```java
/**
 * @author ${Yossef Hai Gavriel, Omer Pesah}
 */

package il.ac.hit;

import il.ac.hit.model.DataBaseModel;
import il.ac.hit.viewmodel.Management;

public class main {

    public static void main(String[] args)
    {
        DataBaseModel model = DataBaseModel.getInstance();
        Management.getManage().setModel(model);
        Management.getManage().startProgram();

    }
}

```

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

/**
 * This interface handle for all the view
 */
public interface IView {

    /**
     * show the form to the client
     * @throws Exception General exception
     */
    public void showForm() throws Exception;

    /**
     * Hide the form from the client
     */
    public void visibleOffForm();
}

```

```java
package il.ac.hit.view;
/*
NO TEST NEEDED BECAUSE THE METHODS ARE ALL ABOUT SHOW AND EDIT THE FORM
 */
import il.ac.hit.viewmodel.Management;

import javax.swing.*;
import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.util.logging.Logger;

public class LoginForm extends JFrame implements ActionListener, IView{

/**
This class creates a login form that allows the customer to identify with the system and log into the app.
This class includes methods such as:
* Show/hide form - show/hide the login form to/from the main screen.
* Set location of the components.
* Add action listeners and action events.
 */

    Container frame=getContentPane();
    JLabel titleLabel = new JLabel("BarberShop Appointment Manager");
    JLabel userLabel = new JLabel("Username");
    JLabel passwordLabel = new JLabel("Password");
    JTextField userTextField = new JTextField();
    JPasswordField passwordField = new JPasswordField();
    JButton loginButton = new JButton("Login");
    JButton resetButton = new JButton("Reset");
    JButton signUpButton = new JButton(("SignUp"));
    JCheckBox showPassword = new JCheckBox("Show Password");
    static Logger logger= Logger.getLogger(String.valueOf(LoginForm.class));

    /**
     * Constructor Create the login form
     */
    public LoginForm()
    {
```

```
40
41          showForm();
42          setResizable(false);
43          setLayoutManager();
44          setLocationAndSize();
45          addComponentsToContainer();
46          addActionEvent();
47          showPassword.setBackground(new Color(204,229,255));
48          loginButton.setBackground(new Color(153,204,255));
49          signUpButton.setBackground(new Color(153,204,255));
50          resetButton.setBackground(new Color(153,204,255));
51      }
52
53      /**
54       * Show the login form
55       */
56      @Override
57      public void showForm()
58      {
59
60          setTitle("Login Form");
61          setVisible(true);
62          setBounds(10, 10, 400, 600);
63          setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
64          logger.info("Show login form");
65      }
66
67      /**
68       * Hide the login form
69       */
70      @Override
71      public void visibleOffForm()
72      {
73          setVisible(false);
74          userTextField.setText("");
75          passwordField.setText("");
76      }
77
78      /**
79       * Set layout
80       */
81      public void setLayoutManager()
82      {
83
84          frame.setLayout(null);
85          frame.setBackground(new Color(204,229,255));
```

```
 86      }
 87
 88      /**
 89       * Set Location and size in frame
 90       */
 91      public void setLocationAndSize() {
 92          titleLabel.setBounds(50,50,300,50);
 93          titleLabel.setFont(new Font("David", Font.
    ROMAN_BASELINE, 20));
 94          userLabel.setBounds(50, 150, 100, 30);
 95          passwordLabel.setBounds(50, 220, 100, 30);
 96          userTextField.setBounds(150, 150, 150, 30);
 97          passwordField.setBounds(150, 220, 150, 30);
 98          showPassword.setBounds(150, 250, 150, 30);
 99          loginButton.setBounds(50, 300, 100, 30);
100          signUpButton.setBounds(125,350,100,30);
101          resetButton.setBounds(200, 300, 100, 30);
102      }
103
104      /**
105       * Add components
106       */
107      public void addComponentsToContainer() {
108          frame.add(titleLabel);
109          frame.add(userLabel);
110          frame.add(passwordLabel);
111          frame.add(userTextField);
112          frame.add(passwordField);
113          frame.add(showPassword);
114          frame.add(loginButton);
115          frame.add(resetButton);
116          frame.add(signUpButton);
117      }
118
119      /**
120       * Add action listeners
121       */
122      public void addActionEvent()
123      {
124          loginButton.addActionListener(this);
125          resetButton.addActionListener(this);
126          showPassword.addActionListener(this);
127          signUpButton.addActionListener(this);
128      }
129
130      /**
```

```
131      *
132      * @param e the event that occurred
133      *
134      */
135     @Override
136     public void actionPerformed(ActionEvent e) {
137         //Coding Part of LOGIN button
138         if (e.getSource() == loginButton) {
139             String userText;
140             String pwdText;
141             userText = userTextField.getText();
142             pwdText = String.valueOf(passwordField.
    getPassword());
143             if (Management.getManage().userTools.
    checkIfUserExist(userText,pwdText))//Validation test (User
     existence)
144             {
145                 JOptionPane.showMessageDialog(this, "
    Welcome " +userText+" Login Successful");
146                 visibleOffForm();
147                Management.getManage().showCalendarForm();
148                 logger.info("Login successful");
149             } else {
150                 JOptionPane.showMessageDialog(this, "
    Invalid Username or Password");
151                 logger.info("Login failed - invalid
    Username or Password");
152             }
153         }
154         //Coding Part of RESET button
155         if (e.getSource() == resetButton) {
156             userTextField.setText("");
157             passwordField.setText("");
158         }
159         //Coding Part of showPassword JCheckBox
160         if (e.getSource() == showPassword) //Reviling the
    password
161         {
162             if (showPassword.isSelected()) {
163                 passwordField.setEchoChar((char) 0);
164                 logger.warning("Showing the password");
165             } else {
166                 passwordField.setEchoChar('*');
167             }
168         }
169         if(e.getSource()== signUpButton)
```

```
170         {
171             logger.info("preparing to registration");
172             visibleOffForm();
173             Management.getManage().showSignUpForm();
174         }
175     }
176 }
177
178
```

```java
package il.ac.hit.view;
/*
NO TEST NEEDED BECAUSE THE METHODS ARE ALL ABOUT SHOW AND EDIT THE FORM
 */
import il.ac.hit.exceptions.DeleteException;
import il.ac.hit.viewmodel.Management;
import org.jetbrains.annotations.NotNull;

import javax.swing.*;
import javax.swing.table.DefaultTableModel;
import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.sql.ResultSet;
import java.sql.ResultSetMetaData;
import java.sql.SQLException;
import java.util.Vector;
import java.util.logging.Logger;

public class TableForm extends JFrame implements ActionListener, IView {

/**
This class displays a table of all appointments to the barber set on a specific date selected. The user can also mark an appointment and delete it.
This class includes methods such as:
* Show/hide form - show/hide the appointment form to/from the main screen.
* Set location of the components.
* Add action listeners and action events.
* Delete an appointment or go back to calendar form.
 */

    static Logger logger=Logger.getLogger("TableForm");
    String date;
    ResultSet rs;
    JTable table = new JTable();
    JPanel northPanel =new JPanel(new FlowLayout(FlowLayout.CENTER));
    JPanel centerPanel =new JPanel(new FlowLayout(FlowLayout.CENTER));
    JPanel southPanel =new JPanel(new FlowLayout(FlowLayout.CENTER));
    JButton backButton =new JButton("Back");
```

```
39      JButton deleteButton =new JButton("Delete");
40      JFrame frame =new JFrame();
41      JLabel dateLabel =new JLabel("");
42      JScrollPane pane = new JScrollPane(table);
43
44      /**
45       * Constructor - create table form
46       * @param date getting the picked date, Can't be null
47       */
48      public TableForm(@NotNull String date) {
49          setDate(date);
50          setLayoutManager();
51          editPanel();
52          addActionEvent();
53          backButton.setBackground(new Color(153,204,255));
54          deleteButton.setBackground(new Color(153,204,255));
55          northPanel.setBackground(new Color(204,229,255));
56          centerPanel.setBackground(new Color(204,229,255));
57          southPanel.setBackground(new Color(204,229,255));
58          frame.add(northPanel,BorderLayout.NORTH);
59          frame.add(centerPanel,BorderLayout.CENTER);
60          frame.add(southPanel,BorderLayout.SOUTH);
61          frame.pack();
62      }
63
64      /**
65       * Set layout
66       */
67      public void setLayoutManager()
68      {
69          frame.setLayout(new BorderLayout());
70          frame.setBackground(new Color(204,229,255));
71      }
72
73      /**
74       * Edit panel
75       */
76      public void editPanel()
77      {
78          dateLabel.setFont(new Font("David", Font.
    ROMAN_BASELINE, 20));
79          northPanel.add(dateLabel);
80
81          table.setFillsViewportHeight(true);
82
83          centerPanel.add(new JScrollPane(table));
```

```
 84
 85          southPanel.add(deleteButton);
 86          southPanel.add(backButton);
 87      }
 88
 89      /**
 90       * Show the form
 91       * @throws Exception general exception
 92       */
 93      @Override
 94      public void showForm() throws Exception
 95      {
 96          dateLabel.setText(date);
 97          showApp(rs);
 98          editPanel();
 99          frame.setTitle("Table Form");
100          frame.setVisible(true);
101          frame.setSize(500,600);
102          frame.setResizable(false);
103          logger.info("Show Table Form");
104          frame.setDefaultCloseOperation(JFrame.
    EXIT_ON_CLOSE);
105      }
106
107      /**
108       * Hide the form
109       */
110      @Override
111      public void visibleOffForm()
112      {
113          frame.setVisible(false);
114      }
115
116      /**
117       * Add action listeners
118       */
119      public void addActionEvent()
120      {
121          deleteButton.addActionListener(this);
122          backButton.addActionListener(this);
123      }
124
125      /**
126       * Action Performed
127       * @param e - the event that occurred
128       */
```

```
129     @Override
130     public void actionPerformed(ActionEvent e){
131         if (e.getSource() == deleteButton) {
132             int index= table.getSelectedRow();
133             if(index == -1 )// No selection
134             {
135                 JOptionPane.showMessageDialog(this, "No
    Appointment selected!");
136             }else {
137                 String date = (String) table.getValueAt(
    index, 0);
138                 String hour = (String) table.getValueAt(
    index, 1);
139                 try {
140                     Management.getManage().
    removeAppointment(date, hour);
141                 } catch (DeleteException ex) {
142                     ex.printStackTrace();
143                 }
144                 visibleOffForm();
145                 Management.getManage().showCalendarForm();
146                 logger.info("Delete from table was success
    ");
147             }
148         }
149         if(e.getSource()== backButton)
150         {
151             northPanel.removeAll();
152             visibleOffForm();
153             Management.getManage().showCalendarForm();
154             logger.info("Back to calendar");
155         }
156     }
157
158     /**
159      * Build new table model with the result set of the
    specified date.
160      * @param rs result set from data base, Can't be null
161      * @return updated table model
162      * @throws SQLException - syntactic query
163      */
164     public DefaultTableModel buildTableModel(@NotNull
    ResultSet rs) throws SQLException
165     {
166
167         DefaultTableModel defaultTableModel=new
```

```
167 DefaultTableModel(){
168 
169         @Override
170         public boolean isCellEditable(int row, int column
    ) {
171         //all cells false
172         return false;
173     }
174     };
175         ResultSetMetaData metaData = rs.getMetaData();
176 
177         //Name of columns:
178         Vector<String> columnNames = new Vector<String>();
179         int columnCount = metaData.getColumnCount();
180         for (int column = 1; column < columnCount; column
    ++) {
181             columnNames.add(metaData.getColumnName(column
    ));
182             defaultTableModel.addColumn(metaData.
    getColumnName(column));
183         }
184 
185         //Data of the table:
186         while(rs.next()){
187             Object[] objects = new Object[columnCount];
188             for(int i=0;i<columnCount;i++){
189                 objects[i]=rs.getObject(i+1);
190             }
191             defaultTableModel.addRow(objects);
192         }
193         logger.info("Building new model to show");
194         return defaultTableModel;
195     }
196 
197     /**
198      * Show appointment
199      * @param rs result set from data base, Can't be null
200      * @throws Exception general exception
201      */
202     public void showApp(@NotNull ResultSet rs) throws
    Exception {
203 
204         table.setModel(buildTableModel(rs));
205         logger.info("Updating table");
206 
207     }
```

```
208
209     /**
210      * Setting all the appointments that found in the specified date.
211      * @param getAppointments get result set of the asked appointments, Can't be null
212      */
213     public void setRs(@NotNull ResultSet getAppointments)
214     {
215         this.rs = getAppointments;
216     }
217
218     /**
219      * Set the date
220      * @param date getting updated date, Can't be null
221      */
222     public void setDate(@NotNull String date) {
223         this.date = date;
224     }
225 }
226
227
```

```java
package il.ac.hit.view;
/*
NO TEST NEEDED BECAUSE THE METHODS ARE ALL ABOUT SHOW AND EDIT THE FORM
 */
import il.ac.hit.exceptions.AddException;
import il.ac.hit.viewmodel.Management;

import javax.swing.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.*;
import java.util.logging.Logger;

public class SignUpForm extends JFrame implements ActionListener, IView {

/**
This class creates a registration form where the customer is given the option to add new users to the system using a username and password.
This class includes methods such as:
* Show/hide form - show/hide the registration form to/from the main screen.
* Set location of the components.
* Add action listeners and action events.
 */
    static Logger logger=Logger.getLogger(String.valueOf(SignUpForm.class));

    Container frame = getContentPane();
    JTextField tfUsername = new JTextField(10);
    JPasswordField tfPassword = new JPasswordField(10);
    JPasswordField tfRePassword = new JPasswordField(10);
    JButton btRegister = new JButton("Register");
    JButton btCancel = new JButton("Cancel");
    JLabel titleLabel = new JLabel("Registration");
    JLabel userLabel = new JLabel("Username: ");
    JLabel passwordLabel = new JLabel("Password: ");
    JLabel rePasswordLabel = new JLabel("Re-Password: ");

    /**
     * Constructor- create sign up form
     */
    public SignUpForm()
    {
```

```
41          setLayoutManager();
42          setLocationAndSize();
43          addComponentsToContainer();
44          addActionEvent();
45
46          logger.info("in sign process");
47      }
48
49      /**
50       * Show the form
51       */
52      @Override
53      public void showForm()
54      {
55          setTitle("SignUp");
56          setSize(400,600);
57          setVisible(true);
58          setResizable(false);
59          setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
60          logger.info("Show sign up form");
61      }
62
63      /**
64       * Hide the form
65       */
66      @Override
67      public void visibleOffForm()
68      {
69          setVisible(false);
70          tfPassword.setText("");
71          tfRePassword.setText("");
72          tfUsername.setText("");
73      }
74      /**
75       * Set layout
76       */
77      private void setLayoutManager()
78      {
79          frame.setLayout(null);
80          frame.setBackground(new Color(204,229,255));
81      }
82
83      private void setLocationAndSize()  //Set Location and
    size in frame
84      {
85          titleLabel.setBounds(150,50,150,30);
```

```
 86          titleLabel.setFont(new Font("David", Font.
    ROMAN_BASELINE, 20));
 87          userLabel.setBounds(50, 150, 100, 30);
 88          passwordLabel.setBounds(50, 220, 100, 30);
 89          rePasswordLabel.setBounds(50, 270, 100, 30);
 90          tfUsername.setBounds(150, 150, 150, 30);
 91          tfPassword.setBounds(150, 220, 150, 30);
 92          tfRePassword.setBounds(150, 270, 150, 30);
 93          btRegister.setBounds(50, 350, 100, 30);
 94          btCancel.setBounds(200, 350, 100, 30);
 95      }
 96
 97      /**
 98       * Add Components
 99       */
100      private void addComponentsToContainer()
101      {
102          frame.add(titleLabel);
103          frame.add(userLabel);
104          frame.add(passwordLabel);
105          frame.add(rePasswordLabel);
106          frame.add(tfUsername);
107          frame.add(tfPassword);
108          frame.add(tfRePassword);
109          frame.add(btRegister);
110          frame.add(btCancel);
111      }
112
113      /**
114       * Add action listeners
115       */
116      private void addActionEvent()
117      {
118          btRegister.addActionListener(this);
119          btCancel.addActionListener(this);
120      }
121      /**
122       * Action performed
123       * @param e - the event that occurred
124       */
125      @Override
126      public void actionPerformed(ActionEvent e) {
127          if(e.getSource()==btRegister){
128              String pass, rePass,userText;
129              pass = tfPassword.getText();
130              rePass = tfRePassword.getText();
```

```
131             userText = tfUsername.getText();
132             if((pass.equals(rePass)) && (!userText.equals(
    """)) && (!pass.equals(""))) {
133                 try {
134                     if(Management.getManage().addUserName(
    userText,pass))//If return false - the User is already in
    the User table (database).
135                     {
136                         JOptionPane.showMessageDialog(this
    , "Sign up Successfully");
137                         visibleOffForm();
138                         Management.getManage().
    showLoginForm();
139                         logger.info("Sign up success");
140                     }else {
141                         JOptionPane.showMessageDialog(this
    , "User is already registered");
142                         logger.info("Sign up failed - User
     is already registered");
143                     }
144                 } catch (AddException ex) {
145                     ex.printStackTrace();
146                 }
147             }
148             else{
149                 JOptionPane.showMessageDialog(this, "Sign
    up failed - no matching information");
150                 logger.info("Sign up failed - no matching
    information");
151             }
152         }
153         else if(e.getSource()==btCancel){
154             visibleOffForm();
155             Management.getManage().showLoginForm();
156             logger.info("Cancel from sign up");
157         }
158     }
159 }
```

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

import org.jetbrains.annotations.NotNull;

/*
NO TEST NEEDED BECAUSE THE METHODS ARE ALL GETTERS AND SETTERS WITHOUT CALCULATIONS
 */
public class Appointment {

/**
Build an appointment that includes date, time, client name, hairstyle type and customer phone.
This class has getters and setters methods.
 */
    private String date;
    private String hour;
    private String customerName;
    private String typeHairCut;
    private String customerTel;

    /**
     * Constructor Details appointment
     * @param date the date of the appointment, Can't be null
     * @param hour the hour of the appointment, Can't be null
     * @param customerName the customer's name, Can't be null
     * @param typeHairCut the haircut type, Can't be null
     * @param customerTel the customer's telephone number, Can't be null
     */
    public Appointment(@NotNull String date, @NotNull String hour,@NotNull String customerName,@NotNull String typeHairCut,@NotNull String customerTel)
    {
        setDate(date);
        setHour(hour);
        setCustomerName(customerName);
        setTypeHairCut(typeHairCut);
        setCustomerTel(customerTel);
    }

    /**
     * Get the date
```

```
39       * @return date
40       */
41      public String getDate() {
42          return date;
43      }
44
45      /**
46       * Set date
47       * @param date appointment date, Can't be null
48       */
49      public void setDate(@NotNull String date) {
50          this.date = date;
51      }
52
53      public String getHour() {
54          return hour;
55      }
56
57      /**
58       * Set hour
59       * @param hour appointment time, Can't be null
60       */
61      public void setHour(@NotNull String hour) {
62          this.hour = hour;
63      }
64
65      /**
66       * Get the name of customer
67       * @return name name of customer
68       */
69      public String getCustomerName() {
70          return customerName;
71      }
72
73      /**
74       * Set custom name
75       * @param customerName customer's name, Can't be null
76       */
77      public void setCustomerName(@NotNull String
    customerName) {
78          this.customerName = customerName;
79      }
80
81      public String getTypeHairCut() {
82          return typeHairCut;
83      }
```

```
 84
 85      /**
 86       * Set the hair cut type
 87       * @param typeHairCut haircut type, Can't be null
 88       */
 89      public void setTypeHairCut(@NotNull String typeHairCut
    ) {
 90          this.typeHairCut = typeHairCut;
 91      }
 92
 93      /**
 94       * Get the telephone customer
 95       * @return customerTel telephone of customer
 96       */
 97      public String getCustomerTel() {
 98          return customerTel;
 99      }
100
101      /**
102       * Set customer telephone
103       * @param customerTel customer's telephone number, Can
    't be null
104       */
105      public void setCustomerTel(@NotNull String customerTel
    ) {
106          this.customerTel = customerTel;
107      }
108
109  }
110
```

```java
package il.ac.hit.view;
/*
NO TEST NEEDED BECAUSE THE METHODS ARE ALL ABOUT SHOW AND EDIT THE FORM
 */
 import il.ac.hit.viewmodel.Management;

 import javax.swing.*;
import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
 import java.util.logging.Logger;

public class CalendarForm extends JFrame implements IView {

    /**
    This class displays a calendar where the user can switch between months and select a specific date. When selected, three options appear: create a meeting, view appointments on that date, cancel.
    This class includes methods such as:
    * Show/hide form - show/hide the calendar form to/from the main screen.
    * Add action listeners and action events.
    * button array to pick specific date.
    * Set picked date.
     */

        static Logger logger= Logger.getLogger(String.valueOf(CalendarForm.class));
        int month = java.util.Calendar.getInstance().get(java.util.Calendar.MONTH);
        int year = java.util.Calendar.getInstance().get(java.util.Calendar.YEAR);;
        JLabel l = new JLabel("", JLabel.CENTER);
        private String day =new String();
        JFrame frame;
        Object[] options = {"Make appointment", "Show all appointments","Cancel"};
        JButton[] button = new JButton[49];

    /**
     * Constructor
     * Create the frame and the calendar
     */
        public CalendarForm() {
```

```
38              frame = new JFrame();
39              String[] header = { "Sun", "Mon", "Tue", "Wed"
    , "Thur", "Fri", "Sat" };
40              JPanel p1 = new JPanel(new GridLayout(7, 7));
41              p1.setBackground(new Color(204,229,255));
42              p1.setPreferredSize(new Dimension(430, 120));
43              logger.info("Building calendar...");
44              for (int x = 0; x < button.length; x++) {
45                  final int selection = x;
46                  button[x] = new JButton();
47                  //add event to button
48                  button[x].setFocusPainted(false);
49                  button[x].setBackground(new Color(204,229,
    255));
50                  if (x > 6)
51                      button[x].addActionListener(new
    ActionListener() {
52                          public void actionPerformed(
    ActionEvent ae) {
53                              day = button[selection].
    getActionCommand();
54                              int choice = JOptionPane.
    showOptionDialog(null,
    "                                           Date Selected
     : "+setPickedDate(), "Choice Message",JOptionPane.
    DEFAULT_OPTION, JOptionPane.PLAIN_MESSAGE, null, options,
    options[2]);
55                              if (choice == 0 )//Make
    Appointment
56                              {
57                                  visibleOffForm();
58                                  Management.getManage().
    showMakeApp(setPickedDate());
59                                  logger.info("Make
    appointment");
60                              }
61                              else if(choice==1)//Show all
    appointments
62                              {
63                                  visibleOffForm();
64                                  Management.getManage().
    showTableForm(setPickedDate());
65                                  logger.info("Preparing show
     all appointments...");
66                              }
67                              else//cancel
```

```
 68                                  {
 69                                      visibleOffForm();
 70                                      Management.getManage().
    showCalendarForm();
 71
 72                                  }
 73                              }
 74                          });
 75                  if (x < 7) {
 76                      button[x].setText(header[x]);
 77                      button[x].setForeground(new Color(0,
    128,255));
 78                  }
 79                  p1.add(button[x]);
 80              }
 81              JPanel p2 = new JPanel(new GridLayout(1, 3));
 82              p2.setBackground(new Color(204,229,255));
 83              JButton previous = new JButton("<< Previous");
 84              previous.setBackground(new Color (153,204,255
    ));
 85              previous.addActionListener(new ActionListener
    () {
 86                  public void actionPerformed(ActionEvent ae
    ) {
 87                      month--;
 88                      displayDate();
 89                  }
 90              });
 91              p2.add(previous);
 92              p2.add(l);
 93              JButton next = new JButton("Next >>");
 94              next.setBackground(new Color (153,204,255));
 95              next.addActionListener(new ActionListener() {
 96                  public void actionPerformed(ActionEvent ae
    ) {
 97                      month++;
 98                      displayDate();
 99                  }
100              });
101              p2.add(next);
102              frame.add(p1, BorderLayout.CENTER);
103              frame.add(p2, BorderLayout.SOUTH);
104              frame.pack();
105              displayDate();
106
107          }
```

```
108
109     /**
110      * Show the form
111      */
112     @Override
113         public void showForm()
114         {
115             frame.setTitle("Calendar Form");
116             frame.setVisible(true);
117             frame.setSize(450,600);
118             frame.setDefaultCloseOperation(JFrame.
    EXIT_ON_CLOSE);
119
120         }
121
122     /**
123      * Hide the form
124      */
125     @Override
126     public void visibleOffForm()
127         {
128             frame.setVisible(false);
129         }
130
131     /**
132      * Get the array of button
133      * @return array button
134      */
135     public JButton[] getButton() {
136             return button;
137         }
138
139     /**
140      * Display the date
141      */
142     public void displayDate()
143         {
144             for (int x = 7; x < button.length; x++)
145                 button[x].setText("");
146             java.text.SimpleDateFormat sdf = new java.text
    .SimpleDateFormat(
147                     "MMMM yyyy");
148             java.util.Calendar cal = java.util.Calendar.
    getInstance();
149             cal.set(year, month, 1);
150             int dayOfWeek = cal.get(java.util.Calendar.
```

```
150 DAY_OF_WEEK);
151             int daysInMonth = cal.getActualMaximum(java.
    util.Calendar.DAY_OF_MONTH);
152             for (int x = 6 + dayOfWeek, day = 1; day <=
    daysInMonth; x++, day++)
153                 button[x].setText("" + day);
154             l.setText(sdf.format(cal.getTime()));
155         }
156 
157     /**
158      * Set the date
159      * @return - time
160      */
161     public String setPickedDate() {
162             if (day.equals(""))
163                 return day;
164             java.text.SimpleDateFormat sdf = new java.text
    .SimpleDateFormat(
165                     "dd-MM-yyyy");
166             java.util.Calendar cal = java.util.Calendar.
    getInstance();
167             cal.set(year, month, Integer.parseInt(day));
168             logger.info("Date was picked");
169             return sdf.format(cal.getTime());
170         }
171 }
172 
173 
```

```java
package il.ac.hit.view;
/*
NO TEST NEEDED BECAUSE THE METHODS ARE ALL ABOUT SHOW AND EDIT THE FORM
 */
import il.ac.hit.exceptions.AddException;
import il.ac.hit.viewmodel.Management;
import org.jetbrains.annotations.NotNull;

import javax.swing.*;
import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.util.logging.Logger;

public class MakeAppointmentForm extends  JFrame implements ActionListener, IView {

/**
This class displays the form of adding a new appointment to the barber where you can set a date, time, customer name, hairstyle type and phone number.
This class includes methods such as:
* Show/hide form - show/hide the appointment form to/from the main screen.
* Set location of the components.
* Add action listeners and action events.
* Save button that insert the appointment to the table.
 */
    static Logger logger= Logger.getLogger(String.valueOf(MakeAppointmentForm.class));
    Container frame = getContentPane();
    JTextField tfClientName = new JTextField(10);
    JTextField tfClientTel = new JTextField(10);
    JTextField tfClientType = new JTextField(10);
    JButton btSave = new JButton("Save");
    JButton btBack = new JButton("Back");
    JLabel clientNameLabel = new JLabel("Name: ");
    JLabel clientTelLabel = new JLabel("Tel: ");
    JLabel clientTypeLabel = new JLabel("Type: ");
    JLabel hourLabel = new JLabel("Hour: ");
    JLabel dateLabel;
    String[] m_Hours = {"08:00","08:30","09:00","09:30","10:00",
                        "10:30","11:00","11:30","12:00","12:30",
```

```
39                          "13:00","13:30","14:00","14:30","15
   :00",
40                          "15:30","16:00","16:30","17:00","17
   :30",
41                          "18:00","18:30","19:00","19:30","20
   :00"};
42
43      JComboBox hourComboBox = new JComboBox(m_Hours);
44      String date;
45      Appointment appointment =new Appointment("","","","","","
   );
46
47      /**
48       * Constructor - Create appointment form
49       * @param date - getting the picked date
50       */
51      public MakeAppointmentForm(@NotNull String date)
52      {
53          dateLabel = new JLabel(date);
54          appointment.setDate(date);
55          dateLabel.setFont(new Font("Serif", Font.PLAIN, 20
   ));
56          hourComboBox.setBackground(new Color(153,204,255));
57          btSave.setBackground(new Color(153,204,255));
58          btBack.setBackground(new Color(153,204,255));
59          setLayoutManager();
60          setLocationAndSize();
61          addComponentsToContainer();
62          addActionEvent();
63      }
64
65      /**
66       * Show the form
67       */
68      @Override
69      public void showForm()
70      {
71          appointment.setDate(date);
72          dateLabel.setText(appointment.getDate());
73          setTitle("MakeAppointment");
74          setSize(400, 600);
75          setVisible(true);
76          setResizable(false);
77          logger.info("Show make appointment form");
78          setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
79      }
```

```
 80
 81      /**
 82       * Hide the form
 83       */
 84      @Override
 85      public void visibleOffForm()
 86      {
 87          setVisible(false);
 88          tfClientName.setText("");
 89          tfClientTel.setText("");
 90          tfClientType.setText("");
 91      }
 92      /**
 93       * Set layout
 94       */
 95
 96      private void setLayoutManager() {
 97          frame.setLayout(null);
 98          frame.setBackground(new Color(204,229,255));
 99      }
100
101      /**
102       * Set location and size in frame
103       */
104      private void setLocationAndSize() {
105          dateLabel.setBounds(150, 50, 100, 70);
106          clientNameLabel.setBounds(50, 150, 100, 30);
107          clientTypeLabel.setBounds(50, 200, 100, 30);
108          clientTelLabel.setBounds(50, 250, 100, 30);
109          hourLabel.setBounds(50, 300, 100, 30);
110          tfClientName.setBounds(150, 150, 150, 30);
111          tfClientType.setBounds(150, 200, 150, 30);
112          tfClientTel.setBounds(150, 250, 150, 30);
113          hourComboBox.setBounds(150,300,100,30);
114          btSave.setBounds(50, 430, 100, 30);
115          btBack.setBounds(200, 430, 100, 30);
116      }
117
118      /**
119       * Add components
120       */
121      private void addComponentsToContainer() {
122          frame.add(dateLabel);
123          frame.add(clientNameLabel);
124          frame.add(clientTypeLabel);
125          frame.add(clientTelLabel);
```

```
126         frame.add(hourLabel);
127         frame.add(hourComboBox);
128         frame.add(tfClientName);
129         frame.add(tfClientType);
130         frame.add(tfClientTel);
131         frame.add(btSave);
132         frame.add(btBack);
133     }
134 
135     /**
136      * Add action listeners
137      */
138     private void addActionEvent()
139     {
140         btSave.addActionListener(this);
141         btBack.addActionListener(this);
142     }
143     /**
144      *  Action performed in event
145      * @param e - the event that occurred
146      */
147     @Override
148     public void actionPerformed(ActionEvent e) {
149         if (e.getSource() == btSave) {
150             addAppointment();
151             try {
152                 if(Management.getManage().
    addNewAppointment(appointment))//if return false - the
    date and hour that was selected is occupied.
153                 {
154                     logger.info("Appointment added");
155                     JOptionPane.showMessageDialog(this, "
    Saved!");
156                 }
157                 else JOptionPane.showMessageDialog(this, "
    Hour is occupied");
158             } catch (AddException ex) {
159                 ex.printStackTrace();
160             }
161             logger.info("Hour is occupied");
162 
163         }
164          if (e.getSource() == btBack) {
165             visibleOffForm();
166             Management.getManage().showCalendarForm();
167         }
```

```
168     }
169
170     /**
171      * Set the appointment details
172      */
173     public void addAppointment()
174     {
175         appointment.setHour((String)hourComboBox.getItemAt
    (hourComboBox.getSelectedIndex()));
176         appointment.setCustomerName(tfClientName.getText
    ());
177         appointment.setTypeHairCut(tfClientType.getText
    ());
178         appointment.setCustomerTel(tfClientTel.getText());
179         logger.info("Setting the new appointment");
180     }
181
182     /**
183      * Set the date
184      * @param date updated date, Can't be null
185      */
186     public void setDate(String date) {
187         this.date = date;
188     }
189 }
190
191
```

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

import il.ac.hit.exceptions.AddException;
import il.ac.hit.exceptions.DeleteException;
import org.jetbrains.annotations.NotNull;

import java.sql.ResultSet;

/**
 * Interface for IModel
 */
public interface IModel {
    /**
     * Create new table in database if not exist (table)
     * @param nameOfTable name of the table to create, Can't be null
     * @param parameters parameters that include the columns names and size of every argument, Can't be null
     * @throws Exception General exception
     */
    public void createTableIfNotExist(String nameOfTable, String... parameters) throws Exception;

    /**
     * Add a value to specific table in database
     * @param nameOfTable name of the table, Can't be null
     * @param parameters parameters by order to insert to the table, Can't be null
     * @return true if success, false if not
     * @throws AddException specific exception for add new value to database tables
     */
    public boolean addValue(@NotNull String nameOfTable,@NotNull String parameters) throws AddException;

    /**
     *
     * @param nameOfTable delete a value from a specific table in database, Can't be null
     * @param date date to delete, Can't be null
     * @param hour hour to delete, Can't be null
     * @return true id success, false if not
     * @throws DeleteException specific exception for delete value from database tables
     */
    public boolean deleteValue(@NotNull String nameOfTable
```

```
38 ,@NotNull String date,@NotNull String hour) throws
   DeleteException;
39
40     /**
41      * Build and execute SQL query to the database
42      * @param query string query to execute, Can't be null
43      * @return result set came from the database that
   contain the information that requested
44      */
45     public ResultSet query(@NotNull String query);
46
47     /**
48      * Delete table from database
49      * @param nameOfTable name of the table to delete, Can'
   t be null
50      * @throws DeleteException specific exception for
   delete value to database tables
51      */
52     public void deleteTable(@NotNull String nameOfTable)
   throws DeleteException;
53
54
55 }
56
```

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

import il.ac.hit.exceptions.AddException;
import il.ac.hit.exceptions.DataBaseException;
import org.jetbrains.annotations.NotNull;

import java.sql.ResultSet;
import java.sql.SQLException;
import java.util.logging.Logger;

public class UserTools implements IUserTools{

/**
This class makes contact with the Data Base with regards to the user, receiving, removing and adding users to the Data Base
This class include methods such as:
* addUserName-add user to database.
* createTableIfNotExist-this method check if the user name in data base is exist.
*/
    private IModel dataBase;
    private final String nameOfTable = "User1";
    static Logger logger=Logger.getLogger(String.valueOf(UserTools.class));

    /**
     * Constructor create some user tool to use it in data base
     */
    public UserTools()
    {
        try {
            this.dataBase = DataBaseModel.getInstance();
            dataBase.createTableIfNotExist(
                    nameOfTable,
                    "UserName",
                    "varchar(300)",
                    "Password",
                    "varchar(300)");
            logger.info("Creating Table");
        } catch (Exception e) {
            e.printStackTrace();
        }
    }

```

```
42      /**
43       * Add user to data base.
44       * @param userName username to add query, Can't be null
45       * @param password password to add query, Can't be null
46       * @return true if success, false if not
47       * @throws AddException specific exception to for new
   value to database tables
48       */
49      @Override
50      public boolean addUserName(@NotNull String userName,@
   NotNull String password) throws AddException {
51          String parameters =
52                  "(" +"'"+userName+"'" +
53                          "," +"'"+password+"')";
54
55          dataBase.addValue(nameOfTable, parameters);
56          logger.info("Add new user");
57          return true;
58      }
59
60      /**
61       * Check if the user exist.
62       * @param userName username to check, Can't be null
63       * @param password password to check, Can't be null
64       * @return true if exist, false if not
65       */
66      @Override
67      public boolean checkIfUserExist(@NotNull String
   userName,@NotNull String password)
68      {
69          boolean valToReturn = false;
70          try {
71          ResultSet rs = dataBase.query("select * from "+
   nameOfTable +" where "+ nameOfTable +".USERNAME ='"+
   userName+"' and "+
72                  nameOfTable +".PASSWORD ='"+password+"'");
73              if(rs.next())
74              {
75                  logger.info("User is exist");
76                  return valToReturn = true;
77              }
78          } catch (SQLException e) {
79              e.printStackTrace();
80              logger.info("Failed finding user existence");
81              throw new DataBaseException("Create failed");
82          }
```

```
83          finally {
84              return valToReturn;
85          }
86      }
87
88      /**
89       * Set model chose.
90       * @param m class that implement IModel interface, Can
   't be null
91       */
92      @Override
93      public void setModel(@NotNull IModel m){
94          this.dataBase = m;
95      }
96  }
97
```

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

import il.ac.hit.exceptions.AddException;
import org.jetbrains.annotations.NotNull;

/**
 * interface for user tools.
 */
public interface IUserTools {
    /**
     * Add new user to the system using user table in database
     * @param userName user name to add, Can't be null
     * @param password password to add, Can't be null
     * @return true if add success, false if not
     * @throws AddException specific exception to for new value to database tables
     */
    public boolean addUserName(@NotNull String userName,@NotNull String password) throws AddException;

    /**
     * Check if user exist in user table in database
     * @param userName user name to check, Can't be null
     * @param password password to check, Can't be null
     * @return true is exist, false if not
     */
    public boolean checkIfUserExist(@NotNull String userName,@NotNull String password);

    /**
     * Set the model (class that implements IModel interface)
     * @param m class that implements IModel interface, Can't be null
     */
    public void setModel(@NotNull IModel m);
}


```

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

import il.ac.hit.exceptions.AddException;
import il.ac.hit.exceptions.DataBaseException;
import il.ac.hit.exceptions.DeleteException;
import org.jetbrains.annotations.NotNull;

import java.sql.DatabaseMetaData;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;
import java.util.logging.Logger;

public class DataBaseModel implements IModel{

/**
This class uses the data base connection class and sends SQL queries and data management.
This class use in Singleton design pattern.
This class include methods such as:
* query-send the questions to database and get response.
* addValue-insert appointment to the system.
* deleteValue-delete appointment from the system.
* createTableIfNotExist-this method check if the table in data base is exist.
 */

    private static Object lock = new Object();
    static Logger logger=Logger.getLogger(String.valueOf(DataBaseModel.class));
    private static DataBaseModel s_Instance;
    private DataBaseConnection mainDataBase = DataBaseConnection.getInstance();
    private Statement statement = DataBaseConnection.getInstance().getStatement();
    private DatabaseMetaData metaData = null;

    /**
     * Constructor - database model that implements IModel interface
     */
    private DataBaseModel() {
        try {
            metaData = mainDataBase.getConnection().getMetaData();
        } catch (SQLException e) {
```

```
40              e.printStackTrace();
41          }
42      }
43
44      public static DataBaseModel getInstance() {
45
46          if (s_Instance==null) {
47              synchronized (lock)
48              {
49                  if (s_Instance == null)
50                  {
51                      s_Instance = new DataBaseModel();
52                  }
53              }
54          }
55          return s_Instance;
56      }
57
58      /**
59       * Execute sql query.
60       * @param query string edited query to database, Can't
   be null
61       * @return result set that contains the requested
   information
62       */
63      @Override
64      public ResultSet query(@NotNull String query){
65          ResultSet resultSet=null;
66
67          try
68          {
69              resultSet = statement.executeQuery(query);
70              logger.info("Query executed successfully ");
71          }
72          catch (SQLException e)
73          {
74              logger.info("Query execute failed");
75              e.printStackTrace();
76          }
77          return  resultSet;
78      }
79
80      /**
81       * Add value with the help of sql query
82       * @param nameOfTable name of the table in the database
   , Can't be null
```

```
 83      * @param parameters parameters to insert, Can't be
    null
 84      * @return true if success, false if not
 85      * @throws AddException specific exception for add new
     value to database tables
 86      */
 87     //
 88     @Override
 89     public boolean addValue(@NotNull String nameOfTable,@
    NotNull String parameters) throws AddException {
 90         boolean valToReturn=false;
 91         try {
 92             statement.executeUpdate("insert into "+
    nameOfTable.toUpperCase()+" values "+parameters);
 93             logger.info("Insert Value");
 94             valToReturn = true;
 95         }catch (Exception e)
 96         {
 97             e.printStackTrace();
 98             logger.info("Insert value failed");
 99             throw new AddException("Added Failed",e);
100         }
101         finally {
102             return valToReturn;
103         }
104     }
105
106     /**
107      * Delete value with the help of sql query.
108      * @param nameOfTable name of the table in database,
    Can't be null
109      * @param date date to the query, Can't be null
110      * @param hour hour ti the query, Can't be null
111      * @return true if success, false if not
112      * @throws DeleteException specific exception for
    delete value from database tables
113      */
114     @Override
115     public boolean deleteValue(@NotNull String nameOfTable
    ,@NotNull String date,@NotNull String hour) throws
    DeleteException
116     {
117         boolean valToReturn = false;
118         try {
119             statement.executeUpdate("delete from "+
    nameOfTable+" where "+nameOfTable+".Date ='"+date+"' and "
```

```
119 +
120                     nameOfTable+".Time ='"+hour+"'");
121             logger.info("Delete success");
122             valToReturn = true;
123         }catch (Exception e)
124         {
125             e.printStackTrace();
126             logger.info("Delete Failed");
127             throw new DeleteException("Delete failed");
128         }
129         finally {
130 
131             return valToReturn;
132         }
133     }
134 
135     /**
136      * Create new table in database
137      * @param nameOfTable name of the table to create, Can
    't be null
138      * @param parameters parameters of the table, Can't be
     null
139      * @throws Exception General exception
140      */
141     @Override
142     public void createTableIfNotExist(@NotNull String
    nameOfTable,@NotNull String... parameters) throws
    Exception {
143         if (parameters.length % 2 != 0) {
144             throw new DataBaseException
145                     ("Please enter the parameters as
    follows!");
146         }
147         String str = stringsLinking(parameters);
148         try {
149             ResultSet rs = metaData.getTables(null, "APP"
    , nameOfTable.toUpperCase(), null);
150             if (!rs.next()) {
151                 statement.execute("create table " +
    nameOfTable + str);
152                 logger.info("Creating Table");
153             }
154         } catch (SQLException e) {
155             e.printStackTrace();
156             logger.info("Failed creating table in DB");
157             throw new DataBaseException("creating Failed",
```

```
157 e);
158             }
159         }
160 
161         /**
162          * Linking the params in database table.
163          * @param strings parameters to the query, Can't be
    null
164          * @return StringBuffer.toString() - edited string
    ready to query
165          */
166         private String stringsLinking(@NotNull String...
    strings) {
167             StringBuffer stringBuffer = new StringBuffer();
168             int length = strings.length - 1;
169             stringBuffer.append("(");
170             for (int i = 0; i < length; i += 2) {
171                 stringBuffer.append(strings[i]);
172                 stringBuffer.append(" ");
173                 stringBuffer.append(strings[i + 1]);
174                 if (i < length - 1) {
175                     stringBuffer.append(", ");
176                 }
177             }
178             stringBuffer.append(")");
179             return stringBuffer.toString();
180         }
181 
182         /**
183          * Delete table from database
184          * @param nameOfTable name of the table to delete, Can
    't be null
185          * @throws DeleteException delete table
186          */
187         @Override
188         public void deleteTable(@NotNull String nameOfTable)
    throws DeleteException {
189             try {
190                 statement.execute("DROP TABLE "+nameOfTable);
191             } catch (SQLException e) {
192                 e.printStackTrace();
193                 throw new DeleteException("Table delete
    problem",e);
194             }
195         }
196 }
```

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

import il.ac.hit.exceptions.AddException;
import il.ac.hit.exceptions.DeleteException;
import il.ac.hit.view.Appointment;
import org.jetbrains.annotations.NotNull;

import java.sql.ResultSet;
import java.sql.SQLException;
import java.util.logging.Logger;

public class AppointmentTools implements IAppointmentTools{

/**
This class is in contact with the Data Base when it comes to setting queues, receiving, removing and adding queues
This class include methods such as:
* getAppointment-response the answer from database.
* addNewAppointment-add the appointment to data base.
* removeAppointment-delete appointment from the data base.
* createTableIfNotExist-this method check if the appointment data base is exist.
*/
    private IModel dataBase;
    private final String nameOfTable = "Appointment";
    static Logger logger=Logger.getLogger(String.valueOf(AppointmentTools.class));

    /**
     * Constructor create some appointment tool to use it in data base
     */
    public AppointmentTools()
    {
        try {
            this.dataBase = DataBaseModel.getInstance();
            dataBase.createTableIfNotExist(
                    nameOfTable,
                    "Date",
                    "varchar(300)",
                    "Time",
                    "varchar(300)",
                    "Name",
                    "varchar(300)",
                    "Type",
                    "varchar(300)",
```

```
43                      "Telephone",
44                      "varchar(300)");
45              logger.info("Creating Table");
46          } catch (Exception e) {
47              e.printStackTrace();
48              logger.info("Failed creating table");
49          }
50      }
51
52      /**
53       * Get the appointment from data base.
54       * @param date date picked to the query, Can't be null
55       * @return ResultSet result set from data base with the
     date request
56       */
57      public ResultSet getAppointments(@NotNull String date){
58              ResultSet rs = dataBase.query("SELECT * FROM "
     + nameOfTable + " WHERE " + nameOfTable + ".Date ='" +
    date +"'"+ "ORDER BY Time");
59              logger.info("Getting all appointments...");
60              return rs;
61      }
62
63      /**
64       * Add appointment to data base.
65       * @param appointment getting the appointment to add,
    Can't be null
66       * @return true if success, false if not
67       * @throws AddException specific exception for new
    value to database tables
68       */
69      @Override
70      public boolean addNewAppointment(@NotNull Appointment
    appointment) throws AddException {
71          String parameters =
72                  "(" +"'"+appointment.getDate()+"'" +
73                          "," +"'"+appointment.getHour()+"'"
     +
74                          "," +"'"+appointment.
    getCustomerName()+"'" +
75                          "," +"'"+appointment.getTypeHairCut
    ()+"'" +
76                          "," +"'"+appointment.getCustomerTel
    ()+"')";
77
78          dataBase.addValue(nameOfTable, parameters);
```

```
 79          logger.info("Adding new appointment");
 80          return true;
 81      }
 82
 83      /**
 84       * Remove appointment from data base.
 85       * @param date date for the delete query, Can't be
    null
 86       * @param hour hour for the delete query, Can't be
    null
 87       * @return true if success, false if not
 88       * @throws DeleteException specific exception for
    delete value to database tables
 89       */
 90      @Override
 91      public boolean removeAppointment(@NotNull String date
    ,@NotNull String hour) throws DeleteException {
 92          dataBase.deleteValue(nameOfTable,date,hour);
 93          logger.info("Deleting appointment");
 94          return true;
 95      }
 96
 97      /**
 98       * Check if the appointment exist.
 99       * @param date date for the query, Can't be null
100       * @param hour hour for the query, Can't be null
101       * @return true if exist, false if not
102       */
103      public boolean checkIfExist(@NotNull String date,@
    NotNull String hour){
104          ResultSet rs = dataBase.query("select * from "+
    nameOfTable +" where "+ nameOfTable +".Date ='"+date+"'
    and "+
105                  nameOfTable +".Time ='"+hour+"'");
106          boolean valToReturn = false;
107          try {
108              if(rs.next())
109              {
110                  logger.info("Appointment exist");
111                  return valToReturn = true;
112              }
113          } catch (SQLException e) {
114              e.printStackTrace();
115              logger.info("Failed checking if appointment is
     exist");
116          }
```

```
117             finally {
118                 return valToReturn;
119             }
120         }
121 
122         /**
123          * Set model chose.
124          * @param m class that implement IModel interface, Can
    't be null
125          */
126         @Override
127         public void setModel(@NotNull IModel m)
128         {
129             this.dataBase = m;
130         }
131 }
132 
```

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

import il.ac.hit.exceptions.AddException;
import il.ac.hit.exceptions.DeleteException;
import il.ac.hit.view.Appointment;
import org.jetbrains.annotations.NotNull;

/**
 * Interface for appointment tools
 */
public interface IAppointmentTools {
    /**
     * Add new appointment to the appointment table in database
     * @param appointment appointment to add, Can't be null
     * @return true if success, false if not
     * @throws AddException specific exception for new value to database tables
     */
    public boolean addNewAppointment(@NotNull Appointment appointment) throws AddException;

    /**
     *
     * @param date date to remove, Can't be null
     * @param hour hour to remove, Can't be null
     * @return true if success false if not
     * @throws DeleteException specific exception for delete value to database tables
     */
    public boolean removeAppointment(@NotNull String date,@NotNull String hour) throws DeleteException;

    /**
     * Set the model (class that implements IModel interface)
     * @param m class that implements IModel interface, Can't be null
     */
    public void setModel(IModel m);
}

```

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

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;
import java.sql.Statement;
import java.util.logging.Logger;


public class DataBaseConnection {

/**
* This class is the connection to data base with the help of drive and protocol. .
* This class include methods such as:
* Getters statements and connection.
*/

    private static DataBaseConnection s_Instance;
    private static Object s_Lock = new Object();
    static Logger logger=Logger.getLogger(String.valueOf(DataBaseConnection.class));

    private final String drive = "org.apache.derby.jdbc.EmbeddedDriver";
    private final String protocol = "jdbc:derby:BarberShopDB;create=true";
    private Connection connection = null;
    private Statement statement = null;

    /**
     *  Constructor - connect to the database using driver and protocol
     */
    private DataBaseConnection()  {
        try {
            Class.forName(drive);
            connection = DriverManager.getConnection(protocol);
            statement = connection.createStatement();
            logger.info("Connection to DB server was success");
        } catch (ClassNotFoundException e) {
            e.printStackTrace();
            logger.info("Connection to DB failed");
        } catch (SQLException e) {
```

```
40              e.printStackTrace();
41          }
42      }
43
44      /**
45       *  Get instance of DataBaseCreator implements
   singleton
46       * @return s_Instance
47       */
48      public static DataBaseConnection getInstance()
49      {
50          if (s_Instance==null)
51          {
52              synchronized (s_Lock)
53              {
54                  if (s_Instance == null)
55                  {
56                      s_Instance = new DataBaseConnection();
57                  }
58              }
59          }
60          return s_Instance;
61      }
62
63      /**
64       * Get Statement
65       * @return statement variable
66       */
67      public Statement getStatement()
68      {
69          return statement;
70      }
71
72      /**
73       * Get Connection
74       * @return connection variable
75       */
76      public Connection getConnection()
77      {
78          return connection;
79      }
80  }
81
```

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

import il.ac.hit.model.IAppointmentTools;
import il.ac.hit.model.IModel;
import il.ac.hit.model.IUserTools;

/**
 * interface for View Model that extends IUserTools and IAppointmentTools interfaces.
 * This interface also manage the forms change using the "showForm" method from IView interface.
 */
public interface IViewModel extends IUserTools, IAppointmentTools {
    /**
     * Set the model (class that implements IModel interface)
     * @param m class that implements IModel interface
     */
    public void setModel(IModel m);
}

```

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

import il.ac.hit.model.*;
import il.ac.hit.exceptions.AddException;
import il.ac.hit.exceptions.DeleteException;
import il.ac.hit.view.*;

import javax.swing.*;
import java.util.logging.Logger;

public class Management implements IViewModel {

    /**
    This class manages the program (VM), selects the order of forms and links them to the model (DataBaseModel) and classes that assist it.
    Through this class we can communicate with almost all the existing classes in the program.
    This class includes methods such as:
    * Show/hide all forms with its needs.
    * check if user exist
    * Add new user or appointment.
    * Delete appointment using Appointment tools class.
     */

    private static Object lock = new Object();
    static Logger logger= Logger.getLogger(String.valueOf(MakeAppointmentForm.class));
    private static Management s_Manage=null;
    private LoginForm loginForm;
    private CalendarForm calendarForm;
    private TableForm tableForm;
    private SignUpForm signUpForm;
    public UserTools userTools;
    public AppointmentTools appointmentTools;
    public MakeAppointmentForm makeAppointmentForm;
    public String userName;
    public IModel model;
    public IView view;

    /**
     * Constructor - Create the management between UI to Data base and inverse
     */
    public Management(){
        loginForm = new LoginForm();
```

```
42          calendarForm = new CalendarForm();
43          tableForm = new TableForm("");
44          signUpForm =new SignUpForm();
45          makeAppointmentForm =new MakeAppointmentForm("");
46          userTools = new UserTools();
47          appointmentTools = new AppointmentTools();
48          userTools.setModel(DataBaseModel.getInstance());
49          appointmentTools.setModel(DataBaseModel.getInstance
    ());
50      }
51
52      public static Management getManage() {
53          if (s_Manage == null)
54          {
55              synchronized (lock)
56              {
57                  if (s_Manage == null)
58                  {
59                      s_Manage = new Management();
60                  }
61              }
62          }
63
64          return s_Manage;
65      }
66
67      /**
68       * Show make appointment form
69       * @param date getting the picked date
70       */
71      public void showMakeApp(String date)
72      {
73          SwingUtilities.invokeLater(new Runnable() {
74              @Override
75              public void run()
76              {
77                  makeAppointmentForm.setDate(date);
78                  makeAppointmentForm.showForm();
79
80              }
81          });
82      }
83
84      /**
85       * Show sign Up form
86       */
```

```
 87      public void showSignUpForm()
 88      {
 89          SwingUtilities.invokeLater(new Runnable() {
 90              @Override
 91              public void run()
 92              {
 93                   signUpForm.showForm();
 94                   logger.info("Showing form..");
 95
 96              }
 97          });
 98      }
 99
100      /**
101       * Show calendar form
102       */
103      public void showCalendarForm()
104      {
105          SwingUtilities.invokeLater(new Runnable() {
106              @Override
107              public void run()
108              {
109                  calendarForm.showForm();
110                  logger.info("Showing form..");
111
112              }
113          });
114      }
115
116      /**
117       * Show Login Form
118       */
119      public void showLoginForm()
120      {
121          SwingUtilities.invokeLater(new Runnable() {
122              @Override
123              public void run()
124              {
125                  loginForm.showForm();
126                  logger.info("Showing form..");
127              }
128          });
129      }
130
131      /**
132       * Start the program
```

```
133      */
134     public void startProgram()
135     {
136         SwingUtilities.invokeLater(new Runnable() {
137             @Override
138             public void run()
139             {
140                 logger.info("Starting program...");
141                 loginForm.showForm();
142 
143 
144             }
145         });
146     }
147 
148     /**
149      * Show appointments table form
150      * @param date getting the picked date
151      */
152     public void showTableForm(String date)
153     {
154         SwingUtilities.invokeLater(new Runnable() {
155             @Override
156             public void run()
157             {
158 
159                 try {
160                     tableForm.setRs(appointmentTools.
    getAppointments(date));
161                     tableForm.setDate(date);
162                     tableForm.showForm();
163                     logger.info("Showing form..");
164                 } catch (Exception e) {
165                     e.printStackTrace();
166                     logger.info("Error on show table form"
    );
167 
168                 }
169 
170             }
171         });
172     }
173 
174     /**
175      *
176      * @param userName username to check
```

```
177      * @param password password to check
178      * @return true if exist, false if not
179      */
180     public boolean checkIfUserExist(String userName,
    String password) {
181         boolean valueToReturn = false;
182 
183         if (userTools.checkIfUserExist(userName, password
    )) //checking with user tools class that communicate with
    the model
184         {
185             valueToReturn = true;
186             this.userName = userName;
187             logger.info("Checking if user exist...");
188         }
189 
190         return valueToReturn;
191     }
192 
193     /**
194      *
195      * @param userName username to add
196      * @param password password to add
197      * @return true if success, false if not
198      * @throws AddException specific exception for new
    value to database tables
199      */
200         public boolean addUserName(String userName, String
     password) throws AddException {
201 
202         if(!userTools.checkIfUserExist(userName,password
    )) //NOT user existence
203         {
204             if(userTools.addUserName(userName, password));
205                 logger.info("Registering...");
206                 return true;
207         }
208         else return false;
209     }
210 
211     /**
212      * Add new appointment using appointment tools class
    that communicate with database model
213      * @param appointment appointment to add
214      * @return true if success, false if not
215      * @throws AddException specific exception for new
```

```
215 value to database tables
216      */
217     public boolean addNewAppointment(Appointment
    appointment) throws AddException
218     {
219         if (!(appointmentTools.checkIfExist(appointment.
    getDate(), appointment.getHour())))
220         {
221             appointmentTools.addNewAppointment(appointment
    );
222             logger.info("Appointment added");
223             return true;
224         }
225         else {
226             logger.info("Failed to add new appointment");
227             return false;
228         }
229     }
230 
231     /**
232      * Delete appointment using appointment tools class
    that communicate with database model
233      * @param date date to the query
234      * @param hour hour to the query
235      * @return true if success, false if not
236      * @throws DeleteException specific exception for
    delete value to database tables
237      */
238     public boolean removeAppointment(String date,String
    hour) throws DeleteException
239     {
240         logger.info("Deleting appointment...");
241         appointmentTools.removeAppointment(date, hour);
242         return true;
243     }
244 
245     /**
246      * Set the database model
247      * @param m class that implements IModel interface
248      */
249     @Override
250     public void setModel(IModel m)
251     {
252         this.model = m;
253     }
254 
```

```
255     /**
256      *Set view
257      * @param v - class that implements IView interface
258      */
259 }
260
```

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

/**
 *
 */
public class AddException extends Exception {

    /**
     *
     */
    public AddException(){
        super();
    }

    public AddException(String msg,Throwable d)
    {
        super(msg,d);
    }

    public AddException(String msg)
    {
        super(msg);
    }
}

```

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

public class DeleteException extends Exception {

    public DeleteException(){
        super();
    }

    public DeleteException (String msg,Throwable d)
    {
        super(msg,d);
    }

    public DeleteException (String msg)
    {
        super(msg);
    }
}

```

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

public class DataBaseException extends Exception{

    public DataBaseException(){
        super();
    }

    public DataBaseException (String msg,Throwable d)
    {
        super(msg,d);
    }

    public DataBaseException (String msg)
    {
        super(msg);
    }


}

```

File - D:\DUsers\JavaProject\BarberShopAppointmentManager\src\META-INF\MANIFEST.MF

```
1 Manifest-Version: 1.0
2 Main-Class: il.ac.hit.main
3
4
```