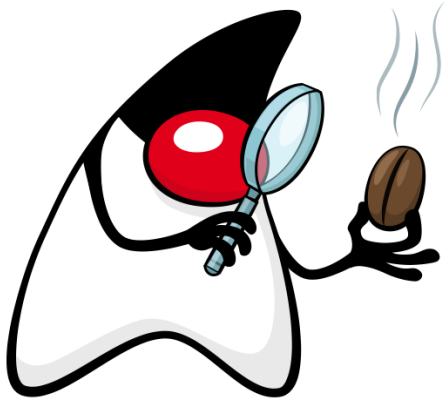
Bean Validation



Nikolaos Ntaountakis
Andreas Geissel

aformatik®
TRAINING & CONSULTING GMBH & CO. KG

Bean Validation

Motivation

- Überblick
- Constraints
- Einsatz
- Custom Constraints
- Fragen

Warum validieren wir Daten?

- Syntaktische Korrektheit sicherstellen
  Vermeidung von Problemen, die im weiteren Verlauf auftreten würden
  - Falsches Format (Zahlen, Datum, …)
    - `3.141` statt `3,141`
    - `05.07.18` statt `05.07.2018`
    - `info@aformatik-de` statt `info@aformatik.de`
  - Falsche oder unbekannte fachliche Werte bzw. Bezeichnungen
    - `Schifffffffahrt` statt `Schifffahrt`
    - `Desoxyribonukleinsäure` statt `DNA`
  - Sicherheitsrelevante Probleme (SQL-Injection, XSS, …)
- Inhaltliche Korrektheit sicherstellen
  - „Nur inner-betriebliche E-Mail-Adresse“
  - „Max. ein Jahr in der Vergangenheit“

Warum Bean Validation?

- Nutzt Java-Mittel
  - Annotationen
  - ggfs. Service-Loader-API
- Datenstrukturen mit Gültigkeitsanforderungen zusammen formulieren
- Deklarativer Ansatz zur Formulierung von Gültigkeitsregeln
  - Vermeiden von mehrfach formulierten *Regeln*
  - Validierung in mehreren Schichten
    - UI (Swing , JavaFX , JSF, …)
    - Service-Schicht (EJBs , Spring Services)
    - Backend (JPA, ggfs. Java Stored Procedures)

Bean Validation

- Motivation
- **Überblick**
- Constraints
- Einsatz
- Custom Constraints
- Fragen

Historie – 2009-2018

- Bean Validation 1.0 – 2009 – JSR 303 – http://beanvalidation.org/1.0/
  - Annotations und XML zur Formulierung von Constraints
  - s. Vortrag von Sandro Sonntag auf dem Java Forum Stuttgart 2009
    http://alt.java-forum-stuttgart.de/jfs/2009/folien/F7.pdf
- Bean Validation 1.1 – 2013 – JSR 349 – http://beanvalidation.org/1.1/
  - Constraints an Parametern und Rückgabewerten
  - Integration mit CDI
  - Fehler-Meldungen mit EL möglich
- Bean Validation 2.0 – 2017 – JSR 380 – http://beanvalidation.org/2.0/
  - Unterstützung von Java-8-Features
  - Unterstützung des neuen DateTime-APIs
    s. Vortrag auf dem JFS 2014 http://2014.java-forum-stuttgart.de/_data/D3_2014.pdf

Heute

- Java EE
  - JPA
  - EJBs (Methoden-Parameter von Session Beans)
  - JAX-RS (Methoden-Parameter)
  - JSF
  - CDI
- Spring Boot
  - REST-Controller (Methoden-Parameter)
- mit JavaFX verwendbar
- mit Java SE (manuell) verwendbar

javax.validation

- Constraint – Annotation zur Formulierung einer Regel
- ValidatorFactory – Factory zur Erzeugung eines Validators
- Validator – Zugang zu Validierungsmethoden
- ConstraintValidator – Implementierung der Constraint Prüfung
- ConstraintViolation (ConstraintViolationException) – Ergebnis einer fehlgeschlagenen Prüfung
- Value Extractor – *Adapter* zu Nicht-Standard-Container

Bean Validation

- Motivation
- Überblick
- **Constraints**
- Einsatz
- Custom Constraints
- Fragen

Built-in Constraints

Constraint	Annotiertes Element…	[null]
@Null	… muss null sein	valide
@NotNull	… darf nicht null sein	
@AssertTrue / @AssertFalse	… muss true / false sein	valide
@Min / @Max	… muss größer-gleich / kleiner-gleich sein	valide
@Negative /@Positive	… muss kleiner als 0 bzw. größer als 0 sein	valide
@Size	… muss bestimmte Größe haben (min, max, …)	valide
@Past / @Future	… muss in der Vergangenheit / Zukunft liegen	valide
@Email	… muss eine E-Mail-Adresse sein (seit 2.0)	valide
@NotEmpty	… darf weder null noch leer sein	
@Pattern	… muss zu dem Regulären Ausdruck passen	valide

Weitere

- @NotBlank
- @PositiveOrZero
- @NegativeOrZero
- @PastOrPresent
- @FutureOrPresent
- @DecimalMin
- @DecimalMax

- @Valid

Beispiel - Genehmigung

Firmen-interner Prüfer stellt Genehmigung (intern „Lizenz“) für einen Prozess oder ein Verfahren aus.

Anforderungen an eine Genehmigung

- Name des Ausstellers / Unterzeichners
  - Pflichtangabe
  - firmen-interne Nutzer-Kennung
- E-Mail-Adresse des Ausstellers / Unterzeichners
  - firmen-interne E-Mail-Adresse
- Ausstellungsdatum
  - Pflichtangabe
  - In der Vergangenheit

Beispiel - Genehmigung

```
public class License {
   String     signerName;
   String     internalEmail;
   LocalDate  date;
}
```

Ohne Bean Validation

- Methoden prüfen Gültigkeit
  - In Datenstruktur
  - oder pro Schicht (UI, Server, …)
- *Abstand* zwischen Daten-Struktur und Gültigkeits*definition* relativ groß

```
... validate(License l) {
   if (l.getSignerName()     != null && ...) { ... }
   if (l.getInternalEmail() != null && ...) { ... }
   if (l.getDate()           != null && ...) { ... }
}
```

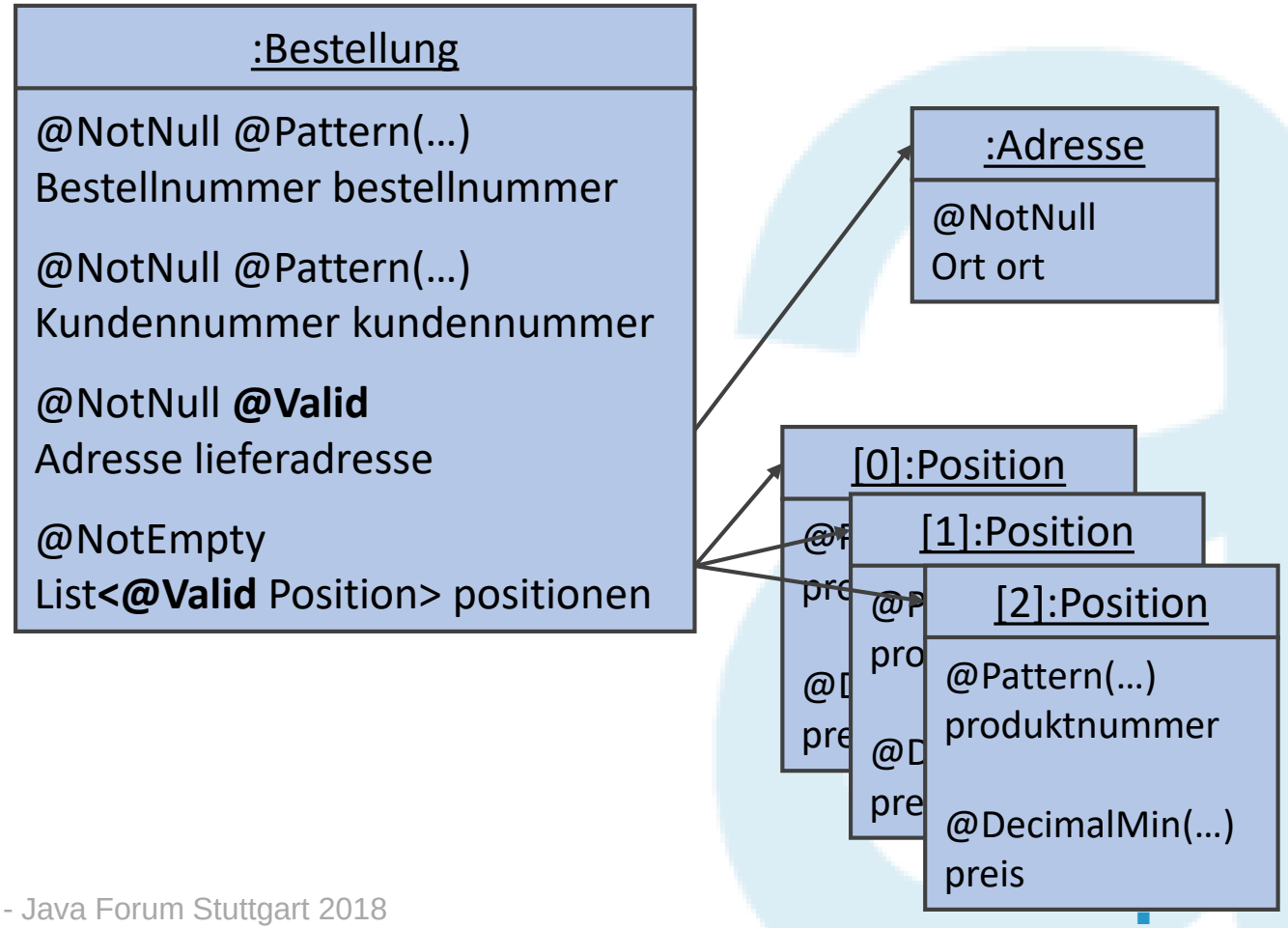
Beispiel - Genehmigung

```
public class License {
    @NotNull
    @Pattern(regexp = "^.{5,10}\\.my-company$")
    private   String   signerName;

    @Email
    @Pattern(regexp = "^.+?@my-company\\.de$")
    private   String   internalEmail;

    @NotNull
    @Past
    private   LocalDate   date;
}
```

Validieren von Objekt-Graphen

- Komplexere Objekt-Strukturen
  z. B. Bestellung mit n Positionen

- Annotation @Valid weist Validator an, das Objekt *hinter* dem annotierten Feld *auch* zu validieren
  - alle Positionen
    <@Valid Position>
  - die Lieferadresse
    @Valid Lieferadresse



Bean Validation

- Motivation
- Überblick
- Constraints
- **Einsatz**
- Custom Constraints
- Fragen

Was kann validiert werden?

- Felder

```
class MyClass {
    @NotNull          String name;
    @Size(min = 42)   String foo;
}
```

- Parameter von Methoden und Konstruktoren

```
void doIt(@NotNull       Long   id,
          @Pattern(...)  String login)
{ ... }
```

- Rückgabewerte von Methoden

```
@Future LocalDate getExpireDate(...)
{ ... }
```

Parametrisierung

- Message

```
@Future(message = "{error.expirationDateInPast}")
LocalDate expirationDate;
```

- Gruppen

```
public interface DateValidation {};
...
@Future(groups = DateValidation.class) LocalDate expirationDate;
```

- Payload

```
public class Severity {
        public static class Error implements Payload {};
}
...
@Future(payload = Severity.Error.class) LocalDate expirationDate;
```

Manuelle Validierung – Aufruf

```
try (ValidatorFactory  factory =
    Validation.buildDefaultValidatorFactory())
{
  Validator validator = factory.getValidator();

  Set<ConstraintViolation<>> violations =
    validator.validate(bean);
   //validator.validate(bean, groups);
}
```

Manuelle Validierung – Ergebnis auswerten

- von validator.validate(…) erhält man eine Menge (Set) von ConstraintViolation-Objekten
- ConstraintViolation
  - getMessage() – Nachricht für den Anwender (EL-Ausdrücke aufgelöst)
  - getMessageTemplate() – unaufgelöste Nachricht
  - getPropertyPath() – Pfad zum Feld, auf das sich die Meldung bezieht z. B. bestellung.pos[17].produktnummer
  - getInvalidValue() – Wert, der für ungültig befunden wurde
  - getLeafBean() – Objekt, das ungültiges Feld enthält
  - getConstraintDescription() – Informationen zur konkreten Annotation

Automatische Validierung

- EJBs
  (Session Beans)
- CDI
- JAX-RS
- JSF

```
@Stateless
public class MySessionBean {
  public LocalDate getDate(@NotBlank String key) { ... }
}
```

```
public class Bean {
  @Inject
  public Bean(@NotNull OtherBean other) { ... }
}
```

```
public class ResourceX {
  @POST
  @Path
  public Response setX(@Pattern(...) String body) { ... }
}
```

Bean Validation

- Motivation
- Überblick
- Constraints
- Einsatz
- **Custom Constraints**
- Fragen

Eigene Constraints

- Built-In Constraints oft nicht ausreichend
  - fachlicher Bezug schlecht darstellbar
  - wiederkehrende Kombination von Constraints
  - komplexere Zusammenhänge innerhalb eines Objektes oder -Graphen
- Definition von zusätzlichen Constraints als Annotation
  - Annotiert mit @Constraint(validatedBy = { ... })
  - Pflicht-Attribute, die von der Bean Validation verlangt werden
    - group zur Festlegung von Fällen, in denen der Constraint angewendet werden soll
    - payload zusätzliche Informationen für den *ConstraintValidator*
    - message Message(-Template) für die Fehlermeldung
  - ggfs. Attribute zur Parametrisierung

Aufbau einer Constraint-Annotation

- Annotation MyConstraint
- annotiert mit @Constraint
  - zeigt an, dass MyConstraint ein Constraint der Bean Validation sein soll
  - Attribut validatedBy gibt ggfs. Klasse zur Prüfung des Constraints an.
- notwendige Attribute
  - message
  - payload
  - groups

```
@Documented
@Constraint(validatedBy = {...})
@Target({ METHOD, FIELD, PARAMETER, TYPE })
@Retention(RetentionPolicy.RUNTIME)
public @interface MyConstraint
{
   String message()
         default "message";
   Class<? ...>[] payload()
         default {};
   Class<?>[] groups()
         default {};

   int moreInfo() default 42;
}
```

Composed Constraints

- Zusammenfassung wiederkehrender Kombinationen von Constraints
- *Sinnvolle / fachlich motivierte* Benennung von Constraint-Kombinationen

```
@NotNull
@Pattern(regexp = "...")
private   String   signerName;
```

```
@RequiredInternalSigner
private   String   signerName;
```

Composed Constraints

```
@Documented
@Constraint(validatedBy = {})
@NotNull(message = "{internal.signer.required}")
@Pattern(regexp  = "^.{5,10}\\.my-company$",
         message = "{internal.signer.suffix}")
@Target({ METHOD, FIELD, PARAMETER, TYPE })
@Retention(RetentionPolicy.RUNTIME)
@ReportAsSingleViolation //optional
public @interface RequiredInternalSigner
{
   String message() default "ignored-msg";
   Class<? extends Payload>[] payload() default {};
   Class<?>[] groups() default {};
}
```

Composed Constraints

```
public class License {
    @RequiredInternalSigner
    private String signerName;

    @InternalEmail
    private String internalEmail;

    @NotNull @Past
    private LocalDate date;
}
```

Custom Constraints

- Constraints für eigene Klassen
- Constraints für Nicht-Standard-Prüfungen

```
@Documented
@Constraint(validatedBy = InThePastValidator.class)
@Target({ METHOD, FIELD, PARAMETER, TYPE })
@Retention(RetentionPolicy.RUNTIME)
public @interface InThePast
{
    long min() default           0L;
    long max() default 100_000_000L;
    ChronoUnit timeUnit();
    String message() default "{date.in-the-past}";

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

Custom Constraints - ConstraintValidator

A: Eigene Constraint Annotation

T: Überprüfter Typ

```
public interface ConstraintValidator<A extends Annotation,T>
{
   default void initialize(A constraintAnnotation) { /* optional */ }

   boolean isValid(T value, ConstraintValidatorContext context);
}
```

Custom Constraints - ConstraintValidator

```
public class InThePastValidator implements ConstraintValidator<InThePast, LocalDate>
{
   private InThePast constraint;

   @Override
   public void initialize(InThePast constraintAnnotation)
   {
      this.constraint = constraintAnnotation;
   }

   @Override
   public boolean isValid(LocalDate value, ConstraintValidatorContext context)
   {
     ChronoUnit unit = this.currentAnnotation.timeUnit();
     return value.isBefore(LocalDate .now().minus(this.constraint.min(), unit))
         && value.isAfter(LocalDate.now().minus(this.constraint.max(), unit));
   }
}
```

Das wird benötigt, um die Attribute der konkreten Annotation zu erhalten:

- min
- max
- timeUnit

Value Extractors

- Validierung von Werten in Container-Klassen
  - z. B. Guava Table
- werden explizit registriert
  - über Service-Loader
    META-INF/services/javax.validation.valueextraction.ValueExtractor
  - beim Erstellen der ValidatorFactory
- implementieren ValueExtractor<T>
- lesen der Reihe nach Objekte aus einem Container und übergeben sie an Bean Validation zur Überprüfung

Fragen - Anregungen

Weitere Antworten am aformatik-Stand (Nr. 5) vor dem Hegel Saal oder per email an

nikolaos.ntaountakis@aformatik.de
andreas.geissel@aformatik.de