Lesson 11
Spring MVC Framework
Infinite Diversity Arising from Unity

MVC Pattern

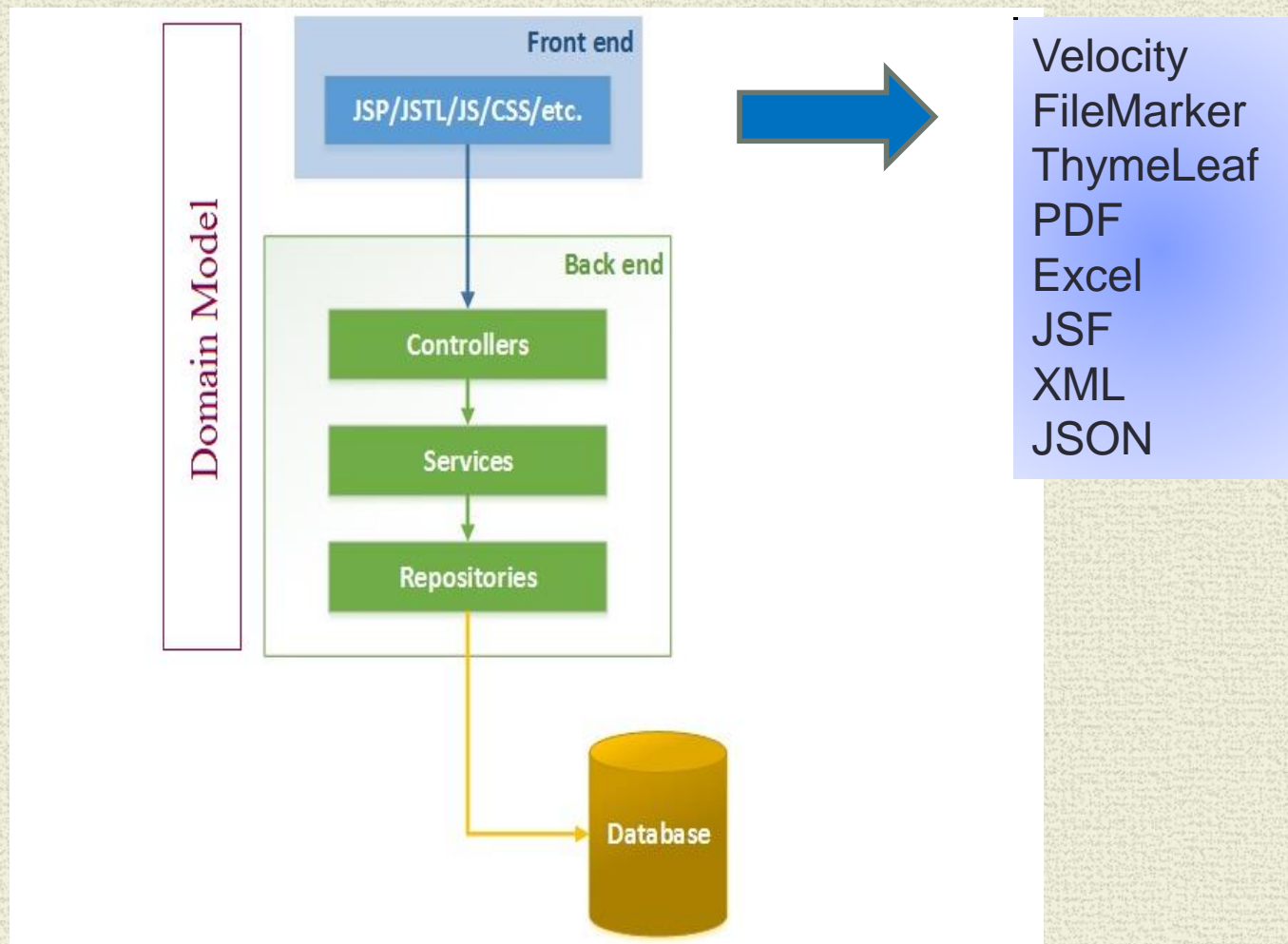*See "Spring-less MVC" DEMO -- Product4aJSP*
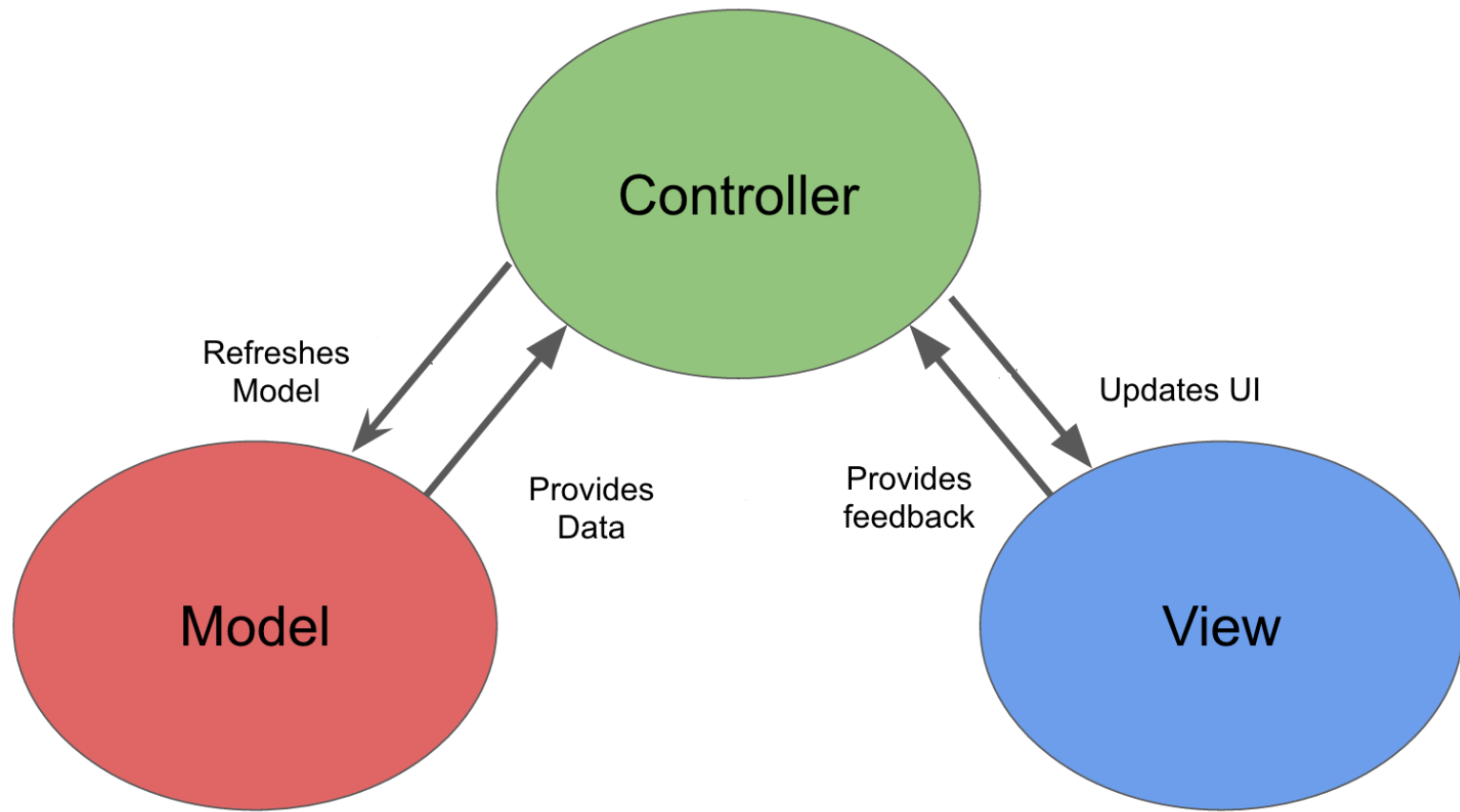
Spring MVC

Distinct Separation of Concerns

Clearly defined interfaces for role/responsibilities "beyond" Model-View-Controller

- Single Central Servlet

    Manages HTTP level request/response

    delegates to defined interfaces

- Models integrate/communicate with views

    No need for separate form objects

- Views are plug and play

- Controllers allowed to be HTTP agnostic

# Spring N-Tier With Spring MVC Presentation Tier

Front end

JSP/JSTL/JS/CSS/etc.

Domain Model

Back end

Controllers

Services

Repositories

Database

Velocity
FileMarker
ThymeLeaf
PDF
Excel
JSF
XML
JSON

Spring MVC DispatcherServlet Front Controller

- DispatcherServlet - "Front Controller" design pattern

   Common pattern used by MVC frameworks

- Single Central Servlet that dispatches requests to controllers and offers other functionality that facilitates the development of web applications.

- Completely integrated with the Spring container

   Able to "exploit" Spring framework features

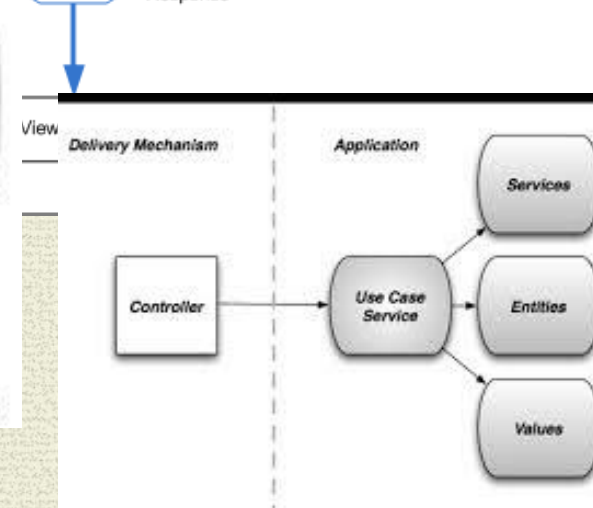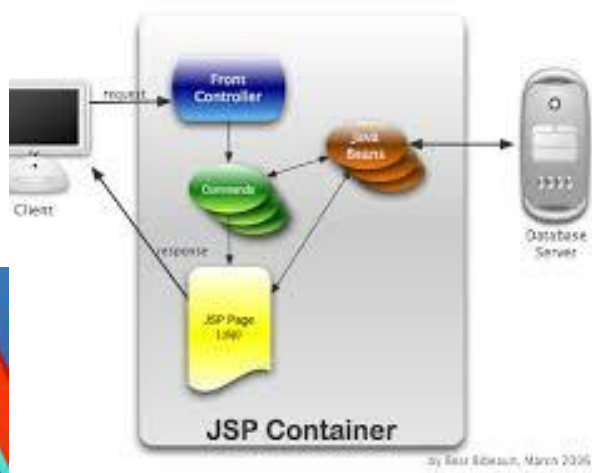- Has a WebApplicationContext, which inherits all the beans already defined in the root WebApplicationContext.
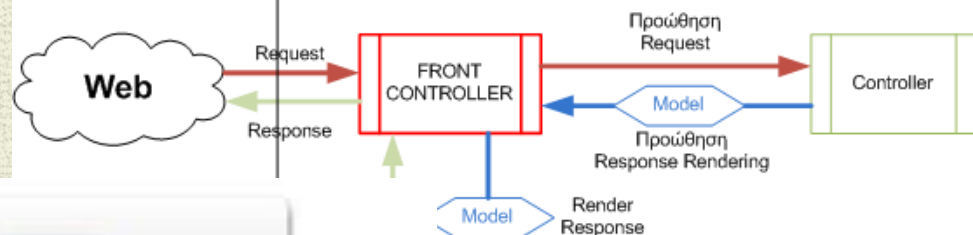
FRONT CONTROLLER
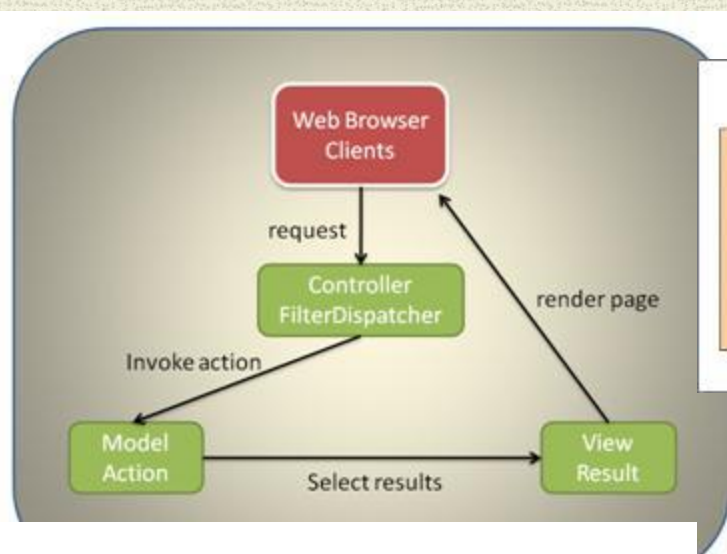
Front Controller

Problema

- Si vuole fornire un punto di accesso centralizzato per la gestione delle richieste al livello dello strato di presentazione, in modo da sparare la logica di presentazione da quella di *processamento* delle richieste stesse.
- Inoltre si vuole evitare di avere del codice duplicato e si vuole applicare una logica comune a più richieste.

Incoming request → Front Controller

Return response ← Front Controller

Delegate request → Controller (Handle request)

Model ← Controller (Create model)

Delegate rendering of response

Return control

Model

Render response

View Template

Servlet engine (e.g. Tomcat)
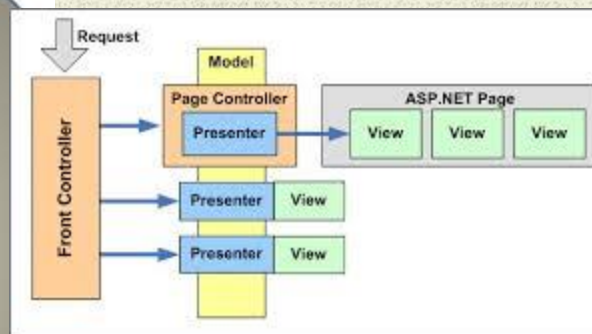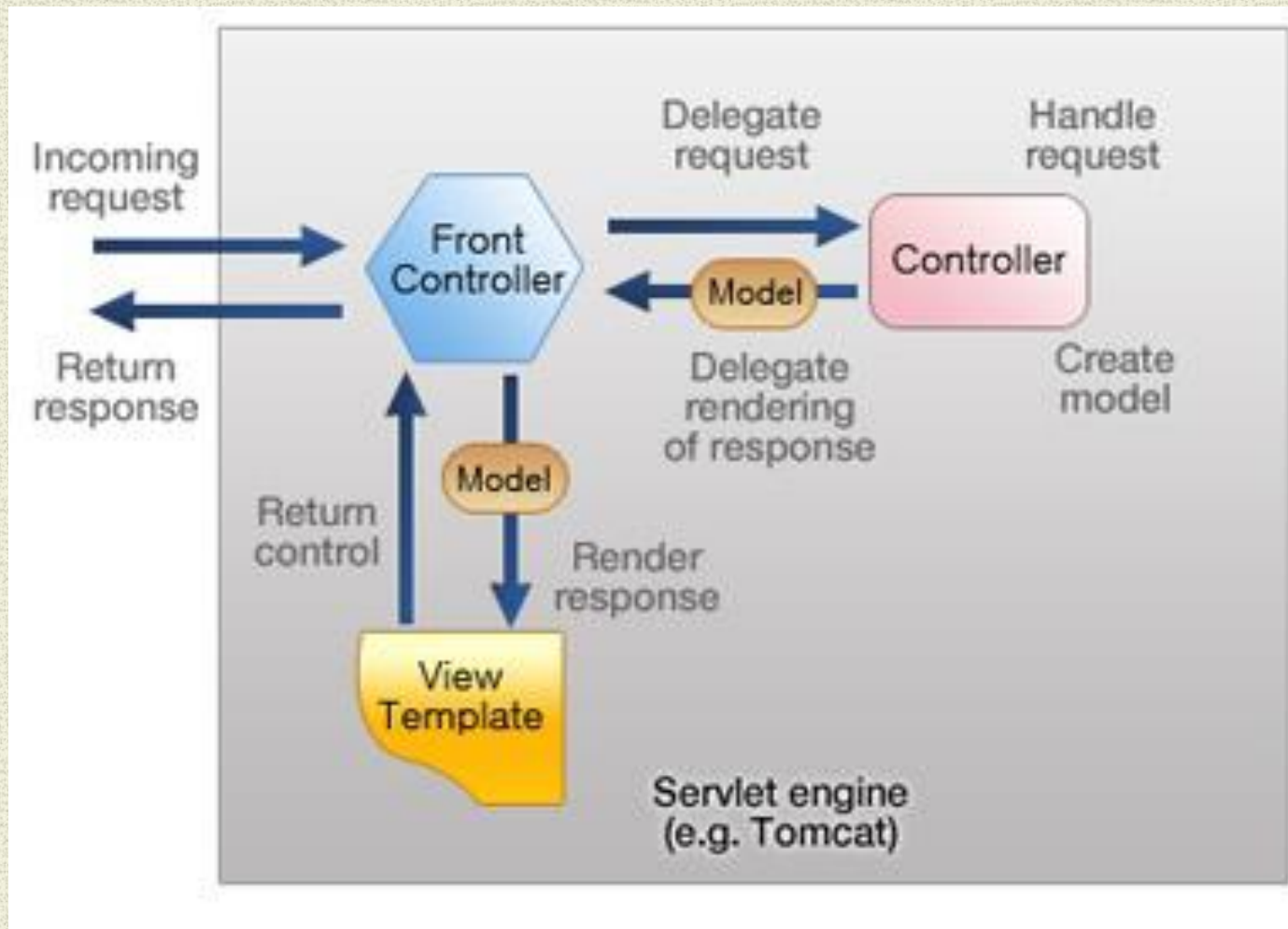
Spring MVC Front Controller Configuration

```xml
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="3.0"
        xmlns="http://java.sun.com/xml/ns/javaee"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd
    <servlet>
        <servlet-name>springmvc</servlet-name>
        <servlet-class>
            org.springframework.web.servlet.DispatcherServlet
        </servlet-class>
        <init-param>
            <param-name>contextConfigLocation</param-name>
            <param-value>/WEB-INF/config/springmvc-config.xml</param-value>
        </init-param>
        <load-on-startup>1</load-on-startup>
    </servlet>

    <servlet-mapping>
        <servlet-name>springmvc</servlet-name>
        <url-pattern>/</url-pattern>
    </servlet-mapping>
</web-app>
```

Spring MVC XML Configuration File

```xml
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:security="http://www.springframework.org/schema/security"
    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/security
        http://www.springframework.org/schema/security/spring-security.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">

    <mvc:annotation-driven validator="validator"/>

    <context:component-scan base-package="edu.mum.controller" />

    <bean class="org.springframework.web.servlet.view.InternalResourceViewResolver">
        <property name="prefix" value="/WEB-INF/views/" />
        <property name="suffix" value=".jsp" />
    </bean>

    <mvc:resources  location="/resources/"  mapping="/resource/**"/>
```

XML Configuration file – enable annotations

<context:component-scan base-package= "pkg,pkg…" **>**

Scans defined packages to find and register @Component-annotated classes and activate basic annotations[e.g. @Autowired] - within the current application context

See Product4a DEMO

<mvc:annotation-driven/>

Enables support for specific annotations [e.g. @RequestMapping, etc.] that are required for Spring MVC to dispatch requests to @Controllers. It is based on MVC XML namespace

<tx:annotation-driven />

Enables support for specific annotations that are required for Spring Transactions @Transaction

Web Application Context

- Spring has multilevel application context hierarchies.

- Web apps by default have two hierarchy levels, root and servlet contexts:

Root WebApplicationContext

Service & Persistence tiers

WebApplicationContext
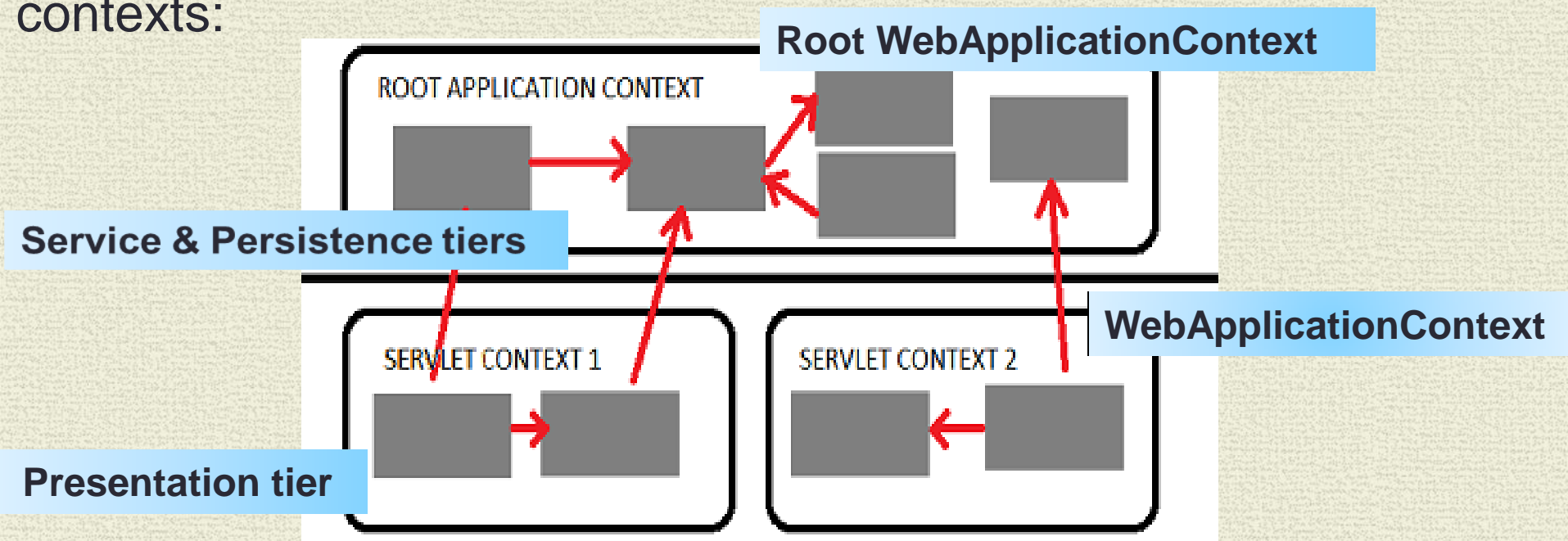
Presentation tier

- **Presentation tier has a WebApplicationContext [Servlet Context] which inherits all the resources already defined in the root WebApplicationContext [Services, Persistence]**

Data Binding

- Automatically maps request parameters domain objects
- Simplifies code by removing repetitive tasks
- Built-in Data Binding handles simple String to data type conversions
- HTTP request parameters [String types] are converted to model object properties of varying data types.
- Does NOT handle COMPLEX data types; that requires custom formatters
- Does handle complex nested relationships

Data Binding example

```java
package app04a.controller;

import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;

import app04a.domain.Product;

@Controller
public class ProductController {

    @RequestMapping(value={"/","/product"}, method = RequestMethod.GET)
    public String inputProduct() {
        return "ProductForm";
    }

    @RequestMapping(value="/product", method = RequestMethod.POST)
    public String saveProduct(Product product ) {
        return "ProductDetails";
    }

}
```
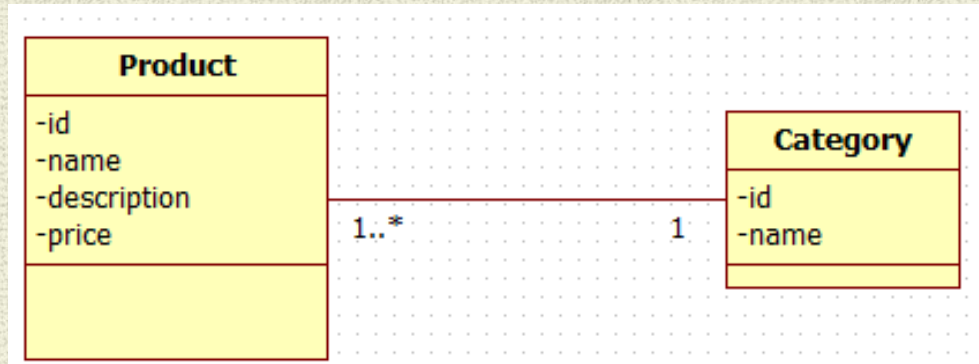
Data Binding - Relationships


```
<form action="product" method="post">
    <legend>Add a product</legend>
      <p>
<label for="category">Category </label>
    <select name="category.id">
            <option value="-">--Select Category--</option>
        <c:forEach var="category" items="${categories}">
            <option value="${category.id}" > ${category.name}</option>
        </c:forEach>
      </select>

  <label for="name">Product Name: </label>
      <input type="text" id="name" name="name" >
```

See SimpleMVC Demo

Main Point

- The basic ingredients of a Spring MVC application include web pages for the view (the known), the back end domain logic and data (knower or underlying intelligence), and the Spring framework and Dispatcher Servlet and managed beans as the controller to connect the view and model. *Knowledge is the wholeness of knower, known, and process of knowing.*

Spring MVC Annotations

- Handler Mapping
- Spring Annotations
  - @Controller
    - @RequestMapping
    - @ModelAttribute
    - @RequestParam
    - @PathVariable
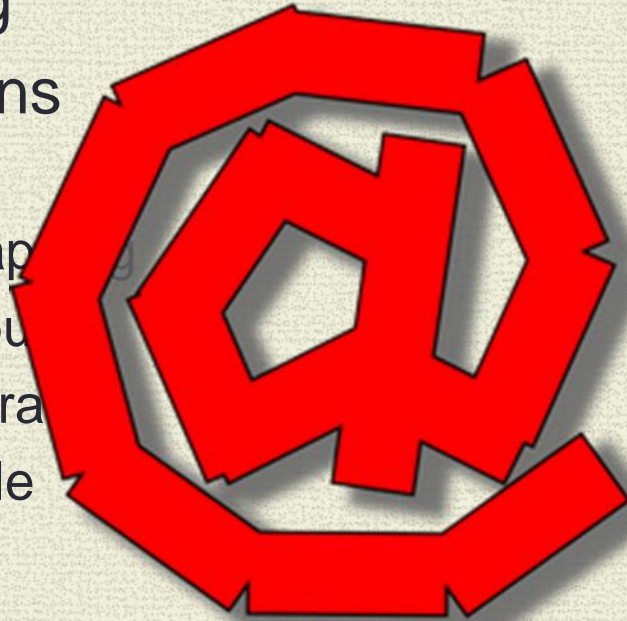
Spring MVC @Controller

- Spring implements a controller in a very abstract way, which enables you to create a wide variety of controllers.

- Spring Controllers do not extend specific base classes or implement specific interfaces

- They do not have direct dependencies on Servlet APIs, although you can easily configure access to Servlet facilities. [actual request, response objects, etc.]

-

Mapping Example

```java
@Controller
@RequestMapping({"/members"})
public class MemberController {

@Autowired
private MemberService  memberService;

@RequestMapping(value={"","/all"}, method = RequestMethod.GET)
public String listMembers(Model model) {
        model.addAttribute("members", memberService.findAll());
        return "members";
}

@RequestMapping(value="/{id}", method = RequestMethod.GET)
public String getMemberById(@PathVariable("id") Long id,Model model) {
        Member member = memberService.findOne(id);
        model.addAttribute("member", member);

         return "member";
}
```

See MemberSpringMVC

Mapping Example [Cont.] & Validation

```java
@RequestMapping(value = "/add", method = RequestMethod.GET)
public String getAddNewMemberForm(@ModelAttribute("newMember") Member newMember)
{
    return "addMember";
}


@RequestMapping(value = "/add", method = RequestMethod.POST)
public String processAddNewMemberForm( @Valid @ModelAttribute("newMember")
                            Member memberToBeAdded, BindingResult result) {

if(result.hasErrors()) {
return "addMember";
}

    memberService.saveFull(memberToBeAdded);

    return "redirect:/members";
```

See Member in MemberSpringMVC for Validation

@RequestParam

- Placed on Method argument

- http://localhost:8080/MemberSpringMVC/products/product?id=1

- @RequestMapping("/product")
- public String getProductById(Model model, @RequestParam("id") Long id) {

    Product product = productService.findOne(id);

- **Handling multiple values [e.g., multiple selection list]**

http://localhost:8091/store/sizechoices?sizes=small&sizes=medium&sizes=large

    public String getSizes(@RequestParam("sizes")String sizeArray[])

@RequestMapping Template with @PathVariable

-
- Facility to pass resource request as part of URL INSTEAD of as a @RequestParam
  - Conforms to RESTful service syntax

- http://localhost:8080/MemberSpringMVC/members/1

- `@RequestMapping("/{id}")`
- **public String getMemberById(@PathVariable("id") Long id,Model model) {**
- `    Member member = memberService.findOne(id);`

@PathVariable is used in conjunction with @RequestMapping URL template.
In this case it is a means to get the category string passed in the method signature.

The @PathVariable param needs to be the same as the param in the @RequestMapping

See Member in MemberSpringMVC

Main Point

- Annotations are metadata that allow the knowledge about the creation of an object to reside with the object itself
- *The self-referral nature of Transcendental Consciousness makes all our actions clearer and more powerful*

Spring MVC Form Tag Library

- Facilitates the development of JSP pages

- Integrated with Spring MVC data binding features

- Each tag provides support for the set of attributes of its corresponding HTML tag counterpart, making the tags familiar and intuitive to use.

- Access Form Tag Library: **See MemberSpringMVC**

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

Form Tag Library

Tag Name	Description
form	Renders an HTML form tag and exposes the binding path to the other form tags.
input	Renders an input element with the default type of text. The type of the input element can be can be (optionally) specified (like email, date etc.) . Note that you can't use radiobutton or checkbox for those types.
password	Renders an input element of type password.
hidden	Renders an input element of type hidden.
select	Renders an HTML select element. The option and/or options tag are used to specify the options to render.
option	Renders a single HTML option element inside a select element.
options	Renders a collection of HTML option elements inside a select element.
radiobutton	Renders an HTML input element of the type radio button.
radiobuttons	Renders multiple HTML input elements of the type radio button.
checkbox	Renders an HTML input element of the type checkbox.
checkboxes	Renders multiple HTML input elements of the type checkbox.
textarea	Renders an HTML Textarea element.
errors	Displays binding and/or validation errors to the user. It can be used to either specify the path for field-specific error messages or to specify an * to display all error messages.
label	Renders a HTML Label and associate it with an input element.
button	Renders an HTML Button element.

Form Tag

- This tag renders an HTML 'form' tag and exposes a **binding path** to inner tags for binding. *All the other tags in this library are nested tags of the form tag.*

- @RequestMapping(value = "/add", method = RequestMethod.*GET*)
- **public String AddProduct(@ModelAttribute("newProduct")**
            Product newProduct) {

```
<form:form  modelAttribute="newProduct" action="add" method="post" >
```

- @RequestMapping(value = "/add", method = RequestMethod.*POST*)
- **public String saveProduct(@Valid @ModelAttribute("newProduct")**
Product productToBeAdded, BindingResult result) {

Form examples with HTML output

```
<form:input id="name" path="name"/>
```

Generated HTML:

`<input id="name" name="name" type="text" value=""/>`

- **When categories is a LIST**

```
<form:select id="category" path="category.id"
items="${categories}" itemValue="id" itemLabel="name" />
```

Generated HTML:

`<select id="category" name="category.id">`

`<option value="1">Computing</option>`

`<option value="2">Travel</option >`

`<option value="3">Health</option >`

`</select>`

NOTE: *path is the "binding Path" defined previously*

General Purpose Spring Tag Library

- Not Spring MVC specific

- Available to any Java Server Page used in the Spring Framework

- Tags for evaluating errors, setting themes and outputting internationalized messages.

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

Spring Tag Library

Tag Name	Description
htmlEscape	Sets the default HTML escape value for the current page. If set, this tag overrides the value set in the defaultHtmlEscape context-parameter.
escapeBody	Escapes the enclosed body.
message	Displays a message with the given code and for the selected locale. (See the section about internationalization (I18N) later in this chapter for more information.)
theme	Uses the variable from the currently selected theme. (See Chapter 9 for more information.)
hasBindErrors	Shows or hides part of the page (or element) based on whether an model attribute has bind (or validation) errors.
nestedPath	Selects a nested path to be used by the bind tag's path. For example, it can be used to bind customer.address to street instead of to customer.address.street.
bind	Binds an attribute of the model (attribute) to the enclosed input element.
transform	Transforms the bound attribute using Spring's type-conversion system. This tag must be used inside a spring:bind tag.
url	Similar to the jstl core URL tag. It is enhanced so the URL can contain URL template parameters.
param	Specifies a parameter and value to be assigned to the URL. This tag is used inside an url tag.
eval	Evaluates a SpEL expression and prints the result or assigns it to a variable.

Message Tag

- Message tag

  Internationalization support through externalization of messages

  Text from MessageSource configured in DispatcherServlet

```
<spring:message code="greeting" text ="Hi" />
```

"code" isn't set or cannot be resolved, "text" will be used as default message.

Also, the spring:message- tag, works with the locale support that comes with Spring. [Internationalization]

Controller return "view"

- View Resolver[s] can simplify view declaration
- For example with the view resolver:

```
<bean id="viewResolver"
class="org.springframework.web.servlet.view.InternalResourceViewResolver
        <property name="prefix" value="/WEB-INF/jsp/"/>
        <property name="suffix" value=".jsp"/>
    </bean>
```

```
 return "ProductForm";
resolves to:
            /WEB-INF/jsp/ProductForm.jsp

The subsequent RequestDispatcher forward is done by
the framework
```

Main Point

The Spring MVC tag library facilitates JSP development with specialized Form tags. *The practice of the TM technique, by structuring the laws of nature in one's life makes everything go more smoothly.*