Module 22: Validation

CS544: Enterprise Architecture

Wholeness

- In this lecture we will look at validating data.
- We can do this by specifying constraints for what we expect the data to be like.
- *Purification leads to progress.*

Validation

ABOUT VALIDATION

Validation

- Validation is the act of ensuring that the your data is what you expect it to be.
- Ensuring that the program operates on clean, correct and useful data*

- Important for:
  - Consistency, Reliability, Security

*http://en.wikipedia.org/wiki/Data_validation

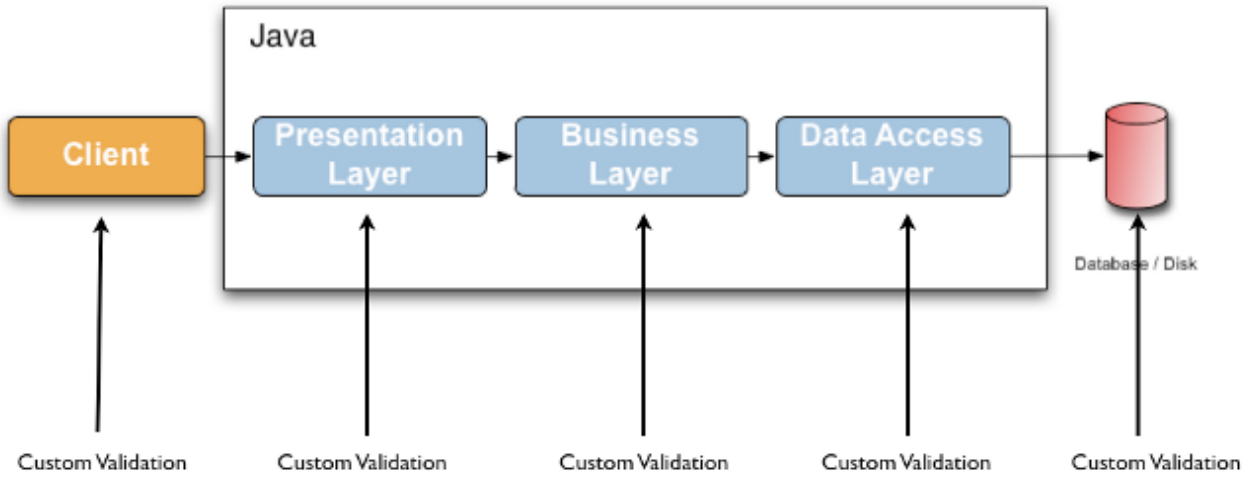
Hibernate Bean Validation*

- Validating data is a common task that occurs throughout the application layers
  - Presentation layer (user input)
  - Business layer (input from clients)
  - Data Access Layer (check before persisting)

Many of the slides are based on the hibernate validation reference documentation:
http://docs.jboss.org/hibernate/stable/validator/reference/en-US/html_single/

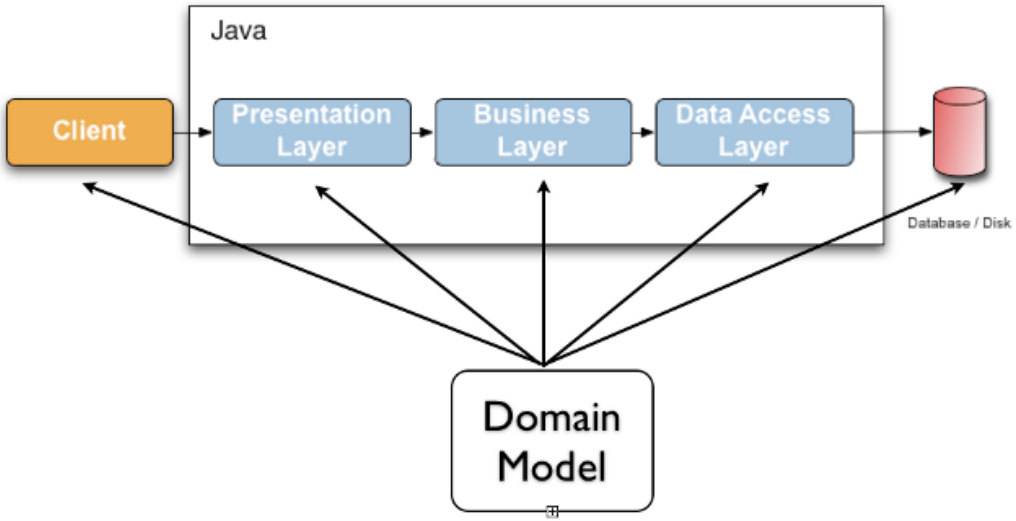
Validation Logic

- The same validation logic could be implemented in each layer
  - time consuming
  - error-prone (violates DRY)



Validation in Domain

- Validation logic can be added to the domain
  - But code would clutter domain classes
  - Such code is really just metadata about each class
  - Meta data is best expressed by annotations



Code Validation VS. Annotations

Code

```
@Entity
public class Car {

  ...

  private int year;

  public void setYear(int year) {
    if (year > 1940 && year < 2015){
      this.year = year;
    } else {
     throw new IllegalArgumentException
         ("Invalid year for a Car");
    }
  }
```

Annotations

```
@Entity
public class Car {

    ...

    @Range(min = 1940, max = 2015)
    private int year;
```

Main Point

- Validation allows us to ensure that our program operates on clean, correct, and useful data. *Seek the highest first.*

Validation

VALIDATION ANNOTATIONS

Declaring Bean Constraints

- Constraints can be declared on:
  - Fields (validator framework will use reflection)
  - Properties (Class needs to adhere to JavaBean)
  - Constraint Inheritance (super class / interface)
  - References / creating a valid Object Graph
  - Class Level Constraints (always custom)
    - Useful if for checking related properties
    - Eg. Car.passengers <= Car.seats

Provided Constraints 1/3

Annotation	Data Types	Description
@Null	Any	Check if it's null (affects column)
@NotNull	Any	Check that it's not null
@Valid	Any non-primitive	Go into the object and validate it
@AssertFalse	Boolean	Check that it's false
@AssertTrue	Boolean	Check that it's true
@Future	Date or Calendar	Check that it's in the future
@Past	Date or Calendar	Check that it's in the past
@Size(min=,max=)	String / Collection	Check size is >= min and <= max, column length set to max
@Pattern(regex=,flag=)	String	Check that it matches the regex

Numeric Constraints

Annotation	Data Types	Description
@Min(value=)	Numeric types	Check that it's not lower
@Max(value=)	Numeric types	Check that it's not higher
@DecimalMin(value=,inclusive=)	Numeric types	Check that it's not lower
@DecimalMax(value=,inclusive=)	Numeric types	Check that it's not higher
@Digits(integer=,fraction=)	Numeric types	Checks if it has less digits / fractional points then given

@Min, @Max and @Digits also affect DDL, adding constraints on the table column

@DecimalMin and @DecimalMax do not, but their min/max values can be specified
As strings which allows you to check beyond Long.MAX_VALUE / Long.MIN_VALUE

Additional Constraints

Annotation	Data Types	Description
@CreditCardNumber(..)	String	Credit Cards
@EAN	String	Barcode
@Email	String	Email address
@URL(…)	String	URL
@Length(min=,max=)	String	Column length set to max
@LuhnCheck(…)	String	Checksum (mod 10) CC
@Mod10Check(…)	String	Checksum (mod 10)
@Mod11Check(…)	String	Checksum (mod 11) ISBN
@NotBlank	String	Not null, trimmed length > 0
@NotEmpty	String / Collection	Not null or empty
@Range(min=,max=)	Numeric	Checks >= min and <= max
@SafeHtml(…)	String	Requires jsoup, checks for <script> etc
@ScriptAssert(…)	Any Type	Executes JSR 233 script against target

Fields and Properties

Fields

```
public class Car {

  @NotNull
  private String manufacturer;

  @AssertTrue
  private boolean isRegistered;

  public Car(String manufacturer,
             boolean isRegistered) {
    this.manufacturer = manufacturer;
    this.isRegistered = isRegistered;
  }

  //getters and setters...
}
```

Properties

```
public class Car {
    private String manufacturer;
    private boolean isRegistered;
    public Car(String manufacturer, boolean
     isRegistered) {
        this.manufacturer = manufacturer;
        this.isRegistered = isRegistered;
    }

    @NotNull
    public String getManufacturer() {
        return manufacturer;
    }
    public void setManufacturer(String manufacturer) {
        this.manufacturer = manufacturer;
    }

    @AssertTrue
    public boolean isRegistered() {
        return isRegistered;
    }
    public void setRegistered(boolean isRegistered) {
        this.isRegistered = isRegistered;
    }
}
```

Inheritance

```java
public class Car {
    private String manufacturer;

    @NotNull
    public String getManufacturer() {
        return manufacturer;
    }

    // ...
}
```

```java
public class RentalCar extends Car {

    private String rentalStation;

    @NotNull
    public String getRentalStation() {
        return rentalStation;
    }

    //...
}
```

When validating RentalCar, both manufacturer and rentalStation will be validated

Also works with interfaces

Object Graph

```
public class Car {

public class Car {

    @NotNull
    @Valid
    private Person driver;

    //...
}
```

```
public class Person {

    @NotNull
    private String name;

    //...
}
```

When validating Car, @Valid makes the validator ‘cascade’ into Person, and check name is @NotNull

Class Level

```
@ValidPassengerCount
public class Car {

    private int seatCount;

    private List<Person> passengers;

    //...
}
```

You can make custom classlevel annotations to check the relationship between properties

@ValidPassengerCount for example could check that:
passengers.size() <= seatcount

Custom Constraint Annotation

```
@Target({ TYPE, ANNOTATION_TYPE })
@Retention(RUNTIME)
@Constraint(validatedBy = { ValidPassengerCountValidator.class })
@Documented
public @interface ValidPassengerCount {

    String message()
    default"{org.hibernate.validator.referenceguide.chapter06.classlevel." +
            "ValidPassengerCount.message}";

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

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

Custom Validator

```
public class ValidPassengerCountValidator
    Implements ConstraintValidator<ValidPassengerCount, Car> {

  @Override
  public void initialize(ValidPassengerCount constraintAnnotation) {
  }

  @Override
  public boolean isValid(Car car, ConstraintValidatorContext context) {
    if (car == null) {
      return true;
    }
    return car.getPassengers().size() <= car.getSeatCount();
  }
}
```

Main Point

- Using annotations we can easily declare what constraints should be applied to our data fields/references/classes.
- While there are many built in annotations, the hibernate validation framework is setup to easily allow for custom annotations.
- All kinds of things can be validated, *Harmony exists in diversity.*

Validation

PROGRAMMATIC VALIDATION

Programmatic Validation

```
public class App {
  public static void main(String[] args) {
    ValidatorFactory factory = Validation.buildDefaultValidatorFactory();
    Validator validator = factory.getValidator();

    Car car = new Car( null, true );

    Set<ConstraintViolation<Car>> constraintViolations =
        validator.validate( car );

    assertEquals( 1, constraintViolations.size() );
    assertEquals( "may not be null",
        constraintViolations.iterator().next().getMessage() );
  }
}
```

Create a validator

Car from slide 15, property constraints

Validate something

Check if it worked

Checking a Single Prop / Value

- If you don't want to validate an entire object:
  - You can validate individual values
  - You can validate individual properties
    - JavaBean property name (take off get, lowercase first)
  - These will not follow @Valid annotations

```
Set<ConstraintViolation<Car>> constraintViolations = validator.validateValue(
        Car.class,
        "manufacturer",
        null );

Set<ConstraintViolation<Car>> constraintViolations = validator.validateProperty(
        Car.class,
        "manufacturer",
        null );
```

Constraint Violation Methods

Method	Description	Example
getMessage()	The error message	“may not be null”
getMessageTemplate()	The name in the bundle	{…NotNull.message}
getRootBean()	Root of object graph	Car
getRootBeanClass()	Class or root bean	Car.class
getLeafBean()	‘leaf’ the constraint is on	Person
getPropertyPath()	From root to property	Car.Person.name
getInvalidValue()	Value failing the constraint	null
getConstraintDescriptor()	Access to annotation etc.	@NotNull

Main Point

- A ConstraintViolation object is created for every item that fails to validate.
- Based on this information we can then take the appropriate action to fix the incorrect data.
- *Thought leads to action, action leads to achievement, and achievement leads to fulfillment.*

Validation

SPRING MVC INTEGRATION

web.xml

```
<web-app ... >
    <servlet>
        <servlet-name>springmvc</servlet-name>
        <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
        <load-on-startup>1</load-on-startup>
    </servlet>
    <servlet-mapping>
        <servlet-name>springmvc</servlet-name>
        <url-pattern>/</url-pattern>
    </servlet-mapping>

    <context-param>
        <param-name>contextConfigLocation</param-name>
        <param-value>/WEB-INF/rootconfig.xml</param-value>
    </context-param>
    <listener>
        <listener-class>org.springframework.web.context.ContextLoaderListener</listener-class>
    </listener>

    <filter>
        <filter-name>OpenSessionInView</filter-name>
        <filter-class>org.springframework.orm.hibernate4.support.OpenSessionInViewFilter</filter-class>
    </filter>
    <filter-mapping>
        <filter-name>OpenSessionInView</filter-name>
        <url-pattern>/*</url-pattern>
    </filter-mapping>
</web-app>
```

SpringMVC Servlet

Root Config for Service & DAO

Open Session in View Filter

SpringMVC-servlet.xml

```xml
<?xml version="1.0" encoding="UTF-8"?>
<beans ...>
    <!-- find controller beans -->
    <mvc:annotation-driven />
    <context:component-scan base-package="cs544" />

    <!-- enable custom validation messages -->
    <bean id="messageSource1"
          class="org.springframework.context.support.ResourceBundleMessageSource">
        <property name="basename" value="messages" />
    </bean>

    <!--Resolves views selected for rendering by @Controllers to .jsp resources in the /WEB-INF/views directory -->
    <bean class="org.springframework.web.servlet.view.InternalResourceViewResolver">
        <property name="viewClass" value="org.springframework.web.servlet.view.JstlView" />
        <property name="prefix" value="/WEB-INF/views/" />
        <property name="suffix" value=".jsp" />
    </bean>
    <!-- Forwards requests to the "/addCar" resource to the "addCar" view -->
    <mvc:view-controller path="/index" view-name="index"/>
    <mvc:view-controller path="/login" view-name="login"/>
    <!-- Lets us find resources (.html etc) through the default servlet -->
    <mvc:default-servlet-handler/>
    <!-- Handles HTTP GET requests for /resources/** -->
    <mvc:resources mapping="/resources/**" location="/resources/" />
</beans>
```

View

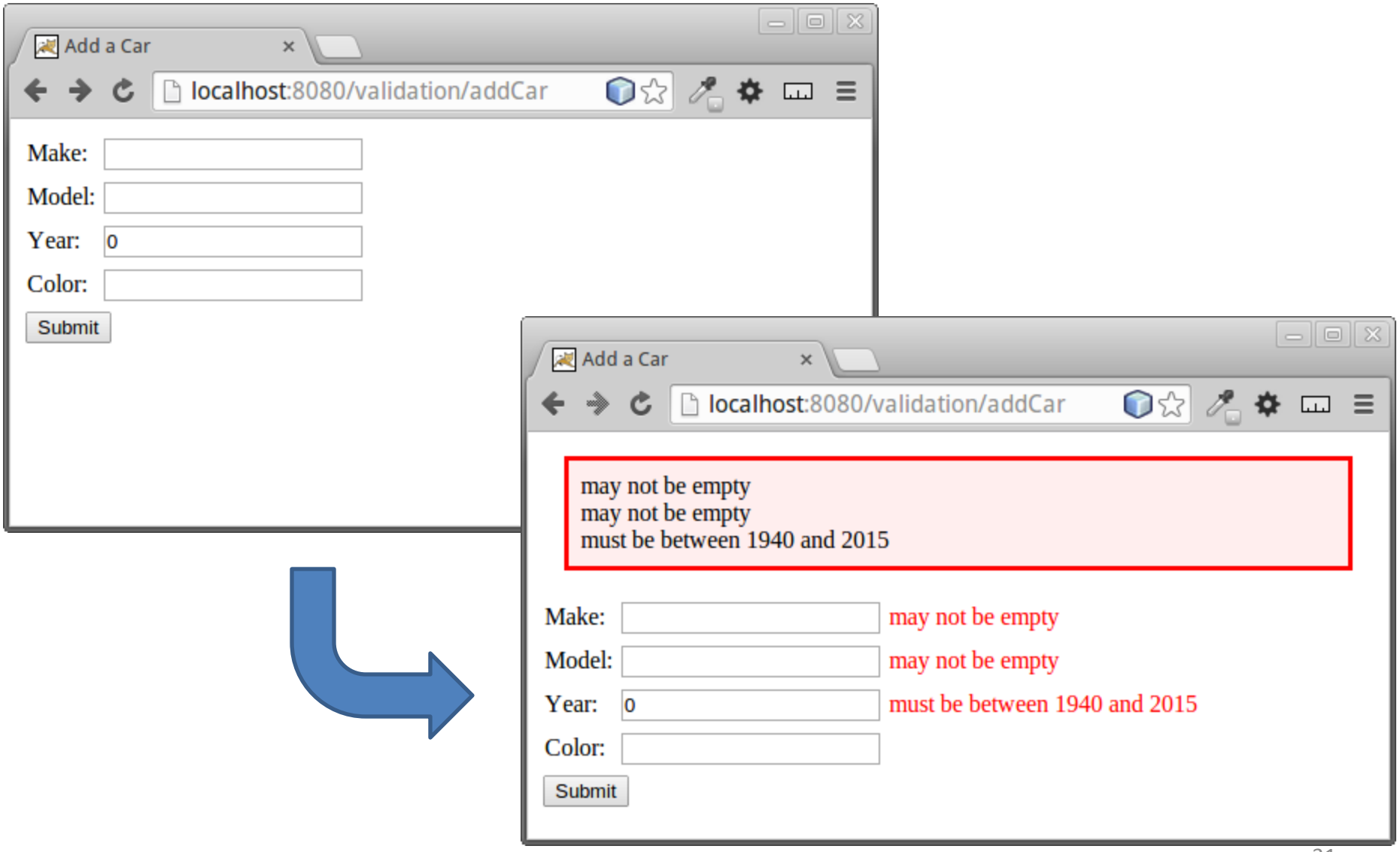
```
<%@ taglib prefix="form" uri="http://www.springframework.org/tags/form"%>
<!DOCTYPE html>
<html>
    <head>
        <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
        <title>Add a Car</title>
        <link href="resources/style.css" rel="stylesheet" type="text/css" />
    </head>
    <body>
        <form:form commandName="car" action="addCar" method="post">
            <form:errors path="*" cssClass="errorblock" element="div" />
            <table>
                <tr>
                    <td>Make:</td>
                    <td><form:input path="make" /> </td>
                    <td><form:errors path="make" cssClass="error" /> </td>
                </tr>
              <!-- Model year and color removed to keep the slide shorter -->
            </table>
            <input type="submit"/>
        </form:form>
    </body>
</html>
```

commandName should be called: beanName

Spring Form error tags display validation errors for their fields

path attribute specifies what 'their' field is

Form / Validation Errors



Controller

```
@Controller
public class CarController {
  @Autowired
  private CarService carService;

  @RequestMapping(value = "/addCar", method = RequestMethod.GET)
  public String addCar(@ModelAttribute("car") Car car) {
        return "addCar";
  }

  @RequestMapping(value = "/addCar", method = RequestMethod.POST)
  public String add(@Valid Car car, BindingResult result){
    String view = "redirect:/cars";
    if (!result.hasErrors()) {
      carService.add(car);
    } else {
      view = "addCar";
    }
    return view;
  }
```

The Car 'command' object always has to be available, even to just display the form

Results of validating and putting it into the Car object (binding)

Important!
No P/R/G when there are errors!

Car Class

```
@Entity
public class Car {
    @Id
    @GeneratedValue
    private int id;

    @NotEmpty
    private String make;

    @NotEmpty
    private String model;

    @Range(min = 1940, max = 2015)
    private int year;

    private String color;
```

http://docs.spring.io/spring/docs/current/spring-framework-reference/html/spring-form.tld.html

Spring Form Tags

Tag	Description
<form:form>	Creates an HTML form, that can also hold Spring Form Tags
<form:errors>	path attribute can specify which field, shows all without
<form:input>	Wrapper for HTML element
<form:password>	Wrapper for HTML element
<form:hidden>	Wrapper for HTML element
<form:textarea>	Wrapper for HTML element
<form:select>	Wrapper for HTML element
<form:option>	Wrapper for HTML element
<form:options>	Creates multiple option elements from a list
<form:checkbox>	Wrapper for HTML element
<form:checkboxes>	Creates multiple checkboxes from a list
<form:radiobutton>	Wrapper for HTML element
<form:radiobuttons>	Creates multiple radio buttons from a list
<form:label>	Wrapper for HTML element

Main Point

- Spring MVC will integrate with a validator implementation if it detects it on the classpath
- Spring Form error tags display any error messages related to their field
- *Do less and accomplish more*