# BÀI TẬP TUẦN 05\_06\_07

***Chương 5: Spring MVC***

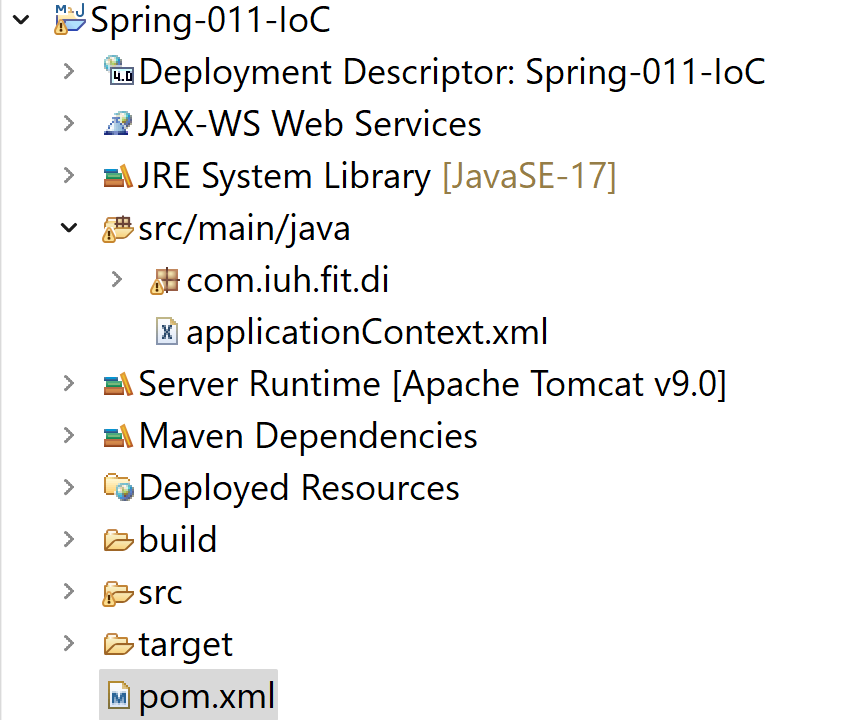
*Phần 1. Spring Core*

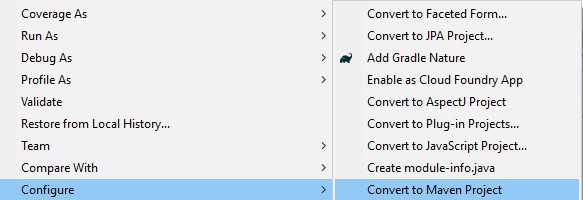
*Lưu ý:*

Sử dụng Maven Project

- Các dependencies tra cứu trong <https://mvnrepository.com/>

Project của Eclipse nhấn chuột phải 🡪 Chọn Configure 🡪 Chọn Convert to Maven Project





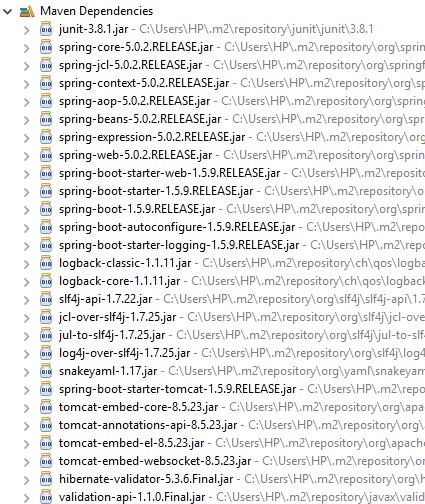
Các thay đổi về dependencies sẽ thay đổi trong file pom.xml

Hoặc có thể thay đổi trực tiếp trong file pom.xml.

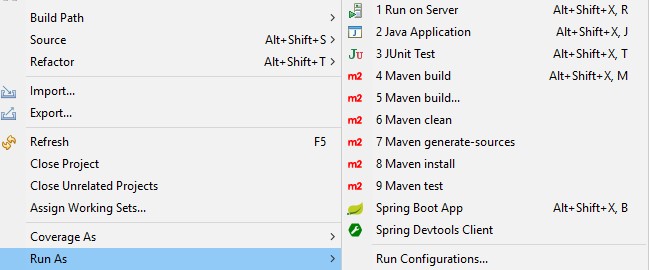
Các dependencies tra cứu trong <https://mvnrepository.com/>, khi tra cứu lưu ý các groupId, artifactId, version cần dùng.



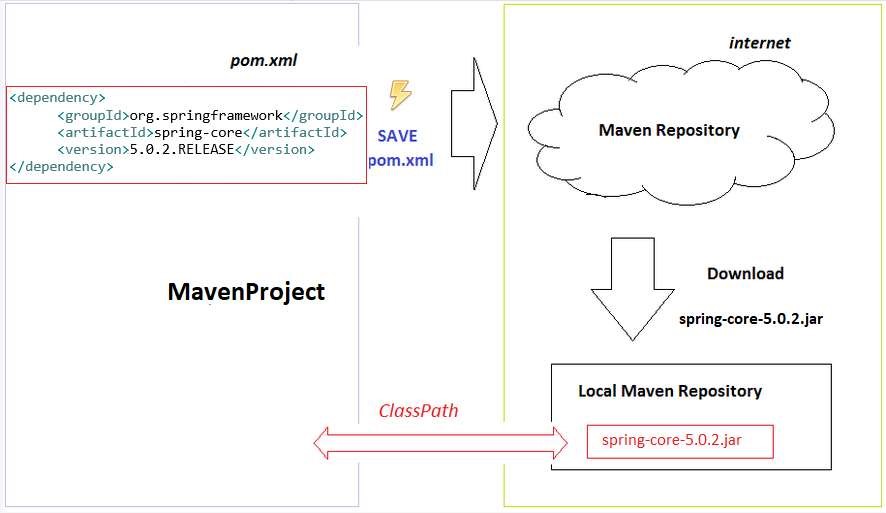
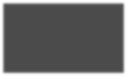
Các Maven dependencies



Tạo file jar từ Maven: **SpringMVCMultiLanguages0.0.1-SNAPSHOT.jar**



*Nguyên tắc hoạt động của Maven*



* Khai báo trên tập tin ***pom.xml*** các dependencies phụ thuộc vào các thư viện có version rõ ràng.
* Khi lưu file pom.xml, Maven kiểm tra các thư viện này đã tồn tại ở máy tính (Local Repository) chưa. Nếu chưa Maven sẽ download về từ kho dữ liệu từ Internet (Repository).
* Maven sẽ tự động khai báo CLASSPATH cho Project tới vị trí file jar được download.

**IoC Container trong Spring** là core của Spring Framework. IoC Container, IoC tạo ra các đối tượng, nối các đối tượng lại với nhau, cấu hình, và quản lý life cycle (từ khi tạo ra đến khi bị hủy). IoC Container sử dụng DI để quản lý các thành phần tạo nên một ứng dụng. Các đối tượng này được gọi là Spring Bean.

IoC Container được cung cấp thông tin từ các tập tin XML.

Hai loại IoC containers:

1. **BeanFactory**
2. **ApplicationContext**

* ApplicationContext (*org.springframework.context.ApplicationContext* interface) cũng tương tự như BeanFactory (*org.springframework.beans.factory.BeanFactory*)*,* sử dụng để mô tả Spring Container. ApplicationContext được xây dựng trên interface BeanFactory.
* BeanFactory cung cấp các chức năng cơ bản trong khi ApplicationContext cung cấp các tính năng mở rộng hơn cho các ứng dụng Spring nên ApplicationContext phù hợp với các ứng dụng J2EE hơn. Lưu ý ApplicationContext bao gồm tất cả các chức năng của BeanFactory.
* Đối tượng của interface ApplicationContext như là một khung chứa của Spring để gọi đến đối tượng cần lấy bằng cách sử dụng phương thức getBean().

|  |
| --- |
| ApplicationContext context = **new** ClassPathXmlApplicationContext("Beans.xml");  Student obj = (Student) context.getBean("studentbean"); obj.displayInfo();  Resource resource=**new** ClassPathResource("Beans.xml");  BeanFactory factory=**new** XmlBeanFactory(resource);  Student obj =(Student)factory.getBean("studentbean");  obj.displayInfo(); |

Dependency Injection:

* Các module không giao tiếp trực tiếp với nhau, mà thông qua interface**.** Module cấp thấp sẽ implement interface, module cấp cao sẽ gọi module cấp thấp.

Để giao tiếp với database, sử dụng interface *IDatabase*, các module cấp thấp là *XMLDatabase*, *SQLDatabase*. Module cấp cao là *CustomerBusiness* sẽ sử dụng interface *IDatabase*.

* Việc khởi tạo các module cấp thấp sẽ do DI Container thực hiện**.**

Trong module *CustomerBusiness*, không khởi tạo IDatabase db = new XMLDatabase(), việc này sẽ do DI Container thực hiện. Module CustomerBusiness sẽ không biết gì về module XMLDatabase hay SQLDatabase.

* Các Module kết hợp với các interface cụ thể sẽ được config trong code hoặc trong file XML**.**

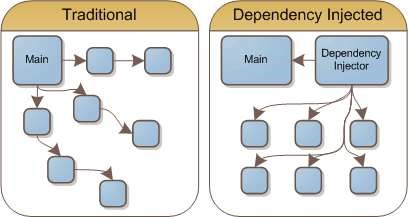
* DI được dùng để làm giảm sự phụ thuộc giữa các module**,** dễ dàng trong việc thay đổi module, bảo trì code và testing.

Thực hiện Dependency Injection theo 3 cách:

1. **Constructor Injection**: Các dependency sẽ được container truyền/**inject vào** 1 class thông qua constructor của class.
2. **Setter Injection**: Các dependency sẽ được truyền vào 1 class thông qua các hàm setter.
3. **Interface Injection**: Class cần truyền/inject sẽ implement 1 interface. Interface này có chứa 1 hàm tên *Inject*. Container sẽ injection dependency vào 1 class thông qua việc gọi hàm *Inject* của interface.

## Bài 1. Spring DI

IoC trong Spring và phương pháp tiếp cận truyền thống.



Một nhân viên (Employee) trong công ty sẽ có các thông tin như họ tên, địa chỉ …. Thiết kế đối tượng Employee và Address dùng Dependency Injection (DI) trong Spring để loại bỏ sự phụ thuộc giữa các mã chương trình.

B1. Tạo Project Java

B2. Configure Project 🡪 Convert to Maven Project

B3. Cấu hình các dependency của Spring trong pom.xml



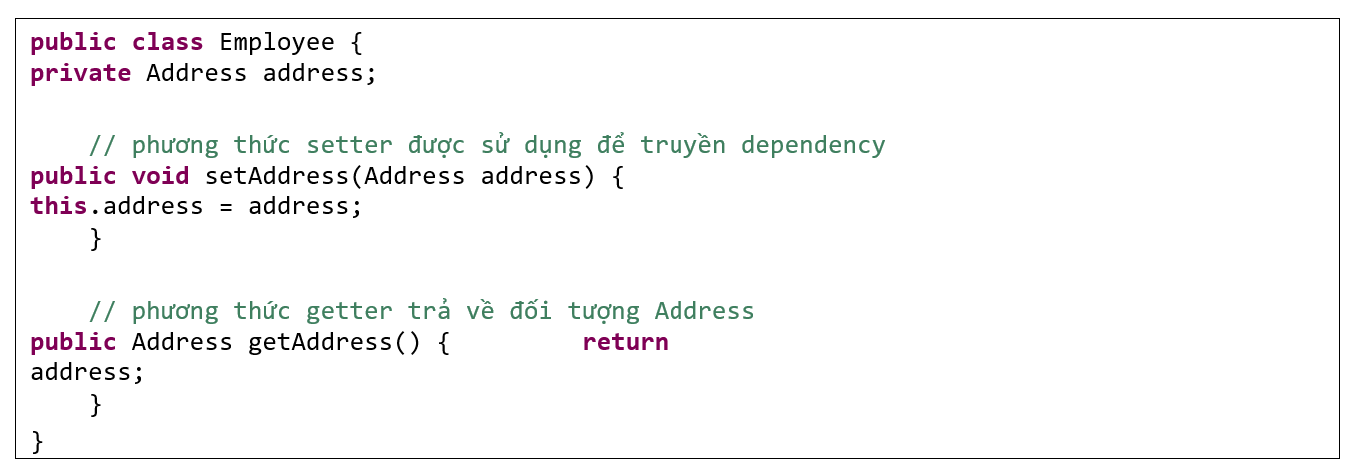
B4. Tạo lớp Address

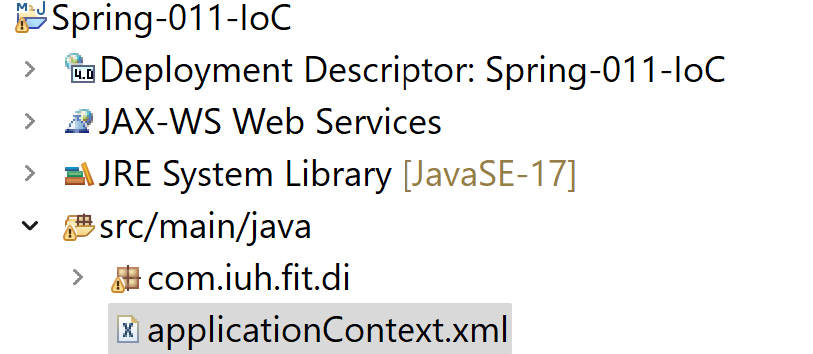


B5. Tạo lớp Employee. Làm giảm sự phụ thuộc giữa Employee và Address, tạo lớp Address riêng và truyền/inject vào Employee thông qua Constructor (hoặc phương thức Setter). *Dùng Contructor*

**

*Dùng Setter:*



B6. Cấu hình file bean *applicationContext.xml.* File lưu trong thư mục **src/main/java**



B7. Test chương trình : Tạo Class Java Demobean chứa void main()



Run Project: R\_Click project 🡪 Run As 🡪 Java Application

## Bài 2. Spring Ioc\_DI

Tình huống chương trình có đối tượng Coach(Huấn luyện viên), với mỗi môn thể thao sẽ có các huấn luyện viên khác nhau như BaseballCoach, TennisCoach, CricketCoach. Thiết kế các đối tượng Coach, BasebalCoach, CriketCoach,… để loại bỏ sự phụ thuộc lẫn nhau

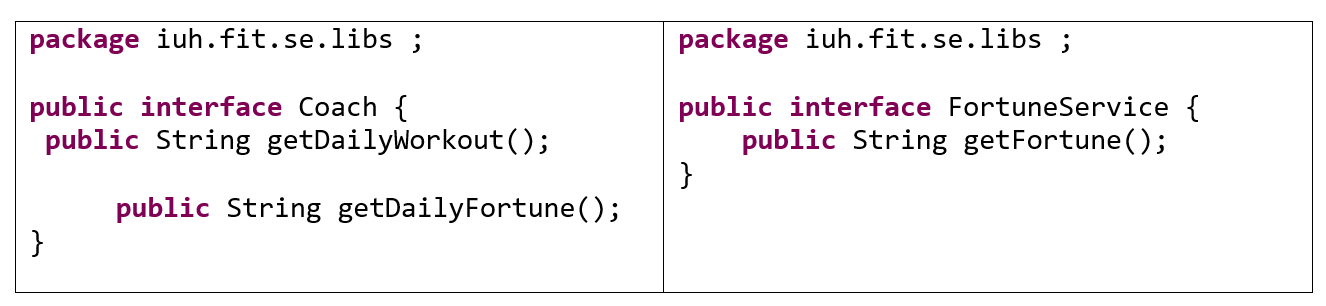
B1. Tạo Project Java

B2. Configure Project 🡪 Convert to Maven Project

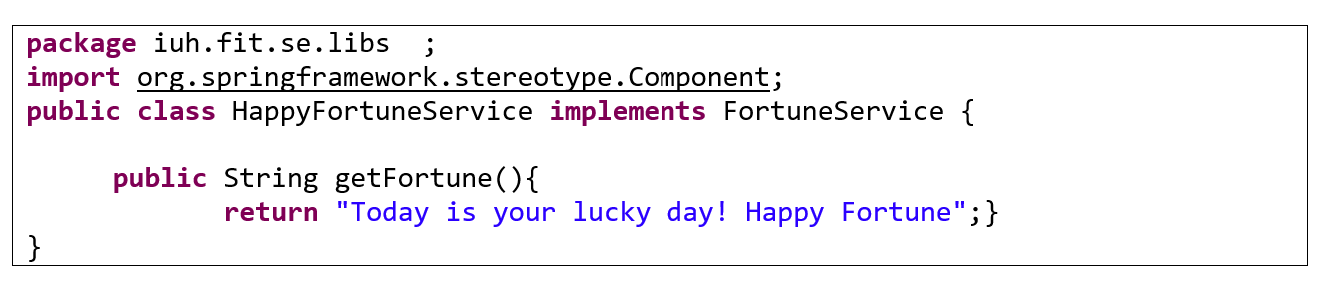
B3. Cấu hình các dependency của Spring trong pom.xml tương tự như bài 1

B4: Tạo backage iuh.fit.se.libs ; chứa các Class java

Tạo Class Interface: Coach.java, FortuneService.java



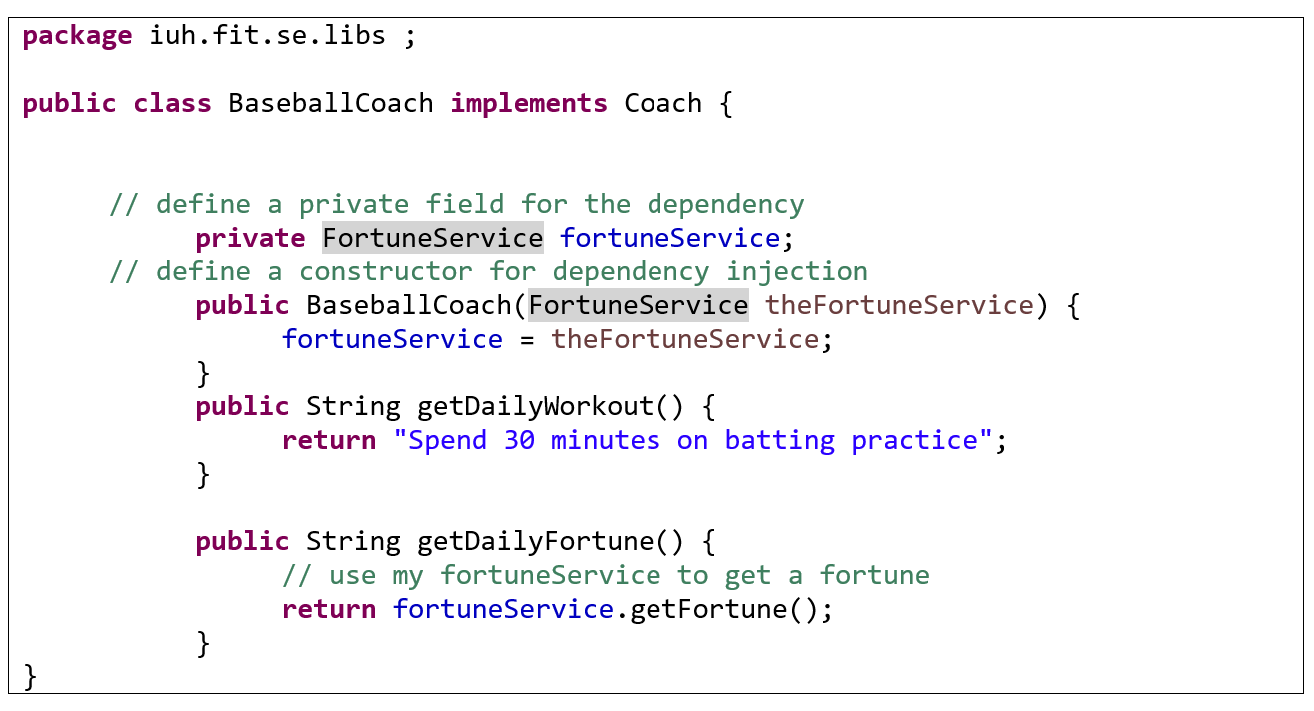
B4: Tạo Class HappyFortuneServiceimplement tới class Fortune chứa các phương thức hoạt động cho các class BaseballCoach, TennisCoach, CriketCoach phía sau:



B5: Tạo class BaseballCoach.java implement class Coach

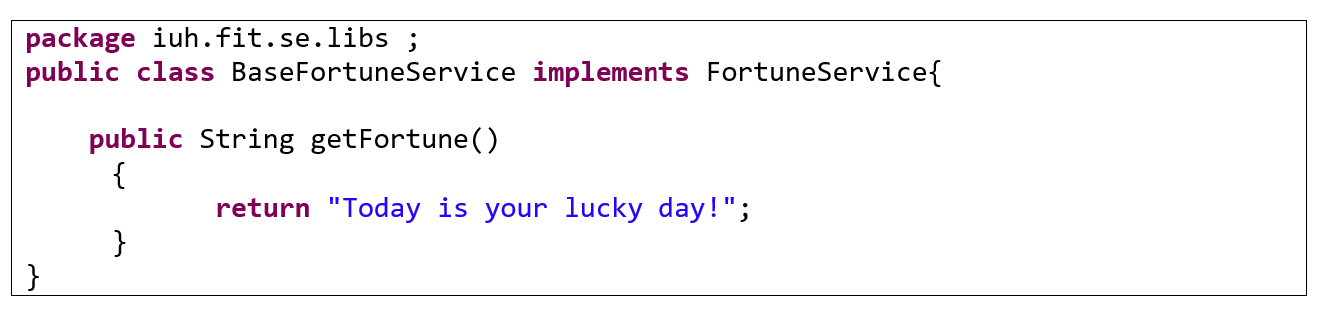
Làm giảm sự phụ thuộc giữa BaseballCoach và Coach, tạo lớp BaseballCoach riêng và truyền/inject vào Coach thông qua Constructor.

Trong BaseballCoach có thêm thuộc tính của FortuneService (interface) khác,

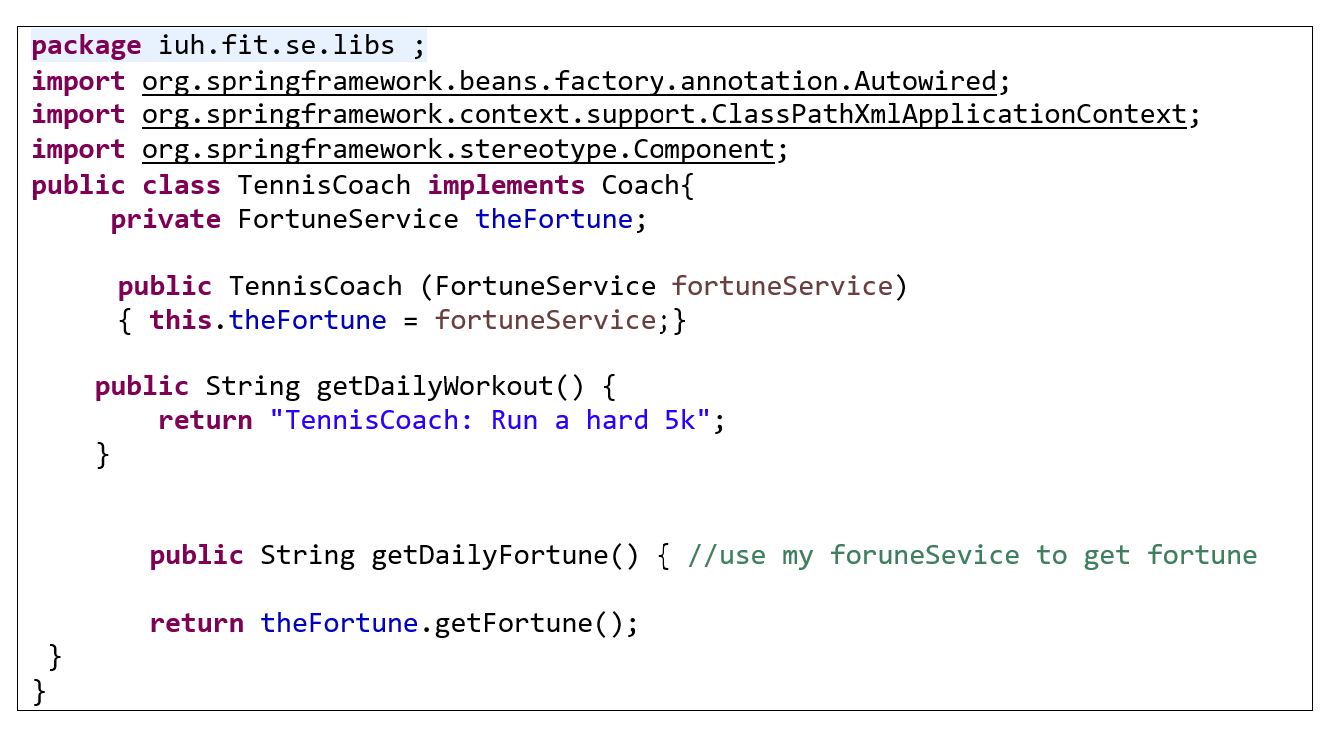


*Depenence Injection Contructor*

Tạo Class BaseFortuneService

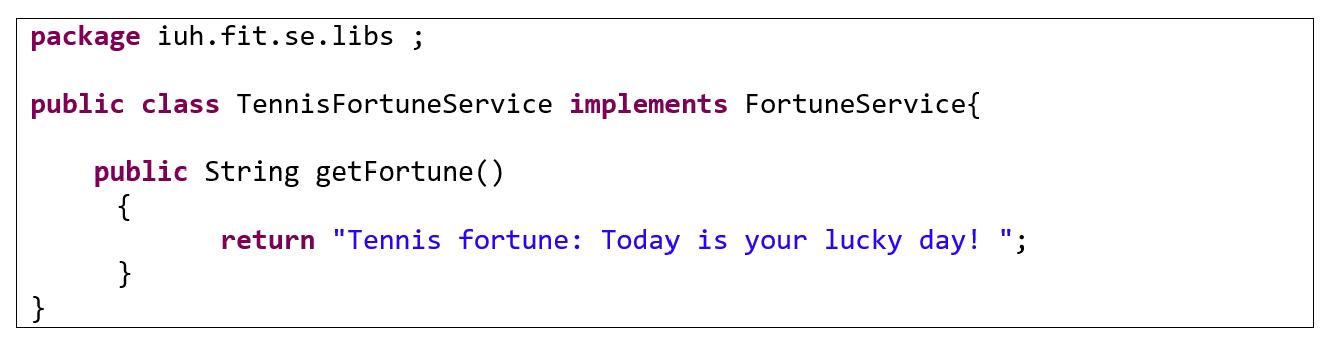


B6: Tương tự tạo thêm class TennisCoach



*Depenence Injection Contructor*

Tạo TennisFortuneService.java



B7: Tạo Criket Coach.java Class này thực hiện DI bằng Setter



*Depenence Injection Contructor*

B8: Cấu hình file bean *applicationContext.xml.* File lưu trong thư mục **src/main/java**



B9: Tạo pagekage iuh.fit.se.apps chứa class SpringHelloApp. java có hàm void main() để chạy chương trình, test các phương thức và các bean đã tạo trong applicationContext.xml



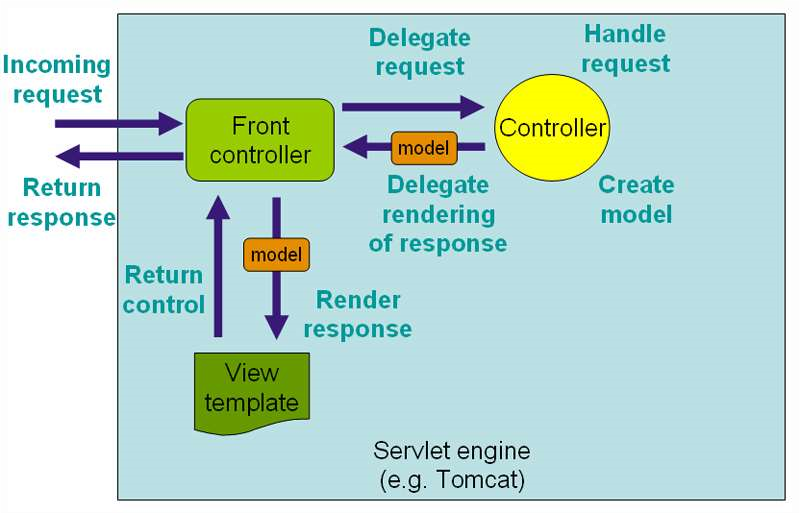
*Phần 2. Spring MVC*

0. Mô hình Spring MVC trong môi trường Web

 Front Controller tiếp cận tất cả các request gửi tới, chuyển tới Controller cần xử lý.

 Controller trả về các Model cần thiết cho Front Controller và chọn View phù hợp.

 Front Controller cũng trả về kết quả cho người dùng.



Với Java, *Front Controller* thường là đối tượng implement interface **javax.servlet.Servlet.** Trong Spring MVC thì Front Controller là **org.springframework.web.servlet.DispatcherServlet.**

**Các bước thực hiện:**

B0: Khai báo Dependencies trong pom.xml cho project Maven

B1. Request tới ứng dụng Web, *DispacherServlet*là đối tượng nhận tất cả các request.

B2. Tìm và chuyển hướng request tới Request Handler phù hợp là các Controllers trong ứng dụng Web.

B3. Các Controller xử lý request yêu cầu.

B4. Chuẩn bị Model và chọn View hiển thị.

B5. Trả về kết quả xử lý request cho *DispacherServlet***.**

B6. *DispacherServlet*sẽ gọi View Template phù hợp để xử lý việc hiển thị trên giao diện bằng cách sử dụng Model.

B7. View Template trả kết quả về cho *DispacherServlet***.**

B8. Trả Response.

Thực thi 1 project Application Web Spring MVC cần phải cấu hình một số các thành phần cho project:

#### Cấu hình Front Controller *org.springframework.web.servlet.DispatcherServlet*: thường để trong web.xml

|  |
| --- |
| <?xml version=*"1.0"* encoding=*"UTF-8"*?>  <web-app version=*"3.1"*  xmlns=*"http://xmlns.jcp.org/xml/ns/javaee"*  xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"*  xsi:schemaLocation=*"http://xmlns.jcp.org/xml/ns/javaee*  *http://xmlns.jcp.org/xml/ns/javaee/web-app\_3\_1.xsd"*>  <display-name>spring-mvc-demo</display-name>  <absolute-ordering />  <!-- 1. Configure Spring dispatcher servlet -->  <servlet>  <servlet-name>dispatcher</servlet-name>  <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>  <init-param>  <param-name>contextConfigLocation</param-name>  <param-value>/WEB-INF/spring-mvc-context.xml</param-value>  </init-param>  <load-on-startup>1</load-on-startup>   * Đường dẫn tới file xml * Servlet-name: giống nhau * Servlet-class: DispatcherServlet * url-pattern: / mapping request từ client   </servlet>  <!-- 2. setup url mapping for dispatcher servlet -->  <servlet-mapping>  <servlet-name>dispatcher</servlet-name>  <url-pattern>/</url-pattern>  </servlet-mapping>  <session-config>  <session-timeout>30</session-timeout>  </session-config>  <jsp-config>  <jsp-property-group>  <url-pattern>\*.jsp</url-pattern>  <page-encoding>UTF-8</page-encoding>  </jsp-property-group>  </jsp-config>  </web-app> |

Mặc định khi *DispatcherServlet*được khởi tạo, đối tượng này sẽ khởi tạo một đối tượng *org.springframework.web.context.WebApplicationContext* với hiện thực là đối tượng *org.springframework.web.context.support.XmlWebApplicationContext*.

Đối tượng *XmlWebApplicationContext*chứa cấu hình tất cả các beans mà ứng dụng định nghĩa trong khung chứa (container) của Spring. Đối tượng *XmlWebApplicationContext* này sẽ dùng một tập tin cấu hình của Spring với tên gọi mặc định là ***[tên-servlet]-servlet.xml*** nằm trong thư mục */src/webapp/WEB-INF* với ứng dụng SpringMVC tạo bằng Dynamic Web Project.

#### Khai báo các beans trong root-servlet

*Cấu hình root Servlet*

Trường hợp định nghĩa nhiều Servlet trong một tập tin *web.xml*, mỗi servlet có thể được khởi tạo, ánh xạ đến các URL khác nhau. Dẫn đến trường hợp này sẽ có nhiều Spring container tương ứng với từng Servlet. Trong trường hợp đó, một số định nghĩa bean có thể lặp đi lặp lại trong nhiều Spring container.

Đối tượng *org.springframework.web.context.ContextLoaderListener* được tạo ra để giải quyết vấn đề lặp đi lặp lại này bằng cách tạo ra một root *org.springframework.web.context.WebApplicationContext*sử dụng chung cho tất cả các Servlet. *ContextLoaderListener* sẽ sử dụng tập tin được định nghĩa trong *contextConfigLocation* của tag *<context-param>.*

Thường trong tập tin *root-context.xml*, định nghĩa các bean, các thuộc tính dùng chung giữa các container của Spring, mỗi container Spring trong mỗi Servlet sẽ sử dụng các bean.

***[tên-servlet]-servlet.xml***

|  |
| --- |
| <?xml version=*"1.0"* encoding=*"UTF-8"*?>  <beans xmlns=*"http://www.springframework.org/schema/beans"*  xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"*  xmlns:context=*"http://www.springframework.org/schema/context"*  xmlns:mvc=*"http://www.springframework.org/schema/mvc"*  xsi:schemaLocation=*"*  *http://www.springframework.org/schema/beans*  *http://www.springframework.org/schema/beans/spring-beans.xsd*  *http://www.springframework.org/schema/context*  *http://www.springframework.org/schema/context/spring-context.xsd*  *http://www.springframework.org/schema/mvc*  *http://www.springframework.org/schema/mvc/spring-mvc.xsd"*>  <!-- 3: Add support for component scanning : các package chứa các thành phần xử ký -->  <context:component-scan base-package=*"iuh.fit.se.springmvc.demo, package iuh.fit.se.springmvc.validation"* >  </context:component-scan>    <!-- 4: Add support for conversion, formatting and validation support -->  <!-- 5: Define Spring MVC view resolver: ViewResolver (xử lý View), các View là tập tin JSP được ánh xạ từ thư mục /WEB-INF/view/ -->  <bean  class=*"org.springframework.web.servlet.view.InternalResourceViewResolver"*>  <property name=*"prefix"* value=*"/WEB-INF/view/"* />  <property name=*"suffix"* value=*".jsp"* />  </bean>    <!—Khai báo các bean sẽ dùng trong Controller như: database,… -->  <bean id=*"messageSource"*  class=*"org.springframework.context.support.ResourceBundleMessageSource"*>  <property value=*"resources/messages"* name=*"basenames"* />  </bean>  </beans> |

*InternalResourceViewResolver* sẽ map tên View mà trả về trong Controller với một tên file nằm trong một thư mục cụ thể.

Khai báo bean *InternalResourceViewResolver* sẽ map tất cả các tên View với các file có phần mở rộng .jsp nằm trong thư mục /src/main/WEB-INF/view để render giao diện.

* Định nghĩa request “/” với tên View trả về trong method home() là “home”, tương ứng với file /src/main/WEB-INF/view/home.jsp.
* Định nghĩa request “/login” với tên View trả về trong method loginPage () là “login”, tương ứng với file /src/main/WEB-INF/view/login/login.jsp.
* Định nghĩa request “/home” với tên View trả về trong method login() là “user”, tương ứng với file /src/main/WEB-INF/view/home/user.jsp.

@Controller

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

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

**public** String home(Locale locale, Model model) {

***…***

**return** "home";

}

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

**public** String loginPage(Locale locale, Model model) {

**return** "login";

}

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

**public** String login(@Validated User user, Model model) {

model.addAttribute("userName", user.getUserName());

**return** "user";

}

}

Cách ánh xạ giữa View và tập tin View:

* home 🡪 home.jsp
* login 🡪 login.jsp
* user 🡪 user.jsp

#### Tạo Controller trong Spring MVC

* Annotation @Controller khai báo cùng với định nghĩa của lớp **XXXController**.
* Annotation **@RequestMapping** khai báo cùng với định nghĩa của phương thức xử lý nào đó **xulyData()**. Với annotation **@RequestMapping** có @RequestMapping(value = "/", method = RequestMethod.***GET***)
  + thuộc tính **value** để định nghĩa request URL
  + thuộc tính **method** để định nghĩa HTTP request method.

|  |
| --- |
| **package** iuh.fit.se.springmvc.demo;  **import** javax.servlet.http.HttpServletRequest;  **import** org.springframework.stereotype.Controller;  **import** org.springframework.ui.Model;  **import** org.springframework.web.bind.annotation.RequestMapping;  **import** org.springframework.web.bind.annotation.RequestParam;  @Controller  @RequestMapping("/hello")  *Client Request URL như sau:* [*http://localhost:8080/.../hello/showForm*](http://localhost:8080/.../hello/showForm)  *🡪helloworld-form.jsp*  [*http://localhost:8080/.../hello/*](http://localhost:8080/.../hello/)*processForm*  *🡪helloworld.jsp*  **public** **class** HelloWorldController {  @RequestMapping("/showForm")  **public** String showForm() {  **return** "helloworld-form";  }//    @RequestMapping("/processForm")  **public** String processForm() {  **return** "helloworld";  }// action nhận |

* Tạo Controller để kiểm soát handle request từ client gửi tới dạng QueryString:

/login?username=xxx&password=xxx

|  |
| --- |
| **import** javax.servlet.http.HttpServletRequest;    **import** org.springframework.stereotype.Controller;  **import** org.springframework.web.bind.annotation.RequestMapping;  @Controller  **public** **class** LoginController {  @RequestMapping(value = "/login")  **public** **void** login(HttpServletRequest request){  String username = request.getParameter("username");  String username = request.getParameter("username");  }  } |

Hoặc dùng @RequestParam annotation

|  |
| --- |
| **import** org.springframework.stereotype.Controller;  **import** org.springframework.web.bind.annotation.RequestMapping;  **import** org.springframework.web.bind.annotation.RequestParam;  @Controller  **public** **class** LoginController {    @RequestMapping(value = "/login")  **public** **void** login(@RequestParam String username, @RequestParam String password) {  …  }  } |

Nếu biến trong QueryString và biến phương thức khác tên thì thay đổi:

*public void login(@RequestParam(value = "username") String user, @RequestParam String password) {…}*

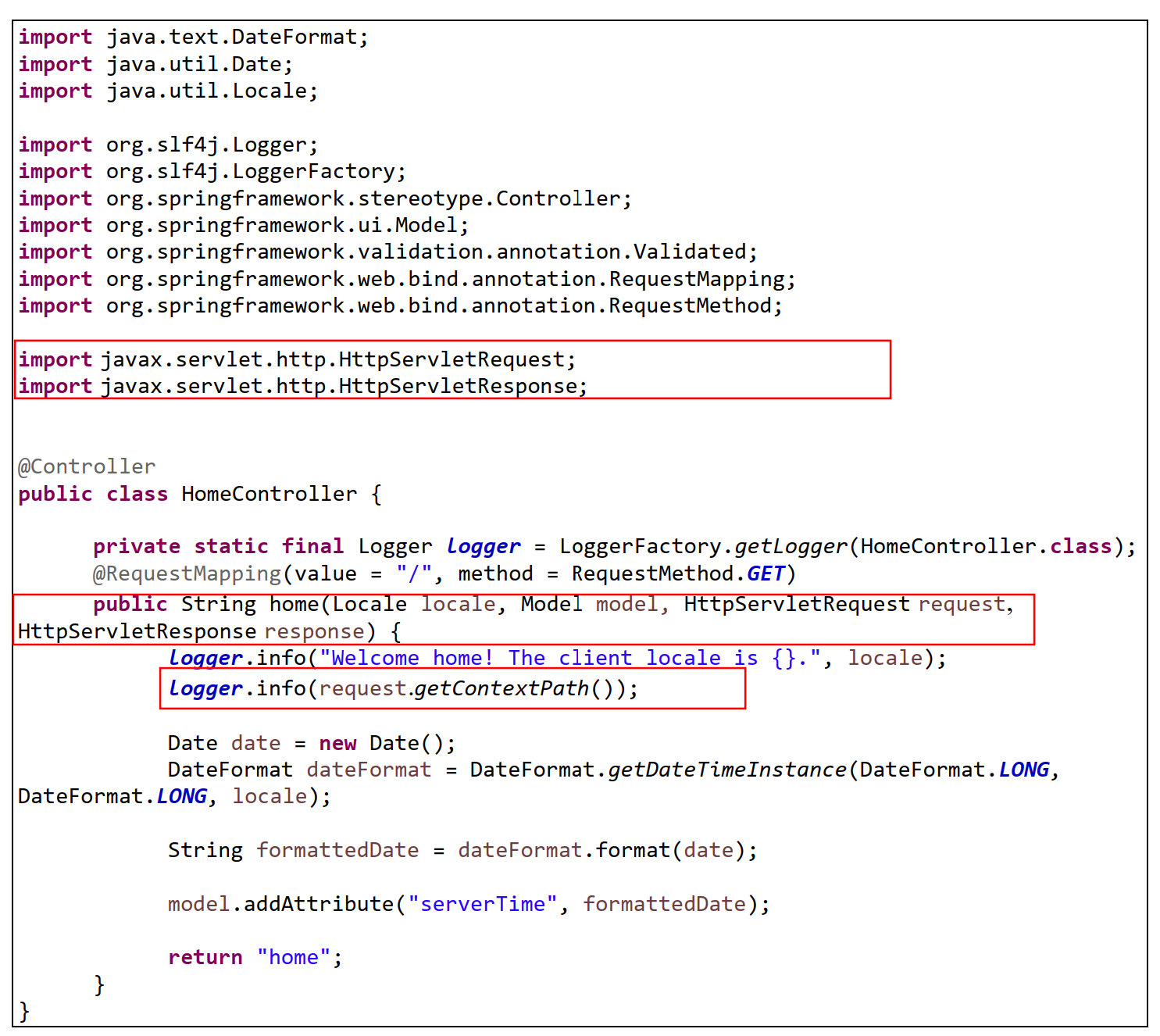
Nếu bắt buộc phải có giá trị (khác null):

*public void login(@RequestParam(value = "username", required = false) String user, @RequestParam String password) {…}*

Nếu muốn gán giá trị mặc định:

*public void login(@RequestParam(value = "username", required = false, defaultValue = "xxxx") String user, @RequestParam String password) {…}*

#### Đối tượng HttpServletRequest và HttpServletResponse trong Controller của Spring MVC



#### Truyền dữ liệu vào Model trong Controller (Spring MVC)

*Cách 1.* Dùng đối tượng *org.springframework.ui.Model*, và phương thức addAttribute(“Tên thuộc tính”, giá trị);

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

public String home(Locale locale, Model model) {

…

model.addAttribute("Tên thuộc tính", giá trị);

}

*Cách 2.* Dùng *java.util.Map*

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

public String home(Locale locale, Map<String, Object> model) {

…

model.put("Tên thuộc tính", giá trị);

}

*Cách 3.* Dùng *org.springframework.ui.ModelMap*

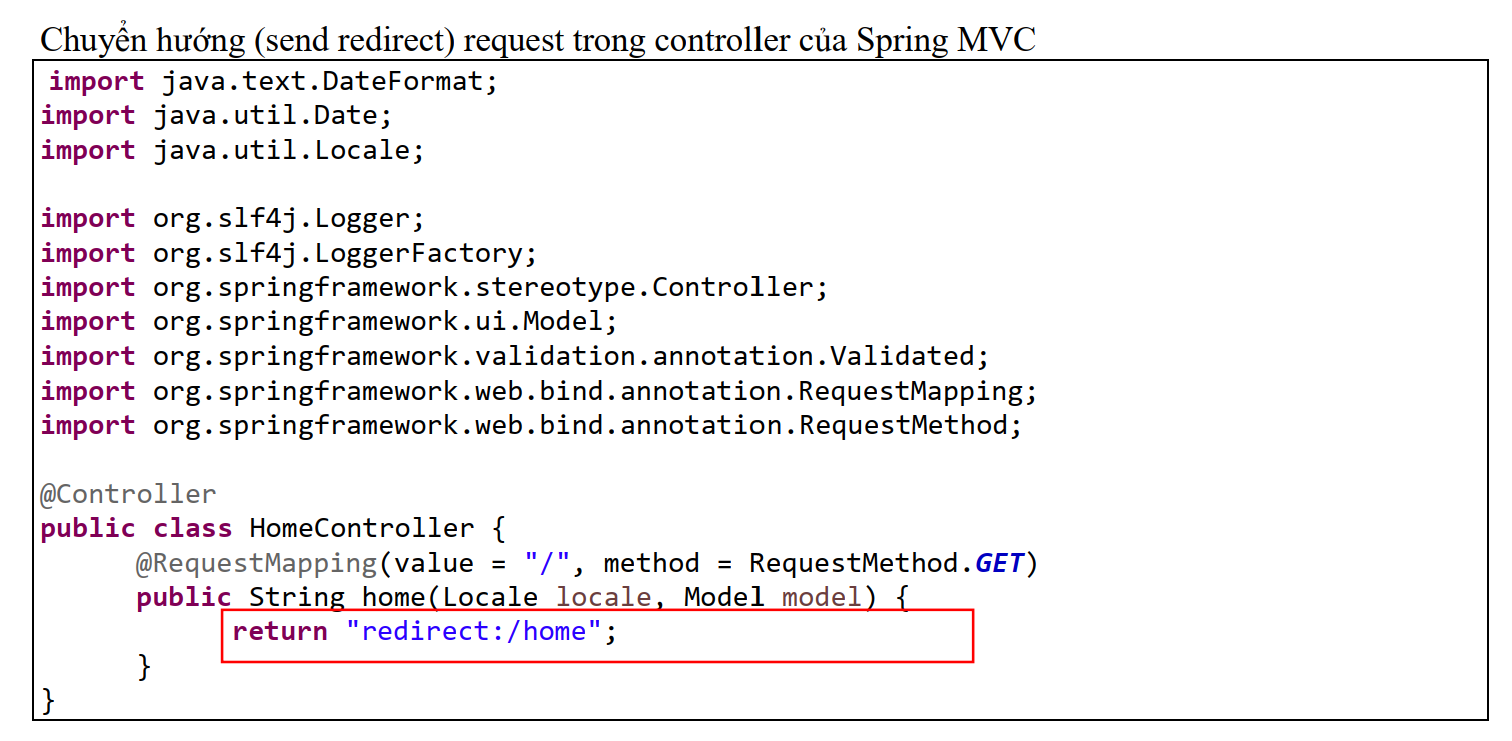
@RequestMapping(value = "/", method = RequestMethod.GET)

public String home(Locale locale, ModelMap model) {

…

model.addAttribute(“Tên thuộc tính”, giá trị)

}



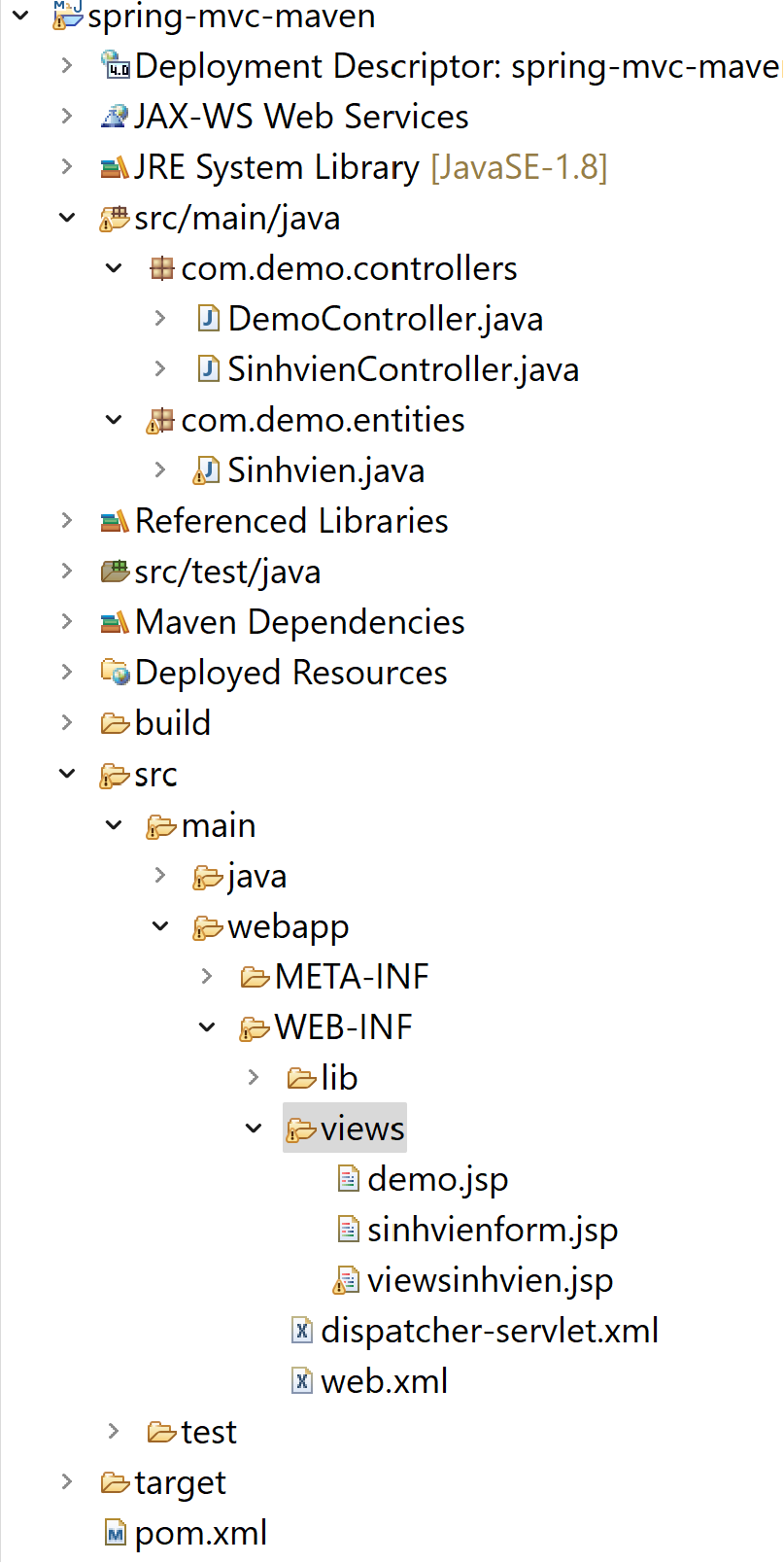
#### Đối tượng Enum: org.springframework.web.bind.annotation.RequestMethod gồm các request method

* + RequestMethod.GET
  + RequestMethod.HEAD
  + RequestMethod.POST
  + RequestMethod.PUT
  + RequestMethod.PATCH
  + RequestMethod.DELETE
  + RequestMethod.OPTIONS
  + RequestMethod.TRACE

## Bài 3: Spring MVC

Thực hiện kiểm soát dữ liệu truyền qua form dùng MVC

Thực hiện ứng dụng Spring MVC Sinhvien chứa form nhập thông tin sinh viên vào danh sách sinh viên



B1. Tạo *Dynamic Web Project*, khai báo các thông tin liên quan

B2. *pom.xml:*

Khai báo các dependencies, Maven 🡪 Update Project

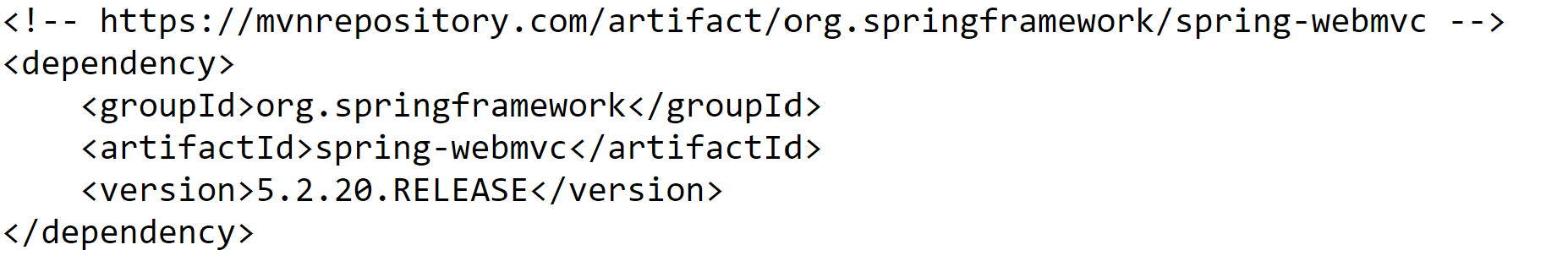
*Khai báo dependencies như bài 2 gồm các dependences cơ bản:*

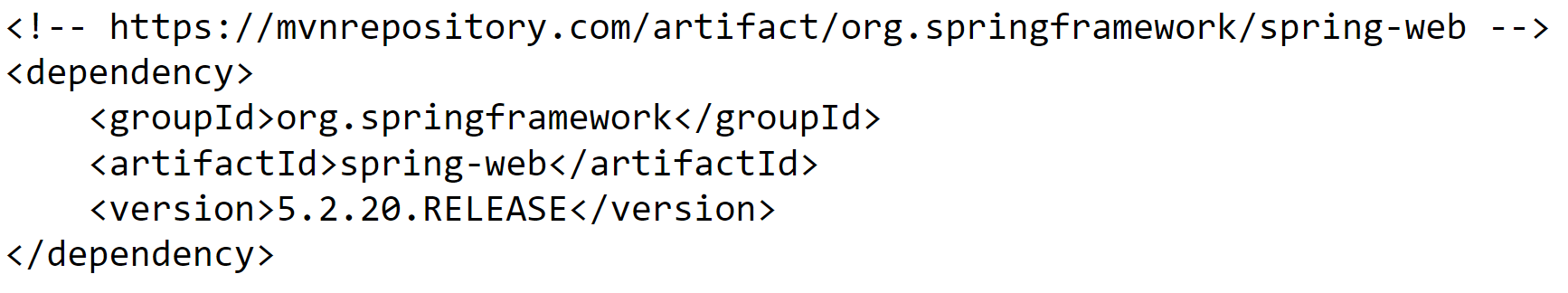
*/springframework/spring-core*

*/springframework/spring-beans*

*/springframework/spring-context*

*Thêm dependency cần thiết cho spring mvc, spring web*





B3: *web.xml*: Khai báo cấu hình cần thiết của ứng dụng Web *(Cấu hình 1. DispatcherServlet)*

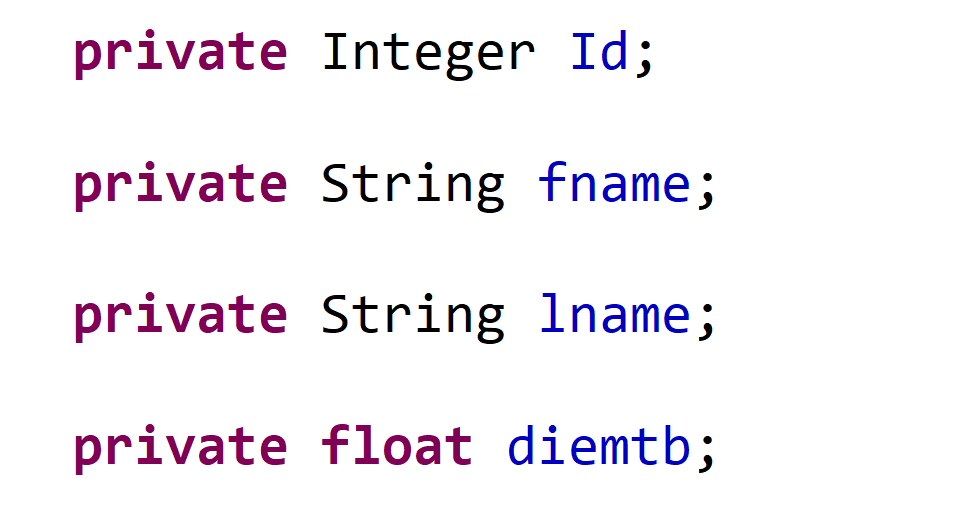
{1. Cấu hình Spring dispatcher servlet, 2. Mapping cho Spring dispatcher servlet }

B4: *WEB-INF/dispatcher-servlet.xml*:   
Khai báo cấu hình tối thiểu về Spring MVC *(Cấu hình 2. servlet, servlet mapping của project)*

{3. Scan-context, 4. Annotation validation, 5. ViewResolver : như bài 2}

B5: Thực hiện Code, tạo Model com.demo.entities.Sinhvien: Model

Các thuộc tính của Model Sinhvien, tạo contructor, get, set, toString() cho Model Sinhvien



B6: com.demo.controllers.SinhvienController: Controller bao gồm action hiển thị form, nhập thông tin sinh viên, hiển thị danh sách sinh viên



B7. Tạo view sinhvienform.jsp



B8. Tạo view viewsinhvien.jsp



R-Click Project 🡪 Run On Server Project

## Bài 4: Spring MVC - Truyền dữ liệu Model qua lại View, Controller

Thực hiện ứng dụng Spring MVC 2 với Model Student và Customer theo form với các view tương ứng sau:

*Request mặc định : /*

*Request mặc định : /*

*🡪url: serletname/*

*Request hiển thị form hello*

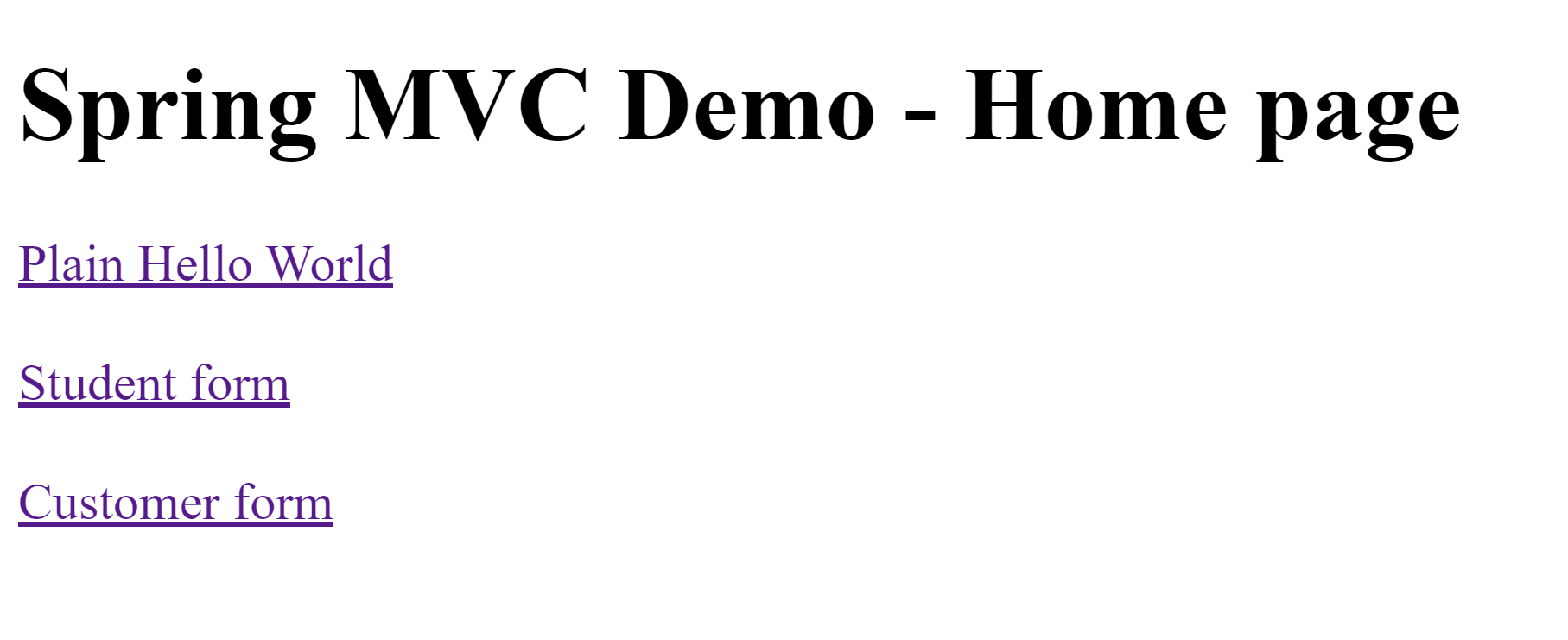
*🡪url: serletname/hello/showForm*

*Request để hiển thị form student*

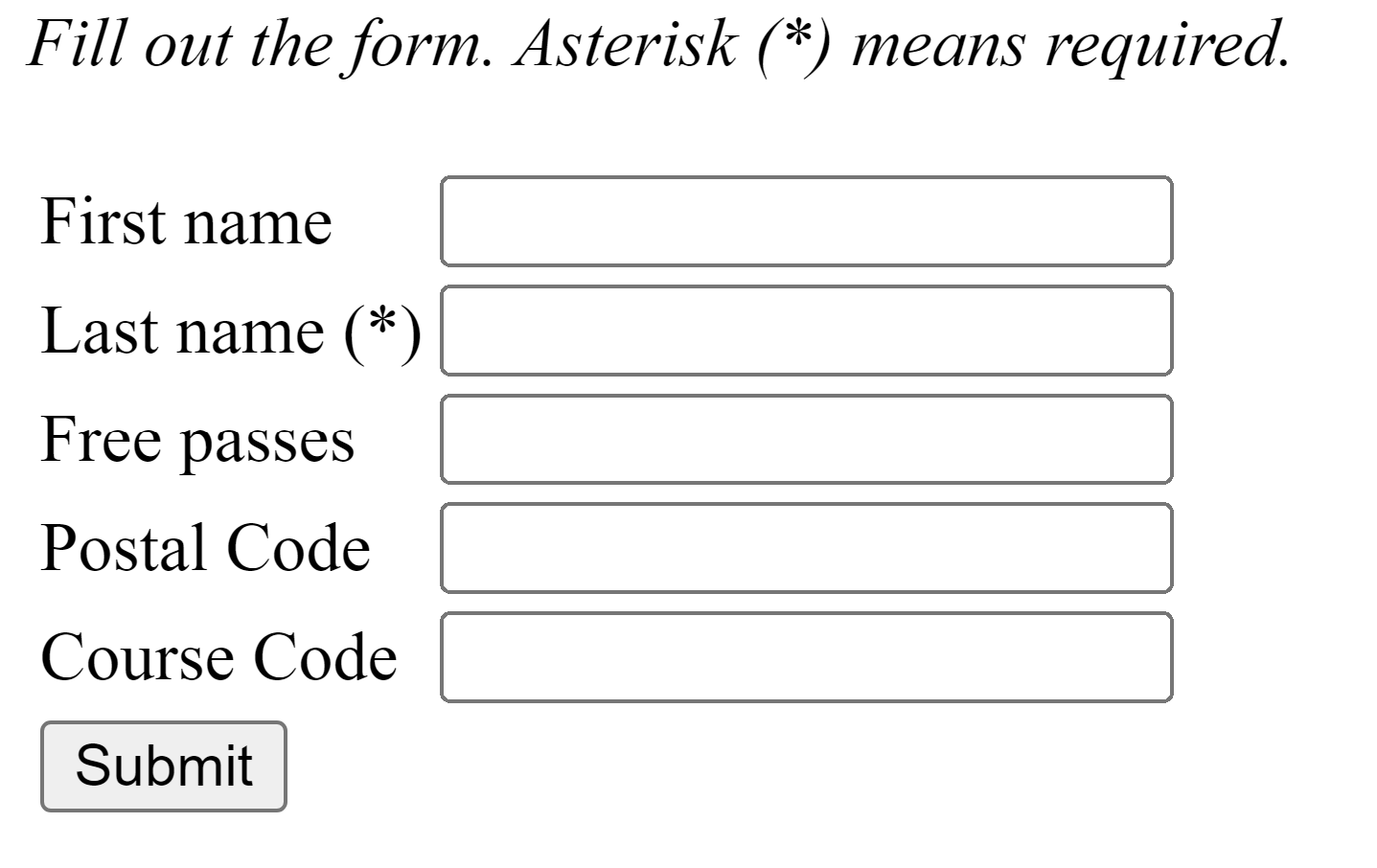
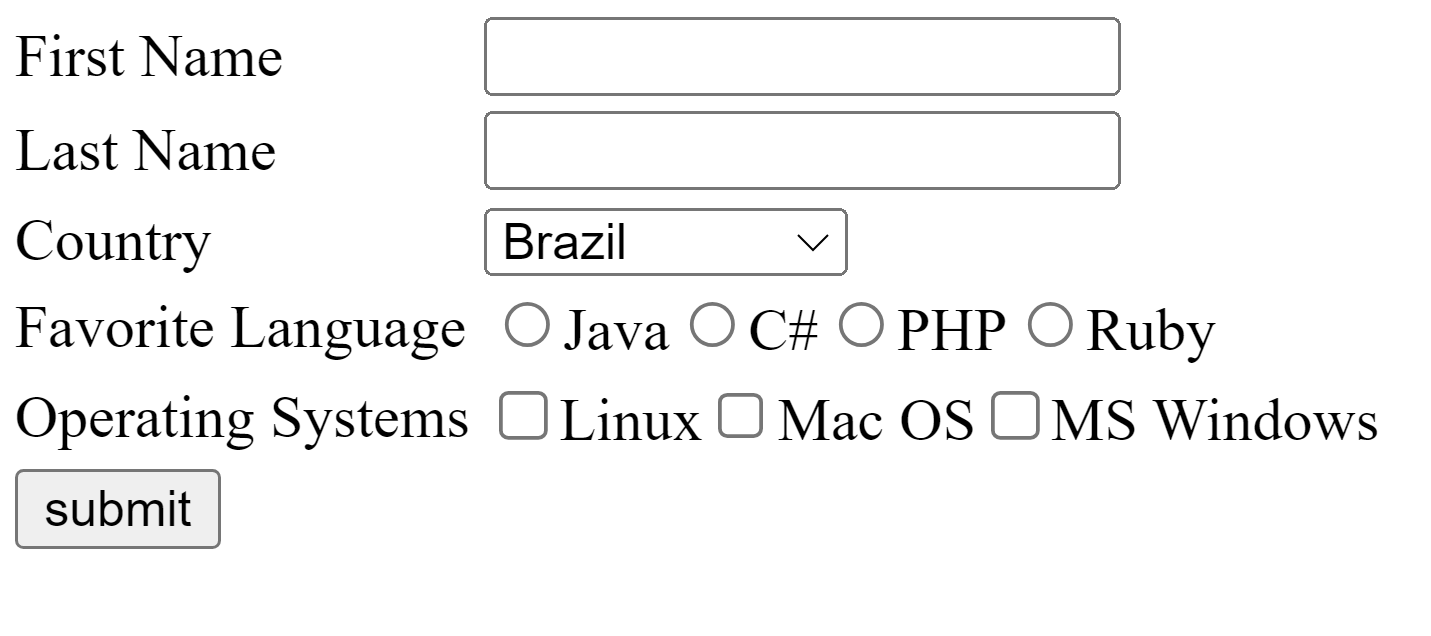
*🡪 url: serletname/student/showForm*

*Request để hiển thị form customer*

*🡪url: serletname/customer/showForm*



*Form student*  *Form customer*

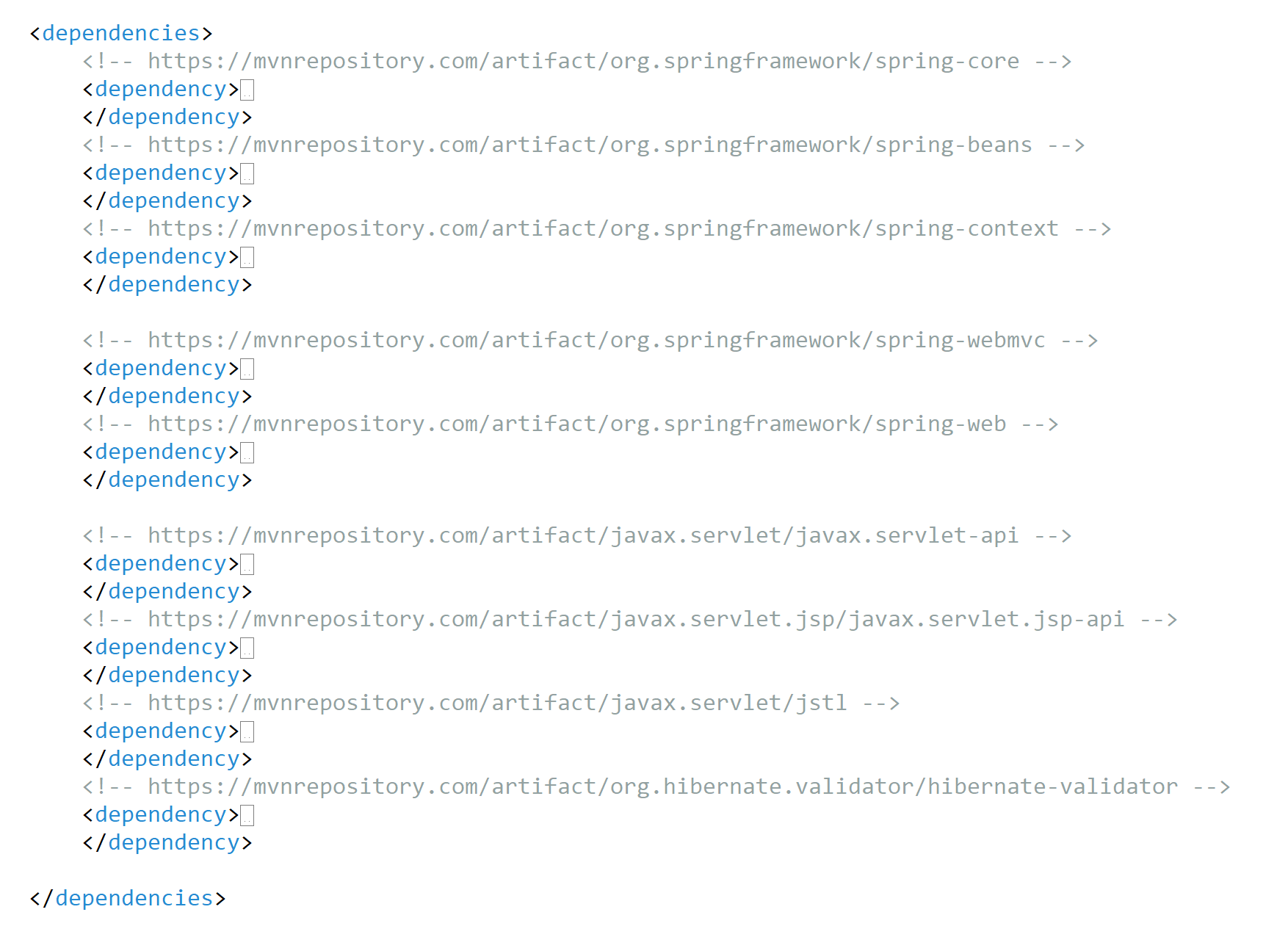


Tạo *Dynamic Web Project*, khai báo các thông tin liên quan

*B1. web.xml*: Khai báo cấu hình cần thiết của ứng dụng Web *(Cấu hình DispatcherServlet – 1.)*

{1. Cấu hình Spring dispatcher servlet, 2. Mapping cho Spring dispatcher servlet }

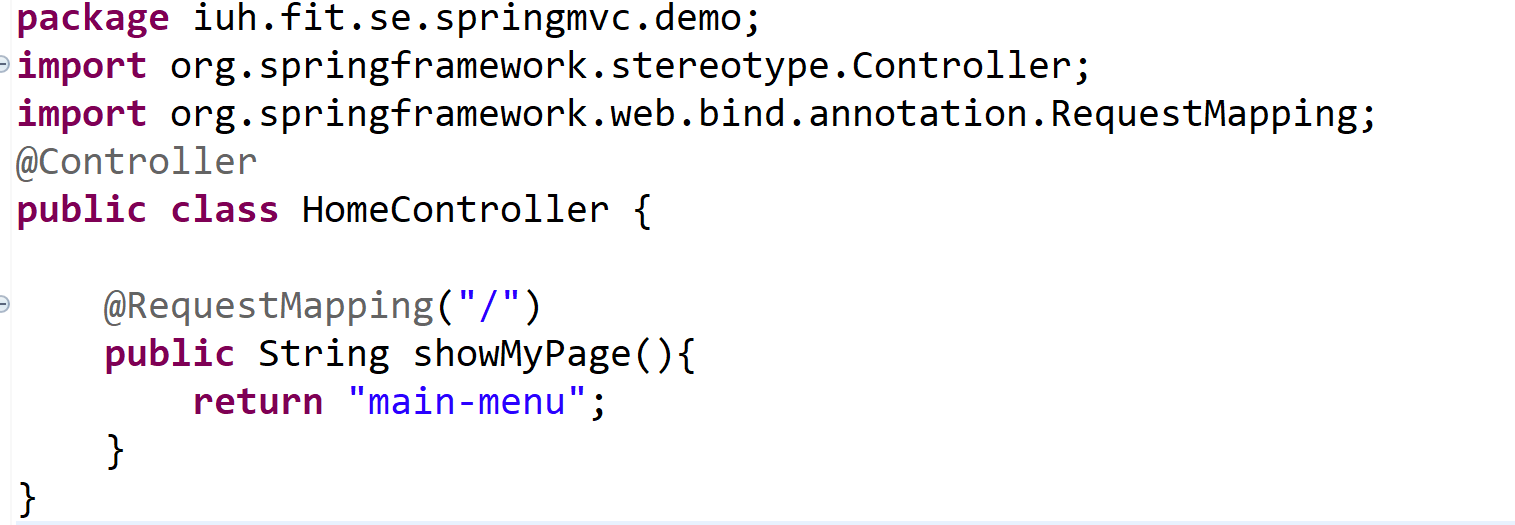
*B2. pom.xml*: Khai báo các dependencies Khai báo các dependencies, Maven 🡪 Update Project



*B3. WEB-INF/spring-mvc-content.xml*:   
Khai báo cấu hình tối thiểu về Spring MVC *(Cấu hình servlet của project – 2.)*

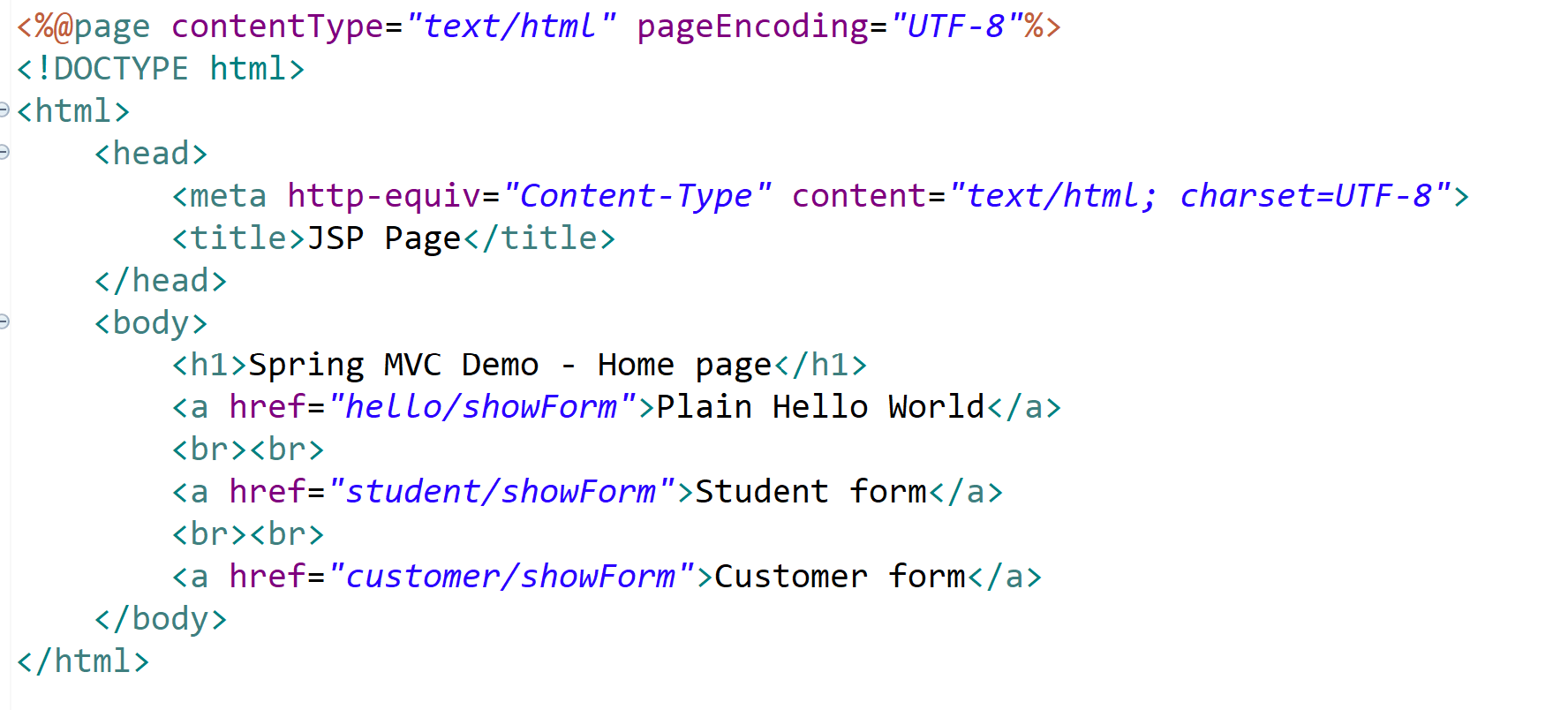
{3. Scan-context, 4. Annotation validation, 5. ViewResolver}

B4. Tạo Controller *iuh.fit.se.springmvc.demo.HomeController.java* hiển thị danh sách các liên kết ứng với các request /student hay /customer từ client gửi lên



Request Mapping (/): mặc định của Project chạy method showMyPage() 🡪 main-menu.jsp

B5. Trong thư mục *WEB-INF/view/main-menu.jsp*:



*Request Mapping:*

*1.hello/showForm*

*🡪 Controller cho request mapping hello, method showForm*

*2.student/showForm*

*🡪Controller cho request mapping student, method showForm*

*3.customer/showForm*

*🡪Controller cho request mapping customer, method showForm*

B6. Theo như Request trên, project cần có các Controller tương ứng cho các Request

Tạo Controller *iuh.fit.se.springmvc.demo.HelloWorldController.java*



*Client Request URL như sau:* [*http://localhost:8080/.../hello/showForm*](http://localhost:8080/.../hello/showForm)

*🡪helloworld-form.jsp*

[*http://localhost:8080/.../hello/*](http://localhost:8080/.../hello/)*processForm*

*🡪helloworld.jsp*

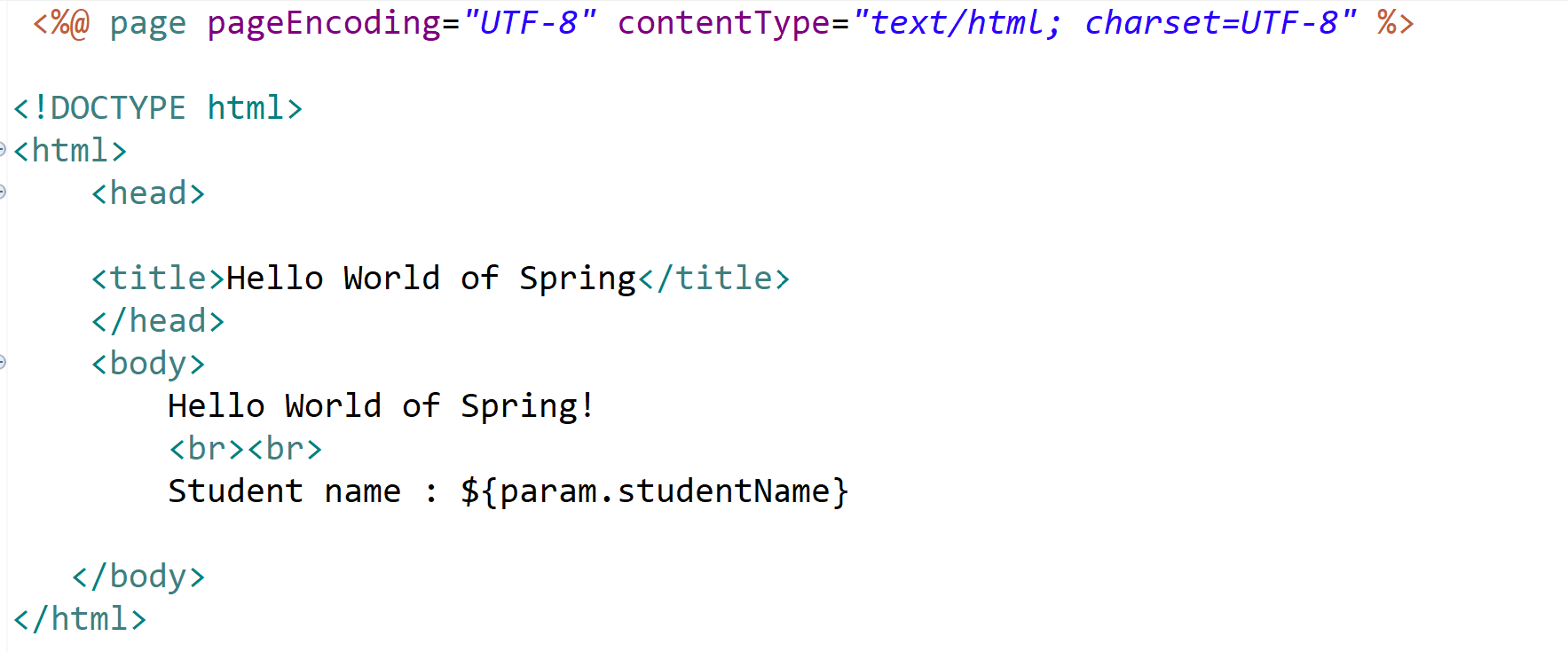
B7. Tạo view /WEB-INF/view/*helloworld-form.jsp*

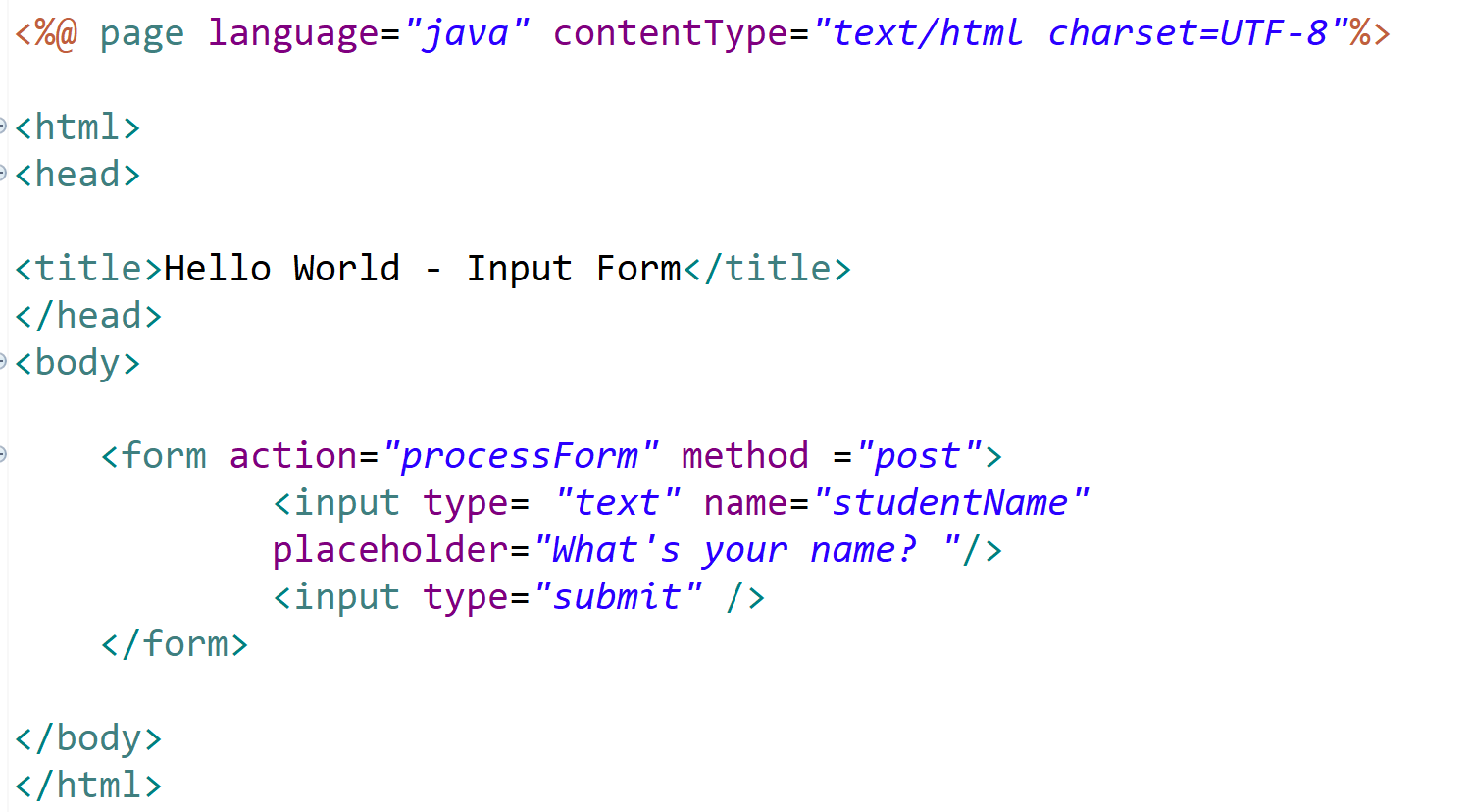
*helloworld-form.jsp* chứa form nhập dữ liệu truyền qua trang *helloworld.jsp*

Truyền dữ liệu giữa các view jsp

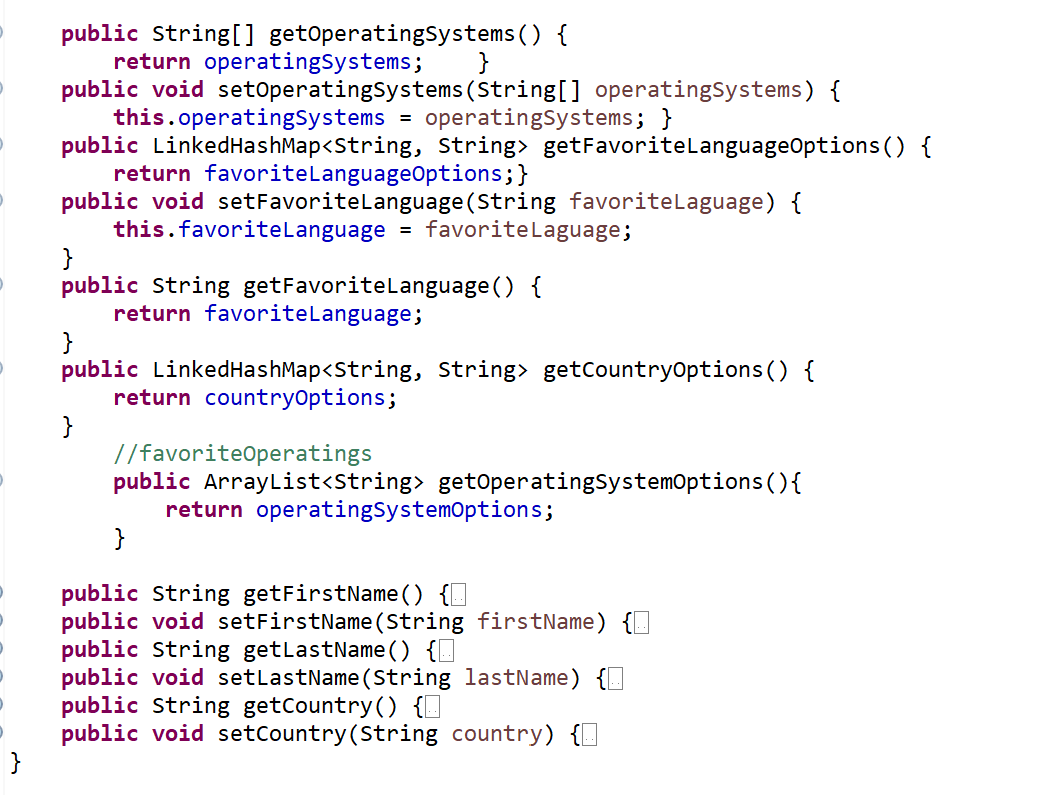
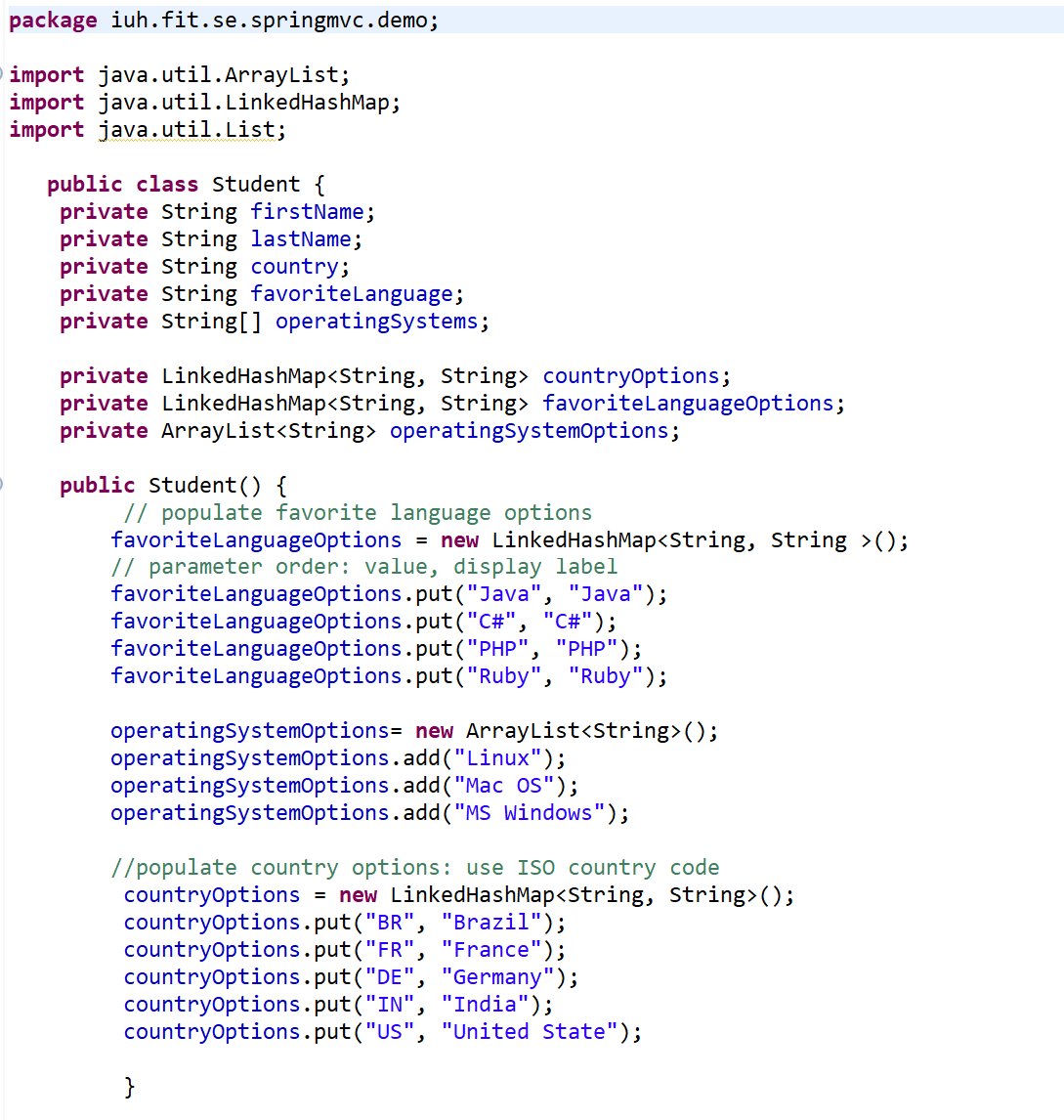
*helloworld-form.jsp*

*helloworld.jsp* nhận parameter *name= “studentName”*

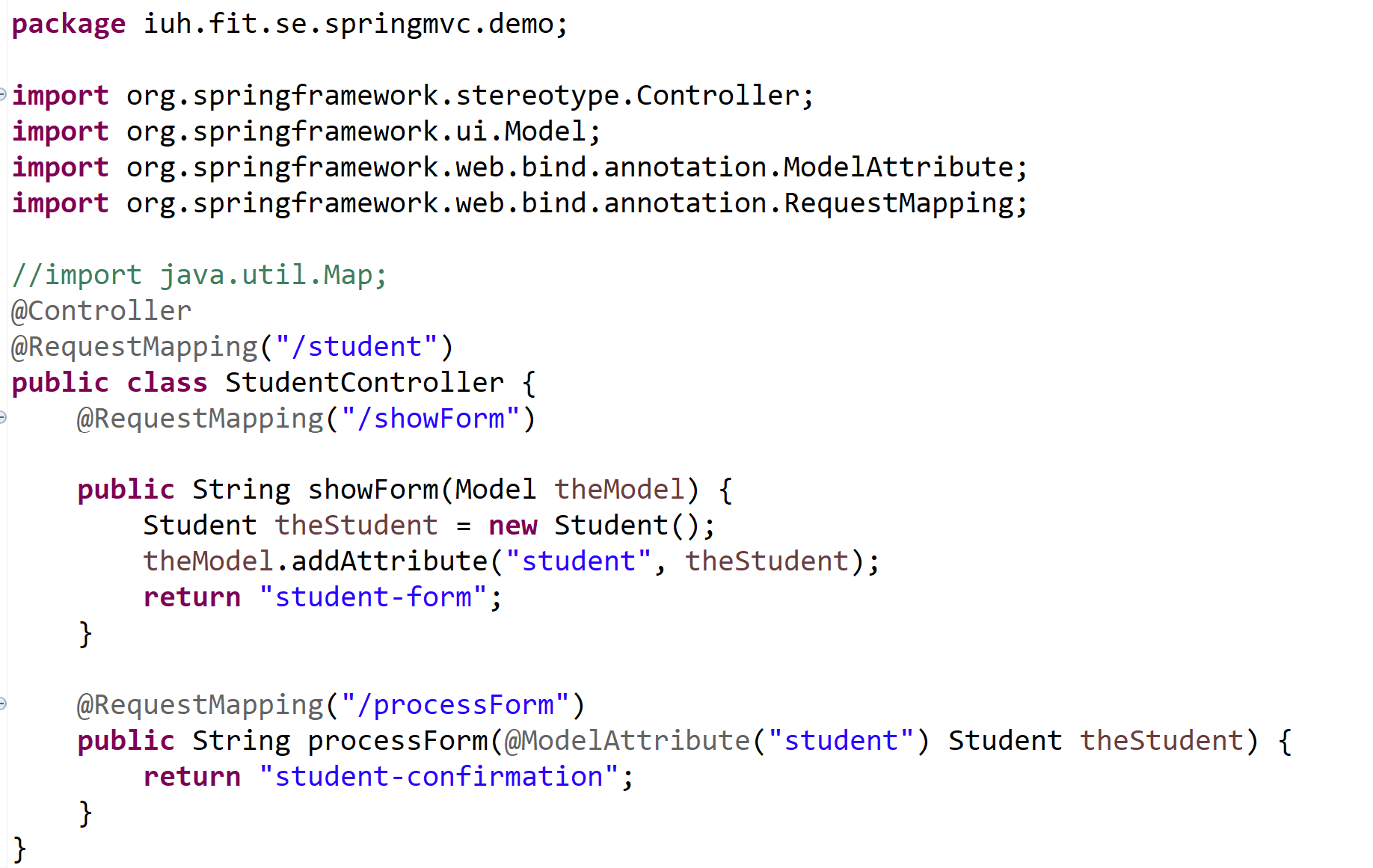




B8. Theo form Student thì cần tạo model iuh.fit.se.springmvc.demo.Student với các thuộc tính như sau:



B9. Tạo Controller iuh.fit.se.springmvc.demo.StudentController



*Request Mapping*

1. *student/showForm*

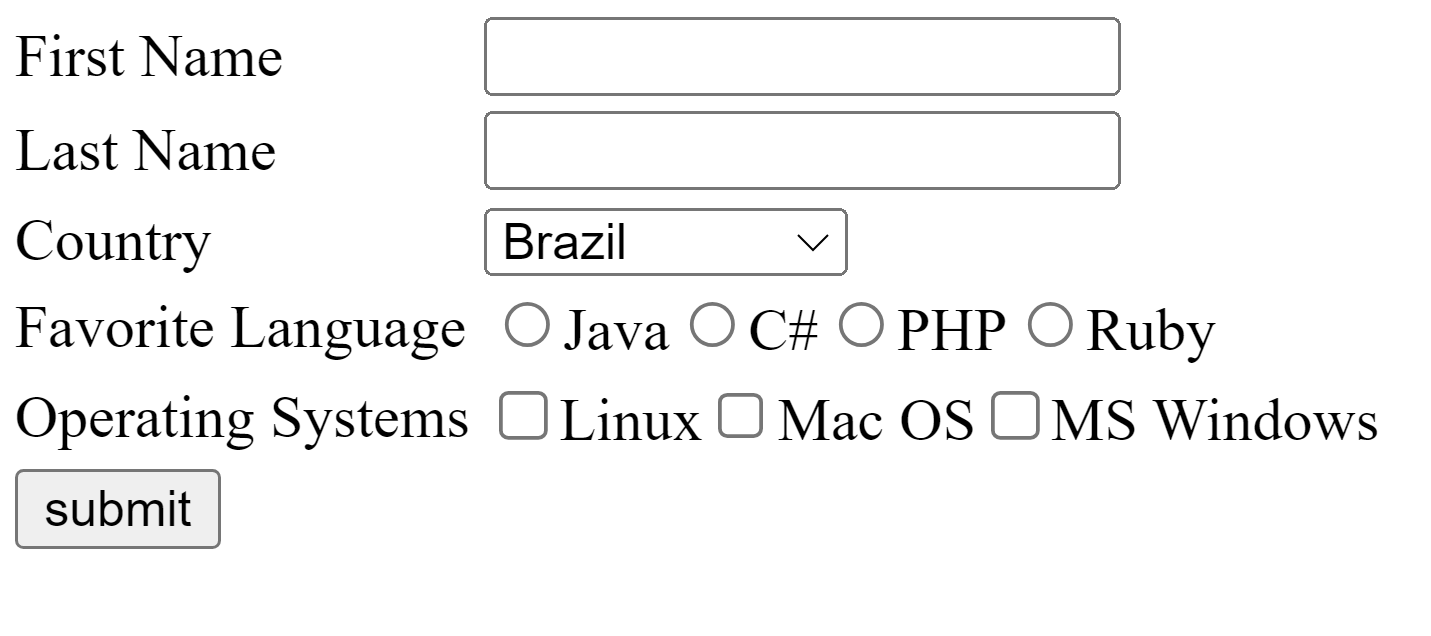
*Truyền Model qua view*

1. *student/processForm*

*(@ModelAtribute(“Student”)*

B10. Trong view /WEB-INF/view/student-form.jsp trong đó sử dụng form MVC cho Listbox Country, Radio Favorite Language, Checkbox Operationg System được nhận items từ Model Student ở B3





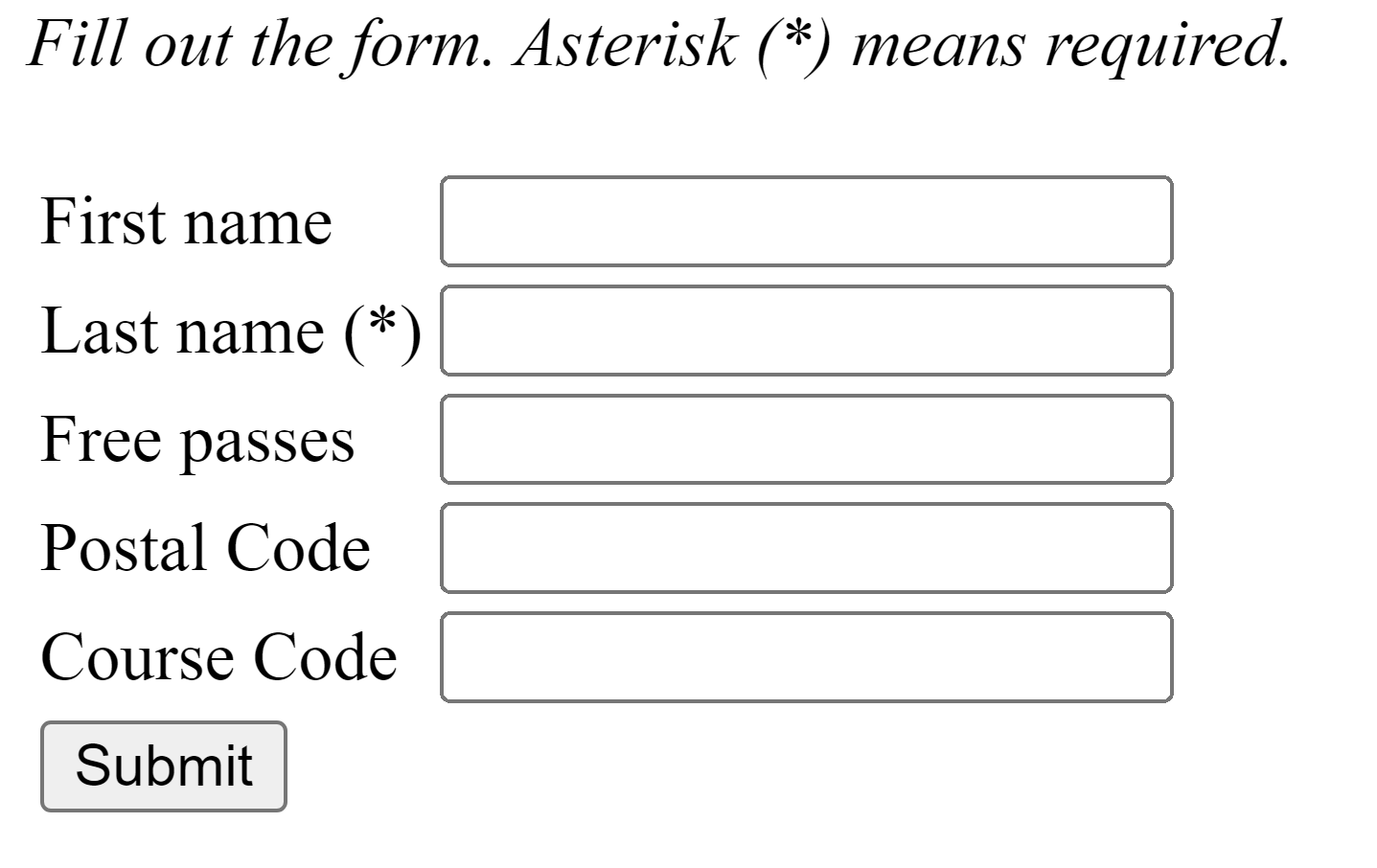
B11. Trong view /WEB-INF/view/student-confirmation.jsp nhận Model student truyền từ view sang view



🡪Run Project 🡪Run As🡪Run on Server

B12: Tương tự như Model Student, tạo Model iuh.fit.se.springmvc.demo.Customer theo form Customer

Trong phần này kết hợp kiểm tra Spring Validation cho các giá trị nhập vào trên Form:

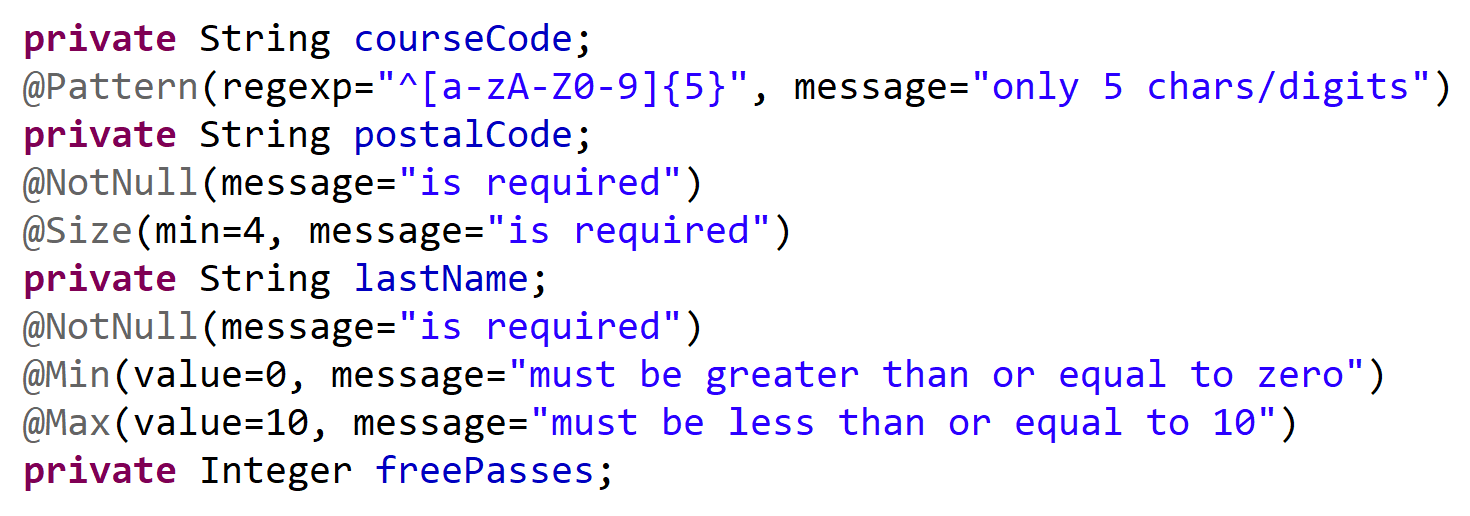


Các Validation và Validation Annotations thường dùng do Spring cung cấp :

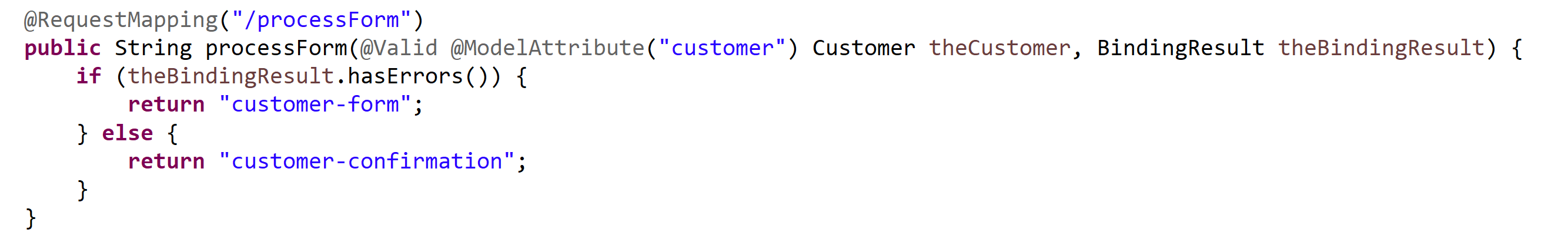
|  |  |
| --- | --- |
| **Annotation** | **Description** |
| @NotNull | The value can't be null. |
| @Min | Number must be equal or greater than the specified value. |
| @Max | Number must be equal or less than the specified value. |
| @Size | Size must be equal to the specified value. |
| @Pattern | Sequence follows the specified regular expression. |
| @Future/Past | Date must be in future or past of given date |
| Others… |  |

|  |
| --- |
| **Validation Features** |
| Required |
| Validate length |
| Validate numbers |
| Validate with regular expressions |
| Custom validation |

B13. Tại Model: Các validation được cung cấp qua các Annotation trực tiếp trên các thuộc tính.



B14. Tại Controller: sử dụng @Valid trên các Method có các Model đã thiết lập Validation, sử dụng Model **BindingResult** để nhận kết quả của Validation.



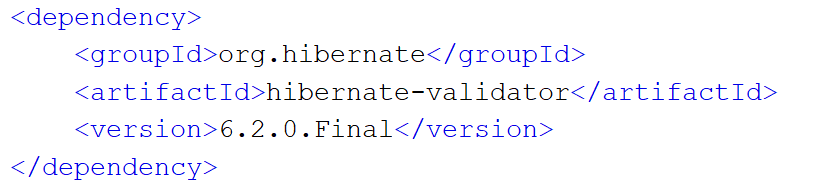
*View cho customer*

B15. Tại View: Hiển thị thông báo lỗi trên mỗi thành phần của form Validation thông qua <form:errors>

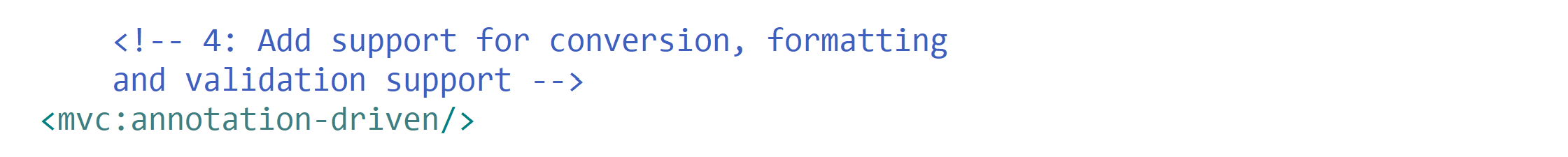


*Lưu ý:*

* Trong file pom.xlm thêm dependency của Hibernate-Validator cho project

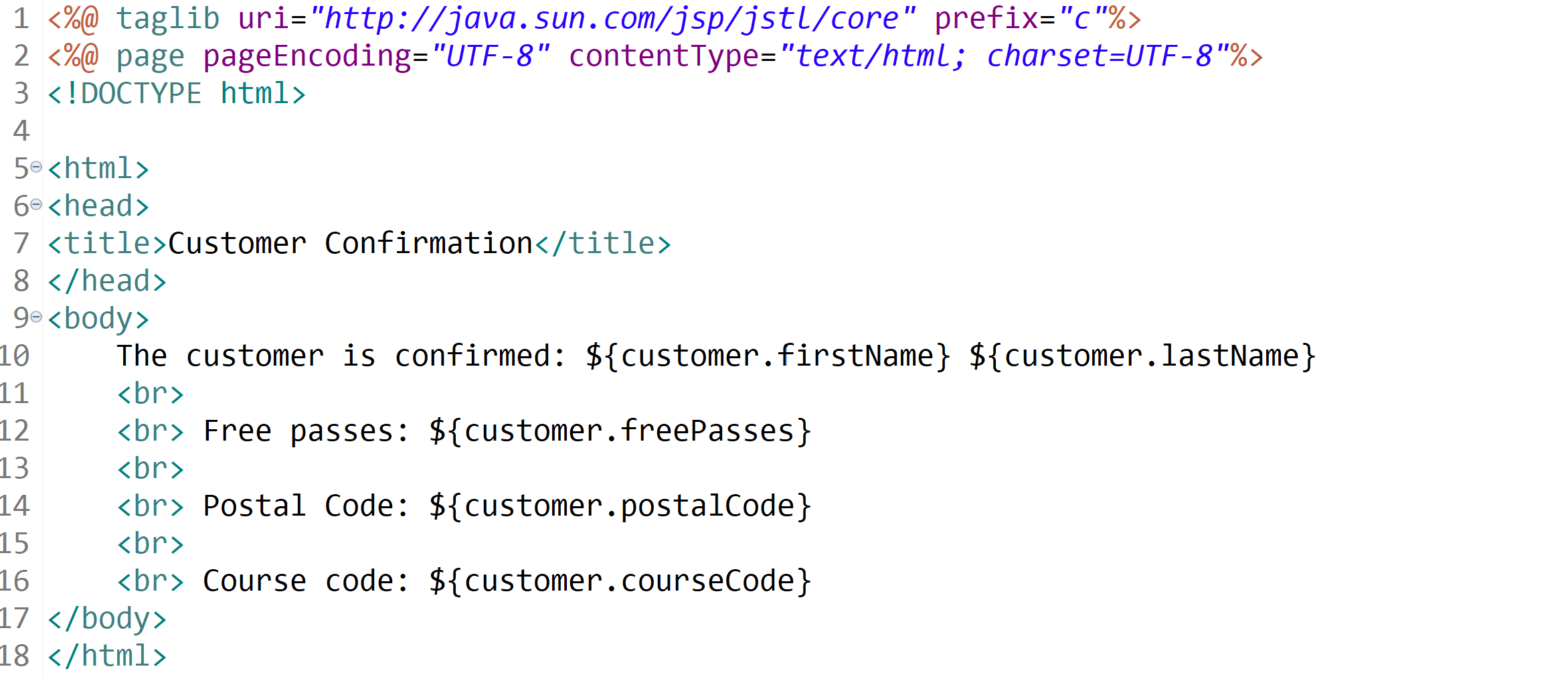


* Trong file *WEB-INF/ spring-mvc-content.xml* khai báo bean validation



B16: Tạo view /WEB-INF/view/*customer-form.jsp, customer-confirmation*

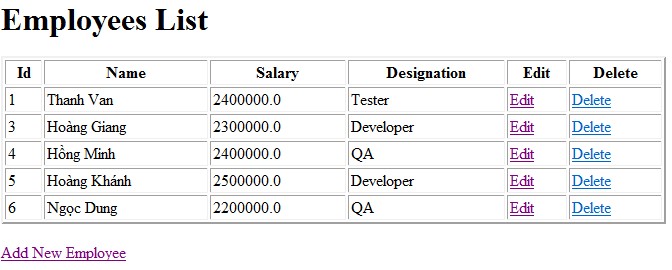




## Bài 5: Spring MVC – CRUD – EMPLOYEE

Spring MVC với các thao tác CRUD thêm, xóa, sửa với cơ sở dữ liệu.

Thực hiện thao tác thêm xóa sửa với CSDL quản lý thông tin nhân viên. CSDL quản lý bao gồm bảng Employee (ID, NAME, SALARY, DESIGNATION), trong đó mã nhân viên ID tự động tăng.



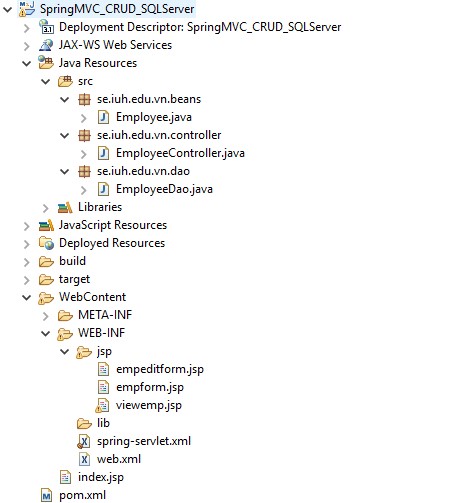
Thực hiện các chức năng:

* Khi chương trình chạy cho phép hiển thị danh sách nhân viên.
* Chọn “Edit” cho phép sửa thông tin của từng nhân viên, chọn “Delete” cho phép xóa thông tin của nhân viên và cập nhật CSDL..

Chọn “Add New Employee” hiển thị form cho phép thêm nhân viên mới vào CSDL.

*Tham khảo hướng dẫn:*

B1. Tạo Dynamic Web Project, khai báo các thông tin liên quan



B2. *pom.xml:*

Khai báo các dependencies, Maven 🡪 Update Project

*Khai báo dependencies như bài 2 gồm các dependences cơ bản:*

*/springframework/spring-core*

*/springframework/spring-beans*

*/springframework/spring-context*

*/springframework/spring-webmvc*

*/springframework/spring-web*

*Khai báo thêm dependencies cho việc thực hiện kết nối database:*



B3: *web.xml*: Khai báo cấu hình cần thiết của ứng dụng Web *(Cấu hình 1. DispatcherServlet)*

{1. Cấu hình Spring dispatcher servlet, 2. Mapping cho Spring dispatcher servlet }

B4: *WEB-INF/spring-servlet.xml*:   
Khai báo cấu hình tối thiểu về Spring MVC *(Cấu hình 2. servlet, servlet mapping của project)*

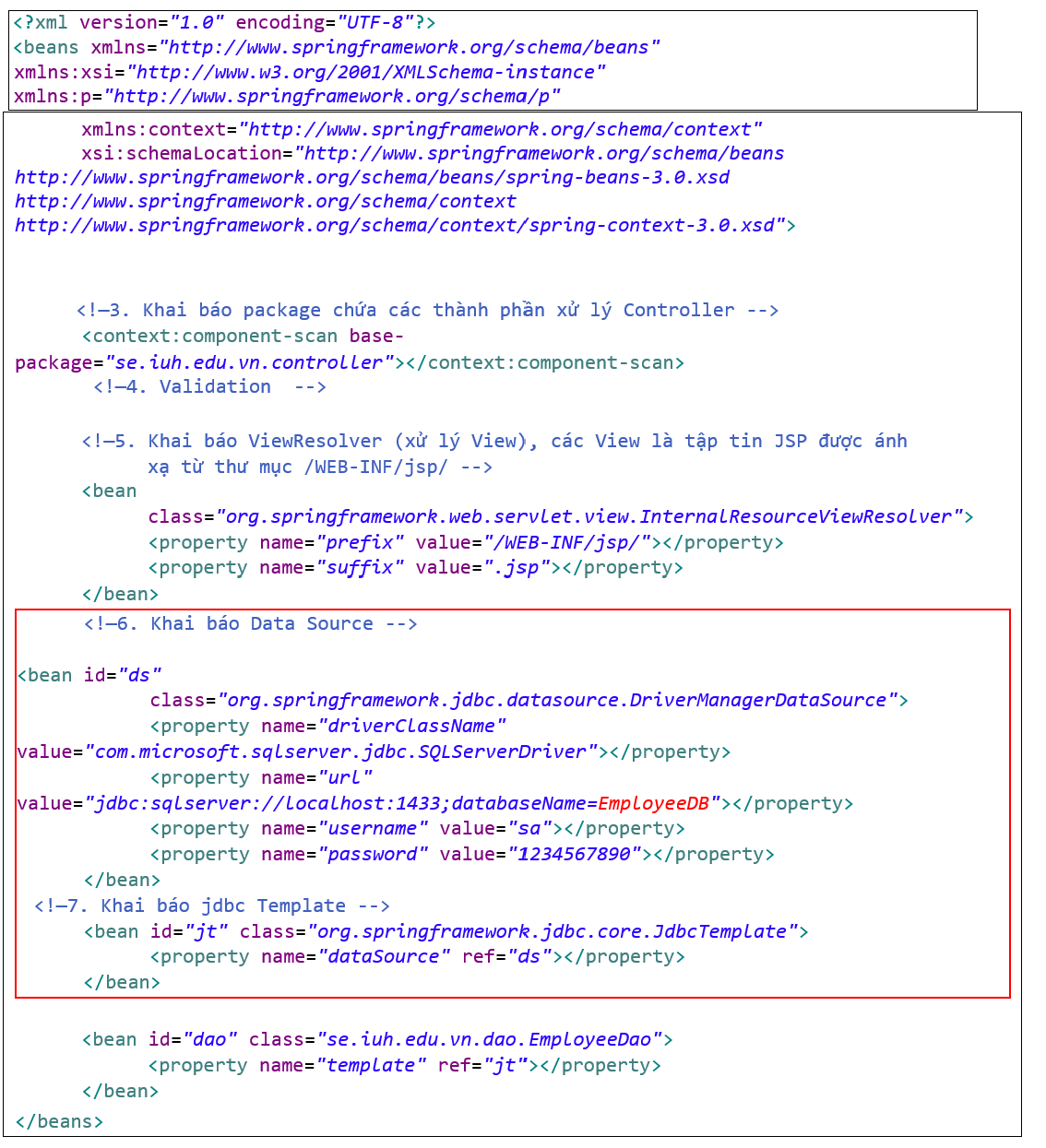
{3. Scan-context, 4. Annotation validation, 5. ViewResolver : như bài 2}

Thêm cấu hình cho kết nối SQL

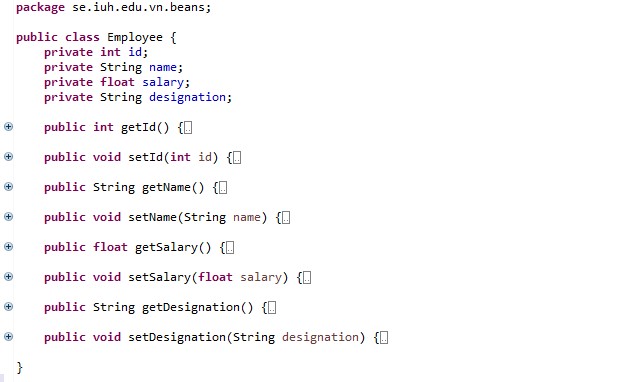
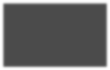
*{6. khai báo bean ds cấu hình các thông số kết nối CSDL }*

*{7. Khai báo bean jt đến class JdbcTemplate làm việc với CSDL được inject vào và sử dụng sau này trong ứng dụng }*

Tham khảo code dưới:



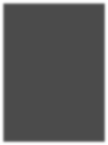
B5. Lớp mô tả dữ liệu (Entity)



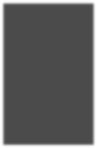
B6. Lớp truy xuất dữ liệu (DAO): Lớp này chứa các phương thức thao tác dữ liệu (thêm, sửa, xóa) và truy vấn dữ liệu.

* insert(): thêm
* update(): sửa
* delete(): xóa
* getXyz(): truy vấn

Lớp nào được chú thích bởi @Repository để có thể inject vào Controller trong ứng dụng bởi @Autowire để sử dụng sau này. Khai báo này có thể để trong lớp EmployeeDao hoặc khai báo bên Controller.

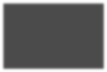


B7. Lớp xử lý (Controller)

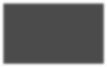


B8. Các giao diện xử lý xem danh sách, thêm nhân viên mới, cập nhật nhân viên đã tồn tại trong CSDL.

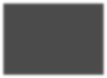
*viewemp.jsp*



*empform.jsp*



*empeditform.jsp*



## Bài 6. Spring MVC - Upload file

Thực hiện ứng dụng Spring MVC Upload File

*Tham khảo hướng dẫn:*

B1. Khai báo Maven *pom.xml*:

Ngoài những thư viện của Spring, khai báo thêm các dependencies:

<!-- https://mvnrepository.com/artifact/jstl/jstl -->

<dependency>

<groupId>jstl</groupId>

<artifactId>jstl</artifactId>

<version>1.2</version>

</dependency>

<!-- https://mvnrepository.com/artifact/commons-io/commons-io -->

<dependency>

<groupId>commons-io</groupId>

<artifactId>commons-io</artifactId>

<version>2.6</version>

</dependency>

<!-- https://mvnrepository.com/artifact/commons-fileupload/commons-fileupload --> <dependency>

<groupId>commons-fileupload</groupId>

<artifactId>commons-fileupload</artifactId>

<version>1.3.3</version>

</dependency>

B2. Cấu hình Spring *spring-servlet.xml*

Ngoài các khai báo Khai báo package chứa các thành phần xử lý Controller, khai báo bean ViewResolver, khai báo thêm bean org.springframework.web.multipart.commons.CommonsMultipartResolver:

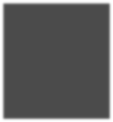
<bean id=*"multipartResolver"*

class=*"org.springframework.web.multipart.commons.CommonsMultipartResolver"* />

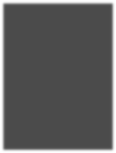
B3. Cấu hình Project web.xml

Khai báo FrontController: DispatcherServlet,

B4. Tạo UploadController



B5. Tạo View



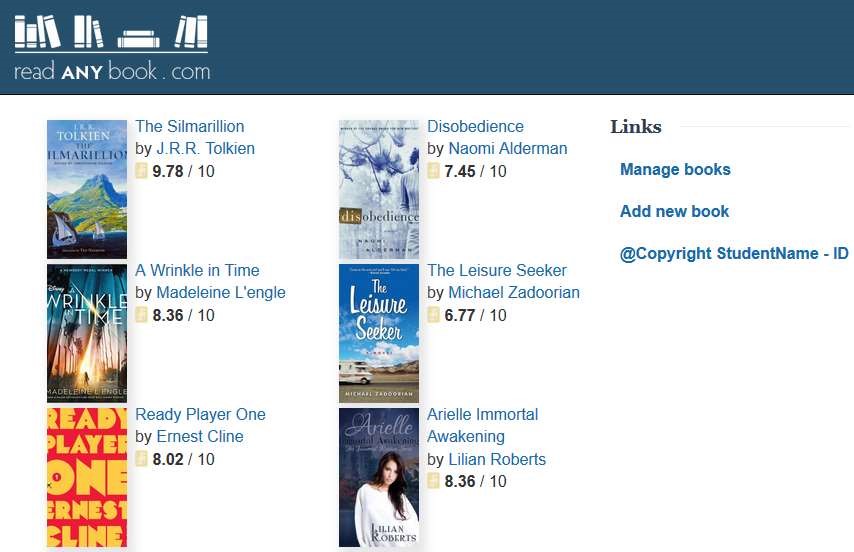
## Bài 7. Spring MVC – CRUD - BOOKONLINE

Xây dựng ứng dụng quản lý thông tin sách, các trang Web cần thực hiện các chức năng: *xem danh sách các quyển sách, thêm mới sách và xóa sách không cần thiết.* Một phần của cơ sở dữ liệu được mô tả như sau:

BookOnline(BookID, BookName, Author, PublishYear, Rating, ImageURL)

Xây dựng ứng dụng dùng Spring MVC Framework kết hợp với SQL Server và Web server Tomcat thực hiện các công việc sau:

* Tạo cơ sở dữ liệu với các quan hệ như mô tả trên và nhập dữ liệu cần thiết để kiểm tra chương trình.
* Dùng HTML/CSS tạo Layout chung cho chương trình tương tự:



*Hình 1. Layout của trang Web quản lý thông tin sách cho website readanybook.com*

* Viết lớp kết nối CSDL và lớp thực hiện thao tác với việc truy xuất cơ sở dữ liệu (bao gồm các phương thức: liệt kê danh sách sách, thêm mới, xóa thông tin sách).

* Tạo Controller cho phép hiển thị giao diện Web với danh sách các quyển sách online lấy từ cơ sở dữ liệu.

* Tạo Controller cho phép hiển thị giao diện thao tác với việc thêm dữ liệu (*nhấn vào link “Add new book”*). Dữ liệu phải được kiểm tra trước khi lưu vào cơ sở dữ liệu:
  + Tất cả dữ liệu *BookID, BookName, Author, PublishYear, Rating, ImageURL* đều bắt buộc.
  + Năm xuất bản (*PublishYear*): từ 1900 – 2018
  + Tên sách (*BookName*): Các ký tự đầu của mỗi từ là ký tự hoa, không có số hay ký tự đặc biệt trong tên.

Sau khi thực hiện thêm thành công, hiển thị danh sách mới.

* Tạo Controller cho phép hiển thị giao diện Web cho phép thực hiện chức năng quản lý (*nhấn vào*

*link “Manage books”*): thao tác hủy dữ liệu. Trước khi hủy hỏi người dùng có muốn hủy không.