Hamcrest Library

In this lesson, we'll learn how to write assertions using the Hamcrest library.

We'll cover the following

- What is the Hamcrest library?
- Examples
- Most commonly-used matchers

What is the `Hamcrest` library?

`Hamcrest` is a framework that comes with a library of useful matchers for writing *match* rules using matcher objects.

The matchers are most commonly used for:

1. Writing test assertions
2. Data validation
3. Filtering

> **Note:** In Java, we use the Matcher class to search for the regular expression in a particular piece of text.

Examples

We will use `Hamcrest` with `TestNG` to demonstrate its usage.

Let's take a look at the code below and its *explanation* in the **method comments**.

```
import static org.hamcrest.MatcherAssert.assertThat;
import static org.hamcrest.Matchers.anyOf;
import static org.hamcrest.Matchers.anything;
import static org.hamcrest.Matchers.containsString;
import static org.hamcrest.Matchers.equalToIgnoringCase;
import static org.hamcrest.Matchers.greaterThan;
import static org.hamcrest.Matchers.greaterThanOrEqualTo;
import static org.hamcrest.Matchers.hasEntry;
import static org.hamcrest.Matchers.hasItem;
```

```
import static org.hamcrest.Matchers.hasItem;
import static org.hamcrest.Matchers.hasItemInArray;
import static org.hamcrest.Matchers.instanceOf;
import static org.hamcrest.Matchers.is;
import static org.hamcrest.Matchers.nullValue;

import java.util.Arrays;
import java.util.HashMap;
import java.util.List;
import java.util.Map;

import org.testng.annotations.Test;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

public class APIDemo {

        private static Logger LOG = LoggerFactory.getLogger(APIDemo.class);

  /**
   *  anything() : Creates a matcher that always matches, regardless of the examined object.
   */
  @Test
  public void test_anything() {

             LOG.info("Test for anything()");
       String name = "xyz";
       assertThat(name, is(anything()));


  }

  /**
   * hasEntry() :  Creates a matcher for Maps, matching when the examined
   *                       Map contains at least one entry whose key equals the specified key
   *                       and whose value equals the specified value.
   */
  @Test
  public void test_hasEntry() throws Exception {

           LOG.info("Test for hasEntry()");
      Integer id = 1;
      String val = "one";

      Map<Integer, String> testMap = new HashMap<>();
      testMap.put(id, val);

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

  }


 /**
  * anyOf() : Creates a matcher that matches if the examined object matches ANY of the specified ma
  *           For example: assertThat("myValue", anyOf(startsWith("foo"), containsString("Val")))
  */

  @Test
  public void test_anyOf() throws Exception {

           LOG.info("Test for anyOf()");
      String check = "It's a great day today!";
      assertThat(check, anyOf(containsString("great"), containsString("bad")));
```

```
}

/**
 * instanceOf() : Creates a matcher that matches when the examined object is an instance of the
 */
@Test
public void test_instanceOf() throws Exception {

        LOG.info("Test for instanceOf()");
  Object string = "hello!";
  assertThat(string, instanceOf(String.class));

}


/**
 * nullValue() :Creates a matcher that matches if examined object is null.
                                For example: assertThat(cheese, is(nullValue())
 */
@Test
public void test_nullValue() throws Exception {

        LOG.info("Test for nullValue()");
  String nullString = null;
  assertThat(nullString, nullValue());

}


/**
 * hasItem() : Creates a matcher that matches the item in the Iterable.
 */
@Test
public void test_hasItem() throws Exception {

          LOG.info("Test for hasItem()");
    List<String> testList = Arrays.asList("one","two","three","four");
    assertThat(testList, hasItem("two"));
}

/**
 * hasItemInArray() : A shortcut to the frequently used hasItemInArray(equalTo(x)).
 */
@Test
public void test_hasItemInArray() throws Exception {

          LOG.info("Test for hasItemInArray()");
    Integer[] check = {1,2,3,4,5,6};
    assertThat(check, hasItemInArray(2));
}

/**
 *  greaterThan(), greaterThanOrEqual() : Creates a matcher for comparison.
 */
@Test
public void test_greaterThan() throws Exception {

          LOG.info("Test for greaterThan() and greaterThanOrEqual()");
    int testValue = 5;
    assertThat(testValue, is(greaterThan(3)));
    assertThat(testValue, is(greaterThanOrEqualTo(5)));
```

```
  }

  /**
   * equalToIgnoringCase() : Creates a matcher of String that matches when the examined
   *                                                 string is equal to the specified expectedString,
   */
  @Test
  public void test_equalToIgnoringCase() throws Exception {

            LOG.info("Test for equalToIgnoringCase()");
      String check = "hello";
      assertThat(check, equalToIgnoringCase("heLLO"));

  }

}
```




Most commonly-used matchers

Matcher	Description
`contains`	Matches if the examined object contains the specified value
`empty`	Matches if the examined object is empty
`equalTo`	Matches if the examined object is logically equal to the specified value
`greaterThan`	Matches if the examined object is greater than the specified value
`hasEntry`	Matches if the examined `Map` has specified (key,value) entry
`isEmptyString`	Matches if the examined object is an empty string
`startsWith`	Matches if the examined object starts with a specific value

	with a specific value
`instanceOf`	Matches if the examined object is an instance of a specific object type
`notNullValue`	Matches if the examined object does not have a null value

That's all about the `Hamcrest` Matchers. In the next lesson, we'll learn about object serialization and deserialization in Java.