* Spring MVC Form Tags sunt blocul de baza pentru o pagina web.
* Form tags sunt configurabile si reutilizabile pentru o pagina web.
* Ele ne permit sa setam si sa returnam automat datele de la Java obiecte si beans.
* Sunt comode, prin faptul ca putem returna deodata toate datele din forma, asa cum forma va fi un atribut, si nu fiecare field form din ea va fi ca un atribut, pe care trebuie pe fiecare sa il prelucram.

**Form Tags**

* Form:form – main form containter
* Form:input – text field
* Form:textatea – multi-line text field
* Form:checkbox
* Form:radiobutton
* Form:select

Pentru ca ele sa functioneze, in pagins JSP trebuie pus:

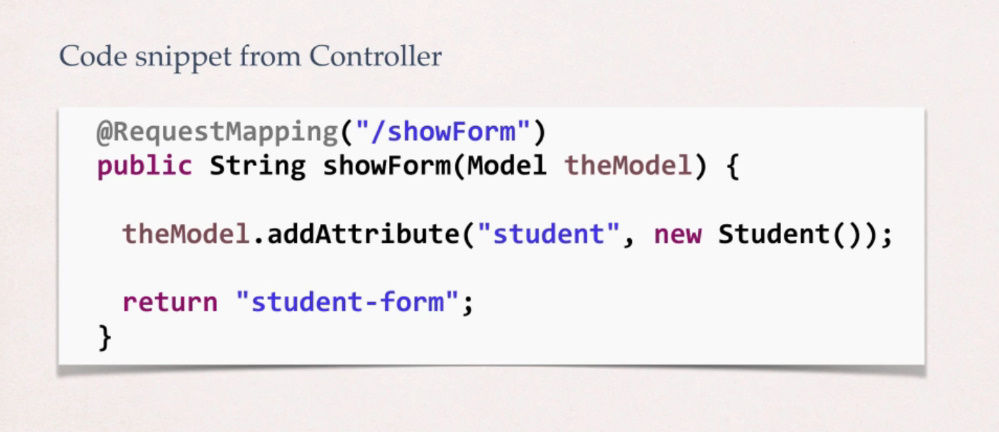
<%@ **taglib** prefix="form" uri="http://www.springframework.org/tags/form"%>

Vom avea nevoie iar de un Model object. **Model object este tot un bean!**

**Model object transfera date intre controllers si views.**

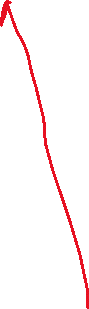
**Acest tip de form, desi e identic cu cel din HTML, e de tip Spring MVC, si ne permite sa facem mai multe lucruri.**

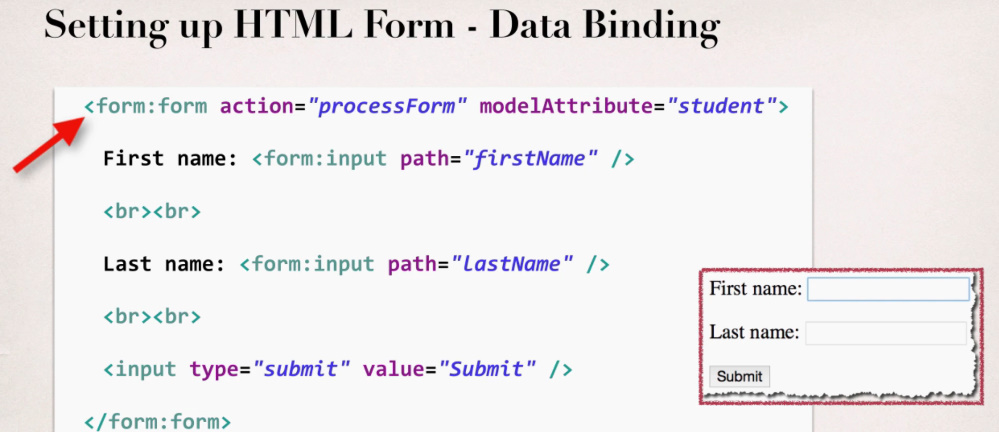
**Configurarea unui form**

****



In model, nu-i mai dam cate un String, ci un obiect Student, ce va contine toate datele introduse de student in forma, pe care noi le vom recupera cu numele “student” . Asa nu trebuie sa prelucram fiecare atribut din forma HTML, ci forma MVC are ea un atribut .





Cand folosim forme de la Spring MVC, ele se diferentiaza de formele clasice prin:

* Forma are **modelAttribute** atribut
* Fiecare <input> nu mai are atributul **name**, ci **path**
* Numele la modelAttribute din form si din addAtribute() de la model, trebuie sa fie identice. Forma are modelAttribute, din cauza ca nu e nevoie sa se recurga la un obiect model creat de pagina care sa duca datele la Controller, asa cum se facea cu forma clasica din HTML, unde <form> nu avea modelAttribute, dar fieldurile aveau name, ce le puneau in link trimis de pagina si transformate apoi de Spring in obiectul param, si era trimis la Controller. Aici noi nu vrem sa se utilizeze obiecul param,deci parametri, caci el nu stie ca noi vrem un obiect de tip Student, nu Stringuri, ci vrem sa se foloseasca anume obiectul trimis de Controller, cand s-a accesat pagina cu forma. Deci, modelAttribute acceseaza obiectul trimis de Controller, caci el doar vrea sa ia obiectul de tip Student trimis de Controller,ca sa poata afisa datele in form, si apoi deci sa **creeze un nou obiect de tipul celui trimis**, ca sa adauge doar date in el, adica sa fie un nou obiect de tipul celui venit cu Model object cu datele inregistrate in el, care sa fie inregistrat in Model object cu acelasi atribut si deci sa fie trimis inapoi la Controller acest obiect nou, dar cu numele atributul celui vechi. Obiectul theModel e ca o minge ce va fi trimisa la Pagina si apoi inapoi la Controller.

Daca vom verifica adresa obiectului trimis cu Model Object formei si adresa obiectului ce a venit de la forma la Controller, cu acelasi Atribut, adresele vor fi diferite.

public String formPage(Model model){  
 ClassPathXmlApplicationContext context = new ClassPathXmlApplicationContext("classpath:applicationContext.xml");  
  
 model.addAttribute("client",context.getBean("MyClient",Client.class));  
 System.*out*.println(context.getBean("MyClient",Client.class));  
 return "form";  
}  
  
@RequestMapping(value="/processForm", method = RequestMethod.*POST*)  
public String registerPage(@Valid @ModelAttribute("client") Client client, BindingResult bindingResult, Model model){  
  
  
 if(bindingResult.hasErrors()){  
 return "form";  
 }  
 else {  
 System.*out*.println(client);  
 return "register";  
 }  
}

[org.app.test.Client@3b4a06c3](mailto:org.app.test.Client@3b4a06c3)

org.app.test.Client@315271c2

**Operatiile efectuate de Spring**

* 1. Path e folosit pentru a lega un form field la o proprietate la un bean sau mai bine zis pentru a executa metodele beanului.

Cand Spring va incepe sa procese forma, el va popula form field,adica o va umple cu date de la Obiectul venit. Mai sus, forma are **modelAttribute=”student”** ,deci acceseaza obiectul “student” venit de la Controller prin Model Object.

De ex prima form field are

**path=”firstName”,**

deci cand Spring o va procesa, el va executa **student.getFirstName()** din noul obiect creat

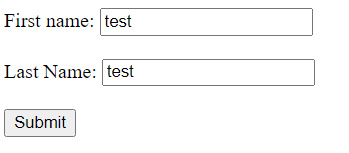
!!!get e pus automat, si se face iar capitalizarea la prima litera**.Path incepe mereu cu litera mica**

Prin getFirstName(), spring va executa metoda get si ce va colecta din obiect va insera in field form. In unele cazuri, am putea vrea ca Spring sa introduca ceva acolo deodata ce userul acceseaza forma, dar in cazul nostru cu studentul, e inutil, si deci daca se returneaza null, field form ramane gol.

De ex, daca avem o clasa unde avem:

public class Student {  
  
 private String firstName="test";  
 private String lastName="test";

se va intampla asta la deschiderea paginii cu forma:



* 2. Cand forma e trimisa, Spring va crea un nou obiect de tipul celui venit, ca sa fie un obiect cu datele puse de noi, fara a copia ceva din obiectul venit de la Controller, si in el va rula metodele **student.setFirstName() ca sa adauge in Object creat de tip de cel de la Controller valoarea din field introdusa.**

Cu ea el se asigura ca se seteaza varibila Name(daca asa o numim) cu metoda setFirstName cu datele introduse de user. Apoi acest obiect nou creat va fi daugat in Model object si va fi trimis la Controller.

Putem crea un controller si asa



Folosind @ModelAttribute in loc de @RequestParam, deoarece @RequestParam se refera la param, care acum e gol, dar @ModelAttribute se va referi la attribut, adica cel tot trimis de Controller la pagina si apoi returnat,care defapt va fi unul nou, doar ca cu acelasi atribut. @ModelAttribute se asigura ca sa se creeze in Model obj al Controller o referinta a atributului venit de la pagina in Model obj al paginii si returneaza aceasta referinta parametrului.

* 3. Pentru a accesa datele in pagini JSP, folosim ${attributeName.path}, unde attributeName e numele obiectului. Nu folosim param. **Spring mereu trimite obiectul Model catre pagina ce se deschide din return**

**Exemplu**

1. Student class

public class Student {  
  
 private String firstName;  
 private String lastName;  
  
 public Student(){ }  
  
 public String getFirstName() {  
 return firstName;  
 }  
  
 public void setFirstName(String firstName) {  
 this.firstName = firstName;  
 }  
  
 public String getLastName() {  
 return lastName;  
 }  
  
 public void setLastName(String lastName) {  
 this.lastName = lastName;  
 }  
}

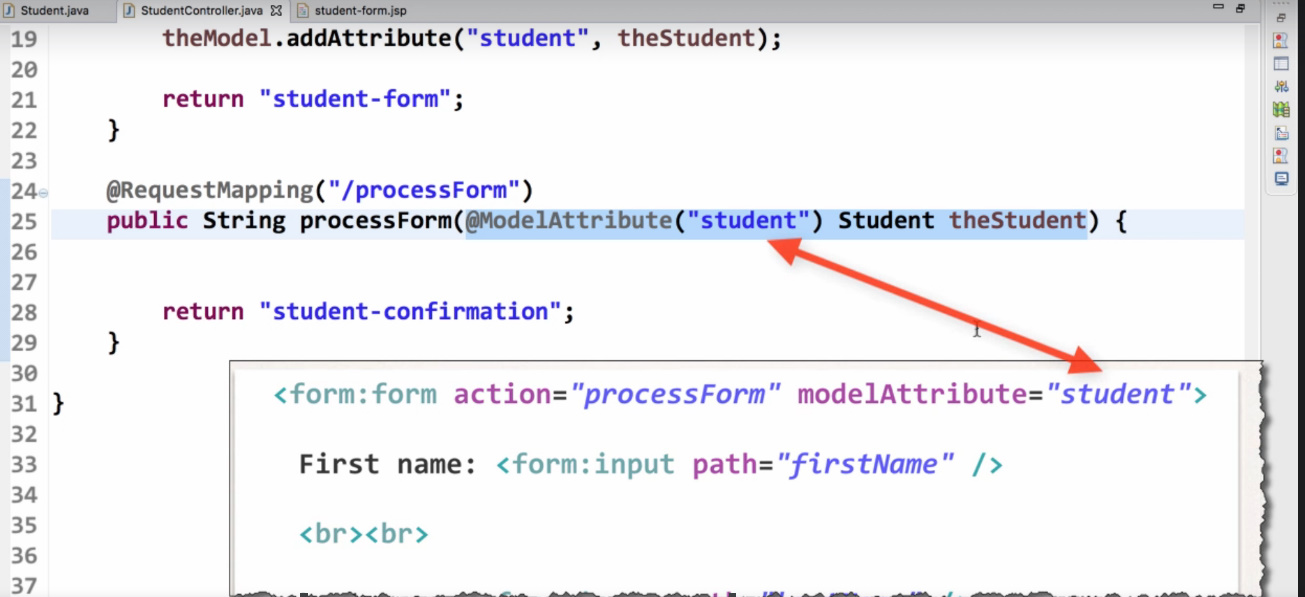
1. StudentController class

@Controller  
@RequestMapping("/student")  
public class StudentController {  
  
 @RequestMapping(value = "/showForm", method = RequestMethod.*GET*)  
 public String showForm(Model model){  
  
 Student theStudent = new Student();  
  
 model.addAttribute("student",theStudent);  
 return "student-form";  
 }  
  
 @RequestMapping("/processForm")  
 public String procesForm(@ModelAttribute("student") Student student){  
 System.*out*.println(student.getFirstName() + " "+student.getLastName());

return "student-confirmation";  
 }  
}

1. Form

<**form:form** action="processForm" modelAttribute="student">  
 First name: <**form:input** path="firstName"/>  
 <br><br>Last Name: <**form:input** path="lastName"/>  
  
 <br><br>  
 <input type="submit" value="Submit"/>  
</**form:form**>



Deci, forma, data expediata, va trimite toate informatiile la clasa StudentController, catre metoda processForm, si toate datele sale vor fi trimsie obiectului theStudent(student in ex)

1. Afisarea datelor pe o noua pagina
2. <body>  
   The studdent is confirmed: ${student.firstName} ${student.lastName}  
   </body>

**Cum a functionat programul**

1. Deci, am creat o clasa ce va primi Request de tipul /student, adica toate metodele sale vor primi request ce incep mereu cu /student. Metoda showForm() se ocupa de afisarea formei de completat. Insa, in momentul cand se afiseaza forma, ne trebuie sa cream un obiect de tip student in Model Object, ca forma sa il poata accesa, si noi apoi sa returnam datele introduse de ea cumva, dar Model object al paginii nu va sti de asta. Deci in Controller, cream un model object, ce va avea un atribut “student”,deci un obiect cu numele “student”. Apoi acest model Object va fi trimis la Pagina.Vezi mai jos la forma.
2. Pagina acum are 2 model Objects, al ei si cel venit de la Controller. Deoarece forma are atributul modelAttribute, ea nu va adauga obiecte model object al paginii, ci in cel al Controller. modelAttribute de la forma coincide cu cel din metoda showForm , adica din metoda ce l-a trimis. Ele coincid din simplul motiv ca toate apeleaza la obiectul Student cu atribut sau id/nume “student” din Model Object veni de la Controller.
3. Model object trimis de Controller la pagina are datele necesare pentru a afisa forma, insa dupa trimiterea formei, se va crea un nou obiect identic ca tipul celui trimis si in el se vor pune toate datele colectate din forma, adica introduse in forma, si el va fi adaugat in noul Model Object pe care pagina trebuie sa il trimita la Controller si asa la Controller nu va fi returnat obiectul trimis de el, ci altul nou cu datele din formular, dar sub acelas i atribut.Metoda processForm se ocupa de obtinerea datelor dupa procesarea formei si afisarea unei noi pagini in care apar datele colectate. Selectam cu @ModelAttribute obiectul din Model Object inregistrat cu Atributul “student” sau id, adica obiectul caruia forma i-a trimis datele. Mai apoi doar il afisam. Folosim @ModelAttribute dar nu @RequestParam, deoarece @RequestParam s-ar referi la obiectul param, dar noi vrem referinta la un atribut, caci nici nu avem ceva in param. In plus, Atributele nu se trimit automat, si @ModelAttribute se asigura ca atributul sa fie adaugat in Model al Controller ce va di trimis paginii din return
4. Pentru a afisa datele, folosim pur si simplu atributul(numele la obj) si . path, care exact ca la form va pune get+nume path pentru a obtine obiectele. De remarcat ca Spring se va asigura sa trimita Model Object al Controller si catre pagina din return, chiar de nu are un Model model ca parametru. Daca vrem, putem defini noi unul, sa adaugam obiectul venit de la forma in el, sau sa il modificam, dar e inutil daca nu modificam datele, caci oricum @ModelAttribute va adauga o referintaa a acestui atribut in Model al Controller ce se va duce la pagina din return.

@RequestMapping("/registed")  
public String showRegisted(@ModelAttribute("studentObj") Student student, Model model){  
 System.*out*.println("Student registed");  
 System.*out*.println("FirstName: "+student.getFirstName());  
 System.*out*.println("LastName: "+student.getLastName());  
  
 model.addAttribute("studentka",student);  
  
 return "registed";  
}



**Atentie! Clasele trebuie sa fie publice, caci Spring va avea nevoie sa faca casturi cu ele**

**Atributele si referintele sunt stocate intr-un Map**

* ATENTIE! Atributele venite de la pagina nu se trimit automat fara @ModelAttribute! Folosirea lui garanteaza ca se va crea o referinta in Model obj al Controller.
* @Controller  
  public class control {  
   @RequestMapping("/")  
   public String index(Model model){  
   model.addAttribute("test1",new Teacher("Mititiuc","Eduard",40));  
   return "index";  
   }  
    
   @RequestMapping("/form")  
   public String form(){  
   return "test";  
   }  
  }

**Daca pagina principala are o forma cu modelAttribute = “test1”, si daca in metoda form() nu vom folosi @ModelAttribute(“test1”) Teacher teacher, atributul test1 nu se va trimite automat!@ModelAttribute asigura si trimiterea acestui atribut catre noua pagina.**

* **Chiar daca HttpServletRequest nu e setat ca parametru, oricum Model object venit de la pagina va trimite automat ce a inregistrat de la forma la noua pagina, daca facem o forma fara atribut,insa asta nu se face si in cazul atributului daca forma e cu atribut, fara @ModelAttribute**
* @Controller  
  public class control {  
   @RequestMapping("/")  
   public String index(Model model){  
   model.addAttribute("test1",new Teacher("Mititiuc","Eduard",40));  
   return "index";  
   }  
    
   @RequestMapping("/form")  
   public String form(HttpServletRequest servletRequest){  
   return "test";  
   }  
  }



**@ModelAttribute vs @RequestParameter**

* ModelAttribute face referinta la un atribut, iar @RequestParameter la obiectul param
* @ModelAttribute poate returna orice tip de obiecte,depinde noi ce am trimis, insa @ReqeuestParameter doar String sau String[]
* @RequesParameter e ecvhivalent cu **new HttpServletRequest.getParameter(“nume**”), insa pentru @ModelAttribute nu exista o metoda asa simpla.
* ***fara @ModelAttribute, atributul nu va fi trimis automat catre pagina returnata, caci doar param si ce setaam cu Model se trimit automat, nu si model venite de la pagina!!!Insa, parametrii se trimit automat si fara @RequestParameter***

**Ordinea de adaugare a atributelor**

* @ModelAttribute e primul care adauga referinta atributului de la Model venit de la pagina in Model al Controller
* Un obiect Model model creat de noi va primi referinta la Model al Controller. Daca el creaza un atribut deja existent in Model al Controller, el pur si simplu va schimba referinta lui si deci si obiectul de dupa @ModelAttribute va avea referinta la acest obiect deja, caci atributele sunt pastrate intr-un fel de HashMap sub forma numeAtribut : referinta, si @ModelAttribute ofera referinta la acest numeAtribut , si apoi foloseste deja referinta pastrata de el.