# Arrange, Act, Assert

|  |  |
| --- | --- |
| phase | explanation |
| arrange | creation of all objects (except for the SUT) that are necessary for test execution  creation of the object whose functionality will be tested, and setting it in  some initial state |
| act | execution of SUT methods to be tested |
| assert | verification of the test results |

## Übung: Benennen Sie die 3 Unit Test-Phasen am Beispiel **ch06.x03JUnitExecutionModel.JUnitExecutionModelBeforeTest.shouldAllowToAddAddress()**

Lösung:

## **Übung: die JUnit @Rule ExpectedException und “Arrange, Act, Assert”**

1. Bringen Sie Klasse HasExpectedException unter <https://github.com/junit-team/junit4/wiki/Rules> zum Laufen und analysieren Sie sie.   
   Hinweis: Dazu nötig ist der   
   **import** **static** org.hamcrest.core.StringStartsWith.*startsWith*;
2. Wenden Sie das unter a) gewonnen Wissen an und schreiben Sie ch05.x01ExpectedException.MoneyIAETest um, so dass die ExpectedException Rule verwendet wird.
3. Bewerten Sie diesen Ansatz hinsichtlich des Arrange, Act, Assert-Ablaufes

Lösung

Zu a) **ch07.x01ExpectedExceptionRule\_junit4\_wikiLösung.HasExpectedException**

Zu b) Lösung: **ch07.x02MoneyIAETestMitExpectedExceptionRuleLösung.MoneyIAETest**

zu c)

## **Demo: Wie man früher Exceptions getestet hat: die JUnit-fail()-Methode**

Test **ch07.x03PhoneIAETestMitTryFailCatch.ExpectedExceptionMessageTryCatchTest**

Bewertung:

* Der Ablauf des Tests ist verschleiert durch das try-catch statement.
* Die “geschluckte” Exception im catch-block trägt dazu bei.
* Die Benutzung von fail() als seine "invertierte" assertion sieht merkwürdig aus.

Wenn man doch nur dieses try catch loswerden könnte…

## **Neue Dependency**

|  |
| --- |
| <!-- https://mvnrepository.com/artifact/com.googlecode.catch-exception/catch-exception -->  <dependency>  <groupId>com.googlecode.catch-exception</groupId>  <artifactId>catch-exception</artifactId>  <version>1.2.0</version>  <scope>test</scope>  </dependency>  <!— benötigt folgende Abhängigkeit:  https://mvnrepository.com/artifact/org.mockito/mockito-core -->  <dependency>  <groupId>org.mockito</groupId>  <artifactId>mockito-core</artifactId>  <version>2.11.0</version>  <scope>test</scope>  </dependency> |

## **Anleitung: Überprüfen von Exception-Details unter Bewahrung von „Arrange, Act, Assert“ mit der Catch-Exception Library**

Gegeben: SUT **ch07.x03PhoneIAETestMitCatchExceptionLibrary.Phone**

Test: **ch07.x04PhoneIAETestMitCatchExceptionLibrary.PhoneTestMitCatchExceptionLibrary**

Man sieht: mit der catchException-library sind die Cons aus 7.2c) sind hier gelöst:

1. Die Reihenfolge Arrange Act Assert wird beibehalten

|  |  |
| --- | --- |
| phase | code example |
| arrange | Phone phone = **new** Phone(); |
| act | *catchException*(phone).setNumber(**null**); |
| assert | *assertTrue*(*caughtException*() **instanceof** IllegalArgumentException); |

1. Die geworfene Exception ist in der assert-Phase noch analysierbar:

*assertEquals*("number can not be null or empty", *caughtException*().getMessage());

## **Java 8 Lambdas vs catchException-Library**

In einem Blog-Beitrag mit dem Titel [8 WAYS OF HANDLING EXCEPTIONS IN JUNIT. WHICH ONE TO CHOOSE?](http://blog.codeleak.pl/2013/07/3-ways-of-handling-exceptions-in-junit.html) ( <http://blog.codeleak.pl/2013/07/3-ways-of-handling-exceptions-in-junit.html> ) führt der Autor nicht weniger als 8 Möglichkeiten auf, Exceptions in JUnit zu überprüfen.

4 davon,

* @Test(expected = IllegalArgumentException.**class**),
* @Rule ExpectedException,
* try{ fail();}catch() und die
* catchException-Bibliothek

wurden bereits demonstriert. Von diesen 4 Varianten bewahrt lediglich die catchException-Bibliothek die arrange-act-assert-Reihenfolge (die AssertJ-Variante sieht jedoch vielversprechend aus: <http://blog.codeleak.pl/2015/04/junit-testing-exceptions-with-java-8.html> mit demos: <https://github.com/kolorobot/unit-testing-demo> )

Der originale Autor der catchException-library, Rod Woo, erkannte, dass sich das Verhalten seiner library mit Java 8-Lambdas leichter und eleganter darstellen lässt und stellte daraufhin die Weiterentwicklung ein (s. <http://mariuszs.github.io/blog/2014/catch-exception.html> ). Die catchException-library wird nun an anderer Stelle weiterentwickelt: <https://github.com/Codearte/catch-exception>

### **Demo: Java 8 Lambdas zum Auffangen einer geworfenen Exception**

Ein Code-Beispiele wie mit Java 8 Lambdas analoges zur catchException-library erreicht werden kann ist z.B. unter

<http://www.codeaffine.com/2014/07/28/clean-junit-throwable-tests-with-java-8-lambdas/>

Das dortige Beispiel ist im workspace unter package **ch07.x05JUnit4\_ExceptionsMitJava8Lambda**

### **Übung: Java 8 Lambdas zum Auffangen einer geworfenen Exception**

Schreiben Sie den Test unter **ch07.x04PhoneIAETestMitCatchExceptionLibrary.PhoneTestMitCatchExceptionLibrary** um sodass er Java 8 Lambdas verwendet.

Lösung: package **ch07.x05JUnit4\_ExceptionsMitJava8LambdaLösung**

Man sieht: auch hier ist die *arrange, act, assert*-Reihenfolge bewahrt.

## **JUnit 5 und assertThrows**

Die neue JUnit-Version 5 enthält wiederum ein feature, welches analog zu der Java 8 Lambda-Variante aufgebaut ist.

Zum Zeitpunkt des Schreibens dieses Scripts existieren jedoch diverse Inkompatibilitäten beim Zusammenspiel zwischen JUnit 5, catchException 1.x und Mockito 2.x. Dies beginnt schon bei den Maven Dependencies. Deshalb ist das folgende Demo in einen separaten workspace mit eigener pom.xml ausgelagert.

### **Demo: Wie man das Beispiel unter** http://blog.codeleak.pl/2017/06/testing-exceptions-with-junit-5.html **zum Laufen bringt.**

Über die notwendigen maven-dependencies wird unter <https://www.petrikainulainen.net/programming/testing/junit-5-tutorial-running-unit-tests-with-maven/> geschrieben.

Abhängig von der verwendeten IDE (z.B. Eclipse) wird JUnit 5 (noch) nicht unterstützt. In diesem Fall kann man, wie schon in **ch04.x01JUnitParameterized** den (optionalen) **TestRunner** schreiben. Dieser muss jedoch für JUnit 5 gänzlich anders aussehen, wie unter <http://junit.org/junit5/docs/current/user-guide/#launcher-api> gezeigt.

Die maven-dependencies sind

|  |
| --- |
| <dependency>  <groupId>org.junit.jupiter</groupId>  <artifactId>junit-jupiter-api</artifactId>  <version>5.0.1</version>  <scope>test</scope>  </dependency>  <dependency>  <groupId>org.junit.jupiter</groupId>  <artifactId>junit-jupiter-engine</artifactId>  <version>5.0.1</version>  <scope>test</scope>  </dependency>  <dependency>  <groupId>org.junit.vintage</groupId>  <artifactId>junit-vintage-engine</artifactId>  <version>4.12.1</version>  <scope>test</scope>  </dependency>  <dependency>  <groupId>org.junit.platform</groupId>  <artifactId>junit-platform-launcher</artifactId>  <version>1.0.1</version>  <scope>test</scope>  </dependency>  <dependency>  <groupId>org.junit.platform</groupId>  <artifactId>junit-platform-runner</artifactId>  <version>1.0.1</version>  <scope>test</scope>  </dependency> |

Die Klassen sind im package **ch07.x05JUnit5\_assertThrows**

## **Weitere (interessantere) Exception-Details überprüfen: die cause der Exception**

In vielen APIs gibt erst die cause einer geworfenen Exception Aufschluss über die wahre Ursache des Fehlers. Beispiel: *java.sql.SQLException* Dies ist ein viel lohnenderes Ziel für *asserts* in Unit-Tests.

Das BeanValidation-Framework ( <http://beanvalidation.org/> ), als weiteres Beispiel, enthält die ConstraintViolationException, welche im Falle einer missglückten Validierung von z.B. Eingabedaten geworfen werden kann. Diese Exception wiederum enthält ein ein Set von COnstraintViolation-Objekten, welche genaue Informationen über die erfolge Validierungsverletzung enthalten.

Nach der folgenden kurzen Einführung folgt hierzu eine Übung:

### **BeanValidation** [**http://beanvalidation.org/**](http://beanvalidation.org/)

Die software-architektonischen Vorteile, die auf der Homepage genannt sind, sprechen für sich.

Besonders hervorzuheben sind:

1. “While you can run validation manually, it is more natural to let other specifications and frameworks validate data at the right time (user input in presentation frameworks, business service execution by CDI, entity insert or update by JPA).”
   * BeanValidation ist also automatisch(!) integriert in
     + JPA
     + CDI
     + JSF
     + und (auf der Seite nicht einmal erwähnt) JAX-RS (für Ressourcen-Parameter und Rückgabewerte oder auch Fields von Ressourcen-Klassen)
   * Integrierungen für andere Dependency-Injection-Frameworks wie Guice oder Spring existiert ebenfalls.
2. der Bulletpoint: “reports the set of violations (localized)”
3. der Bulletpoint: “lets you write custom constraints in an extensible way”

zu a) Benutzt man also BeanValidation in seiner integrierten Form, so wird eine Exception aus der jeweiligen API geworfen. Im Falle von JPA z.B. bekäme man beim Fehlschlagen der (dort automatischen) Validierung eine PersistenceException oder RollbackException. Diese enthält als cause eine BeanValidation-eigene *ConstrainViolationException.*

Die auf der Homepage erwähnte manuelle Validierung wiederum kann man über eine injizierte Instanz des interfaces [*javax*](eclipse-javadoc:%E2%98%82=di-google-guiceBeanValidation/C:%5C/Users%5C/User%5C/.m2%5C/repository%5C/javax%5C/validation%5C/validation-api%5C/1.1.0.Final%5C/validation-api-1.1.0.Final.jar%3Cjavax)*.*[*validation*](eclipse-javadoc:%E2%98%82=di-google-guiceBeanValidation/C:%5C/Users%5C/User%5C/.m2%5C/repository%5C/javax%5C/validation%5C/validation-api%5C/1.1.0.Final%5C/validation-api-1.1.0.Final.jar%3Cjavax.validation)*.Validator* vornehmen.

Zu b) Ein Aufruf von Validator.validate(zuValidierendesObjekt) gibt ein *Set<ConstraintViolation>*. Im Falle der automatischen Validierung ist dieses Set in der *ConstrainViolationException* welche ja als cause der jeweiligen API-eigenen Exception (s.o.) mitgegeben wurde. Dieses Set kann dann in einem catch-Block ausgelesen werden.

Zu c) Hier implementiert man das Interface javax.validation.ConstraintValidator und schreibt eine eigene Annotation, welche auf diesen Validator Bezug nimmt (tatsächlich ist die Bezugnahme bidirektional):

|  |
| --- |
| **public** **class** CommonsEmailValidator **implements** ConstraintValidator<**Email**, String> {  (..)  } |

Mit

|  |
| --- |
| @Constraint(validatedBy = {**CommonsEmailValidator**.**class**})  **public** **@interface** Email {  (…)  } |

Folgende Demos führen BeanValidation mit einfachen Code-Beispielen ein:

### **Demo: BeanValidation in Java SE**

Maven-dependencies:

|  |
| --- |
| <dependency>  <groupId>javax.validation</groupId>  <artifactId>validation-api</artifactId>  <version>2.0.0.Final</version>  </dependency>  <dependency>  <groupId>org.hibernate.validator</groupId>  <artifactId>hibernate-validator</artifactId>  <version>6.0.2.Final</version>  </dependency>  <dependency>  <groupId>org.hibernate.validator</groupId>  <artifactId>hibernate-validator-annotation-processor</artifactId>  <version>6.0.2.Final</version>  </dependency>  <!-- https://stackoverflow.com/questions/42718869/hibernate-validation-unable-to-initialize-javax-el-expressionfactory-error -->  <dependency>  <groupId>org.glassfish</groupId>  <artifactId>javax.el</artifactId>  <version>3.0.1-b08</version>  </dependency> |

Klassen in package **ch07.x07BeanValidationJavaSEdemo**

### **Demo: BeanValidation in Spring**

Maven-dependencies:

|  |
| --- |
| <dependency>  <groupId>org.springframework</groupId>  <artifactId>spring-context</artifactId>  <version>4.3.10.RELEASE</version>  </dependency>  <dependency>  <groupId>org.springframework</groupId>  <artifactId>spring-test</artifactId>  <version>4.3.10.RELEASE</version>  </dependency>  <!-- https://mvnrepository.com/artifact/javax.inject/javax.inject -->  <dependency>  <groupId>javax.inject</groupId>  <artifactId>javax.inject</artifactId>  <version>1</version>  </dependency> |

Klassen in package **ch07.x08BeanValidationMitSpringDemo**

### **Demo: BeanValidation in JPA integriert (wirft als cause eine ConstraintViolationException )**

Maven-dependencies:

|  |
| --- |
| <dependency>  <groupId>org.springframework</groupId>  <artifactId>spring-tx</artifactId>  <version>4.3.10.RELEASE</version>  </dependency>  <dependency>  <groupId>org.springframework</groupId>  <artifactId>spring-orm</artifactId>  <version>4.3.10.RELEASE</version>  </dependency>  <dependency>  <groupId>org.hsqldb</groupId>  <artifactId>hsqldb</artifactId>  <version>2.3.4</version>  </dependency>  <dependency>  <groupId>org.hibernate.javax.persistence</groupId>  <artifactId>hibernate-jpa-2.1-api</artifactId>  <version>1.0.0.Final</version>  </dependency>  <dependency>  <groupId>org.hibernate</groupId>  <artifactId>hibernate-core</artifactId>  <version>5.2.5.Final</version>  </dependency> |

Klassen in package **ch07.x08BeanValidationMitSpringDemo**

Lässt man die Anwendung laufen, so sieht man in der Konsole:

|  |
| --- |
| Caused by: javax.validation.ConstraintViolationException: Validation failed for classes [ch07.x09BeanValidationMitSpringInJPAintegriert.User] during persist time for groups [javax.validation.groups.Default, ]  List of constraint violations:[  ConstraintViolationImpl{interpolatedMessage='About Me must be between 10 and 200 characters', propertyPath=aboutMe, rootBeanClass=class ch07.x09BeanValidationMitSpringInJPAintegriert.User, messageTemplate='About Me must be between 10 and 200 characters'}  ConstraintViolationImpl{interpolatedMessage='Age should not be less than 18', propertyPath=age, rootBeanClass=class ch07.x09BeanValidationMitSpringInJPAintegriert.User, messageTemplate='Age should not be less than 18'}  ConstraintViolationImpl{interpolatedMessage='muss wahr sein', propertyPath=working, rootBeanClass=class ch07.x09BeanValidationMitSpringInJPAintegriert.User, messageTemplate='{javax.validation.constraints.AssertTrue.message}'}  ConstraintViolationImpl{interpolatedMessage='Name cannot be null', propertyPath=name, rootBeanClass=class ch07.x09BeanValidationMitSpringInJPAintegriert.User, messageTemplate='Name cannot be null'}  ConstraintViolationImpl{interpolatedMessage='Email should be valid', propertyPath=email, rootBeanClass=class ch07.x09BeanValidationMitSpringInJPAintegriert.User, messageTemplate='Email should be valid'} |

### **Übung: Schreiben Sie einen Junit-Test, der das Set con ConstrainViolations überprüft für entsprechende Daten in einem User-Objekt**

Lösung: TODO

### **Demo: BeanValidation in einer einfachen Guice-Anwendung mit manueller Validierung**

Dazu notwendig:

|  |
| --- |
| <dependency>  <groupId>com.google.inject</groupId>  <artifactId>guice</artifactId>  <version>3.0</version>  </dependency>  <dependency>  <groupId>javax.inject</groupId>  <artifactId>javax.inject</artifactId>  <version>1</version>  </dependency>  <!-- BeanValidation and Hibernate Validator. -->  <dependency>  <groupId>javax.validation</groupId>  <artifactId>validation-api</artifactId>  <version>1.1.0.Final</version>  </dependency>  <dependency>  <groupId>org.hibernate</groupId>  <artifactId>hibernate-validator</artifactId>  <version>5.2.4.Final</version>  </dependency>  <dependency>  <groupId>javax.el</groupId>  <artifactId>javax.el-api</artifactId>  <version>2.2.4</version>  </dependency>  <dependency>  <groupId>org.glassfish.web</groupId>  <artifactId>javax.el</artifactId>  <version>2.2.4</version>  </dependency>  <dependency>  <groupId>org.hibernate</groupId>  <artifactId>hibernate-validator-cdi</artifactId>  <version>5.2.4.Final</version>  </dependency>  <!-- Apache Commons Email-Validator (zur Implementierung einer eigenen ConstrainValidator-Implementierung )-->  <dependency>  <groupId>commons-validator</groupId>  <artifactId>commons-validator</artifactId>  <version>1.5.0</version>  </dependency>  <!-- End BeanValidation and Hibernate Validator. --> |

Gegeben: SUT **ch07.x05BeanValidationMitGuiceDemo.MyApplication**

Sowie ein Java SE Client: **ch07.x05BeanValidationMitGuiceDemo.client.ClientApplication**

Hinweise:

Beachten Sie:

* die BeanValidation Annotations in Klasse ch07.x05BeanValidationMitGuiceDemo.domain.Message:

|  |
| --- |
| **public** **class** Message {  @NotNull  @Size(min=30)  **private** String msg;  @Email  **private** String rec;  (…)  } |

* die Konfiguration des Validators in ch07.x05BeanValidationMitGuiceDemo.AppInjector.configure()

|  |
| --- |
| **protected** **void** configure() {  ValidatorFactory factory = Validation.*buildDefaultValidatorFactory*();  Validator validator= factory.getValidator();  (…)  bind(Validator.**class**).toInstance(validator);  } |

Sowie das anschließende Injizieren des Validators in die SUT

|  |
| --- |
| **public** **class** MyApplication {  @Inject  Validator validator;  (…)  } |

Beachten Sie ferner die MyGuiceException. Mit ihr wird jene Exception nachgebildet, welche von jener API geworfen wird in die BeanValidation integriert ist.

Beachten Sie nun, welche umfangreichen Informationen über die fehlgeschlagene Validierung sie aus den ConstrainViolations entnehmen können:

|  |
| --- |
| Set<ConstraintViolation<?>> set = ((ConstraintViolationException)cause).getConstraintViolations();  **for** (ConstraintViolation<?> cv : set) {  System.***err***.println(cv.getLeafBean().getClass().getSimpleName()+"."+cv.getPropertyPath()+ "= "+cv.getInvalidValue()+" "+cv.getMessage());  } |

Beachten Sie schließlich insbesondere die Fehlermeldung für die fehlerhafte email-Adresse:

|  |
| --- |
| Message.rec= chris@abc keine gültige E-Mail-Adresse |

Und vergleichen Sie sie mit jener Stelle, in der sie festgelegt wird:

|  |
| --- |
| @Constraint(validatedBy = {CommonsEmailValidator.**class**})  **public** **@interface** Email {  String message() **default** "{org.hibernate.validator.constraints.Email.message}";  (…)  } |

Dies belegt das “(localized)“ in 7.6.1 b) : Hierbei ist nämlich org.hibernate.validator.constraints.Email.message nur der Key für die properties-Datei mit den Fehlermeldungen, welche, wie man an der deutschen Fehlermeldung sieht, zudem noch vom BV-Provider Hibernate-Vaidation internationalisiert sind!

### Übung: Schreiben Sie eine Testklasse, welche 2 @Test-Methoden enthält, die für jede der beiden annotierten Fields einen parametrisierten Test für verschiedene repräsentative Fehlerwerte ausführt. Benutzen Sie die catch-exception library.

Lösung: **ch07.x05BeanValidationMitGuiceDemoLösung.MyApplicationTest**