LESSON 8 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) for all validation needs.

- Additionally an application 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 know as the Bean Validation API
- JSR-303 provider library, e.g., Hibernate-Validator.jar
- Annotate model to be validated in the Controller method
- signature with @Valid: BindingResult IMMEDIATELY after model attribute

```
Public String save(@Valid @ModelAttribute User user, BindingResult result) {
        if (result.hasErrors()) {…
```

- Annotate model properties as necessary:

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

- Externalize error messages in resource

Display errors in View

```
<form:form commandName="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>
```

Show ALL errors on Page

Show field level error

External error message file setup

```
<bean id= "errorMessageSource" class="org.springframework.context.support.ResourceBundleMessageSource">
    <property name="basename" value="errormessages"/>
</bean>

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

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

errormessages.properties:

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

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

Domain object annotations

It's for Strings and collections.

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

use for Objects

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

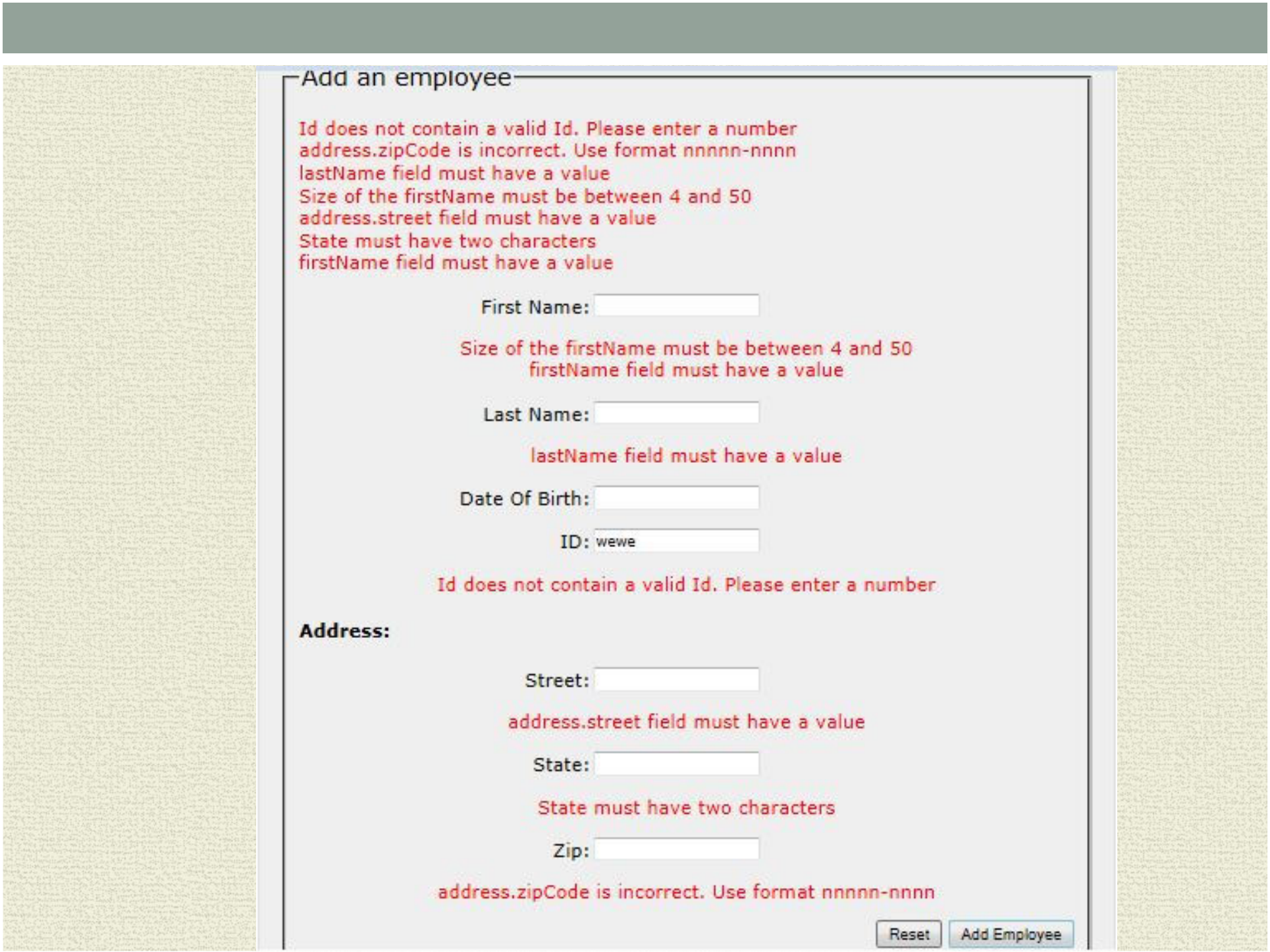
ADDRESS:

```
@NotEmpty(message="String.empty")
private String street;
@Size(min=2, max=2, message="Size.state")
private String state;
@Pattern(regexp="^\\d{5}(-\\d{4})?$",message="{Pattern.zipcode}")
 private String zipCode;
```

Errormessages.properties entries

```
typeMismatch.id= Id is not valid . Please enter a number
NotEmpty= {0} field must have a value
String.empty = {0} must have value
Size.state = State must have two  characters
Size.name.validation = Size of the {0} must be between 4 and 50
typeMismatch.java.util.Date={0} is an invalid date. Use format
MM-DD-YYYY.
Pattern.zipcode= {0} is incorrect. Use format nnnnn-nnnn
```

- **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}*


Add an employee
Id does not contain a valid Id. Please enter a number
address.zipCode is incorrect. Use format nnnnn-nnnn
lastName field must have a value
Size of the firstName must be between 4 and 50
address.street field must have a value
State must have two characters
firstName field must have a value
First Name:
Size of the firstName must be between 4 and 50
firstName field must have a value
Last Name:
lastName field must have a value
Date Of Birth:
ID: wewe
Id does not contain a valid Id. Please enter a number
Address:
Street:
address.street field must have a value
State:
State must have two characters
Zip:
address.zipCode is incorrect. Use format nnnnn-nnnn
Reset
Add Employee

Typemismatch

- Non-String – if value cannot be converted to the data-type then an Exception is thrown.
- 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
  ```

Main Point

- Validation checks the correctness of data against business rules. This prevents problems in the business model from arising. *In Cosmic Consciousness, life is lived stress- free; problem-free*

Manual Validation [W/O Annotations]

Object Validator implements Validator interface.

```
public class MemberValidator implements Validator {
...
@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");
 }
```

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:
  1. Define a default error message
  2. Create a constraint annotation
  3. Implement a validator
- Step 1 Define Default Error Message
- Put message in errormessage.properties file
- 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.

Step 3 Implement Validator

Annotation & Types to be validated

```
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 constrair
    }

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

        } catch (ProductNotFoundException e) {
            return true;
        }

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

add additional error messages or completely disable the default error message

Usage:

```
@Pattern(regexp="P[1-9]+",message="{Pattern.Product.productId.validation}")
@ProductId
private String productId;
```

Cross Field Validation

- NEED: validate the combination of two or more fields
- Similar to field level Validator BUT different
- Class Level…Validation against entire Class object

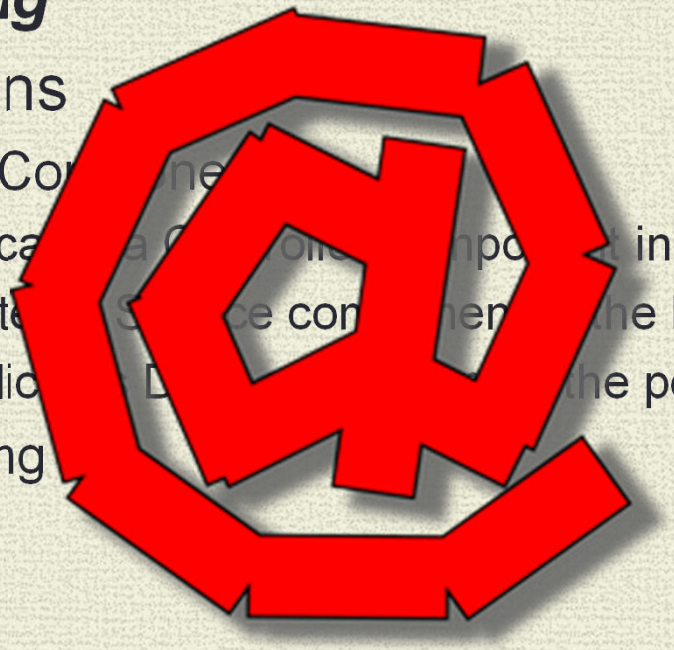
```
public class StockMaximumValidator implements ConstraintValidator<StockMaximum,
Product> {
public boolean isValid(Product product,final ConstraintValidatorContext context){
    BigDecimal unitPrice;
    Long unitsInStock;

    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

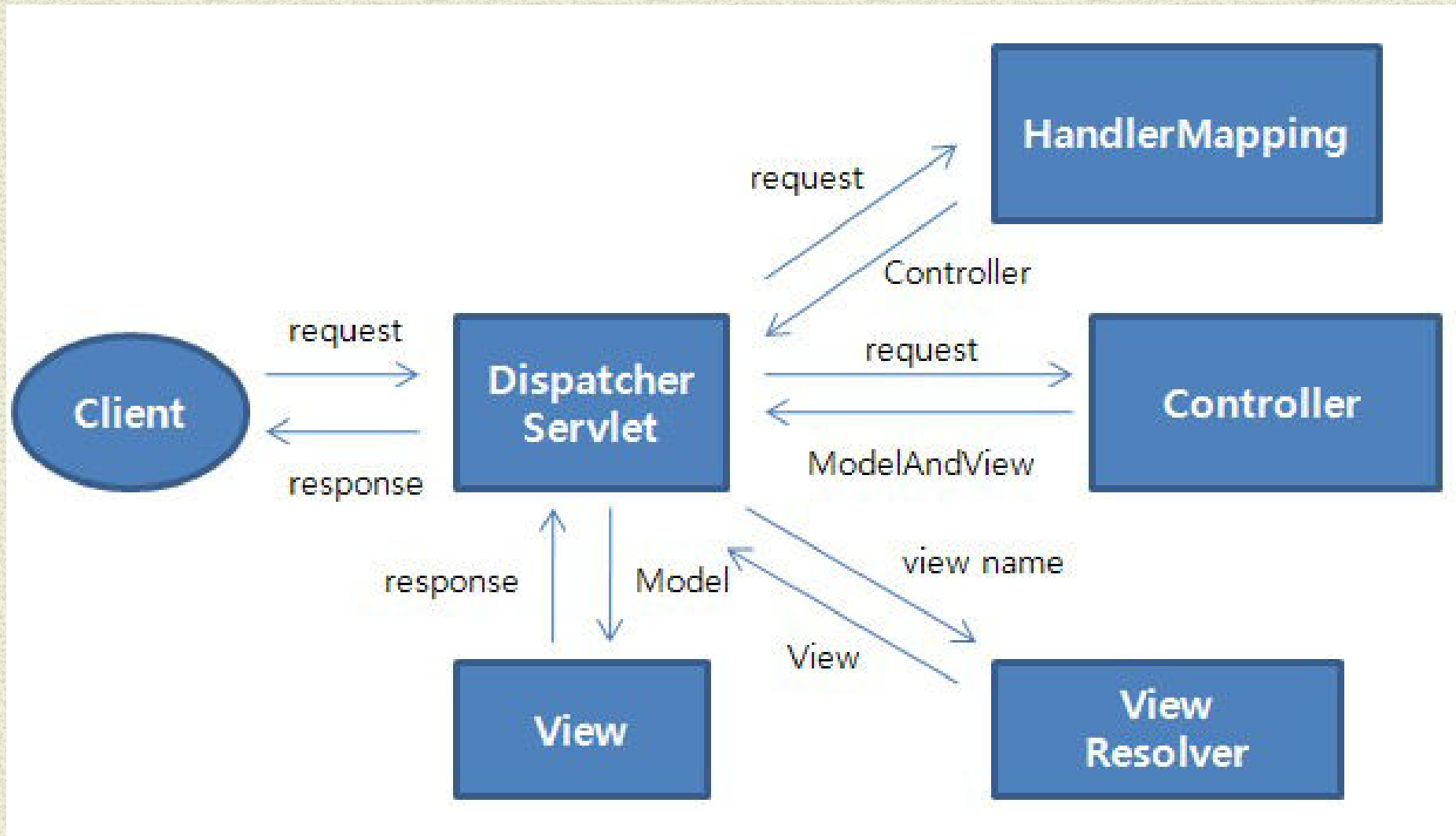
- Custom validation allows for handling more complex, extraordinary verification issues. *A quality of Cosmic Consciousness is the ability to know what is true and right in every situation.*

Spring MVC Architecture & Annotations

- ***Handler Mapping***
- Spring Annotations
  - Spring Managed Co ne
    - @Controller Indica a C ntro mpo t in the presentation layer.
    - @Service Indicat S ce con ner he business layer
    - @Repository Indic D he persistence layer.
  - @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.do“ class="app03a.controller.InputProductController"/>

The URL of the Client is directly mapped to the Controller

DefaultAnnotationHandlerMapping *******default******

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

ControllerClassNameHandlerMapping

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

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

<bean class="com.mkyong.common.controller.WelcomeController" />

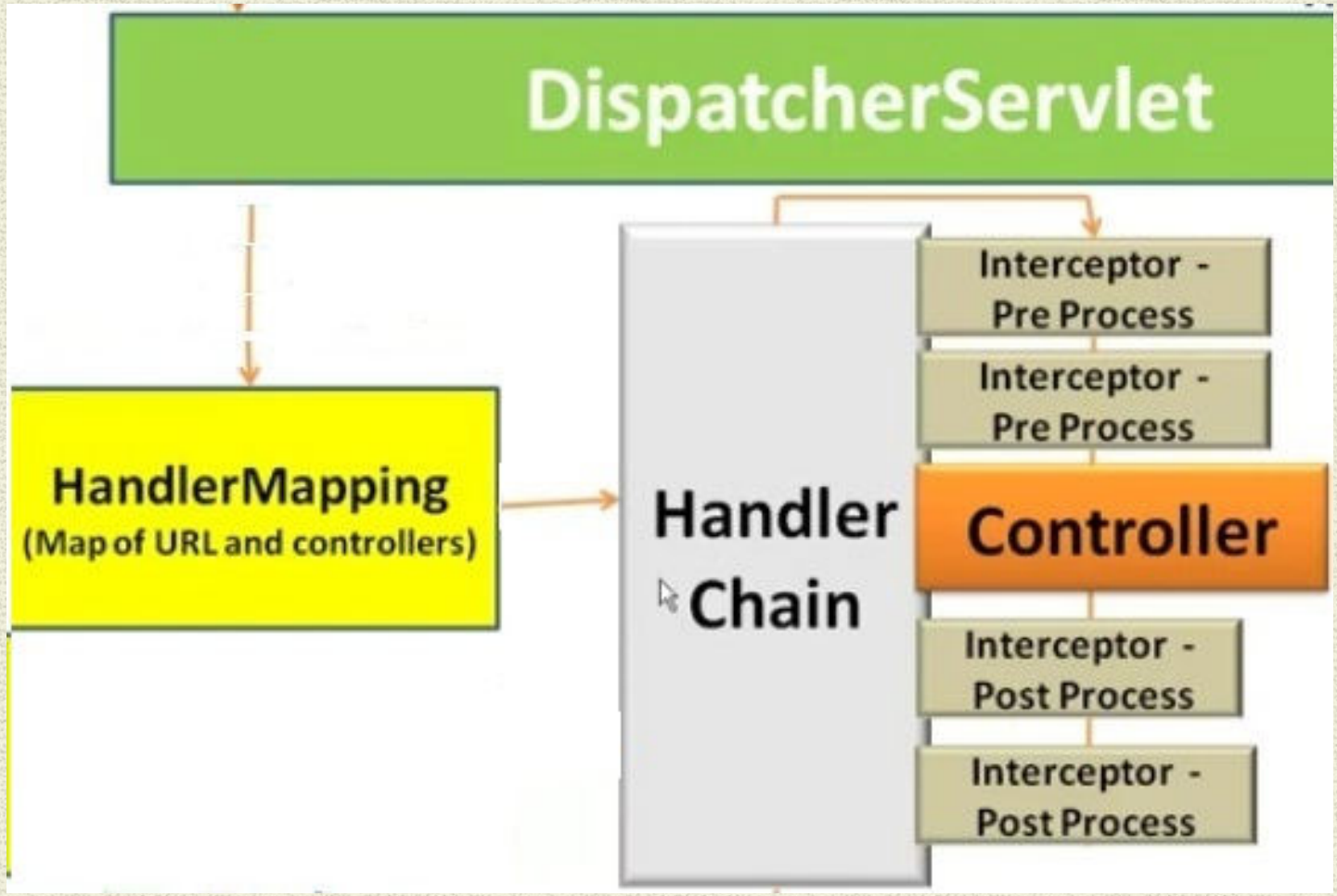
SimpleUrl HandlerMapping

Keys defined on bean definition:

<prop key='/showAllMails.jsp'>showController</prop>

<prop key='/**/help.html'>helpController</prop>

Handler Chaining



Interceptor Configuration

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

Interceptor Implementation

```
public class VolunteerInterceptor extends HandlerInterceptorAdapter
{

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

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

@Override
public void afterCompletion(HttpServletRequest request,
HttpServletResponse response, Object handler, Exception ex)
throws Exception {
```

Main Point

- Handler Mapping & Chaining aids in organizing functionality in layers. As a result the design is simpler & more consistent. *Life is structured in layers. This orderliness within us and around us allows us to enjoy more efficiency in our life*