VALIDATION

Continue to

Avoid the Danger that has not yet come

Data Validation is Fundamental

"The Data Warehousing Institute (DWI) estimates the cost of bad or 'dirty' data exceeds $600 billion annually"

--- Data Cleaning & Validation

Estimates show dirty data is a big problem for the U.S. economy. How big? About 2x the national deficit.

-- Dirty Data costs U.S. $3 Trillion

Retail company found over 1m records contained home telephone number of "000000000" and addresses containing flight numbers

Healthcare company found 9 different values in gender field

Et cetera, et cetera, et cetera…

Approaches to Validation

- **Front-end validation**
  - Essentially user input validation
  - Manual form validation
  - Transfer “secure” data to persistence layer
  - Complexity grows as variety of front ends increase
- **Back-end validation**
  - Database handles validation
  - Constraints are enforced in the database
    - e.g., Insert 50 character string into a VARCHAR(20) column
  - Significant Performance cost
- **JSR 303/349 Validation:**
  - “End-agnostic” - you can use it anywhere
  - Scaleable
  - Supports data driven approach
  - DRY

JSR 303 - 349

- Java **Bean Validation** Framework
- Defines a metadata model [Java Annotations]

- **Raison d'etre**

  Data Validation occurs throughout an application presentation layer thru persistence layer

  Same validation logic is often duplicated in each layer time consuming and error-prone.

Hibernate Validator is the reference implementation for JSR 303

Spring Core Technology
Data 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) and the corresponding annotations for all validation needs.**
- Additionally an application can use the Spring Validator directly without the use of annotations.

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
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;
```

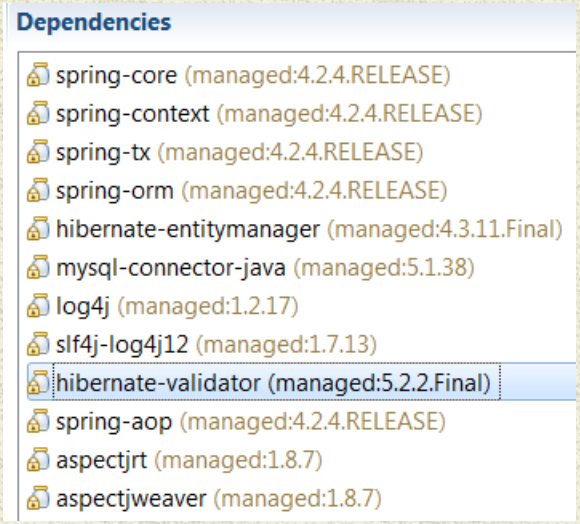
Error message externalized in .properties file

```
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 {2} and {1}
typeMismatch.java.util.Date={0} is an invalid date. Use format MM-DD-YYYY.
Pattern.zipcode= {0} is incorrect. Use format nnnnn-nnnn
```

- **NOTE:**
- *"placeholders" are in alphabetical order. @Size(min=1,max=5), field name as {0} , the max value as {1} , and the min value as {2}*

Spring Validation Config

Maven Dependency




```
<bean id="messageSource"
  class=
  "org.springframework.context.support.ReloadableResourceBundleMessageSource">
       <property name="basename" value="classpath:errorMessages" />
</bean>
<bean id="messageAccessor"
        class="org.springframework.context.support.MessageSourceAccessor">
            <constructor-arg  ref="messageSource"/>
</bean>
<bean id="validator"
    class=
     "org.springframework.validation.beanvalidation.LocalValidatorFactoryBean">
         <property name="validationMessageSource" ref="messageSource" />
</bean>
```

Data Validation Application

- **Presentation**
  - simple input validation
  - do not proceed if the input is in the wrong format
  - "gate" client requests to the server to reduce round-trips, for ***better usability and reduced bandwidth/time***
- **Service**
  - business logic and **authorization**
  - don't let users do things they aren't allowed to do
  - handle "derived" properties and state here (things that would be denormalized in the database)
- **Persistence**
  - the essential data integrity layer
  - **ABSOLUTELY REFUSE** to store any junk

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

Constraint Composition "Convenience" Feature

- **USE CASE Example:**
- `@NotEmpty`
- `Size(min=5, max = 9, message= "{EmptyOrSize}")`
- `private String lastName;`

- **For `lastName = ""`**
- **2 messages:**

  Last name is a required field

  Last name must be between 5 and 9 characters

Composition Alternative:

- `@EmptyOrSize(min=5, max = 9, message= "{EmptyOrSize}")`
- **1 message:**
- `Last Name must be between 5 & 9 characters`

Annotation Implementation

```
@NotEmpty( )
@Size
@Target( { ElementType.METHOD, ElementType.FIELD })
@Retention(RetentionPolicy.RUNTIME)
@Constraint(validatedBy = {})
@ReportAsSingleViolation
@Documented
public @interface EmptyOrSize {
String message() default “Must be a value and the right size.";
Class<?>[] groups() default {};
Class<? extends Payload>[] payload() default {  };

@OverridesAttribute(constraint=Size.class, name="min")
    int min() default 10;

    @OverridesAttribute(constraint=Size.class, name="max")
    int max() default 15;
```

- **SEE DEMO Validation**

JSR 303 GROUPS

- **Constraints may be added to one or more groups**
- **Groups allow you to restrict the set of constraints applied during validation.**

- **USE CASE Scenario:**
- **Overnight batch job** loads new products with INVALID Status

If product "passes" default validation — Default group

product is set to BASIC status

& assigned to an Admin [Sean or Bill]

- **Admin** "fixes" product

If product "passes" details validation — Details group

product is set to DETAILS status

& assigned to the Admin's Supervisor [Paul or Pete]

Supervisor "fixes" product

If product "passes" production validation — Production group

product is set to PRODUCTION status

Hibernate Groups Example

- A Group is an interface
- A group can extend another group
- When validator evaluates a specific group's constraints it also evaluates all of its super groups (interfaces) constraints.

```
public interface Details extends Default {}
public interface Production extends Details {}
```

Default group

```
@EmptyOrSize(min=5, max = 32, message= "{EmptyOrSize}")
private String name;

@EmptyOrSize(min=20,max=2000,message="{EmptyOrSize}",groups={Details.class})
private String description;

@NullMinNumber(value=6,message="{NullMinNumber}",groups={Production.class})
private Integer quantity;
```

Demo
Business Process Management [BPM]

- BPM is designing processes, executing them across people and systems, managing tasks, and continually optimizing it all.

- Workflow Management - is automation tool for directing tasks to the responsible users in a business process for further actions.

- Workflow is a component of BPM, but it is more about task management and how repeatable, less complex individual processes get accomplished.

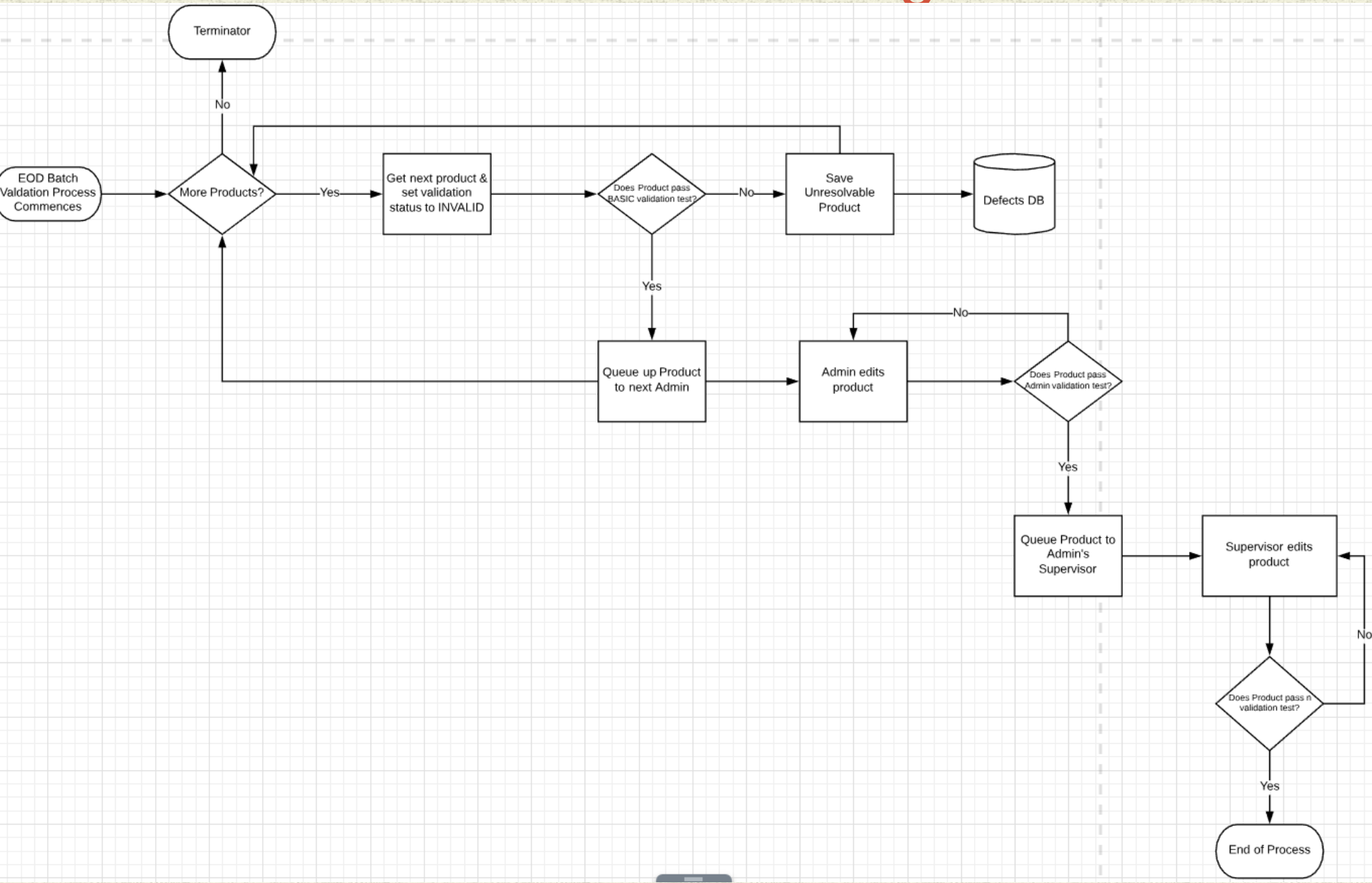
Demo
Workflow/Business Process

EOD Batch job Loads the products

Status set **INVALID**

- If product "passes" default validation
- product is set to **BASIC** status & assigned to an Admin [Sean or Bill]
- **Admin "fixes" product**
- If product "passes" details validation
- product is set to **DETAILS** status & assigned to the Admin's Supervisor
- **Supervisor "fixes" product**
- product is set to **PRODUCTION** status

Demo Workflow Diagram



Hibernate Validator

- Built in to Hibernate [***DIFFERENT from Spring Validator***]

Last line of Defense

Entities are verified before inserts, updates or deletes are made by Hibernate.

- On constraint violation – throws ConstraintViolationException

  contains a set of ConstraintViolations describing each failure.

- If Hibernate Validator is present in the classpath, Hibernate Annotations (or Hibernate EntityManager) will ***AUTOMATICALLY*** use it transparently.
  - To avoid validation even though Hibernate Validator is in the classpath set **javax.persistence.validation.mode to none.**

- **Demo Uses Hibernate Validator**

  BECAUSE Spring no longer supports calling Spring Validator with Group !!!
- **Hibernate Validator**
- Properties file defaults to ValidationMessages…

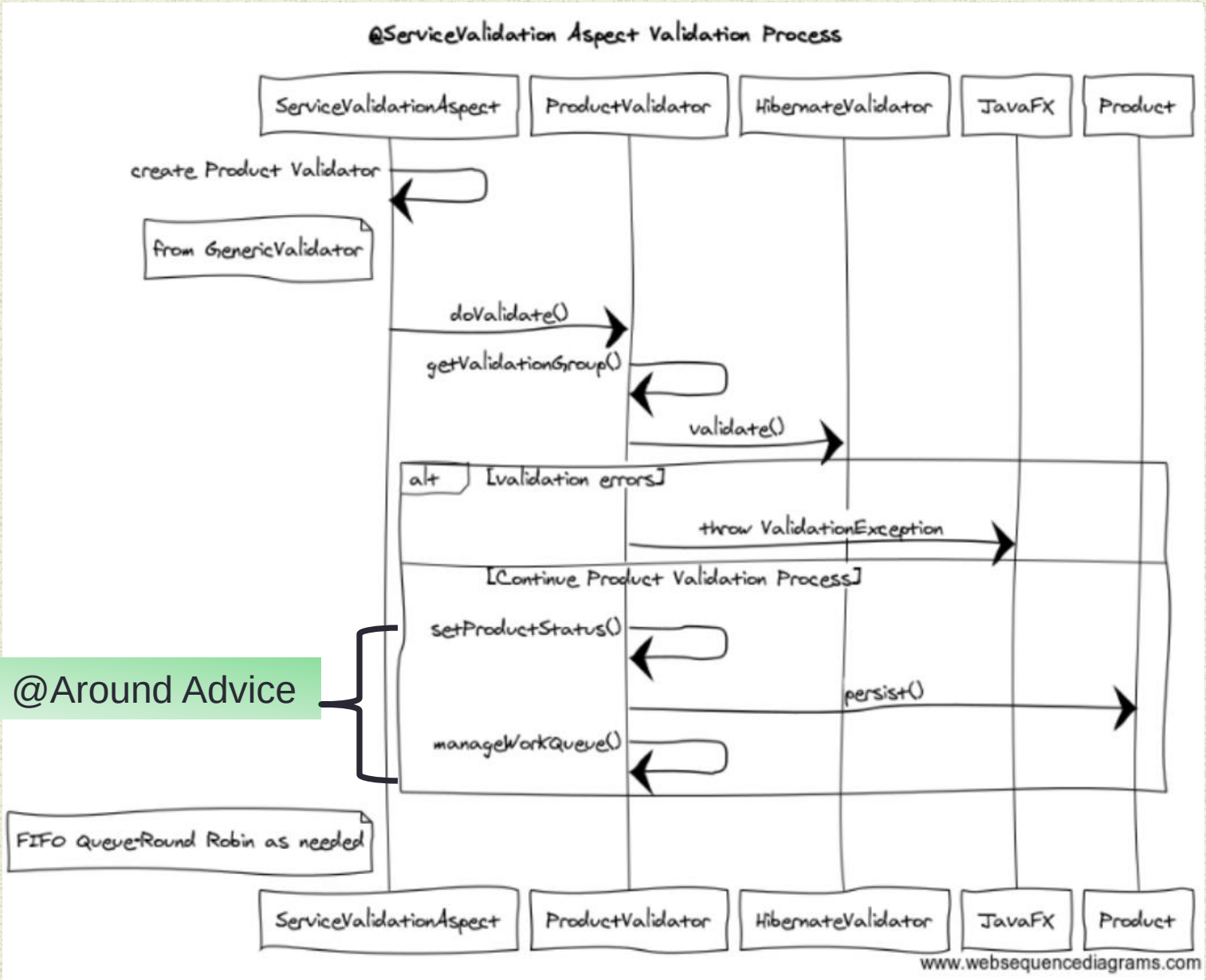
Demo
Validation Aspect

- **ServiceValidationAspect.java**
- `@Aspect`
- `@Component`
- `public class ServiceValidationAspect {`

DEMO uses AOP

- `//Pass in Object under validation`
- `@Around("validate() && applicationMethod() && argsMethod(object)")`
- `public void doValidate( ProceedingJoinPoint joinPoint, Object object)`
- **ProductServiceImpl.java**
- `@ServiceValidation`
- `public void update( Product product) {`
- `this.performUpdate(product);`
- **SEE DEMOs ValidationGroupsBatch && ValidationGroupsDesktop**

Validation Process



This is @Around Advice

Main Point

- JSR 303/349 validation allows for handling more complex, extraordinary verification issues with such features as Groups and Constraint Composition. *A quality of Cosmic Consciousness is the ability to know what is true and right in every situation.*