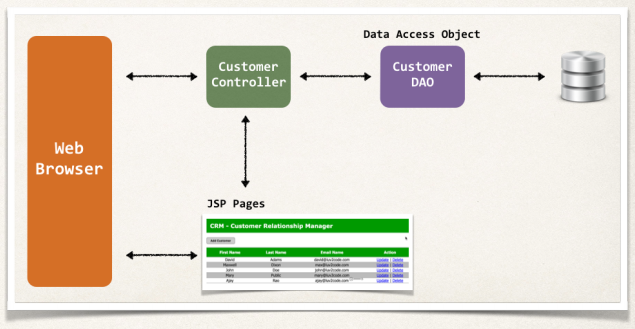
**Development Process**

**List Customer**:

For display the customer in browser as a list from the database, we have to do follow the following process.

1. Define **Customer.java** entity class
2. Define Data-Access-Object (DAO)
   1. **CustomerDAO.java**
   2. **CustomerDAOImpl.java**
3. Define **CustomerController.java**
4. Define JSP page: **list-customers.jsp**



**1) Create Customer.java entity class**:

Entity class is a Java class that mapped to a database table.

**package** com.odduu.ruhul.entity;

@Entity

@Table(name = "customer")

**public** **class** Customer {

@Id

@GeneratedValue(strategy = GenerationType.***IDENTITY***)

@Column(name = "id")

**private** **int** id;

@Column(name = "first\_name")

**private** String firstName;

@Column(name = "last\_name")

**private** String lastName;

@Column(name = "email")

**private** String email;

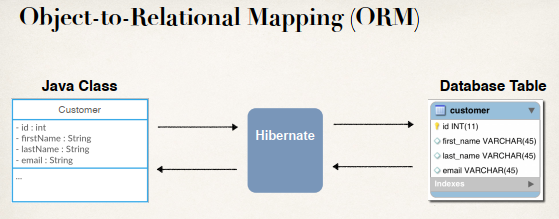
// create constructor

// generate getter setter method

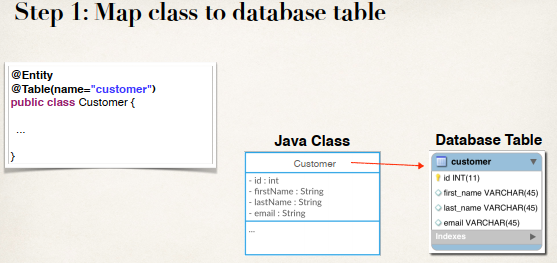
// generate toString method

}

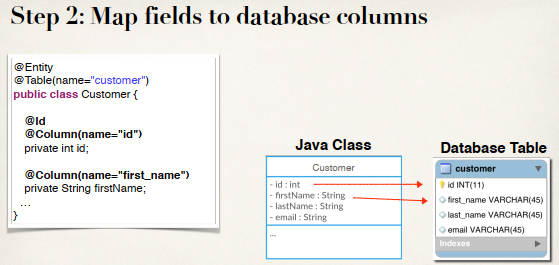
**Object-Relational-mapping demo**:



**Map class to database table**:



**Step 2: Map fields to database columns**:



**Entity Scanning**:

By using Entity scanning Spring/Hibernate knows about our actual entity classes.

<property name=*"packagesToScan"* value=*"com.odduu.ruhul.entity"* />

<!-- Step 2: Setup Hibernate session factory -->

<bean id=*"sessionFactory"*

class=*"org.springframework.orm.hibernate5.LocalSessionFactoryBean"*>

<property name=*"dataSource"* ref=*"myDataSource"* />

<property name=*"packagesToScan"* value=*"com.odduu.ruhul.entity"* />

<property name=*"hibernateProperties"*>

<props>

<prop key=*"hibernate.dialect"*>org.hibernate.dialect.MySQLDialect</prop>

<prop key=*"hibernate.show\_sql"*>true</prop>

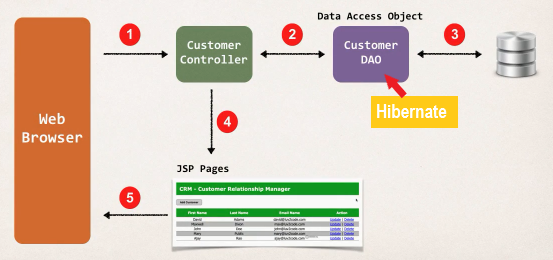
</props>

</property>

</bean>

If we have multiple packages use a comma (,) and use the name.

**2) Define Data-Access-Object (DAO)**:

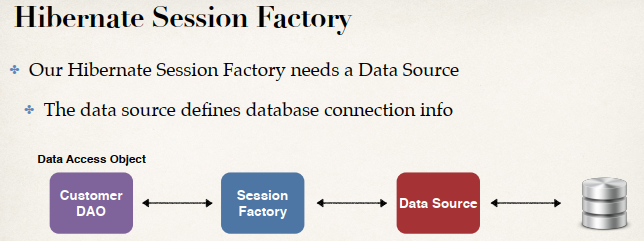


**Customer Data Access Object**:

For Hibernate, our DAO needs a Hibernate SessionFactory. So, we somehow need to get a SessionFactory over to our Data Access Object. And then our SessionFactory actually need a DataSource. The DataSource actually defines the database connection information userid, password etc.

1. Our Hibernate Session Factory needs a Data Source
2. The data source defines database connection info

**Hibernate Session Factory**:



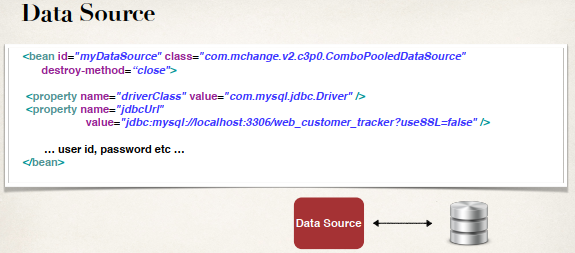
**Dependencies**:

All of the above thing are dependencies. We will write them together with Dependency Injection (DI).

Development Process

**Data Source**:

The DataSource basically tell us how to connect to the database. We define our driver class name, our JDBC URL, user id, password etc. here.



<!-- Step 1: Define Database DataSource / connection pool -->

<bean id=*"myDataSource"* class=*"com.mchange.v2.c3p0.ComboPooledDataSource"*

destroy-method=*"close"*>

<property name=*"driverClass"* value=*"com.mysql.cj.jdbc.Driver"* />

<property name=*"jdbcUrl"*

value=*"jdbc:mysql://localhost:3306/web\_customer\_tracker?*

*useSSL=false&amp;serverTimezone=UTC"* />

<property name=*"user"* value=*"hbstudent"* />

<property name=*"password"* value=*"hbstudent"* />

<!-- these are connection pool properties for C3P0 -->

<property name=*"initialPoolSize"* value=*"5"*/>

<property name=*"minPoolSize"* value=*"5"* />

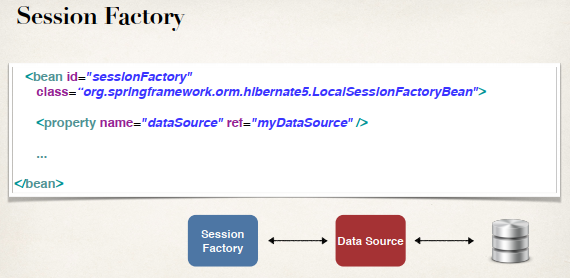
<property name=*"maxPoolSize"* value=*"20"* />

<property name=*"maxIdleTime"* value=*"30000"* />

</bean>

**SessionFactory**:

SessionFactory actually depend on the DataSource. We have SessionFactory and then we inject a property name “*dataSource*” and then ref=*"myDataSource"*. Here *myDataSource* is the bean id that is defined in dataSource.



<!-- Step 2: Setup Hibernate session factory -->

<bean id=*"sessionFactory"*

class=*"org.springframework.orm.hibernate5.LocalSessionFactoryBean"*>

<property name=*"dataSource"* ref=*"myDataSource"* />

<property name=*"packagesToScan"* value=*"com.odduu.ruhul.entity"* />

<property name=*"hibernateProperties"*>

<props>

<prop key=*"hibernate.dialect"*>org.hibernate.dialect.MySQLDialect</prop>

<prop key=*"hibernate.show\_sql"*>true</prop>

</props>

</property>

</bean>



**Customer DAO (Step-by-Step)**:

1. Define DAO interface
2. Define DAO implementation
3. Inject the session factory

**Define DAO interface**:

**package** com.odduu.ruhul.dao;

**import** java.util.List;

**import** com.odduu.ruhul.entity.Customer;

**public** **interface** CustomerDAO {

**public** List<Customer> getCustommers();

**...**

}

**Define DAO implementation**:

**public** **class** CustomerDAOImpl **implements** CustomerDAO {

// need to inject the session factory into the DAO

**private** SessionFactory sessionFactory;

@Override

**public** List<Customer> getCustommers() {

...

}

}

We inject the SessionFactory into the DAO. So, Spring will actually look into the configuration, see the bean id named “*sessionFactory*”, that will inject the bean into the given class “**CustomerDAOImpl**”.



**Spring @Transactional**:

1. Spring provides a special annotation named **@Transactional**
2. Automagically begin and end a transaction for our Hibernate code
3. No need to explicitly do this in our code.
4. This Spring magic happens behind the scenes.

**Spring @Transactional**:

@Transactionalpublic List<Customer> getCustomers() {

// get the current hibernate session Session currentSession = sessionFactory.getCurrentSession();

// create a query Query<Customer> theQuery = currentSession.createQuery("from Customer", Customer.class);

// get the result list List<Customer> customers = theQuery.getResultList(); return customers;}

**Specialized Annotation for DAOs**:

1. **@Repository** Applied to DAO implementations
2. Spring will automatically register the DAO implementation
3. Spring also provides translation of any JDBC related exceptions

**Updates for the DAO implementation**:

@Repository

**public** **class** CustomerDAOImpl **implements** CustomerDAO {

// need to inject the session factory into the DAO

@Autowired

**private** SessionFactory sessionFactory;

@Override

@Transactional

**public** List<Customer> getCustommers() {

...

}

}

**3) Define CustomerController.java:**

Create a controller class “**CustomerController.class**” under the package “**com.odduu.ruhul.controller**”.

**File: CustomerController.class:**

**import** com.odduu.ruhul.dao.CustomerDAO;

@Controller

@RequestMapping("/customer")

**public** **class** CustomerController {

@RequestMapping("/list")

**public** String listCustomer(Model theModel) {

**return** "list-customer";

}

}

**Inject DAO into Controller**:

Now we inject “**CustomerDAO**” in our controller “**CustomerController**”.

**import** com.odduu.ruhul.dao.CustomerDAO;

@Controller

@RequestMapping("/customer")

**public** **class** CustomerController {

// need to inject the customerDAO

**private** CustomerDAO customerDAO;

@RequestMapping("/list")

**public** String listCustomer(Model theModel) {

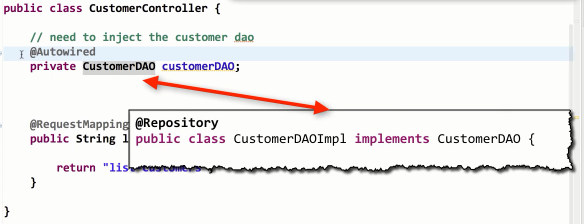
**return** "list-customer";

}

}

**Note**:

1. Now for **@Autowired** Spring will scan for a component that implements CustomerDAO interface.
2. Here CustomerDAOImp.class implement CustomerDAO.
3. In CustomerDAOImp.class we add the annotation **@Repository**.
4. annotation **@Repository** able to find it out and make it available and inject it.



**4) Define JSP page: list-customers.jsp**:

Create a jsp page “**list-customer.jsp**” inside the path “**WebContent/WEB-INF/view/list-customer.jsp**”.

<%@taglib uri=*"http://java.sun.com/jsp/jstl/core"* prefix=*"c"*%>

<!DOCTYPE html>

<html>

<head>

<title>Customer List</title>

</head>

<body>

<div id=*"wrapper"*>

<div id=*"header"*>

<h2>CRM - Customer Relationship Manager</h2>

</div>

</div>

<div id=*"container"*>

<div id=*"content"*>

<!-- Add out html table here -->

<table>

<tr>

<th>First Name</th>

<th>Last Name</th>

<th>Email</th>

</tr>

<!-- Loop over and print out customer -->

<c:forEach var=*"tempCustomer"* items=*"*${customers}*"*>

<tr>

<td>${tempCustomer.firstName}</td>

<td>${tempCustomer.lastName}</td>

<td>${tempCustomer.email}</td>

</tr>

</c:forEach>

</table>

</div>

</div>

</body>

</html>

vdvjvd