

## DAY 6

### DAO Design Pattern

In every JDBC program, we have written till now, there are two components namely-

- Database Details
- Database Operations

Database Details is the details/code required to connect to the database. So we have URL, Username, Password, MySQL jar file to load the driver.

Database Operations is nothing but the **CRUD(Create, Read, Update, Delete)** operations. And we have written code in such a way that both Database details and database operations are present in the single class. But whenever we write such code, few problems may arise What are the problems?

- Maintainability
- Flexibility

So developers came up with a solution,i.e., **Single Responsibility Principle** and according to this principle, a single class should have a Single Responsibility.

If a class has multiple responsibilities, then Maintainability becomes difficult because one change in the code might affect the entire program and the code which we have written is not flexible in nature.

Assume that code which we have written, has to connect to a database and we are connecting to MySQL database, but what if we had another database that is managed by another RDBMS(for eg, Oracle DBMS) and oracle driver has a different driver and the Database Details code will not work for oracle since it is written specifically for MySQL. So our code is not flexible enough.

How do we bring the changes so that both the problems i.e., Maintainability and Flexibility will be resolved?

Developers before us have come up with **Tried and Tested Solutions** to resolve the issue.

These Tried and Tested Solutions are technically called as **Design Pattern** and there are many Design Pattern such as-

- Data Access Object Design Pattern
- Model View Controller Design Pattern
- Singleton Design Pattern
- Factory Design Pattern

And we will be discussing the **Data Access Object [DAO] Design Pattern** in this session. There are few steps DAO Design Pattern, which if we follow then the problems will be resolved

## Steps to implement DAO Design Pattern:

1. Create a Data Transfer Object class[DTO class]
2. Create a DAO interface and implement the interface
3. Create a ConnectorFactory class

Previously we had discussed the problems we would face as a programmer using JDBC if we do not use the concept of **DAO(Data Access Object)** Design Pattern.

Now let us see how we can implement DAO in our program.

Assume we have an employee table in our database and we have to perform **CRUD(Create, Read, Update, Delete)** operations on that table and we do that by separating the code into logical parts as shown below-

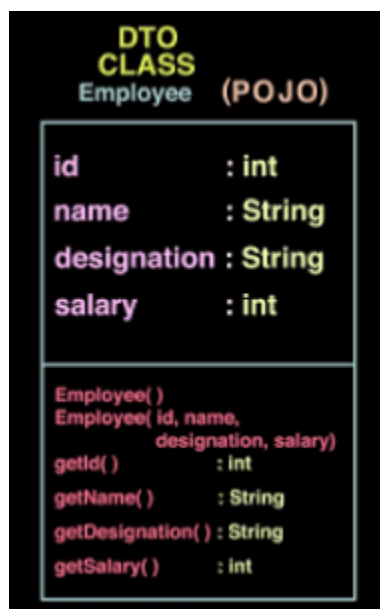


We will create 3 classes -

- DTO [Data Transfer Object] class
- DAO [Data Access Object] class
- Connector class

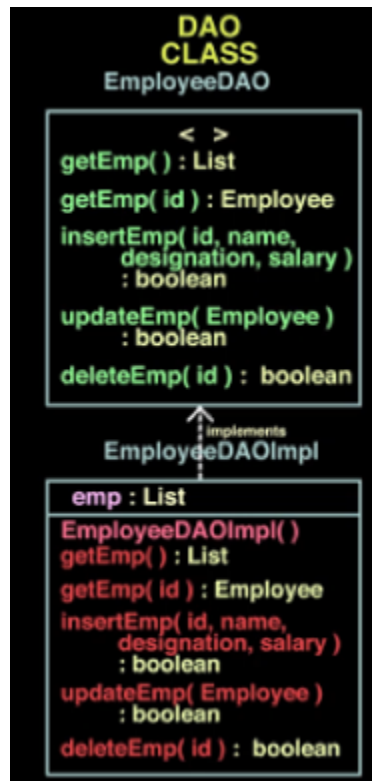
And each of these classes we have its responsibility -

DTO class is a **POJO [Plain Old Java Object]** class, which contains instance variables and getters and setters and the responsibility of this class is to create objects of the Entities present in our table.

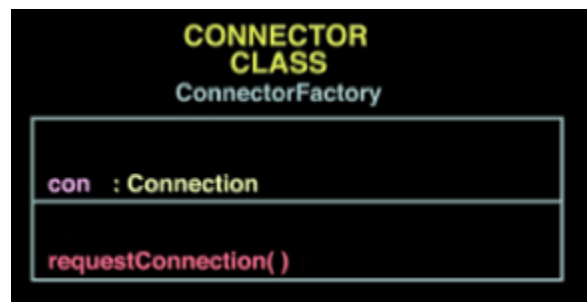


DAO class is a class through which we can perform operations on the table like selecting all the employee details or fetching one employee information, etc.,

So we will create an interface **EmployeeDAO** which contains abstract methods like **getEmp()**, **insertEmployee(id, name, designation, salary)**, **updateEmployee(Employee)** and **deleteEmployee(id)**, etc., and this interface will be implemented by another class called as **EmployeeDAOImpl** and this class implements all the methods of **EmployeeDAO** interface and gives its implementation by overriding it.

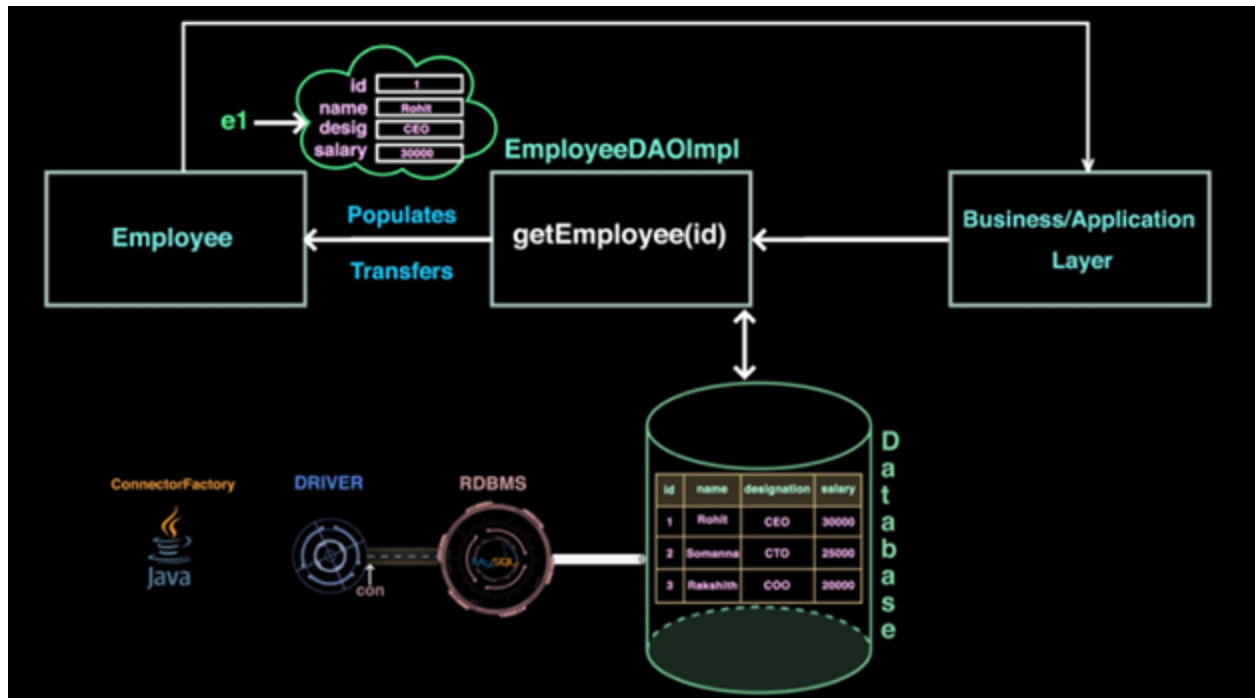


Connector class is used to get a connection to our database. So the connector class has to establish the connection and return the reference **con** so that we can perform CRUD operations on the table.



### How do all these different classes work?

The Application Layer wanted the details of the employee whose id is 1 and pass on the information to **EmployeeDAOImpl** class and **EmployeeDAOImpl** class requested **ConnectorFactory** class to give the connection to the database and using this connection, **EmployeeDAOImpl** selected the record of that employee and using the **Employee** class created an object and populated the records into the instance variables and **Employee** class transferred the object back to the Application Layer



Let us now implement this in the code-  
First, we have to create 3 packages-

- com.tap.dto
- com.tap.dao
- com.tap.connector

Now create Employee class inside com.tap.dto package

```
package com.tap.dto;

public class Employee {

    private int id;
    private String name;
    private String designation;
    private int salary;

    public Employee(int id, String name, String designation, int salary)
    {
        this.id = id;
        this.name = name;
        this.designation = designation;
        this.salary = salary;
    }

    public int getId() {
```

```
        return id;
    }

    public void setId(int id) {
        this.id = id;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public String getDesignation() {
        return designation;
    }

    public void setDesignation(String designation) {
        this.designation = designation;
    }

    public int getSalary() {
        return salary;
    }

    public void setSalary(int salary) {
        this.salary = salary;
    }
}
```

Create EmployeeDAO interface and EmployeeDAOImpl class inside com.tap.dto package

```
package com.tap.dao;

import java.util.List;
import com.tap.dto.Employee;
```

```

public interface EmployeeDAO {

    List getEmployees();
    Employee getEmployee(int id);
    boolean insertEmployee(int id, String name, String designation, int salary);
    boolean updateEmployee(Employee e);
    boolean deleteEmployee(int id);

}

```

EmployeeDAOImpl class

```

package com.tap.dao;

import java.sql.Connection;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;
import java.util.ArrayList;
import java.util.List;
import com.tap.connector.ConnectorFactory;
import com.tap.dto.Employee;

public class EmployeeDAOImpl implements EmployeeDAO{

    public List getEmployees() {

        ArrayList<Employee> emplist = null;

        try {
            Connection con = ConnectorFactory.requestConnection();
            String query = "select * from emp";
            Statement stmt = con.createStatement();
            ResultSet res = stmt.executeQuery(query);
            emplist = new ArrayList<Employee>();

            while(res.next()==true)
            {
                int id = res.getInt(1);
                String name = res.getString(2);
                String designation = res.getString(3);
                int salary= res.getInt(4);
            }
        }
    }
}

```

```
        Employee e = new
Employee(id,name,designation,salary);

        emplist.add(e);

    }

    } catch (Exception e) {
        e.printStackTrace();
    }

    return emplist;
}

public Employee getEmployee(int id) {

    return null;
}

public boolean insertEmployee(int id, String name, String
designation, int salary) {

    return false;
}

public boolean updateEmployee(Employee e) {

    return false;
}

public boolean deleteEmployee(int id) {

    return false;
}

}
```

Create ConnectorFactory class inside com.tap.connector package

```
package com.tap.connector;

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

public class ConnectorFactory {

    static Connection con = null;
    static String url = "jdbc:mysql://localhost:3306/employee";
    static String un = "root";
    static String pwd = "root";

    static public Connection requestConnection() throws
ClassNotFoundException, SQLException {
        Class.forName("com.mysql.cj.jdbc.Driver");
        con = DriverManager.getConnection(url, un, pwd);
        return con;
    }
}
```

Create a class so that we can execute our program-

```
package jdbc;

import java.util.List;

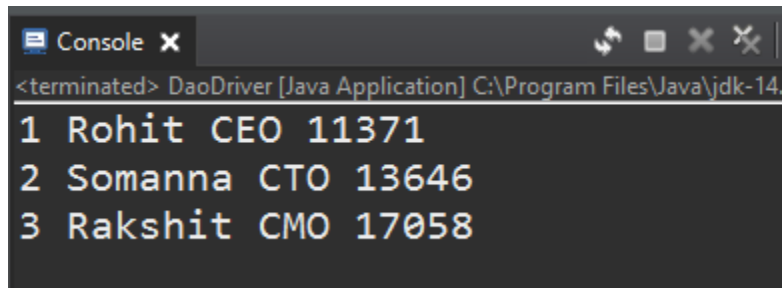
import com.tap.dao.EmployeeDAOImpl;
import com.tap.dto.Employee;

public class DaoDriver {
    public static void main(String[] args) {
        EmployeeDAOImpl emplDAOImpl = new EmployeeDAOImpl();
        List<Employee> employees = emplDAOImpl.getEmployees();

        for(Employee e :employees) {
            System.out.println(e);
        }
    }
}
```



Output:



```
<terminated> DaoDriver [Java Application] C:\Program Files\Java\jdk-14.0.2\bin\java.exe
1 Rohit CEO 11371
2 Somanna CTO 13646
3 Rakshit CMO 17058
```

Let's now implement other methods in EmployeeDAOImpl class-

**getEmployee(id)-**

```
public Employee getEmployee(int id) {
    Employee e = null;
    try {
        Connection con = ConnectorFactory.requestConnection();
        String query = "select * from emp where id = ?";
        PreparedStatement pstmt = con.prepareStatement(query);
        pstmt.setInt(1, id);
        ResultSet res = pstmt.executeQuery();
        res.next();

        e = new
Employee(res.getInt(1),res.getString(2),res.getString(3),res.getInt(4));
    } catch (Exception e2) {
        e2.printStackTrace();
    }
    return e;
}
```

Let us now write the code in DaoDriver class so that we can get one employee row/record.

```
package jdbc;

import java.util.List;
import java.util.Scanner;

import com.tap.dao.EmployeeDAOImpl;
import com.tap.dto.Employee;

public class DaoDriver {
    public static void main(String[] args) {
        EmployeeDAOImpl emplDAOImpl = new EmployeeDAOImpl();
        Scanner scan = new Scanner(System.in);
    }
}
```

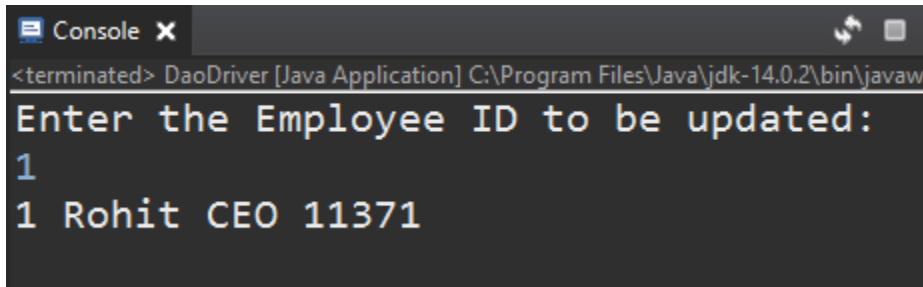
```

        System.out.println("Enter the Employee ID to be updated:");
        int id = scan.nextInt();

        Employee e = emplDAOImpl.getEmployee(id);
        System.out.println(e);
    }
}

```

Output:



```

<terminated> DaoDriver [Java Application] C:\Program Files\Java\jdk-14.0.2\bin\javaw.
Enter the Employee ID to be updated:
1
1 Rohit CEO 11371

```

Let us now try to update employee details:

```

public boolean updateEmployee(Employee e) {
    int i = 0;

    try {
        Connection con = ConnectorFactory.requestConnection();
        String query = "update emp set salary = ? where id = ?";
        PreparedStatement pstmt = con.prepareStatement(query);
        pstmt.setInt(1, e.getSalary());
        pstmt.setInt(2, e.getId());
        i = pstmt.executeUpdate();
    }
    catch (Exception e2) {
        e2.printStackTrace();
    }

    if(i==1) {
        return true;
    }
    else {
        return false;
    }
}

```

Let us now write the code in DaoDriver class so that we can update employee's salary-

```

package jdbc;

```

```

import java.util.List;
import java.util.Scanner;

import com.tap.dao.EmployeeDAOImpl;
import com.tap.dto.Employee;

public class DaoDriver {
    public static void main(String[] args) {
        EmployeeDAOImpl emplDAOImpl = new EmployeeDAOImpl();

        Scanner scan = new Scanner(System.in);
        System.out.println("Enter the Employee ID to be updated:");
        int id = scan.nextInt();

        Employee e = emplDAOImpl.getEmployee(id);
        System.out.println(e);

        System.out.println("Enter the salary to be updated:");
        int newSalary = scan.nextInt();

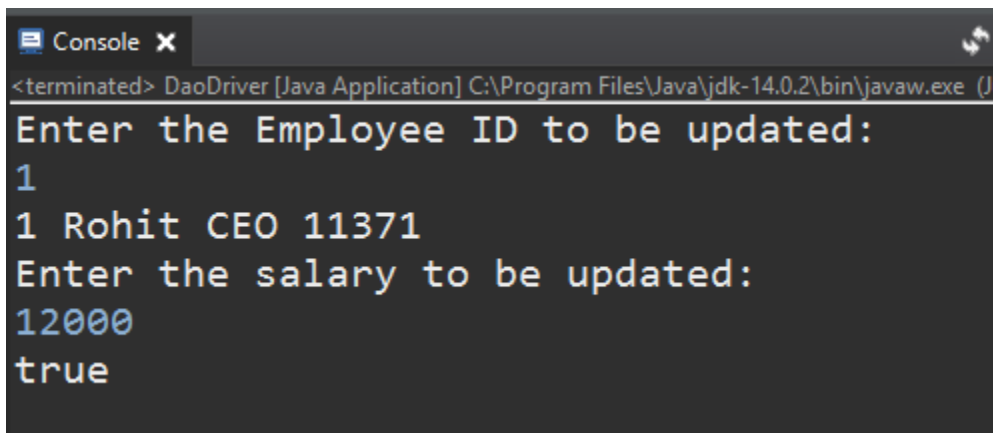
        e.setSalary(newSalary);

        System.out.println(emplDAOImpl.updateEmployee(e));

    }
}

```

## Output:



The screenshot shows a console window titled "Console" with a close button. The title bar also includes the application name "DaoDriver [Java Application]" and the path "C:\Program Files\Java\jdk-14.0.2\bin\javaw.exe (J". The output text is as follows:

```

Enter the Employee ID to be updated:
1
1 Rohit CEO 11371
Enter the salary to be updated:
12000
true

```

And the data would be updated in the database also-

	id	name	dsig	salary
▶	1	Rohit	CEO	12000
	2	Somanna	CTO	13646
	3	Rakshit	CMO	17058
★	NULL	NULL	NULL	NULL

#### **Advantages of using DAO Design Pattern:**

1. Since we have separated Database Details[ConnectorFactory class] and Database Operations[EmployeeDAOImpl class] into separate classes, maintainability becomes very easy. Let us assume that we want to change from MySQL database to Oracle Database, then the only place where we have to do changes is in the ConnectorFactory class
2. Similarly, if we have to change the database-related operations in the database then we just have to do changes in EmployeeDAOImpl class, and this makes our code flexible.