

    /**
     * Parameterless C'tor
     */
    public FormCastingException() {
        super();
    }

    /**
     * C'tor
     * @param message message that represent the exception
     * @param cause warp the throwable
     */
    public FormCastingException(String message, Throwable cause) {
        super(message, cause);
    }

    /**
     * C'tor
     * @param message message that represent the exception
     */
    public FormCastingException(String message) {
        super(message);
    }
}
```

```java
package il.ac.hit.CashFlowManagement.exception;

/**
 * Exception for JDBCDataBase
 */
public class JDBCDataBaseException extends Exception {
    /**
     * Parameterless C'tor
     */
    public JDBCDataBaseException() {
        super();
    }

    /**
     * C'tor
     * @param message message that represent the exception
     * @param cause warp the throwable
     */
    public JDBCDataBaseException(String message, Throwable cause) {
        super(message, cause);
    }

    /**
     * C'tor
     * @param message message that represent the exception
     */
    public JDBCDataBaseException(String message) {
        super(message);
    }
}
```

```java
package il.ac.hit.CashFlowManagement.exception;

/**
 * Exception for JDBCDataBaseLogic
 */
public class JDBCDataBaseLogicException extends Exception{

    /**
     * Parameterless C'tor
     */
    public JDBCDataBaseLogicException() {
        super();
    }

    /**
     * C'tor
     * @param message message that represent the exception
     * @param cause warp the throwable
     */
    public JDBCDataBaseLogicException(String message, Throwable cause
) {
        super(message, cause);
    }

    /**
     * C'tor
     * @param message message that represent the exception
     */
    public JDBCDataBaseLogicException(String message) {
        super(message);
    }
}

```

```java
package il.ac.hit.CashFlowManagement.exception;

/**
 * Exception for getAllUserExpenses query
 */
public class GetAllUserExpensesException extends Exception {

    /**
     * Parameterless C'tor
     */
    public GetAllUserExpensesException() {
        super();
    }

    /**
     * C'tor
     * @param message message that represent the exception
     * @param cause warp the throwable
     */
    public GetAllUserExpensesException(String message, Throwable cause) {
        super(message, cause);
    }

    /**
     * C'tor
     * @param message message that represent the exception
     */
    public GetAllUserExpensesException(String message) {
        super(message);
    }
}
```

```java
package il.ac.hit.CashFlowManagement.viewmodel;
import il.ac.hit.CashFlowManagement.exception.
GetAllUserExpensesException;
import org.apache.log4j.Logger;

import javax.swing.*;
import javax.swing.table.DefaultTableModel;
import javax.swing.table.TableModel;
import javax.swing.table.TableRowSorter;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.time.LocalDate;
import java.time.format.DateTimeFormatter;
import java.util.Comparator;
import java.util.Locale;
/**
 *
 *  The Table Data class
 */

public class TableData
{
    private static Logger logger;
    private IViewModel viewModel;
    private final String[] columnNames = { "Date", "Classification",
"Details", "Cost"};

    //creating gui components
    private JTable dataTable;
    private JScrollPane tableJScrollPane;
    private DefaultTableModel model;
    private JButton back = new JButton("Back");

    /**
     * TableData C'tor
     */
    public TableData() {
        init();
    }

    /**
     * initialize table view and sorters logic
     */
    private void init() {
        setLogger();
        setViewModel();
```

```
45          dataTable = new JTable(new DefaultTableModel(null,
    columnNames));
46          dataTable.setAutoCreateRowSorter(true);
47          dataTable.setAutoResizeMode(JTable.AUTO_RESIZE_OFF);
48          dataTable.setAutoscrolls(true);
49          dataTable.setAutoResizeMode(JTable.AUTO_RESIZE_ALL_COLUMNS);
50          tableJScrollPane = new JScrollPane(dataTable);
51          model = (DefaultTableModel) dataTable.getModel();
52          TableRowSorter<TableModel> sorter = new TableRowSorter<>(
    dataTable.getModel());
53          sorter.setComparator(dataTable.getColumnCount() -1 , new
    Comparator<String>() {
54              @Override
55              public int compare(String cost1, String cost2) {
56                  Double cost1d, cost2d;
57
58                  cost1d = Double.parseDouble(cost1);
59                  cost2d = Double.parseDouble(cost2);
60
61                  return cost1d.compareTo(cost2d);
62              }
63          });
64          sorter.setComparator(0 , new Comparator<String>() {
65              @Override
66              public int compare(String date1, String date2) {
67                  LocalDate date1LD, date2LD;
68                  DateTimeFormatter formatter = DateTimeFormatter.
    ofPattern("dd/MM/yyyy", Locale.getDefault());
69
70                  date1LD = LocalDate.parse(date1, formatter);
71                  date2LD = LocalDate.parse(date2, formatter);
72
73                  return date1LD.compareTo(date2LD);
74              }
75          });
76          dataTable.setRowSorter(sorter);
77          dataTable.getRowSorter().toggleSortOrder(0);
78          dataTable.getRowSorter().toggleSortOrder(0);
79      }
80
81      /**
82       * get all expenses data from DB
83       */
84      private void getData() {
85          Runnable run = () -> {
86              // 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
    GUI creation task (GUI thread) who updates the table information
100                      String finalClassification, finalDetails,
    finalDate, finalCost;
101
102                      date = rs.getString("Date");
103                      LocalDate localDate = LocalDate.parse(date,
    DateTimeFormatter.ofPattern("dd/MM/yyyy", Locale.getDefault()));
104                      cost = rs.getString("Cost");
105                      classification = rs.getString("Classification");
106                      details = rs.getString("Details");
107                      finalDate = localDate.format(DateTimeFormatter.
    ofPattern("dd/MM/yyyy"));
108                      finalCost = cost;
109                      finalClassification = classification;
110                      finalDetails = details;
111
112                      SwingUtilities.invokeLater(new Runnable() {
113                          @Override
114                          public void run() {
115                              Object[] row = new Object[]{finalDate,
    finalClassification, finalDetails, finalCost};
116                              model.insertRow(dataTable.getRowCount
    (), row);
117                          }
118                      });
119                  }
120                  logger.info("TableData.getData ended successfully");
121              } catch (SQLException | GetAllUserExpensesException e)
122              {
123                  logger.info("TableData.getData failed because" + e.
    getMessage());
124                  e.printStackTrace();
```

```
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
```

```java
package il.ac.hit.CashFlowManagement.viewmodel;

import il.ac.hit.CashFlowManagement.exception.FormCastingException;
import il.ac.hit.CashFlowManagement.exception.
GetAllUserExpensesException;
import org.jetbrains.annotations.NotNull;

import java.sql.ResultSet;

/**
 * this interface define the operations that each Form/Frame must
implement
 */
public interface IViewModel {

    /**
     * add viewModel to all forms and Show the dialog of the
LoginForm
     */
    void startProgram();

    /**
     * The method verifies that the login information (username and
password) is correct
     * @param iUserName the username of the user who want to login,
can't be null
     * @param iPassword the password of the user who want to login,
can't be null
     * @return true either false if the login information correct
     */
    boolean verifyUser(@NotNull String iUserName, @NotNull String
iPassword);

    /**
     * The method check if the user exist and if not created a new
user
     * @param iUserName the new user username, can't be null
     * @param iPassword the new user password, can't be null
     * @param iCountry the new user country, can't be null
     * @param iGander the new user gander, can't be null
     * @return true if new user registered or false if the user
already exists
     */
    boolean register(@NotNull String iUserName, @NotNull String
iPassword, @NotNull String iCountry, @NotNull String iGander);

```

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

```java
package il.ac.hit.CashFlowManagement.viewmodel;

import il.ac.hit.CashFlowManagement.exception.FormCastingException;
import il.ac.hit.CashFlowManagement.exception.
GetAllUserExpensesException;
import il.ac.hit.CashFlowManagement.model.Expense;
import il.ac.hit.CashFlowManagement.model.ExpensesLogic;
import il.ac.hit.CashFlowManagement.model.UserLogic;
import il.ac.hit.CashFlowManagement.view.IView;
import il.ac.hit.CashFlowManagement.view.LoginForm;
import il.ac.hit.CashFlowManagement.view.MainForm;
import il.ac.hit.CashFlowManagement.view.RegisterForm;
import org.apache.log4j.BasicConfigurator;
import org.apache.log4j.Logger;
import org.jetbrains.annotations.NotNull;

import javax.swing.*;
import java.sql.ResultSet;
import java.util.Arrays;


/**
 * The  Management View Model class
 */
public class ManagementViewModel implements IViewModel{
    private static Logger logger;
    private static ManagementViewModel instance = null;
    private static Object lock = new Object();
    private IView loginForm, mainForm, registerForm;
    private ExpensesLogic expensesLogic;
    private UserLogic userLogic;
    private String newExpenseDay, newExpenseMonth, newExpenseYear,
newExpenseClassification, newExpenseDetail, newExpenseCost;

    /**
     * ManagementViewModel C'tor
     */
    private ManagementViewModel() {
        BasicConfigurator.configure();
        setLogger(Logger.getLogger(ManagementViewModel.class));
        setMainForm(new MainForm());
        setLoginForm(new LoginForm());
        setRegisterForm(new RegisterForm());
        setUserLogic(new UserLogic());
        logger.info("ManagementViewModel was create successfully");
    }
```

```
45
46      /**
47       * check if there is a singleton instance of ManagementViewModel
    and create it's if needed
48       * @return instance of ManagementViewModel
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
    String iPassword) {
123          Boolean valueToReturn = false;
124
125          logger.info("ManagementViewModel.verifyUser started");
126          if(getUserLogic().checkUserPassword(iUserName, iPassword))
127          {
128              valueToReturn = true;
129          }
130
131          logger.info("ManagementViewModel.verifyUser ended");
132
133          return valueToReturn;
134      }
```

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

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

```
204
205                 if(day.equalsIgnoreCase("")) message.append("select
    day");
206                 else if(month.equalsIgnoreCase("")) message.append("
    select month");
207                 else if(year.equalsIgnoreCase("")) message.append("
    select year");
208                 else if(classification.equalsIgnoreCase("")) message
    .append("select classification");
209                 else if(detail.equalsIgnoreCase("")) message.append(
    "add some detail");
210                 else if (cost.equalsIgnoreCase("")) message.append("
    fill cost");
211                 JOptionPane.showMessageDialog(null, message);
212             }
213             Double.parseDouble(cost);
214         }
215         catch (NumberFormatException e){
216             JOptionPane.showMessageDialog(null, "Cost must be only
    number");
217             correctInformation = false;
218         }
219         finally {
220             logger.info("ManagementViewModel.
    correctInfoForNewExpense ended successfully");
221             return correctInformation;
222         }
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().
    correctInfoForNewExpense()) {
233                 double cost = Double.parseDouble(getNewExpenseCost
    ());
234                 String dateStr = getNewExpenseDay() + "/" + Arrays.
    asList(form.getMonths()).indexOf(getNewExpenseMonth()) + "/" +
    getNewExpenseYear();
235
236                 Expense expense = new Expense(dateStr,
    getNewExpenseClassification(), getNewExpenseDetail(), cost);
```

```
237                     expensesLogic.addExpense(expense);
238                 }
239                 logger.info("ManagementViewModel.addNewExpense ended
    successfully, new expense added");
240             } catch (ClassCastException e){
241                 throw new FormCastingException("ManagementViewModel.
    addNewExpense failed, IForm to MainForm casting problem",e);
242             }
243             catch (NumberFormatException ex){
244                 throw new FormCastingException("ManagementViewModel.
    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 }
```