ABOUT HUGH HAMILL

Hugh is a Senior Software Engineer and Certified Scrum Master based in Galway, Ireland. He achieved his B.Sc. in Applied Computing from Waterford Institute of Technology in 2002 and has been working in industry since then. He has worked for a several large blue chip software companies listed on both the NASDAQ and NYSE.

Hamcrest matchers tutorial

Posted by: Hugh Hamill in Core Java November 15th, 2015

This article is part of our Academy Course titled Testing with Mockito.

In this course, you will dive into the magic of Mockito. You will learn about Mocks, Spies and Partial Mocks, and their corresponding Stubbing behaviour. You will also see the process of Verification with Test Doubles and Object Matchers. Finally, Test Driven Development (TDD) with Mockito is discussed in order to see how this library fits in the concept of TDD. Check it out here!

In this tutorial we will look at the Hamcrest Matcher library and how to integrate it with JUnit and Mockito.

Table Of Contents

1. What is Hamcrest?
2. Including Hamcrest
3. Meet the Matchers
3.1. Simple Matchers
3.2. Simple Matchers Combining Other Matchers
4. Conclusion

1. What is Hamcrest?

Hamcrest is a framework for creating matcher objects. These matcher objects are predicates and are used to write rules which can be satisfied under certain conditions. They are most often used in automated testing, though the can be used in other scenarios such as data write and to understand. When used in conjunction with JUnit and Mockito it allows us to write clear, concise tests which satisfy the property of good unit testing which is to 'test one thing'.

Suppose we have a String called and we want to test that it is equal to another, expected, string we can use JUnit asserts to test this:

```
assertEquals(expected, actual);
```

In Hamcrest we make use of the JUnit

```
assertThat(valueUnderTest, matcher)
```

method. This method always forms the basis of the Hamcrest assert; we are asserting that the value under test satisfies the matcher predicate. To rewrite the above test in hamcrest at it's most basic level we could write:

```
assertThat(actual, equalTo(expected));
```

Note the assertThat convention is to have the actual value under test as the first parameter, this is opposite to the assertEquals convention. This is an improvement on readability, but Hamcrest additionally gives us some nice syntactic sugar in the form of the

```
is()
```

matcher. This matcher does nothing itself, it simply relays the result of its input matcher allowing your assertion code to read just like English. Let's rewrite the above using

```
is()
```

:

```
assertThat(actual, is(equalTo(expected)));
```

Very nice, very readable!

Hamcrest generates detailed output when its matchers fail, specifying the expected values and the actual values, to assist you in figuring out why the test should fail. Look at the following test case:

```
@Test
    public void test_failed() throws Exception {
        // Given
        Integer number = 7;

        // Then
        assertThat(number, greaterThan(10));
    }
```

Obviously this test will fail, but Hamcrest will give detailed information about the failure:

```
java.lang.AssertionError:
Expected: a value greater than <10>
     but: <7> was less than <10>
```

Throughout this tutorial we will stick to the convention of having just one assertion as part of our unit tests. This may seem repetitive, particularly where the setup part of the test is the same across a number of tests, however this is good practice in Unit Testing. It allows us to create tests which are targeted – tests will fail only if their single assertion fails, every other assertion will continue to execute. It allows us to create tests which are readable – we can see at a glance the purpose of the test. It allows us to create test which document the code – we can use test names which convey the granular purpose of the test, and therefore the behaviour of the code under test (think

```
customer_should_have_balance_updated_by_input_order_amount()
```

rather than

```
verifyOrderMethod()
```

). It allows us to create tests which are not brittle – if a test does too much it may break if unrelated functionality is changed, forcing us to rewrite the test just to get it working again without changing the actual code under test.

If we adopt the 'test one thing' habit we will be writing much better unit tests into the future!

2. Including Hamcrest

If you are using Maven you can add Hamcrest to your project with the following dependency to your pom.xml

```
<dependency>
    <groupId>org.hamcrest</groupId>
    <artifactId>hamcrest-all</artifactId>
    <version>1.3</version>
  </dependency>
```

If you are using Gradle add the following

```
dependencies {
    testCompile "org.hamcrest:hamcrest-all:1.3"
```

a location on your hard drive.

Right click on your eclipse project and select 'Properties' and then select 'Java Build Path' in the left pane and 'Libraries' on the right.

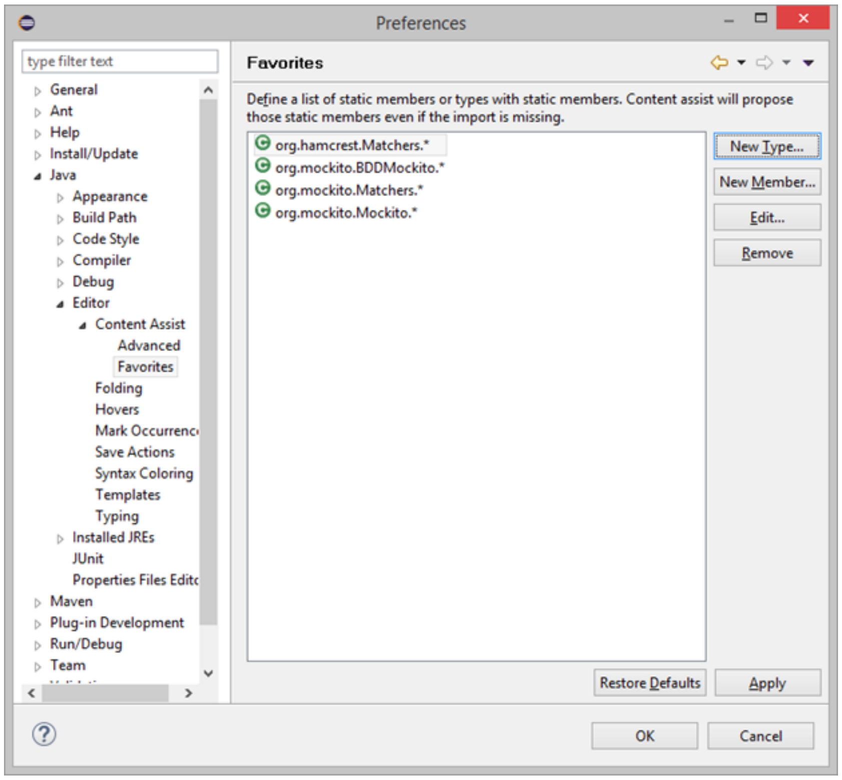
On the 'Libraries' tab click the 'Add External Jars' button and navigate to the hamcrest-all jar you previously downloaded. Select the jar and it is now added to your project and available to use.

Note that JUnit gets bundled with a stripped down version of Hamcrest (Hamcrest Core) so your compiler will pick up that version if JUnit appears before Hamcrest on the classpath. To counteract this please ensure Hamcrest appears before JUnit on the classpath. You can achieve this in Maven by listing the hamcrest-all dependency before all other dependencies.

As with the Mockito static methods we can add the Hamcrest library to Eclipse content assist by launching Window -> Preferences and go to Java/Editor/Content Assist/Favorites in the left nav. After that add the following as "New Type…" as per Figure 1

- org.hamcrest.Matchers

launch Window -> Preferences and go to Java/Editor/Content Assist/Favorites in the left nav. After that add the following as "New Type…" as per Figure 1



Define a list of static members or types with static members. Content assist will propose those static members even if the import is missing.

- org.hamcrest.Matchers.*
- org.mockito.BDDMockito.*
- org.mockito.Matchers.*
- org.mockito.Mockito.*

Figure 1 – Content Assist Favorites

3. Meet the Matchers

Hamcrest provides a library of static factory methods for creating Matchers in the class org.hamcrest.Matchers so you can bring in all the matchers with a static import

```java
import static org.hamcrest.Matchers.*
```

However you run the risk of a naming clash if you do this because both Hamcrest and Mockito provide a static

```
any()
```

method so it is recommended to import each static method you use individually.
We will now look at all of the Matchers available to us in the Hamcrest Matchers library. They will be broken into two broad categories; Matchers which work to test values (Simple), and Matchers which work to combine other Matchers (Aggregate).

3.1. Simple Matchers

The following matchers primarily work to test input values.

3.1.1. any()

Matches any variable of the given type.

```

        // Then
        assertThat(myString, is(any(String.class)));
    }
```

3.1.2. anything()

Matches anything.

```
@Test
    public void test_anything() throws Exception {
        // Given
        String myString = "hello";
        Integer four = 4;

        // Then
        assertThat(myString, is(anything()));
        assertThat(four, is(anything()));
    }
```

3.1.3. arrayContaining()

Various matchers for Arrays, length of the array must match the number of matchers, and their order is important.

Does the array contain all given items in the order in which they are input to the matcher?

```
@Test
    public void test_arrayContaining_items() throws Exception {
        // Given
        String[] strings = {"why", "hello", "there"};

        // Then
        assertThat(strings, is(arrayContaining("why", "hello", "there")));
    }
```

Does the array contain items which match the input list of matchers, in order?

```
@Test
    public void test_arrayContaining_list_of_matchers() throws Exception {
        // Given
        String[] strings = {"why", "hello", "there"};

        // Then
        java.util.List<org.hamcrest.Matcher<? super String>> itemMatchers = new ArrayList<>();
        itemMatchers.add(equalTo("why"));
        itemMatchers.add(equalTo("hello"));
        itemMatchers.add(endsWith("here"));
        assertThat(strings, is(arrayContaining(itemMatchers)));
    }
```

Does the array contain items which match the input vararg matchers, in order?

```
@Test
    public void test_arrayContaining_matchers() throws Exception {
        // Given
        String[] strings = {"why", "hello", "there"};

        // Then
        assertThat(strings, is(arrayContaining(startsWith("wh"), equalTo("hello"), endsWith("here"))));
    }
```

3.1.4. arrayContainingInAnyOrder()

Various matchers for Arrays, length of the array must match the number of matchers, but their order is not important.

Does the array contain all the given items?

```
@Test
    public void test_arrayContainingInAnyOrder_items() throws Exception {
        // Given
        String[] strings = { "why", "hello", "there" };

        // Then
        assertThat(strings, is(arrayContainingInAnyOrder("hello", "there", "why")));
    }
```

Does the array contain items which match the input collection of Matchers?

```
@Test
    public void test_arrayContainingInAnyOrder_collection_of_matchers() throws Exception {
        // Given
        String[] strings = { "why", "hello", "there" };

        // Then
        Set<org.hamcrest.Matcher<? super String>> itemMatchers = new HashSet<>();
        itemMatchers.add(equalTo("hello"));
        itemMatchers.add(equalTo("why"));
        itemMatchers.add(endsWith("here"));
        assertThat(strings, is(arrayContainingInAnyOrder(itemMatchers)));
    }
    public void test_arrayContainingInAnyOrder_matchers() throws Exception {
        // Given
        String[] strings = { "why", "hello", "there" };

        // Then
        assertThat(strings, is(arrayContainingInAnyOrder(endsWith("lo"), startsWith("the"),
equalTo("why"))));
    }
```

3.1.5. arrayWithSize()

Various matchers to check if an array is of a certain length.

Does the input array have exactly the specified length?

```
@Test
    public void test_arrayWithSize_exact() throws Exception {
        // Given
        String[] strings = { "why", "hello", "there" };

        // Then
        assertThat(strings, is(arrayWithSize(3)));
    }
```

Does the input array have a length which matches the specified matcher?

```
@Test
    public void test_arrayWithSize_matcher() throws Exception {
        // Given
        String[] strings = { "why", "hello", "there" };

        // Then
        assertThat(strings, is(arrayWithSize(greaterThan(2))));
    }
```

3.1.6. closeTo()

Matcher which can be used with either Double or BigDecimal to check if a value is within a specified error margin of an expected value.

Double

```
@Test
    public void test_closeTo_double() throws Exception {
        // Given
        Double testValue = 6.3;

        // Then
        assertThat(testValue, is(closeTo(6, 0.5)));
    }
```

BigDecimal

```
@Test
    public void test_closeTo_bigDecimal() throws Exception {
        // Given
        BigDecimal testValue = new BigDecimal(324.0);

        // Then
        assertThat(testValue, is(closeTo(new BigDecimal(350), new BigDecimal(50))));
    }
```

3.1.7. comparesEqualTo()

Creates a Comparable matcher which attempts to match the input matcher value using the

```
compareTo()
```

method of the input value. The matcher will match if the

```
compareTo()
```

method returns 0 for the input matcher value, otherwise it would not match.

```
@Test
    public void test_comparesEqualTo() throws Exception {
        // Given
        String testValue = "value";

        // Then
        assertThat(testValue, comparesEqualTo("value"));
    }
```

3.1.8. contains()

Various matchers which can be used to check if an input Iterable contains values. The order of the values is important and the number of items in the Iterable must match the number of values being tested.

Does the input list contain all of the values, in order?

```
        // Given
        List<String> strings = Arrays.asList("why", "hello", "there");

        // Then
        assertThat(strings, contains("why", "hello", "there"));
    }
```

Does the input list contain items which match all of the matchers in the input matchers list, in order?

```
@Test
    public void test_contains_list_of_matchers() throws Exception {
        // Given
        List<String> strings = Arrays.asList("why", "hello", "there");

        // Then
        List<org.hamcrest.Matcher<? super String>> matchers = new ArrayList<>();
        matchers.add(startsWith("wh"));
        matchers.add(endsWith("lo"));
        matchers.add(equalTo("there"));
        assertThat(strings, contains(matchers));
    }
```

Does the input list contain only one item which matches the input matcher?

```
@Test
    public void test_contains_single_matcher() throws Exception {
        // Given
        List<String> strings = Arrays.asList("hello");

        // Then
        assertThat(strings, contains(startsWith("he")));
    }
```

Does the input list contain items which match all of the matchers in the input vararg matchers, in order?

```
@Test
    public void test_contains_matchers() throws Exception {
        // Given
        List<String> strings = Arrays.asList("why", "hello", "there");

        // Then
        assertThat(strings, contains(startsWith("why"), endsWith("llo"), equalTo("there")));
    }
```

3.1.9. containsInAnyOrder()

Various matchers which can be used to check if an input Iterable contains values. The order of the values is not important but the number of items in the Iterable must match the number of values being tested.

Does the input list contain all of the values, in any order?

```
@Test
    public void test_containsInAnyOrder_items() throws Exception {
        // Given
        List<String> strings = Arrays.asList("why", "hello", "there");

        // Then
        assertThat(strings, containsInAnyOrder("hello", "there", "why"));
    }
```

Does the input list contain items which match all of the matchers in the input matchers list, in any order?

```
@Test
    public void test_containsInAnyOrder_list_of_matchers() throws Exception {
        // Given
        List<String> strings = Arrays.asList("why", "hello", "there");

        // Then
        List<org.hamcrest.Matcher<? super String>> matchers = new ArrayList<>();
        matchers.add(equalTo("there"));
        matchers.add(startsWith("wh"));
        matchers.add(endsWith("lo"));
        assertThat(strings, containsInAnyOrder(matchers));
    }
```

Does the input list contain items which match all of the matchers in the input vararg matchers, in any order?

```
@Test
    public void test_containsInAnyOrder_matchers() throws Exception {
        // Given
        List<String> strings = Arrays.asList("why", "hello", "there");

        // Then
        assertThat(strings, containsInAnyOrder(endsWith("llo"), equalTo("there"), startsWith("why")));
    }
```

3.1.10. containsString()

Matcher which matches if the String under test contains the given substring.

```

        // Then
        assertThat(testValue, containsString("alu"));
    }
```

3.1.11. empty()

Matcher which matches if an input Collections

```
isEmpty()
```

method returns true.

```
@Test
    public void test_empty() throws Exception {
        // Given
        Set<String> testCollection = new HashSet<>();

        // Then
        assertThat(testCollection, is(empty()));
    }
```

3.1.12. emptyArray()

Matcher which matches if the input array has a length of 0.

```
@Test
    public void test_emptyArray() throws Exception {
        // Given
        String[] testArray = new String[0];

        // Then
        assertThat(testArray, is(emptyArray()));
    }
```

3.1.13. emptyCollectionOf()

Typesafe matcher which matches if the input collection is of the given type and is empty.

```
@Test
    public void test_emptyCollectionOf() throws Exception {
        // Given
        Set<String> testCollection = new HashSet<>();

        // Then
        assertThat(testCollection, is(emptyCollectionOf(String.class)));
    }
```

3.1.14. emptyIterable()

Matcher which matches if the input Iterable has no values.

```
@Test
    public void test_emptyIterable() throws Exception {
        // Given
        Set<String> testCollection = new HashSet<>();

        // Then
        assertThat(testCollection, is(emptyIterable()));
    }
```

3.1.15. emptyIterableOf()

Typesafe Matcher which matches if the input Iterable has no values and is of the given type.

```
@Test
    public void test_emptyIterableOf() throws Exception {
        // Given
        Set<String> testCollection = new HashSet<>();

        // Then
        assertThat(testCollection, is(emptyIterableOf(String.class)));
    }
```

3.1.16. endsWith()

Matcher which matches if the input String ends with the given substring.

```
@Test
    public void test_endsWith() throws Exception {
        // Given
        String testValue = "value";

        // Then
        assertThat(testValue, endsWith("lue"));
    }
```

length of the Array and ensure that all the values in the input test array are logically equal to the values of the specified array.

Single value.

```
@Test
    public void test_equalTo_value() throws Exception {
        // Given
        String testValue = "value";

        // Then
        assertThat(testValue, equalTo("value"));
    }
```

Array.

```
@Test
    public void test_equalTo_array() throws Exception {
        // Given
        String[] testValues = { "why", "hello", "there" };

        // Then
        String[] specifiedValues = { "why", "hello", "there" };
        assertThat(testValues, equalTo(specifiedValues));
    }
```

3.1.18. equalToIgnoringCase()

Matcher which matches if the input String value is equal to the specified String while ignoring case.

```
@Test
    public void test_equalToIgnoringCase() throws Exception {
        // Given
        String testValue = "value";

        // Then
        assertThat(testValue, equalToIgnoringCase("VaLuE"));
    }
```

3.1.19. equalToIgnoringWhiteSpace()

Matcher which matches if the input String value is equal to the specified String while ignoring superfluous white space. All leading and trailing whitespace are ignored, and all remaining whitespace is collapsed to a single space.

```
@Test
    public void test_equalToIgnoringWhiteSpace() throws Exception {
        // Given
        String testValue = "this    is   my    value    ";

        // Then
        assertThat(testValue, equalToIgnoringWhiteSpace("this is my value"));
    }
```

3.1.20. eventFrom()

Matcher which matches if an input

```
EventObject
```

is from the given Source. Can also accept an

```
EventObeject
```

of a specified subtype.

```
@Test
    public void test_eventFrom() throws Exception {
        // Given
        Object source = new Object();
        EventObject testEvent = new EventObject(source);

        // Then
        assertThat(testEvent, is(eventFrom(source)));
    }
```

With subtype specified.

```
@Test
    public void test_eventFrom_type() throws Exception {
        // Given
        Object source = new Object();
        EventObject testEvent = new MenuEvent(source);

        // Then
        assertThat(testEvent, is(eventFrom(MenuEvent.class, source)));
    }
```

3.1.21. greaterThan()

```
    public void test_greaterThan() throws Exception {
        // Given
        Integer testValue = 5;

        // Then
        assertThat(testValue, is(greaterThan(3)));
    }
```

3.1.22. greaterThanOrEqual()

Matcher which matches if an input test value is greater than or equal to a specified value.

```
@Test
    public void test_greaterThanOrEqualTo() throws Exception {
        // Given
        Integer testValue = 3;

        // Then
        assertThat(testValue, is(greaterThanOrEqualTo(3)));
    }
```

3.1.23. hasEntry()

Matchers which match if a given map contains an entry which matches the specified key and value, or matchers.

Actual Values

```
@Test
    public void test_hasEntry() throws Exception {
        // Given
        Integer testKey = 1;
        String testValue = "one";

        Map<Integer, String> testMap = new HashMap<>();
        testMap.put(testKey, testValue);

        // Then
        assertThat(testMap, hasEntry(1, "one"));
    }
```

Matchers

```
@Test
    public void test_hasEntry_matchers() throws Exception {
        // Given
        Integer testKey = 2;
        String testValue = "two";

        Map<Integer, String> testMap = new HashMap<>();
        testMap.put(testKey, testValue);

        // Then
        assertThat(testMap, hasEntry(greaterThan(1), endsWith("o")));
    }
```

3.1.24. hasItem()

Matchers which match if the input Iterable has at least one item that matches the specified value or matcher.

Actual Value

```
@Test
    public void test_hasItem() throws Exception {
        // Given
        List<Integer> testList = Arrays.asList(1,2,7,5,4,8);

        // Then
        assertThat(testList, hasItem(4));
    }
```

Matcher

```
@Test
    public void test_hasItem_matcher() throws Exception {
        // Given
        List<Integer> testList = Arrays.asList(1,2,7,5,4,8);

        // Then
        assertThat(testList, hasItem(is(greaterThan(6))));
    }
```

3.1.25. hasItemInArray()

Matchers which match if the input Array has at least one item that matches the specified value or matcher.

Actual Value

```
@Test
```

```
        // Then
        assertThat(test, hasItemInArray(4));
    }
```

Matcher

```
@Test
    public void test_hasItemInArray_matcher() throws Exception {
        // Given
        Integer[] test = {1,2,7,5,4,8};

        // Then
        assertThat(test, hasItemInArray(is(greaterThan(6))));
    }
```

3.1.26. hasItems()

Matchers which match if the input Iterable has all of the specified values or matchers, in any order.

Actual Values

```
public void test_hasItems() throws Exception {
        // Given
        List<Integer> testList = Arrays.asList(1,2,7,5,4,8);

        // Then
        assertThat(testList, hasItems(4, 2, 5));
    }
```

Matchers

```
@Test
    public void test_hasItems_matcher() throws Exception {
        // Given
        List<Integer> testList = Arrays.asList(1,2,7,5,4,8);

        // Then
        assertThat(testList, hasItems(greaterThan(6), lessThan(2)));
    }
```

3.1.27. hasKey()

Matchers which match if the input Map has at least one key which matches the specified value or matcher.

Actual Value

```
@Test
    public void test_hasKey() throws Exception {
        // Given
        Map<String, String> testMap = new HashMap<>();
        testMap.put("hello", "there");
        testMap.put("how", "are you?");

        // Then
        assertThat(testMap, hasKey("hello"));
    }
```

Matcher

```
@Test
    public void test_hasKey_matcher() throws Exception {
        // Given
        Map<String, String> testMap = new HashMap<>();
        testMap.put("hello", "there");
        testMap.put("how", "are you?");

        // Then
        assertThat(testMap, hasKey(startsWith("h")));
    }
```

3.1.28. hasProperty()

Matcher which matches if the input Object satisfies the Bean Convention and has a property with the specified name and optionally the value of the property matches the specified matcher.

Property Name

```
@Test
    public void test_hasProperty() throws Exception {
        // Given
        JTextField testBean = new JTextField();
        testBean.setText("Hello, World!");

        // Then
        assertThat(testBean, hasProperty("text"));
    }
```

Property Name and Value Matcher

```
        testBean.setText("Hello, World!");

        // Then
        assertThat(testBean, hasProperty("text", startsWith("H")));
    }
```

3.1.29. hasSize()

Matchers which match if the input Collection has the specified size, or it's size matches the specified matcher.
Actual Value

```
@Test
    public void test_hasSize() throws Exception {
        // Given
        List<Integer> testList = Arrays.asList(1,2,3,4,5);

        // Then
        assertThat(testList, hasSize(5));
    }
```

Matcher

```
@Test
    public void test_hasSize_matcher() throws Exception {
        // Given
        List<Integer> testList = Arrays.asList(1,2,3,4,5);

        // Then
        assertThat(testList, hasSize(lessThan(10)));
    }
```

3.1.30. hasToString()

Matchers which match if the input Object's toString() method matches either the specified String or the input matcher.

Atual Value

```
@Test
    public void test_hasToString() throws Exception {
        // Given
        Integer testValue = 4;

        // Then
        assertThat(testValue, hasToString("4"));
    }
```

Matcher

```
@Test
    public void test_hasToString_matcher() throws Exception {
        // Given
        Double testValue = 3.14;

        // Then
        assertThat(testValue, hasToString(containsString(".")));
    }
```

3.1.31. hasValue()

Matchers which match if the input Map has at least one value that matches the specified value or matcher.

Actual Value

```
@Test
    public void test_hasValue() throws Exception {
        // Given
        Map<String, String> testMap = new HashMap<>();
        testMap.put("hello", "there");
        testMap.put("how", "are you?");

        // Then
        assertThat(testMap, hasValue("there"));
    }
```

Matcher

```
@Test
    public void test_hasValue_matcher() throws Exception {
        // Given
        Map<String, String> testMap = new HashMap<>();
        testMap.put("hello", "there");
        testMap.put("how", "are you?");

        // Then
        assertThat(testMap, hasValue(containsString("?")));
    }
```

3.1.32. hasXPath()

```
@Test
    public void test_hasXPath() throws Exception {
        // Given
        DocumentBuilder docBuilder = DocumentBuilderFactory.newInstance().newDocumentBuilder();
        Node testNode = docBuilder.parse(
                new InputSource(new StringReader("<xml><top><middle><bottom>value</bottom></middle></top></xml>")))
                .getDocumentElement();

        // Then
        assertThat(testNode, hasXPath("/xml/top/middle/bottom"));
    }
```

Does the Node contain a Node which matches the input XPath expression and does that Node have a value which matches the specified matcher?

```
@Test
    public void test_hasXPath_matcher() throws Exception {
        // Given
        DocumentBuilder docBuilder = DocumentBuilderFactory.newInstance().newDocumentBuilder();
        Node testNode = docBuilder.parse(
                new InputSource(new StringReader("<xml><top><middle><bottom>value</bottom></middle></top></xml>")))
                .getDocumentElement();

        // Then
        assertThat(testNode, hasXPath("/xml/top/middle/bottom", startsWith("val")));
    }
```

Does the Node contain a Node in the specified namespace which matches the input XPath expression?

```
@Test
public void test_hasXPath_namespace() throws Exception {
   // Given
   DocumentBuilderFactory docFactory = DocumentBuilderFactory.newInstance();
   docFactory.setNamespaceAware(true);
   DocumentBuilder docBuilder = docFactory.newDocumentBuilder();
   Node testNode = docBuilder.parse(
           new InputSource(new StringReader("<xml xmlns:prefix='http://namespace-uri'><top><middle><prefix:bottom>value</prefix:bottom></middle></top></xml>")))
           .getDocumentElement();

   NamespaceContext namespace = new NamespaceContext() {
       public String getNamespaceURI(String prefix) {
           return "http://namespace-uri";
       }

       public String getPrefix(String namespaceURI) {
           return null;
       }

       public Iterator<String> getPrefixes(String namespaceURI) {
           return null;
       }
   };

   // Then
   assertThat(testNode, hasXPath("//prefix:bottom", namespace));
}
```

Does the Node contain a Node in the specified namespace which matches the input XPath expression and does that Node have a value which matches the specified matcher?

```
@Test
public void test_hasXPath_namespace_matcher() throws Exception {
  // Given
  DocumentBuilderFactory docFactory = DocumentBuilderFactory.newInstance();
  docFactory.setNamespaceAware(true);
  DocumentBuilder docBuilder = docFactory.newDocumentBuilder();
  Node testNode = docBuilder.parse(
        new InputSource(new StringReader("<xml xmlns:prefix='http://namespace-uri'><top><middle><prefix:bottom>value</prefix:bottom></middle></top></xml>")))
        .getDocumentElement();

  NamespaceContext namespace = new NamespaceContext() {
     public String getNamespaceURI(String prefix) {
        return "http://namespace-uri";
     }

     public String getPrefix(String namespaceURI) {
        return null;
     }

     public Iterator<String> getPrefixes(String namespaceURI) {
        return null;
     }
  };

  // Then
  assertThat(testNode, hasXPath("//prefix:bottom", namespace, startsWith("val")));
}
```

3.1.33. instanceOf()

```java
public void test_instanceOf() throws Exception {
  // Given
  Object string = "Hello, World!";

  // Then
  assertThat(string, instanceOf(String.class));
}
```

3.1.34. isEmptyOrNullString()

Matcher which matches when the input string is either empty or null.

```java
@Test
public void test_isEmptyOrNullString() throws Exception {
  // Given
  String emptyString = ";
  String nullString = null;

  // Then
  assertThat(emptyString, isEmptyOrNullString());
  assertThat(nullString, isEmptyOrNullString());
}
```

3.1.35. isEmptyString()

Matcher which matches when the input string is empty.

```java
@Test
public void test_isEmptyString() throws Exception {
  // Given
  String emptyString = ";

  // Then
  assertThat(emptyString, isEmptyString());
}
```

3.1.36. isIn()

Matcher which matches when the input item is found within the given Collection or Array.

```java
@Test
public void test_isIn() throws Exception {
  // Given
  Set<Integer> set = new HashSet<>();
  set.add(3);
  set.add(6);
  set.add(4);

  // Then
  assertThat(4, isIn(set));
}
```

3.1.37. isOneOf()

Matcher which matches when the input object is one of the given objects.

```java
@Test
public void test_isOneOf() throws Exception {
  // Then
  assertThat(4, isOneOf(3,4,5));
}
```

3.1.38. iterableWithSize()

Matchers which match when the input Iterable has the specified size, or matches the specified size matcher.

Actual Value

```java
@Test
public void test_iterableWithSize() throws Exception {
  // Given
  Set<Integer> set = new HashSet<>();
  set.add(3);
  set.add(6);
  set.add(4);

  // Then
  assertThat(set, iterableWithSize(3));
}
```

Matcher

```java
@Test
public void test_iterableWithSize_matcher() throws Exception {
  // Given
  Set<Integer> set = new HashSet<>();
  set.add(3);
  set.add(6);
}
```

3.1.39. lessThan()

Matcher which matches Comparable objects where the input object is less than the specified value, using the compareTo method.

```
@Test
public void test_lessThan() throws Exception {
  // Then
  assertThat("apple", lessThan("zoo"));
}
```

3.1.40. lessThanOrEqualTo()

Matcher which matches Comparable objects where the input object is less than or equal to the specified value, using the compareTo method.

```
@Test
public void test_lessThanOrEqualTo() throws Exception {
   // Then
   assertThat(2, lessThanOrEqualTo(2));
}
```

3.1.41. not()

Matcher which wraps an existing matcher and inverts it's matching logic

```
@Test
public void test_not_matcher() throws Exception {
   // Then
   assertThat("zoo", not(lessThan("apple")));
}
```

Also an alias for

```
not(equalTo(...))
```

when used with a value instead of a matcher

```
@Test
public void test_not_value() throws Exception {
   // Then
   assertThat("apple", not("orange"));
}
```

3.1.42. notNullValue()

Matcher which matches when the input value is not null.

```
@Test
public void test_notNullValue() throws Exception {
   // Then
   assertThat("apple", notNullValue());
}
```

3.1.43. nullValue()

Matcher which matches when the input value is null.

```
@Test
public void test_nullValue() throws Exception {
   // Given
   Object nothing = null;

   // Then
   assertThat(nothing, nullValue());
}
```

3.1.44. sameInstance()

Matcher which matches when the input object is the same instance as the specified value.

```
@Test
public void test_sameInstance() throws Exception {
   // Given
   Object one = new Object();
   Object two = one;

   // Then
   assertThat(one, sameInstance(two));
}
```

3.1.45. samePropertyValuesAs()

Given the following Java class:

```
public class Bean {

    private Integer number;
    private String text;

    public Integer getNumber() {
        return number;
    }

    public void setNumber(Integer number) {
        this.number = number;
    }

    public String getText() {
        return text;
    }

    public void setText(String text) {
        this.text = text;
    }
}
```

We can write the following test:

```
@Test
    public void test_samePropertyValuesAs() throws Exception {
        // Given
        Bean one = new Bean();
        one.setText("text");
        one.setNumber(3);

        Bean two = new Bean();
        two.setText("text");
        two.setNumber(3);

        // Then
        assertThat(one, samePropertyValuesAs(two));
    }
```

3.1.46. startsWith()

Matcher which matches if the input string starts with the given prefix.

```
@Test
    public void test_startsWith() throws Exception {
        // Given
        String test = "Beginnings are important!";

        // Then
        assertThat(test, startsWith("Beginning"));
    }
```

3.1.47. stringContainsInOrder()

Matcher which matches if the input String contains the substrings in the given Iterable, in the order in which they are returned from the Iterable.

```
@Test
    public void test_stringContainsInOrder() throws Exception {
        // Given
        String test = "Rule number one: two's company, but three's a crowd!";

        // Then
        assertThat(test, stringContainsInOrder(Arrays.asList("one", "two", "three")));
    }
```

3.1.48. theInstance()

Matcher which matches when the input object is the same instance as the specified value. Behaves the same as 'sameInstance()'

```
@Test
public void test_theInstance() throws Exception {
    // Given
    Object one = new Object();
    Object two = one;

    // Then
    assertThat(one, theInstance(two));
}
```

3.1.49. typeCompatibleWith()

Matcher which matches when objects of the input Type can be assigned to references of the specified base Type.

```
@Test
    public void test_typeCompatibleWith() throws Exception {
        // Given
        Integer integer = 3;
```

3.2. Simple Matchers Combining Other Matchers

The following matchers primarily work to combine other matchers.

3.2.1. allOf()

Matcher which matches when all of the input Matchers match, behaves like a Logical AND. Can take individual Matchers or an Iterable of Matchers.

Individual Matchers

```
@Test
    public void test_allOf_individual() throws Exception {
        // Given
        String test = "starting off well, gives content meaning, in the end";

        // Then
        assertThat(test, allOf(startsWith("start"), containsString("content"), endsWith("end")));
    }
```

Iterable of Matchers

```
@Test
    public void test_allOf_iterable() throws Exception {
        // Given
        String test = "Hello, world!";

        List<Matcher<? super String>> matchers = Arrays.asList(containsString("world"),
startsWith("Hello"));

        // Then
        assertThat(test, allOf(matchers));
    }
```

3.2.2. anyOf()

Matcher which matches when any of the input Matchers match, behaves like a Logical OR. Can take individual Matchers or an Iterable of Matchers.

Individual Matchers

```
@Test
    public void test_anyOf_individual() throws Exception {
        // Given
        String test = "Some things are present, some things are not!";

        // Then
        assertThat(test, anyOf(containsString("present"), containsString("missing")));
    }
```

Iterable of Matchers

```
@Test
    public void test_anyOf_iterable() throws Exception {
        // Given
        String test = "Hello, world!";

        List<Matcher<? super String>> matchers = Arrays.asList(containsString("Hello"),
containsString("Earth"));

        // Then
        assertThat(test, anyOf(matchers));
    }
```

3.2.3. array()

Matcher which matches when the elements of an input array individually match using the specified Matchers, in order. The number of Matchers must be equal to the size of the array.

```
@Test
    public void test_array() throws Exception {
        // Given
        String[] test = {"To be", "or not to be", "that is the question!"};

        // Then
        assertThat(test, array(startsWith("To"), containsString("not"), instanceOf(String.class)));
    }
```

3.2.4. both()

Matcher which, when used in combination with it's combinable matcher .and() will match when both specified matchers match.

```
@Test
    public void test_both() throws Exception {
        assertThat(test, both(startsWith("Hello")).and(endsWith("world!")));
    }
```

3.2.5. either()

Matcher which, when used in combination with it's combinable matcher .or() will match when either if the specified matchers match.

```
@Test
    public void test_either() throws Exception {
        // Given
        String test = "Hello, world!";

        // Then
        assertThat(test, either(startsWith("Hello")).or(endsWith("universe!")));
    }
```

3.2.6. is()

Matcher which matches when it's input matcher matches, used simply for convenience and to make assertions read more like English.

```
@Test
    public void test_is_matcher() throws Exception {
        // Given
        Integer test = 5;

        // Then
        assertThat(test, is(greaterThan(3)));
    }
```

Also used as an alias for

```
is(equalTo(...))
```

, similar to

```
not(...)
```

and

```
not(equalTo(...))
```

```
@Test
    public void test_is_value() throws Exception {
        // Given
        Integer test = 5;

        // Then
        assertThat(test, is(5));
    }
```

3.2.7. describedAs()

Matcher which is used to override the failure output of another matcher. Used when a custom failure output is needed. Arguments are the failure message, the original Matcher and then any values which will be formatted into the failure message using placeholders %0, %1, %2...

```
@Test
    public void test_describedAs() throws Exception {
        // Given
        Integer actual = 7;
        Integer expected = 10;

        // Then
        assertThat(actual, describedAs("input > %0", greaterThan(expected), expected));
    }
```

4. Conclusion

We have now visited with all the Matchers defined in Hamcrest and seen examples of each one in action. There are lot of very useful and powerful Matchers in the library, particularly when used in combination with each other. But sometimes we need to do further than what's there already. In the next tutorial we will examine how to create our own custom Matchers, to extend Hamcrest and make it even more useful!

Tagged with: MOCKITO TESTING