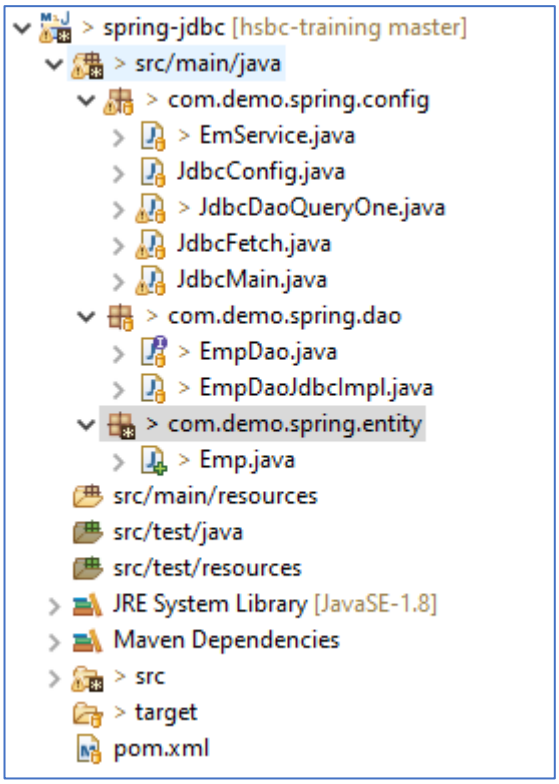
Implement a DAO with Spring JDBC

Aim: the base infrastructure code has been provided below. Implement the methods which are kept blank

1. The Project Structure:



2. Create a Maven project with the following dependencies

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
	xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
	xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
	<modelVersion>4.0.0</modelVersion>
	<groupId>com.demo.spring</groupId>
	<artifactId>spring-jdbc</artifactId>
	<version>1.0</version>
	<name>spring-jdbc</name>
	<properties>
		<maven.compiler.source>11</maven.compiler.source>
		<maven.compiler.target>11</maven.compiler.target>
	</properties>
	<dependencies>

		<dependency>
			<groupId>org.springframework</groupId>
			<artifactId>spring-context</artifactId>
			<version>5.2.15.RELEASE</version>
		</dependency>

		<dependency>
			<groupId>org.springframework</groupId>
			<artifactId>spring-jdbc</artifactId>
			<version>5.2.15.RELEASE</version>
		</dependency>
```

```
		<dependency>
			<groupId>org.springframework</groupId>
			<artifactId>spring-tx</artifactId>
			<version>5.2.15.RELEASE</version>
		</dependency>

		<dependency>
			<groupId>org.mariadb.jdbc</groupId>
			<artifactId>mariadb-java-client</artifactId>
			<version>2.7.2</version>
		</dependency>
	</dependencies>
	</project>
```

3. The Configuration Bean

```
package com.demo.spring.config;

import javax.sql.DataSource;

import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.ComponentScan;
import org.springframework.context.annotation.Configuration;
import org.springframework.jdbc.core.JdbcTemplate;
import org.springframework.jdbc.datasource.DriverManagerDataSource;

@Configuration
@ComponentScan(basePackages = "com.demo.spring")
public class JdbcConfig {

	@Bean
	public DriverManagerDataSource dataSource() {
		DriverManagerDataSource ds= new DriverManagerDataSource();
		ds.setDriverClassName("org.apache.derby.jdbc.ClientDriver");
		ds.setUrl("jdbc:derby://localhost:1527/demodb");
		//ds.setUsername("");
		//ds.setPassword("");
		return ds;
	}

	@Bean
	public JdbcTemplate jdbcTemplate(DataSource ds) {
		return new JdbcTemplate(ds);
	}
}
```

4. The entity class

```
package com.demo.spring.entity;

public class Emp {
	private int empId;
	private String name;
	private String city;
	private double salary;

	public Emp() {

	}

	public Emp(int empId, String name, String city, double salary) {
		this.empId = empId;
		this.name = name;
		this.city = city;
		this.salary = salary;
	}



	//generate setter and getter for all the variables/properties

}
```

5. The Dao Interface:

```
	package com.demo.spring.dao;

	import java.util.List;
	import com.demo.spring.entity.Emp;

	public interface EmpDao {

		public String save(Emp e);
		public Emp findById(int id);
		public List<Emp> getAll();
		public String update(Emp e);
		public String delete(int id);

	}
```

6. The Implementation class:

```
package com.demo.spring.dao;

import java.sql.ResultSet;
import java.sql.SQLException;
import java.util.List;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.jdbc.core.JdbcTemplate;
import org.springframework.jdbc.core.RowMapper;
import org.springframework.stereotype.Repository;

import com.demo.spring.entity.Emp;

@Repository
```

```
public class EmpDaoJdbcImpl implements EmpDao {

	@Autowired
	private JdbcTemplate jt;

	@Override
	public String save(Emp e) {
		// TODO Auto-generated method stub
		return null;
	}

	@Override
	public Emp findById(int id) {
		Emp emp = jt.queryForObject("select * from EMP where empno=" + id,
	new RowMapper<Emp>() {

			@Override
			public Emp mapRow(ResultSet rs, int rowNum) throws SQLException
{

				return new Emp(rs.getInt(1),
						rs.getString(2),
						rs.getString(3),
						rs.getDouble(4));
			}

		});
		return emp;
	}


	@Override
	public List<Emp> getAll() {

		List<Emp> empList = jt.query("select * from EMP", new RowMapper<Emp>()
	{

		@Override
		public Emp mapRow(ResultSet rs, int rowNum) throws SQLException {

				return new Emp(rs.getInt(1),
						rs.getString(2),
						rs.getString(3),
						rs.getDouble(4));
			}

		});
		return empList;
	}

	@Override
	public String update(Emp e) {

		return null;
	}

	@Override
	public String delete(int id) {

		return null;
	}

}
```

7. The Service class

```
package com.demo.spring.service;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;

import com.demo.spring.dao.EmpDao;
import com.demo.spring.entity.Emp;

@Service
public class EmService {

	@Autowired
	EmpDao dao;

	public String registerEmp(int id, String name, String city, double salary) {
		return dao.save(new Emp(id, name, city, salary));
	}
}
```

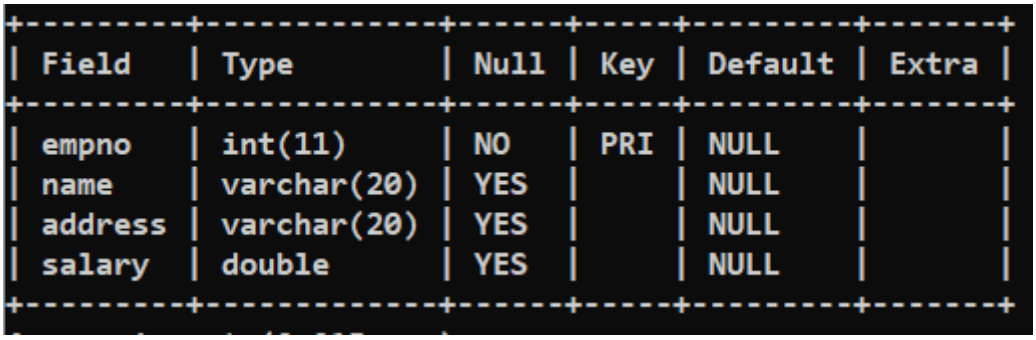
8. The Client Applications:

```
package com.demo.spring.clients;

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

import org.springframework.context.ApplicationContext;
import org.springframework.context.annotation.AnnotationConfigApplicationContext;
import org.springframework.jdbc.core.JdbcTemplate;
import org.springframework.jdbc.core.PreparedStatementCreator;
import org.springframework.jdbc.core.RowMapper;

import com.demo.spring.dao.EmpDao;
import com.demo.spring.entity.Emp;

public class JdbcDaoQueryOne {

	public static void main(String[] args) {

		ApplicationContext ctx = new
		AnnotationConfigApplicationContext(JdbcConfig.class);

		EmpDao dao = ctx.getBean(EmpDao.class);
		Emp e=dao.findById(103);
		System.out.println(e.getEmpId()+" "+e.getName());


	}

}
```

9. Database Schema:

```
CREATE DATABASE springdb;
USE DATABASE springdb;
DROP TABLE IF EXISTS `emp`;
CREATE TABLE `emp` (
  `empno` int(11) NOT NULL,
  `name` varchar(20) DEFAULT NULL,
  `address` varchar(20) DEFAULT NULL,
  `salary` double DEFAULT NULL,
  PRIMARY KEY (`empno`)
)

INSERT INTO `emp` VALUES
(100,'Amitabh','Mumbai',20000),(101,'Shekhar','Hyderabad',30000),(102,'Rekha','Mum
bai',23000),(103,'Kalluram','Delhi',60000),(104,'Ajay','Bangalore',80000);
```

10. The Table Structure:



Field	Type	Null	Key	Default	Extra
empno	int(11)	NO	PRI	NULL	
name	varchar(20)	YES		NULL	
address	varchar(20)	YES		NULL	
salary	double	YES		NULL	

Conclusion:

You have created a DAO implented with Spring JDBC and tested with Application Code.