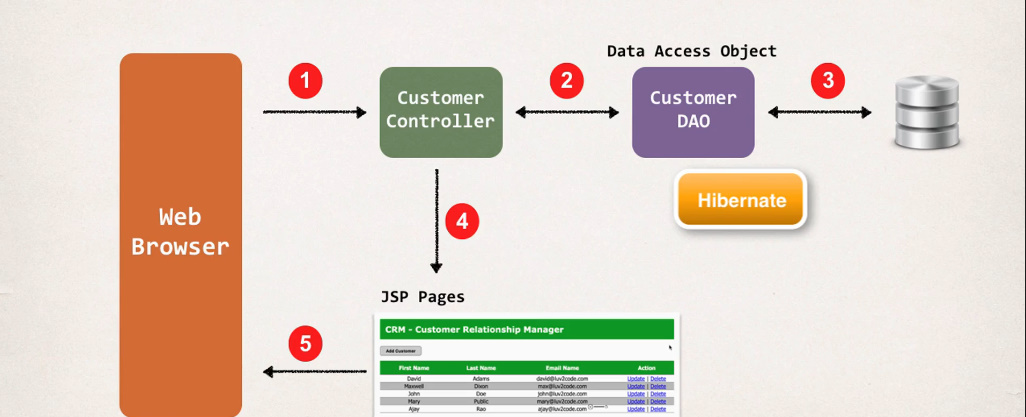
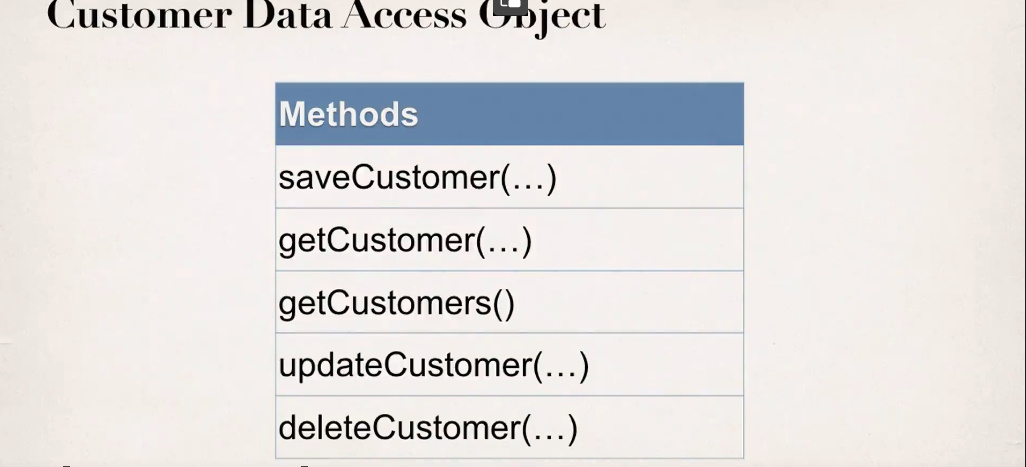
Proiectul va lucra asa:

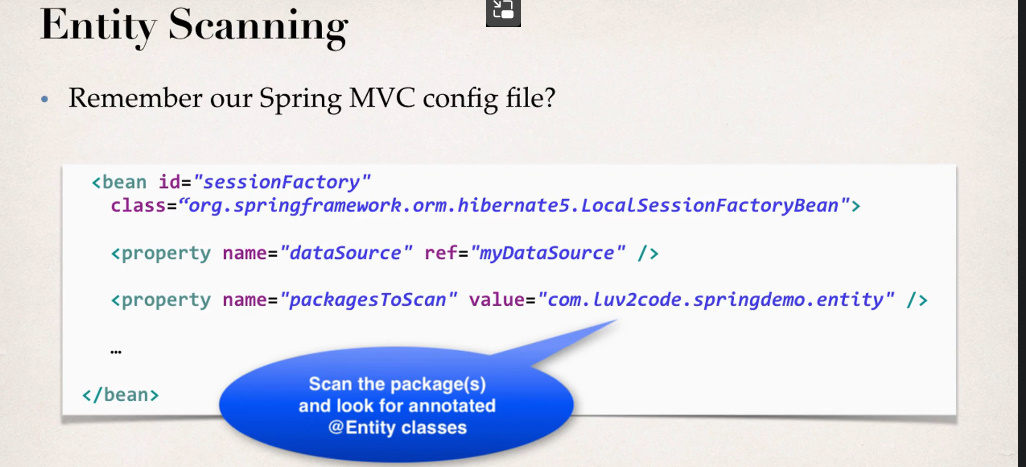


**DAO**

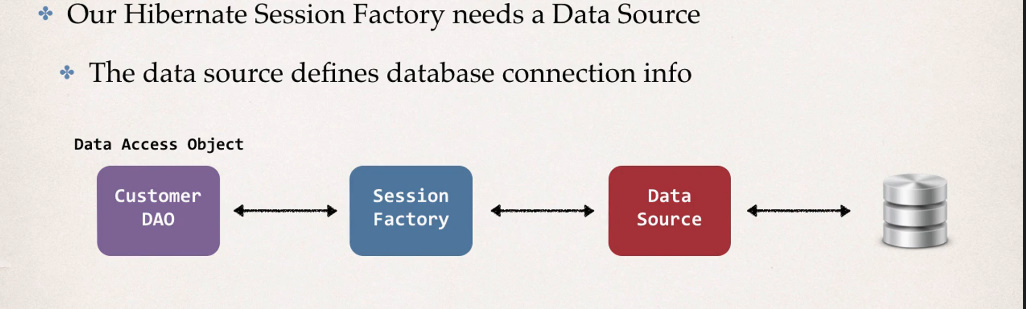
* DAO – data acces object
* Responsabil pentru interactiunea cu baza de date
* Este un design pattern
* Este deobicei o **interfata** ce defineste metodele care se ocupa de diferite operatii referitoare la baza de date.
* De ex, in proiectul dat vom avea o asa interfata care va fi DAO, ce va avea metodele date:



**Entity Scan**



Putem da si mai multe package. Punem dupa primul , si specificam urmatorul.

****Pentru a putea interactiona cu baza de date, avem nevoie de o Session Factory. Session Factory are nevoie de o Data Source. Data Source contine informatiile despre baza de date, pentru a se putea face conexiunea la ea.

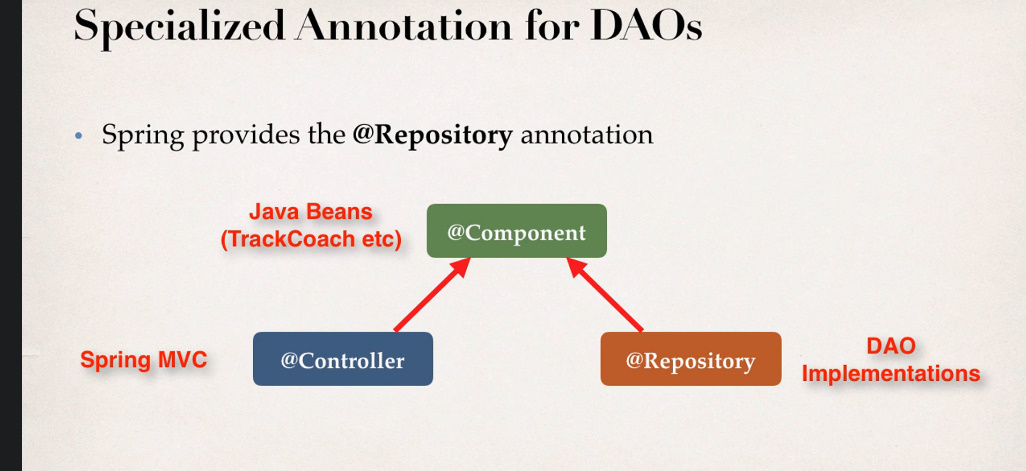
* Toate acestea sunt dependente, deci le putem seta prin Dependency Injection in Customer DAO

**@Transactional**

* Aceasta anotatie ne scuteste de necesitatea de a incepe si inchide mereu tranzactia. Ea o va face.Intai va crea o tranzactie automt pentru sesiune,dand beginTransaction pentru noi, si dupa ce ajunge la final de metoda unde lucram, i va da commit, iar apoi va inchide automat sesiunea, desi de asta se ocupa metoda de destroy setata in bean.
* Totodata se asigura sa fie folosita aceeasi sesiune si Transaction in toate elementele din metoda, chiar de sunt folosite si alte metode din mai multe DAO in aceasta metoda.
* **Session** – interactioneaza cu baza de date. Incepe cand se logheaza in baza de date si se termina cand se delogheaza.
* **Transaction** – operatie facuta in baza de date. Incepe cand in baza de date se efectueaza o operatie concreta si se termina cand ea e finisata. De ex, scriem date, stergem, modificam etc. O sesiune poate avea mai multe transactii, dar doar una la un moment dat e executata.

**O punem mereu deasupra metodei!**

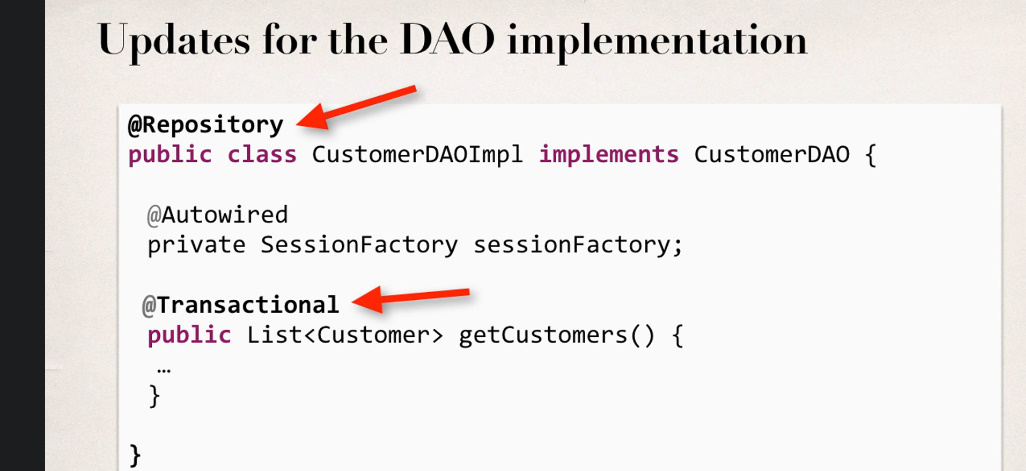
**@Repository**



**@Repository** – este o anotatie care la fel este un @Component,deci e valabilaa pentru autoscanning, dar care trebuie pus la fiecare DAO clasa pentru a arata ca clasa data va comunica cu baza de date.

Ce face ea:

* Spring automat inregistreaza clasele DAO ce sunt @Repository ca beanuri
* orice checked exception e transformata in unchecked



**Cum functioneaza**

Cand cream o sesiune cu bean sessionFactory,Hibernate se va asigura sa creeze o tranzactie pentru sesiune creata cu beanul TransationManager,asa cum obtine sesiunea folosind referinta la beanul sessionFactory ce stie care sesiune tocmai a returnat. Odata ce metoda in care sesiunea s-a generat ajunge la final, beanul pentru Tranzactie da commit automat, si apoi sesiunea, deoarece are destroy-method=”close” se va inchide automat.

@Repository  
public class CustomerDAOImpl implements CustomerDAO{  
  
 @Autowired  
 private SessionFactory sessionFactory;  
  
 @Override  
 @Transactional  
 public List<Customer> getCustomers() {  
 Session session = sessionFactory.getCurrentSession();  
 Query query = session.createQuery("from Customer",Customer.class);  
  
 List<Customer> customers = query.getResultList();  
 return customers;  
 }  
}

Datorita faptului ca @Repository e tot un @Component, el creaza la fel un bean cu numele clasei, dar cu lower-case prima litera, si deci putem face asa:

@Controller  
@RequestMapping("/customer")  
public class CustomerController {  
  
 @Autowired  
 private CustomerDAO customerDAO;  
 @RequestMapping("/list")  
 public String listCustomers(Model model){  
List<Customer> customerList = customerDAO.getCustomers();  
model.addAttribute("customers",customerList);  
 return "list-customers";  
 }  
}

customerDAO va fi automat setat de Spring, asa cum am pus @Repository