Using Hamcrest for testing - Tutorial

Lars Vogel, (c) 2014, 2016 vogella GmbHVersion 1.8,29.05.2016

Table of Contents

This tutorial explains testing with the Hamcrest matcher framework.

1. Purpose of the Hamcrest matcher framework

Hamcrest is a framework for software tests. Hamcrest allows checking for conditions in your code via existing matchers classes. It also allows you to define your custom matcher implementations.

To use Hamcrest matchers in JUnit you use the `assertThat` statement followed by one or several matchers.

Hamcrest is typically viewed as a third generation matcher framework. The first generation used `assert(logical statement)` but such tests were not easily readable. The second generation introduced special methods for assertions, e.g., `assertEquals()`. This approach leads to lots of assert methods. Hamcrest uses `assertThat` method with a matcher expression to determine if the test was succesful. See Wiki on Hamcrest for more details.

Hamcrest has the target to make tests as readable as possible. For example, the `is` method is a thin wrapper for `equalTo(value)`.

```
import static org.hamcrest.MatcherAssert.assertThat;
import static org.hamcrest.Matchers.is;
import static org.hamcrest.Matchers.equalTo;

boolean a;
boolean b;
```

```
// all statements test the same
assertThat(a, equalTo(b));
assertThat(a, is(equalTo(b)));
assertThat(a, is(b));
```

The following snippets compare pure JUnit 4 assert statements with Hamcrest matchers.

```
// JUnit 4 for equals check
assertEquals(expected, actual);
// Hamcrest for equals check
assertThat(actual, is(equalTo(expected)));

// JUnit 4 for not equals check
assertNotEquals(expected, actual)
// Hamcrest for not equals check
assertThat(actual, is(not(equalTo(expected))));
```

It is also possible to chain matchers, via the `anyOf` of `allOf` method.

```
assertThat("test", anyOf(is("testing"), containsString("est")));
```

In general the Hamcrest error messages are also much easier to read.

```
assertTrue(result instanceof String);
// error message:
java.lang.AssertionError
    at org.junit.Assert.fail(Assert.java:86)
    at org.junit.Assert.assertTrue(Assert.java:41)
    at org.junit.Assert.assertTrue(Assert.java:52)
// ...


assertEquals(String.class, result.getClass());
// error message:
java.lang.NullPointerException
    at com.vogella.hamcrest.HamcrestTest.test(HamcrestTest.java:30)
// ....


assertThat(result, instanceOf(String.class));
// error message:
java.lang.AssertionError:
Expected: an instance of java.lang.String
     but: null
    at org.hamcrest.MatcherAssert.assertThat(MatcherAssert.java:20)
    at org.hamcrest.MatcherAssert.assertThat(MatcherAssert.java:8)
// ...
```

Using Hamcrest matchers also provides more type safety as these matchers use generics.

2. Making the Hamcrest library available

2.1. Defining a Hamcrest dependency for Gradle

To use Hamcrest matchers for a project based on the Gradle build system, add the following dependencies to it.

```
dependencies {
    // Unit testing dependencies
    testCompile 'junit:junit:4.12'
    // Set this dependency if you want to use Hamcrest matching
    testCompile 'org.hamcrest:hamcrest-library:1.3'
}
```

2.2. Defining a Hamcrest dependency for Maven

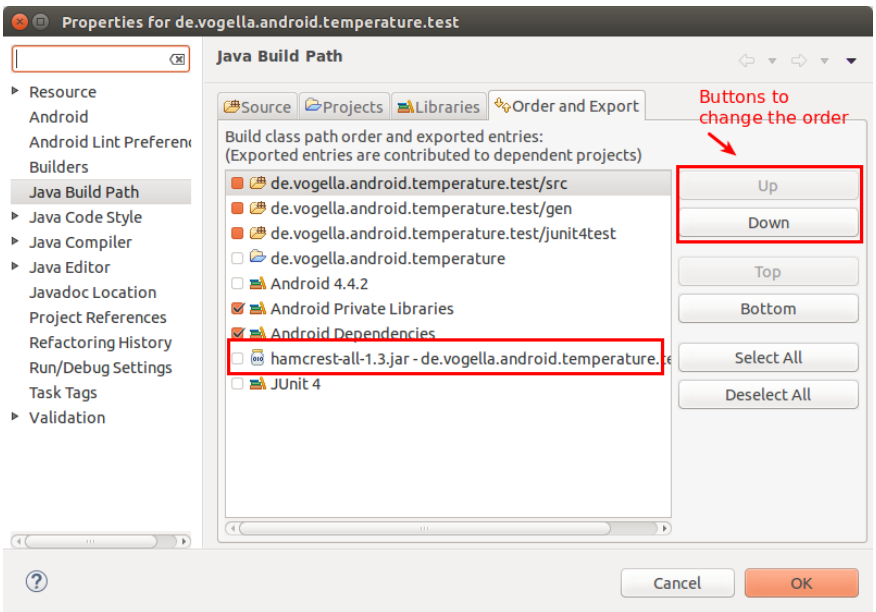
To use the library for a Maven based project, the following dependency to your pom file.

```
<dependency>
<groupId>org.hamcrest</groupId>
<artifactId>hamcrest-library</artifactId>
<version>1.3</version>
<scope>test</scope>
</dependency>
```

2.3. Adding Hamcrest directly to the classpath in Eclipse

The JUnit distribution included in Eclipse only contain the core Hamcrest matcher. To use all available matchers, download the latest hamcrest-all-*.jar from https://code.google.com/p/hamcrest/downloads/list and add it to your projects classpath.

> If you get the following exception "java.lang.SecurityException: class "org.hamcrest.Matchers"'s signer information does not match signer information of other classes in the same package", ensure that the hamcrest jar is before the Junit library in the build path. You an configure the order in the project properties in the Eclipse IDE under *Java Build Path* on the *Order and Export* tab.



3. Using Hamcrest

3.1. Example

The usage of Hamcrest matchers is demonstrates by the following code snippet.

```
assertThat(Long.valueOf(1), instanceOf(Integer.class));
// shortcut for instanceOf
assertThat(Long.valueOf(1), isA(Integer.class));
```

3.2. Static import

To make all matchers available in your file add an static import. This also makes it easier to find matchers through code completion.

```
import static org.hamcrest.MatcherAssert.assertThat;
import static org.hamcrest.Matchers.*;
```

3.3. Hamcrest matchers for lists

The usage of the Hamcrest matchers for lists are demonstrated by the following example.

```
import org.junit.Test;

import java.util.Arrays;
import java.util.List;

import static org.hamcrest.MatcherAssert.assertThat;
import static org.hamcrest.Matchers.contains;
import static org.hamcrest.Matchers.containsInAnyOrder;
import static org.hamcrest.Matchers.greaterThan;
import static org.hamcrest.collection.IsCollectionWithSize.hasSize;
import static org.hamcrest.core.Every.everyItem;

public class HamcrestListMatcherExamples {
    @Test
    public void listShouldInitiallyBeEmpty() {
        List<Integer> list = Arrays.asList(5, 2, 4);

        assertThat(list, hasSize(3));

        // ensure the order is correct
        assertThat(list, contains(5, 2, 4));

        assertThat(list, containsInAnyOrder(2, 4, 5));

        assertThat(list, everyItem(greaterThan(1)));

    }
}
```

```
// Check that a list of objects has a property race and
// that the value is not ORC
assertThat(fellowship, everyItem(hasProperty("race", is(not((ORC))))));
```

3.4. Overview of Hamcrest mather

The following are the most important Hamcrest matchers:

- `allOf` - matches if all matchers match (short circuits)
- `anyOf` - matches if any matchers match (short circuits)
- `not` - matches if the wrapped matcher doesn't match and vice
- `equalTo` - test object equality using the equals method
- `is` - decorator for equalTo to improve readability
- `hasToString` - test Object.toString
- `instanceOf`, `isCompatibleType` - test type
- `notNullValue`, `nullValue` - test for null
- `sameInstance` - test object identity
- `hasEntry`, `hasKey`, `hasValue` - test a map contains an entry, key or value
- `hasItem`, `hasItems` - test a collection contains elements
- `hasItemInArray` - test an array contains an element
- `closeTo` - test floating point values are close to a given value
- `greaterThan`, `greaterThanOrEqualTo`, `lessThan`, `lessThanOrEqualTo`
- `equalToIgnoringCase` - test string equality ignoring case
- `equalToIgnoringWhiteSpace` - test string equality ignoring differences in runs of whitespace
- `containsString`, `endsWith`, `startsWith` - test string matching

To see all matchers, use API reference.

4. Exercise - Using Hamcrests built-in matchers

4.1. Target

The target of this exercise is to make yourself familiar with Hamcrest matchers.

4.2. Using Hamcrest collection matchers for lists

4.2.1. Target

Assume the following code:

```
List<Integer> list = Arrays.asList(5, 2, 4);
```

Ensure via tests with Hamcrest matchers that `list`:

- has a size of 3
- contains the elements 2, 4, 5 in any order
- every item is greater than 1

4.3. Solutions for collection matchers

```
@Test
public void hasSizeOf3() {
    List<Integer> list = Arrays.asList(5, 2, 4);

    assertThat(list, hasSize(3));
}
```

```
@Test
public void containsNumbersInAnyOrder() {
    List<Integer> list = Arrays.asList(5, 2, 4);

    assertThat(list, containsInAnyOrder(2, 4, 5));
}
```

```
@Test
public void everyItemGreaterThan1() {
    List<Integer> list = Arrays.asList(5, 2, 4);

    assertThat(list, everyItem(greaterThan(1)));
}
```

4.4. Using Hamcrest collection matchers for arrays

4.4.1. Target

Assume the following code:

```
Integer[] ints = new Integer[] {7, 5, 12, 16};
```

1. Ensure via tests with Hamcrest matchers that the `ints` array
    - has a size of 4
    - contains 7, 5, 12, 16 in the given order

4.4.2. Solution - Array Exercises

```
@Test
public void arrayHasSizeOf4() {
    Integer[] ints = new Integer[] { 7, 5, 12, 16 };

    assertThat(ints, arrayWithSize(4));
}
```

```
@Test
public void arrayContainsNumbersInGivenOrder() {
    Integer[] ints = new Integer[] { 7, 5, 12, 16 };

    assertThat(ints, arrayContaining(7, 5, 12, 16));
}
```

4.5. Using Hamcrest beans matchers

4.5.1. Target

Assume the following code:

```
public class Todo {

    private final long id;
    private String summary;
    private String description;
    private int year;

    public Todo(long id, String summary, String description) {
        this.id = id;
        this.summary = summary;
        this.description = description;
    }

    // getters/setters
}
```

Write tests that ensure that:

- Todo has a property called "summary"
- If Todo is constructed with the summary "Learn Hamcrest" that the summary property was initialized with this value
- Two objects created with the same values, have the same property values

4.5.2. Solution

```
@Test
public void objectHasSummaryProperty () {

    Todo todo = new Todo(1, "Learn Hamcrest", "Important");

    assertThat(todo, hasProperty("summary"));
}
```

```
@Test
public void objectHasCorrectSummaryValue () {

    Todo todo = new Todo(1, "Learn Hamcrest", "Important");

    assertThat(todo, hasProperty("summary", equalTo("Learn Hamcrest")));
}
```

```
@Test
public void objectHasSameProperties () {

    Todo todo1 = new Todo(1, "Learn Hamcrest", "Important");
    Todo todo2 = new Todo(1, "Learn Hamcrest", "Important");

    assertThat(todo1, samePropertyValuesAs(todo2));
}
```

4.6. Using Hamcrest String matchers

4.6.1. Target

Write tests that ensure that:

- "" is an empty string
- a given string is either empty or null

4.6.2. Solution

```
import org.junit.Test;

import static org.junit.Assert.*;
import static org.hamcrest.text.IsEmptyString.isEmptyString;
import static org.hamcrest.text.IsEmptyString.isEmptyOrNullString;

public class StringMatcherTest {
    @Test
    public void isStringEmpty() {
        String stringToTest = "";
        assertThat(stringToTest, isEmptyString());
    }

    @Test
    public void isStringEmptyOfNull() {
        String stringToTest = "";
        assertThat(stringToTest, isEmptyOrNullString());
    }

}
```

5. Exercise - Writing a custom Hamcrest matcher using FeatureMatcher

5.1. Target

The target of this exercise is to write a custom matcher with Hamcrest.

5.2. Create Hamcrest Matchers

Define a custom matcher for Hamcrest which provides the `length` matcher for a String. We want to use the class `FeatureMatcher`. With FeatureMatcher we can wrap an existing Matcher, decide which field of the given Object under test to match and provide a nice error message. The constructor of FeatureMatcher takes the following arguments in this order:

- The matcher we want to wrap
- a description of the feature that we tested
- a description of the possible mismatch

The only method we have to overwrite is `featureValueOf(T actual)` which returns the value which will get passed into the wrapped `matches()` / `matchesSafely()` method.

```
public static Matcher<String> length(Matcher<? super Integer> matcher) {
    return new FeatureMatcher<String, Integer>(matcher, "a String of length
that", "length") {
        @Override
        protected Integer featureValueOf(String actual) {
            return actual.length();
        }
    };
}
```

5.3. Validate

Use your custom matcher to check that "Gandalf" has a lenght of 8.

```
@Test
public void fellowShipOfTheRingShouldContainer7() {
    assertThat("Gandalf", length(is(8)));
}
public static  Matcher<String> length(Matcher<? super Integer> matcher) {
    return new FeatureMatcher<String, Integer>(matcher, "a String of length
that", "length") {
        @Override
        protected Integer featureValueOf(String actual) {
          return actual.length();
        }
    };
}
```

6. Exercise: Writing your custom Hamcrest matcher using TypeSafeMatcher

It is possible to write your custom Hamcrest matcher by extending TypeSafeMatcher. In contrast to BaseMatcher the TypeSafeMatcher class automatically checks for `null` values, checks the type and casts appropriately before delegating to `matchesSafely()`. It provides type safety by default. The following is an example for defining a matcher which allows testing if a String matches a regular expression.

```
import org.hamcrest.Description;
import org.hamcrest.TypeSafeMatcher;

public class RegexMatcher extends TypeSafeMatcher<String> {

    private final String regex;
```

```
    public RegexMatcher(final String regex) {
        this.regex = regex;
    }

    @Override
    public void describeTo(final Description description) {
        description.appendText("matches regular expression=`" + regex +
"`");
    }

    @Override
    public boolean matchesSafely(final String string) {
        return string.matches(regex);
    }


     // matcher method you can call on this matcher class
    public static RegexMatcher matchesRegex(final String regex) {
        return new RegexMatcher(regex);
    }
}
```

The following snippet gives an example how to use it.

```
package com.vogella.android.testing.applicationtest;


import org.junit.Test;

import static org.hamcrest.MatcherAssert.assertThat;

public class TestCustomMatcher {

    @Test
    public void testRegularExpressionMatcher() throws Exception {
        String s ="aaabbbaaaa";
        assertThat(s, RegexMatcher.matchesRegex("a*b*a*"));
    }

}
```