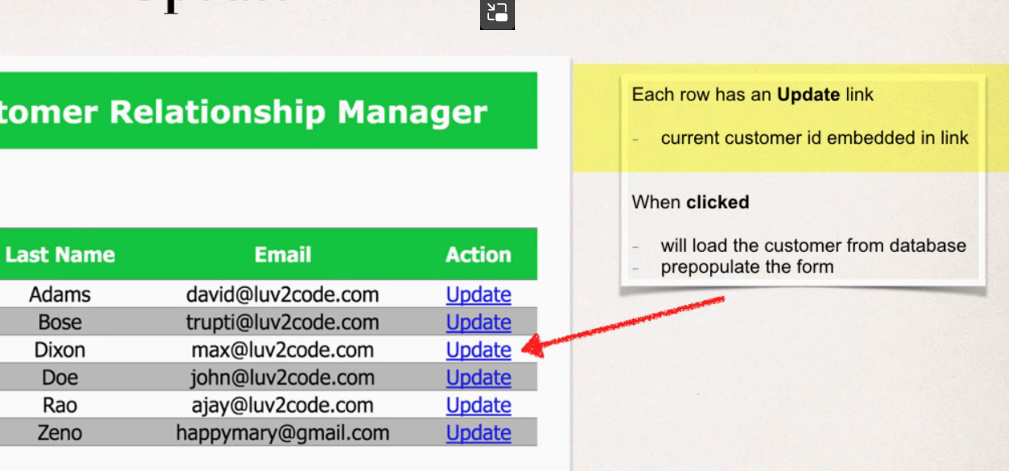
La lista de customers, cream pentru fiecare customer un buton de delete si update. Problema e ca trebuie cumva sa trimitem la Service informatiile despre acel customer ce vrem sa fie sters sau actualizat. Nu mai trimitem tot atributul ce contine lista intreaga de clienti.

**Update**



* Deci, pentru fiecare customer trebuie sa cream cate un param cu atribute(in cazul dat simple atribute ce le are orice obiect in Java, nu atribute ale paginii!!!).
* Putem crea si **variabilele atribut** ale noastre in JSP. De ex, pentru a crea o variabila atribut ce stocheaza un request facem asa:

<**c:url** var="updateLink" value="/customer/showFormForUpdate"/>

deci se va crea o variabila atribut de tip String pentru fiecare Customer ce va stoca valoarea din value, adica acel request, si apoi ne putem referi la el prin ${updateLink}.

* Acum, avem nevoie de un parametru pentru fiecare customer care sa fie creat si trimis cu un request la final de link,in dependenta de customer. Forma are grija ca impreuna cu request din action=”” sa se trimita param si attribute din modelAttribute=””, insa daca forma nu e, noi trebuie sa facem asta.
* <c:url var=”” value=””> creaza o variabila atribut, dar **anume de tip url**, adica request, si deci Spring stie ca acesta e o variabila atribut ce va fi folosita ca request, deci mai trebuie cumva sa facem ca impreuna cu el sa se trimita un param creat de noi la final de link

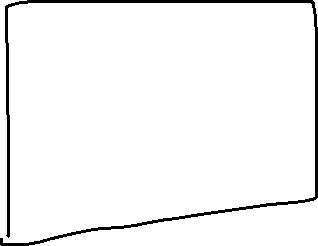
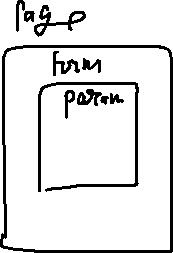
<**c:url** var="updateLink" value="/customer/showFormForUpdate"/>

Param il cream o in interiorul la <c:url>, deoarece cand se va trimite requestul, el va fi sub forma request?name1=valaore1&name2=valoare2&..., adica asa

[http://localhost:8080/customer/**showFormForUpdate?customerId=1**](http://localhost:8080/customer/showFormForUpdate?customerId=1)

**Param niciodata, nici in form, niciunde, nu se creaza ca obiect Java! El se creaza ca obiect JS, adica cu perechi name:value, si e trimis impreuna cu requestul,la final, asa cum e mai sus. Deja in Controller el e transformat in Java Object. Atributele in schimb se trimit ca stream.**

* Pagina are un scope, forma tot are un scope, dar si c:foreach tot are un scope, si fiecare iteratie creaza un scop nou separat de celelalte. In Java, bucla for are un singur scop, insa c:foreach creaza cate un scop pentru fiecare iteratie, caci nu vrem ca datele din iteratiile precedente sa dispara, asa cum in pagina ele pot sa fie refolosite, retrimise etc.



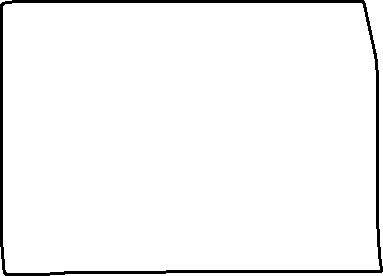
* Pentru a-i spune lui Spring ca vrem sa cream un param ce sa fie trimis cu acel request var attribute, trebuie sa legam param de el asa:

<**c:url** var="updateLink" value="/customer/showFormForUpdate">  
 <**c:param** name="customerId" value="${customer.id}"/>  
 </**c:url**>

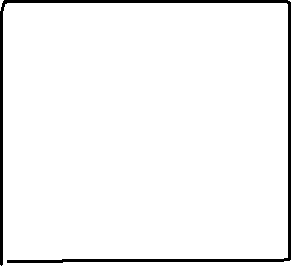
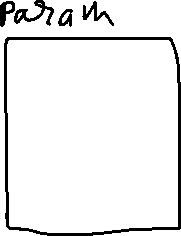
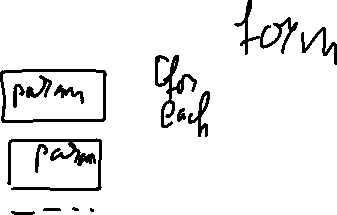
Putem pune si mai multe atribute la acest param, nu conteaza cate,

**Acest param apartine variabilei atribut ce e folosita ca request!Acest param nu exista de sine statator si nu poate fi folosit decat prin acea var atribut**

Chiar daca se creaza mai multe variabile atribute(updateLink) cu acelasi nume, nu e nicio problema, asa cum ele sunt in scops diferite, si nu au nimic unul cu altul. Fiecare param apartine anume acelei var atributAsta ar arata asa:



Deci, fiecare buton update va avea acces la o var atribut si param a ei din scop al sau, si totul e in regula. Chiar si o forma poate avea un c:foreach ce creaza la fiecare iteratie acelasi atribut si param a lui, caci param global al formei si cele din foreach sunt in scopuri separate, si deci daca forma trimite datele, e logic ca se trimite param global, daca se trimit datele din forma, dar dintr-un bloc a lui foreach, se va tribite param din acel bloc.



* **Nu conteaza cate param avem in total in pagina, oricum doar unul mereu se trimite cu orice request!Fiecare buton va avea un param separat la linkul final.**
* Tabel final:

<table>  
 <tr>  
 <th>First Name</th>  
 <th>Last Name</th>  
 <th>Email</th>  
 <th>Action</th>  
 </tr>  
 <**c:forEach** var="customer" items="${customers}">  
  
 <**c:url** var="updateLink" value="/customer/showFormForUpdate">  
 <**c:param** name="customerId" value="${customer.id}"/>  
 </**c:url**>  
  
 <tr>  
 <td>${customer.firstName}</td>  
 <td>${customer.lastName}</td>  
 <td>${customer.email}</td>  
 <td><a href="${updateLink}">Update</a> </td>  
 </tr>  
 </**c:forEach**>  
</table>

**Nu putem adauga param in afara la c:url!!! Asa nu ar fi clar pentru Spring unde se trimit aceste parametri si ce sa faca cu ele.**

* **Nu e nicio problema ca cream mai multe var atribut, asa cum oricum fiecare exista in fiecare iteratie din for si are un param propriu, nu e vorba de unul global, ci de unul din fiecare iteratie, ce va exista doar in scope la *c:for***

Acum adaugam in Controller:

@GetMapping("/showFormForUpdate")  
public String showFormForId(@RequestParam("customerId")int id, Model model){  
 Customer customer = customerService.getCustomer(id);  
 model.addAttribute("customer",customer);  
  
 return "customer-form";  
}

**De ce e @GetMapping?**

1. Cu GET, datele sunt mereu trimise in url la request, insa cu POST nu, ci si sub flux de biti. Deci, param il putem trimite si cu POST.
2. Desi GET e folosit pentru a obtine date, nu iseamna ca el nu poate trimite un param pentru a spune ce fel de date vrea anume. Atribute nu poate trimite, dar param pentru a spune ce date vrea poate. In plus, <a> e un tag ce va face trimitere la o noua pagina, si acea pagina, in cazul nostru, vrea sa afiseze date, deci foloseste GET.

**form:hidden**

* Odata ce vom actualiza datele despre Customer, asa ca deobicei, Forma va crea un nou Customer object pe care il va adauga in Model object al paginii, identic cu attribute venit cu Model object al Controller. Model object al paginii doar nu trimite ce a venit din Model object al Cotroller, insa forma creaza un nou atribut cu nume identic cu cel venit.
* Dar, problema e ca se va trimite obiectul Customer fara id.
* Pentru a rezolva problema asta, putem sa nu folosim neaparat @SessionAttributes ca sa impiedicam forma sa creze un nou obiect, ci facem un hidden camp unde se va scrie id si pur si simplu nu va putea fi modificat, caci e invizibil, si se va crea obiectul tot cu acel id.

<**form:hidden** path="id"/>

**save(), update() saveOrUpdate()**

* save() – creaza un nou tuple in baza de date
* update() – actualizeaza unul nou
* saveOrUpdate() – daca id este null, atunci se face save(), daca nu e null se face update()

Deci modificam:

@Override  
public void saveCustomer(Customer customer) {  
 Session session = sessionFactory.getCurrentSession();  
 session.saveOrUpdate(customer);  
}

**Delete**

Totul e ca si la update, doar ca facem asa pentru a sterge:

@Override  
public void deleteCustomer(int id) {  
 Session session = sessionFactory.getCurrentSession();  
 Query query = session.createQuery("DELETE FROM Customer WHERE id=:id").setParameter("id",id);  
 query.executeUpdate();  
}

query.executeUpdate() pur si simplu executa query, fara a returna ceva.