Spring 5.2
Data Access

Author
Srinivas Dande

Spring Data Access

Spring Connection management

- Spring supports Various DataSources to get the Connections.
  1) DriverManagerDataSource connections
  2) DBCP DataSource Connections
  3) DBCP2 DataSource Connections
  4) Tomcat DataSource Connections
  5) Hikari DataSource Connections
  6) C3P0 DataSource Connections

1. DriverManagerDataSource connections

- When you want to use DriverManagerDataSource connections, you need to configure the DriverManagerDataSource class in Spring Configuration class:

```
@Bean
public DataSource mysqlDS() {
DriverManagerDataSource ds = new DriverManagerDataSource();
ds.setDriverClassName("com.mysql.jdbc.Driver");
ds.setUrl("jdbc:mysql://localhost:3306/myspringdb");
ds.setUsername("root");
ds.setPassword("srinivas");
return ds;
}
```

2. DBCP connections:

- When you want to use DriverManagerDataSource connections, you need to configure the DriverManagerDataSource class in Spring Configuration class:

```
@Bean
public DataSource dataSource() {
BasicDataSource ds = new BasicDataSource();
ds.setDriverClassName("com.mysql.jdbc.Driver");
ds.setUrl("jdbc:mysql://localhost:3306/myspringdb");
ds.setUsername("root");
ds.setPassword("srinivas");
ds.setInitialSize(10);
ds.setMaxActive(15);
return ds;
}
```

Lab56: Working Steps:

1. Create the Java Project : Lab56

2. Add 21 Spring Jars to Project Build Path.

3. Add mysql-connector-java-8.0.15 to Project Build Path.

4. Setup the Database

```
create database myspringdb;
use myspringdb;
create table mycustomers(
cid int primary key,
cname char(15),
email char(15),
phone long,
city char(15)
);
```

5. Write Spring Configuration Class and Enable with **@ComponentScan**

6. Choose the DataSource and configure the DataSource Bean in Spring Configuration Class.

```
@Bean
public DriverManagerDataSource getDS() {
DriverManagerDataSource ds=new DriverManagerDataSource();

ds.setDriverClassName("com.mysql.jdbc.Driver");
ds.setUrl("jdbc:mysql://localhost:3306/myspringdb");
ds.setUsername("root");
ds.setPassword("srinivas");

return ds;
}
```

7. Write the Customer pojo based o table.

```
public class Customer {
private int cid;
private String cname;
private String email;
private long phone;
private String city;
...
}
```

8. Write the CustomerDAO interface.

```
public interface CustomerDAO {
    public void addCustomer(Customer cust);
    public Customer getCustomerByCid(int cid);
    public List<Customer> getAllCustomers();
}
```

9. Write the Implementation class for CustomerDAO interface and Override all the methods.

10) Mark the CustomerDAOImpl class **@Repository**

```
@Repository("mycustDAO")
public class CustomerDAOImpl implements CustomerDAO {
...
}
```

11)Write the Client Code :Lab56.java

Lab56: Files required

1. Lab56.java	**2. Customer.java**
3. CustomerDAO.java	**4. CustomerDAOImpl.java**
5. JLCAppConfig.java	

1. Lab56.java

```
package com.coursecube.spring;

import org.springframework.context.ApplicationContext;
import org.springframework.context.annotation.AnnotationConfigApplicationContext;
/*
* @Author : Srinivas Dande
* @Company : CourseCube
* @Website : www.coursecube.com
* */
public class Lab56 {
public static void main(String[] args) {
ApplicationContext ctx=new AnnotationConfigApplicationContext(JLCAppConfig.class);
System.out.println("Spring Container is Ready");

CustomerDAO custDAO=ctx.getBean("mycustDAO",CustomerDAO.class);

Customer cust1=new Customer(103,"SD","SD@jlc",123,"Blore");
custDAO.addCustomer(cust1);

Customer cust2=custDAO.getCustomerByCid(201);
System.out.println(cust2);

System.out.println("Done");
}
}
```

2. Customer.java

```
package com.coursecube.spring;
/*
* @Author : Srinivas Dande
* @Company : CourseCube
* @Website : www.coursecube.com
* */
```

```
public class Customer {
	private int cid;
	private String cname;
	private String email;
	private long phone;
	private String city;

	public Customer() {
	}
	public Customer(String cname, String email, long phone, String city) {
		this.cname = cname;
		this.email = email;
		this.phone = phone;
		this.city = city;
	}
	public Customer(int cid, String cname, String email, long phone, String city) {
		super();
		this.cid = cid;
		this.cname = cname;
		this.email = email;
		this.phone = phone;
		this.city = city;
	}

	//Setters and Getters

	@Override
	public String toString() {
		return cid + ", " + cname + ", " + email + ", " + phone + ", " + city;
	}
}
```

3. CustomerDAO.java

```
package com.coursecube.spring;

/*
* @Author : Srinivas Dande
* @Company : CourseCube
* @Website : www.coursecube.com
* */

public interface CustomerDAO {
public void addCustomer(Customer cust);
public Customer getCustomerByCid(int cid);
}
```

4. CustomerDAOImpl.java

```
package com.coursecube.spring;

import java.sql.*;
import javax.sql.DataSource;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Repository;
/*
* @Author : Srinivas Dande
* @Company : CourseCube
* @Website : www.coursecube.com
* */
@Repository("mycustDAO")
public class CustomerDAOImpl implements CustomerDAO {

	@Autowired
	DataSource ds;

	@Override
	public void addCustomer(Customer cust) {
		Connection con=null;
		PreparedStatement ps=null;
		try {
			con=ds.getConnection();
			String SQL="insert into mycustomers values(?,?,?,?,?)";
			ps=con.prepareStatement(SQL);

			ps.setInt(1, cust.getCid());
			ps.setString(2, cust.getCname());
			ps.setString(3, cust.getEmail());
			ps.setLong(4, cust.getPhone());
			ps.setString(5, cust.getCity());

			int x=ps.executeUpdate();
			System.out.println(x);
		}catch(Exception ex) {
			System.out.println(ex);
		}finally {
			try {
			if(ps!=null) {
				ps.close();
			}
			if(con!=null) {
				con.close();
			}
			}catch(Exception ex1) { }
		}
	}
```

```
	@Override
	public Customer getCustomerByCid(int cid) {
		Customer cust=null;
		Connection con=null;
		PreparedStatement ps=null;
		ResultSet rs=null;
		try {
			con=ds.getConnection();
			String SQL="select * from mycustomers where cid=?";
			ps=con.prepareStatement(SQL);

			ps.setInt(1, cid);

			rs=ps.executeQuery();
			if(rs.next()) {
				cust=new Customer();
				cust.setCid(rs.getInt(1));
				cust.setCname(rs.getString(2));
				cust.setEmail(rs.getString(3));
				cust.setPhone(rs.getLong(4));
				cust.setCity(rs.getString(5));
			}
		}catch(Exception ex) {
			System.out.println(ex);
		}finally {
			try {
			 if(rs!=null) {
				rs.close();
			}
			if(ps!=null) {
				ps.close();
			}
			if(con!=null) {
				con.close();
			}
			}catch(Exception ex1) {

			}
		}
		return cust;
	}

}
```

5. JLCAppConfig.java

```
package com.coursecube.spring;

import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.ComponentScan;
import org.springframework.context.annotation.Configuration;
import org.springframework.jdbc.datasource.DriverManagerDataSource;
/*
* @Author : Srinivas Dande
* @Company : CourseCube
* @Website : www.coursecube.com
* */
@Configuration
@ComponentScan(basePackages = {"com.coursecube.spring"})
public class JLCAppConfig {


	@Bean
	public DriverManagerDataSource getDS() {
		DriverManagerDataSource ds=new DriverManagerDataSource();

		//ds.setDriverClassName("com.mysql.jdbc.Driver");
		ds.setUrl("jdbc:mysql://localhost:3306/myspringdb");
		ds.setUsername("root");
		ds.setPassword("srinivas");

		return ds;
	}


}
```

Spring DAO Support

- **Spring Supports DAO's in 2 ways:**
  1) **Provides new Exception Hierarchy with DataAccessException as root exception for DAO Exceptions. DataAccessException is a runtime Exception so all the Spring DAO Exceptions are runtime Exceptions.**

     org.springframework.dao.DataAccessException

     java.lang.Object
     └ java.lang.Throwable
       └ java.lang.Exception
         └ java.lang.RuntimeException
           └ org.springframework.core.NestedRuntimeException
             └ org.springframework.dao.DataAccessException

  2) **Provides Various Exceptions classes for various problems coming with Database interaction.**

     a) abstract class DataAccessException extends NestedRuntimeException
     b) class CleanupFailureDataAccessException extends DataAccessException
     c) class DataAccessResourceFailureException extends DataAccessException
     d) class DataIntegrityViolationException extends DataAccessException
     e) class DataRetrievalFailureException extends DataAccessException
     f) class DeadlockLoserDataAccessException extends PessimisticLockingFailureException
     g) class IncorrectUpdateSemanticsDataAccessException extends InvalidDataAccessResourceUsageException
     h) class InvalidDataAccessApiUsageException extends DataAccessException
     i) class InvalidDataAccessResourceUsageException extends DataAccessException
     j) class PessimisticLockingFailureException extends ConcurrencyFailureException
     k) class TypeMismatchDataAccessException extends InvalidDataAccessResourceUsageException
     l) abstract class UncategorizedDataAccessException extends DataAccessException

Spring Data access with JDBC

- When you want to perform any persistent operation then you need to write the JDBC Code with the following Steps:

```
try {                          // 1
        Take connection        // 2
        Create statement       // 3
        Prepare SQL            // 4
        Submit the SQL         // 5
        Process results        // 6
}catch (){ }
finally {
        Cleanup                // 7
}
```

You can see the following problems with above code:

1) All the above statements other than 4 and 6 are common for all the persistent operations which give you code duplication problem.
2) All the methods in JDBC API are throwing one common exception called java.sql.SQLException which is checked exception. Because of checked exception, you need to write try and catch blocks for every program .
3) There is no clear categorization of exceptions in JDBC.

Above Problems are solved as follows:

1) JdbcTemplate is provided which centralizes the JDBC code.

Usage:

```
String sql="insert into customers values(?,?,?,?)";
Object args[]={"C-101","Sri","sri@jlc","123345"}
jdbcTemp.update(sql,args);

String sql="update customers set email=? where cid=?";
Object args[]={"sri@jlc","C-101"}
jdbcTemp.update(sql,args);

String sql="delete from customers where cid=?";
Object args[]={"C-101"}
jdbcTemp.update(sql,args);
```

2) In Spring Data Access, There is one root exception called DataAccessException which is unchecked or runtime exception. Because of unchecked exception, you no need to write try and catch blocks for every program.
3) In Spring Data Access, There is clear categorization of exceptions.

Important methods of JdbcTemplate

1) int update(sql) - Delete
2) int update(sql,args) - Insert, Update, Delete

3) <ClassType> queryForObject(sql,Class type) - Select
4) Object queryForObject(sql,rowMapper) - Select
5) Object queryForObject(sql,args,rowMapper) - Select
6) Object queryForObject(sql,args,Class) - Select

7) List query(sql,rowMapper) - Select
8) List query(sql,args,rowMapper) - Select

Lab57: Files required

1. Lab57.java	2. Customer.java
3. CustomerDAO.java	4. CustomerDAOImpl.java
5. JLCAppConfig.java	

1. Lab57.java

```
package com.coursecube.spring;

import org.springframework.context.ApplicationContext;
import org.springframework.context.annotation.AnnotationConfigApplicationContext;
/*
* @Author : Srinivas Dande
* @Company : CourseCube
* @Website : www.coursecube.com
* */
public class Lab57 {
public static void main(String[] args) {
ApplicationContext ctx=new AnnotationConfigApplicationContext(JLCAppConfig.class);
System.out.println("Spring Container is Ready");

CustomerDAO custDAO=ctx.getBean("mycustDAO",CustomerDAO.class);
//1.Add Customer
Customer cust1=new Customer(201,"SD","SD@jlc",123,"Blore");
custDAO.addCustomer(cust1);

//2.Update Customer
Customer cust2=new Customer(103,"hello","hello@jlc",55555,"Hyd");
custDAO.updateCustomer(cust2);

//3.Delete Customer
custDAO.deleteCustomer(102);

System.out.println("Done");
}  }
```

3. CustomerDAO.java

```
package com.coursecube.spring;
/*
* @Author : Srinivas Dande
* @Company : CourseCube
* @Website : www.coursecube.com
* */
public interface CustomerDAO {
public void addCustomer(Customer cust);
public void updateCustomer(Customer cust);
public void deleteCustomer(int cid);
}
```

4. CustomerDAOImpl.java

```
package com.coursecube.spring;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.jdbc.core.JdbcTemplate;
import org.springframework.stereotype.Repository;
/*
* @Author : Srinivas Dande
* @Company : CourseCube
* @Website : www.coursecube.com
* */
@Repository("mycustDAO")
public class CustomerDAOImpl implements CustomerDAO {

@Autowired
JdbcTemplate jdbcTemp;

@Override
public void addCustomer(Customer cust) {
String SQL="insert into mycustomers values(?,?,?,?,?)";
jdbcTemp.update(SQL,cust.getCid(),cust.getCname(),cust.getEmail(),cust.getPhone(),cust.getCity());
}

@Override
public void updateCustomer(Customer cust) {
String SQL="update mycustomers set cname=? , email=? , phone=?, city=? where cid=?";
jdbcTemp.update(SQL,cust.getCname(),cust.getEmail(),cust.getPhone(),cust.getCity(),cust.getCid());
}

@Override
public void deleteCustomer(int cid) {
String SQL="delete from mycustomers where cid=?";
jdbcTemp.update(SQL,cid);
}
}
```

5. JLCAppConfig.java

```
package com.coursecube.spring;

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;
/*
* @Author : Srinivas Dande
* @Company : CourseCube
```

```
* @Website : www.coursecube.com
* */
@Configuration
@ComponentScan(basePackages = {"com.coursecube.spring"})
public class JLCAppConfig {

	@Bean
	public DriverManagerDataSource getDS() {
		DriverManagerDataSource ds=new DriverManagerDataSource();

		//ds.setDriverClassName("com.mysql.jdbc.Driver");
		ds.setUrl("jdbc:mysql://localhost:3306/myspringdb");
		ds.setUsername("root");
		ds.setPassword("srinivas");

		return ds;
	}
	@Bean
	public JdbcTemplate getJdbcTemp(DataSource myds) {
		JdbcTemplate jdbcTemp=new JdbcTemplate(myds);
		return jdbcTemp;
	}
}
```

Working with RowMapper:

- When you execute any select statement then data will be stored in the ResultSet object. Now you have to write the code for collecting data from ResultSet object and storing that data in your TO's.
  Ex:

```
List list=new ArrayList();
while(rs.next()){
    CustomerTo cto=new CustomerTO();
    cto.setCid(rs.getInt(1));
    cto.setCname(rs.getString(2));
    ...
    ..
    list.add(cto);
}
```

  - You may get the requirement to write the same code at different places whenever you execute select statement. This gives you the code duplication problem.
  - You can centralize this kind of code with the help of RowMapper.

Steps:

- Write you own class by Implementing **RowMapper** interface.
- Override the following method.
  - **public CustomerTO mapRow(ResultSet rs, int rn) throws SQLException**
- Implement the code inside the mapRow() to move the data from ResultSet to TO's.

Lab58: Files required

1. Lab58.java	**Updated in Lab58**
2. Customer.java	Same as Lab57
3. CustomerRowMapper.java	**Newly Added**
4. CustomerDAO.java	**Updated in Lab58**
5. CustomerDAOImpl.java	**Updated in Lab58**
6. JLCAppConfig.java	Same as Lab57

1. Lab58.java

```
package com.coursecube.spring;

import org.springframework.context.ApplicationContext;
import org.springframework.context.annotation.AnnotationConfigApplicationContext;
/*
* @Author : Srinivas Dande
* @Company : CourseCube
* @Website : www.coursecube.com
* */
public class Lab58 {
public static void main(String[] args) {
ApplicationContext ctx=new AnnotationConfigApplicationContext(JLCAppConfig.class);
System.out.println("Spring Container is Ready");

CustomerDAO custDAO=ctx.getBean("mycustDAO",CustomerDAO.class);
// 1. getCustomersByCid
System.out.println("-----getCustomersByCid------");
Customer cust1 = cdao.getCustomerByCid(1);
System.out.println(cust1);

// 2. getAllCustomers
System.out.println("-----getAllCustomers-----");
List<Customer> list = cdao.getAllCustomers();

list.forEach(cust -> System.out.println(cust));

// 3. getCustomersByEmail
System.out.println("-----getCustomersByEmail-----");
cust1 = cdao.getCustomerByEmail("sri@jlc");
System.out.println(cust1);

// 4. getCustomersByCity
System.out.println("-----getCustomersByCity-----");
list = cdao.getCustomersByCity("Blore");

list.forEach(cust -> System.out.println(cust));

// 5. getCustomerCount
```

```
System.out.println("-----getCustomerCount-----");
int count = cdao.getCustomersCount();
System.out.println("No of Cust : " + count);
// 6. getCustomerCityByEmail
System.out.println("-----getCustomerCityByEmail-----");
String city = cdao.getCustomerCityByEmail("sri@jlc");
System.out.println(city);

// 7. getCustomerPhoneByEmail
System.out.println("-----getCustomerPhoneByEmail-----");
long phone = cdao.getCustomerPhoneByEmail("sri@jlc");
System.out.println(phone);
}
}
```

3. CustomerRowMapper.java

```
package com.coursecube.spring;

import java.sql.ResultSet;
import java.sql.SQLException;
import org.springframework.jdbc.core.RowMapper;
/*
* @Author : Srinivas Dande
* @Company : CourseCube
* @Website : www.coursecube.com
* */
public class CustomerRowMapper implements RowMapper<CustomerTO> {
	@Override
	public CustomerTO mapRow(ResultSet rs, int rn) throws SQLException {

		CustomerTO cto=new CustomerTO();
		cto.setCid(rs.getInt(1));
		cto.setCname(rs.getString(2));
		cto.setEmail(rs.getString(3));
		cto.setPhone(rs.getLong(4));
		cto.setCity(rs.getString(5));
		return cto;
	}
}
```

4. CustomerDAO.java

```
package com.coursecube.spring;
/*
* @Author : Srinivas Dande
* @Company : CourseCube
* @Website : www.coursecube.com
* */
public interface CustomerDAO {
public List<Customer> getAllCustomers();
public Customer getCustomerByCid(int cid);
public Customer getCustomerByEmail(String email);
public List<Customer> getCustomersByCity(String city);
public int getCustomersCount();
public String getCustomerCityByEmail(String email);
public Long getCustomerPhoneByEmail(String email);
}
```

5. CustomerDAOImpl.java

```
package com.coursecube.spring;

import java.util.List;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.jdbc.core.JdbcTemplate;
import org.springframework.stereotype.Repository;
/*
* @Author : Srinivas Dande
* @Company : CourseCube
* @Website : www.coursecube.com
* */

@Repository("mycustDAO")
public class JdbcCustomerDAO implements CustomerDAO {

@Autowired
JdbcTemplate jdbcTemp;

public List<Customer> getAllCustomers() {
String sql = "select * from customers";
List<Customer> list = jdbcTemp.query(sql, new CustomerRowMapper());
return list;
}

public int getCustomersCount() {
String sql = "select count(*) from customers";
return jdbcTemp.queryForObject(sql, Integer.class);
}
```

```
public Customer getCustomerByCid(int cid) {
String sql = "select * from customers where cid=?";
Customer cto =  jdbcTemp.queryForObject(sql, new CustomerRowMapper(), cid);
return cto;
}

public List<Customer> getCustomersByCity(String city) {
String sql = "select * from customers where city=?";
List<Customer> list = jdbcTemp.query(sql, new CustomerRowMapper(), city);
return list;
}

public Customer getCustomerByEmail(String email) {
String sql = "select * from customers where email=?";
Customer cust =  jdbcTemp.queryForObject(sql, new CustomerRowMapper(), email);
return cust;
}

public String getCustomerCityByEmail(String email) {
String sql = "select city from customers where email=?";
String city =  jdbcTemp.queryForObject(sql, String.class, email);
return city;
}

public Long getCustomerPhoneByEmail(String email) {
String sql = "select phone from customers where email=?";
Long ph =  jdbcTemp.queryForObject(sql, Long.class, email);
return ph;
}
}
```

Working with NamedParameterJdbcTemplate

With JdbcTemplate:

```
String sql="delete from customers where cid=?";
int x=jdbcTemp.update(sql,cid);
```

With NamedParameterJdbcTemplate:

```
String sql = "delete from customers where cid = :CID ";
Map<String, Object> params = new HashMap<String, Object>();
params.put("CID", cid);
nameParameterJdbcTemp.update(sql, params);
```

Lab59: Files required

1. Lab59.java	**Updated in Lab59**
2. Customer.java	Same as Lab58
3. CustomerRowMapper.java	Same as Lab58
4. CustomerDAO.java	**Updated in Lab59**
5. CustomerDAOImpl.java	**Updated in Lab59**
6. JLCAppConfig.java	**Updated in Lab59**

1. Lab59.java

```
package com.coursecube.spring;

import java.util.*;
import org.springframework.context.ApplicationContext;
import org.springframework.context.annotation.AnnotationConfigApplicationContext;
/*
* @Author : Srinivas Dande
* @Company : CourseCube
* @Website : www.coursecube.com
* */
public class Lab58 {
public static void main(String[] args) {
ApplicationContext ctx=new AnnotationConfigApplicationContext(JLCAppConfig.class);
System.out.println("Spring Container is Ready");
CustomerDAO custDAO=ctx.getBean("mycustDAO",CustomerDAO.class);

// 1. addCustomer
Customer cust1 = new Customer(8, "SD", "SD@jlc", 1234, "Hyd");
cdao.addCustomer(cust1);

// 2. updateCustomer
Customer cust2 = new Customer(1, "dddd", "dddd@jlc", 8888, "Blore");
cdao.updateCustomer(cust2);

// 3. deleteCustomer
cdao.deleteCustomer(4);

// 4. getCustomerByCid
System.out.println("getCustomerByCid");
Customer cust3 = cdao.getCustomerByCid(1);
System.out.println(cust3);

// 5. getAllCustomers
System.out.println("getAllCustomers");
List<Customer> list = cdao.getAllCustomers();
list.forEach(cust -> System.out.println(cust));
}
}
```

4. CustomerDAO.java

```
package com.coursecube.spring;

import java.util.*;
/*
* @Author : Srinivas Dande
* @Company : CourseCube
* @Website : www.coursecube.com
* */
public interface CustomerDAO {
        public void addCustomer(Customer cust);
         public void updateCustomer(Customer cto);
         public void deleteCustomer(int cid);
         public Customer getCustomerByCid(int cid);
         public List<Customer> getAllCustomers();
}
```

5. CustomerDAOImpl.java

```
package com.coursecube.spring;

import java.util.*;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.jdbc.core.namedparam.NamedParameterJdbcTemplate;
import org.springframework.stereotype.Repository;
/*
* @Author : Srinivas Dande
* @Company : CourseCube
* @Website : www.coursecube.com
* */
@Repository("mycustDAO")
public class JdbcCustomerDAO implements CustomerDAO {

@Autowired
NamedParameterJdbcTemplate  nameParameterJdbcTemp;

public void addCustomer(Customer cust) {
String sql = "insert into customers values(:cid,:cname,:email,:phone,:city)";
Map<String, Object> params = new HashMap<String, Object>();

params.put("cid", cust.getCid());
params.put("cname", cust.getCname());
params.put("email", cust.getEmail());
params.put("phone", cust.getPhone());
params.put("city", cust.getCity());

nameParameterJdbcTemp.update(sql, params);
}
```

```
public void deleteCustomer(int cid) {
String sql = "delete from customers where cid=:cid";
Map<String, Object> params = new HashMap<String, Object>();
params.put("cid", cid);
nameParameterJdbcTemp.update(sql, params);
}

public void updateCustomer(Customer cust) {
String sql = "update customers set cname=:cname,email=:email,phone=:phone,city=:city where
cid=:cid";

Map<String, Object> params = new HashMap<String, Object>();

params.put("cname", cust.getCname());
params.put("email", cust.getEmail());
params.put("phone", cust.getPhone());
params.put("city", cust.getCity());
params.put("cid", cust.getCid());

nameParameterJdbcTemp.update(sql, params);
}

public Customer getCustomerByCid(int cid) {
String sql = "select * from customers where cid = :cid";
Map<String, Object> params = new HashMap<String, Object>();
params.put("cid", cid);

Customer cust = nameParameterJdbcTemp.queryForObject(sql, params, new
CustomerRowMapper());

return cust;
}

public List<Customer> getAllCustomers() {
String sql = "select * from customers";
Map<String, Object> params = new HashMap<String, Object>();

List<Customer> list = nameParameterJdbcTemp.query(sql, params, new
CustomerRowMapper());

return list;
}
}
```

6. JLCAppConfig.java

```
package com.coursecube.spring;

import javax.sql.DataSource;
import org.springframework.context.annotation.*;
import org.springframework.jdbc.core.namedparam.NamedParameterJdbcTemplate;
import org.springframework.jdbc.datasource.DriverManagerDataSource;

/*
* @Author : Srinivas Dande
* @Company : CourseCube
* @Website : www.coursecube.com
* */

@Configuration
@ComponentScan(basePackages="com.coursecube.spring" )
public class JLCConfig {

@Bean
public DataSource mysqlDS() {
DriverManagerDataSource ds = new DriverManagerDataSource();
//ds.setDriverClassName("com.mysql.jdbc.Driver");
ds.setUrl("jdbc:mysql://localhost/myspringdb");
ds.setUsername("root");
ds.setPassword("srinivas");
return ds;
}

@Bean
public NamedParameterJdbcTemplate jdbcTemplate(DataSource dataSource) {
return new NamedParameterJdbcTemplate(dataSource);
}

}
```