VI. MVC Controllers: Using Annotations

171

Spring MVC Annotation Types ...

- The spring-context schema

  xmlns:context="http://www.springframework.org/schema/context"

- A <component-scan/> element

  context:component-scan base-package="basePackage"/>

  - Integrating in configuration file

    base-package="com.example.controller"

- A RequestMapping-annotated method

  @RequestMapping(value = "/customer_input")

173

Notes:

Writing Request-Handling Methods

- Method arguments

```
myOtherMethod(HttpServletRequest request,
    Locale locale
```

- Argument types

```
org.springframework.ui.Model
```

- Every time a request-handling method is invoked, Spring MVC creates a Model object and populates its Map with potentially various objects.

175

Notes:

Using An Annotation-Based Controller ...

- Controller Class

  @Controller

  value="/product_input"

  value="/product_save"

  model.addAttribute("product", product);

  - The main purpose of having a Model is for adding attributes that will be displayed in the view.
    - In this example, a Product instance was added by calling model.addAttribute

177

Notes:

Dependency Injection

- In order for a dependency to be found, its class must be annotated with **@Service**.

```
@Service
public class ProductServiceImpl implements ProductService
```

179

Notes:

Dependency Injection ...

- The Spring MVC configuration file has two <component-scan/> elements, one for scanning controller classes and one for scanning service classes.

```
<context:component-scan base-package="moose.controller"/>
<context:component-scan base-package="moose.service"/>
```

181

Notes:

Redirect and Flash Attributes ...

- With a forward, attributes can be added to the Model object and the attributes will be accessible to the view.
  - Spring version 3.1 and later provide a way of preserving values in a redirect by using flash attributes.
- Must add a new argument of type org.springframework.web.servlet.mvc.support.RedirectAttributes in the method.

183

Notes:

Request Parameters and Path Variables

- Both request parameters and path variables are used to send values to the server.
  - Both are also part of a URL.
  - The request parameter takes the form of key=value pairs separated by an ampersand.

/product_retrieve?productId=3

185

Notes:

@ModelAttribute

- The ModelAttribute annotation type can be used to decorate a Model instance in a method.
- Instance of it retrieved or created and added to the Model object if the method body does not do it explicitly.
  - Spring MVC will create an instance of **Order** every time this **submitOrder** method is invoked.

```
@RequestMapping(method = RequestMethod.POST)
public String submitOrder(@ModelAttribute("newOrder") Order order,
    Model model) {

  ...
}
```

187

Notes:

VII. Bindings and Tags

189

Data Binding Overview ...

```
@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) {
   }
```

191

Notes:

The Form Tag Library

- A taglib needs to be added to the page:

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

- The commandName attribute is probably the most important attribute as it specifies the name of the model attribute that contains a backing object whose properties will be used to populate the generated form.
  - If this attribute is present, the corresponding model attribute must be added in the request-handling method that returns the view containing this form.
- The following form tag is specified in the **BookAddForm.jsp**.

```
<form:form commandName="book" action="book_save" method="post">
  ...
</form:form>
```

193

Notes:

Data Binding Example

- The application will list books, add a new book, and edit a book.
  - The **Book** class and the **Category** class are the domain classes.
  - The tags from the form tag library are used.

195

Notes:

Data Binding Example ...

The **BookAddForm.jsp** page:

```
<%@ taglib prefix="form"
      uri="http://www.springframework.org/tags/form" %>
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<!DOCTYPE HTML>
<html>
<head>
<title>Add Book Form</title>
<style type="text/css">@import url("<c:url value="/css/main.css"/>");</style>
</head>
<body>

<div id="global">
<form:form commandName="book" action="book_save" method="post">
  <fieldset>
    <legend>Add a book</legend>
    <p>
      <label for="category">Category: </label>
      <form:select id="category" path="category.id"
        items="${categories}" itemLabel="name"
        itemValue="id"/>
    </p>
    <p>
      <label for="title">Title: </label>
      <form:input id="title" path="title"/>
    </p>
    <p>
      <label for="isbn">ISBN: </label>
      <form:input id="isbn" path="isbn"/>
    </p>

    <p id="buttons">
      <input id="reset" type="reset" tabindex="4">
      <input id="submit" type="submit" tabindex="5"
        value="Add Book">
    </p>
  </fieldset>
</form:form>
</div>
</body>
</html>
```

197

Notes:

@ ModelAttribute

VIII. AspectJ and AOP Principles

Why Aspect Oriented Programming?

- Problem:
  - Just rolled out a new, flagship, multimillion dollar enterprise application
    - New requirements arise that it must now support

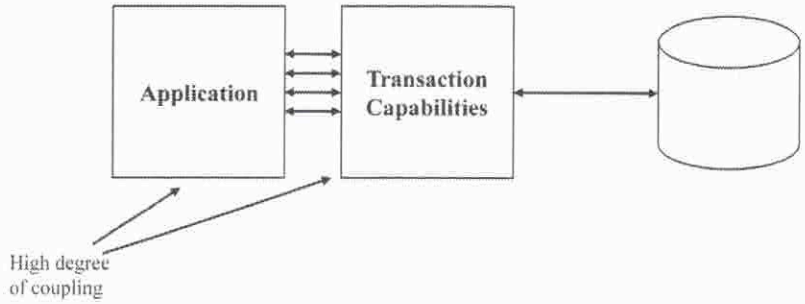
- What can you do?
  1) Jump ship now!
  2) Get back to work refactoring countless classes to incorporate these capabilities
  3) Refactor application adding these features in by "weaving" them externally

201

Notes:

Why Aspect Oriented Programming? ...

- AOP is about avoiding the high cost of refactoring applications when requirements change over time



203

Notes:

To incorporate transaction logic into the application, there will be many points throughout the code where transaction-based objects will be invoked. These points cause the dependency on those objects to rise. This makes it difficult to remove those transaction-based objects later on if they are no longer needed.

Aspect Oriented Programming

- Bring us the module!
  - In an AOP application, not only are the core components modularized (as they tend to be in J2EE apps), but *the secondary systems will also be modularized*

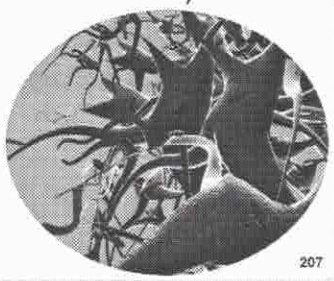
First you must understand AOP terminology!...

205

Notes:

Core vs. Crosscutting Concerns ...

- ***Crosscutting Concerns:***
  - Requirements (subsystems) that interact with several modules within a system
    - *Example: The Human Body*
      - Core concerns are represented by the heart, head, legs, arms. *Crosscutting concerns* include the nervous system.
      - Our nerves represent a "tangling" of the core components with the nervous system.



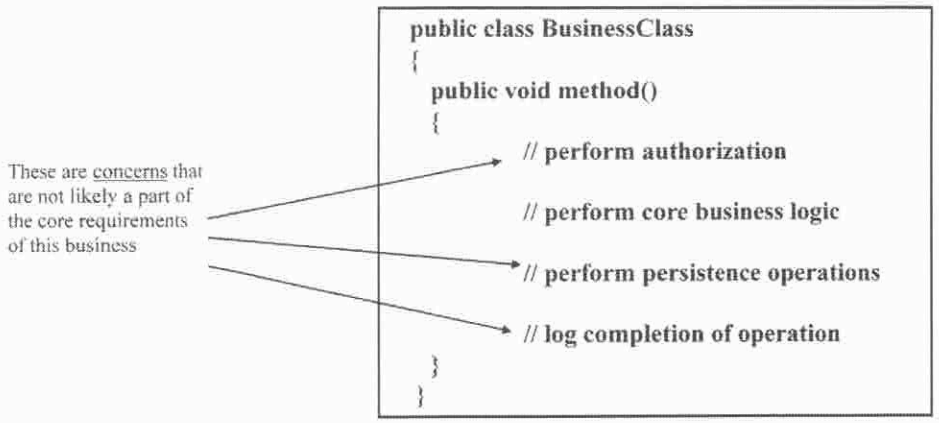
207

Notes:

Crosscutting concerns are those concerns that are utilized (referred to) by the core concerns. Core concerns tend to make up the primary, business logic.

Tangling Concerns

- Code Example illustrating Crosscutting Concerns mixed with Core Concerns (***tangling***)



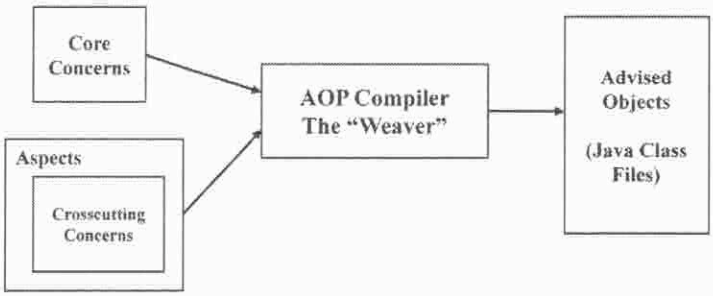
209

Notes:

When various concerns are found implemented within the code of the core components, this is known as **tangling concerns**.

Bringing Concerns Together

- *Core concerns* must eventually implement *crosscutting concerns*
  - In AOP, the concerns are first built separately
  - Then a special compiler, called a "*weaver*" is used



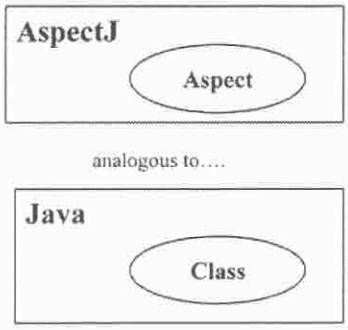
211

Notes:

Aspects Are Like Classes

- Aspects
  - Modules that identify the rules for "weaving" crosscutting concerns into the system

The structure of an AspectJ aspect is similar to that of a class



213

Notes:

AspectJ is the most well-known implementation of Aspect-Oriented Programming Project for Java. It can be freely obtained from http://eclipse.org/aspectj

Spring AOP vs. AspectJ AOP

- ***Spring AOP*** represents a limited form of AOP
  - Only a few types of **Join Points** are supported
  - Spring AOP is *invoked at runtime*
  - The runtime advised object is a *proxy object* not a Java class file
- ***AspectJ*** *has a compiler called* ***ajc***
  - It *runs at compile-time*
  - Very powerful
  - Supports annotation-based declarations and aspect class structures
  - *Pointcut* and *advice* language is somewhat complex

215

Notes:

A Simple Example

```
public class DayOfWeekService
{
    public String getDayOfWeek(int year, int month, int day)
    {
        Calendar c = new GregorianCalendar(year, month, day);
        return new SimpleDateFormat("EEEE").format(c.getTime());
    }
}
```

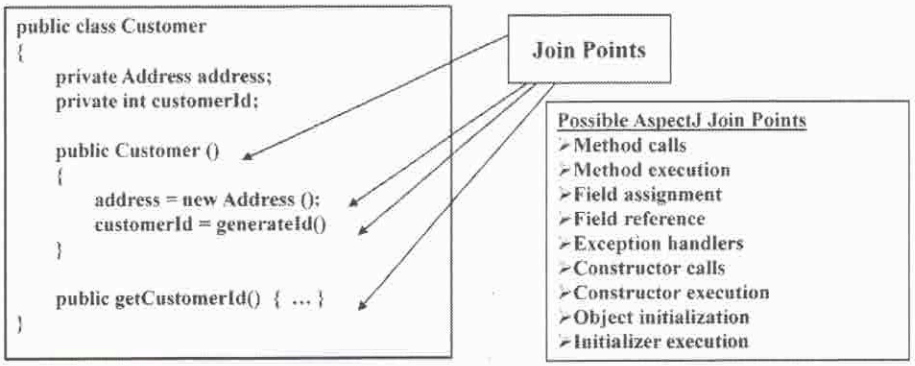
```
public class Driver
{
    public static void main(String[] args)
    {
        String dayOfWeek =
            new DayOfWeekService().getDayOfWeek(2009, 9, 16);
        System.out.println(dayOfWeek);
    }
}
```

217

Notes:

Join Points

- *Join Points* are locations within source code where aspects may specify behavior
  - Points within *core concerns* where the ***weaver*** can modify your code



219

Notes:

In Spring AOP, only the execution of methods can be join points.

Pointcut Designators

- Defines what "*activities*" the AOP compiler should look for in the source
- Types of designators (activities) include

```
call( method signature )
execution( method signature )
get( method signature )
within( method signature )
target( type name )
this( type name )
cflow( join point designator )
handler( exception name )
```

- Example

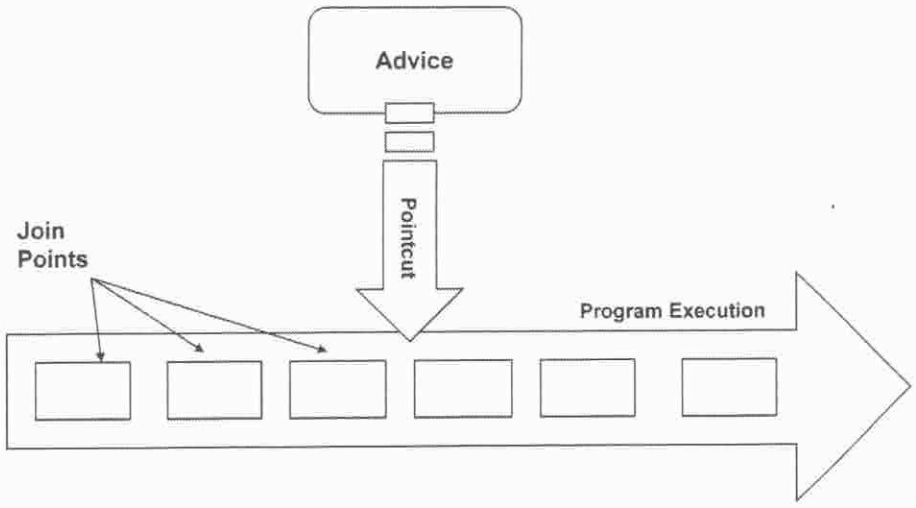
```
pointcut call_GetDayOfWeekCall( ) :
                    call(* DayOfWeekService.getDayOfWeek(..));
```

221

Notes:

This example identifies a pointcut called 'call_GetDayOfWeekCall'. It tells the AOP compiler to identify any call to methods called *getDayOfWeek()* within the DayOfWeekService class that accept any number of parameters and return any types.

Applying Advice



223

Notes:

Summary

- AOP provides a means for easily inserting new requirements into an application
  - *Crosscutting concerns* are great examples where AOP can improve development lifecycles
  - AOP has **two** learning curves:
    - Numerous new terms, such as *Pointcuts*, *Advice*, and *Introductions*
    - A tricky syntax for specifying *pointcut designators*

225

Notes: