BY NAGOOR BABU

JDBC COURSE MATERIAL BY NAGOOR BABU

CONTACT US:

Mobile: +91- **8885 25 26 27**

+91- **7207 21 24 27/28**

US NUM: **4433326786**

Mail ID: **durgasoftonlinetraining@gmail.com**

WEBSITE: **www.durgasoftonline.com**

FLAT NO: 202, HMDA MYTRIVANUM, AMEERPET, HYDERABAD.,

Spring-DAO

-->DAO [Data Access Object], it is a design pattern, it able to provide very good environment to separate Data Access logic from Business Processing logic.
-->In enterprise Applications, to prepare Data Access Layer we will use DAO Design pattern.

Advantages of DAOs in Enterprise Applications:

1)We are able to hide all data access implementation from Business/ Service Layer.
2)It is very simple to switch data access layer from one data access tech to anther data access tech. with out giving any effect to Service/Business Layer, that is from JDBC to Hibernate,....
3)DAOs are able to provide centeralized Data Access in enterprise Application, it simplifies the maintanance of Enterprise Applications.
4)While preparing Service/Business layer Code Simplification is possible, that is, it is very simple to prepare Service/Business layer.
5)We are able get Standardization to prepare Data Access layer with DAOs.

Drawbacks With DAOs:

1)It adds one more layer to enterprise Application, may get maintanence problems.
2)Along with DAOs, we have to impement some other Design patterns like Factory Classes, DTO[Data Transfer Objects],....in enterprise applications.

Guidelines to prepare DAOs in Enterprise Applications:

1. Prepare a seperate DAO interface:

Prepare a seperate DAO interface with the required DAO methods, which must represent CRUD operations.

EX:

```
public interface StudentDao{
public String addStudent(Student std) throws DAOException;
public String updateStudent(Student std)throws DAOException;
public String deleteStudent(String sid)throws DAOException;
public List<Student> fildAllStudents()throws DAOException;
public Student findStudentByID(String sid)throws DAOException;
public Student findStudentByName(String sname)throws DAOException;
-----
-----
}
```

2.Provide an implementation to DAO interface:

```
public class StudentDAOImpl implements StudentDao{
---implementation for all StudentDao interface methods-----
---> We can provide our methods inorder to improve code reusability along with DAO interface methods---
}
```

3.Prepare DTOs as per the requirement:

```
public class Student{
private String sid;
private String sname;
private String saddr;
-----
-----
setXXX()  and getXXX()
-----
-----
}
```

4.Creat Factory Methods/ Factory Classes to generate DAOs:

```
public class StudentDaoFactory{
private static Student Dao dao;
static{
dao = new StudentDaoImpl();
}
public static StudentDao getStudentDao(){
return dao;
}
}
```

In Service layer
StudentDao dao = StudentDao.getStudentDao();

5) We must not cache DAO references, because, Factory classes/ Factory methods are providing single instances of DAO to the service layer, if DAO is required in multiple modules then it is requyired to create more than one DAO reference.

6) In case of DAOs, It is suggestible to interact with Databases by using Connection Pooling mechanisms, not by using DriverManager approach.

7) DAO is not threadsafe, we must not use DAOs in multi threadded environment.

8) In DAOs we can access close() method inorder to close the resources like connections,.... , so here, before calling close() method we must ensure that whether the resources are going to be released or not with our close() method call.

9) We have make sure that all the objects which are used by DAOs are following Java bean conventions or not.

Application on Servlets and JSPs with DAO:

Resources:

1. htmls:
   a)addform.html
   b)searchform.html
   c)deleteform.html
   d)existed.html
   e)notexisted.html
   f)success.html
   g)failure.html
   h)layout.html
   i)header.html
   j)menu.html
   k)welcome.html
   l)footer.html
2.jsps
   a)display.jsp
3.Servlets
   a)ControllerServlet
4.Services:
   a)StudentService
5.DAOs
   a)StudentDao
6)DTOs
   a)StudentTo
7)Factories
   a)ConnectionFactory
   b)StudentServiceFactory
   c)StudentDaoFactory
8)JARS:
   a)ojdbc6.jar

Design Patterns:
 a)DAO
 b)MVC
 c)DTO
 d)Factory

Example:

layout.html

```
<frameset rows="20%,65%,15%">
        <frame src="header.html"/>
        <frameset cols="20%,80%">
                <frame src="menu.html"/>
                <frame src="welcome.html" name="body"/>
        </frameset>
        <frame src="footer.html"/>
</frameset>
```

header.html

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<title>Insert title here</title>
</head>
<body bgcolor="maroon">
<center>
<font color="white" size="7">
<b>
DURGA SOFTWARE SOLUTIONS
</b>
</font>
</center>
</body>
</html>
```

footer.html

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<title>Insert title here</title>
</head>
<body bgcolor="blue">
<center>
<font color="white" size="5">
<b>
Durga Software Solutions, 202, Mitrivanam, Ameerpet, Hyd-38
</b>
</font>
</center>
</body>
</html>
```

menu.html

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<title>Insert title here</title>
</head>
<body bgcolor="lightyellow">
<center>
<h3>
<br><br>
<a href="./addform.html" target="body">Add Student</a><br><br>
<a href="./searchform.html" target="body">Search Student</a><br><br>
<a href="./deleteform.html" target="body">Delete Student</a>
</h3>
</center>
</body>
</html>
```

welcome.html

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<title>Insert title here</title>
</head>
<body bgcolor="lightblue">
<center>
<br><br><br>
<font color="red" size="6">
<b>
<marquee>
Welcome To Durga Software Solutions
</marquee>
</b>
</font>
</center>
</body>
</html>
```

addform.html

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<title>Insert title here</title>
</head>
<body bgcolor="lightblue">
<form method="POST" action="./controller">
<center>
<br><br><br>
<table>
<tr>
	<td>Student Id</td><td><input type="text" name="sid"/></td>
</tr>
<tr>
	<td>Student Name</td><td><input type="text" name="sname"/></td>
</tr>
```

```
<tr>
        <td>Student Address</td><td><input type="text" name="saddr"/></td>
</tr>
<tr>
        <td><input type="submit" value="ADD" name="button"/>
</tr>
</table>
</center>
</form>
</body>
</html>
```

searchform.html

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<title>Insert title here</title>
</head>
<body bgcolor="lightblue">
<form method="POST" action="./controller">
<br><br><br>
<center>
<table>
<tr>
        <td>Student Id</td><td><input type="text" name="sid"/></td>
</tr>
<tr>
        <td><input type="submit" value="SEARCH" name="button"/></td>
</tr>
</table>
</center>
</form>
</body>
</html>
```

deleteform.html

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
```

```
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<title>Insert title here</title>
</head>
<body bgcolor="lightblue">
<form method="POST" action="./controller">
<br><br><br>
<center>
<table>
<tr>
        <td>Student Id</td><td><input type="text" name="sid"/></td>
</tr>
<tr>
        <td><input type="submit" value="DELETE" name="button"/></td>
</tr>
</table>
</center>
</form>
</body>
</html>
```

display.jsp

```
<%@page import="com.durgasoft.to.StudentTo"%>
<%!
StudentTo sto;
%>
<%
sto = (StudentTo)request.getAttribute("sto");
%>
<html>
<body bgcolor="lightblue">
<center>
<br><br><br>
<table border = "1" bgcolor="white">
<tr>
        <td>Student Id</td><td><%= sto.getSid() %></td>
</tr>
<tr>
        <td>Student Name</td><td><%= sto.getSname() %></td>
</tr>
<tr>
        <td>Student Address</td><td><%= sto.getSaddr() %></td>
```

```
</tr>

</table>
</center>
</body>
</html>
```

existed.html

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<title>Insert title here</title>
</head>
<body bgcolor="lightblue">
<center>
<br><br><br>
<font color="red" size="6">
<b>
Student Existed Already
</b>
</font>
</center>
</body>
</html>
```

notexisted.html

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<title>Insert title here</title>
</head>
<body bgcolor="lightblue">
<center>
<br><br><br>
<font color="red" size="6">
<b>
Student Not Existed
```

```
</b>
</font>
</center>
</body>
</html>
```

success.html

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<title>Insert title here</title>
</head>
<body bgcolor="lightblue">
<center>
<br><br><br>
<font color="red" size="6">
<b>
Success
</b>
</font>
</center>
</body>
</html>
```

failure.html

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<title>Insert title here</title>
</head>
<body bgcolor="lightblue">
<center>
<br><br><br>
<font color="red" size="6">
<b>
Failure
</b>
```

```
</font>
</center>
</body>
</html>
```

ControllerServlet.java

```
package com.durgasoft.controller;
import java.io.IOException;

import javax.servlet.RequestDispatcher;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

import com.durgasoft.factory.StudentServiceFactory;
import com.durgasoft.services.StudentService;
import com.durgasoft.to.StudentTo;
public class ControllerServlet extends HttpServlet {
        private static final long serialVersionUID = 1L;
        protected void doPost(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException {
                String button_Label = request.getParameter("button");
                String status = "";
                RequestDispatcher rd = null;
                if(button_Label.equals("ADD")){
                        StudentService service = StudentServiceFactory.getStudentService();
                        StudentTo sto = new StudentTo();
                        sto.setSid(request.getParameter("sid"));
                        sto.setSname(request.getParameter("sname"));
                        sto.setSaddr(request.getParameter("saddr"));
                        status = service.addStudent(sto);
                        if(status.equals("success")){
                                rd = request.getRequestDispatcher("./success.html");
                                rd.forward(request, response);
                        }
                        if(status.equals("failure")){
                                rd = request.getRequestDispatcher("./failure.html");
                                rd.forward(request, response);
                        }
                        if(status.equals("existed")){
```

```
                rd = request.getRequestDispatcher("./existed.html");
                rd.forward(request, response);
            }
        }
        if(button_Label.equals("SEARCH")){
            String sid = request.getParameter("sid");
            StudentService service = StudentServiceFactory.getStudentService();
            StudentTo sto = service.searchStudent(sid);
            RequestDispatcher dispatcher = null;
            if( sto == null) {
                dispatcher = request.getRequestDispatcher("notexisted.html");
                dispatcher.forward(request, response);
            }else {
                request.setAttribute("sto", sto);
                dispatcher = request.getRequestDispatcher("display.jsp");
                dispatcher.forward(request, response);
            }
        }
        if(button_Label.equals("DELETE")){
            String sid = request.getParameter("sid");
            StudentService service = StudentServiceFactory.getStudentService();
            status = service.deleteStudent(sid);
            RequestDispatcher dispatcher = null;
            if( status.equals("success")) {
                dispatcher = request.getRequestDispatcher("success.html");
                dispatcher.forward(request, response);
            }
            if( status.equals("failure")) {
                dispatcher = request.getRequestDispatcher("failure.html");
                dispatcher.forward(request, response);
            }
            if( status.equals("notexisted")) {

                dispatcher = request.getRequestDispatcher("notexisted.html");
                dispatcher.forward(request, response);
            }
        }
    }

}
```

StudentService.java

```
package com.durgasoft.services;

import com.durgasoft.to.StudentTo;

public interface StudentService {
	public String addStudent(StudentTo sto);
	public StudentTo searchStudent(String sid);
	public String deleteStudent(String sid);
}
```

StudentServiceImpl.java

```
package com.durgasoft.services;

import com.durgasoft.dao.StudentDao;
import com.durgasoft.factory.StudentDaoFactory;
import com.durgasoft.to.StudentTo;

public class StudentServiceImpl implements StudentService {
	String status="";
	@Override
	public String addStudent(StudentTo sto) {
		StudentDao dao = StudentDaoFactory.getStudentDao();
		status = dao.add(sto);
		return status;
	}

	@Override
	public StudentTo searchStudent(String sid) {
		StudentTo sto = null;
		StudentDao dao = StudentDaoFactory.getStudentDao();
		sto = dao.search(sid);
		return sto;
	}

	@Override
	public String deleteStudent(String sid) {
		StudentDao dao = StudentDaoFactory.getStudentDao();
		status = dao.delete(sid);
		return status;
	}
}
```

StudentDao.java

```
package com.durgasoft.dao;

import com.durgasoft.to.StudentTo;

public interface StudentDao {
	public String add(StudentTo sto);
	public StudentTo search(String sid);
	public String delete(String sid);
}
```

StudentDaoImpl.java

```
package com.durgasoft.dao;

import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;

import com.durgasoft.factory.ConnectionFactrory;
import com.durgasoft.to.StudentTo;

public class StudentDaoImpl implements StudentDao{
	String status = "";
	@Override
	public String add(StudentTo sto) {
		try {
			Connection con = ConnectionFactrory.getConnection();
			PreparedStatement pst = con.prepareStatement("select * from student where 
sid = ?");
			pst.setString(1, sto.getSid());
			ResultSet rs = pst.executeQuery();
			boolean b = rs.next();
			if( b == true ) {
				status = "existed";
			}else {
				pst = con.prepareStatement("insert into student values(?,?,?)");
				pst.setString(1,sto.getSid());
				pst.setString(2, sto.getSname());
				pst.setString(3, sto.getSaddr());
```

```
                pst.executeUpdate();
                status = "success";
            }
        } catch (Exception e) {
            status = "failure";
            e.printStackTrace();
        }
        return status;
    }
    @Override
    public StudentTo search(String sid) {
        StudentTo sto = null;
        try {
            Connection con = ConnectionFactrory.getConnection();
            PreparedStatement pst = con.prepareStatement("select * from student where
sid = ?");
            pst.setString(1, sid);
            ResultSet rs = pst.executeQuery();
            boolean b = rs.next();
            if(b == true) {
                sto = new StudentTo();
                sto.setSid(rs.getString("SID"));
                sto.setSname(rs.getString("SNAME"));
                sto.setSaddr(rs.getString("SADDR"));
            }else {
                sto = null;
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
        return sto;
    }
    @Override
    public String delete(String sid) {
        try {
            Connection con = ConnectionFactrory.getConnection();
            PreparedStatement pst = con.prepareStatement("select * from student where
sid = ?");
            pst.setString(1, sid);
            ResultSet rs = pst.executeQuery();
            boolean b = rs.next();
            if(b == true) {
                pst = con.prepareStatement("delete from student where sid = ?");
```

```
                                        pst.setString(1, sid);
                                        pst.executeUpdate();
                                        status = "success";
                              }else {
                                        status = "notexisted";
                              }
                    } catch (Exception e) {
                              status = "failure";
                              e.printStackTrace();
                    }
                    return status;
          }
}
```

StudentTo.java

```
package com.durgasoft.to;

public class StudentTo {
          private String sid;
          private String sname;
          private String saddr;

          public String getSid() {
                    return sid;
          }
          public void setSid(String sid) {
                    this.sid = sid;
          }
          public String getSname() {
                    return sname;
          }
          public void setSname(String sname) {
                    this.sname = sname;
          }
          public String getSaddr() {
                    return saddr;
          }
          public void setSaddr(String saddr) {
                    this.saddr = saddr;
          }

}
```

StudentServiceFactory.java

```
package com.durgasoft.factory;

import com.durgasoft.services.StudentService;
import com.durgasoft.services.StudentServiceImpl;

public class StudentServiceFactory {
        private static StudentService service;
        static{
                service = new StudentServiceImpl();
        }
        public static StudentService getStudentService(){
                return service;
        }
}
```

StudentDaoFactory.java

```
package com.durgasoft.factory;

import com.durgasoft.dao.StudentDao;
import com.durgasoft.dao.StudentDaoImpl;

public class StudentDaoFactory {
        private static StudentDao dao;
        static {
                dao = new StudentDaoImpl();
        }
        public static StudentDao getStudentDao() {
                return dao;
        }
}
```

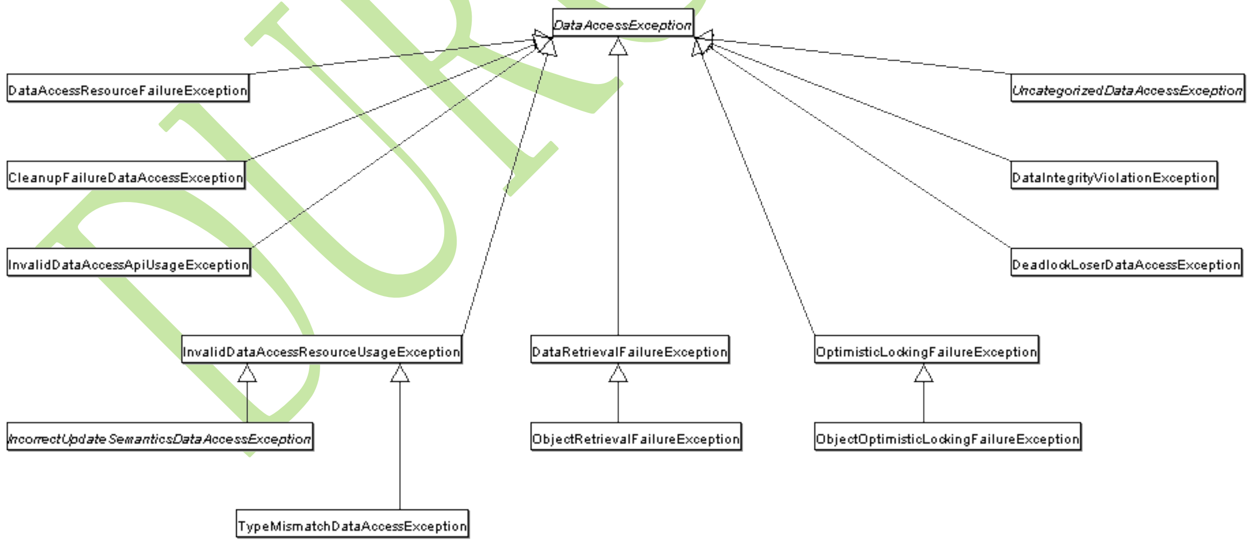
ConnectionFactory.java

```
package com.durgasoft.factory;

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

public class ConnectionFactrory {
        private static Connection con;
```

```
	static {
		try {
			Class.forName("oracle.jdbc.OracleDriver");
			con = DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:xe",
"system", "durga");
		} catch (Exception e) {
			e.printStackTrace();
		}

	}
	public static Connection getConnection() {
		return con;
	}
}
```

To provide support for DAOs kind of implementations in Spring Applications, Spring has provided a separate module called as “Spring DAO”.
Spring DAO modules has provided a set of predefined classes and interfaces in order to provide DAO support in the form of “org.springframework.dao” package.

Spring provides a convenient translation from technology-specific exceptions like SQLException , HibernateException,…… to its own exception class hierarchy with the DataAccessException as the root exception.


DataAccessException
DataAccessResourceFailureException
UncategorizedDataAccessException
CleanupFailureDataAccessException
DataIntegrityViolationException
InvalidDataAccessApiUsageException
DeadlockLoserDataAccessException
InvalidDataAccessResourceUsageException
DataRetrievalFailureException
OptimisticLockingFailureException
IncorrectUpdateSemanticsDataAccessException
ObjectRetrievalFailureException
ObjectOptimisticLockingFailureException
TypeMismatchDataAccessException

Spring JDBC

In Enterprise Applications, to prepare Data Access Layer or DAO layer Spring has provided Modules in the form of JDBC and ORM . IN ORM we may use no of ORM implementation tools like Hibernate, JPA, Ibatis,....

Q)In Enterprise Applications, to prepare Data Access Layer we have already Plain JDBC tech. then what is the requirement to go for Spring JDBC Module?

Ans:

1.To prepare Data Access Layer in enterprise applications, if we use JDBC then we must take explicit responsibility to prepare the steps load and register the driver, Establisg Connection, creating Statement, executing SQl Queries and closing the resources like ResultSet, Statement and Connection.

If we use Spring JDBC module to prepare Data Access Layer, we must take explicit responsibility to write and execute SQL Queries only, not to take any responsibility to load and register driver, connection establishment, creating Statement and closing the resources.

2.In case of Plain JDBC, almost all the exceptions are checked exceptions, we have to handle them explicitly by providing some java code.

In case of Spring JDBC module, all the internal checked exceptions are converted into Unchecked Exceptions which are defined by Spring DAO module , it is very simple to handle these unchecked Exceptions.

3.In Plain JDBC, limited support is available for Transactions.

In Spring JDBC Module, very good support is available for transactions, we may use Transaction module also to provide transactions.

4.In Plain JDBC, to hold the results we are able to use only ResultSet object, which is not implementing java.io.Serializable interface, which is not transferable in network.

In Spring JDBC, we are able to get results of SQL Queries in our required form like in the form of RowSet, Collections, ... which are implementing java.io.Serializable interface and which are transferable in Network.

5.In plain JDBC, we are able to get Connections either by using DriverManager or by using Datasource.

In Spring JDBC, we are able to get Connection internally by using Datasource only, that is through Connection Pooling only.

6.In plain JDBC, to map records to Bean objects in the form of Collection Object we have to write java code explicitly, no predefined support is provided by JDBC tech.

In case of Spring JDBC, to map Database records to Bean objects in the form of Collection Spring JDBC has provided predefined support in the form of "RowMapper".

7.In Plain JDBC, no callback interfaces support is available to create and execute the sql queries in PrfeparedStatement style.

In Spring JDBC, callback interfaces support is available to create and execute sql queries in PreparedStatement style.

To prepare Data Access Layer in enterprise applications, Spring JDBC module has provided the complete predefined library in the from of the following classes and interfaces in "org.springframework.jdbc" and its sub packages.

JdbcTemplate
NamedParameterJdbcTemplate
SimpleJdbcTemplate
SimpleJdbcInsert and SimpleJdbcCall
SQL Mapping through SQLUpdate and SQLInsert

In Spring Applications, if we want to JdbcTemplate[Jdbc Module] then we have to use the following steps.

1)Create DAO interface with the required methods.
2)Create DAO implementation class with implementation for DAO interface   methods.
3)In Configuration File provide configuration for DataSource class , JdbcTemplate class and DAO implementation class.
4)Prepare Test Application to access Dao methods.

IN Spring configuration file we have to configure DataSource with the following properties.

driverClassName
url
username
password

In Spring applications, to configure DataSource Spring has provided a seperate a predefined Datasource class in the form of "org.springframework.jdbc.datasource.DriverManagerDataSource", it is not suggestible for production mode, it is suggestible for testing mode of our applications , In spring applications, it is always suggestible to use third party Connection Pooling mechanisms like dbcp, C3P0, Proxool,..

JdbcTemplate class is providing basic environment to interact with Database like Loading Driver class, Getting Connection between Java application and DB, Creating Statement , PreparedStatement and CallableStatement and closing the connection with the help of the provided Datasource and JdbcTemplate class has provided the following methods to execute SQL Queries.

1)For Non Select sql queries and DML SQL queries

public int update(String query)

2)For DDL Sql Queries

public void execute(String query)

3)For Select sql queries

public int queryForInt(String query)
public int queryForLong(String query)
public String queryForString(String query)
public Object queryForObject(String query)
public List query(String query)
public List queryForList(String query)
public Map queryForMap(String query)
public RowSet queryForRowSet(String query)

While performing retrival operations to convert data from ResultSet object[records] to Bean objects Spring Framework has provided a predefined interface in the form of "org.springframework.jdbc.core.RowMapper" which contains the following method .

public Object mapRow(ResultSet rs, int rowCount)

Example

StudentDao.java

```
package com.durgasoft.dao;
import org.springframework.jdbc.core.JdbcTemplate;
import com.durgasoft.beans.Student;
public interface StudentDao {
        public void setJdbcTemplate(JdbcTemplate jdbcTemplate);
        public String add(Student std);
        public Student search(String sid);
        public String update(Student std);
        public String delete(String sid);

}
```

StudentDaoImpl.java

```
package com.durgasoft.dao;



import org.springframework.jdbc.core.JdbcTemplate;

import com.durgasoft.beans.Student;

public class StudentDaoImpl implements StudentDao {
        private JdbcTemplate jdbcTemplate;
        String status = "";

        @Override
        public void setJdbcTemplate(JdbcTemplate jdbcTemplate) {
                this.jdbcTemplate = jdbcTemplate ;

        }
        @Override
        public String add(Student std) {

                try {
                                jdbcTemplate.update("insert into student
values('"+std.getSid()+"','"+std.getSname()+"','"+std.getSaddr()+"')");
                                status = "success";

                }catch(Exception e) {
                        status = "failure";
                        e.printStackTrace();
                }

                return status;
        }

        @Override
        public Student search(String sid) {
                Student std = null;
                try {
```

```
			std = jdbcTemplate.queryForObject("select * from student where 
sid='"+sid+"'", new StudentMapper());
		} catch (Exception e) {
			e.printStackTrace();
		}
		return std;
	}

	@Override
	public String update(Student std) {
		try {
			jdbcTemplate.update("update student set 
sname='"+std.getSname()+"',saddr='"+std.getSaddr()+"' where sid='"+std.getSid()+"'");
			status="success";
			jdbcTemplate.qu
		}catch(Exception e) {
			status="failure";
			e.printStackTrace();
		}
		return status;
	}

	@Override
	public String delete(String sid) {
		try {
			jdbcTemplate.update("delete from student where sid='"+sid+"'");
			status="success";
		} catch (Exception e) {
			status="failure";
			e.printStackTrace();
		}
		return status;
	}
}
```

Student.java

```
package com.durgasoft.beans;

public class Student {
	private String sid;
	private String sname;
```

```
	private String saddr;

	setXXX() and getXXX()

}
```

StudentMapper.java

```
package com.durgasoft.dao;

import java.sql.ResultSet;
import java.sql.SQLException;

import org.springframework.jdbc.core.RowMapper;

import com.durgasoft.beans.Student;

public class StudentMapper implements RowMapper<Student> {
@Override
public Student mapRow(ResultSet rs, int row_No) throws SQLException {

		Student std = new Student();
		std.setSid(rs.getString("SID"));
		std.setSname(rs.getString("SNAME"));
		std.setSaddr(rs.getString("SADDR"));
		return std;

}
}
```

Test.java

```
package com.durgasoft.test;

import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;

import com.durgasoft.beans.Student;
import com.durgasoft.dao.StudentDao;

public class Test {

	public static void main(String[] args)throws Exception {
```

```
ApplicationContext context = new ClassPathXmlApplicationContext("applicationContext.xml");
		StudentDao dao = (StudentDao) context.getBean("studentDao");
		//----Inserting Records------
		Student std = new Student();
		std.setSid("S-111");
		std.setSname("Durga");
		std.setSaddr("Hyd");
		String status = dao.add(std);
		System.out.println("Student Insertion :"+status);

		std.setSid("S-222");
		std.setSname("Anil");
		std.setSaddr("Hyd");
		status = dao.add(std);
		System.out.println("Student Insertion :"+status);

		std.setSid("S-333");
		std.setSname("Sekhar");
		std.setSaddr("Hyd");
		status = dao.add(std);
		System.out.println("Student Insertion :"+status);
		System.out.println();

		//----Retriving Record-----
		Student std1 = dao.search("S-111");
		if(std1 == null) {
			System.out.println("Student Search Status :NotExisted");
		}else {
			System.out.println("Student Details");
			System.out.println("--------------------");
			System.out.println("Student Id      :"+std1.getSid());
			System.out.println("Student Name    :"+std1.getSname());
			System.out.println("Student Address :"+std1.getSaddr());
		}
			System.out.println();

		//----Updating a Record------
		std.setSid("S-111");
		std.setSname("XXX");
		std.setSaddr("YYY");
		status = dao.update(std);
		System.out.println("Student Updation :"+status);
		System.out.println();
```

```
				//----Deleting a record-----
				status = dao.delete("S-111");
				System.out.println("Student Deletion :"+status);
		}
}
```

In Spring JDBC Applications, we will use positional parameters[?] also in sql queries which we are providing along with JdbcTemplate class provided query execution methods.

If we provide positional parameters in sql queries then JdbcTemplate class will use "PreparedStatement" internally to execute sql query instead of Statement.

To provide values to the Positional parameters in SQL Queries we have to use Object[] with values as parametyer to all JdbcTemplate class provided query execution methods.

```
public int update(String query, Object[] param_Values)
public int queryForInt(String query, Object[] param_Values)
public long queryForLong(String query, Object[] param_Values)
public Object queryForObject(String query, Object[] param_Values, RowMapper rm)
-----
-----
-----
```

Ex:

```
String query = "insert into student values(?,?,?)";
int rowCount = jdbcTemplate.update(query, new Object[]{"S-111", "AAA", "Hyd"});
```

Example:

StudentDao.java

```
package com.durgasoft.dao;


import org.springframework.jdbc.core.JdbcTemplate;

import com.durgasoft.beans.Student;

public interface StudentDao {
	public void setJdbcTemplate(JdbcTemplate jdbcTemplate);
	public String add(Student std);
	public Student search(String sid);
```

```
	public String update(Student std);
	public String delete(String sid);

}
```

StudentDaoImpl.java

```
package com.durgasoft.dao;



import org.springframework.jdbc.core.JdbcTemplate;

import com.durgasoft.beans.Student;

public class StudentDaoImpl implements StudentDao {
	private JdbcTemplate jdbcTemplate;
	String status = "";

	@Override
	public void setJdbcTemplate(JdbcTemplate jdbcTemplate) {
		this.jdbcTemplate = jdbcTemplate ;

	}
	@Override
	public String add(Student std) {

		try {
			String query = "insert into student values(?,?,?)";
			jdbcTemplate.update(query, new Object[] {std.getSid(), std.getSname(),
std.getSaddr()});
			status = "success";

		}catch(Exception e) {
			status = "failure";
			e.printStackTrace();
		}

		return status;
	}

	@Override
	public Student search(String sid) {
```

```
                    Student std = null;
                    try {
                              std = jdbcTemplate.queryForObject("select * from student where sid=?", new
Object[] {sid}, new StudentMapper());
                    } catch (Exception e) {
                              e.printStackTrace();
                    }
                    return std;
          }

          @Override
          public String update(Student std) {
                    try {
                              jdbcTemplate.update("update student set sname=?, saddr=? where
sid=?",new Object[] { std.getSname(), std.getSaddr(), std.getSid()});
                              status="success";

                    }catch(Exception e) {
                              status="failure";
                              e.printStackTrace();
                    }
                    return status;
          }

          @Override
          public String delete(String sid) {
                    try {
                              jdbcTemplate.update("delete from student where sid=?", new Object[] {sid});
                              status="success";
                    } catch (Exception e) {
                              status="failure";
                              e.printStackTrace();
                    }
                    return status;
          }
}
```

applicationContext.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xmlns:context="http://www.springframework.org/schema/context"
```

```
   xsi:schemaLocation="
      http://www.springframework.org/schema/beans
      http://www.springframework.org/schema/beans/spring-beans.xsd
      http://www.springframework.org/schema/context
      http://www.springframework.org/schema/context/spring-context.xsd">

   <bean id="studentDao" class="com.durgasoft.dao.StudentDaoImpl">
      <property name="jdbcTemplate" ref="jdbcTemplate"/>
   </bean>
        <bean id="jdbcTemplate" class="org.springframework.jdbc.core.JdbcTemplate">
                <property name="dataSource" ref="dataSource"/>
        </bean>

   <bean id="dataSource"
class="org.springframework.jdbc.datasource.DriverManagerDataSource" >
      <property name="driverClassName" value="oracle.jdbc.OracleDriver"/>
      <property name="url" value="jdbc:oracle:thin:@localhost:1521:xe"/>
      <property name="username" value="system"/>
      <property name="password" value="durga"/>
   </bean>


</beans>
```

StudentMapper.java

```
package com.durgasoft.dao;

import java.sql.ResultSet;
import java.sql.SQLException;

import org.springframework.jdbc.core.RowMapper;

import com.durgasoft.beans.Student;

public class StudentMapper implements RowMapper<Student> {
@Override
public Student mapRow(ResultSet rs, int row_No) throws SQLException {

                Student std = new Student();
                std.setSid(rs.getString("SID"));
                std.setSname(rs.getString("SNAME"));
                std.setSaddr(rs.getString("SADDR"));
```

```
                return std;

}
}
```

Test.java

```
package com.durgasoft.test;

import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;

import com.durgasoft.beans.Student;
import com.durgasoft.dao.StudentDao;

public class Test {

        public static void main(String[] args)throws Exception {
                ApplicationContext context = new
ClassPathXmlApplicationContext("applicationContext.xml");
                StudentDao dao = (StudentDao) context.getBean("studentDao");
                //----Inserting Records------
                Student std = new Student();
                std.setSid("S-111");
                std.setSname("Durga");
                std.setSaddr("Hyd");
                String status = dao.add(std);
                System.out.println("Student Insertion :"+status);

                std.setSid("S-222");
                std.setSname("Anil");
                std.setSaddr("Hyd");
                status = dao.add(std);
                System.out.println("Student Insertion :"+status);

                std.setSid("S-333");
                std.setSname("Sekhar");
                std.setSaddr("Hyd");
                status = dao.add(std);
                System.out.println("Student Insertion :"+status);
                System.out.println();

                //----Retriving Record-----
```

```
                    Student std1 = dao.search("S-111");
                    if(std1 == null) {
                              System.out.println("Student Search Status :NotExisted");
                    }else {
                              System.out.println("Student Details");
                              System.out.println("--------------------");
                              System.out.println("Student Id       :"+std1.getSid());
                              System.out.println("Student Name     :"+std1.getSname());
                              System.out.println("Student Address :"+std1.getSaddr());
                    }
                              System.out.println();

                    //----Updating a Record------
                    std.setSid("S-111");
                    std.setSname("XXX");
                    std.setSaddr("YYY");
                    status = dao.update(std);
                    System.out.println("Student Updation :"+status);
                    System.out.println();

                    //----Deleting a record-----
                    status = dao.delete("S-111");
                    System.out.println("Student Deletion :"+status);
          }

}
```

NamedParameterJdbcTemplate

NamedParameterJdbcTemplate class is same as JdbcTemplate class , but, NamedParameterJdbcTemplate class is able to define and run sql queries with Named Parameters instead of positional parameters.
EX:
String query = "insert into student values(:sid, :sname, :saddr)";

Where :sid, :sname, :saddr are named parameters for which we have to provide values.

In case of NamedParameterJdbcTemplate , we are able to provide values to the named parameters in the following two approaches.
1.By Using Map directly.
2.By using SqlParameterSource interface.

1.By Using Map directly.

```
String query = "insert into student values(:sid, :sname, :saddr)";
Map map = new HashMap();
map.put("sid", "S-111");
map.put("sname", "AAA");
map.put("saddr", "Hyd");
namedParameterJdbcTemplate.update(query, map);
```

2.By using SqlParameterSource interface.

To provide values to the Named parameters Spring has provided the following two implementation classes for SqlParameterSoure interface.

a)MapSqlParameterSource
b)BeanPropertySqlParameterSource

To provide values to the named parameters if we want to use MapSqlParameterSource then first we have to create object for MapSqlParameterSource and we have to use the following method to add values to the named parameters.

```
public MapSqlParameterSource addValue(String name, Object val)
```

EX:

```
String query = "insert into student values(:sid, :sname, :saddr)";
SqlParameterSource param_Source = new MapSqlParameterSource("sid", "S-111");
param_Source = param_Source.addValue("sname", "AAA");
param_Source = param_Source.addValue("saddr", "Hyd");
namedParameterJdbcTemplate.update(query, param_Source);
```

To provide values to the named parameters if we want to use BeanPropertySqlParameterSource then first we have to create bean object with data then we have to create Object for BeanPropertySqlParameterSource with the generated Bean reference then provide BeanPropertySqlParameterSource object to query methods.

EX:

```
String query = "insert into student values(:sid, :sname, :saddr)";
Student std = new Student();
std.setSid("S-111");
std.setSname("AAA");
std.setSaddr("Hyd");
SqlParameterSource param_Source = new BeanPropertySqlParameterSource(std );
```

```
namedParameterJdbcTemplate.update(query, param_Source);
```

Note: JdbcTemplate is allowing DataSource object injection through setter method, but, NamedParameterJdbcTemplate class is allowing DataSource object injection through Constructor Dependency Injection.

Example

CustomerDao.java

```
package com.durgasoft.dao;

import com.durgasoft.beans.Customer;

public interface CustomerDao {

	public String add(Customer c);
	public Customer search(String cid);
	public String update(Customer c);
	public String delete(String cid);
}
```

CustomerDaoImpl.java

```
package com.durgasoft.dao;

import java.util.HashMap;
import java.util.Map;

import org.springframework.jdbc.core.namedparam.BeanPropertySqlParameterSource;
import org.springframework.jdbc.core.namedparam.MapSqlParameterSource;
import org.springframework.jdbc.core.namedparam.NamedParameterJdbcTemplate;
import org.springframework.jdbc.core.namedparam.SqlParameterSource;
import org.springframework.jdbc.datasource.DriverManagerDataSource;

import com.durgasoft.beans.Customer;

public class CustomerDaoImpl implements CustomerDao {
	String status = "";
	private NamedParameterJdbcTemplate namedParameterJdbcTemplate;
	public void setNamedParameterJdbcTemplate(NamedParameterJdbcTemplate 
namedParameterJdbcTemplate) {
		this.namedParameterJdbcTemplate = namedParameterJdbcTemplate;
```

```
	}
	@Override
	public String add(Customer c) {
		String query = "insert into customer values(:cid, :cname, :caddr)";
		Map<String, Object> map = new HashMap<String, Object>();
		map.put("cid", c.getCid());
		map.put("cname", c.getCname());
		map.put("caddr", c.getCaddr());
		namedParameterJdbcTemplate.update(query, map);
		return "SUCCESS";
	}

	@Override
	public Customer search(String cid) {
		String query = "select * from customer where cid=:cid";

		Map<String, Object> map = new HashMap<String, Object>();
		map.put("cid", cid);
		//SqlParameterSource paramSource = new MapSqlParameterSource("cid", cid);

		Customer c = namedParameterJdbcTemplate.queryForObject(query, map, new
CustomerMapper());

		return c;
	}

	@Override
	public String update(Customer c) {
		String query = "update customer set  CNAME=:cname, CADDR=:caddr where
CID=:cid";
		SqlParameterSource paramSource = new BeanPropertySqlParameterSource(c);
		namedParameterJdbcTemplate.update(query, paramSource);

		return "SUCCESS";
	}
	@Override
	public String delete(String cid) {
		String query = "delete from customer where cid=:cid";
		SqlParameterSource paramSource = new MapSqlParameterSource("cid", cid);
		namedParameterJdbcTemplate.update(query, paramSource);
		return "SUCCESS";
	}
}
```

Customer.java

```
package com.durgasoft.beans;

public class Customer {
	private String cid;
	private String cname;
	private String caddr;

	public String getCid() {
		return cid;
	}
	public void setCid(String cid) {
		this.cid = cid;
	}
	public String getCname() {
		return cname;
	}
	public void setCname(String cname) {
		this.cname = cname;
	}
	public String getCaddr() {
		return caddr;
	}
	public void setCaddr(String caddr) {
		this.caddr = caddr;
	}

}
```

CustomerMapper.java

```
package com.durgasoft.dao;

import java.sql.ResultSet;
import java.sql.SQLException;

import org.springframework.jdbc.core.RowMapper;

import com.durgasoft.beans.Customer;

public class CustomerMapper implements RowMapper<Customer> {
@Override
```

```
public Customer mapRow(ResultSet rs, int row_No) throws SQLException {

                Customer c = new Customer();
                c.setCid(rs.getString("CID"));
                c.setCname(rs.getString("CNAME"));
                c.setCaddr(rs.getString("CADDR"));
                return c;

        }
}
```

applicationContext.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xmlns:context="http://www.springframework.org/schema/context"
   xsi:schemaLocation="
      http://www.springframework.org/schema/beans
      http://www.springframework.org/schema/beans/spring-beans.xsd
      http://www.springframework.org/schema/context
      http://www.springframework.org/schema/context/spring-context.xsd">

   <bean id="customerDao" class="com.durgasoft.dao.CustomerDaoImpl">
      <property name="namedParameterJdbcTemplate" ref="namedParameterJdbcTemplate"/>
   </bean>
        <bean id="namedParameterJdbcTemplate"
class="org.springframework.jdbc.core.namedparam.NamedParameterJdbcTemplate">
                <constructor-arg ref="dataSource"/>
        </bean>
   <bean id="dataSource"
class="org.springframework.jdbc.datasource.DriverManagerDataSource">
      <property name="driverClassName" value="oracle.jdbc.OracleDriver"/>
      <property name="url" value="jdbc:oracle:thin:@localhost:1521:xe"/>
      <property name="username" value="system"/>
      <property name="password" value="durga"/>
   </bean>

</beans>
```

Test.java

```
package com.durgasoft.test;
```

```
import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;

import com.durgasoft.beans.Customer;
import com.durgasoft.dao.CustomerDao;

public class Test {

        public static void main(String[] args)throws Exception {
                ApplicationContext context = new
ClassPathXmlApplicationContext("applicationContext.xml");
                CustomerDao dao = (CustomerDao)context.getBean("customerDao");
                Customer c = new Customer();
                c.setCid("C-111");
                c.setCname("AAA");
                c.setCaddr("Hyd");
                String status = dao.add(c);
                System.out.println("Student Insertion :"+status);

                Customer c1 = dao.search("C-111");
                System.out.println("Customer Details");
                System.out.println("--------------------");
                System.out.println("Customer Id      :"+c1.getCid());
                System.out.println("Customer Name    :"+c1.getCname());
                System.out.println("Customer Address :"+c1.getCaddr());
                System.out.println();
                Customer c2 = new Customer();
                c2.setCid("C-111");
                c2.setCname("BBB");
                c2.setCaddr("Sec");
                status = dao.update(c2);
                System.out.println("Student Updation Status :"+status);
                Customer c3 = dao.search("C-111");
                System.out.println("Customer Updated Details");
                System.out.println("--------------------");
                System.out.println("Customer Id      :"+c3.getCid());
                System.out.println("Customer Name    :"+c3.getCname());
                System.out.println("Customer Address :"+c3.getCaddr());
                System.out.println();
                status = dao.delete("C-111");
                System.out.println("Student Deletion Status :"+status);
        }
}
```

SimpleJdbcTemplate:

In Spring JDBC module, the main intention of SimpleJdbcTemplate class is to provide support for JDK5.0 version feratures like Auto Boxing, Auto Unboxing, Var-Arg methods,.....

SimpleJdbcTemplate class was provided in Spring2.5 version only and it was deprecated in the later versions Spring3.x and Spring4.x , in Spring5.x version SimpleJdbcTemplate class was removed.

If we want to use SimpleJdbcTemplate class we have to use Spring2.5 version jar files in Spring applications.

To execute SQL queries , SimpleJdbcTemplate class has provided the following methods.

public Object execute(String sqlQuery)
Note: To use this method we have to get JdbcOperations class by using getJdbcOperations() method.

public int update(String query, Object ... params)
public Object queryForInt(String query, Object ... params)
public Object queryForLong(String query, Object ... params)
public Object query(String query, Object ... params)
public Object queryForObject(String query,Object ... params)

Note: In case of SimpleJdbcTemplate class, to perform retrival operations, we have to use "ParameterizedRowMapper" inplace of RowMapper interface.

Example:

EmployeeDao.java

```
package com.durgasoft.dao;

import com.durgasoft.beans.Employee;

public interface EmployeeDao {
	public String add(Employee emp);
	public Employee search(int eno);
	public String update(Employee emp);
	public String delete(int eno);
}
```

EmployeeDaoImpl.java

```
package com.durgasoft.dao;

import org.springframework.jdbc.core.simple.SimpleJdbcTemplate;

import com.durgasoft.beans.Employee;

public class EmployeeDaoImpl implements EmployeeDao{
        private SimpleJdbcTemplate simpleJdbcTemplate;

        String status = "";
        public void setSimpleJdbcTemplate(SimpleJdbcTemplate simpleJdbcTemplate) {
                this.simpleJdbcTemplate = simpleJdbcTemplate;

        }
        @Override
        public String add(Employee emp) {
                String query = "insert into emp1 
values("+emp.getEno()+",'"+emp.getEname()+"',"+emp.getEsal()+",'"+emp.getEaddr()+"')";
                simpleJdbcTemplate.getJdbcOperations().execute(query);
                status = "SUCCESS";
                return status;
        }
        @Override
        public Employee search(int eno) {
                String query = "select * from emp1 where eno=?";
                Employee emp = simpleJdbcTemplate.queryForObject(query, new 
EmployeeMapper(), eno);

                return emp;
        }
        @Override
        public String update(Employee emp) {
                String query = "update emp1 set ename=?, esal =?, eaddr=? where eno=?";
                simpleJdbcTemplate.update(query, emp.getEname(), emp.getEsal(), 
emp.getEaddr(), emp.getEno());
                status = "SUCCESS";
                return status;
        }
        @Override
        public String delete(int eno) {
                String query = "delete from emp1 where eno = ?";
```

```
                simpleJdbcTemplate.update(query, eno);
                status = "SUCCESS";

                return status;
        }
}
```

Employee.java

```
package com.durgasoft.beans;

public class Employee {
        private int eno;
        private String ename;
        private float esal;
        private String eaddr;

        public int getEno() {
                return eno;
        }
        public void setEno(int eno) {
                this.eno = eno;
        }
        public String getEname() {
                return ename;
        }
        public void setEname(String ename) {
                this.ename = ename;
        }
        public float getEsal() {
                return esal;
        }
        public void setEsal(float esal) {
                this.esal = esal;
        }
        public String getEaddr() {
                return eaddr;
        }
        public void setEaddr(String eaddr) {
                this.eaddr = eaddr;
        }

}
```

EmployeeMapper.java

```
package com.durgasoft.dao;

import java.sql.ResultSet;
import java.sql.SQLException;

import org.springframework.jdbc.core.RowMapper;
import org.springframework.jdbc.core.simple.ParameterizedRowMapper;

import com.durgasoft.beans.Employee;

public class EmployeeMapper implements ParameterizedRowMapper<Employee> {
@Override
public Employee mapRow(ResultSet rs, int row_No) throws SQLException {

                Employee emp = new Employee();
                emp.setEno(rs.getInt("ENO"));
                emp.setEname(rs.getString("ENAME"));
                emp.setEsal(rs.getFloat("ESAL"));
                emp.setEaddr(rs.getString("EADDR"));
                return emp;

     }
}
```

applicationContext.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xmlns:context="http://www.springframework.org/schema/context"
   xsi:schemaLocation="
      http://www.springframework.org/schema/beans
      http://www.springframework.org/schema/beans/spring-beans.xsd
      http://www.springframework.org/schema/context
      http://www.springframework.org/schema/context/spring-context.xsd">

   <bean id="employeeDao" class="com.durgasoft.dao.EmployeeDaoImpl">
      <property name="simpleJdbcTemplate" ref="simpleJdbcTemplate"/>
   </bean>
         <bean id="simpleJdbcTemplate"
class="org.springframework.jdbc.core.simple.SimpleJdbcTemplate">
```

```
                <constructor-arg ref="dataSource"/>
        </bean>
   <bean id="dataSource"
class="org.springframework.jdbc.datasource.DriverManagerDataSource">
      <property name="driverClassName" value="oracle.jdbc.OracleDriver"/>
      <property name="url" value="jdbc:oracle:thin:@localhost:1521:xe"/>
      <property name="username" value="system"/>
      <property name="password" value="durga"/>
   </bean>


</beans>
```

Test.java

```
package com.durgasoft.test;

import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;

import com.durgasoft.beans.Employee;
import com.durgasoft.dao.EmployeeDao;

public class Test {

        public static void main(String[] args)throws Exception {
                ApplicationContext context = new
ClassPathXmlApplicationContext("applicationContext.xml");
                EmployeeDao dao = (EmployeeDao)context.getBean("employeeDao");
                Employee emp = new Employee();
                emp.setEno(111);
                emp.setEname("AAA");
                emp.setEsal(5000);
                emp.setEaddr("Hyd");
                String status1 = dao.add(emp);
                System.out.println("Employee Insertion Status :"+status1);
                System.out.println();
                Employee emp1 = dao.search(111);
                System.out.println("Employee Details");
                System.out.println("--------------------");
                System.out.println("Employee Number  :"+emp1.getEno());
                System.out.println("Employee Name    :"+emp1.getEname());
                System.out.println("Employee Salary  :"+emp1.getEsal());
```

```
                    System.out.println("Employee Address :"+emp1.getEaddr());
                    System.out.println();
                    Employee emp2 = new Employee();
                    emp2.setEno(111);
                    emp2.setEname("BBB");
                    emp2.setEsal(6000);
                    emp2.setEaddr("Sec");
                    String status2 = dao.update(emp2);
                    System.out.println("Employee Updation Status :"+status2);
                    System.out.println();
                    String status3 = dao.delete(111);
                    System.out.println("Employee Deletion Status :"+status3);
          }
}
```

DAO Support Classes:

In Spring JDBC , we have to prepare DAO implementation classes with XXXTemplate property and the corresponding setXXX() method inorder to inject XXXTemplate class.

In Spring JDBC applications, if we want to get XXXTemplate classes with out declaring Template properties and corresponding setXXX() methods we have to use DAO Support classes provided Spring JDBC module.

There are three types of DAOSupport classes inorder to get Template object in DAO classes.

1.JdbcDaoSupport
2.NamedParameterJdbcDaoSupport
3.SimpleJdbcDaoSupport

Where JdbcDaoSupport class will provide JdbcTemplate reference in DAO classes by using the following method.

public JdbcTemplate getJdbcTemplate()

Where NamesParameterJdbcDaoSupport class will provide NamedParameterJdbcTempate reference in DAO classes by using the following method.

public NamedParameterJdbcTemplate getNamedparameterJdbctemplate()

Where SimpleJdbcDaoSupport class is able to provide SimpleJdbctemplate reference in Dao class by using the following method.

```
public SimpleJdbcTemplate getSimpleJdbcTemplate()
```

EX:

```
public class EmployeeDaoImpl extends JdbcDaoSupport implements EmployeeDao{

public String insert(int eno, String ename, float esal, String eaddr){
getJdbcTemplate().update("insert into emp1 values("+eno+",'"+ename+"',"+esal+",'"+eaddr+"')");
return "SUCCESS";
}
----
----
}
```

Spring Batch Updations

Batch Processing:

To perform Batch Updations in Spring JDBC we have to use the following method from JdbcTemplate class.

```
public int[] batchUpdate(String sql_prepared_Statement, BatchPreparedStatementSetter setter)
```

Where BatchPreparedStatementSetter interface contains the following two methods

```
public void setValues(PreparedStatement ps, int index)
public int getBatchSize()
```

Where setValues() method will be executed for each and every record to set values to the positional parameters existed in PreparedStatement object by getting values from the provided List.

Employee.java

```
package com.durgasoft.beans;

public class Employee {
	private int eno;
	private String ename;
	private float esal;
	private String eaddr;
```

```
	public int getEno() {
		return eno;
	}
	public void setEno(int eno) {
		this.eno = eno;
	}
	public String getEname() {
		return ename;
	}
	public void setEname(String ename) {
		this.ename = ename;
	}
	public float getEsal() {
		return esal;
	}
	public void setEsal(float esal) {
		this.esal = esal;
	}
	public String getEaddr() {
		return eaddr;
	}
	public void setEaddr(String eaddr) {
		this.eaddr = eaddr;
	}


}
```

EmployeeDao.java

```
package com.durgasoft.dao;

import java.util.List;

import com.durgasoft.beans.Employee;

public interface EmployeeDao {
	public int[] insert(List<Employee> list);
}
```

EmployeeDaoImpl.java

```
package com.durgasoft.dao;

import java.sql.PreparedStatement;
import java.sql.SQLException;
import java.util.List;

import javax.sql.DataSource;

import org.springframework.jdbc.core.BatchPreparedStatementSetter;
import org.springframework.jdbc.core.JdbcTemplate;

import com.durgasoft.beans.Employee;

public class EmployeeDaoImpl implements EmployeeDao{
        private DataSource dataSource;
        JdbcTemplate jdbcTemplate;
        public void setDataSource(DataSource dataSource) {
                this.dataSource = dataSource;
                jdbcTemplate = new JdbcTemplate(dataSource);
        }
        @Override
        public int[] insert(List<Employee> list) {
                int[] rowCounts = null;
                try {
                        String sql = "insert into emp1 values(?,?,?,?)";
                        rowCounts = jdbcTemplate.batchUpdate(sql, new
BatchPreparedStatementSetter() {
                                @Override
                                public void setValues(PreparedStatement ps, int i) throws
SQLException {
                                        ps.setInt(1, list.get(i).getEno());
                                        ps.setString(2, list.get(i).getEname());
                                        ps.setFloat(3, list.get(i).getEsal());
                                        ps.setString(4, list.get(i).getEaddr());

                                }
                                @Override
                                public int getBatchSize() {
                                        // TODO Auto-generated method stub
                                        return list.size();
                                }
                        });
                } catch (Exception e) {
```

CONTACT US:
Mobile: +91- 8885 25 26 27
+91- 7207 21 24 27/28
US NUM: 4433326786
Mail ID: durgasoftonlinetraining@gmail.com
WEBSITE: www.durgasoftonline.com
FLAT NO: 202, HMDA MYTRIVANUM, AMEERPET, HYDERABAD.,

```
                        e.printStackTrace();
                }
                return rowCounts;
        }
}
```

Jdbc.properties

```
jdbc.driverClassName = oracle.jdbc.OracleDriver
jdbc.url = jdbc:oracle:thin:@localhost:1521:xe
jdbc.username = system
jdbc.password = durga
```

applicationContext.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xmlns:context="http://www.springframework.org/schema/context"
   xsi:schemaLocation="
      http://www.springframework.org/schema/beans
      http://www.springframework.org/schema/beans/spring-beans.xsd
      http://www.springframework.org/schema/context
      http://www.springframework.org/schema/context/spring-context.xsd">

   <bean id="empDao" class="com.durgasoft.dao.EmployeeDaoImpl">
      <property name="dataSource" ref="dataSource"/>
   </bean>

   <bean id="dataSource"
class="org.springframework.jdbc.datasource.DriverManagerDataSource">
      <property name="driverClassName" value="${jdbc.driverClassName}"/>
      <property name="url" value="${jdbc.url}"/>
      <property name="username" value="${jdbc.username}"/>
      <property name="password" value="${jdbc.password}"/>
   </bean>

   <context:property-placeholder location="jdbc.properties"/>

</beans>
```

Test.java

```
package com.durgasoft.test;

import java.util.ArrayList;
import java.util.List;

import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;

import com.durgasoft.beans.Employee;
import com.durgasoft.dao.EmployeeDao;

public class Test {

	public static void main(String[] args) {
		ApplicationContext context = new
ClassPathXmlApplicationContext("applicationContext.xml");
		EmployeeDao dao = (EmployeeDao)context.getBean("empDao");
		List<Employee> list = new ArrayList<Employee>();
		Employee e1 = new Employee();
		e1.setEno(111);
		e1.setEname("AAA");
		e1.setEsal(5000);
		e1.setEaddr("Hyd");
		list.add(e1);
		Employee e2 = new Employee();
		e2.setEno(222);
		e2.setEname("BBB");
		e2.setEsal(6000);
		e2.setEaddr("Hyd");
		list.add(e2);
		Employee e3 = new Employee();
		e3.setEno(333);
		e3.setEname("CCC");
		e3.setEsal(6000);
		e3.setEaddr("Hyd");
		list.add(e3);

		int[] rowCounts = dao.insert(list);
		for(int i = 0; i<rowCounts.length; i++) {
			System.out.println(rowCounts[i]);
		}
```

```
        }
}
```

Stored Procedure and Functions in Spring JDBC:

If we want to access stored procedures and functions ehich are available at database from Spring Jdbc application then we have to use "SimpleJdbcCall".

To use SimpleJdbcCall in Spring Jdbc applications we have to use the following steps.
1)Create DAO interface and its implementation class.
2)IN DAO implementation class, we have to declare DataSource and JdbcTemplate and its respective setter method .
3)In side setter method we have to create SimpleJdbcCall object.

```
   SimpleJdbcCall jdbcCall = new SimpleJdbcCall();
   jdbcCall.withProcedureName("proc_Name");
```

4)Configure DataSource and DAO implementation class in beans configuration file.
5)Access "execute" method by passing IN type parameters values in the form of "SQLParameterSource".

```
  public Map execute(Map m)
  pubhlic Map execute(SqlParameterSource paramSource)
  public Map execute(Object ... obj)
```

Procedure to copy at Database:

```
create or replace procedure getSalary(no IN number, sal OUT
int)
AS
BEGIN
select esal into sal from emp1 where eno = no;
END getSalary;
/
```

Employee.java

```
package com.durgasoft.beans;

public class Employee {
private int eno;
private String ename;
private float esal;
private String eaddr;
public int getEno() {
return eno;
}
public void setEno(int eno) {
this.eno = eno;
}
public String getEname() {
return ename;
}
public void setEname(String ename) {
this.ename = ename;
}
public float getEsal() {
return esal;
}
public void setEsal(float esal) {
this.esal = esal;
}
public String getEaddr() {
return eaddr;
}
public void setEaddr(String eaddr) {
this.eaddr = eaddr;
}


}
```

EmployeeDao.java

```
package com.durgasoft.dao;

import java.util.Map;

import com.durgasoft.beans.Employee;

public interface EmployeeDao {
public void create(Employee emp);
public Object getEmployeeSalary(int eno);
}
```

EmployeeDaoImpl.java

```
package com.durgasoft.dao;

import java.util.Map;

import javax.sql.DataSource;

import org.springframework.jdbc.core.JdbcTemplate;
import
org.springframework.jdbc.core.namedparam.MapSqlParameterSourc
e;
import
org.springframework.jdbc.core.namedparam.SqlParameterSource;
import org.springframework.jdbc.core.simple.SimpleJdbcCall;

import com.durgasoft.beans.Employee;

public class EmployeeDaoImpl implements EmployeeDao {
private DataSource dataSource;
private SimpleJdbcCall jdbcCall;
public void setDataSource(DataSource dataSource) {
this.dataSource = dataSource;
jdbcCall = new SimpleJdbcCall(dataSource).withProcedureName("getSalary");

}
@Override
public void create(Employee emp) {
try {
JdbcTemplate jdbcTemplate = new JdbcTemplate(dataSource);
String sql= "insert into emp1 values("+emp.getEno()+",'"+emp.getEname()+"',"+emp.getEsal()+"
,'"+emp.getEaddr()+"')";
jdbcTemplate.update(sql);
}catch(Exception e) {
e.printStackTrace();
}

}

@Override
public Object getEmployeeSalary(int eno) {
SqlParameterSource in = new MapSqlParameterSource().addValue("no", eno);
Map<String, Object> map = jdbcCall.execute(in);
// System.out.println(map);
return map.get("SAL");
}
}
```

applicationContext.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xmlns:context="http://www.springframework.org/schema/context"
xsi:schemaLocation="
http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-
beans.xsd
http://www.springframework.org/schema/context

http://www.springframework.org/schema/context/spring-
context.xsd">
<bean id="empDao" class="com.durgasoft.dao.EmployeeDaoImpl">
<property name="dataSource" ref="dataSource"/>
</bean>
<bean id = "dataSource" class =
"org.springframework.jdbc.datasource.DriverManagerDataSource">
<property name = "driverClassName" value = "oracle.jdbc.OracleDriver"/>
<property name = "url" value = "jdbc:oracle:thin:@localhost:1521:xe"/>
<property name = "username" value = "system"/>
<property name = "password" value = "durga"/>
</bean>
</beans>
```

Test.java

```
package com.durgasoft.test;

import org.springframework.context.ApplicationContext;
import
org.springframework.context.support.ClassPathXmlApplicationCo
ntext;

import com.durgasoft.beans.Employee;
```

```
import com.durgasoft.dao.EmployeeDao;

public class Test {

public static void main(String[] args)throws
Exception {
ApplicationContext context = new ClassPathXmlApplicationContext("applicationContext.xml");
EmployeeDao dao = (EmployeeDao)context.getBean("empDao");

Employee emp1 = new Employee();
emp1.setEno(111);
emp1.setEname("AAA");
emp1.setEsal(5000);
emp1.setEaddr("Hyd");
dao.create(emp1);
Object salary1 = dao.getEmployeeSalary(emp1.getEno());
System.out.println(emp1.getEno()+"---->"+salary1);

Employee emp2 = new Employee();
emp2.setEno(222);
emp2.setEname("BBB");
emp2.setEsal(6000);
emp2.setEaddr("Hyd");
dao.create(emp2);
Object salary2 = dao.getEmployeeSalary(emp2.getEno());
System.out.println(emp2.getEno()+"---->"+salary2);

Employee emp3 = new Employee();
emp3.setEno(333);
emp3.setEname("CCC");
emp3.setEsal(7000);
emp3.setEaddr("Hyd");
dao.create(emp3);
Object salary3 = dao.getEmployeeSalary(emp3.getEno());
System.out.println(emp3.getEno()+"---->"+salary3);

}

}
```

Using CURSOR Types in Procedures:

Procedure from Spring JDBC Application by using SimpleJdbcCall :

If we want to use CURSOR types in Stored Procedures inorder to retrive multiple Records data then we have to use the following method on SimpleJdbcCall reference.

```
SimpleJdbcCall jdbcCall = new SimpleJdbcCall(dataSource)
jdbcCall = jdbcCall.withProcedureName("getAllEmployees");
jdbcCall =
jdbcCall.returningResultSet("emps",BeanPropertyRowMapper.newInstance(Employee.class));
```

After adding returningResultSet(--,--) method, if we access execute() method on SimpleJdbcCall then execute() method will execute procedure, it will get all the results from CURSOR type variable and stored all recordfs in the form of Employee objects in an ArrayList object with "emps"[CURSOR TYPE variable] key in a Map.

Example:

COPY this Procedure in Database

```
create or replace procedure getAllEmployees(emps OUT SYS_REFCURSOR)
AS
BEGIN
open emps for
select * from emp1;
END getAllEmployees;
/
```

Employee.java

```
package com.durgasoft.beans;

public class Employee {
private int eno;
private String ename;
private float esal;
private String eaddr;

public int getEno() {
return eno;
}
public void setEno(int eno) {
this.eno = eno;
}
```

```
public String getEname() {
return ename;
}
public void setEname(String ename) {
this.ename = ename;
}
public float getEsal() {
return esal;
}
public void setEsal(float esal) {
this.esal = esal;
}
public String getEaddr() {
return eaddr;
}
public void setEaddr(String eaddr) {
this.eaddr = eaddr;
}


}
```

EmployeeDao.java

```
package com.durgasoft.dao;

import java.util.Map;

public interface EmployeeDao {
public Map<String, Object> getAllEmployees();

}
```

EmployeeDaoImpl.java

```
package com.durgasoft.dao;

import java.util.Map;

import javax.sql.DataSource;

import org.springframework.jdbc.core.BeanPropertyRowMapper;
import org.springframework.jdbc.core.simple.SimpleJdbcCall;
import com.durgasoft.beans.Employee;

public class EmployeeDaoImpl implements EmployeeDao {
private SimpleJdbcCall jdbcCall;
private DataSource dataSource;
public void setDataSource(DataSource dataSource) {
this.dataSource = dataSource;
jdbcCall = new SimpleJdbcCall(dataSource).withProcedureName("getAllEmployees");
jdbcCall = jdbcCall.returningResultSet("emps",
BeanPropertyRowMapper.newInstance(Employee.class));
}
@Override
public Map<String, Object> getAllEmployees() {

Map<String, Object> map = jdbcCall.execute();

//System.out.println(map);
return map;
}
}
```

applicationContext.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:context="http://www.springframework.org/schema/context"
xsi:schemaLocation="
http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans.xsd
http://www.springframework.org/schema/context
http://www.springframework.org/s…/context/spring-context.xsd">
<bean id="empDao" class="com.durgasoft.dao.EmployeeDaoImpl">
<property name="dataSource" ref="dataSource"/>
</bean>
<bean id = "dataSource"
class = "org.springframework.jdbc.datasource.DriverManagerDataSource">
<property name = "driverClassName" value = "oracle.jdbc.OracleDriver"/>
<property name = "url" value = "jdbc:oracle:thin:@localhost:1521:xe"/>
<property name = "username" value = "system"/>
<property name = "password" value = "durga"/>
</bean>
```

```
</beans>
```

Test.java

```
package com.durgasoft.test;

import java.util.ArrayList;
import java.util.Map;

import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;

import com.durgasoft.beans.Employee;
import com.durgasoft.dao.EmployeeDao;

public class Test {

public static void main(String[] args)throws Exception {
ApplicationContext context = new ClassPathXmlApplicationContext("applicationContext.xml");
EmployeeDao dao = (EmployeeDao)context.getBean("empDao");
Map<String, Object> map = dao.getAllEmployees();
System.out.println(map);
ArrayList<Employee> list =(ArrayList<Employee>) map.get("emps");
System.out.println(list);

System.out.println("Employee Details");
System.out.println("ENO\tENAME\tESAL\tEADDR");
System.out.println("-----------------------------");
for(Employee e: list) {
System.out.println(e.getEno()+"\t"+e.getEname()+"\t"+e.getEsal()+"\t"+e.getEaddr());
}

}

}
```

Blob and Clob processing in Spring JDBC:

BLOB: It is a data type available at Databases to represent large volumes of binary data.
CLOB: It is a data type available at Database to represent large volumes of character data.

In Spring JDBC Applications, to process BLOB and CLOB Data, Spring JDBC has provided the following three interfaces mainly.

1.AbstractLobCreatingPreparedStatementCallback
--> It will be used to store Blob and Clob related data in Database.
  protected void setValues(PreparedStatement ps, LobCreator lobCreator) throws SQLException, DataAccessException

2.AbstractLobStreamingResultSetExtractor
--> It will be used to retrive BLOB and CLOB data from database.
  streamData(ResultSet rs)throws SQLException, IOException, DataAccessException

3.LobCreator
--> It contains the following methods to prepare Binary stream and character streams to send blob and clob data to database.
  setBlobAsBinaryStream()
  setClobAsCharacterStream()

4.LobHolder
--> It contains the following methods to get Binary stream and character stream to get blob and clob data.
  getBlobAsBinaryStream()
  getClobAsCharacterStream()

Example:

Employee.java

```
package com.durgasoft.beans;

import java.io.File;

public class Employee {
	private int eno;
	private String ename;
	private File emp_Image;
	private File emp_Resume;

	public int getEno() {
		return eno;
	}
	public void setEno(int eno) {
		this.eno = eno;
```

```
	}
	public String getEname() {
		return ename;
	}
	public void setEname(String ename) {
		this.ename = ename;
	}
	public File getEmp_Image() {
		return emp_Image;
	}
	public void setEmp_Image(File emp_Image) {
		this.emp_Image = emp_Image;
	}
	public File getEmp_Resume() {
		return emp_Resume;
	}
	public void setEmp_Resume(File emp_Resume) {
		this.emp_Resume = emp_Resume;
	}
}
```

EmployeeDao.java

```
package com.durgasoft.dao;

import com.durgasoft.beans.Employee;

public interface EmployeeDao {
	public void insertEmployee(Employee emp);
	public Employee readEmployee();
}
```

EmployeeDaoImpl.java

```
package com.durgasoft.dao;

import java.io.File;
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;
import java.sql.PreparedStatement;
```

```
import java.sql.ResultSet;
import java.sql.SQLException;

import javax.sql.DataSource;

import org.springframework.dao.DataAccessException;
import org.springframework.jdbc.core.JdbcTemplate;
import org.springframework.jdbc.core.support.AbstractLobCreatingPreparedStatementCallback;
import org.springframework.jdbc.core.support.AbstractLobStreamingResultSetExtractor;
import org.springframework.jdbc.support.lob.LobCreator;
import org.springframework.jdbc.support.lob.LobHandler;
import org.springframework.util.FileCopyUtils;

import com.durgasoft.beans.Employee;

public class EmployeeDaoImpl implements EmployeeDao {
	private LobHandler lobHolder;
	private DataSource dataSource;
	private JdbcTemplate jdbcTemplate;

	public void setDataSource(DataSource dataSource) {
		this.dataSource = dataSource;
		jdbcTemplate = new JdbcTemplate(dataSource);
	}
	public void setLobHolder(LobHandler lobHolder) {
		this.lobHolder = lobHolder;
	}
	public LobHandler getLobHolder() {
		return lobHolder;
	}
	@Override
	public void insertEmployee(Employee emp) {
		String sql_Query = "insert into emp10 values(?,?,?,?)";
		jdbcTemplate.execute(sql_Query, new
AbstractLobCreatingPreparedStatementCallback(lobHolder) {

			@Override
			protected void setValues(PreparedStatement ps, LobCreator lobCreator)
throws SQLException, DataAccessException {
				FileInputStream fis = null;
				FileReader fr = null;
				try {
					ps.setInt(1, emp.getEno());
```

```
                    ps.setString(2, emp.getEname());
                    fis = new FileInputStream(emp.getEmp_Image());
                    fr = new FileReader(emp.getEmp_Resume());
                    lobCreator.setBlobAsBinaryStream(ps, 3, fis,
(int)emp.getEmp_Image().length());
                    lobCreator.setClobAsCharacterStream(ps, 4, fr,
(int)emp.getEmp_Resume().length());
                }catch(IOException e) {
                    e.printStackTrace();
                }
            }
        } );

    }

    @Override
    public Employee readEmployee() {
        Employee emp = new Employee();

            String sql = "select * from emp10";
            jdbcTemplate.query(sql, new
AbstractLobStreamingResultSetExtractor<Object>() {
                @Override
                protected void streamData(ResultSet rs) throws SQLException,
IOException, DataAccessException {
                    emp.setEno(rs.getInt(1));
                    emp.setEname(rs.getString(2));

                    File file1 = new File("E:/spring/my_image.jpg");
                    FileOutputStream fos = new FileOutputStream(file1);
                FileCopyUtils.copy(lobHolder.getBlobAsBinaryStream(rs, 3), fos);
                    emp.setEmp_Image(file1);

                    File file2 = new File("E:/spring/my_resume.docx");
                    FileWriter fw = new FileWriter(file2);
                FileCopyUtils.copy(lobHolder.getClobAsCharacterStream(rs, 4), fw);
                    emp.setEmp_Resume(file2);

                }
            });

        return emp;
    }
```

```
}
```

applicationContext.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xmlns:context="http://www.springframework.org/schema/context"
   xsi:schemaLocation="
      http://www.springframework.org/schema/beans
      http://www.springframework.org/schema/beans/spring-beans.xsd
      http://www.springframework.org/schema/context
      http://www.springframework.org/schema/context/spring-context.xsd">

   <bean id="empDao" class="com.durgasoft.dao.EmployeeDaoImpl">
      <property name="dataSource" ref="dataSource"/>
      <property name="lobHolder" ref="lobHolder"/>
   </bean>

   <bean id="dataSource"
class="org.springframework.jdbc.datasource.DriverManagerDataSource">
      <property name="driverClassName" value="${jdbc.driverClassName}"/>
      <property name="url" value="${jdbc.url}"/>
      <property name="username" value="${jdbc.username}"/>
      <property name="password" value="${jdbc.password}"/>
   </bean>
   <bean id="lobHolder" class="org.springframework.jdbc.support.lob.DefaultLobHandler">
   </bean>
   <context:property-placeholder location="jdbc.properties"/>

</beans>
```

jdbc.properties

```
jdbc.driverClassName = oracle.jdbc.OracleDriver
jdbc.url = jdbc:oracle:thin:@localhost:1521:xe
jdbc.username = system
jdbc.password = durga
```

Test.java

```
package com.durgasoft.test;
```

```
import java.io.File;

import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;

import com.durgasoft.beans.Employee;
import com.durgasoft.dao.EmployeeDao;

public class Test {

	public static void main(String[] args)throws Exception {
		ApplicationContext context = new
ClassPathXmlApplicationContext("applicationContext.xml");
		EmployeeDao dao = (EmployeeDao)context.getBean("empDao");
		File file1 = new File("E:/spring/nag.jpg");
		File file2 = new File("E:/spring/nag_resume.docx");
		Employee emp1 = new Employee();
		emp1.setEno(111);
		emp1.setEname("Nag");
		emp1.setEmp_Image(file1);
		emp1.setEmp_Resume(file2);
		dao.insertEmployee(emp1);
		System.out.println("Employee Inserted Successfully");

		Employee emp2 = dao.readEmployee();
		System.out.println("Employee Retrived Successfully");
		System.out.println("Employee Details");
		System.out.println("---------------------");
		System.out.println("Employee Number  :"+emp2.getEno());
		System.out.println("Employee Name    :"+emp2.getEname());
		System.out.println("Employee Image   :"+emp2.getEmp_Image().getAbsolutePath());
		System.out.println("Employee Resume
:"+emp2.getEmp_Resume().getAbsolutePath());
	}
}
```

Spring JDBC Connection Pooling Mechanism:

In Database related applications, if we want to perform database operations first we have to creater Connection object then we have to close connection object when the database operations are completed.

IN Database related applications, every time creating Connection object and every time destroying Connection object may reduce application performance, because, Creating Connection object and destroying Connection object are two expensive processes, which may reduce application performance.

To overcome the above problem we have to use Connection Pooling in applications.In Connection pooling we will create a set of Connection object in the form of a pool at the application startup time and we will reuse that COnnection objects while executing applications , when database operations are completed then we will send back that connection objects to Pool object with out destroying that connection objects.

In SPring JDBC applications there are three approaches to provide connection pooling.

1.Default Connection Pooling Mech.
2.Third Party Connection Pooling Mechanisms
3.Application Servers provided Connection Pooling Mechanism

1.Default Connection Pooling Mech.

In SPring Framework, Default Connection pooling mechanism is existed in the form of org.springframework.jdbc.datasource.DriverManagerDataSource, it is usefull upto testging only, it is usefull for production environment of the application.

If we want to use default Connection Pooling mechanism in SPring JDBC application then we have to configure org.springframework.jdbc.datasource.DriverManagerDataSource in beans configuration file with the following properties .

1.driverClassName
2.url
3.username
4.password

EX:

```
<bean id="dataSource" class="org.springframework.jdbc.datasource.DriverManagerDataSource">
        <property name="driverClassName" value="oracle.jdbc.OracleDriver"/>
        <property name="url" value="jdbc:oracle:thin:@localhost:1521:xe"/>
        <property name="username" value="system"/>
        <property name="password" value="durga"/>
</bean>
```

2.Third Party Connection Pooling Mechanisms

In Spring JDBC applications we are able to use the following third party connection pooling mechanisms
1.Apache DBCP
2.C3P0
3.Proxool

1.Apache DBCP:

To use Apcahes DBCP connection pooling mechanism then we have to configure org.apache.commons.dbcp2.BasicDataSource class with the following properties in spring beans configuration file.

1.driverClassName
2.url
3.username
4.password
5.initialSize:It will take Initial pool size.
6.maxTotal: It will allow the specified no of max connections.

EX:

```
<bean id="dataSource" class="org.apache.commons.dbcp2.BasicDataSource">
   <property name="driverClassName" value="oracle.jdbc.OracleDriver" />
   <property name="url" value="jdbc:oracle:thin:@localhost:1521:xe" />
   <property name="username" value="root" />
   <property name="password" value="root" />
   <property name="initialSize" value="20" />
   <property name="maxTotal" value="30" />
</bean>
```

Note: To use this mechanism in Spring JDBC Applications then we have to add the following two jar files to Library.
1.commons-dbcp2-2.2.0.jar
2.commons-pool2-2.5.0.jar

2.C3P0

To use C3P0 connection pooling mechanism then we have to configure com.mchange.v2.c3p0.ComboPooledDataSource class with the following properties in spring beans configuration file.

1.driverClass
2.jdbcUrl
3.user
4.password
5.minPoolSize:It will take Initial pool size.
6.maxPoolSize: It will allow the specified no of max connections.
7.maxStatements: Max statements it allows.
8.testConnectionOnCheckOut:true/false for Checking Connection before use.

EX:

```
<bean id="dataSource" class="com.mchange.v2.c3p0.ComboPooledDataSource">
        <property name="driverClass" value="oracle.jdbc.OracleDriver" />
        <property name="jdbcUrl" value="jdbc:oracle:thin:@localhost:1521:xe" />
        <property name="user" value="system" />
        <property name="password" value="durga" />
        <property name="maxPoolSize" value="30" />
        <property name="minPoolSize" value="10" />
        <property name="maxStatements" value="100" />
        <property name="testConnectionOnCheckout" value="true" />
</bean>
```

Note: To use this mechanism in Spring JDBC Applications then we have to add the following two jar files to Library.
1.c3p0-0.9.5.2.jar
2.mchange-commons-java-0.2.11.jar

3.Proxool:

To use Proxool connection pooling mechanism then we have to configure org.logicalcobwebs.proxool.ProxoolDataSource class with the following properties in spring beans configuration file.

1.driver
2.driverUrl
3.user

4.password
5.minimumConnectionCount:It will take Initial pool size.
6.maximumConnectionCount: It will allow the specified no of max connections.

EX:

```
<bean id="dataSource" class="org.logicalcobwebs.proxool.ProxoolDataSource">
	<property name="driver" value="oracle.jdbc.OracleDriver" />
	<property name="driverUrl" value="jdbc:oracle:thin:@localhost:1521:xe" />
	<property name="user" value="system" />
	<property name="password" value="durga" />
	<property name="maximumConnectionCount" value="30" />
	<property name="minimumConnectionCount" value="10" />
</bean>
```

Note: To use this mechanism in Spring JDBC Applications then we have to add the following two jar files to Library.
1.proxool-0.9.1.jar
2.proxool-cglib.jar

Example:

Employee.java

```
package com.durgasoft.beans;

public class Employee {
	private int eno;
	private String ename;
	private float esal;
	private String eaddr;

	public int getEno() {
		return eno;
	}
	public void setEno(int eno) {
		this.eno = eno;
	}
	public String getEname() {
		return ename;
	}
	public void setEname(String ename) {
```

```
                this.ename = ename;
        }
        public void setEsal(float esal) {
                this.esal = esal;
        }
        public float getEsal() {
                return esal;
        }
        public void setEaddr(String eaddr) {
                this.eaddr = eaddr;
        }
        public String getEaddr() {
                return eaddr;
        }
}
```

EmployeeDao.java

```
package com.durgasoft.dao;

import com.durgasoft.beans.Employee;

public interface EmployeeDao {
        public void insertEmployee(Employee emp);

}
```

EmployeeDaoImpl.java

```
package com.durgasoft.dao;


import javax.sql.DataSource;

import org.springframework.jdbc.core.JdbcTemplate;

import com.durgasoft.beans.Employee;

public class EmployeeDaoImpl implements EmployeeDao {
        private DataSource dataSource;
        private JdbcTemplate jdbcTemplate;

        public void setDataSource(DataSource dataSource) {
```

```
                this.dataSource = dataSource;
                jdbcTemplate = new JdbcTemplate(dataSource);
        }
        @Override
        public void insertEmployee(Employee emp) {
                String sql_Query = "insert into emp1
values("+emp.getEno()+",'"+emp.getEname()+"',"+emp.getEsal()+",'"+emp.getEaddr()+"')";
                jdbcTemplate.execute(sql_Query);

        }
}
```

applicationContext.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xmlns:context="http://www.springframework.org/schema/context"
   xsi:schemaLocation="
      http://www.springframework.org/schema/beans
      http://www.springframework.org/schema/beans/spring-beans.xsd
      http://www.springframework.org/schema/context
      http://www.springframework.org/schema/context/spring-context.xsd">

   <bean id="empDao" class="com.durgasoft.dao.EmployeeDaoImpl">
      <property name="dataSource" ref="dataSource"/>
   </bean>
<!--  Default Connection Pooling Mechanism
<bean id="dataSource" class="org.springframework.jdbc.datasource.DriverManagerDataSource">
      <property name="driverClassName" value="oracle.jdbc.OracleDriver"/>
      <property name="url" value="jdbc:oracle:thin:@localhost:1521:xe"/>
      <property name="username" value="system"/>
      <property name="password" value="durga"/>
</bean>
-->
<!--  DBCP Connection Pooling Mechanism Properties
  <bean id="dataSource" class="org.apache.commons.dbcp2.BasicDataSource">
                <property name="driverClassName" value="oracle.jdbc.OracleDriver" />
                <property name="url" value="jdbc:oracle:thin:@localhost:1521:xe" />
                <property name="username" value="system" />
                <property name="password" value="durga" />
                <property name="initialSize" value="20" />
                <property name="maxTotal" value="30" />
```

```
        </bean>
-->
<!--  C3P0 Connection Pooling Mechanism Properties
<bean id="dataSource" class="com.mchange.v2.c3p0.ComboPooledDataSource">
        <property name="driverClass" value="oracle.jdbc.OracleDriver" />
        <property name="jdbcUrl" value="jdbc:oracle:thin:@localhost:1521:xe" />
        <property name="user" value="system" />
        <property name="password" value="durga" />
        <property name="maxPoolSize" value="30" />
        <property name="minPoolSize" value="10" />
        <property name="maxStatements" value="100" />
        <property name="testConnectionOnCheckout" value="true" />
</bean>
-->
<!-- Proxool Connection Pooling Mechanism Properties -->
<bean id="dataSource" class="org.logicalcobwebs.proxool.ProxoolDataSource">
        <property name="driver" value="oracle.jdbc.OracleDriver" />
        <property name="driverUrl" value="jdbc:oracle:thin:@localhost:1521:xe" />
        <property name="user" value="system" />
        <property name="password" value="durga" />
        <property name="maximumConnectionCount" value="30" />
        <property name="minimumConnectionCount" value="10" />
</bean>
</beans>
```

Test.java

```
package com.durgasoft.test;
import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;
import com.durgasoft.beans.Employee;
import com.durgasoft.dao.EmployeeDao;
public class Test {

        public static void main(String[] args)throws Exception {
                ApplicationContext context = new
ClassPathXmlApplicationContext("applicationContext.xml");
                EmployeeDao dao = (EmployeeDao)context.getBean("empDao");
                Employee emp = new Employee();
                emp.setEno(111);
                emp.setEname("Nag");
                emp.setEsal(5000);
                emp.setEaddr("Hyd");
```

```
			dao.insertEmployee(emp);
			System.out.println("Employee Inserted Successfully");

		}
}
```

3.Application Servers provided Connection Pooling Mechanism throw JNDI:

JNDI[Java Naming And Directory Interface]: JNDI is a Middleware Service or an abstraction provided by SUN Micreosystems as part of J2EE and which is implemented by all the Application Servers vendors like Weblogic, JBOSS, Glassfish,.....

JNDI is existed inside the application Servers to provide any resource with Global Scope, that is, JNDI will share any resource like "DataSource" to all the applications which are running in the present application server.

In general, almost all the Application Servers are having their own Connection Pooling mechanisms, if we want to use Application Servers provided Connection pooling mechanisms we have to use the following steps.

1)Install Application Server.
2)COnfigure Connection Pooling and Datasource in JNDI provided by Application Servers.
3)Add the required new JARs to Library.
4)Provide JNDI Setups in beans configuration File.

1) Install Application Server[Weblogic Server]

1.Download fmw_12.2.1.3.0_wls_quick.jar from internet[oracle.com]
2.Open command prompt in Administrator mode.
3.Set JAVA8 or JAVA7 in path.
   set path=C:\Java\jdk1.8.0_144\bin;
4.Goto setup file loaction and execute JAR file with the following command.
   F:\softwares\servers\weblogic>java -jar fmw_12.2.1.3.0_wls_quick.jar
5.Click on "Next" button.
6.Click on "Next" Button
7.Specify Home directory "Oracle"
8.Click on "Next" button.
9.Click on "Next" button.
10.Click on "Next" button
11.Click on "Install" button.
12.Click on "Next" button.
13.Click on "Finish" button.
14.Provide domain name "durga_domain".

15.Click on "Next" button.
16.Click on "Next" button.
17.Provide user name and password.
   user name: weblogic
   password: weblogic_weblogic
   confirm password: weblogic_weblogic
18.Click on "Next" button.
19.Click on "Next" button.
20.Select "Adminstration Server"
21.CLick on "Next" button.
22.Click on "Next" button.
23.Click on "Create" button.
24.Click on "Next" button.
25.Click on "Finish" button.

2)COnfigure Connection Pooling and Datasource in JNDI provided by Application Servers:

1.Goto durga_domain location
   C:\Oracle\user_projects\domains\durga_domain
2.double click on "startWeblogic" batch file.
3.Open Browser and provide the following url to open Administration console.
   http://localhost:7001/console
4.Provide domain user name and password.
   user name: weblogic
   password: weblogic_weblogic
5.Click on "Login" button.
6.Go for Domain Structer and select "Services".
7.Select "DataSource".
8.Click on "New" button.
9.Select "Generic datasource".
10.Provide the following details.
   Name: durgads
   Scope: GLOBAL
   JNDI Name: durgajndi
   Database Type: Oracle
11.Click on "Next" button.
12.Click on "Next" button.
13.Click on "Next" button.
14.Provide the following details.
   Database Name: xe
   Host Name: localhost
   Port : 1521
   Database User Name: system

   password: durga
   confirm password: durga
15.Click on "Next" button.
16.Click on "Next" button.
17.Select "admin server".
18.Click on "Finish" button.

3)Add the required new JARs to Library:

To use Weblogic Server provided Connection Pooling mechanism in Spring JDBC Application then we have to use the following jars along with the regular jars.
1)weblogic.jar
2)spring-jdbc-4.0.4.RELEASE.jar
3)spring-tx-4.0.4.RELEASE.jar

4)Provide JNDI Setups in beans configure:

To use Weblogic Server provided Connection Pooling mechanism in Spring JDBC Application then we have to provide the following DataSource configuration in beans configuration file.\

```
<bean id="dataSource" class="org.springframework.jndi.JndiObjectFactoryBean">
        <property name="jndiName" value="durgajndi"/>
        <property name="jndiEnvironment">
      <props>
         <prop key="java.naming.factory.initial">weblogic.jndi.WLInitialContextFactory</prop>
         <prop key="java.naming.provider.url">t3://localhost:7001</prop>
      </props>
        </property>
</bean>

<bean id="empDao" class="com.durgasoft.dao.EmployeeDaoImpl">
      <property name="dataSource" ref="dataSource"/>
</bean>
```

Example:

Employee.java

```
package com.durgasoft.beans;


public class Employee {
        private int eno;
```

```
	private String ename;
	private float esal;
	private String eaddr;

	public int getEno() {
		return eno;
	}
	public void setEno(int eno) {
		this.eno = eno;
	}
	public String getEname() {
		return ename;
	}
	public void setEname(String ename) {
		this.ename = ename;
	}
	public void setEsal(float esal) {
		this.esal = esal;
	}
	public float getEsal() {
		return esal;
	}
	public void setEaddr(String eaddr) {
		this.eaddr = eaddr;
	}
	public String getEaddr() {
		return eaddr;
	}
}
```

EmployeeDao.java

```
package com.durgasoft.dao;

import com.durgasoft.beans.Employee;

public interface EmployeeDao {
	public void insertEmployee(Employee emp);

}
```

EmployeeDaoImpl.java

```
package com.durgasoft.dao;


import javax.sql.DataSource;

import org.springframework.jdbc.core.JdbcTemplate;

import com.durgasoft.beans.Employee;

public class EmployeeDaoImpl implements EmployeeDao {
	private DataSource dataSource;
	private JdbcTemplate jdbcTemplate;

	public void setDataSource(DataSource dataSource) {
		this.dataSource = dataSource;
		jdbcTemplate = new JdbcTemplate(dataSource);
	}
	@Override
	public void insertEmployee(Employee emp) {
		String sql_Query = "insert into emp1 
values("+emp.getEno()+",'"+emp.getEname()+"',"+emp.getEsal()+",'"+emp.getEaddr()+"')";
		jdbcTemplate.execute(sql_Query);

	}
}
```

applicationContext.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:context="http://www.springframework.org/schema/context"
    xsi:schemaLocation="
        http://www.springframework.org/schema/beans
        http://www.springframework.org/schema/beans/spring-beans.xsd
        http://www.springframework.org/schema/context
        http://www.springframework.org/schema/context/spring-context.xsd">

   <bean id="dataSource" class="org.springframework.jndi.JndiObjectFactoryBean">
        <property name="jndiName" value="durgajndi"/>
        <property name="jndiEnvironment">
```

```
        <props>
            <prop key="java.naming.factory.initial">weblogic.jndi.WLInitialContextFactory</prop>
            <prop key="java.naming.provider.url">t3://localhost:7001</prop>
        </props>
          </property>
   </bean>

   <bean id="empDao" class="com.durgasoft.dao.EmployeeDaoImpl">
        <property name="dataSource" ref="dataSource"/>
   </bean>

</beans>
```

Test.java

```
package com.durgasoft.test;
import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;
import org.springframework.remoting.rmi.JndiRmiProxyFactoryBean;

import com.durgasoft.beans.Employee;
import com.durgasoft.dao.EmployeeDao;

public class Test {

        public static void main(String[] args)throws Exception {

                ApplicationContext context = new
ClassPathXmlApplicationContext("applicationContext.xml");
                EmployeeDao dao = (EmployeeDao)context.getBean("empDao");
                Employee emp = new Employee();
                emp.setEno(111);
                emp.setEname("AAA");
                emp.setEsal(5000);
                emp.setEaddr("Hyd");
                dao.insertEmployee(emp);
                System.out.println("Employee Inserted Successfully");
        }
}
```