The Super Light Version

Students:

Name: Noor Haj Dawood

ID: 314997602

Phone Number: 052-723-6852

Email: noorhajdawood@gmail.com

Name: Bader Daka

ID: 208219212

Phone Number: 050-532-2621

Email: baderdaka28@gmail.com

Demo Link: https://youtu.be/uQ5gkjQ6yA4

```java
package il.ac.shenkar.java.project;

import il.ac.shenkar.java.project.model.IModel;
import il.ac.shenkar.java.project.model.Model;
import il.ac.shenkar.java.project.view.IView;
import il.ac.shenkar.java.project.view.View;
import il.ac.shenkar.java.project.viewmodel.IViewModel;
import il.ac.shenkar.java.project.viewmodel.ViewModel;

import javax.swing.*;

/**
 * main application class to run the main
 */
public class Application {
    /**
     * main function of the application
     * @param args
     * @throws ClassNotFoundException
     */
    public static void main(String[] args) throws ClassNotFoundException {
        IModel model = new Model();
        IViewModel vm = new ViewModel();
        // run the view in the proper thread
        SwingUtilities.invokeLater(() -> {
            IView view = new View();
            view.start();
```

```
28              // set the viewmodel in the view
29              view.setViewModel(vm);
30              // set the model and view in viewmodel
31              vm.setModel(model);
32              vm.setView(view);
33              // sync the categories in the view with the categories in the database
34              view.syncCategories();
35          });
36      }
37  }
38
```

File - H:\Google Drive\Visual Studio Code Workspace\Java Programming\SuperLighter\src\il\ac\shenkar\java\project\CostsManagerException.java

```java
package il.ac.shenkar.java.project;

/**
 * exception class for the project
 */
public class CostsManagerException extends Exception{
    /**
     * constructor with only message
     * @param message
     */
    public CostsManagerException(String message) {
        super(message);
    }

    /**
     * constructor with message and cause
     * @param message
     * @param cause
     */
    public CostsManagerException(String message, Throwable cause) {
        super(message, cause);
    }
}

```

```java
package il.ac.shenkar.java.project.dao;

import java.sql.Date;

/**
 * Cost class to represent cost object in database
 */
public class Cost {
    /**
     * id of the cost
     */
    private int id;
    /**
     * category id of the cost
     */
    private int categoryId;
    /**
     * category name of the cost
     */
    private String categoryName;
    /**
     * sum of the cost
     */
    private float sum;
    /**
     * currency of the cost
     */
```

```
28      private String currency;
29      /**
30       * description of the cost
31       */
32      private String description;
33      /**
34       * date of the cost
35       */
36      private Date date;
37
38      /**
39       * constructor including id if the id is known
40       * @param id
41       * @param categoryId
42       * @param categoryName
43       * @param sum
44       * @param currency
45       * @param description
46       * @param date
47       */
48      public Cost(int id, int categoryId, String categoryName, float sum, String
    currency, String description, Date date) {
49          setId(id);
50          setCategoryId(categoryId);
51          setCategoryName(categoryName);
52          setSum(sum);
53          setCurrency(currency);
```

```
54          setDescription(description);
55          setDate(date);
56      }
57
58      /**
59       * constructor if the id is unkown
60       * @param categoryId
61       * @param sum
62       * @param currency
63       * @param description
64       * @param date
65       */
66      public Cost(int categoryId, float sum, String currency, String description, Date
    date) {
67          setCategoryId(categoryId);
68          setSum(sum);
69          setCurrency(currency);
70          setDescription(description);
71          setDate(date);
72      }
73
74      /**
75       * return the id of the cost
76       * @return
77       */
78      public int getId() {
79          return id;
```

```
80      }
81
82      /**
83       * set the id of the cost
84       * @param id
85       */
86      public void setId(int id) {
87          this.id = id;
88      }
89
90      /**
91       * return the category id of the cost
92       * @return
93       */
94      public int getCategoryId() {
95          return categoryId;
96      }
97
98      /**
99       * set the category id of the cost
100      * @param categoryId
101      */
102     public void setCategoryId(int categoryId) {
103         this.categoryId = categoryId;
104     }
105
106     /**
```

```
107      * get category name of the cost
108      * @return
109      */
110     public String getCategoryName() {
111         return categoryName;
112     }
113 
114     /**
115      * set category name of the cost
116      * @param categoryName
117      */
118     public void setCategoryName(String categoryName) {
119         this.categoryName = categoryName;
120     }
121 
122     /**
123      * get sum of the cost
124      * @return
125      */
126     public float getSum() {
127         return sum;
128     }
129 
130     /**
131      * set sum of the cost
132      * @param sum
133      */
```

```
134     public void setSum(float sum) {
135         this.sum = sum;
136     }
137 
138     /**
139      * get description of the cost
140      * @return
141      */
142     public String getDescription() {
143         return description;
144     }
145 
146     /**
147      * set description of the cost
148      * @param description
149      */
150     public void setDescription(String description) {
151         this.description = description;
152     }
153 
154     /**
155      * get date of the cost
156      * @return
157      */
158     public Date getDate() {
159         return date;
160     }
```

```
161
162     /**
163      * set date of the cost
164      * @param date
165      */
166     public void setDate(Date date) {
167         this.date = date;
168     }
169
170     /**
171      * get currency of the cost
172      * @return
173      */
174     public String getCurrency() {
175         return currency;
176     }
177
178     /**
179      * set currency of the cost
180      * @param currency
181      */
182     public void setCurrency(String currency) {
183         this.currency = currency;
184     }
185 }
186
```

```java
package il.ac.shenkar.java.project.dao;

/**
 * Category class to represent category object in database
 */
public class Category {
    /**
     * id of the category
     */
    private int id;
    /**
     * name of the category
     */
    private String name;

    /**
     * constructor with two parameters if id is known
     * @param id
     * @param name
     */
    public Category(int id, String name) {
        setId(id);
        setName(name);
    }

    /**
     * constructor with one parameter if id is unknown
```

```
28       * @param name
29       */
30      public Category( String name) {
31          setName(name);
32      }
33
34      /**
35       * return id of the category
36       * @return
37       */
38      public int getId() {
39          return id;
40      }
41
42      /**
43       * set the id of the category
44       * @param id
45       */
46      public void setId(int id) {
47          this.id = id;
48      }
49
50      /**
51       * return name of the category
52       * @return
53       */
54      public String getName() {
```

```
55          return name;
56      }
57
58      /**
59       * set the name of the category
60       * @param name
61       */
62      public void setName(String name) {
63          this.name = name;
64      }
65
66      /**
67       * string representation of the category
68       * @return
69       */
70      @Override
71      public String toString() {
72          return "Category{" +
73                  "id=" + id +
74                  ", name='" + name + '\'' +
75                  '}';
76      }
77  }
78
```

```java
package il.ac.shenkar.java.project.dao;

import il.ac.shenkar.java.project.CostsManagerException;

import java.sql.Date;
import java.util.List;

/**
 * interface of the costs manager DAO
 */
public interface ICostsManagerDAO {
    /**
     * add a new category to the database
     * @param category
     * @throws CostsManagerException
     */
    public void addCategory(Category category) throws CostsManagerException;

    /**
     * get all categories in the database
     * @return
     * @throws CostsManagerException
     */
    public List<Category> getCategories() throws CostsManagerException;

    /**
     * return a specific category in the database by name
```

```
28       * @param name
29       * @return
30       * @throws CostsManagerException
31       */
32      public Category getCategoryByName(String name) throws CostsManagerException;
33
34      /**
35       * add a new cost to the database
36       * @param cost
37       * @throws CostsManagerException
38       */
39      public void addCost(Cost cost) throws CostsManagerException;
40
41      /**
42       * get all costs in the database from start date till end date
43       * @param start
44       * @param end
45       * @return
46       * @throws CostsManagerException
47       */
48      public List<Cost> getCosts(Date start, Date end) throws CostsManagerException;
49  }
50
```

```java
package il.ac.shenkar.java.project.dao;

import il.ac.shenkar.java.project.CostsManagerException;

import java.sql.*;
import java.util.LinkedList;
import java.util.List;

/**
 * database access object implemented for MySQL
 */
public class MySQLCostsManagerDAO implements ICostsManagerDAO {
    /**
     * name of the MySQL driver
     */
    private String driver = "com.mysql.jdbc.Driver";
    /**
     * connection string to the database
     */
    private String connectionString = "jdbc:mysql://localhost:3306/costsmanager";

    /**
     * constructor to make sure that the MySQL driver is imported to the project
     * @throws ClassNotFoundException
     */
    public MySQLCostsManagerDAO() throws ClassNotFoundException {
        Class.forName(driver);
```

```
28      }
29
30      /**
31       * add a new category to the database
32       * @param category
33       * @throws CostsManagerException
34       */
35      @Override
36      public void addCategory(Category category) throws CostsManagerException {
37          // connect to the database and make a prepared statement to insert a
    category
38          try(Connection connection =DriverManager.getConnection(connectionString, "a"
    , "b");
39              PreparedStatement statement = connection.prepareStatement("INSERT INTO
    categories (name) VALUES(?)")){
40              // make sure that the category doesn't already exist
41              if(getCategoryByName(category.getName()) != null) {
42                  throw new CostsManagerException("Category already exists");
43              }
44              // import the name of category from function arguments
45              statement.setString(1, category.getName());
46              statement.executeUpdate();
47          } catch (SQLException e) {
48              throw new CostsManagerException("Error connecting to database", e);
49          }
50      }
51
```

```
52      /**
53       * get all categories in the database
54       * @return
55       * @throws CostsManagerException
56       */
57      @Override
58      public List<Category> getCategories() throws CostsManagerException {
59          // a list to hold the result
60          List<Category> list = new LinkedList<Category>();
61          // connect to the database and make a prepared statement to get all
    categories
62          try(Connection connection =DriverManager.getConnection(connectionString, "a"
    , "b");
63              PreparedStatement statement = connection.prepareStatement("SELECT * FROM
     categories ORDER BY id");
64              ResultSet rs = statement.executeQuery()){
65              // iterate on the result set and insert all to the result list
66              while(rs.next()){
67                  list.add(new Category(rs.getInt("id"), rs.getString("name")));
68              }
69          } catch (SQLException e) {
70              throw new CostsManagerException("Error connecting to database", e);
71          }
72          return list;
73      }
74
75      /**
```

```
76       * return a specific category in the database by name
77       * @param name
78       * @return
79       * @throws CostsManagerException
80       */
81      @Override
82      public Category getCategoryByName(String name) throws CostsManagerException {
83          // category result
84          Category category = null;
85          ResultSet rs = null;
86          // connect to the database and make a prepared statement to get a category
    by name
87          try(Connection connection =DriverManager.getConnection(connectionString, "a
    ", "b");
88              PreparedStatement statement = connection.prepareStatement("SELECT *
    FROM categories WHERE name = ?")){
89              // set name of the category from the function arguments
90              statement.setString(1, name);
91              rs = statement.executeQuery();
92              // set the return result
93              if(rs.next()) {
94                  category = new Category(rs.getInt("id"), rs.getString("name"));
95              }
96          } catch (SQLException e) {
97              throw new CostsManagerException("Error connecting to database", e);
98          } finally {
99              if(rs!=null) {
```

```
100                 try {
101                     rs.close();
102                 } catch (SQLException e) {
103                     e.printStackTrace();
104                 }
105             }
106         }
107         return category;
108     }
109 
110     /**
111      * add a new cost to the database
112      * @param cost
113      * @throws CostsManagerException
114      */
115     @Override
116     public void addCost(Cost cost) throws CostsManagerException {
117         // connect to the database and make a prepared statement to insert a cost
118         try(Connection connection = DriverManager.getConnection(connectionString, "
    a", "b");
119             PreparedStatement statement = connection.prepareStatement("INSERT INTO
    costs (category_id, sum, currency, description, date) VALUES(?, ?, ?, ?, ?)")){
120             // set statement parameters from the function arguments
121             statement.setInt(1, cost.getCategoryId());
122             statement.setFloat(2, cost.getSum());
123             statement.setString(3, cost.getCurrency());
124             statement.setString(4, cost.getDescription());
```

```
125             statement.setDate(5, cost.getDate());
126             statement.executeUpdate();
127         } catch (SQLException e) {
128             throw new CostsManagerException("Error connecting to database", e);
129         }
130     }
131 
132     /**
133      * get all costs in the database from start date till end date
134      * @param start
135      * @param end
136      * @return
137      * @throws CostsManagerException
138      */
139     @Override
140     public List<Cost> getCosts(Date start, Date end) throws CostsManagerException {
141         ResultSet rs = null;
142         // query string of the prepared statemnt
143         String query = "SELECT costs.*, categories.name as category_name FROM costs
     JOIN categories ON costs.category_id = categories.id WHERE costs.date >= ? AND
    costs.date <= ?";
144         // result set
145         List<Cost> list = new LinkedList<Cost>();
146         // connect to the database and make a prepared statement to get all costs
147         try(Connection connection =DriverManager.getConnection(connectionString, "a
    ", "b");
148             PreparedStatement statement = connection.prepareStatement(query)){
```

```
149             // set start and end date from the function arguments
150             statement.setDate(1, start);
151             statement.setDate(2, end);
152             rs = statement.executeQuery();
153             while(rs.next()){
154                 list.add(new Cost(rs.getInt("id"), rs.getInt("category_id"), rs.
    getString("category_name"), rs.getFloat("sum"), rs.getString("currency"), rs.
    getString("description"), rs.getDate("date")));
155             }
156         } catch (SQLException e) {
157             throw new CostsManagerException("Error connecting to database", e);
158         }
159         return list;
160     }
161 }
162
```

```java
package il.ac.shenkar.java.project.view;

import il.ac.shenkar.java.project.dao.Category;
import il.ac.shenkar.java.project.dao.Cost;
import il.ac.shenkar.java.project.viewmodel.IViewModel;

import javax.swing.*;
import java.awt.*;
import java.awt.event.WindowAdapter;
import java.awt.event.WindowEvent;
import java.sql.Date;
import java.util.Calendar;
import java.util.GregorianCalendar;
import java.util.List;

import com.toedter.calendar.JDateChooser;

public class View implements IView {
    /**
     * viewmodel reference
     */
    private IViewModel vm;
    /**
     * main frame reference
     */
    private JFrame mainFrame;
    /**
```

```
28      * reference to each panel
29      */
30     private JPanel mainPanel, titlePanel, costPanel, categoryPanel, reportPanel,
   reportWrapper;
31
32     /**
33      * labels in the costs panel
34      */
35     private JLabel costSumLabel, costCategoryLabel, costDescriptionLabel,
   costDateLabel, costCurrencyLabel;
36     /**
37      * text fields in the costs panel
38      */
39     private JTextField costSumTF, costDescriptionTF;
40     /**
41      * combobox in the costs panel
42      */
43     private JComboBox<String> costCategoryCB, costCurrencyCB;
44     /**
45      * date chooser in the costs panel
46      */
47     private JDateChooser costDateDC;
48     /**
49      * button in the costs panel
50      */
51     private JButton costSubmitBtn;
52
```

```
53      /**
54       * labels in the category panel
55       */
56      private JLabel categoryNameLabel;
57      /**
58       * text field in the category panel
59       */
60      private JTextField categoryNameTF;
61      /**
62       * button in the category panel
63       */
64      private JButton categorySubmitBtn;
65      /**
66       * text area in the category panel
67       */
68      private JTextArea categoryTA;
69
70      /**
71       * labels in the report panel
72       */
73      private JLabel reportFromLabel, reportToLabel;
74      /**
75       * date choosers in the report panel
76       */
77      private JDateChooser reportFromDC, reportToDC;
78      /**
79       * button in the report panel
```

```
 80      */
 81     private JButton reportSubmitBtn;
 82     /**
 83      * text area in the report panel
 84      */
 85     private JTextArea reportTA;
 86     /**
 87      * scroll pane in the report panel
 88      */
 89     private JScrollPane reportTASP;
 90 
 91     /**
 92      * constructor to initialize all elements
 93      */
 94     public View() {
 95         // initalize the frames and panels
 96         mainFrame = new JFrame("Costs Manager");
 97         mainPanel = new JPanel();
 98         titlePanel = new JPanel();
 99         costPanel = new JPanel();
100         categoryPanel = new JPanel();
101         reportPanel = new JPanel();
102         reportWrapper = new JPanel();
103 
104         // initialize cost panel elements
105         costSumLabel = new JLabel("sum:");
106         costCategoryLabel = new JLabel("category:");
```

```
107         costDescriptionLabel = new JLabel("Description:");
108         costDateLabel = new JLabel("Date:");
109         costCurrencyLabel = new JLabel("Currency:");
110         costSumTF = new JTextField(10);
111         costDescriptionTF = new JTextField(10);
112         costCategoryCB = new JComboBox<String>();
113         costCurrencyCB = new JComboBox<String>();
114         costDateDC = new JDateChooser();
115         costSubmitBtn = new JButton("Add Cost");
116 
117         // initialize category panel elements
118         categoryNameLabel = new JLabel("Name:");
119         categoryNameTF = new JTextField(10);
120         categorySubmitBtn = new JButton("Add Category");
121         categoryTA = new JTextArea();
122 
123         // initalize report panel elements
124         reportFromLabel = new JLabel("From:");
125         reportToLabel = new JLabel("To");
126         reportFromDC = new JDateChooser();
127         reportToDC = new JDateChooser();
128         reportSubmitBtn = new JButton("Show Costs");
129         reportTA = new JTextArea(10,1);
130         reportTASP = new JScrollPane(reportTA);
131     }
132     /**
133      * set the viewmodel reference
```

```
134      * @param vm
135      */
136     @Override
137     public void setViewModel(IViewModel vm) {
138         this.vm = vm;
139 
140     }
141     /**
142      * start the interface
143      */
144     public void start() {
145         // setup the main frame and main panel
146         BoxLayout boxlayout = new BoxLayout(mainPanel, BoxLayout.Y_AXIS);
147         mainPanel.setLayout(boxlayout);
148         JLabel titleLabel = new JLabel("<html><span style='color: teal;'>Costs
    Manager</span></html>", SwingConstants.CENTER);
149         titleLabel.setFont(titleLabel.getFont().deriveFont(32.0F));
150         titlePanel.add(titleLabel);
151         mainPanel.add(titlePanel);
152         mainPanel.add(Box.createVerticalStrut(10));
153 
154         // setup cost panel and its elements
155         costPanel.setLayout(new FlowLayout(FlowLayout.LEFT));
156         costPanel.setBorder(BorderFactory.createTitledBorder("New Cost"));
157         costPanel.add(costSumLabel);
158         costPanel.add(costSumTF);
159         costPanel.add(costCurrencyLabel);
```

```
160         costCurrencyCB.addItem("NIS");
161         costCurrencyCB.addItem("USD");
162         costCurrencyCB.addItem("EUR");
163         costPanel.add(costCurrencyCB);
164         costPanel.add(costCategoryLabel);
165         costPanel.add(costCategoryCB);
166         costCategoryCB.setPrototypeDisplayValue("Category Name");
167         costPanel.add(costDescriptionLabel);
168         costPanel.add(costDescriptionTF);
169         costPanel.add(costDateLabel);
170         costPanel.add(costDateDC);
171         // add an action listener for adding a new cost
172         costSubmitBtn.addActionListener(action -> {
173             String categoryName = (String) costCategoryCB.getSelectedItem();
174             Float sum = 0F;
175             try {
176                 sum = Float.parseFloat(costSumTF.getText());
177             } catch (NumberFormatException e) {
178                 e.printStackTrace();
179                 displayMSG("Sum must be a float number.","ERROR", JOptionPane.
    ERROR_MESSAGE);
180 
181                 return;
182             }
183             if (categoryName == null) {
184                 displayMSG("Must select a category.","ERROR", JOptionPane.
    ERROR_MESSAGE);
```

```
185
186                 return;
187             }
188             Calendar calendar = new GregorianCalendar();
189             calendar.setTime(costDateDC.getDate());
190
191             vm.addCost(sum, (String)costCurrencyCB.getSelectedItem(), categoryName
    , costDescriptionTF.getText(), new Date(calendar.getTimeInMillis()));
192         });
193         costPanel.add(costSubmitBtn);
194         costPanel.add(Box.createHorizontalStrut(10));
195         mainPanel.add(costPanel);
196         // spacer
197         mainPanel.add(Box.createVerticalStrut(25));
198
199         // setup category panel and its elements
200         categoryPanel.setLayout(new FlowLayout(FlowLayout.LEFT));
201         categoryPanel.setBorder(BorderFactory.createTitledBorder("New Category"));
202         categoryPanel.add(categoryNameLabel);
203         categoryPanel.add(categoryNameTF);
204         // add an action listener to add a new category
205         categorySubmitBtn.addActionListener(action -> {
206             String name = categoryNameTF.getText();
207             if(name == null || name.equals("")) {
208                 displayMSG("Category name can't be empty.","ERROR", JOptionPane.
    ERROR_MESSAGE);
209                 return;
```

```
210             }
211             vm.addCategory(name);
212         });
213         categoryPanel.add(categorySubmitBtn);
214         mainPanel.add(categoryPanel);
215         // spacer
216         mainPanel.add(Box.createVerticalStrut(25));
217 
218         // setup report panel and its elements
219         BoxLayout reportboxlayout = new BoxLayout(reportWrapper, BoxLayout.Y_AXIS);
220         reportWrapper.setLayout(reportboxlayout);
221         reportPanel.setLayout(new FlowLayout(FlowLayout.LEFT));
222         reportWrapper.setBorder(BorderFactory.createTitledBorder("Costs Report"));
223         reportPanel.add(reportFromLabel);
224         reportPanel.add(reportFromDC);
225         reportPanel.add(reportToLabel);
226         reportPanel.add(reportToDC);
227         reportPanel.add(reportSubmitBtn);
228         // add an action listener to list a costs report
229         reportSubmitBtn.addActionListener(action -> {
230             Calendar startCalendar = new GregorianCalendar();
231             Calendar endCalendar = new GregorianCalendar();
232             startCalendar.setTime(reportFromDC.getDate());
233             endCalendar.setTime(reportToDC.getDate());
234             vm.getCosts(new Date(startCalendar.getTimeInMillis()), new Date(
    endCalendar.getTimeInMillis()));
235         });
```

```
236         reportWrapper.add(reportPanel);
237         reportWrapper.add(reportTASP);
238         mainPanel.add(reportWrapper);
239 
240         // change behavior of close button
241         mainFrame.addWindowListener(new WindowAdapter() {
242             @Override
243             public void windowClosing(WindowEvent e) {
244                 System.exit(0);
245             }
246         });
247 
248         // display the main frame
249         mainFrame.add(mainPanel);
250         mainFrame.pack();
251         mainFrame.setVisible(true);
252     }
253 
254     /**
255      * helper function for setCosts
256      * @param costs
257      */
258     private void updateReport(List<Cost> costs) {
259         // print costs report where each line is: [date] [category] cost currency
    => description.
260         StringBuffer report = new StringBuffer();
261         for (Cost cost : costs) {
```

```
262             report.append("[")
263                     .append(cost.getDate())
264                     .append("] [")
265                     .append(cost.getCategoryName())
266                     .append("] ")
267                     .append(cost.getSum())
268                     .append(" ")
269                     .append(cost.getCurrency())
270                     .append(" => ")
271                     .append(cost.getDescription())
272                     .append("\n");
273         }
274         reportTA.setText(report.toString());
275     }
276     /**
277      * set the costs in the view
278      * @param costs
279      */
280     @Override
281     public void setCosts(List<Cost> costs) {
282         // make sure to only execute in the dispatch thread
283         if (SwingUtilities.isEventDispatchThread()) {
284             updateReport(costs);
285         } else {
286             SwingUtilities.invokeLater(() -> {
287                 updateReport(costs);
288             });
```

```
289         }
290     }
291 
292     /**
293      * helper function for setCategories
294      * @param categories
295      */
296     private void updateCategories(List<Category> categories) {
297         // update categories in combobox
298         costCategoryCB.removeAllItems();
299         StringBuffer sb = new StringBuffer();
300         for (Category category : categories) {
301             costCategoryCB.addItem(category.getName());
302             sb.append(category.getName()).append("\n");
303         }
304         categoryTA.setText(sb.toString());
305     }
306     /**
307      * set the categories in the view
308      * @param categories
309      */
310     @Override
311     public void setCategories(List<Category> categories) {
312         // make sure to only execute in the dispatch thread
313         if (SwingUtilities.isEventDispatchThread()) {
314             updateCategories(categories);
315         } else {
```

```
316             SwingUtilities.invokeLater(() -> {
317                 updateCategories(categories);
318             });
319         }
320     }
321     /**
322      * display a message to the user
323      * @param msg
324      * @param type
325      * @param msgType
326      */
327     public void displayMSG(String msg , String type ,int msgType){
328         // make sure to only execute in the dispatch thread
329         if (SwingUtilities.isEventDispatchThread()) {
330             JOptionPane.showMessageDialog(mainFrame, msg,
331                     type, msgType);
332         } else {
333             SwingUtilities.invokeLater(() -> {
334                 JOptionPane.showMessageDialog(mainFrame, msg,
335                         type, msgType);
336             });
337         }
338 
339     }
340     /**
341      * sync the categories in the view with the database
342      */
```

```
343     @Override
344     public void syncCategories() {
345         vm.getCategories();
346     }
347 }
348
```

```java
package il.ac.shenkar.java.project.view;

import il.ac.shenkar.java.project.dao.Category;
import il.ac.shenkar.java.project.dao.Cost;
import il.ac.shenkar.java.project.viewmodel.IViewModel;

import java.util.List;

/**
 * View interface to allow main and viewmodel to interact with the view
 */
public interface IView {
    /**
     * set the viewmodel reference
     * @param vm
     */
    public void setViewModel(IViewModel vm);

    /**
     * start the interface
     */
    public void start();

    /**
     * set the costs in the view
     * @param costs
     */
```

```
28      public void setCosts(List<Cost> costs);
29
30      /**
31       * set the categories in the view
32       * @param categories
33       */
34      public void setCategories(List<Category> categories);
35
36      /**
37       * sync the categories in the view with the database
38       */
39      public void syncCategories();
40
41      /**
42       * display a message to the user
43       * @param msg
44       * @param type
45       * @param msgType
46       */
47      public void displayMSG(String msg , String type ,int msgType);
48  }
49
50
51
52
```

```java
package il.ac.shenkar.java.project.model;

import il.ac.shenkar.java.project.CostsManagerException;
import il.ac.shenkar.java.project.dao.Category;
import il.ac.shenkar.java.project.dao.Cost;
import il.ac.shenkar.java.project.dao.MySQLCostsManagerDAO;

import java.sql.Date;
import java.util.List;

/**
 * Logic of the mvvm
 */
public class Model implements IModel {
    /**
     * DAO reference
     */
    private MySQLCostsManagerDAO costsManagerDB;

    /**
     * deafult constructor to initialize the DAO reference
     * @throws ClassNotFoundException
     */
    public Model() throws ClassNotFoundException{
        costsManagerDB = new MySQLCostsManagerDAO();
    }

```

```
28      /**
29       * call get categories from DAO
30       * @return
31       * @throws CostsManagerException
32       */
33      public List<Category> getCategories() throws CostsManagerException {
34          return costsManagerDB.getCategories();
35      }
36      /**
37       * call add category from DAO
38       * @param name
39       * @throws CostsManagerException
40       */
41      public void addCategory(String name) throws CostsManagerException{
42          Category newCate = new Category(name);
43          costsManagerDB.addCategory(newCate);
44      }
45      /**
46       * call get costs from DAO
47       * @param start
48       * @param end
49       * @return
50       * @throws CostsManagerException
51       */
52      public List<Cost> getCosts(Date start , Date end) throws CostsManagerException{
53          return costsManagerDB.getCosts(start, end);
54      }
```

```
55      /**
56       * call add cost from DAO
57       * @param sum
58       * @param currency
59       * @param categoryName
60       * @param description
61       * @param date
62       * @throws CostsManagerException
63       */
64      public void addCost(float sum, String currency, String categoryName, String
    description, Date date) throws CostsManagerException{
65          Cost newCost = new Cost(costsManagerDB.getCategoryByName(categoryName).getId
    (),sum, currency, description,date);
66          costsManagerDB.addCost(newCost);
67      }
68  }
69
```

```java
package il.ac.shenkar.java.project.model;

import il.ac.shenkar.java.project.CostsManagerException;
import il.ac.shenkar.java.project.dao.Category;
import il.ac.shenkar.java.project.dao.Cost;

import java.sql.Date;
import java.util.List;

/**
 * Model interface
 */
public interface IModel {
    /**
     * call get categories from DAO
     * @return
     * @throws CostsManagerException
     */
    public List<Category> getCategories() throws CostsManagerException;

    /**
     * call add category from DAO
     * @param name
     * @throws CostsManagerException
     */
    public void addCategory(String name) throws CostsManagerException;

```

```
28      /**
29       * call get costs from DAO
30       * @param start
31       * @param end
32       * @return
33       * @throws CostsManagerException
34       */
35      public List<Cost> getCosts(Date start, Date end) throws CostsManagerException;
36
37      /**
38       * call add cost from DAO
39       * @param sum
40       * @param currency
41       * @param categoryName
42       * @param description
43       * @param date
44       * @throws CostsManagerException
45       */
46      public void addCost(float sum, String currency, String categoryName, String
    description, Date date) throws CostsManagerException;
47
48
49  }
50
```

```java
package il.ac.shenkar.java.project.viewmodel;

import il.ac.shenkar.java.project.CostsManagerException;
import il.ac.shenkar.java.project.dao.Category;
import il.ac.shenkar.java.project.dao.Cost;
import il.ac.shenkar.java.project.model.IModel;
import il.ac.shenkar.java.project.view.IView;

import javax.swing.*;
import java.sql.Date;
import java.util.List;
import java.util.concurrent.ExecutorService;
import java.util.concurrent.Executors;

/**
 * viewmodel implementation
 */
public class ViewModel implements IViewModel {
    /**
     * model reference
     */
    private IModel model;
    /**
     * view reference
     */
    private IView view;
    /**
```

```
28       * executor service to run each call in a new thread
29       */
30      private ExecutorService service;
31
32      /**
33       * constructor to create a fixed thread pool
34       */
35      public ViewModel() {
36          service = Executors.newFixedThreadPool(8);
37      }
38
39      /**
40       * set the model
41       * @param model
42       */
43      @Override
44      public void setModel(IModel model) {
45          this.model = model;
46      }
47
48      /**
49       * set the view
50       * @param view
51       */
52      @Override
53      public void setView(IView view) {
54          this.view = view;
```

```
55      }
56
57      /**
58       * call add category in model
59       * @param name
60       */
61      @Override
62      public void addCategory(String name){
63          // submit a thread to handel adding a category
64          service.submit(() -> {
65              try {
66                  model.addCategory(name);
67                  view.syncCategories();
68                  view.displayMSG("The category has been added successfully.","INFO",
    JOptionPane.INFORMATION_MESSAGE);
69              } catch (CostsManagerException e) {
70                  e.printStackTrace();
71                  view.displayMSG("Error adding category. make sure that the category
    doesn't already exist.","ERROR", JOptionPane.ERROR_MESSAGE);
72              }
73          });
74      }
75
76      /**
77       * call add cost in model
78       * @param sum
79       * @param currency
```

```
 80      * @param categoryName
 81      * @param description
 82      * @param date
 83      */
 84     @Override
 85     public void addCost(float sum, String currency, String categoryName, String
    description, Date date) {
 86         // submit a thread to handel adding a cost
 87         service.submit(() -> {
 88             try {
 89                 model.addCost(sum, currency, categoryName, description, date);
 90                 view.displayMSG("The cost has been added successfully.","INFO",
    JOptionPane.INFORMATION_MESSAGE);
 91             } catch (CostsManagerException e) {
 92                 e.printStackTrace();
 93                 view.displayMSG("Error adding cost. Try again!","ERROR",
    JOptionPane.ERROR_MESSAGE);
 94
 95             }
 96         });
 97     }
 98
 99     /**
100      * call get costs in model
101      * @param start
102      * @param end
103      */
```

```
104     @Override
105     public void getCosts(Date start, Date end) {
106         // submit a thread to get all costs
107         service.submit(() -> {
108             try {
109                 List<Cost> list = model.getCosts(start, end);
110                 view.setCosts(list);
111             } catch (CostsManagerException e) {
112                 e.printStackTrace();
113                 view.displayMSG("Error getting costs. Try again!","ERROR",
    JOptionPane.ERROR_MESSAGE);
114 
115             }
116         });
117     }
118 
119     /**
120      * call get categories in model
121      */
122     @Override
123     public void getCategories() {
124         // submit a thread to handel getting all categories
125         service.submit(() -> {
126             try {
127                 List<Category> list = model.getCategories();
128                 view.setCategories(list);
129             } catch (CostsManagerException e) {
```

```
130                 e.printStackTrace();
131                 view.displayMSG("Error getting categories. Try again!","ERROR",
    JOptionPane.ERROR_MESSAGE);
132 
133             }
134         });
135     }
136 }
137 
```

```java
package il.ac.shenkar.java.project.viewmodel;

import il.ac.shenkar.java.project.CostsManagerException;
import il.ac.shenkar.java.project.model.IModel;
import il.ac.shenkar.java.project.view.IView;

import java.sql.Date;

/**
 * viewmodel interface to help view communicate with model
 */
public interface IViewModel {
    /**
     * set the model
     * @param model
     */
    public void setModel(IModel model);

    /**
     * set the view
     * @param view
     */
    public void setView(IView view);

    /**
     * call add category in model
     * @param name
```

```
28      */
29     public void addCategory(String name);
30 
31     /**
32      * call add cost in model
33      * @param sum
34      * @param currency
35      * @param categoryName
36      * @param description
37      * @param date
38      */
39     public void addCost(float sum, String currency, String categoryName, String
   description, Date date);
40 
41     /**
42      * call get costs in model
43      * @param start
44      * @param end
45      */
46     public void getCosts(Date start, Date end);
47 
48     /**
49      * call get categories in model
50      */
51     public void getCategories();
52 }
53 
```

```java
package tests;

import il.ac.shenkar.java.project.CostsManagerException;
import il.ac.shenkar.java.project.dao.Category;
import il.ac.shenkar.java.project.dao.Cost;
import il.ac.shenkar.java.project.dao.ICostsManagerDAO;
import il.ac.shenkar.java.project.dao.MySQLCostsManagerDAO;
import org.junit.Before;
import org.junit.jupiter.api.BeforeAll;
import org.junit.jupiter.api.Test;

import java.sql.Date;

import static org.junit.jupiter.api.Assertions.assertEquals;
import static org.junit.jupiter.api.Assertions.assertTrue;


class MySQLCostsManagerDAOTest {

    static ICostsManagerDAO dao;

    @BeforeAll
    static void setUp() {
        try {
            dao = new MySQLCostsManagerDAO();
        } catch (ClassNotFoundException e) {
            e.printStackTrace();
```

```
28          }
29      }
30
31      @Test
32      void addCategory() {
33          // category name to test
34          String categoryName = "TestCategory";
35          try {
36              // add category to the database
37              dao.addCategory(new Category(categoryName));
38          } catch (CostsManagerException e) {
39              try {
40                  // make sure error thrown because category already exists
41                  assertEquals(dao.getCategoryByName(categoryName).getName(),
    categoryName);
42              } catch (CostsManagerException ex) {
43                  ex.printStackTrace();
44              }
45          }
46      }
47
48      @Test
49      void getCategories() {
50          try {
51              // test that the list is not empty
52              assertTrue(dao.getCategories().size() > 0);
53          } catch (CostsManagerException e) {
```

```
54              e.printStackTrace();
55          }
56      }
57
58      @Test
59      void getCategoryByName() {
60          // category name to test
61          String categoryName = "TestCategory";
62          try {
63              Category category = dao.getCategoryByName(categoryName);
64              assertEquals(category.getName(), categoryName);
65          } catch (CostsManagerException e) {
66              e.printStackTrace();
67          }
68      }
69
70      @Test
71      void addCost() {
72          try {
73              dao.addCost(new Cost(11, 10,"NIS","description",new Date(2022, 8, 18)));
74          } catch (CostsManagerException e) {
75              e.printStackTrace();
76          }
77      }
78
79      @Test
80      void getCosts() {
```

```
81          try {
82              // test that the list is not empty
83              assertTrue(dao.getCosts(new Date(2022, 8, 17), new Date(2022, 8, 19)).
    size() > 0);
84          } catch (CostsManagerException e) {
85              e.printStackTrace();
86          }
87      }
88  }
```