DAO: The Java class that separates persistence logic from other logics of the application to provide the flexibility of modification to persistence logic is called as DAO.

1. Spring DAO module provides abstraction layer on plain JDBC programming and simplifies JDBC style persistence logic development through JDBCTemplate class.
2. Using Spring DAO module we can develop persistence logic by implementing DAO design pattern.

Plain JDBC programming:

1. Load JDBC driver class to register JDBC driver with DriverManager service.
2. Establish communication with Database software.
3. Create Statement object
4. Send and execute SQL query in Database software.
5. Gather and process the results.
6. Take care of exception handling
7. Close JDBC objects

Here 1,2,3 are command logics in every JDBC application

6,7 are command logics

4,5 are application specific loics in every JDBC application

Note: While developing plain JDBC application programmers needs to write application specific and command logics.

Spring JDBC programming (Spring DAO):

1. Inject JDBCTemplate class object.
2. Use JDBCTemplate class object to send and execute SQL Queries in Database software.
3. Gather and process the results in Spring JDBC programming programmer just need to develop Applciation Specific logics because the JDBCTemplate clas internally uses plain JDBC generates common logics.

Advantages of Spring DAO:

1. The process of converting one form of execption to another form of exception is called Exception Rethrowing.
2. In Spring , Hibernate Frameworks the underlying technologies generated Checked exceptions will be converted into un Checked exceptions by using Exception Rethrowing concept as shown in below.

```
Public int queryForInt(String qry)throws DataAccessException //Unchecked Exception

{

try

 {

 -----

 -----

}// try

Catch(SQLException se)  //Checked Exception

{

throw new DataAccessException() ;   //Exception rethowing

}

return obj;

}
```

Note: java.lang.Exception is partially Checked Exception

Advantages of Spring DAO:

1. No need of Working with Plain JDBC API.
2. Provides abstraction layer on Plain JDBC and simplifies JDBC programming.
3. Converting all Checked to Unchecked exceptions by using Exception Rethrowing concept so the Exception Handling is optional.
4. Use Queryxxx(-) methods to gather select query results in different formats.
5. We can send select query results directly over the network because certain Queryxxx methods gives results directly to Collection Framework data structures.
6. To create org.springframework.core.JDBCTemplate class object is required and this can be supplied either through setter or Constructor injection method of JDBCTemplate class to send and execute SQL queries in Database softwares.
7. query, queryForxxx() are given send and execute select SQL queries.
8. update() is give to send and execute non select SQL queries batchUpdate(-) is given to perform batch processing.

More about queryxxx() methods: To execute select sql query that returns numeric values use query for long methods.

EX: select count(*) from temp;

To execute select sql query that gives one record use queryForMap():

EX: select * from emp where empno=7499

To execute select SQL query that gives one recordqueryForMap():

EX: select * from emp where empno=7499

To execute select SQL query that gives multiple records use queryForList() or query ForRowsert() methods:

EX: select * from emp;

To execute select sql query and to get results user defined or predefined Java class objects use queryForObject method:

Details of C3P0 Pool:

Type: third party supplied software for jdbc connection pooling.

Vendor: open community

Version: 0.9.1.2

Jar file: <spring-home>\lib\C3P0\C3P0-0.9.1.2.jar

The class that gives Datasource representing JDBC connection pool.

JDBC Conenction Pool:

Com.mchange.v2.C3P0.ComboPooledDataSource

Important properties of above class:

driverClass()

jdbcUrl()

user()

password()

auireIncrement()

initialPoolSize()

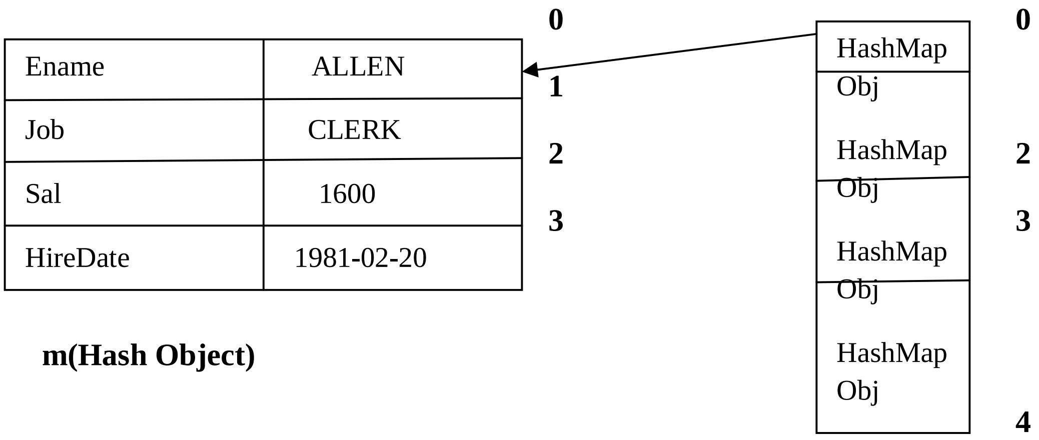
maxPoolSize()

maxIdleTime()

Example code configuration C3P0 pool in Spring configuration file:

```
<bean id=”c3p0ds” class=”com.mchange.v2.c3p0.cobboPoolDatasource”/>

<property name=”jdbcurl” value=”jdbc:oracle:thin:@localhost:1521:orcl”/>

<property name=”user” value=”scott”/>

<property name=”password” value=”tiger”/>

<property name=”acquireIncrement” value=”2”/>

<property name=”maxPoolSize” value=”100”/>

</bean>
```

The above code uses given JDBC details to create JDBC connection pool for Oracle and create JDBC Datasource object representing that Connection Pool.



Ename	ALLEN
Job	CLERK
Sal	1600
HireDate	1981-02-20

m(Hash Object)

l(ArrayList Object)

Each HashMap object represent one record that is selected.

Example Application on Spring DAO using C3P0 JDBC Connection Pool:

Note: When we call queryForxxx methods with one argument then they internally use JDBC simple Statement object to send and execute SQL queries in Database software when the above same methods are called with multiple arguments then they internally use JDBC PreparedStatement object to send and execute in Database software.

For Complete program please refer Spring DAO\NetBeans 7.1 Projects\C3P0ConnectionPoolDAO

Note: Interface's variable prototype declaration done by compiler.

Note: If no Constructor is there Java compiler will generate default constructor

Note: At the time of overriding, modifiers may change in subclass from super class of a program the best example is doXxx(-,-), service(-,-)

Understanding Apache DBCP Connection Pool:

Type: Third party Connection Pool software for standalone applications

Vendor: Apache Foundation

Free software

Jar file that represent this can pool

<Spring home>:\lib\jakartha-commons-dbcp

Class that give JDBC Datasource representing jdb.

Important properties of this basic Datasoruce class:

driverClassName ,url, uname, passwordInitialSize

Example Application on Spring DAO module implementing the DAO design pattern and Apache DBCP Connection Pool:

Procedure to develop by using MyEclipse IDE:

1. Launch MyEclipse IDE and create Java Project.
2. Add Spring Capabilities to the project choosing the following Libraries
   Right Click on the project → MyEclipse → Add Spring Capabilities → Select Spring 2.5 → Select Spring Core Libraries, Persistence Core Libraries, JDBC Libraries,→ next → deselect AOP builder → File → spring.cfg.xml→ finish

3. Right Click on Project → build path → Add External Archives → browse and select ojdbc14.jar
4. Develop the following resources in the src folder of the project

DBOperations.java (Spring interface)

DBOperationBean.java (Spring Bean class)

TestClient.java (Client Application)

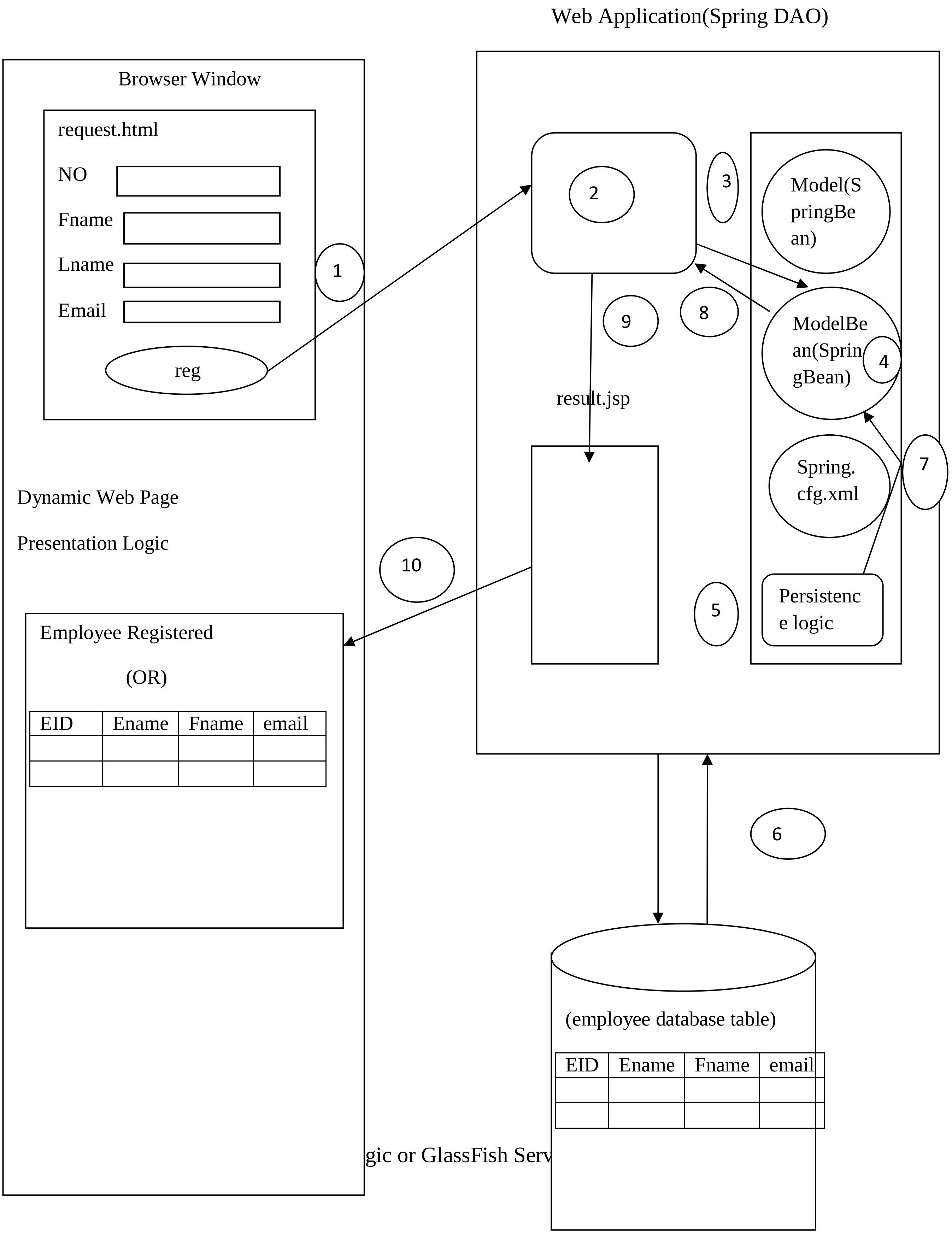
MyDAO.java (DAO Design Pattern implementation).

For Complete Program Please refer Spring DAO\MyEclise 8.X Projects\ApacheDBCPConnectionPoolDAO folder.

Note: It is recommended to use C3P0 Connection Pool in standalone applications.

1. org.springframework.jndi.JndiObjectFactoryBean is predefined FactoryBean class having the ability to gather objects from JndiRegistry and injects them to specified bean properties.
2. This is very useful to gather DataSource object from Registry and to inject them bean properties.
3. If the application that is using Server Managed Connection Pool is running outside the Server then we must specify DataSoruce Jndi along with Jndi properties. If same applciaiton is deployable in the Same server the JNDI properties are optional.

spring source



EID	Ename	Fname	email

EID	Ename	Fname	email

The above web application is MVC1 architecture

M → Model → Spring DAO+CORE Module

V → View → JSP Programs

C → Controller → Servlet

Procedure to develop above application by using MyEclipse IDE:

1. Create web project in MyEclipse IDE
2. Right Click on the project → Select Spring capabilities → select Spring 2.5 core, Persistence core, JDBC, J2EE, Remoting libraries.

JDBC allows only positional parameters in our SQL query which query give confusion towards indexing when more parameters. Spring DAO gives support for two kinds of Parameters those are

1. ? (positional parameters or place holders)
2. (:<name>) Named Parameters

To work Named parameters we need to use NamedParameterJdbcTemplate class this class not to the sub class of JdbcTemplate class.

Note: Internally every Named Parameter will be converted into positional parameter .

1. DriverManagerDataSource represents dummy Connection Pool it creates new Connection object for every getConnection().
2. SingleConnectionDatasource is the sub class of DriverManagerDatasource class giving dummy Connection Pool object for all getConnection method calls.
3. For example Application NamedParameterTemplate class and SingleConnection DataSource class for the application 8 for the booklet.

By using org.springframework.jdbc.core.NamedParam.MapSqlParameterSource class we can set the values to Named Parameters (we can do program with Spring supplied API without Spring Style) gives DataSource object representing ConnectionPool which contains single Connection object. If certain functionality not directly possible with JDBCTemplate, NamedParameter JdbcTemplate classes the use callback interfaces supplied by DAO module. These interfaces can be implemented by utilizing plain JDBC API to complete the functionality of our choice. The important callback interfaces are

1. RowExtractor (To pass single record by SQL query)
2. ResultExtractor (To process multiple records given by SQL query)
3. PreparedStatementCreator {
4. PreparedStatement setter To work with PreparedStatement
5. CallableStatementCreator etc., (To work with CallableStatement)

For Complete program please refer Spring DAO\MyEclise 8.X Projects\DAOConnectionPoolingMiniProject.