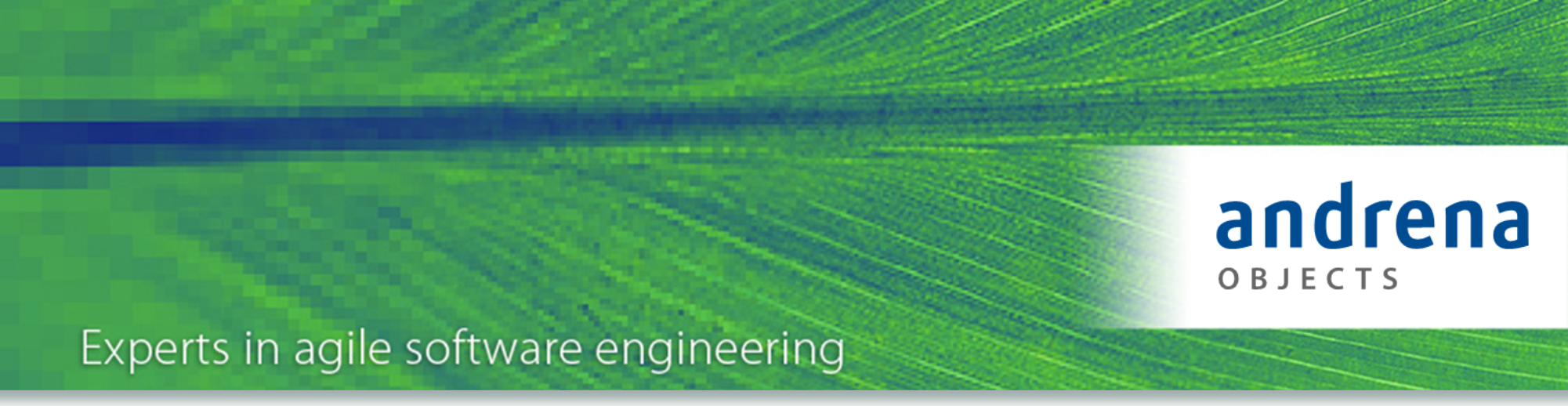
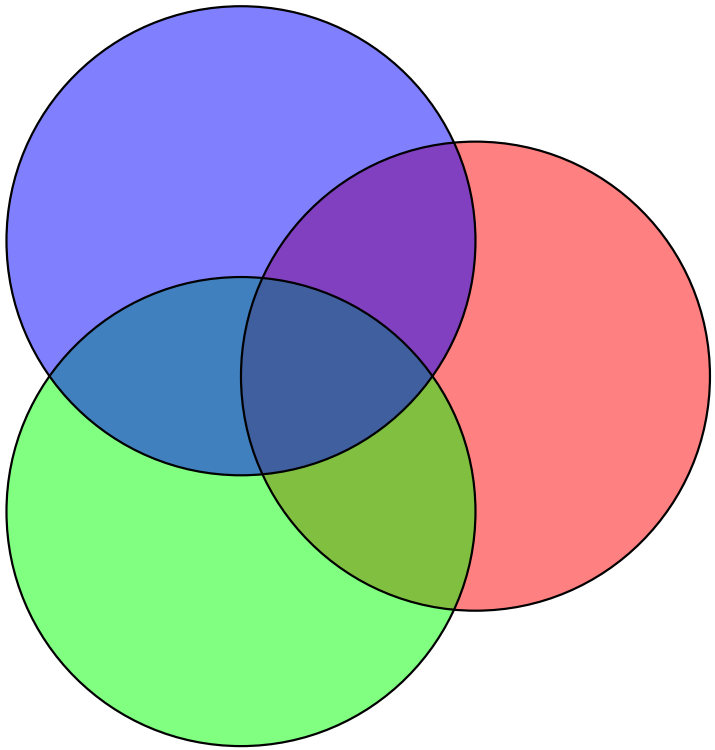


JUnit 4.11 – Die Neuerungen

Marc Philipp

VKSI Sneak Preview, 18. September 2013

Über JUnit



Kent Beck:

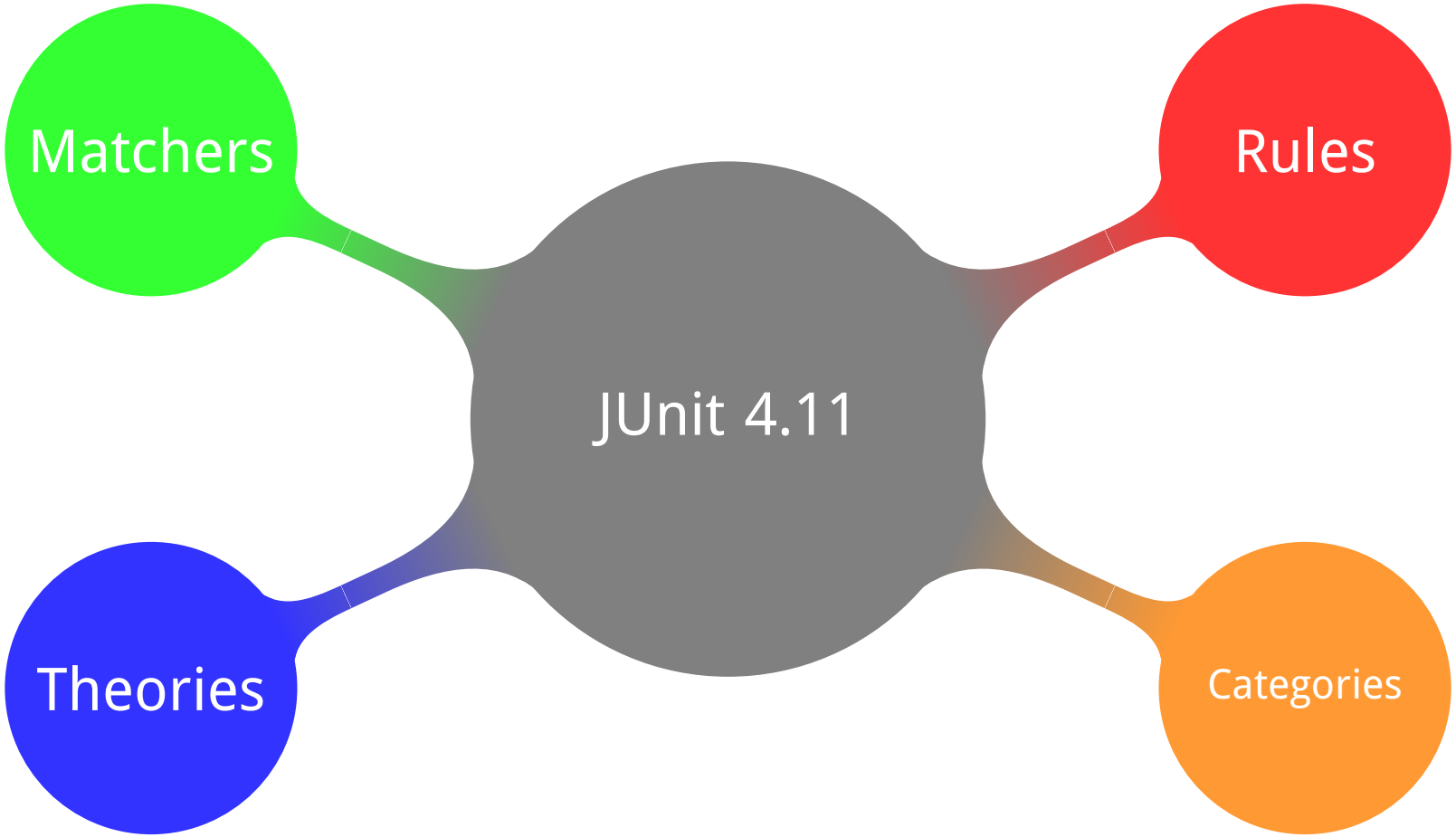
A programmer-oriented testing framework for Java.

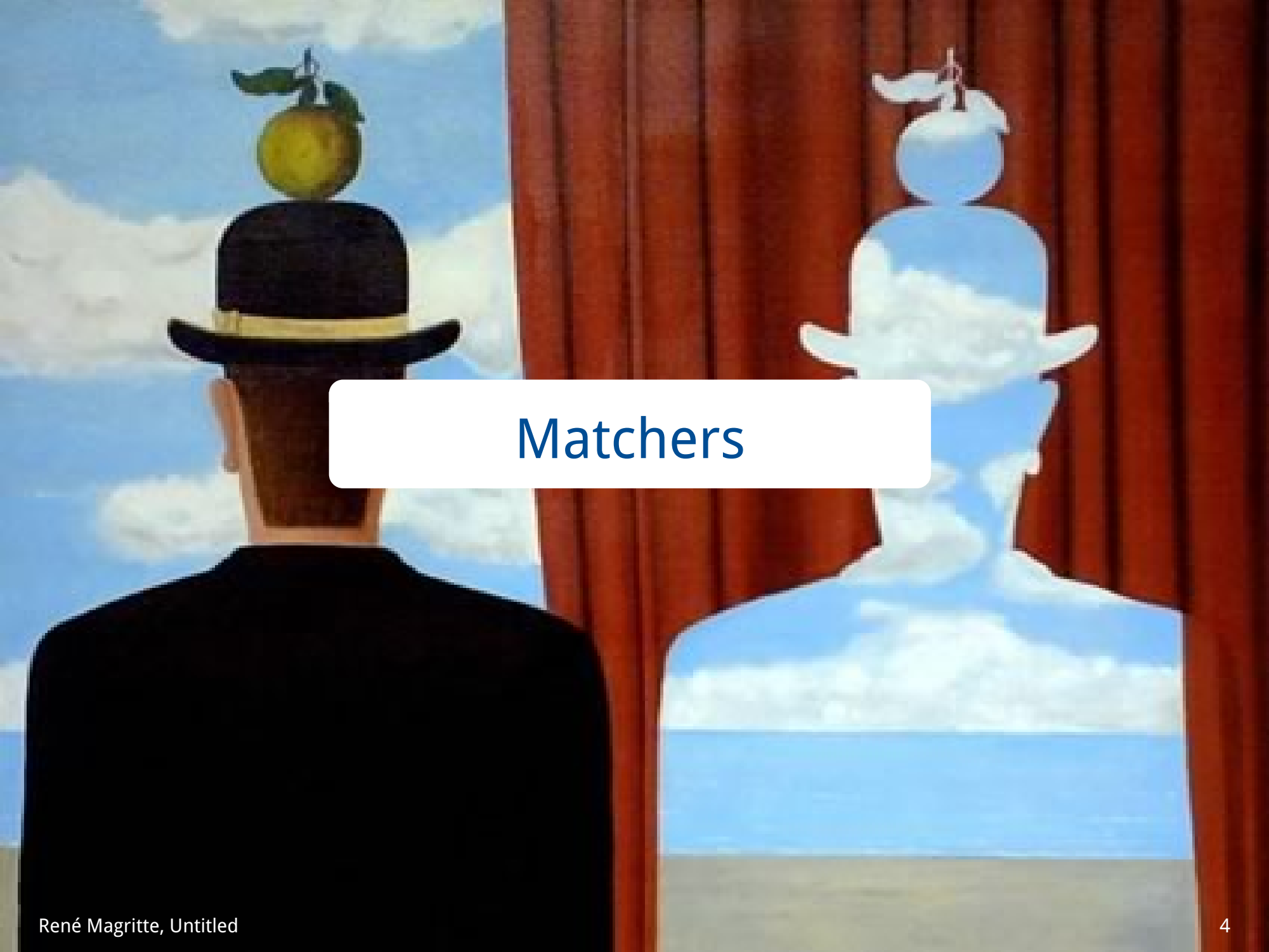
David Saff:

JUnit is the intersection of all possible useful Java test frameworks, not their union.

Nicht nur für Unit Tests!

Neue Features seit Version 4.0





Matchers

René Magritte, Untitled

Neue Assertion: `assertThat(...)`

- Neue `Assert`-Methoden:

```
<T> void assertThat(T actual, Matcher<? super T> matcher)
<T> void assertThat(String reason, T actual, Matcher<? super T> matcher)
```

- Parameters:

reason	Zusätzliche Beschreibung für den Fehlerfall (optional)
actual	Tatsächlicher Wert
matcher	Hamcrest-Matcher überprüft tatsächlichen Wert

Bessere Lesbarkeit

```
import static org.hamcrest.CoreMatchers.is;

public class BetterReadability {

   @Test public void withoutMatchers() {
      assertEquals(2, 1 + 1);
   }

   @Test public void withMatchers() {
      assertThat(1 + 1, is(2));
   }
}
```

- Kein Raten mehr, was Erwartung bzw. tatsächliches Ergebnis ist
- Häufig besser lesbar als herkömmliche Assertions

Matcher kombinieren

Matcher lassen sich einfach kombinieren:

```
assertThat( 1 + 1, is( not( 3 ) ) );

assertThat( 1 + 1, is( both( greaterThan( 1 ) ).and( lessThan( 3 ) ) ) );
assertThat( 1 + 1, is( allOf( greaterThan( 1 ), lessThan( 3 ) ) ) );

assertThat( 1 + 1, is( either( greaterThan( 1 ) ).or( greaterThan( 3 ) ) ) );
assertThat( 1 + 1, is( anyOf( greaterThan( 1 ), greaterThan( 3 ) ) ) );
```

Aussagekräftige Fehlermeldungen

Herkömmliche Zusicherung ohne Beschreibung

```
  assertFalse(asList(1, 2, 3).contains(2));
```

Ergebnis:

```
AssertionError: at de.andrena.junit...
```

Neue Zusicherung mit Matcher

```
  assertThat(asList(1, 2, 3), not(hasItem(2)));
```

Ergebnis:

```
AssertionError:
Expected: not a collection containing <2>
     but: was <[1, 2, 3]>
```

Vordefinierte Matcher

Allgemein

```
assertThat( null, is( nullValue() ) );

int[] results = { 1, 3, 5, 7 };
assertThat( 3, isIn( results ) );    // Auch mit Collections

assertThat( 3, isOneOf( 1, 3, 5, 7 ) );
```

Vordefinierte Matcher

Strings

```
assertThat( "Hallo Welt", containsString( "o We" ) );
assertThat( "Hallo Welt", startsWith( "Hall" ) );
assertThat( "Hallo Welt", endsWith( "elt" ) );

assertThat( "Hallo Welt", is( equalToIgnoringCase( "hallO welT" ) ) );
assertThat( "Hallo Welt", is( equalToIgnoringWhiteSpace( " Hallo    Welt " ) ) );

assertThat( "", isEmptyString() );
assertThat( "", isEmptyOrNullString() );
```

Vordefinierte Matcher

Comparables

```
assertThat( 1 + 1, comparesEqualTo( 2 ) );

assertThat( 1 + 1, greaterThan( 1 ) );

assertThat( 1 + 1, greaterThanOrEqualTo( 0 ) );

assertThat( 2.00001, is( closeTo( 2.0, 0.001 ) ) );
```

Vordefinierte Matcher

Iterables

```
List<Integer> list = Arrays.asList(1, 3, 5, 7);

assertThat( list, hasItem( 1 ) );
assertThat( list, hasItem( greaterThan( 6 ) ) );

assertThat( list, hasItems( 5, 1 ) );
assertThat( list, hasItems( lessThan( 4 ), greaterThan( 6 ) ) );

assertThat( list, everyItem( greaterThanOrEqualTo( 1 ) ) );
```

```
assertThat( list, contains( 1, 3, 5, 7 ) ); // auch mit Matchern

assertThat( list, containsInAnyOrder( 5, 1, 7, 3 ) );
assertThat( list, containsInAnyOrder( is( 5 ), greaterThan( 2 ),
     greaterThanOrEqualTo( 6 ), lessThan( 4 ) ) );

// auch mit Matchern:
assertThat( list, hasSize( greaterThanOrEqualTo( 3 ) ) );
assertThat( list, Matchers.<Integer> iterableWithSize( lessThan( 10 ) ) );
```

Vordefinierte Matcher

Maps

```
Map<Integer, String> map = new HashMap<Integer, String>();
map.put( 1, "one" );


assertThat( map, hasKey( 1 ) );

assertThat( map, hasValue( "one" ) );

assertThat( map, hasEntry( 1, "one" ) );

// natuerlich auch mit Matchern :-)
```

. . . und viele mehr . . .

Hamcrest 1.3 *Quick Reference*

Core | Library

General purpose

```
is(T)
equalTo(T)
not(T)                                  : Matcher<T>
```

```
anything()
anything(String)                        : Matcher<Object>
```

```
any(Class<T>)
instanceOf(Class<?>)
isA(Class<T>)                           : Matcher<T>
```

```
nullValue()                             : Matcher<Object>
nullValue(Class<T>)                     : Matcher<T>
notNullValue()                          : Matcher<Object>
notNullValue(Class<T>)                  : Matcher<T>
```

```
sameInstance(T)
theInstance(T)                          : Matcher<T>
```

```
isIn(Collection<T>)
isIn(T[])
isOneOf(T...)
hasToString(String)
hasToString(Matcher<? super String>)    : Matcher<T>
```

Combining multiple matchers

```
is(Matcher<T>)
not(Matcher<T>)                         : Matcher<T>
```

```
allOf(Matcher<? super T>...)
allOf(Iterable<Matcher<? super T>>)
anyOf(Matcher<? super T>...)
anyOf(Iterable<Matcher<? super T>>)     : Matcher<T>
```

```
both(Matcher<? super LHS>)
either(Matcher<? super LHS>)            : Matcher<LHS>
```

```
describedAs(String, Matcher<T>, Object...)  : Matcher<T>
```

Strings

```
containsString(String)
startsWith(String)
endsWith(String)                        : Matcher<String>
```

```
equalToIgnoringCase(String)
equalToIgnoringWhiteSpace(String)       : Matcher<String>
```

```
isEmptyString()
isEmptyOrNullString()                   : Matcher<String>
```

```
stringContainsInOrder(Iterable<String>) : Matcher<String>
```

Iterables

```
everyItem(Matcher<U>)                   : Matcher<Iterable<U>>
```

```
hasItem(T)
hasItem(Matcher<? super T>)  : Matcher<Iterable<? super T>>
```

```
hasItems(T...)
hasItems(Matcher<? super T>...)         : Matcher<Iterable<T>>
```

```
emptyIterable()              : Matcher<Iterable<? extends E>>
emptyIterableOf(Class<E>)               : Matcher<Iterable<E>>
```

```
contains(E...)
contains(Matcher<? super E>...)
contains(Matcher<? super E>)
contains(List<Matcher<? super E>>)
                             : Matcher<Iterable<? extends E>>
```

```
containsInAnyOrder(T...)
containsInAnyOrder(Collection<Matcher<? super T>>)
containsInAnyOrder(Matcher<? super T>...)
containsInAnyOrder(Matcher<? super E>)
                             : Matcher<Iterable<? extends E>>
```

```
iterableWithSize(Matcher<? super Integer>)
iterableWithSize(int)                   : Matcher<Iterable<E>>
```

Collections

```
hasSize(int)
hasSize(Matcher<? super Integer>)
                          : Matcher<Collection<? extends E>>
```

```
empty()                   : Matcher<Collection<? extends E>>
emptyCollectionOf(Class<E>)       : Matcher<Collection<E>>
```

Arrays

```
array(Matcher<? super T>...)                : Matcher<T[]>
```

```
hasItemInArray(T)
hasItemInArray(Matcher<? super T>)          : Matcher<T[]>
```

```
arrayContaining(E...)
arrayContaining(List<Matcher<? super E>>)
arrayContaining(Matcher<? super E>...)      : Matcher<E[]>
```

```
arrayContainingInAnyOrder(E...)
arrayContainingInAnyOrder(Matcher<? super E>...)
arrayContainingInAnyOrder(Collection<Matcher<? super E>>)
                                            : Matcher<E[]>
```

```
arrayWithSize(int)
arrayWithSize(Matcher<? super Integer>)
emptyArray()                                : Matcher<E[]>
```

Maps

```
hasEntry(K, V)
hasEntry(Matcher<? super K>, Matcher<? super V>)
                  : Matcher<Map<? extends K, ? extends V>>
```

```
hasKey(K)
hasKey(Matcher<? super K>)   : Matcher<Map<? extends K, ?>>
```

```
hasValue(V)
hasValue(Matcher<? super V>) : Matcher<Map<?, ? extends V>>
```

Beans

```
hasProperty(String)
hasProperty(String, Matcher<?>)
samePropertyValuesAs(T)                     : Matcher<T>
```

Comparables

```
comparesEqualTo(T extends Comparable<T>)
greaterThan(T extends Comparable<T>)
greaterThanOrEqualTo(T extends Comparable<T>)
lessThan(T extends Comparable<T>)
lessThanOrEqualTo(T extends Comparable<T>)    : Matcher<T>
```

Numbers

```
closeTo(double, double)                  : Matcher<Double>
closeTo(BigDecimal, BigDecimal)      : Matcher<BigDecimal>
```

Classes

```
typeCompatibleWith(Class<T>)  : Matcher<java.lang.Class<?>>
```

EventObjects

```
eventFrom(Object)
eventFrom(Class<? extends EventObject>, Object)
                                      : Matcher<EventObject>
```

DOM

```
hasXPath(String)
hasXPath(String, NamespaceContext)
hasXPath(String, Matcher<String>)
hasXPath(String, NamespaceContext, Matcher<String>)
                                   : Matcher<org.w3c.dom.Node>
```

Created by Marc Philipp, http://www.marcphilipp.de

http://www.marcphilipp.de/blog/2013/01/02/hamcrest-quick-reference/

Wie verwende ich Hamcrest Matchers?

- Ab Version 4.11 wird JUnit ohne Matcher ausgeliefert
- Um sie zu benutzen, muss Hamcrest als zusätzliche Abhängigkeit hinzugefügt werden.
  - `hamcrest-core.jar` und `hamcrest-library.jar` (oder `hamcrest-all.jar`)
  - `org.hamcrest.Matchers` enthält alle vordefinierten Matcher.
- Darüber hinaus lassen sich eigene Matcher definieren.

Ein eigener Matcher

Implementierung

```
public class IsEmptyCollection extends TypeSafeMatcher<Collection<?>> {

   @Override public void describeTo(Description description) {
      description.appendText("empty collection");
   }

   @Override protected boolean matchesSafely(Collection<?> collection) {
      return collection.isEmpty();
   }

   @Override public void describeMismatchSafely(Collection<?> collection,
        Description description) {
      description.appendText("size was " + collection.size());
   }

   @Factory public static Matcher<Collection<?>> empty() {
      return new IsEmptyCollection();
   }
}
```

Ein eigener Matcher

Benutzung

```
@Test public void isEmpty() {
   Set<String> set = new TreeSet<String>();
   set.add("a");

   assertThat(set, new IsEmptyCollection());   // Direkte Instantiierung
   assertThat(set, IsEmptyCollection.empty()); // Mit @Factory-Methode
   assertThat(set, empty());                   // Per Static Import
   assertThat(set, is(empty()));               // Syntactic Sugar
}
```

Result:

```
Expected: empty collection
     but: size was 1
```

Feature Matchers

Implementierung

```
public class CollectionSizeMatcher extends FeatureMatcher<Collection<?>, Integer> {

   public CollectionSizeMatcher(Matcher<? super Integer> subMatcher) {
      super(subMatcher, "collection with size", "size");
   }

   @Override protected Integer featureValueOf(Collection<?> actual) {
      return actual.size();
   }

   @Factory public static Matcher<Collection<?>> hasSize(Matcher<? super Integer>
        matcher) {
      return new CollectionSizeMatcher(matcher);
   }
}
```

Feature Matchers

Benutzung

```
   @Test public void size() {
      assertThat(Arrays.asList("a"), hasSize(equalTo(2)));
   }
```

Result:

```
Expected: collection with size <2>
     but: size was <1>
```

Fazit: Matchers

- Assertions lassen sich oft eleganter formulieren
- Manchmal sind die alten Assertion-Methoden klarer

Problem

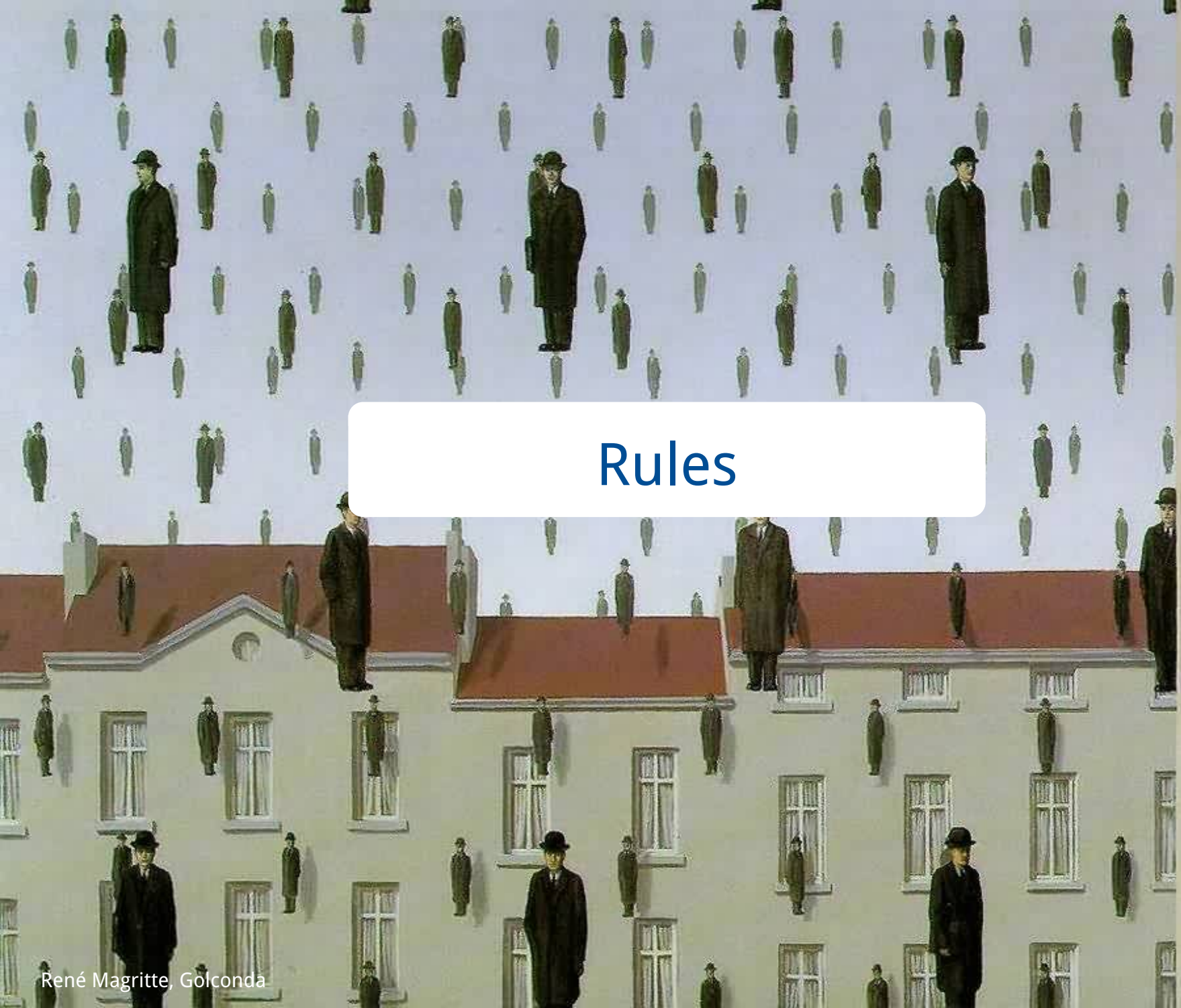
Javas Typsystem macht einem ab und zu einen Strich durch die Rechnung

- Boxing notwendig bei primitiven Typen

```
assertThat( 1 + 1, is( 2 ) )
```

- Mangelnde Typinferenz

```
assertThat( list, Matchers.<Integer> iterableWithSize( lessThan( 10 ) ) );
```



Rules

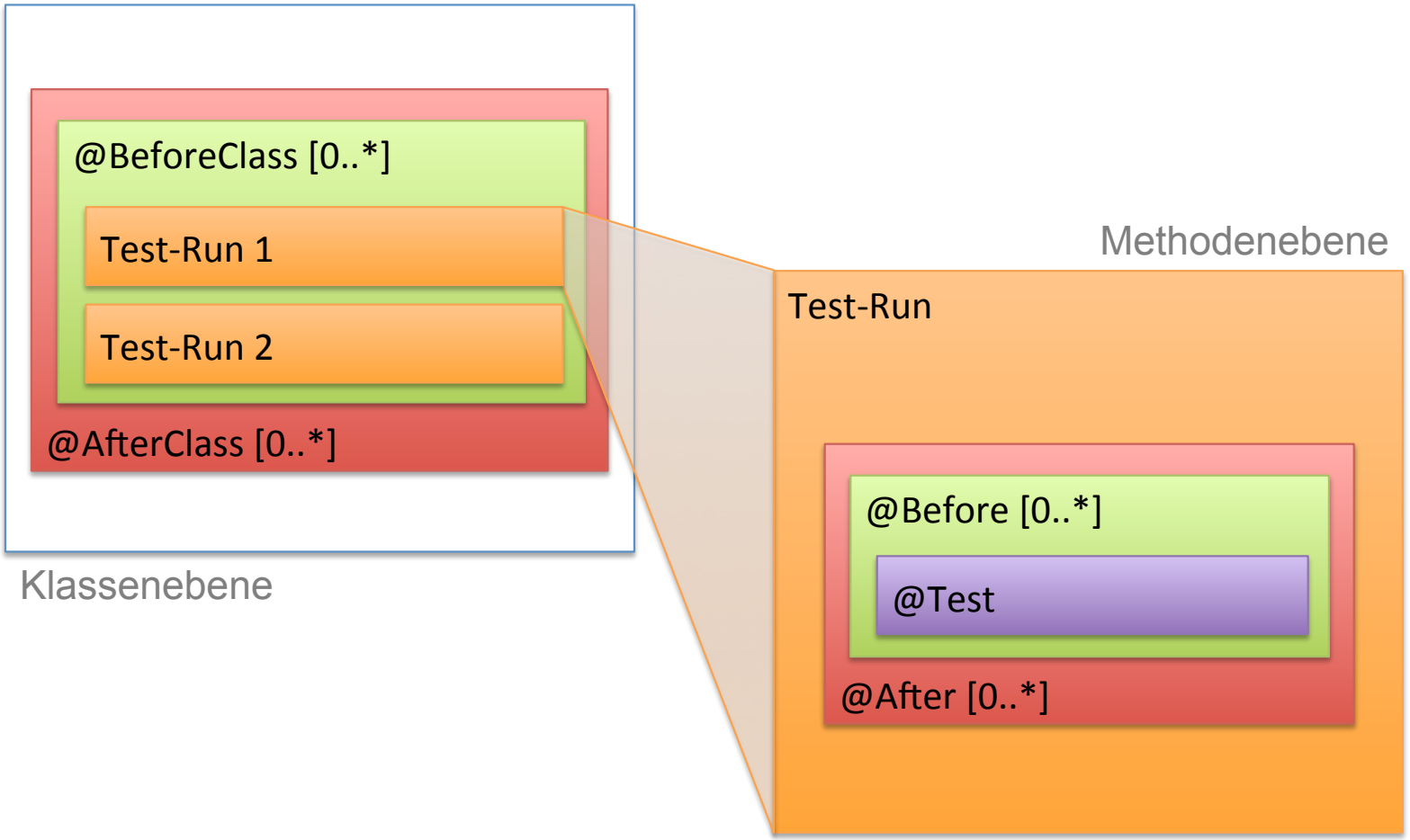
René Magritte, Golconda

Was ist eine Rule?

Erweiterungsmechanismus für Ablauf der Testmethoden

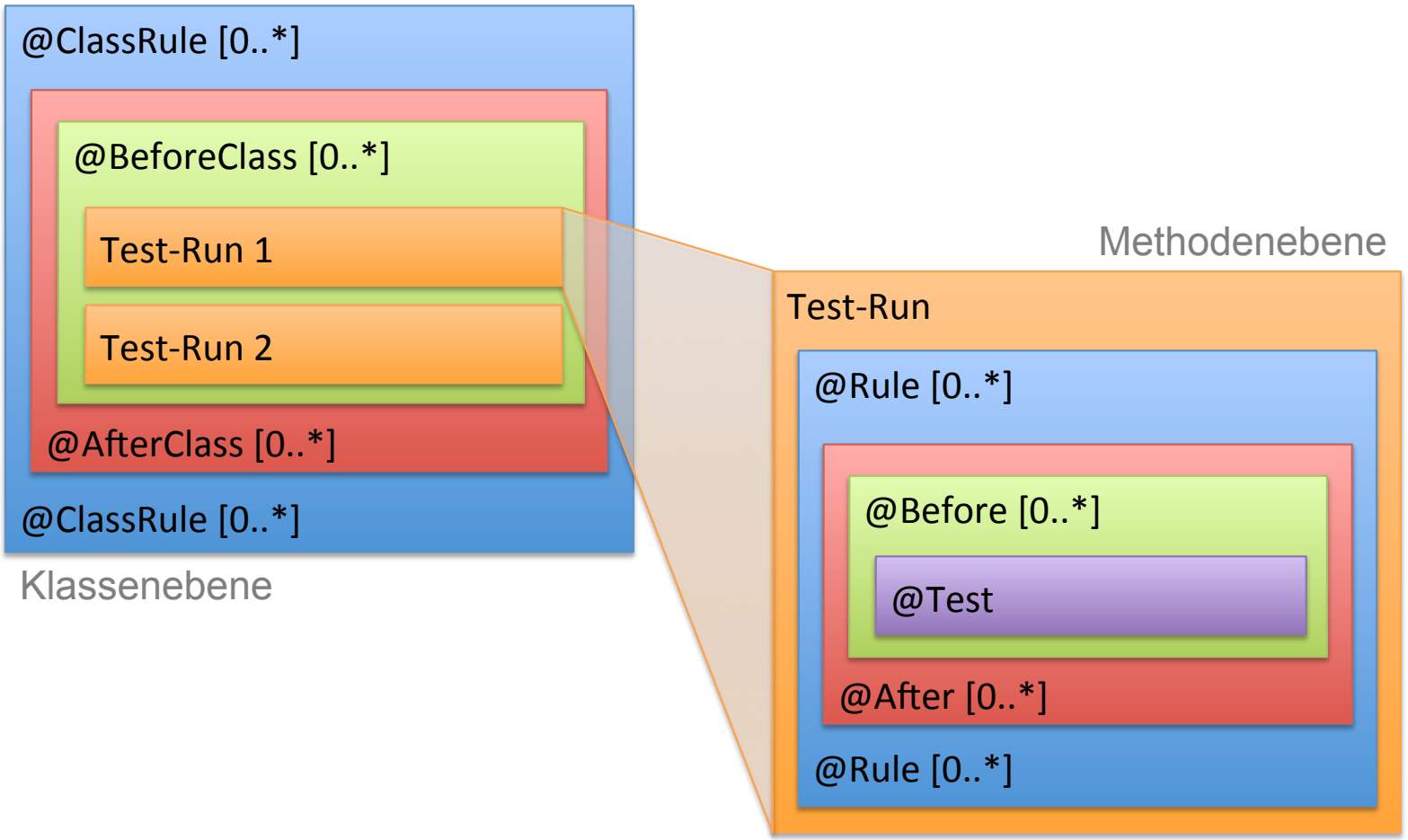
Ablaufreihenfolge von JUnit-Tests

Bisher



Ablaufreihenfolge von JUnit-Tests

Mit Rules



Beispiele für Rules

- Ausführung eigenen Codes vor bzw. nach jeder Testmethode
- Behandlung fehlgeschlagener Tests
- Überprüfung zusätzlicher Kriterien nach einem Tests
- ...

Beispiel: TemporaryFolder

Ohne Benutzung einer Rule

```
public class TemporaryFolderWithoutRule {
   private File folder;

   @Before public void createTemporaryFolder() throws Exception {
      folder = File.createTempFile("myFolder", "");
      folder.delete();
      folder.mkdir();
   }

   @Test public void test() throws Exception {
      File file = new File(folder, "test.txt");
      file.createNewFile();
      assertTrue(file.exists());
   }

   @After public void deleteTemporaryFolder() {
      recursivelyDelete(folder); // does not fit on current slide...
   }
```

Beispiel: TemporaryFolder

Unter Verwendung einer Rule

```
public class TemporaryFolderWithRule {

   @Rule public TemporaryFolder folder = new TemporaryFolder();

   @Test public void test() throws Exception {
      File file = folder.newFile("test.txt");
      assertTrue(file.exists());
   }

}
```

Beispiele für Rules

- Ausführung eigenen Codes vor bzw. nach jeder Testmethode
- Behandlung fehlgeschlagener Tests
- Überprüfung zusätzlicher Kriterien nach einem Tests
- ...

Beispiel: ExpectedException

Ohne Benutzung einer Rule

```
public class ExpectedExceptionWithoutRule {

   int[] threeNumbers = { 1, 2, 3 };

   @Test(expected = ArrayIndexOutOfBoundsException.class)
   public void exception() {
      threeNumbers[3] = 4;
   }

   @Test public void exceptionWithMessage() {
      try {
         threeNumbers[3] = 4;
         fail("ArrayIndexOutOfBoundsException expected");
      } catch (ArrayIndexOutOfBoundsException expected) {
         assertEquals("3", expected.getMessage());
      }
   }
}
```

Beispiel: ExpectedException

Unter Verwendung einer Rule

```
public class ExpectedExceptionWithRule {

   int[] threeNumbers = { 1, 2, 3 };

   @Rule public ExpectedException thrown = ExpectedException.none();

   @Test public void exception() {
      thrown.expect(ArrayIndexOutOfBoundsException.class);
      threeNumbers[3] = 4;
   }

   @Test public void exceptionWithMessage() {
      thrown.expect(ArrayIndexOutOfBoundsException.class);
      thrown.expectMessage("3");
      threeNumbers[3] = 4;
   }
}
```

Weitere vordefinierte Rules

ErrorCollector

Sammelt fehlgeschlagene Assertions innerhalb einer Testmethode und gibt am Ende eine Liste der Fehlschläge aus.

TestName

Merkt sich Namen der aktuell ausgeführten Testmethode und stellt ihn auf Anfrage zur Verfügung.

Timeout

Wendet gleichen Timeout auf alle Testmethoden einer Klasse an.

Rules auf Klassenebene

- Alle Rules, die `TestRule` implementieren, können auch auf Klassenebene verwendet werden.
- Die Regel wird dann einmal pro Testklasse ausgeführt.

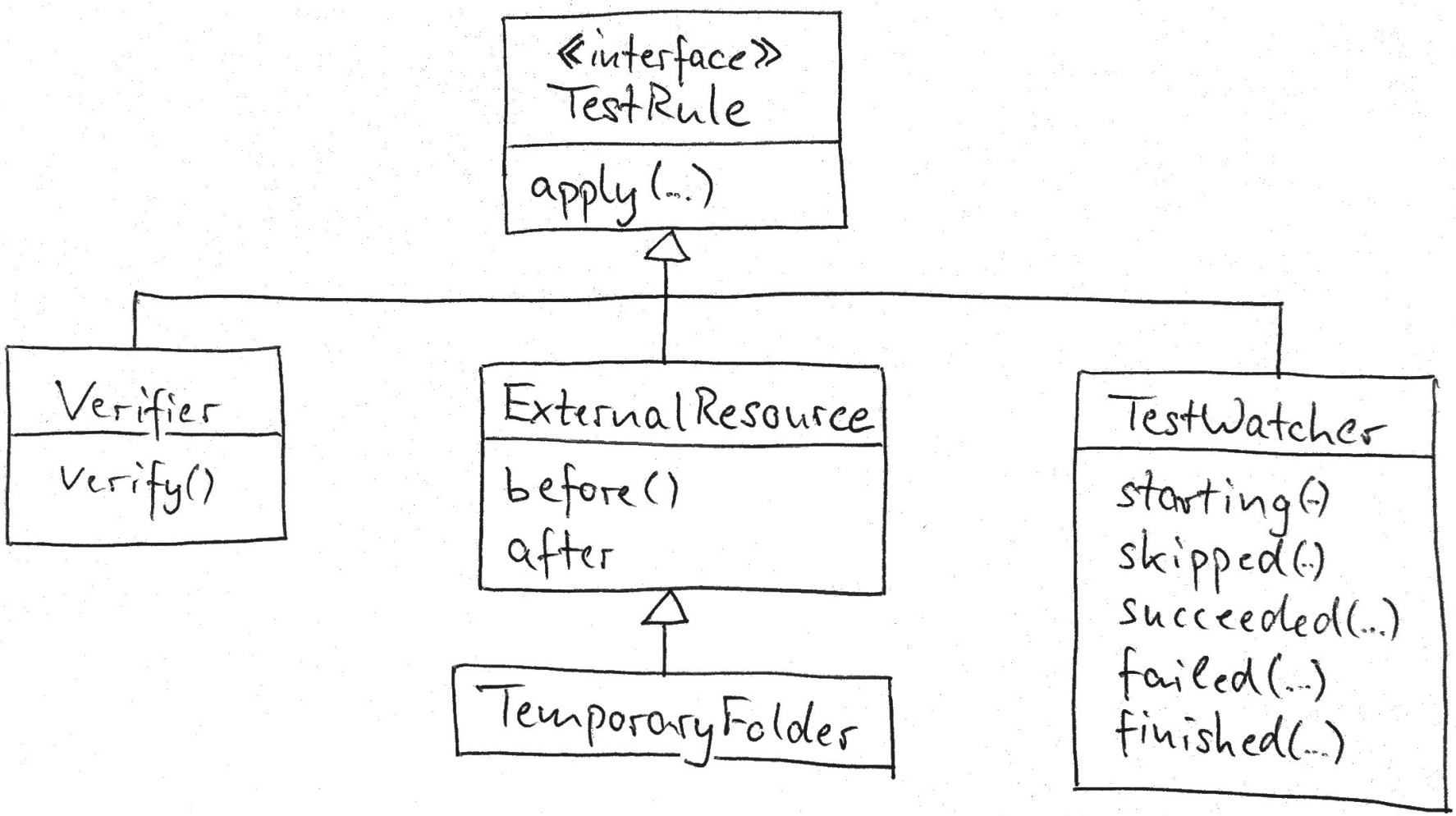
Aus

```
@Rule public TemporaryFolder folder = new TemporaryFolder();
```

wird

```
@ClassRule public static TemporaryFolder folder = new TemporaryFolder();
```

Schreib deine eigenen Regeln!



Eine eigene Regel

Implementierung

```
public class SystemProperty extends ExternalResource {
   private final String key, value;
   private String oldValue;
   public SystemProperty(String key, String value) {
      this.key = key;
      this.value = value;
   }
   @Override protected void before() {
      oldValue = System.getProperty(key);
      System.setProperty(key, value);
   }
   @Override protected void after() {
      if (oldValue == null) {
         System.getProperties().remove(key);
      } else {
         System.setProperty(key, oldValue);
      }
   }
}
```

Eine eigene Regel

Benutzung

```
public class SomeTestUsingSystemProperty {

   private static final String VALUE = "someValue";
   private static final String KEY = "someKey";

   @Rule public SystemProperty systemProperty = new SystemProperty(KEY, VALUE);

   @Test public void test() {
      assertThat(System.getProperty(KEY), is(VALUE));
   }
}
```

Reihenfolge von Rules

Die Ausführungsreihenfolge von Rules ist absichtlich undefiniert.

Falls notwendig, lässt sich per `RuleChain` eine Reihenfolge festlegen:

```
@Rule public RuleChain chain = RuleChain
      .outerRule(new LoggingRule("outer rule"))
      .around(new LoggingRule("middle rule"))
      .around(new LoggingRule("inner rule"));
```

Vorteile von Regeln

Wiederverwendbarkeit

Ermöglichen häufig benötigten Code auszulagern.

Kombinierbarkeit

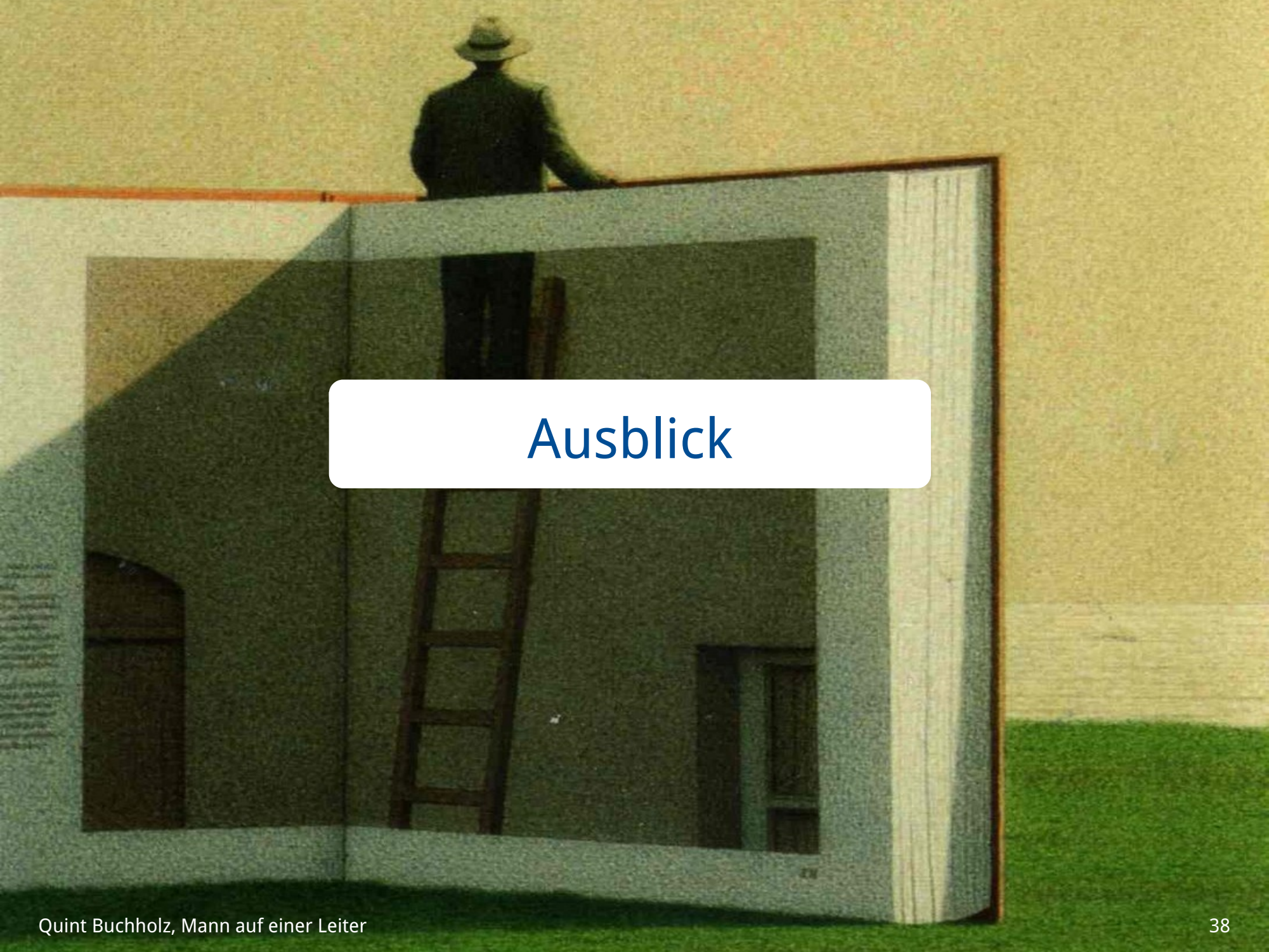
Beliebig viele Regeln in einem Test verwendbar

Delegation statt Vererbung

Helfen Testklassenhierarchien zu vermeiden!

Erweiterbarkeit

Eigene Regeln schreiben ist einfach.



Ausblick

Quint Buchholz, Mann auf einer Leiter

Was jetzt?

- Aktualisierung auf neue Version ist einfach
- Alte Tests funktionieren weiterhin
- Neue Tests profitieren von neuen Features
- Alte Tests können nach und nach vereinfacht werden
- Ausprobieren!

Ausprobieren!

http://www.junit.org/

http://hamcrest.org/JavaHamcrest/

Mail marc@andrena.de

Twitter @marcphilipp

Blog http://www.marcphilipp.de/