# SPRİNG MVC NOTLARI

# Spring MVC Nedir?

MVC veya **M**odel-**V**iew-**C**ontroller geliştirmenin farklı parçalara ayrılarak kolay yönetilmesini sağlayan tasarım mimarisidir.

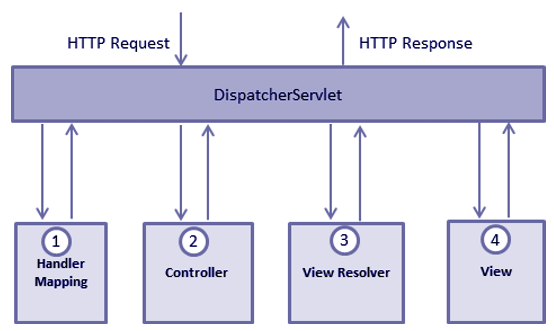
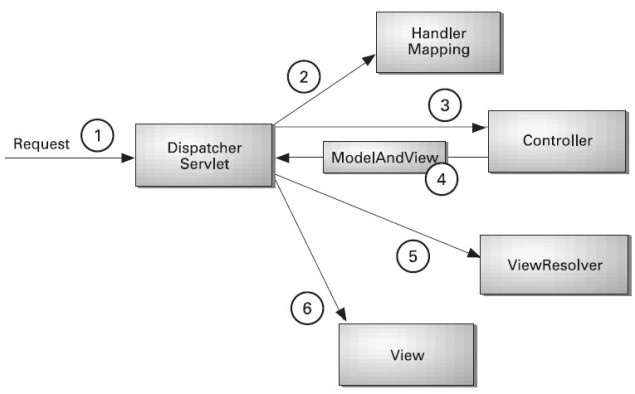
**Model** – Verilerin modellendiği genellikle sıradan Java sınflarının(POJO) kullanıldığı bölümdür.

**View** – Sonucun gösterildiği bölümdür.

**Controller** – Gelen isteğe göre genellikle model içerisinde yer alan verileri kullanarak işlem yapan ve sonucu view katmanına ileten bölümdür.

# Spring MVC Mantığı

# Web sitelerinden bir istek gelir.Bu kullanıcının yapmak istediği herhangi bir şey olabilir.Giriş yap,Hakkımızda sayfasına git,Kaydol gibi herhangi bir şey olabilir.Bu isteği ilk karşılayan dispatcher servlet’tir.Daha sonra dispatcher servlet gelen bu isteğin hangi controller sayfasına gideceğini handler mapping ile tespit eder.Ve ilgili Controllere yollar,Controllerde servis,veri erişim yada diğer işlemlerin olduğu Model’den verileri çeker,günceller,yada siler,Daha sonra sonucu View’e yollar.\*\*Anotasyon yoluyla yaparsak eğer handler mapping ayarlarıyla vs uğraşmamıza gerek yok.



# Nasıl Spring MVC projesi açılır?

Yeni proje diyoruz.Projemiz Spring Legacy Project olucak,Templates’ten Spring MVC Project’i seçiyoruz.Ve daha sonra Ana paketimizin ismini veriyoruz.Burada com.project.controller gibi 3 kelimeli olmalısı şart.Artık projemiz hazır.

Java Resources > src/main/java > com.project.x altındaki class bizim controller classımız.

Deployed Resources > wepapp > WEB INF > Views > xxx.jsp adındaki dosya bizim view dosyamızdır.Bunun içine html,css,js,angular yada jsf,jsp kodları yazılabilir.

Deployed Resources > wepapp > WEB INF > spring > appServlet > servlet-context.xml adındaki dosya bizim dispatcher servletimizdir.Burada gelen istekleri handler mapping ile ilgili controllere yolluyoruz.

NOTLAR

1.Burdaki prefix view dosyasının nerede olduğu,suffix ise bu dosyanın uzantısının ne olduğudur.

servlet-context.xml

<beans:bean

class=*"org.springframework.web.servlet.view.InternalResourceViewResolver"*>

<beans:property name=*"prefix"* value=*"/WEB-INF/views/"* />

<beans:property name=*"suffix"* value=*".jsp"* />

</beans:bean>

2.Deployed Resources > wepapp > WEB INF > web.xml içine yazdığımız şu aşşağıdaki kod index.jsp dosyasının siteye ilk girildiğinde açılacak ilk sayfa olduğunu belirtir.FAKAT bu dosyanın Deployed Resources > wepapp içerisinde olması gerekiyor.

<welcome-file-list>

<welcome-file>index.jsp </welcome-file>

</welcome-file-list>

2.Deployed Resources > wepapp > WEB INF > web.xml içindeki şu hazır gelen kodda url-pattern arasına .abc yazarsak artık,sayfalara gitmek için yazdığımız urlnin sonunda .abc gelmesi gerektiğini belirtiyor.

<servlet-mapping>

<servlet-name>appServlet</servlet-name>

<url-pattern>/</url-pattern>

</servlet-mapping>

3. Soldan projeye sağ tıklayarak properties > web project settings içerisindeki alan ise domain dışında siteye nasıl gidileceğini belirtiyor.Örneğin o alanda /samet yazıyor ise ve /home viewindan ana sayfaya gitmek istiyorsam www.abc.com/samet/home yazmam lazım.O alana yalnızca / koyarsak www.abc.com/home ile direk anasayfaya gidebiliriz.

Basit bir Spring MVC projesi

(Bu projeyi Anotasyon yoluyla yaptığımız için dispatcher servlete handler mapping gibi ekstra bir kod yazmamız gerekmiyor.)

@Controller

**public** **class** HomeController {

@RequestMapping(value = "/hosgeldiniz", method = RequestMethod.***GET***)

**public** String home(Model model) {

model.addAttribute("mesaj", "Merhaba"); //home.jsp'nin içindeki ${mesaj} jstl coduna değer atadık.

**return** "home"; //bu controlleri ilgilendiren view dosyasının adı home.jsp

}

---------BUDA ALTERNATİFİ-----------

@RequestMapping(value = "/hosgeldiniz", method = RequestMethod.GET)

**public** ModelAndView home() {

ModelAndView modelAndView = **new** ModelAndView("home"); //bu controlleri ilgilendiren view dosyasının adı home.jsp

modelAndView.addObject("mesaj", "Merhaba"); //home.jsp'nin içindeki ${mesaj} jstl coduna değer atadık.

**return** modelAndView;

}

home.jsp

<html>

<head>

<title>Ana Sayfa</title>

</head>

<body>

<h1>

${mesaj}

</h1>

</body>

</html>

**@PathVariable Anotasyonu**

/\*RequestMapping anotasyonunun içine url adresini verdik ve jstl olarak {times} yazdık hemen altına home metodunun parametresine -@PathVariable("times") String time- yazdık.Bu şu demek url adresinde 50 yazarsa bunu String türündeki time değişkeni ile eşle.Yani url'ye yazılan sayının değerini alacak.\*/

@RequestMapping(value = "/home/helloworld/{times}", method = RequestMethod.***GET***)

**public** String home(Model model,@PathVariable("times") String time) {

model.addAttribute("sayhello", time+" kere Hello World.!");

**return** "home";

}

@RequestMapping(value = "/home/merhabadunya/{times}", method = RequestMethod.***GET***)

**public** String homes(Model model,@PathVariable("times") String time) {

model.addAttribute("sayhello", time+" kere Merhaba Dünya.!");

**return** "home";

}

//------birden fazla parametreli işlemler-----------//

@RequestMapping(value = "/{user}/{times}", method = RequestMethod.***GET***)

**public** String homess(Model model,@PathVariable("times") String time, @PathVariable("user") String user) {

model.addAttribute("sayhello", time+" kere hosgeldiniz " +user);

**return** "home";

}

**@HandlerMapping**

Bu handler mappingin mantığı şudur.Normalde Controller falan yazarak anotasyon yoluyla mvc projesi yapıyoruz ki çok kullanılanda bu.Fakat bu handler mapping ile ilk servlet context xml(dispatcher servlet)den gelen isteği karşılayıp hangi controllere gideceğini orada belirtiyoruz.Anotasyon kullanmıyoruz.

Artık anotasyon yok

**public** **class** AnasayfaController **extends** AbstractController {

**protected** ModelAndView handleRequestInternal(HttpServletRequest request, HttpServletResponse response)**throws** Exception {

ModelAndView modelAndView = **new** ModelAndView("index");

modelAndView.addObject("mesaj", "Anasayfa");

**return** modelAndView;

}

}

**public** **class** HakkindaController **extends** AbstractController {

@Override //Herhangi bir Request Mapping veya Controller Anotasyonu yazmadık

**protected** ModelAndView handleRequestInternal(HttpServletRequest request, HttpServletResponse response) **throws** Exception {

ModelAndView modelAndView = **new** ModelAndView("index");

modelAndView.addObject("mesaj", "Hakkında");

**return** modelAndView;

}

}

servlet-context.xml

Karşılama ve ilgili controllere yollama işlemini burada yapıyoruz.

-BeanNameUrlHandlerMapping ile

<beans:bean name=*"/anasayfa.htm"*

class=*"com.project.controller.AnasayfaController"*>

</beans:bean>

<beans:bean name=*"/hakkinda.htm"*

class=*"com.project.controller.HakkindaController"*>

</beans:bean>

<beans:bean name=*"BeanNameUrlHandlerMapping"*

class=*"org.springframework.web.servlet.handler.BeanNameUrlHandlerMapping"*>

</beans:bean>

-SimpleUrlHandlerMapping ile (bunun üsttekinden farkı,birden fazla url’yi bir view’a yönlendirebiliriz.)

<beans:bean name=*"anasayfa"*

class=*"com.project.controller.AnasayfaController"*>

</beans:bean>

<beans:bean name=*"hakkinda"*

class=*"com.project.controller.HakkindaController"*>

</beans:bean>

<beans:bean name=*"simpleUrlHandlerMapping"*

class=*"org.springframework.web.servlet.handler.SimpleUrlHandlerMapping"* autowire-candidate=*"false"*>

<beans:property name=*"mappings"*>

<beans:props>

<beans:prop key=*"/anasayfa.htm"*>anasayfa</beans:prop>

<beans:prop key=*"/anasayfa"*>anasayfa</beans:prop>

<beans:prop key=*"/hakkinda.htm"*>hakkinda</beans:prop>

<beans:prop key=*"/aboutus"*>hakkinda</beans:prop>

</beans:props>

</beans:property>

</beans:bean>

index.jsp

<!DOCTYPE html>

<html lang=*"en"*>

<head>

<title>${mesaj} }</title>

</head>

<body>

<div class=*"container"*>

<h2>${mesaj}</h2>

<div class=*"alert alert-success"*>

<strong>${mesaj}!</strong> ${mesaj}

</div>

</div>

</body>

</html>

**MultiActionController(XML)**

**Bu konunun mantığı bir controlleri içerisinde birden fazla işlem yapmadır.**

**web.xml**

web.xml e şu aşağıdak kodu yazarak ilk önce açılacak viewin index.jsp olduğunu belirttik.Daha sonra bu view’da ekleye basınca a href propertisindeki linke gitmesi gerektiğini belirttik ve home jsp içerisinde de bu a hreflerin ilgili olduğu linkleri tanımladık.

<welcome-file-list>

<welcome-file>index.jsp </welcome-file>

</welcome-file-list>

**Controller Class**

**public** **class** HomeController **extends** MultiActionController {

**public** ModelAndView add (HttpServletRequest request, HttpServletResponse response) **throws** Exception {

ModelAndView modelAndView = **new** ModelAndView("home");

modelAndView.addObject("mesaj", "Add");

System.***out***.println("Eklendi");

**return** modelAndView;

}

**public** ModelAndView delete (HttpServletRequest request, HttpServletResponse response) **throws** Exception {

ModelAndView modelAndView = **new** ModelAndView("home");

modelAndView.addObject("mesaj", "Delete");

System.***out***.println("Silindi");

**return** modelAndView;

}

**public** ModelAndView update (HttpServletRequest request, HttpServletResponse response) **throws** Exception {

ModelAndView modelAndView = **new** ModelAndView("home");

modelAndView.addObject("mesaj", "Update");

System.***out***.println("Güncellendi");

**return** modelAndView;

}

**index.jsp**

<body>

<div class=*"container"*>

<h2>SECENEKLER</h2>

</div>

<div class=*"alert alert-success"*>

<a href=*"home/add"*>EKLE</a>

</div>

<div class=*"alert alert-warning"*>

<a href=*"home/delete"*>SIL</a>

</div>

<div class=*"alert alert-info"*>

<a href=*"home/update"*>GUNCELLE</a>

</div>

</body>

**home.jsp**

<body>

<div class=*"container"*>

<h2>${mesaj}</h2>

<div class=*"alert alert-warning"*>

<strong>${mesaj}</strong>

</div>

</div>

</body>

**servlet-context.xml**

<beans:bean name=*"/home/\*.html"*

class=*"com.project.controller.HomeController"*>

</beans:bean>

**MultiActionController(Annotation)**

**Bu konunun mantığı bir controlleri içerisinde birden fazla işlem yapmadır.**

**web.xml**

web.xml e şu aşağıdak kodu yazarak ilk önce açılacak viewin index.jsp olduğunu belirttik.Daha sonra bu view’da ekleye basınca a href propertisindeki linke gitmesi gerektiğini belirttik ve home jsp içerisinde de bu a hreflerin ilgili olduğu linkleri tanımladık.

<welcome-file-list>

<welcome-file>index.jsp </welcome-file>

</welcome-file-list>

**Controller Class**

@Controller

@RequestMapping(value = "/home", method = RequestMethod.***GET***)

**public** **class** HomeController {

@RequestMapping(value = "/add", method = RequestMethod.***GET***)

**public** String homeAdd(Model model) {

model.addAttribute("mesaj", "Eklendi");

**return** "home";

}

@RequestMapping(value = "/delete", method = RequestMethod.***GET***)

**public** String homeDelete(Model model) {

model.addAttribute("mesaj", "Silindi");

**return** "home";

}

@RequestMapping(value = "/update", method = RequestMethod.***GET***)

**public** String homeUpdate(Model model) {

model.addAttribute("mesaj", "Guncellendi");

**return** "home";

}

}

**index.jsp**

<body>

<div class=*"container"*>

<h2>SECENEKLER</h2>

</div>

<div class=*"alert alert-success"*>

<a href=*"home/add"*>EKLE</a>

</div>

<div class=*"alert alert-warning"*>

<a href=*"home/delete"*>SIL</a>

</div>

<div class=*"alert alert-info"*>

<a href=*"home/update"*>GUNCELLE</a>

</div>

</body>

**home.jsp**

<body>

<div class=*"container"*>

<h2>${mesaj}</h2>

<div class=*"alert alert-warning"*>

<strong>${mesaj}</strong>

</div>

</div>

</body>

**Spring MVC Form**

com.project.model altına bir tane User classı oluşturduk.Ve ilk aşağıdaki ilk home viewinde o userin bilgilerini yazdırıp sonuc viewinde ise yazdırdıklarımızı görücez.Eğer istersek siteye girildiğinde ilk açılacak sayfayıda web.xml’de tanımlayabiliriz.

@Controller

**public** **class** HomeController {

@RequestMapping(value = "/home", method = RequestMethod.***GET***)

**public** ModelAndView form() {

ModelAndView modelAndView = **new** ModelAndView("home");

modelAndView.addObject(**new** User());

**return** modelAndView;

}

@RequestMapping(value = "/sonuc", method = RequestMethod.***POST***)

**public** ModelAndView sonuc(@ModelAttribute User user) {

ModelAndView modelAndView = **new** ModelAndView("sonuc");

modelAndView.addObject("user.username", user.getUsername());

modelAndView.addObject("user.email", user.getEmail());

**return** modelAndView;

}

}

**home.jsp**

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

<%@ page session=*"false"*%>

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

//Urinin içerisinde formları kullanacağımızı belirttik.Ve aşağıda daha güzel kullanmak için bir tane takma ad(prefix) koyduk. (<form:form) //

<!DOCTYPE html>

<html>

<head>

<title>Home</title>

</head>

<body>

<form:form action=*"sonuc"* modelAttribute=*"user"*> //Buradaki sonuc post edilen yani bilgilerin gideceği metodun url’s,,user ise get ile buraya gönderilen metodun içinde ki model.addAttribute(“user”,new User());

<table>

<tr>

<td>Username=</td>

<td><form:input path=*"username"*/></td>

</tr>

<tr>

<td>Email=</td>

<td><form:input path=*"email"*/></td>

</tr>

<tr>

<td></td>

<td><input type=*"submit"* value=*"gonder"*></td>

</tr>

</table>

</form:form>

</body>

</html>

**sonuc.jsp**

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

<%@ page session=*"false"*%>

<html>

<head>

<title>Sonuc Sayfasi</title>

</head>

<body>

<table>

<tr>

<td>First Name=</td>

<td>${user.username}</td>

</tr>

<tr>

<td>Email=</td>

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

</table>

</body>

</html>

**User class**

**package** com.project.model;

**public** **class** User {

**private** String username;

**private** String email;

**public** User() {

}

**public** String getUsername() {

**return** username;

}

**public** **void** setUsername(String username) {

**this**.username = username;

}

**public** String getEmail() {

**return** email;

}

**public** **void** setEmail(String email) {

**this**.email = email;

}

}

**Spring MVC Validation Form**

## Burada da yukarıdakinin aynısını yapcağız fakat girilen bilgileri kapsülleyeceğiz.Fakat bu validator işlemini hibernate yapıyor(User classına bak anlarsın).O yüzden projeye mvn repository’den [Hibernate Validator Engine Relocation Artifact](https://mvnrepository.com/artifact/org.hibernate/hibernate-validator) adında bir dependency eklememiz gerekiyor.

Bu aşağıdaki initbinderin amacı şudur diyelim bir form yaptık ve valid ile onu kapsülledik,İsim boş geçilemez soyisim boş geçilemez dedik ama boşluk girince java orayı boş geçmedi sayıyor.Yani boş geçilemez alanda isim soyismine boşluk(space) basan biri validden kurtuluyor işte controller sınıfımızın en üstüne bu üstteki kodu yazarsak artık boşluğu bir karakterden saymayacak.

----------------------

@InitBinder

**public** **void** initBinder(WebDataBinder dataBinder) {

StringTrimmerEditor stringTrimmerEditor = **new** StringTrimmerEditor(**true**);

dataBinder.registerCustomEditor(String.**class**, stringTrimmerEditor);

}

----------------------

@Controller

**public** **class** ValidationController {

@RequestMapping(value = "/validationForm", method = RequestMethod.***GET***)

**public** ModelAndView validationForm() {

ModelAndView modelAndView = **new** ModelAndView("validationForm");

modelAndView.addObject(**new** User());

**return** modelAndView;

}

@RequestMapping(value = "/successForm", method = RequestMethod.***POST***)

**public** ModelAndView successForm(@Valid @ModelAttribute(“user”)User user, BindingResult result) {

ModelAndView modelAndView = **new** ModelAndView();

**if**(result.hasErrors()) {

modelAndView.setViewName("validationForm");

**return** modelAndView;

}

**else** {

modelAndView.setViewName("successForm");

**return** modelAndView;

} } }

validationForm.jsp

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

<%@ page session=*"false"* %>

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

<html>

<head>

<title>Validation Form</title>

</head>

<body>

<form:form action=*"successForm"* modelAttribute=*"user"*>

<table>

<tr>

<td>Username=</td>

<td><form:input path=*"username"*/><form:errors path=*"username"*></form:errors></td>

</tr>

<tr>

<td>Email=</td>

<td><form:input path=*"email"*/><form:errors path=*"email"*></form:errors></td>

</tr>

<tr>

<td>Salary=</td>

<td><form:input path=*"salary"*/><form:errors path=*"salary"*></form:errors></td>

</tr>

<tr>

<td>Password=</td>

<td><form:password path=*"password"*/><form:errors path=*"password"*></form:errors></td>

</tr>

<tr>

<td></td>

<td><input type=*"submit"* value=*"GÖNDER"*></td>

</tr>

</table>

</form:form>

</body>

</html>

**successForm.jsp**

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

<%@ page session=*"false"*%>

<html>

<head>

<title>Home</title>

</head>

<body>

<table>

<tr>

<td>Username=</td>

<td>${user.username}</td>

</tr>

<tr>

<td>Email=</td>

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

</tr>

<tr>

<td>Salary=</td>

<td>${user.salary}</td>

</tr>

<tr>

<td>Password=</td>

<td>${user.password}</td>

</tr>

</table>

</body>

</html>

**User class**

**public** **class** User {

@~~NotEmpty~~(~~message~~ = "Bu alan bos gecilemez")

@Size(min = 5,max = 10,message = "Minimum 5,Max 10 karakter olmalidir")

**private** String username;

**private** String email;

@NotNull

@NumberFormat(style = Style.***NUMBER***)

@Max(10000)

@Min(1000)

**private** String salary;

@~~NotEmpty~~(~~message~~ = "Bu alan bos gecilemez")

@Size(min = 5,max = 10,message = "Minimum 5,Max 10 karakter olmalidir")

**private** String password;

**public** User() {

}

**public** String getUsername() {

**return** username;

}

**public** **void** setUsername(String username) {

**this**.username = username;

}

**public** String getEmail() {

**return** email;

}

**public** **void** setEmail(String email) {

**this**.email = email;

}

**public** String getSalary() {

**return** salary;

}

**public** **void** setSalary(String salary) {

**this**.salary = salary;

}

**public** String getPassword() {

**return** password;

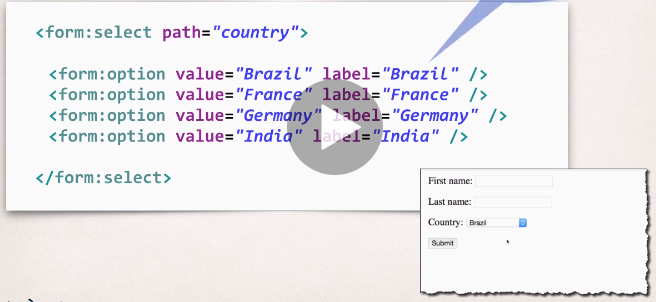
}

**public** **void** setPassword(String password) {

**this**.password = password;

}

}

**DropDown List Kullanımı(Açılır Liste)**

Bu kodu forma yazıyoruz.Daha sonra verileri controllerdan almak için aşağıdaki kodu yazıyoruz.Fakat bu countryi kullanacağımız entitynin içine path=”” içindeki gibi değişken oluşturmamız şart.

@RequestMapping(value = "/sonuc", method = RequestMethod.***POST***)

**public** ModelAndView sonuc(@ModelAttribute User user) {

ModelAndView modelAndView = **new** ModelAndView("sonuc"); //sonuç adında bi jsp’ye gidecek

String sehir = user.getCountry();

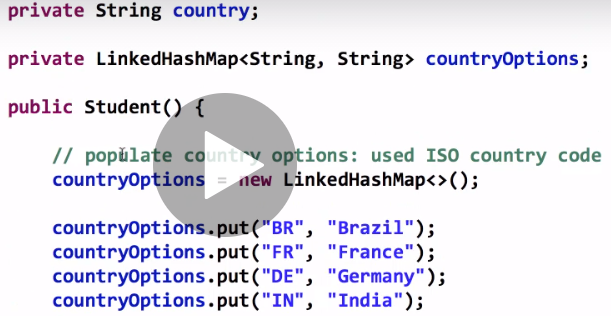
Syso(sehir);

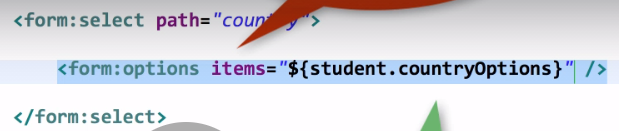
**return** modelAndView;

}

Yukarıdaki işlemde country şehirlerini html üzerinden verdik fakat bunu javadan da verebilirdik oda şöyle: İlk student classına bir hash map yada list oluşturuyoruz.Daha sonra 2.resimde ki kodu html form etiketinin içine koyuyoruz. Fakat bu countryi kullanacağımız entitynin içine path=”” içindeki gibi değişken oluşturmamız şart.

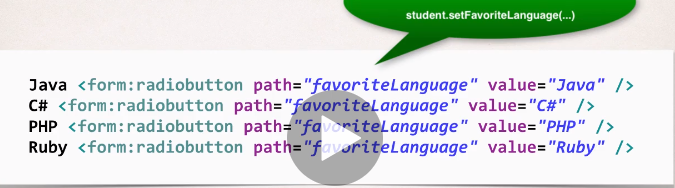
1.Resim



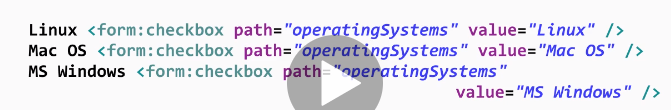
2.Resim

**RadioButton Kullanımı**

Üstteki list ile aynı mantık formu aşağıdaki şekilde dolduruyoruz.Ve bunu kullanacağımız entity’nin içinde path=”” içindekiyle aynı isimden değişken olmalı.Yine bu radiobuttonun değerlerini jsp’ten değilde javadan vermek istersek üstteki dropdown listteki gibi verebiliriz.

****

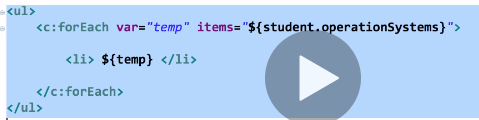
**Checkbox Kullanımı**

Aşağıdakikodu html dosyasına yazıyoruz

Alttaki resimde olduğu gibi “Array” oluşturuyoruz.Çünkü CheckBox’ta birden fazla seçim olabilir



Bu alttaki ise çıktıları gönderdiğimiz html kodu



**initBinder Kullanımı**

@InitBinder

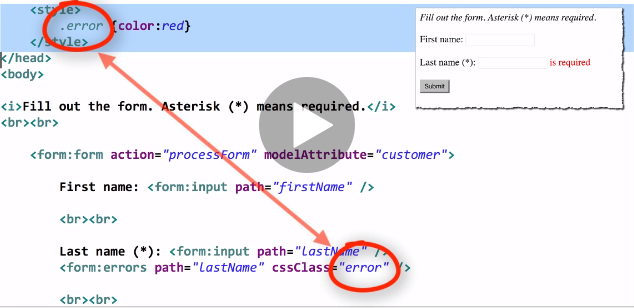
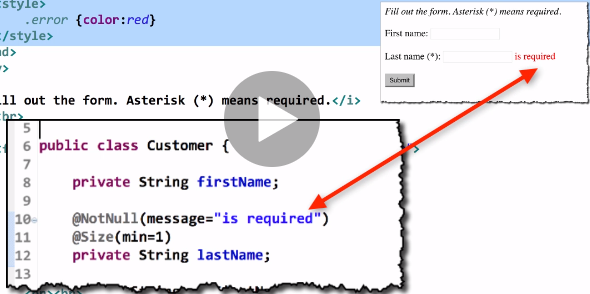
**public** **void** initBinder(WebDataBinder dataBinder) {

StringTrimmerEditor stringTrimmerEditor = **new** StringTrimmerEditor(**true**);

dataBinder.registerCustomEditor(String.**class**, stringTrimmerEditor);

}

Bu initbinderin amacı şudur diyelim bir form yaptık ve valid ile onu kapsülledik,İsim boş geçilemez soyisim boş geçilemez dedik ama boşluk girince java orayı boş geçmedi sayıyor.Yani boş geçilemez alanda isim soyismine boşluk(space) basan biri validden kurtuluyor işte controller sınıfımızın en üstüne bu üstteki kodu yazarsak artık boşluğu bir karakterden saymayacak.

**Error Mesagges**