```
 1 <?xml version='1.0' encoding='utf
   -8'?>
 2 <!DOCTYPE hibernate-configuration
    PUBLIC "-//Hibernate/Hibernate
   Configuration DTD//EN"
 3         "http://hibernate.
   sourceforge.net/hibernate-
   configuration-3.0.dtd">
 4 <hibernate-configuration>
 5     <session-factory>
 6         <property name="hibernate
   .connection.driver_class">
 7             com.mysql.jdbc.Driver
 8         </property>
 9         <property name="hibernate
   .connection.url">
10             jdbc:mysql://
   localhost:3306/haim
11         </property>
12         <property name="hibernate
   .connection.username">haim</
   property>
13         <property name="hibernate
   .connection.password">michael</
   property>
14         <property name="show_sql"
   >true</property>
15         <property name="dialect">
```

```
15 org.hibernate.dialect.
   MySQLDialect</property>
16         <property name="hibernate
   .hbm2ddl.auto">update</property>
17         <!-- Mapping files -->
18         <mapping resource="
   storeinventory.hbm.xml"/>
19     </session-factory>
20 </hibernate-configuration>
```

```
 1 <?xml version="1.0"?>
 2 <!DOCTYPE hibernate-mapping
   PUBLIC
 3         "-//Hibernate/Hibernate
   Mapping DTD 3.0//EN"
 4         "http://hibernate.
   sourceforge.net/hibernate-mapping
   -3.0.dtd">
 5 <hibernate-mapping>
 6     <class name="il.ac.hit.model.
   Item" table="ITEMS">
 7         <id name="
   incrementalValue" type="int"
   column="INCREMENTALVALUE" >
 8             <generator class="
   increment"/>
 9         </id>
10
11         <property name="id">
12             <column name="ID"/>
13         </property>
14         <property name="userId">
15             <column name="USERID"
   />
16         </property>
17         <property name="cost">
18             <column name="COST"/>
19         </property>
```

```
20          <property name="date">
21              <column name="DATE"/>
22          </property>
23          <property name="type">
24              <column name="TYPE"/>
25          </property>
26      </class>
27
28      <class name="il.ac.hit.model.
    User" table="USERS">
29          <id name="userId" type="
    int" column="USERID" >
30              <generator class="
    assigned"/>
31          </id>
32
33          <property name="userName"
    >
34              <column name="
    USERNAME"/>
35          </property>
36          <property name="lastName"
    >
37              <column name="
    LASTNAME"/>
38          </property>
39          <property name="password"
    >
```

```
40                  <column name="
    PASSWORD"/>
41              </property>
42          </class>
43
44  </hibernate-mapping>
```

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

/**
 * this class represents the items in our project Cost Manager.
 */
public class Item {

    private int id;
    private int userId;
    private int cost;
    private String date;
    private String type;
    private int incrementalValue;

    /**
     * Empty constructor
     */
    public Item(){
    }

    /**
     * This is our program's Item constructor
     * @param id item's id
     * @param userId item's user id
```

```
25          * @param cost item's cost
26          * @param date item's date
27          * @param type item's type
28          * @throws ItemDAOException
    if database is corrupted
29          */
30         public Item(int id, int
    userId, int cost, String date,
    String type) throws
    ItemDAOException{
31             setId(id);
32             setUserId(userId);
33             setCost(cost);
34             setDate(date);
35             setType(type);
36         }
37
38         /**
39          * This is our program's Item
     constructor
40          * @param id item's id
41          * @param userId item of user
     id
42          * @param cost item's cost
43          * @param date item's date
44          * @param type item's type
45          * @param incrementalValue
    item's incremental value for
```

```
45 database
46      * @throws ItemDAOException
   if database is corrupted
47      */
48     public Item(int id, int
   userId, int cost, String date,
   String type, int incrementalValue
   ) throws ItemDAOException{
49         this(id,userId,cost,date,
   type);
50         setIncrementalValue(
   incrementalValue);
51     }
52
53     /**
54      * get incremental value of
   the item
55      * @return An int type
   containing the item's incremental
    Value.
56      */
57     public int
   getIncrementalValue() {
58         return incrementalValue;
59     }
60
61     /**
62      * Sets the item's
```

```
62 incrementalValue and checking
   validation
63       * @param incrementalValue An
    int type containing the item's
   incremental Value.
64       * @throws ItemDAOException
   if database is corrupted
65       */
66      public void
   setIncrementalValue(int
   incrementalValue) throws
   ItemDAOException {
67          String temp =
   incrementalValue + "";
68          try {
69              int checkedId =
   Integer.parseInt(temp); //if the
   exception was not thrown
70              this.incrementalValue
    = incrementalValue;
71          } catch (
   NumberFormatException ex) {
72              throw new
   ItemDAOException("wrong input
   type: incrementalValue");
73          }
74      }
75
```

```
76      /**
77       * get id of the item
78       * @return An int type
    containing the item's id.
79       */
80      public int getId() {
81          return id;
82      }
83
84      /**
85       * Sets the item's id and
    checking validation
86       * @param id An int type
    containing the item's id.
87       * @throws ItemDAOException
    if database is corrupted
88       */
89      public void setId(int id)
    throws ItemDAOException {
90          String temp = id + "";
91          try {
92              int checkedId =
    Integer.parseInt(temp); //if the
     exception was not thrown
93              this.id = id;
94          } catch (
    NumberFormatException ex) {
95              throw new
```

```
 95 ItemDAOException("wrong input
    type: id");
 96             }
 97         }
 98         /**
 99          * get userId of the item
100          * @return An int type
    containing the item's userId.
101          */
102         public int getUserId() {
103             return userId;
104         }
105
106
107         /**
108          * Sets the item's userId
    and checking validation
109          * @param userId An int type
     containing the item's userId.
110          * @throws ItemDAOException
    if database is corrupted
111          */
112         public void setUserId(int
    userId) throws ItemDAOException
     {
113             String temp = userId +
    "";
114             try {
```

```
115                 int checkedId =
    Integer.parseInt(temp); //if the
     exception was not thrown
116                 this.userId = userId
    ;
117             } catch (
    NumberFormatException ex) {
118                 throw new
    ItemDAOException("wrong input
    type: userId");
119             }
120         }
121
122         /**
123          * get cost of the item
124          * @return An int type
    containing the item's cost.
125          */
126         public int getCost() {
127             return cost;
128         }
129
130         /**
131          * Sets the item's cost and
    checking validation
132          * @param cost An int type
    containing the item's cost.
133          * @throws ItemDAOException
```

```
133 if database is corrupted
134      */
135     public void setCost(int cost
    ) throws ItemDAOException {
136         String temp = cost + "";
137         try {
138             int checkedCost =
    Integer.parseInt(temp); //if the
     exception was not thrown
139             this.cost = cost;
140         } catch (
    NumberFormatException ex) {
141             throw new
    ItemDAOException("wrong input
    type: id");
142         }
143     }
144
145     /**
146      * get date of the item
147      * @return A string type
    containing the item's date.
148      */
149     public String getDate() {
150         return date;
151     }
152
153     /**
```

```
154         * Sets the item's date and checking validation
155         * @param date A string type containing the item’s date.
156         * @throws ItemDAOException if database is corrupted
157         */
158        public void setDate(String date) throws ItemDAOException {
159            if (date.equals(" ") || date.length() == 0) {
160                throw new ItemDAOException("Empty string: date");
161            } else this.date = date;
162        }
163
164        /**
165         * get type of the item
166         * @return A string type containing the item’s type.
167         */
168        public String getType() {
169            return type;
170        }
171
172        /**
173         * Sets the item's type and
```

```
173 checking validation
174      * @param type A string type
     containing the item's type.
175      * @throws ItemDAOException
    if database is corrupted
176      */
177     public void setType(String
    type) throws ItemDAOException {
178         if (type.equals(" ") ||
    type.length() == 0) {
179             throw new
    ItemDAOException("Empty string:
    type");
180         } else this.type = type;
181     }
182
183 }
184
```

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

/**
 * this class represents the users in our project Cost Manager.
 */
public class User {

    private int userId;
    private String userName;
    private String lastName;
    private String password;

    /**
     * Empty constructor
     */
    public User() {
    }

    /**
     * This is our program's User constructor
     * @param userId user's userId
     * @param userName user's username
     * @param lastName user's
```

```
23 lastName
24      * @param password user's
   password
25      * @throws UserDAOException
   if database is corrupted
26      */
27     public User(int userId,
   String userName, String lastName
   , String password) throws
   UserDAOException {
28         setUserId(userId);
29         setUserName(userName);
30         setLastName(lastName);
31         setPassword(password);
32     }
33
34     /**
35      * get userId of the user
36      * @return An int type
   containing the item’s userId.
37      */
38     public int getUserId() {
39         return userId;
40     }
41
42
43     /**
44      * Sets the user's userId and
```

```
44   checking validation
45        * @param userId An int type
   containing the user's userId.
46        * @throws UserDAOException
   if database is corrupted
47        */
48       public void setUserId(int
   userId) throws UserDAOException {
49            String temp = userId + ""
   ;
50            try {
51                int checkedId =
   Integer.parseInt(temp); //if the
   exception was not thrown
52                this.userId = userId;
53            } catch (
   NumberFormatException ex) {
54                throw new
   UserDAOException("wrong input
   type: incrementalValue");
55            }
56       }
57
58       /**
59        * get userName of the user
60        * @return A string type
   containing the user's userName.
61        */
```

```
62      public String getUserName() {
63          return userName;
64      }
65
66      /**
67       * Sets the user's userName
    and checking validation
68       * @param userName A string
    type containing the user's
    userName.
69       * @throws UserDAOException
    if database is corrupted
70       */
71      public void setUserName(
    String userName) throws
    UserDAOException {
72          if (userName.equals(" "
    ) || userName.length() == 0) {
73              throw new
    UserDAOException("Empty string:
    date");
74          } else this.userName =
    userName;
75      }
76
77      /**
78       * get lastName of the user
79       * @return A string type
```

```
79  containing the user’s lastName.
80       */
81      public String getLastName
    () {
82          return lastName;
83      }
84
85      /**
86       * Sets the user's lastName
    and checking validation
87       * @param lastName A string
    type containing the user’s
    lastName.
88       * @throws UserDAOException
    if database is corrupted
89       */
90      public void setLastName(
    String lastName) throws
    UserDAOException {
91          if (lastName.equals(" "
    ) || lastName.length() == 0) {
92              throw new
    UserDAOException("Empty string:
    date");
93          } else this.lastName =
    lastName;
94      }
95
```

```
 96      /**
 97       * get password of the user
 98       * @return A string type
    containing the user's password.
 99       */
100      public String getPassword
    () {
101          return password;
102      }
103
104      /**
105       * Sets the user's password
    and checking validation
106       * @param password A string
    type containing the user's
    password.
107       * @throws UserDAOException
    if database is corrupted
108       */
109      public void setPassword(
    String password) throws
    UserDAOException {
110          if (password.equals(" "
    ) || password.length() == 0) {
111              throw new
    UserDAOException("Empty string:
    date");
112          } else this.password =
```

```
112 password;
113     }
114 }
115
```

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

import java.util.List;

/**
 * Represents the interface of HibernateCostManagerDAO class
 * gives more accessibility and flexibility to the code
 */
public interface ICostManagerDAO {

    /**
     * getting instance of session ,saving the object of the specific item to our database
     * if not succeed a rollback will be committed and an error message will be sent
     * finally closing the session upon completion
     * @param ob Item's object
     * @throws ItemDAOException if database is corrupted
```

```
20          */
21         public void addItem(Item ob)
   throws ItemDAOException;
22
23         /**
24          * getting instance of
   session.
25          * getting all items from
   database for this specific user's
    id,
26          * if not succeed a rollback
   will be committed and an error
   message will be sent
27          * finally will close the
   session upon completion
28          * @param userId getting user
   's id
29          * @return list of items of
   this specific user's id
30          * @throws ItemDAOException
   if database is corrupted
31          */
32         public List<Item> getItems(
   int userId) throws
   ItemDAOException;
33
34         /**
35          * getting instance of
```

```
35 session ,saving the object of the
    specific user to our database
36         * if not succeed a rollback
   will be committed and an error
   message will be sent
37         * finally closing the
   session upon completion
38         * @param ob User's object
39         * @throws UserDAOException
   if database is corrupted
40         */
41     public void addUser(il.ac.hit
   .model.User ob) throws
   UserDAOException;
42
43     /**
44         * getting instance of
   session.
45         * getting all users from
   database,
46         * if not succeed a rollback
   will be committed and an error
   message will be sent
47         * finally will close the
   session upon completion
48         * @return list of items of
   this specific user's id
49         * @throws UserDAOException
```

```
49 if database is corrupted
50      */
51     public List<User> getUsers()
   throws UserDAOException;
52
53     /**
54      * getting instance of
   session.
55      * getting all the items that
    match the month and year from
   database,
56      * if not succeed a rollback
   will be committed and an error
   message will be sent
57      * finally will close the
   session upon completion
58      * @param month specific
   month of bought items that the
   user want to view
59      * @param year specific year
   of bought items that the user
   want to view
60      * @return list of items that
    corresponding to the input
   parameters
61      * @throws ItemDAOException
   if database is corrupted
62      */
```

```
63      public List<Item> getReport(String month ,String year,int userId) throws ItemDAOException;
64
65      /**
66       * getting instance of session.
67       * checking if the user exists in the database,
68       * if not succeed a rollback will be committed and an error message will be sent
69       * finally will close the session upon completion
70       * @param ob User's object
71       * @return a boolean parameter that is true if user exists and false otherwise
72       * @throws UserDAOException if database is corrupted
73       */
74      public boolean isUserExist(il.ac.hit.model.User ob) throws UserDAOException;
75  }
76
```

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


/**
 * Represents the exception class.
 * which is handling exceptions
 * related to HibernateCostManagerDAO that possible to happen.
 */
public class ItemDAOException extends Exception {

    /**
     * Exception constructor with message
     * @param message Exception's message
     */
    public ItemDAOException(String message) {
        super(message);
    }

    /**
     * Exception constructor with message and cause
```

```
21          * @param message Exception's
     message
22          * @param cause Exception's
    cause
23          */
24         public ItemDAOException(
    String message, Throwable cause
    ) {
25             super(message, cause);
26         }
27
28         /**
29          * Empty exception
    constructor
30          */
31         public ItemDAOException() {
32             super();
33         }
34
35
36
37 }
38
```

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


/**
 * Represents the exception class.
 * which is handling exceptions
 * related to HibernateCostManagerDAO that possible to happen.
 */
public class UserDAOException extends Exception {

    /**
     * Exception constructor with message
     * @param message Exception's message
     */
    public UserDAOException(String message) {
        super(message);
    }

    /**
     * Exception constructor with message and cause
```

```
21          * @param message Exception's
     message
22          * @param cause Exception's
    cause
23          */
24         public UserDAOException(
    String message, Throwable cause
    ) {
25             super(message, cause);
26         }
27
28         /**
29          * Empty exception
    constructor
30          */
31         public UserDAOException() {
32             super();
33         }
34
35
36 }
```

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

import org.hibernate.*;
import org.hibernate.cfg.
AnnotationConfiguration;
import java.util.ArrayList;
import java.util.List;

/**
 * a class representing model in
our project
 * which is managing our database
 */
public class
HibernateCostManagerDAO
implements ICostManagerDAO {

    private static SessionFactory
 factory;
    private static
HibernateCostManagerDAO
costManagerDAO=null;

    /**
     * Empty constructor
     */
    private
HibernateCostManagerDAO() {
```

```
21      }
22
23      /**
24       * A design pattern for our
    database
25       * Singleton implementation
    for HibernateCostManagerDAO
26       * @return static class
    member
27       */
28      public static
    HibernateCostManagerDAO
    getHibernateCostManagerDAO(){
29          if(costManagerDAO==null){
30              costManagerDAO=new
    HibernateCostManagerDAO();
31          }
32          return costManagerDAO;
33      }
34
35      /**
36       * A design pattern for our
    database
37       * for getting a session per
    user
38       * @return return static
    class member ,session factory
39       */
```

```
40      public static SessionFactory
    getSessionFactory(){
41          if(factory==null){
42              factory=new
    AnnotationConfiguration().
    configure().buildSessionFactory
    ();
43          }
44          return factory;
45      }
46
47      /**
48       * getting instance of
    session ,saving the object of the
     specific item to our database
49       * if not succeed a rollback
    will be committed and an error
    message will be sent
50       * finally closing the
    session upon completion
51       * @param ob Item's object
52       * @throws ItemDAOException
    if database is corrupted
53       */
54      @Override
55      public void addItem(Item ob)
    throws ItemDAOException {
56          Transaction tx=null;
```

```
57              Session session =
    getSessionFactory().openSession
    ();
58              try {
59                  tx = session.
    beginTransaction();
60                  session.save(ob);
61                  session.
    getTransaction().commit();
62              } catch (
    HibernateException e) {
63                  if (tx != null)
    tx.rollback();
64                  throw new
    ItemDAOException("Can't add item"
    , e);
65              } finally {
66                  try {
67                      session.close
    ();
68                  } catch (
    HibernateException e) {
69
70                  }
71              }
72      }
73
74      /**
```

```
75        * getting instance of session.
76        * getting all items from database for this specific user's id,
77        * if not succeed a rollback will be committed and an error message will be sent
78        * finally will close the session upon completion
79        * @param userId getting user's id
80        * @return list of items of this specific user's id
81        * @throws ItemDAOException if database is corrupted
82        */
83       @Override
84       public List<Item> getItems(int userId) throws ItemDAOException {
85           Transaction tx=null;
86           List<Item> list =new ArrayList<>();
87
88           Session session = getSessionFactory().openSession();
```

```
 89          try {
 90              tx = session.
    beginTransaction();
 91              List items = session
    .createQuery("from Item").list
    ();
 92              List userIdPerUser
     = session.createQuery("select
    userId from Item").list();
 93
 94              for (int i = 0; i <
    items.size(); i++)
 95                  if (
    userIdPerUser.get(i).equals(
    userId))
 96                      list.add((
    Item) items.get(i));
 97
 98              session.
    getTransaction().commit();
 99          }
100          catch (
    HibernateException e){
101
102              if(tx!=null)
103                  tx.rollback();
104              throw new
    ItemDAOException("Can't get list
```

```
104  of items",e);
105          }
106          finally{
107              try{
108                  session.close();
109              }
110              catch (
    HibernateException e){
111
112              }
113          }
114          return list;
115      }
116
117      /**
118       * getting instance of
    session ,saving the object of
    the specific user to our
    database
119       * if not succeed a rollback
     will be committed and an error
    message will be sent
120       * finally closing the
    session upon completion
121       * @param ob User's object
122       * @throws UserDAOException
    if database is corrupted
123       */
```

```
124     @Override
125     public void addUser(User ob
    ) throws UserDAOException {
126
127         Transaction tx=null;
128         if(!isUserExist(ob)){
129
130         Session session =
    getSessionFactory().openSession
    ();
131             try {
132                 tx = session.
    beginTransaction();
133                 session.save(ob
    );
134                 session.
    getTransaction().commit();
135             } catch (
    HibernateException e) {
136                 if (tx != null)
    tx.rollback();
137                 throw new
    UserDAOException("Can't add user
    ", e);
138             } finally {
139                 try {
140                     session.
    close();
```

```
141                     } catch (HibernateException e) {
142
143                     }
144                 }
145             }
146         }
147
148         /**
149          * getting instance of session.
150          * getting all users from database,
151          * if not succeed a rollback will be committed and an error message will be sent
152          * finally will close the session upon completion
153          * @return list of items of this specific user's id
154          * @throws UserDAOException if database is corrupted
155          */
156         @Override
157         public List<User> getUsers() throws UserDAOException {
158
159             Transaction tx=null;
```

```
160             List<User> list =new
    ArrayList<>();
161
162             Session session =
    getSessionFactory().openSession
    ();
163             try {
164                 tx = session.
    beginTransaction();
165                 List users = session
    .createQuery("from User").list
    ();
166                 for (int i = 0; i <
    users.size(); i++)
167                     list.add((User)
    users.get(i));
168
169                 session.
    getTransaction().commit();
170             }
171             catch (
    HibernateException e){
172
173                 if(tx!=null)
174                     tx.rollback();
175                 throw new
    UserDAOException("Can't get list
     of users",e);
```

```
176             }
177             finally{
178                 try{
179                     session.close();
180                 }
181                 catch (
    HibernateException e){
182
183                 }
184             }
185             return list;
186         }
187
188         /**
189          * getting instance of
    session.
190          * getting all the items
    that match the month and year
    from database,
191          * if not succeed a rollback
     will be committed and an error
    message will be sent
192          * finally will close the
    session upon completion
193          * @param month specific
    month of bought items that the
    user want to view
194          * @param year specific year
```

```
194  of bought items that the user want to view
195      * @return list of items that corresponding to the input parameters
196      * @throws ItemDAOException if database is corrupted
197      */
198     @Override
199     public List<Item> getReport(String month, String year ,int userId) throws ItemDAOException {
200         Transaction tx=null;
201         List<Item> list =new ArrayList<>();
202
203         Session session = getSessionFactory().openSession();
204         try {
205             tx = session.beginTransaction();
206             List items = session.createQuery("from Item").list();
207             List dateFromItem = session.createQuery("select date
```

```
207  from Item").list();
208                List userIdFromItem
     = session.createQuery("select
    userId from Item").list();
209
210                for (int i = 0; i <
    items.size(); i++) {
211                    String str = (
    String) dateFromItem.get(i);
212                    String[]
    afterSplit = str.split("-");
213                    int userIdFromDB
    =(Integer)userIdFromItem.get(i);
214                    if (afterSplit[0
    ].equals(year) && afterSplit[1].
    equals(month) && userId==
    userIdFromDB)
215                        list.add((
    Item) items.get(i));
216                }
217                session.
    getTransaction().commit();
218            }
219            catch (
    HibernateException e){
220
221                if(tx!=null)
222                    tx.rollback();
```

```
223                 throw new
    ItemDAOException("Can't get list
     of items",e);
224             }
225             finally{
226                 try{
227                     session.close();
228                 }
229                 catch (
    HibernateException e){
230
231                 }
232             }
233             return list;
234         }
235
236         /**
237          * getting instance of
    session.
238          * checking if the user
    exists in the database,
239          * if not succeed a rollback
     will be committed and an error
    message will be sent
240          * finally will close the
    session upon completion
241          * @param ob User's object
242          * @return a boolean
```

```
242 parameter that is true if user exists and false otherwise
243      * @throws UserDAOException if database is corrupted
244      */
245     @Override
246     public boolean isUserExist(User ob) throws UserDAOException {
247         Transaction tx=null;
248         Session session = getSessionFactory().openSession();
249         boolean flag = false;
250         try{
251             tx=session.beginTransaction();
252             List users = session.createQuery("select userId from User").list();
253             for (Object x:users) {
254                 if(x.equals(ob.getUserId())) {
255                     flag=true;
256                     break;
257                 }
258             }
```

```
259                 session.
    getTransaction().commit();
260             }
261             catch (
    HibernateException e){
262
263                 if(tx!=null)
264                     tx.rollback();
265                 throw new
    UserDAOException("User not exist
    ",e);
266             }
267             finally{
268                 try{
269                     session.close();
270                 }
271                 catch (
    HibernateException e){
272
273                 }
274             }
275             return flag;
276         }
277 }
278
```

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

import javax.servlet.RequestDispatcher;
import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import java.io.IOException;
import java.io.PrintWriter;
import java.lang.reflect.InvocationTargetException;
import java.lang.reflect.Method;

/**
 * Servlet implementation class RouterServlet
 * directs the incoming requests to the requested Controller
 */
@WebServlet("/controller/*")
public class Router extends
```

```
19 HttpServlet {
20     private static final long
   serialVersionUID = 1L;
21     public static String
   CONTROLLERS_PACKAGE = "il.ac.hit.
   controller";
22     /**
23      * @see HttpServlet#
   HttpServlet()
24      */
25     public Router() {
26         super();
27         // TODO Auto-generated
   constructor stub
28     }
29
30     /**
31      * Called by the server (via
   the service method) to allow a
   servlet to handle a GET request.
32      * @see HttpServlet#doGet(
   HttpServletRequest request,
   HttpServletResponse response)
33      */
34     protected void doGet(
   HttpServletRequest request,
   HttpServletResponse response)
35             throws
```

```
35 ServletException, IOException {
36
37         try {
38             response.
   setContentType("text/html");
39             // TODO Auto-
   generated method stub
40
41             // Returns the part
   of this request's URL from the
   protocol name up to
42             // the query string
   in the first line of the HTTP
   request.
43             String text = request
   .getRequestURI();
44             //Splits this string
   around matches of the given
   regular expression.
45             String[] texts = text
   .split("/");
46             // extracting
   controller and action
47             String controller =
   texts[3];
48             String action = texts
   [4];
49
```

```
50                  // building the full
    qualified name of the controller
51                  String temp =
    controller + "Controller";
52                  String
    controllerClassName =
    CONTROLLERS_PACKAGE + "."
53                          + temp.
    substring(0, 1).toUpperCase() +
    temp.substring(1);
54
55                  // instantiating the
    controller class and calling
56                  // the action method
    on the controller object
57                  Class clazz = Class.
    forName(controllerClassName);
58                  Method method = clazz
    .getMethod(action,
    HttpServletRequest.class,
    HttpServletResponse.class);
59
60                  if(request.getSession
    ().getAttribute("userid")==null
     && action.equals("additem")) {
61                      PrintWriter out
     = response.getWriter();
62                      out.println("User
```

```
62   is not logged in");
63                   }
64                   else{
65                       //activate the
   action method by reflection
66                       method.invoke(
   clazz.getDeclaredConstructor().
   newInstance(), request, response
   );
67                       // creating a
   RequestDispatcher object that
   points at the JSP document
68                       // which is view
   of our action
69                       RequestDispatcher
    dispatcher = getServletContext
   ().getRequestDispatcher("/views/"
    + action + ".jsp");
70                       dispatcher.
   include(request, response);
71                   }
72
73               }catch (
   ClassNotFoundException |
   InstantiationException |
   IllegalAccessException |
74
   NoSuchMethodException |
```

```
74 SecurityException |
   IllegalArgumentException |
   InvocationTargetException e) {
75             //Prints this
   throwable and its backtrace to
   the standard error stream.
76             e.printStackTrace();
77         }
78     }
79
80     /**
81      * Called by the server (via
    the service method) to allow a
   servlet to handle a POST request
   .
82      * @see HttpServlet#doPost(
   HttpServletRequest request,
   HttpServletResponse response)
83      */
84     protected void doPost(
   HttpServletRequest request,
   HttpServletResponse response)
85             throws
   ServletException, IOException {
86         // TODO Auto-generated
   method stub
87         doGet(request, response
   );
```

```
88      }
89  }
90
```

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

/**
 * Represents the interface of the UserController and ItemController
 */
public interface IController {
}

```

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

import il.ac.hit.model.*;
import org.hibernate.Session;

import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.servlet.http.HttpSession;
import java.io.IOException;
import java.io.PrintWriter;
import java.util.List;

/**
 * This class receives Item requests coming from the router,
 * handles them according to the client's requests
 */
public class ItemController implements IController {

    private static HibernateCostManagerDAO dao;
    private static int incval=5;

```

```
22      /**
23       * action that getting the
    requested items from User
    Iterface
24       * @param request  the
    session is a reference to the
    request per user
25       * @param response the
    session is a reference to the
    response per user
26       * @throws ItemDAOException
    if database is corrupted
27       */
28      public void additem(
    HttpServletRequest request,
    HttpServletResponse response)
    throws ItemDAOException,
    IOException {
29          try{
30              dao=
    HibernateCostManagerDAO.
    getHibernateCostManagerDAO();
31
32              //checking if
    parameters not null and getting
    parameters from the received
    request
33              if(request.getSession
```

```
33 ().getAttribute("userid")!=null
    &&
34                  request.
   getParameter("id") !=null &&
35                  request.
   getParameter("cost")!=null &&
36                  request.
   getParameter("date")!=null &&
37                  request.
   getParameter("type")!=null ) {
38                   int userIdPerUser
    = (Integer) request.getSession
   ().getAttribute("userid");
39                   String temp =
   request.getParameter("id");
40                   int idPerUser =
   Integer.parseInt(temp);
41                   int costPerUser
    = Integer.parseInt(request.
   getParameter("cost"));
42                   String
   datePerUser = request.
   getParameter("date");
43                   String
   typePerUser = request.
   getParameter("type");
44
45                   //creating a new
```

```
45 Item object
46                     Item item = new
   Item(idPerUser, userIdPerUser,
   costPerUser, datePerUser,
   typePerUser, incval);
47                     //add item to db
48                     dao.addItem(item
   );
49                     //getting items
   according to input userId
50                     List<Item>
   itemList = dao.getItems(
   userIdPerUser);
51                     if (itemList !=
   null)
52                         request.
   getSession().setAttribute("
   tableitems", itemList);
53                 }
54             } catch (ItemDAOException
    e){
55                 throw new
   ItemDAOException("Error with
   adding item",e);
56             }
57         }
58
59         /**
```

```
60      * action that getting all
   the items that match the month
   and year from User Iterface
61      * @param request  the
   session is a reference to the
   request per user
62      * @param response the
   session is a reference to the
   response per user
63      * @throws ItemDAOException
   if database is corrupted
64      */
65     public void report(
   HttpServletRequest request,
   HttpServletResponse response)
   throws  ItemDAOException {
66         List<Item> reportList;
67         try{
68             dao=
   HibernateCostManagerDAO.
   getHibernateCostManagerDAO();
69             int userId = (Integer
   ) request.getSession().
   getAttribute("userid");
70             //getting parameters
   from the received request
71             String month=request.
   getParameter("month");
```

```
72                  String year=request.
   getParameter("year");
73                  reportList=dao.
   getReport(month,year,userId);
74                  request.getSession().
   setAttribute("report", reportList
   );
75
76              } catch (ItemDAOException
    e){
77                  throw new
   ItemDAOException("Error with
   report list",e);
78              }
79          }
80 }
81
```

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

import il.ac.hit.model.HibernateCostManagerDAO;
import il.ac.hit.model.ICostManagerDAO;
import il.ac.hit.model.User;
import il.ac.hit.model.UserDAOException;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.servlet.http.HttpSession;
import java.io.IOException;
import java.util.List;

/**
 * This class receives User requests coming from the router,
 * handles them according to the client's requests
 */
public class UserController implements IController {
```

```
19
20      private static
    ICostManagerDAO dao;
21
22      /**
23       * login action handles
    request to find and verify the
    user with the database.
24       * @param request  the
    session is a reference to the
    request per user
25       * @param response the
    session is a reference to the
    response per user
26       * @throws UserDAOException
    if database is corrupted
27       */
28      public void login(
    HttpServletRequest request,
    HttpServletResponse response)
    throws UserDAOException {
29          List<User> users;
30          try {
31              dao =
    HibernateCostManagerDAO.
    getHibernateCostManagerDAO();
32              users = dao.getUsers
    ();
```

```
33                  //get all users from
    database
34                  //getting parameters
    from the received request
35                  String username =
    request.getParameter("username");
36                  String pass = request
    .getParameter("password");
37
38                  for (int i = 0; i <
    users.size(); i++) {
39                      if (users.get(i).
    getUserName().equals(username) &&
40                              users.get
    (i).getPassword().equals(pass)) {
41                          request.
    getSession().setAttribute("userid
    ", users.get(i).getUserId());
42                          request.
    getSession().setAttribute("check"
    , "User Logged in successfully");
43                          break;
44                      } else
45
46                          request.
    getSession().setAttribute("check"
    , "User is not signed up");
47                  }
```

```
48          } catch (UserDAOException
     e) {
49              throw new
    UserDAOException("Error with
    Login", e);
50          }
51      }
52
53      /**
54       * action that handles a
    request to add a new user to the
    database if no such user exists
    create a new user.
55       * @param request  the
    session is a reference to the
    request per user
56       * @param response the
    session is a reference to the
    response per user
57       * @throws UserDAOException
    if database is corrupted
58       */
59      public void signup(
    HttpServletRequest request,
    HttpServletResponse response)
    throws UserDAOException {
60
61          System.out.println("the
```

```
61 request is " + request.toString
   ());
62          List<User> usersList;
63          try {
64              dao =
   HibernateCostManagerDAO.
   getHibernateCostManagerDAO();
65              usersList = dao.
   getUsers();
66              boolean exists =
   false;
67              if (request.
   getParameter("userid") != null) {
68                  for (int i = 0; i
    < usersList.size(); i++) {
69                      if (usersList
   .get(i).getUserId() == Integer.
   parseInt(request.getParameter("
   userid"))) {
70                          exists =
   true;
71                          request.
   getSession().setAttribute("check1
   ", "User already exists");
72                          break;
73                      }
74                  }
75
```

```
76                      if (!exists) {
77                          int userId
    = Integer.parseInt(request.
   getParameter("userid"));
78                          String
   userName = request.getParameter(
   "username");
79                          String
   lastName = request.getParameter(
   "lastname");
80                          String pass
    = request.getParameter("
   password");
81
82                          User user =
   new User(userId, userName,
   lastName, pass);
83                          dao.addUser(
   user);
84                          request.
   getSession().setAttribute("
   check1", "User registered
   successfully");
85                      }
86                  }
87              } catch (
   UserDAOException e) {
88                  throw new
```

```
 88 UserDAOException("Error with
    Sign-up", e);
 89             }
 90         }
 91
 92         /**
 93          * action that invalidates
    the session of the user
 94          * @param request  the
    session is a reference to the
    request per user
 95          * @param response the
    session is a reference to the
    response per user
 96          */
 97         public void logout(
    HttpServletRequest request,
    HttpServletResponse response) {
 98             HttpSession session =
    request.getSession();
 99             session.invalidate();
100         }
101 }
102
```