LESSON 6 SPRING MVC VALIDATION

Avoid the Danger that has not yet come

Spring Validation

Validation should not be tied to the web tier,

should be easy to localize

should be possible to plug in any validator available.

Spring Validation uses a Validator interface that is basic and usable in every layer of an application.

An application can choose to enable Bean Validation (JSR-303[349]) and the corresponding annotations for all validation needs.

An application also can use the Spring Validator directly without the use of annotations.

Form Validation through Annotation

To do simple validation, use javax.validation.constraints annotations also known as JSR-303 annotations.

JSR-303 is also called the Bean Validation API

JSR-303 provider library, e.g., [Hibernate-Validator.jar]

Example Usage

In the Controller - activated by annotating model to be validated method signature with @Valid:

```
Public String save(@Valid @ModelAttribute(“employee”) User user, BindingResult result) {

        if (result.hasErrors()) {
                return addEmployeeForm;
```

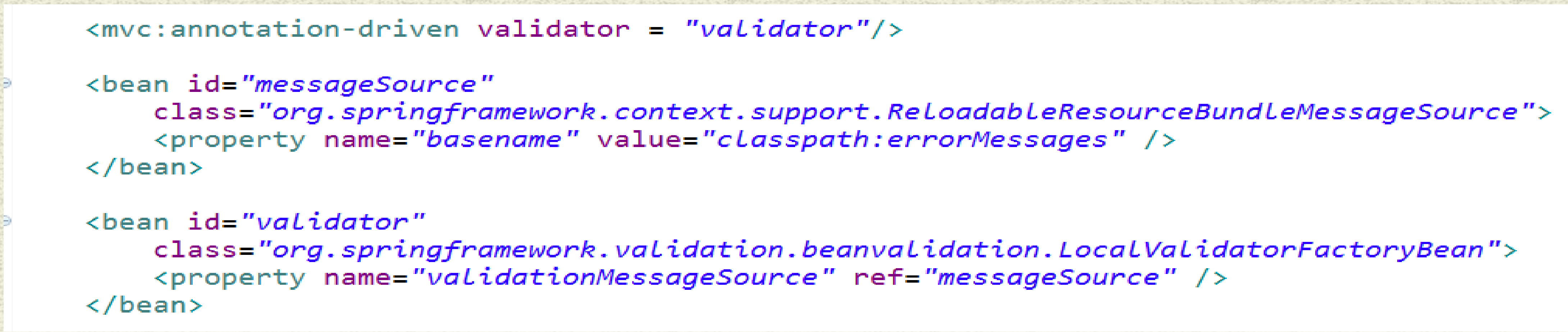
BindingResult IMMEDIATELY after model attribute

In the Domain Model annotate properties as necessary:

```
@Size(min=4, max=50, message="{Size.name.validation}")
private String firstName;
```

Externalize error messages in properties file…

Configure Validator & External error message file



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

<bean id="messageSource"
    class="org.springframework.context.support.ReloadableResourceBundleMessageSource">
    <property name="basename" value="classpath:errorMessages" />
</bean>

<bean id="validator"
    class="org.springframework.validation.beanvalidation.LocalValidatorFactoryBean">
    <property name="validationMessageSource" ref="messageSource" />
</bean>
```

Properties File - Example Usage

In Domain Model:

```
@NotEmpty @Size(min=4, max=50, message="{Size.name.validation}")
private String firstName;
```

In errorMessages.properties:

Size.name.validation = Invalid product name. It should be minimum 4 characters to maximum 50 characters long.

Display errors in View

Show ALL errors on Page

Show field level error

```
<form:form modelAttribute="employee" action="employee_save" method="post">
    <fieldset>
        <legend>Add an employee</legend>
  <p>
       <form:errors path="*" cssStyle="color : red;" />
  </p>
  <p>
       <label for="firstName">First Name: </label>
       <form:input path="firstName" />

        <div style="text-align: center;">
                <form:errors path="firstName" cssStyle="color : red;" />
        </div>
  </p>
```

Validation Property Annotations [JSR-303]

Constraint	Description	Example
@AssertFalse	The value of the field or property must be false.	@AssertFalse boolean isUnsupported;
@AssertTrue	The value of the field or property must be true.	@AssertTrue boolean isActive;
@DecimalMax	The value of the field or property must be a decimal <= the value.	@DecimalMax("30.00") BigDecimal discount;
@DecimalMin	The value of the field or property must be a decimal >= the value.	@DecimalMin("5.00") BigDecimal discount;
@Digits	The value of the field or property must be a number within a specified range.	@Digits(integer=6, fraction=2) BigDecimal price;
@Future	The value of the field or property must be a date in the future.	@Future Date eventDate;
@Max	The value of the field or property must be an integer >= the value.	@Max(10) int quantity;
@Min	The value of the field or property must be an integer <= the value.	@Min(5) int quantity;
@NotNull	The value of the field or property must not be null.	@NotNull String username;
@Null	The value of the field or property must be null.	@Null String unusedString;
@Past	The value of the field or property must be a date in the past.	@Past Date birthday;
@Pattern	The value of the field or property must match the regular expression defined in the regexp element.	@Pattern(regexp="\\(\\d{3}\\)\\d{3}-\\d{4}") String phoneNumber;
@Size	The size of the field or property is evaluated and must match the specified boundaries. Can pertain to String, Collection, Map…	@Size(min=2, max=240) String briefMessage;

Hibernate JSR 303 Annotations

It's for Strings and collections.

Domain object annotations

```
@NotEmpty @Size(min=4, max=50, message="{Size.name.validation}")
private String firstName;
@NotEmpty(message="Enter the last name")
private String lastName;
@NotNull(message=“Birth Date required")

private Date birthDate;
@Valid
private Address address;
                 Address.java:
@NotEmpty(message=“{Street.empty}")
private String street;
@Size(min=2, max=2, message="Size.state")
private String state;
```

use for Objects

Note: Curly {} brackets ensure that the text will be used as a property file lookup

Errormessages.properties entries

```
NotEmpty= {0} field must have a value
Street.empty = {0} must have value
Size.state = State must have two  characters
Size.name.validation= Size of the {0} must be between {2} and {1}

Pattern.zipcode= {0} is incorrect. Use format nnnnn-nnnn

firstName = First Name
```

NOTE:

Spring organizes “placeholders” in alphabetical order. @Size(min=1,max=5), field name as {0} , the max value as {1} , and the min value as {2}

Typemismatch

Non-String – if value cannot be converted to the data-type then a TypeMismatchException is thrown.

Spring will attempt to resolve the error message based on property file entries

Define the error message for type mismatch [e.g.]:

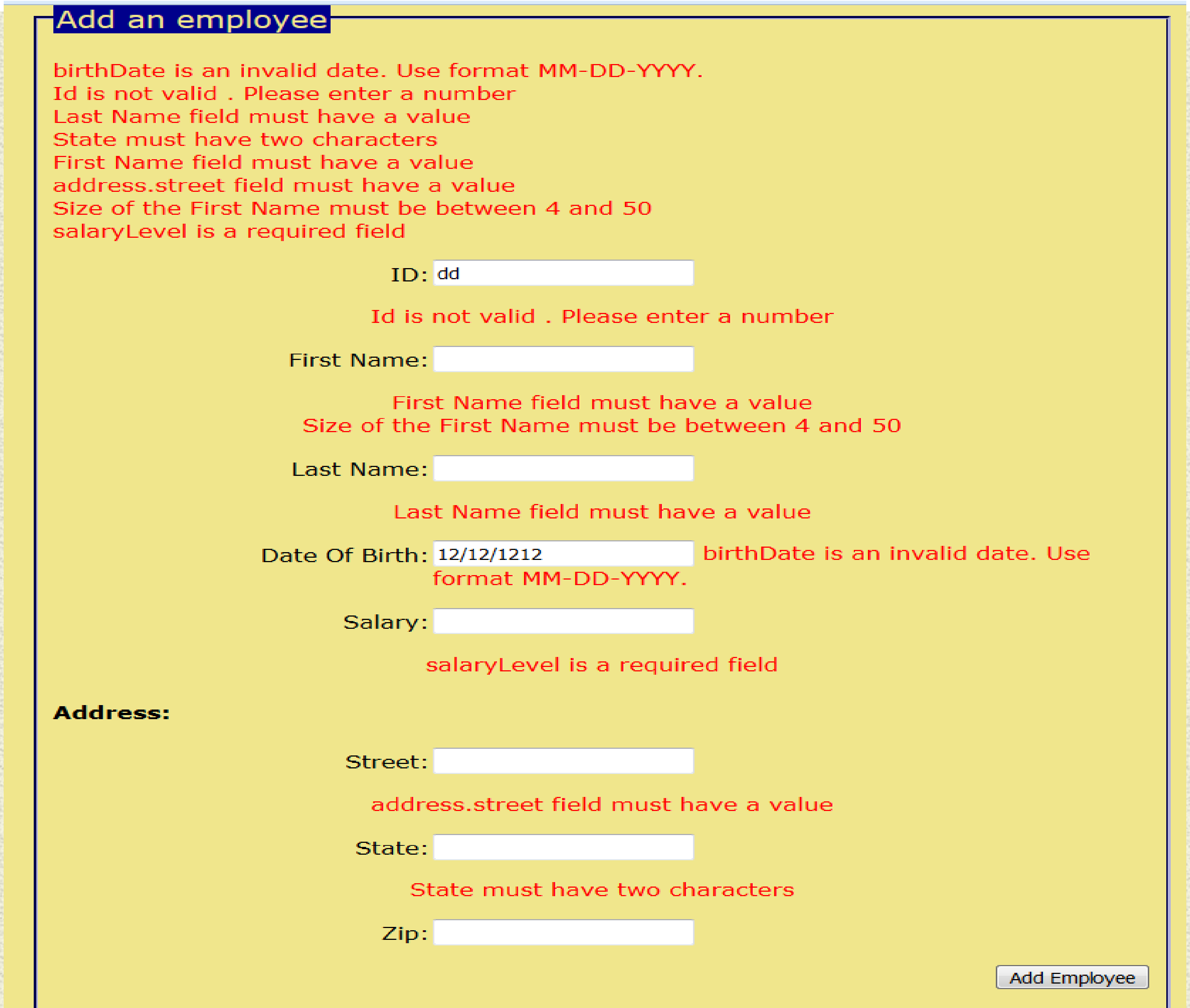
```
typeMismatch.long="{0}" must be a long.
```

```
typeMismatch.java.lang.Integer="{0}" must be an integer.
typeMismatch.java.lang.Double="{0}" must be a double.
typeMismatch.java.lang.Long="{0}" must be a long.
typeMismatch.java.util.Date="{0}" is not a date.
```

Field Specific:

```
typeMismatch.id= Id is not valid.Please enter a number
```

DEMO:



Add an employee

birthDate is an invalid date. Use format MM-DD-YYYY.
Id is not valid . Please enter a number
Last Name field must have a value
State must have two characters
First Name field must have a value
address.street field must have a value
Size of the First Name must be between 4 and 50
salaryLevel is a required field

ID: dd

Id is not valid . Please enter a number

First Name:

First Name field must have a value
Size of the First Name must be between 4 and 50

Last Name:

Last Name field must have a value

Date Of Birth: 12/12/1212 birthDate is an invalid date. Use format MM-DD-YYYY.

Salary:

salaryLevel is a required field

Address:

Street:

address.street field must have a value

State:

State must have two characters

Zip:

Hibernate “specific” Annotations

[Some] Hibernate Annotations that are NOT part of JSR 303

@Range	-- Combines @Min & @Max
@NotEmpty	-- Combines @NotNull & @Size [>= 1]
@Email	-- Tests for Well-formed Email address

Hibernate Specific Validation Annotations

Main Point

Validation checks the correctness of data against business rules. This prevents problems in the business model from arising.

In Cosmic Consciousness, validation of correctness is spontaneous so that life is lived stress- free; problem-free

Manual Validation [W/O Annotations]

Object Validator implements Validator interface.

```
public class MemberValidator implements Validator {
Import org.springframework.validation.Errors

. . .
@Override
public void validate(Object command, Errors errors) {
   ValidationUtils.rejectIfEmptyOrWhitespace(errors,
                          "firstName", "Member.firstname.empty");
   ValidationUtils.rejectIfEmptyOrWhitespace(errors,
                           "LastName", "Member.lastname.empty");
   Member member = (Member)command;
   if( member.getMemberNumber()== null || member.getMemberNumber()<= 0)
       errors.rejectValue("memberNumber","Member.Number.lessthan");
   if( member.getAge() <  18)
       errors.rejectValue("age","Member.age");
 }
```

See webstore07 Demo

Manual Validation[Cont.]

InitBinder setting of validator can be used with @Valid

```
@InitBinder
    protected void initBinder(WebDataBinder binder) {
        binder.setValidator(new MemberValidator());
    }
```

100% Manual Does NOT use @Valid; Looks like this:

```
public String processAddNewMemberForm(@ModelAttribute("newMember")
              Member memberToBeAdded, BindingResult result) {

MemberValidator memberValidator= new MemberValidator();
 memberValidator.validate(memberToBeAdded, result);

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

Custom Validation Annotation

The annotation implementation must conform to Bean Validation API [JSR 303]

There are three steps that are required:

- Define a default error message
- Create a constraint annotation
- Implement a validator

Step 1
Define Default Error Message

Put message in appropriate message properties file

EXAMPLE:

com.packt.webstore.validator.ProductId.message =

A product already exists with this product id.

Step 2 Create the annotation

@Target Indicates the kinds of program element to which an annotation type is applicable.

@Retention Indicates how long annotations with the annotated type are to be retained.

@Constraint Specifies the validator to be used

```
@Target( { METHOD, FIELD, ANNOTATION_TYPE })
@Retention(RUNTIME)
@Constraint(validatedBy = ProductIdValidator.class)

public @interface ProductId {

   String message() default  {com.packt.webstore.validator.ProductId.message}“;

    Class<?>[] groups() default {};

    public abstract Class<? extends Payload>[] payload() default {};
```

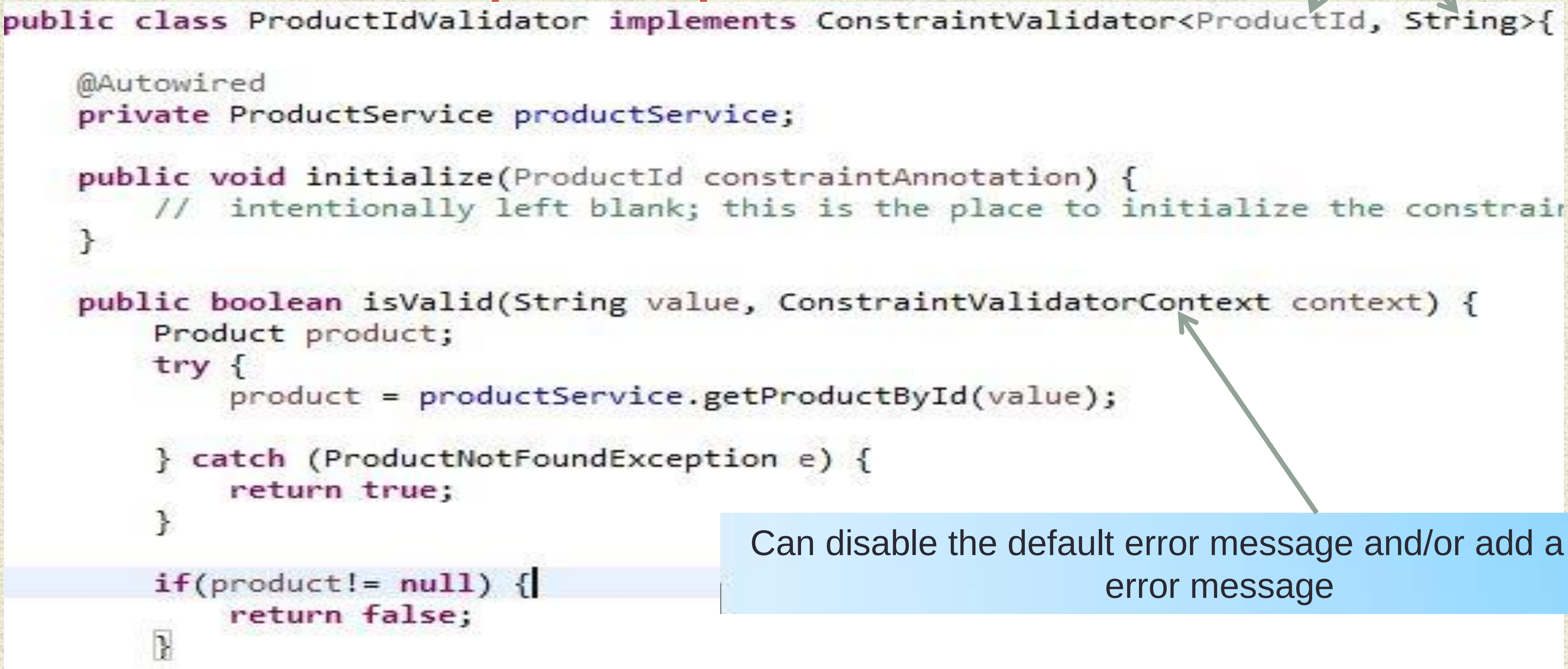
Identifies the default key for creating error messages

Allows assignment of validation groups

Optional custom payload objects assigned to a constraint.

Annotation & Type to be validated

Step 3 Implement Validator



```
public class ProductIdValidator implements ConstraintValidator<ProductId, String>{

    @Autowired
    private ProductService productService;

    public void initialize(ProductId constraintAnnotation) {
        //  intentionally left blank; this is the place to initialize the constrai
    }

    public boolean isValid(String value, ConstraintValidatorContext context) {
        Product product;
        try {
            product = productService.getProductById(value);

        } catch (ProductNotFoundException e) {
            return true;
        }

        if(product!= null) {
            return false;
        }
```

Can disable the default error message and/or add a custom error message

Usage in Domain Model class:

```
@ProductId
private String productId;
```

Cross Field Validation

USE CASE: validate the combination of two or more fields

Similar BUT different to field level Validator

Class Level…Validation against entire Class object

```
public class StockMaximumValidator
           implements ConstraintValidator<StockMaximum, Product> {

 @Override
public boolean isValid(Product product,final ConstraintValidatorContext context){
    BigDecimal unitPrice;
    Long unitsInStock;

    unitsInStock = product.getUnitsInStock();
    unitPrice = product.getUnitPrice();

    BigDecimal currentValue = new BigDecimal(0);
    if (unitsInStock > 0 )
        currentValue = unitPrice.multiply(new BigDecimal(unitsInStock));

    if (currentValue.compareTo(maxValue) >= 0) return false;

    return true;
```

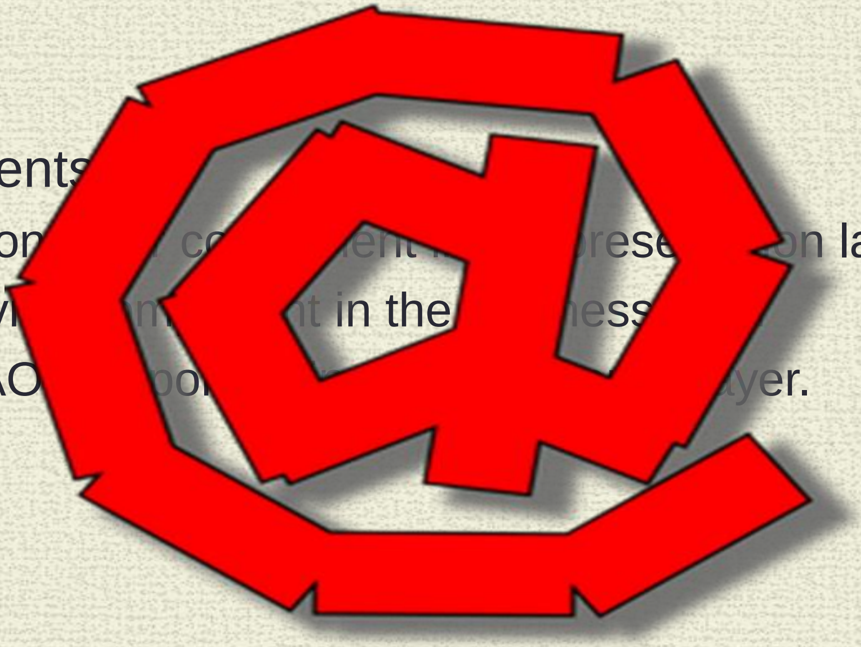
Main Point

- Custom validation allows for handling more complex, extraordinary verification issues.

- *A quality of Cosmic Consciousness is the ability to know what is right in every situation and to handle every situation with maximum effectiveness.*

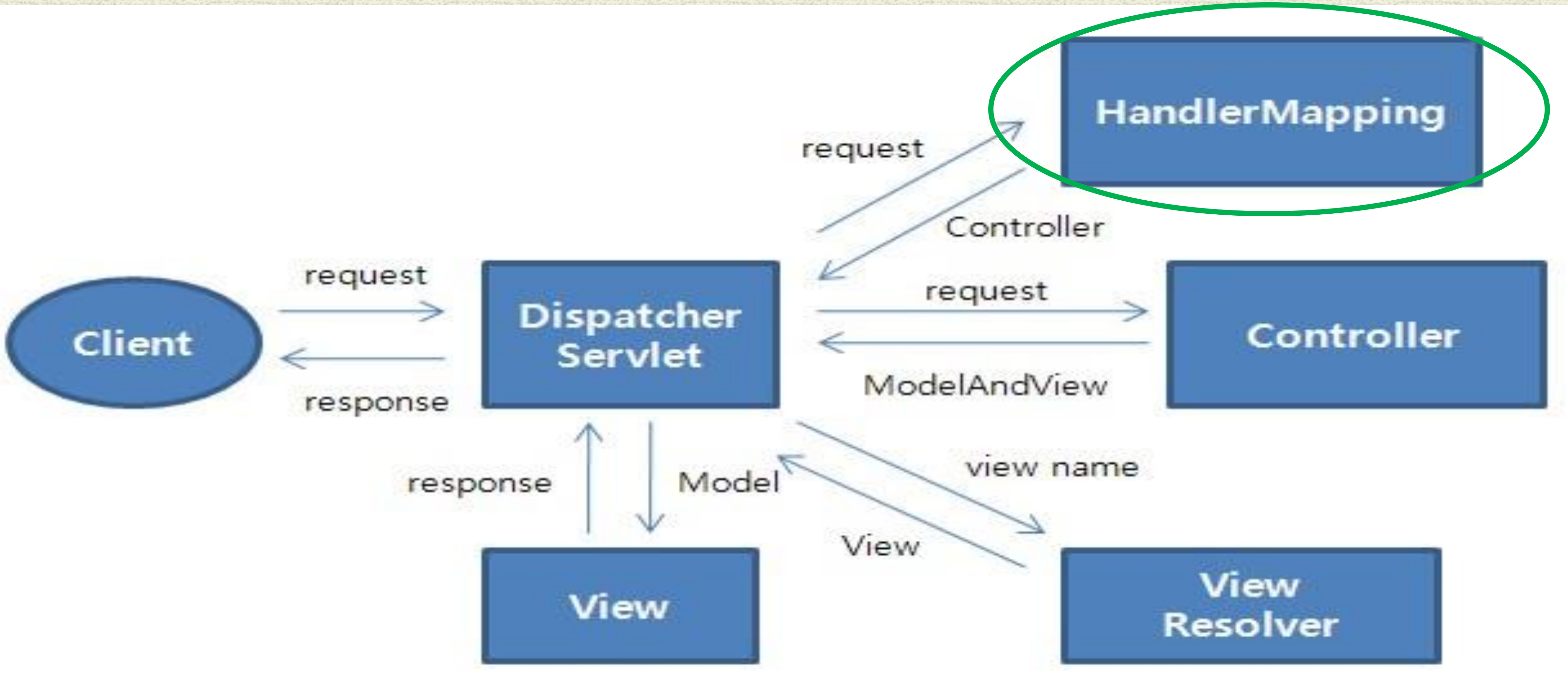
Spring MVC Architecture & Annotations

- ***Handler Mapping***
- Spring Annotations
  - Spring Managed Components
    - @Controller Indicates a Con co ent rese on layer.
    - @Service Indicates a Servi m t in the ness
    - @Repository Indicates DAO bo yer.
  - @RequestMapping
  - @RequestParam
  - @ModelAttribute
  - @PathVariable



- ViewResolvers
- Views

Spring MVC Flow



Handler Mapping

The Handler Mapping is used to map a request from the Client to its Controller object by searching through the various Controllers

BeanNameUrlHandlerMapping *******default******

<bean name="/ProductForm“ class=“edu.mum.controller.InputProductController"/>

The URL of the Client is directly mapped to the Controller

RequestMappingHandlerMapping *******default******

Maps handlers through the RequestMapping annotation at the class or method level.

RequestMappingHandlerAdapter invokes the controller methods

ControllerClassNameHandlerMapping

WelcomeController maps to the '/**welcome***' URL based on naming

class="org.springframework.web.servlet.mvc.support.ControllerClassNameHandlerMapping" **/>**

<bean class="edu.mum.controller.WelcomeController" />

SimpleUrl HandlerMapping

Mapping keys defined on bean[SimpleUrlHandlerMapping] definition:

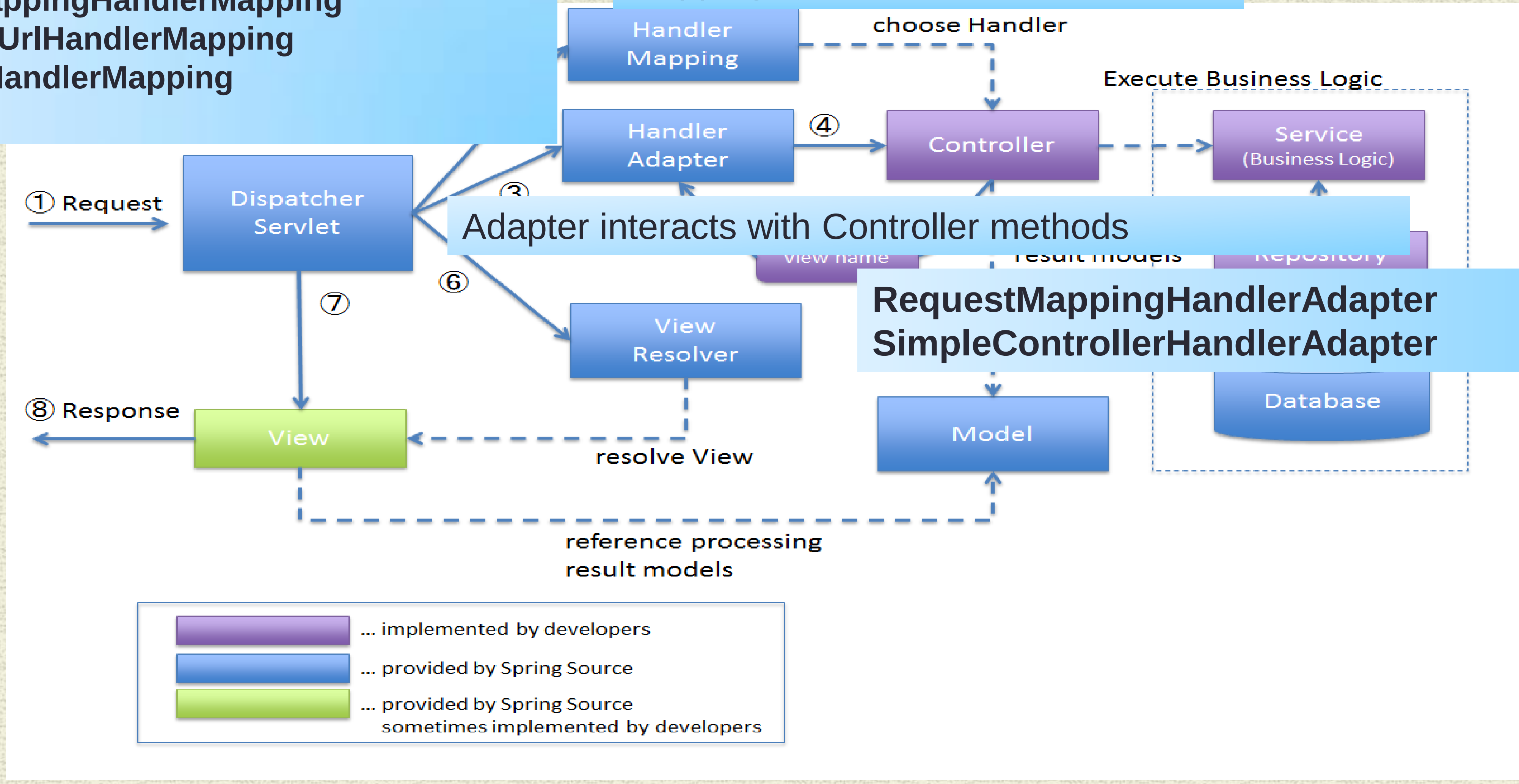
<prop key='/showAll'>showController</prop>

<bean id=“showController” class="edu.mum.controller.ShowController" />

Spring MVC Flow More Details

Mapping chooses Controller

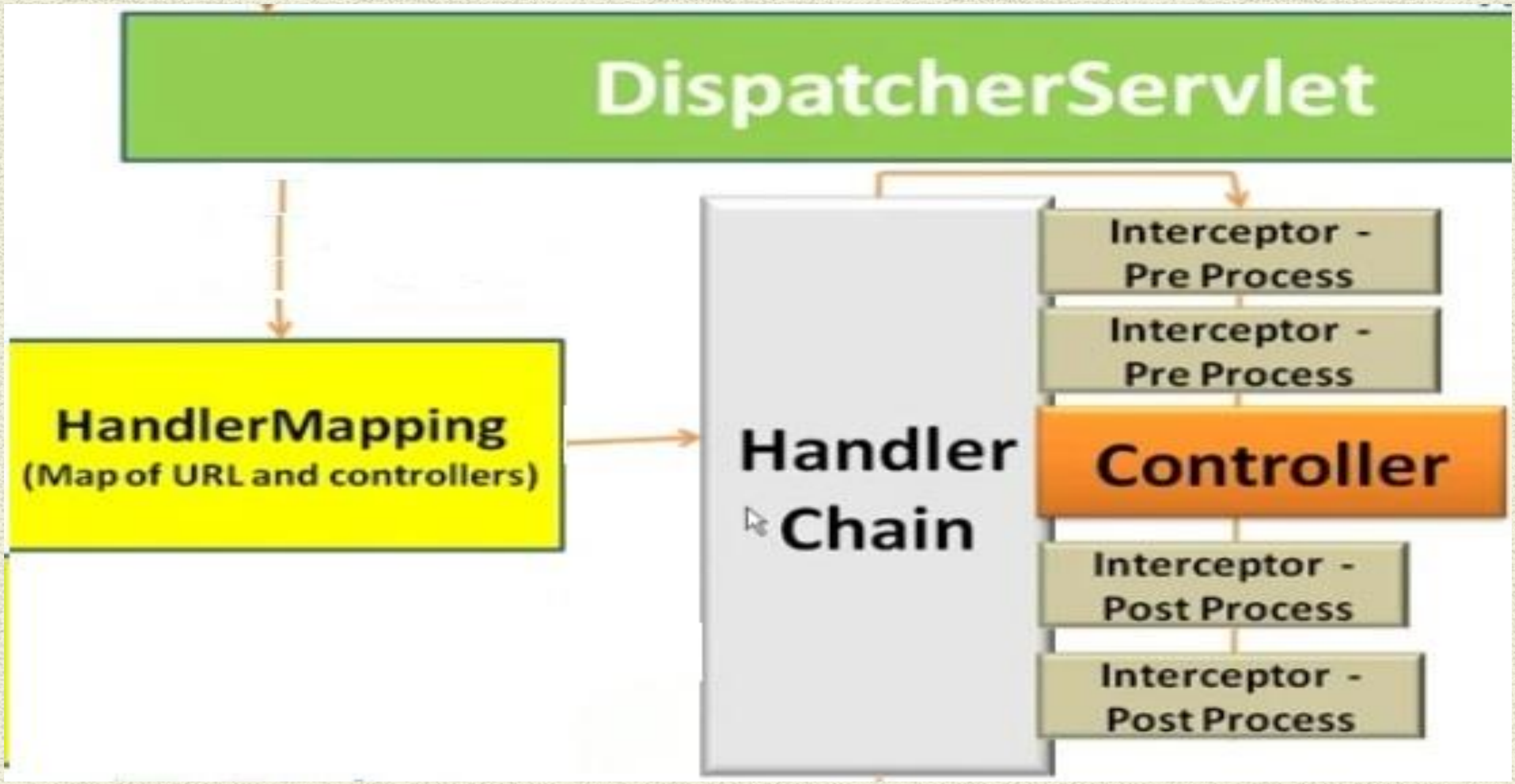
RequestMappingHandlerMapping
BeanNameUrlHandlerMapping
SimpleUrlHandlerMapping
…



Adapter interacts with Controller methods

RequestMappingHandlerAdapter
SimpleControllerHandlerAdapter

Interceptor Handler Chaining



Interceptor Configuration

```
<mvc:interceptors>
<mvc:interceptor>
  <mvc:mapping path="/*"/>
   <bean class="mum.edu.interceptor.VolunteerInterceptor" />
</mvc:interceptor>
</mvc:interceptors>
```

Interceptor Implementation

```java
public class VolunteerInterceptor extends HandlerInterceptorAdapter {
```

Usage example: Add resources need in Handler

```java
@Override
public boolean preHandle(HttpServletRequest request,
HttpServletResponse response, Object handler) throws Exception {
```

Usage example: Add to model for display

```java
@Override
public void postHandle(HttpServletRequest request,
HttpServletResponse response, Object handler,ModelAndView modelAndView) throws Exception {
```

Usage example: Logging…auditing…cleanup…

```java
@Override
// Callback after rendering the view.
public void afterCompletion(HttpServletRequest request,
HttpServletResponse response, Object handler, Exception ex)
throws Exception {
```

Main Point

Handler Mapping provides for the recognition and enhancement of controller actions which is of benefit to the entire application.

Cosmic Consciousness leads to ideal thought and action which is of benefit to everyone