LESSON 7 SPRING MVC TAG LIBRARY

Do Less and Accomplish More

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:

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

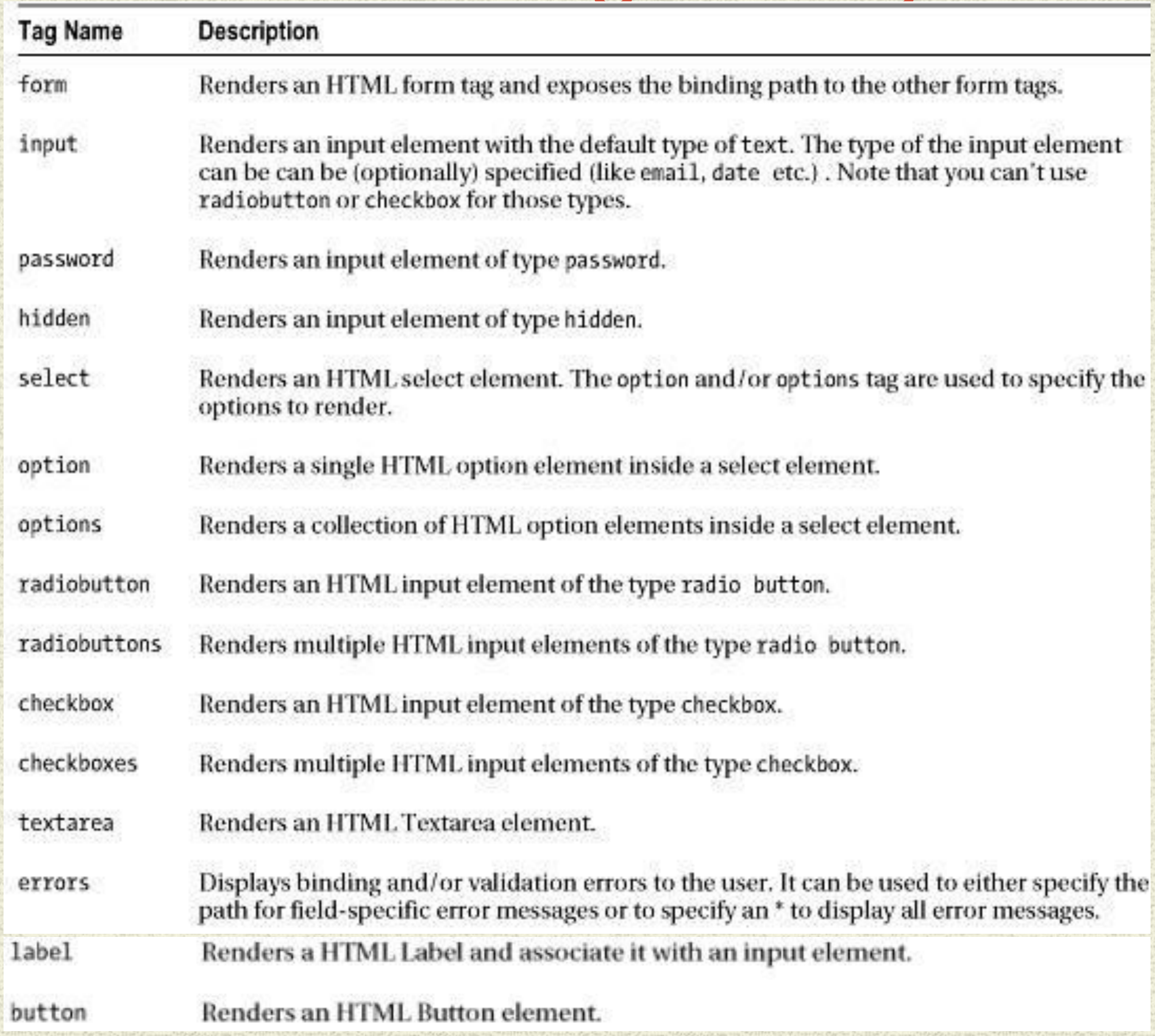
Spring MVC data binding

- 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.

- As a result:
- Data binding makes form beans obsolete
- Any object can be a command AKA a form-backing object
- Can bind directly to **business domain objects**.
- Domain objects exist as POJOs

- **The Spring MVC Form tag library to make it easy to bind form elements to model data with JSP pages.**

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 = "/addBook", method = RequestMethod.GET)
public String inputBook(Model model) {
      model.addAttribute("book", new Book());
```

```
<form:form modelAttribute="book" action="addBook" method="post">
                              OR
<form:form commandName="book" action="addBook" method="post">
```

```
@RequestMapping(value = "/addBook", method = RequestMethod.POST)
public String saveBook(@ModelAttribute Book newBook) {
```

Form Tag

```
@RequestMapping(value = "/add", method = RequestMethod.GET)
public String getAddNewProductForm(@ModelAttribute("newProduct")
Product  product) {
```

No need for Action as long as GET & Post have same URI [HTML 4]

- `<form:form  modelAttribute="newProduct" class="form-horizontal">`

No need for Method as Post is “assumed” [Spring]

```
@RequestMapping(value = "/add", method = RequestMethod.POST)
public String processAddNewProductForm(@ModelAttribute("newProduct")
Product productToBeAdded ) {
```

Model Attribute Scenarios

- `<form:form modelAttribute="book"  action= "addBook" method="post">`
- ***This Works:***
- `public String inputBook( Book book,  Model model) {`
- `public String saveBook( Book book ) {`

This Doesn't Work:

- `public String inputBook(Model model) {`
- **HTTP Status 500 - java.lang.IllegalStateException: Neither BindingResult nor plain target object for bean name 'book' available as request attribute**

- ***THIS WORKS:***
- `public String inputBook( Book book,  Model model) {`
- `public String saveBook( @ModelAttribute("newBook") Book book ) {`

More Scenarios

- `<form:form modelAttribute="newBook"  action= "addBook" method="post">`
-
- ***This Doesn't Work:***
- `public String inputBook(Book book,  Model model) {`
- **HTTP Status 500 - java.lang.IllegalStateException: Neither BindingResult nor plain target object for bean name 'newBook' available as request attribute**

- ***This Works:***

```
 public String inputBook( Book book,  Model model) {
        Book newBook = new Book();
        newBook.setAuthor("FRANK");
        book.setAuthor("Joe");
  AUTHOR is set to "FRANK"
        model.addAttribute("newBook", newBook);
```

Form examples with HTML output

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

Generated HTML:

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

```
<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*

Form examples with HTML output [Cont.]

```
<form:select id="category" path="category.id">
 <form:option value="0" label="--Select Category"/>
 <form:options items="${categories}" itemLabel="name" itemValue= "id"/>
</form:select>
```

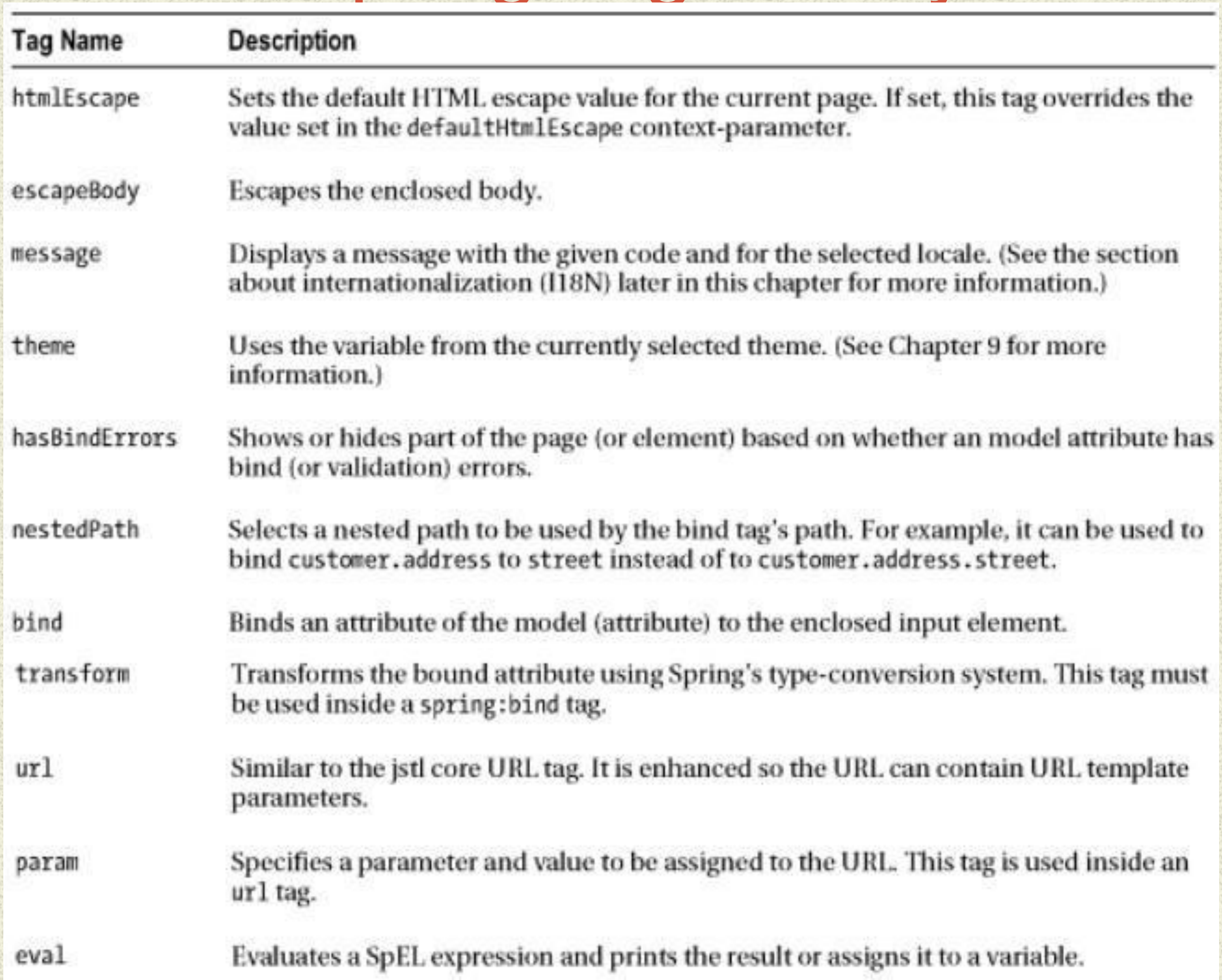
```
  <select id="category" name="category.id">
          < option value="0" selected="selected">--Select Category</ option >
          < option value="1">Computing</ option >
          < option value="2">Travel</ option >
          < option value="3">Health</option>
 </select>
```

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

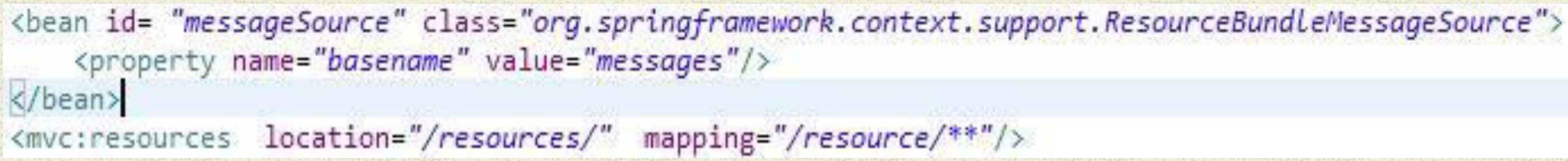
```
<spring:message code="greeting" text ="Hi" />
```

"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.

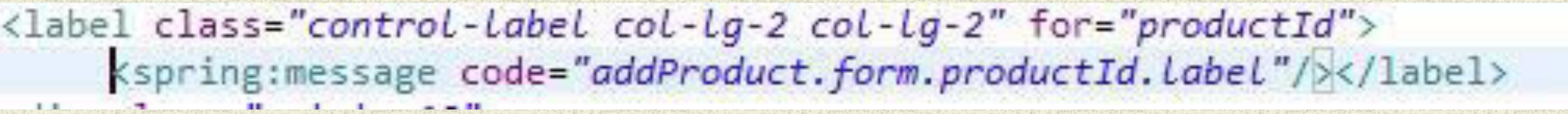
Spring [externalize] message tag example

Dispatcher Servlet configuration file declares message source bean. "messages" file is messages.properties and must reside in the source class path in order to be discovered.



```
<bean id= "messageSource" class="org.springframework.context.support.ResourceBundleMessageSource">
    <property name="basename" value="messages"/>
</bean>
<mvc:resources  location="/resources/"  mapping="/resource/**"/>
```

Spring message tag accesses label name from resource file messages.properties



```
<label class="control-label col-lg-2 col-lg-2" for="productId">
    <spring:message code="addProduct.form.productId.label"/></label>
```

Spring URL Tag

Access static resources:

Resolves the path from context root - irrespective of current URL.

```
<style type="text/css">@import url("<spring:url
                             value="/css/main.css"/>"); </style>
```

- Resolves to:
- `<style type="text/css">@import url("/Book05a/css/main.css");</style>`

Query Parameter passing:

Use spring:param - avoid XSS attacks.

- `<spring:url value="/addBook" var="addBook_url" >`
- `<spring:param name="ISBN" value="1234"/></spring:url>`
- `<a href="${addBook_url}">Add Book</a>`

Resolves to: `<a href="/Book05a/addBook?ISBN=1234">Add Book</a>`

Spring template/@PathVariable:

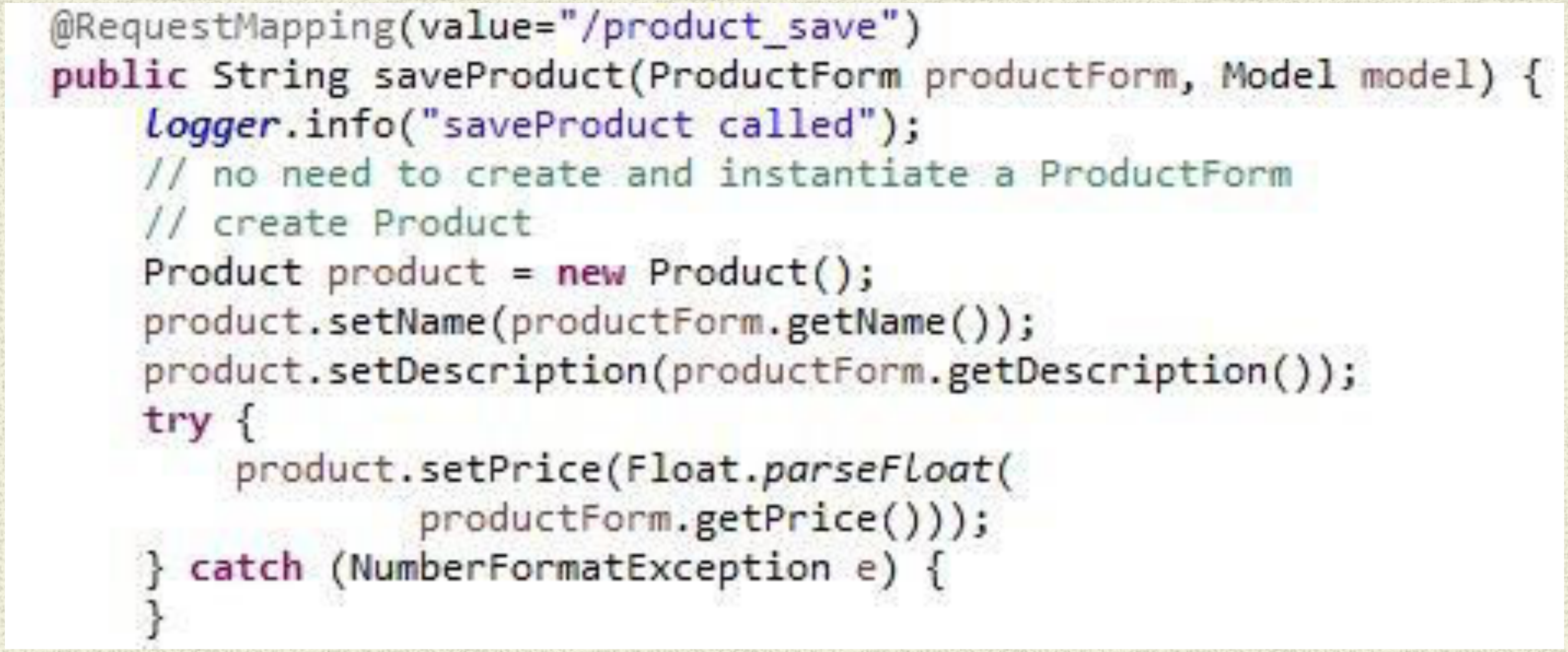
```
        <spring:url value="/book_edit/{id}"  var="edit" >
    <spring:param name="id" value="${book.id}" />
 </spring:url>
        <td><a href="${edit}">Edit</a></td>
```

Resolves to: `<a href="/Book05a/book_edit/2">Edit</a>`

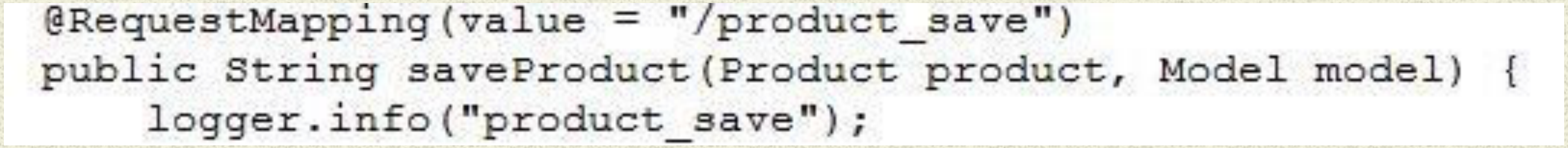
Main Point

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.*

Controller Simplification based on Spring MVC Data binding



```
@RequestMapping(value="/product_save")
public String saveProduct(ProductForm productForm, Model model) {
    logger.info("saveProduct called");
    // no need to create and instantiate a ProductForm
    // create Product
    Product product = new Product();
    product.setName(productForm.getName());
    product.setDescription(productForm.getDescription());
    try {
        product.setPrice(Float.parseFloat(
                productForm.getPrice()));
    } catch (NumberFormatException e) {
    }
```



```
@RequestMapping(value = "/product_save")
public String saveProduct(Product product, Model model) {
    logger.info("product_save");
```

Excluding Fields from Data Binding

- We don’t always want to bind our ENTIRE domain object
- For example, an internal Customer or Product ID
- Need to explicitly restrict binding on specific fields
- WebDataBinder object used by the binding function
- @initBinder annotation identifies Controller method that accesses WebDataBinder
- @InitBinder
- **public void initialiseBinder(WebDataBinder binder) {**
- binder.setDisallowedFields(“id”);
- binder.setRequiredFields(“name”,”description”,”price”);

Custom Data Binding

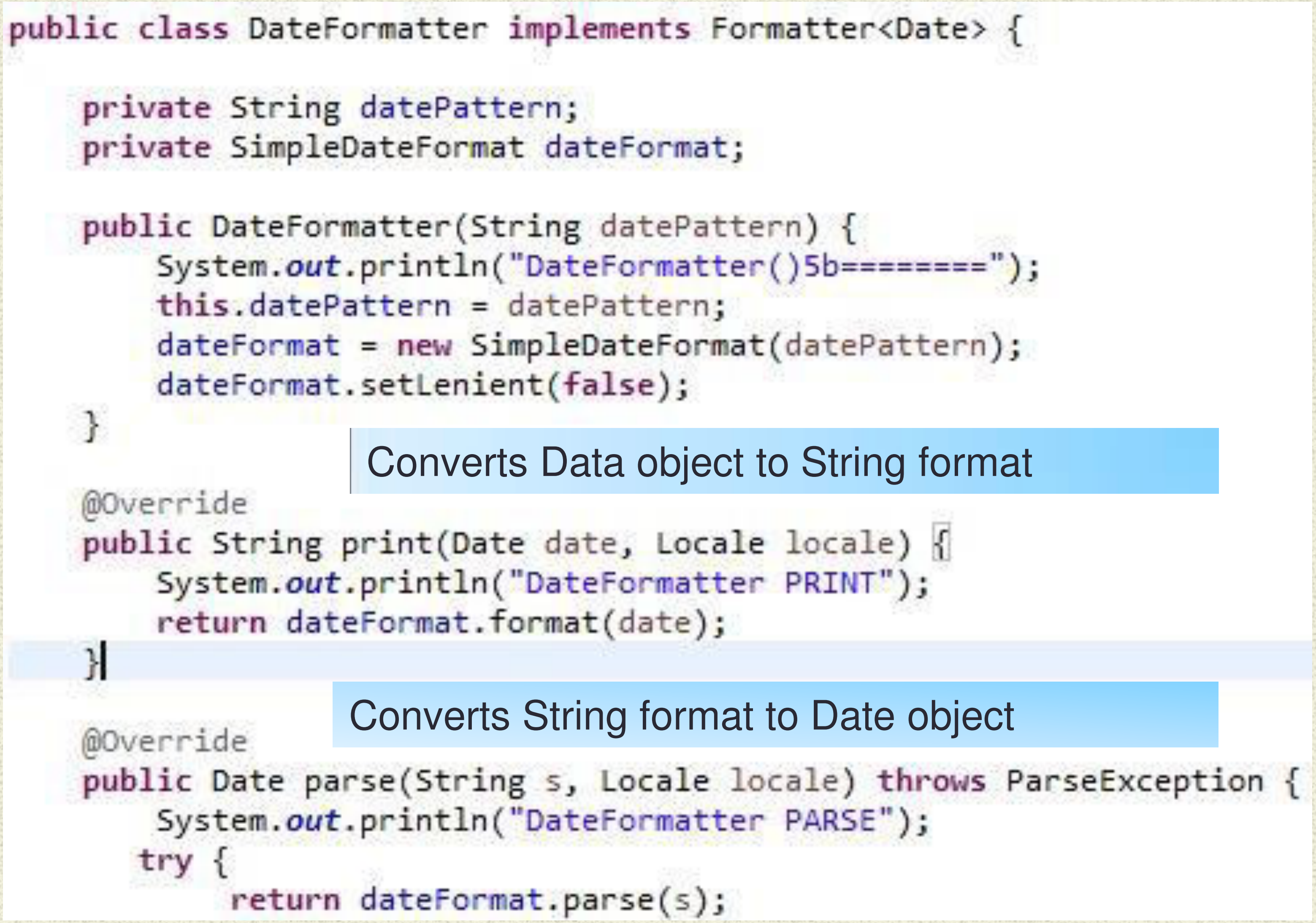
- Built-in Data Binding handles simple String to data type conversions
- Custom Binding is needed to handle more complex cases
- Three Options in Spring MVC:
  - **Custom PropertyEditor**
    - Old-style [“retro”] ,heavyweight based on entire java.beans package
  - **Converter**
  - General-purpose type conversion system introduced Spring 3.0. used internally by Spring. One way conversion, locale agnostic
  - **Formatter**
  - Designed for Spring MVC form conversion. 2 way conversion - to & from String. Locale aware. More lightweight that Converter.

Out-of-the-box Spring Formatters

- The Number Package:
  NumberFormatter,
  CurrencyFormatter
  PercentFormatter
- The DateTime Package:
- DateFormatter API

- Custom Example:
  ISBN Number

DateFormatter Code



```
public class DateFormatter implements Formatter<Date> {

    private String datePattern;
    private SimpleDateFormat dateFormat;

    public DateFormatter(String datePattern) {
        System.out.println("DateFormatter()5b=========");
        this.datePattern = datePattern;
        dateFormat = new SimpleDateFormat(datePattern);
        dateFormat.setLenient(false);
    }
```

Converts Data object to String format

```
    @Override
    public String print(Date date, Locale locale) {
        System.out.println("DateFormatter PRINT");
        return dateFormat.format(date);
    }
```

Converts String format to Date object

```
    @Override
    public Date parse(String s, Locale locale) throws ParseException {
        System.out.println("DateFormatter PARSE");
      try {
          return dateFormat.parse(s);
```

ISBN Formatter Example

```
public class ISBNFormatter implements Formatter<ISBNumber> {

   public String print(ISBNumber isbn, Locale locale) {
         return isbn.getStart() + "-" +
                              isbn.getMiddle() + "-" + isbn.getEnd();
   }

   public ISBNumber parse(String source, Locale locale)
       throws ParseException {
      int start = Integer.parseInt(source.substring(0, 3));
     int middle = Integer.parseInt(source.substring(4, 7));
     int end = Integer.parseInt(source.substring(8, 11));
     return new ISBNumber(start, middle, end);
   }
}
```

Formatter Configuration

```
 <mvc:annotation-driven conversion-service="conversionService"/>

<bean id="conversionService“ class=
    "org.springframework.format.support.FormattingConversionServiceFactoryBean">
•     <property name="formatters">
         <set>
          <bean class="mum.edu.formatter.ISBNFormatter"> </bean>
         </set>
    </property>
</bean>
```

Main Point

- Spring Data binding automatically binds form elements to Model data. Spring MVC tag library makes this easy with JSP Form tags.
- *The practice of the TM technique, in a simple automatic way structures communication between outer and inner values enlivening all aspects of life.*

@ModelAttribute

- **Can be placed on a method parameter:**
  - @RequestMapping(value="/owners/pets", method = RequestMethod.POST)
  - public String processSubmit(@ModelAttribute Pet pet) { }

  The Object should be retrieved from the model or instantiated if doesn't exist. The Object fields should be populated from all request parameters that have matching names.

- **Can be placed on method. Method invoked before methods annotated with @RequestMapping**
- @ModelAttribute

```
public Account addAccount(@RequestParam String number)  {
          return accountManager.findAccount(number); }
```

Object is added to Model – in this example the Account object is added

@RequestMapping Template with @PathVariable

```
@RequestMapping(value = "/book_edit/{id}")
    public String editBook(Model model, @PathVariable(“id) long id) {
            . . .
        Book book = bookService.get(id);
```

- @PathVariable is used in conjunction with @RequestMapping URL template.

It is somewhat like a @RequestParam EXCEPT it is part of the URL path..

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

CONTROLLER METHOD ARGUMENTS

- Map Model/ModelMap
- Command/form object [optional @ModelAttribute]
- RedirectAttributes
- SessionStatus
- BindingResult   Validation
- @RequestParam
- @RequestHeader
- @RequestBody   RESTful Services
- @ResponseBody   RESTful Services
- @PathVariable   Template
- HttpServletRequest HttpServletResponse HttpSession
- java.util.Locale
- java.security.Principal

Controller Method Return Types

1. **ModelAndView** object,
2. **Model** object, with the view name implicitly determined through a **RequestToViewNameTranslator**
3. **Map** object for exposing a model, the view name implicitly determined through a **RequestToViewNameTranslator**
4. **String** value interpreted as the logical view name, the model implicitly determined through command objects
5. **void** if the method handles the response itself (by writing the response content directly, declaring an argument of type ServletResponse / HttpServletResponse for that purpose) or if the view name is supposed to be implicitly determined through a **RequestToViewNameTranslator**

RequestToViewNameTranslator – basically uses the URL from the @RequestMapping

More Model, ModelMap, ModelAndView

- Model is an interface while ModelMap is a class.
- Model has method asMap to get actual map.
- ModelMap is a class that is a custom[convenience] Map implementation that automatically generates a key for an object when an object is added to it.
- ModelAndView is just a container for both a ModelMap and a view object. It allows a controller to return both as a single value.

Main point

Spring MVC is “Open for extension, closed for modification”.

Spring provides a myriad of opportunities to change the behavior of an application based on the framework. *Likewise, Pure Consciousness offers infinite variety & possibilities. They both represent good design.*

CONTROLLER METHOD ARGUMENTS [Continued]

- org.springframework.web.context.request.WebRequest or org.springframework.web.context.request.NativeWebRequest.

- java.io.InputStream / java.io.Reader
- java.io.OutputStream / java.io.Writer

- @RequestPart
- HttpEntity
- org.springframework.web.util.UriComponentsBuilder