```
1  ------------EmployeeDAO.java-----------------
2  package com.ameya.daos;
3
4  import java.util.List;
5
6  import com.ameya.domain.Employee;
7
8  public interface EmployeeDAO {
9
10     void createConnection();
11     void addEmployee(Employee employee);
12     Employee getEmployee(int empId);
13     List<Employee> getAllEmployees();
14     void updateEmployee(Employee employee);
15     void deleteEmployee(int empId);
16     void closeConnection();
17     void transactions();
18     default void batchProcessing() {}
19 }
20 ----------------EmployeeDAOImpl.java-----------------
21 package com.ameya.daos.impl;
22
23 import java.sql.Connection;
24 import java.sql.DatabaseMetaData;
25 import java.sql.DriverManager;
26 import java.sql.PreparedStatement;
27 import java.sql.ResultSet;
28 import java.sql.SQLException;
29 import java.sql.Savepoint;
30 import java.util.ArrayList;
31 import java.util.List;
32
33 import com.ameya.daos.EmployeeDAO;
34 import com.ameya.domain.AppProperties;
35 import com.ameya.domain.Employee;
36 import com.ameya.helpers.PropertiesHelper;
37
38 public class EmployeeDAOImpl implements EmployeeDAO {
39
40     private Connection con;
41     private PreparedStatement ps;
42
43     private static String conUrl;
44     private static String dbDriver;
45     private static String dbUsername;
46     private static String dbPassword;
47
48     static {
```

```
49          PropertiesHelper helper=new PropertiesHelper();
50          conUrl=helper.getProperty(AppProperties.CONURL.toString());
51          dbDriver=helper.getProperty(AppProperties.DRIVERCLASSNAME.toString());
52          dbUsername=helper.getProperty(AppProperties.DBUSERNAME.toString());
53          dbPassword=helper.getProperty(AppProperties.DBPASSWORD.toString());
54          try {
55              Class.forName(dbDriver);
56              System.out.println("++++ MySQL Driver Loaded ++++");
57          } catch (ClassNotFoundException e) {
58              e.printStackTrace();
59          }
60      }
61      @Override
62      public void createConnection() {
63          try {
64              con=DriverManager.getConnection(conUrl, dbUsername, dbPassword);
65              System.out.println("++++ Connected To DB ++++");
66          } catch (SQLException e) {
67              e.printStackTrace();
68          }
69
70      }
71
72      @Override
73      public void addEmployee(Employee employee) {
74          final String SQL = "insert into employee values(?,?,?,?,?)";
75          createConnection();
76          try {
77              ps = con.prepareStatement(SQL);
78              ps.setInt(1, employee.getEmpId());
79              ps.setString(2, employee.getFirstName());
80              ps.setString(3, employee.getLastName());
81              ps.setDouble(4, employee.getSalary());
82              ps.setInt(5, employee.getAge());
83              int cnt = ps.executeUpdate();
84              if (cnt != 0) {
85                  System.out.println("## Row Inserted Into Employee Table ##");
86              }
87          } catch (SQLException e) {
88              e.printStackTrace();
89          } finally {
90              closeConnection();
91          }
92      }
93
94      @Override
95      public Employee getEmployee(int empId) {
96          Employee employee=null;
```

```
97          final String SQL="select * from employee where empId = ?";
98          createConnection();
99          try {
100             ps=con.prepareStatement(SQL);
101             ps.setInt(1, empId);
102             ResultSet rs=ps.executeQuery();
103             if(rs.next()) {
104                 employee=new Employee();
105                 employee.setEmpId(rs.getInt("empid"));
106                 employee.setFirstName(rs.getString("firstname"));
107                 employee.setLastName(rs.getString("lastname"));
108                 employee.setSalary(rs.getDouble("salary"));
109                 employee.setAge(rs.getInt("age"));
110             }
111         } catch (SQLException e) {
112             e.printStackTrace();
113         }finally {
114             closeConnection();
115         }
116         return employee;
117     }
118 
119     @Override
120     public List<Employee> getAllEmployees() {
121         final String SQL="select * from employee";
122         ArrayList<Employee> employees=new ArrayList<Employee>();
123         createConnection();
124         try {
125             ps=con.prepareStatement(SQL);
126             ResultSet rs=ps.executeQuery();
127             while(rs.next()) {
128                 employees.add(new Employee(
129                         rs.getInt("empid"),
130                         rs.getString("firstname"),
131                         rs.getString("lastname"),
132                         rs.getDouble("salary"),
133                         rs.getInt("age")));
134             }
135         } catch (SQLException e) {
136             e.printStackTrace();
137         }finally {
138             closeConnection();
139         }
140         return employees;
141     }
142 
143     @Override
144     public void updateEmployee(Employee employee) {
```

```
145         final String SQL="update employee set firstname = ? , lastname = ? , salary
            = ? , age = ? where empid = ?";
146         createConnection();
147         try {
148             ps=con.prepareStatement(SQL);
149             ps.setString(1, employee.getFirstName());
150             ps.setString(2, employee.getLastName());
151             ps.setDouble(3, employee.getSalary());
152             ps.setInt(4, employee.getAge());
153             ps.setInt(5, employee.getEmpId());
154 
155             int cnt=ps.executeUpdate();
156             if(cnt!=0) {
157                 System.out.println("#### Employee Record Updated ####");
158             }
159         } catch (SQLException e) {
160             e.printStackTrace();
161         }finally {
162             closeConnection();
163         }
164 
165     }
166 
167     @Override
168     public void deleteEmployee(int empId) {
169         final String SQL="delete from employee where empid = ?";
170         createConnection();
171         try {
172             ps=con.prepareStatement(SQL);
173             ps.setInt(1, empId);
174             int cnt=ps.executeUpdate();
175             if(cnt!=0) {
176                 System.out.println("#### Employee Record Deleted ####");
177             }
178         } catch (SQLException e) {
179             e.printStackTrace();
180         }finally {
181             closeConnection();
182         }
183 
184     }
185 
186     @Override
187     public void closeConnection() {
188         if(con!=null) {
189             try {
190                 con.close();
191                 System.out.println("++++ DB Connection Closed ++++");
```

```
192             } catch (SQLException e) {
193                 // TODO Auto-generated catch block
194                 e.printStackTrace();
195             }
196         }
197     }
198
199     @Override
200     public void transactions() {
201         final String SQL="insert into employee values(?,?,?,?,?)";
202         Employee e1=new Employee(101,"AAAA","AAAA",12345,45);
203         Employee e2=new Employee(102,"BBBB","BBBB",23456,56);
204         Savepoint s1=null;
205         createConnection();
206         try {
207                 con.setAutoCommit(false);
208                 DatabaseMetaData dmd=con.getMetaData();
209
                    if(dmd.supportsTransactionIsolationLevel(Connection.TRANSACTION_SER
                    IALIZABLE)) {
210                     System.out.println("Current Isolation level ::
                        "+con.getTransactionIsolation());
211
                        con.setTransactionIsolation(Connection.TRANSACTION_SERIALIZABL
                        E);
212                     System.out.println("Isolation level Set To ::
                        "+con.getTransactionIsolation());
213                 }
214                 ps=con.prepareStatement(SQL);
215                 ps.setInt(1, e1.getEmpId());
216                 ps.setString(2, e1.getFirstName());
217                 ps.setString(3, e1.getLastName());
218                 ps.setDouble(4, e1.getSalary());
219                 ps.setInt(5, e1.getAge());
220                 ps.executeUpdate();
221                 s1=con.setSavepoint("s1");
222                 ps.clearParameters();
223                 //Erroneous Insert
224                 ps.setInt(1, e2.getEmpId());
225                 ps.setString(2, e2.getFirstName());
226                 ps.setString(3, e2.getLastName());
227                 ps.setDouble(4, e2.getSalary());
228                 //ps.setInt(5, e2.getAge());
229                 ps.executeUpdate();
230
231                 con.commit();
232                 System.out.println("## Transaction Committed Successfully ##");
233         }catch(SQLException e) {
```

```
234             System.out.println("## Transaction Rolled Back ##");
235             try {
236                 //con.rollback();
237                 con.rollback(s1);
238             } catch (SQLException e3) {
239                 e3.printStackTrace();
240             }
241         }finally {
242             try {
243                 con.setAutoCommit(true);
244             } catch (SQLException e) {
245                 // TODO Auto-generated catch block
246                 e.printStackTrace();
247             }
248             closeConnection();
249         }
250 
251     }
252 
253     @Override
254     public void batchProcessing() {
255         final String SQL = "insert into employee values(?,?,?,?,?)";
256         Employee e1=new Employee(103, "CCCC", "CCCC", 12345, 44);
257         Employee e2=new Employee(104, "DDDD", "DDDD", 23456, 42);
258         createConnection();
259         try {
260             con.setAutoCommit(false);
261             ps=con.prepareStatement(SQL);
262             ps.setInt(1, e1.getEmpId());
263             ps.setString(2, e1.getFirstName());
264             ps.setString(3, e1.getLastName());
265             ps.setDouble(4, e1.getSalary());
266             ps.setInt(5, e1.getAge());
267             ps.addBatch();//add the query to batch
268             ps.clearParameters();
269             ps.setInt(1, e2.getEmpId());
270             ps.setString(2, e2.getFirstName());
271             ps.setString(3, e2.getLastName());
272             ps.setDouble(4, e2.getSalary());
273             //ps.setInt(5, e2.getAge());
274             ps.addBatch();//add the query to batch
275 
276             /*
277              * Multiple statements to be added to this batch
278              * */
279             int cnt[]=ps.executeBatch();//execute all queries from the batch
280             //for(int c : cnt) {
281             //  if(c==0) {
```

```
                    //throw SomeException add the corresponding catch block and
                    rollback the transaction
                //}
            //}
            con.commit();
            System.out.println("## Batch Executed and Committed ##");
        }catch(SQLException e) {
            System.out.println("## Rolled Back ##");
            try {
                con.rollback();
            } catch (SQLException e3) {
                e3.printStackTrace();
            }
        }finally {
            try {
                con.setAutoCommit(true);
            } catch (SQLException e) {
                e.printStackTrace();
            }
            closeConnection();
        }
    }

}
-----------------EmployeeService.java-----------------
package com.ameya.services;

import java.util.List;

import com.ameya.domain.Employee;

public interface EmployeeService {

    void createEmployee(Employee employee);
    void modifyEmployee(Employee employee);
    void removeEmployee(int empId);
    Employee findEmployeeById(int empId);
    List<Employee> findAll();
    void transaction();
    default void batchProcessing() {}
}
----------------EmployeeServiceImpl.java-----------------
package com.ameya.services.impl;

import java.util.List;

import com.ameya.daos.EmployeeDAO;
import com.ameya.domain.Employee;
```

```
import com.ameya.services.EmployeeService;

public class EmployeeServiceImpl implements EmployeeService {

    private EmployeeDAO employeeDao;
    public EmployeeServiceImpl() {

    }
    public EmployeeServiceImpl(EmployeeDAO employeeDao) {
        this.employeeDao=employeeDao;
    }
    @Override
    public void createEmployee(Employee employee) {
        employeeDao.addEmployee(employee);
    }

    @Override
    public void modifyEmployee(Employee employee) {
        employeeDao.updateEmployee(employee);

    }

    @Override
    public void removeEmployee(int empId) {
        employeeDao.deleteEmployee(empId);

    }

    @Override
    public Employee findEmployeeById(int empId) {
        return employeeDao.getEmployee(empId);
    }

    @Override
    public List<Employee> findAll() {
        return employeeDao.getAllEmployees();
    }
    @Override
    public void transaction() {
        employeeDao.transactions();

    }
    @Override
    public void batchProcessing() {
        employeeDao.batchProcessing();
    }

}
```

```
----------------TestTransactions.java-----------------
package com.ameya.test;

import com.ameya.daos.impl.EmployeeDAOImpl;
import com.ameya.services.EmployeeService;
import com.ameya.services.impl.EmployeeServiceImpl;

public class TestTransactions {

    public static void main(String[] args) {
        EmployeeService service=new EmployeeServiceImpl(new EmployeeDAOImpl());
        service.transaction();

    }

}
----------------TestBatch.java-----------------
package com.ameya.test;

import com.ameya.daos.impl.EmployeeDAOImpl;
import com.ameya.services.EmployeeService;
import com.ameya.services.impl.EmployeeServiceImpl;

public class TestBatch {

    public static void main(String[] args) {
        EmployeeService service=new EmployeeServiceImpl(new EmployeeDAOImpl());
        service.batchProcessing();

    }

}

```