**Écrire des assertions avec JUnit 5 et Hamcrest**

**Si vous souhaitez savoir comment travailler plus intelligemment et gagner du temps lorsque vous rédigez des tests avec JUnit 5, vous devriez jeter un œil à mon**[**cours Introduction à JUnit 5**](https://www.cleantestautomation.com/get-started-with-junit-5/)**. Il comprend 24 leçons, 47 exercices et 13 quiz** .

Cet article de blog décrit comment nous pouvons écrire des assertions avec Hamcrest lorsque nous écrivons des tests avec JUnit 5. Après avoir terminé cet article de blog, nous :

* Peut obtenir les dépendances requises avec Maven et Gradle.
* Sachez comment écrire des assertions de base avec Hamcrest.
* Comprenez comment nous pouvons combiner plusieurs matchers Hamcrest.
* Peut personnaliser le message d'erreur affiché lorsqu'une assertion échoue.

Commençons.

**Cet article de blog suppose que :**

* [Vous pouvez créer des classes de test avec JUnit 5](https://www.petrikainulainen.net/programming/testing/junit-5-tutorial-writing-our-first-test-class/)
* [Vous pouvez écrire des tests imbriqués avec JUnit 5](https://www.petrikainulainen.net/programming/testing/junit-5-tutorial-writing-nested-tests/)

**Obtenir les dépendances requises**

Avant de pouvoir écrire des assertions avec Hamcrest, nous devons nous assurer que la hamcrest-librarydépendance (version 2.2) est trouvée à partir du chemin de classe.

Si nous utilisons Maven, nous devons ajouter la hamcrest-librarydépendance à la testportée. Nous pouvons le faire en ajoutant l'extrait suivant à la dependenciessection de notre fichier *pom.xml :*

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | <**dependency**>      <**groupId**>org.hamcrest</**groupId**>      <**artifactId**>hamcrest-library</**artifactId**>      <**version**>2.2</**version**>      <**scope**>test</**scope**>  </**dependency**> |

Si nous utilisons Gradle, nous devons ajouter la hamcrest-librarydépendance à la testImplementationconfiguration des dépendances. Nous pouvons le faire en ajoutant l'extrait suivant à notre fichier *build.gradle :*

|  |  |
| --- | --- |
| 1  2  3  4  5 | dependencies {      testImplementation(              'org.hamcrest:hamcrest-library:2.2'      )  } |

Après nous être assurés que la hamcrest-librarydépendance est trouvée à partir du chemin de classe, nous pouvons écrire des assertions avec Hamcrest. Voyons comment nous pouvons le faire.

**Écrire des assertions avec Hamcrest**

Si vous avez utilisé Hamcrest avec JUnit 4, vous vous souviendrez probablement que vous deviez utiliser la assertThat()méthode de la org.junit.Assertclasse. Cependant, l'API JUnit 5 ne dispose pas de méthode prenant un matcher Hamcrest comme paramètre de méthode. Le guide de l'utilisateur JUnit 5 [explique cette décision de conception](https://junit.org/junit5/docs/current/user-guide/#writing-tests-assertions-third-party) comme suit :

Cependant, la classe de JUnit Jupiter org.junit.jupiter.api.Assertionsne fournit pas de assertThat()méthode comme celle trouvée dans org.junit.Assertla classe de JUnit 4 qui accepte un Hamcrest Matcher. Au lieu de cela, les développeurs sont encouragés à utiliser la prise en charge intégrée des matchers fournie par les bibliothèques d'assertions tierces.

En d’autres termes, si nous voulons utiliser les matchers Hamcrest, nous devons utiliser la assertThat()méthode de la org.hamcrest.MatcherAssertclasse. Cette méthode prend deux ou trois paramètres de méthode qui sont décrits ci-dessous :

1. Un message d'erreur facultatif qui s'affiche lorsque notre assertion échoue.
2. La valeur ou l'objet réel.
3. Un Matcherobjet qui spécifie la valeur attendue. Nous pouvons créer de nouveaux Matcherobjets en utilisant les staticméthodes d'usine fournies par la org.hamcrest.Matchersclasse.

**Lecture supplémentaire :**

* [Le Javadoc de la MatcherAssertclasse](http://hamcrest.org/JavaHamcrest/javadoc/2.2/org/hamcrest/MatcherAssert.html)
* [Le Javadoc de la Matchersclasse](http://hamcrest.org/JavaHamcrest/javadoc/2.2/org/hamcrest/Matchers.html)

Ensuite, nous examinerons quelques exemples qui démontrent comment nous pouvons écrire des assertions avec Hamcrest. Commençons par découvrir comment écrire des assertions pour booleandes valeurs.

**Affirmation de valeurs booléennes**

Si nous voulons vérifier qu'une booleanvaleur est true, nous devons créer notre matcher Hamcrest en appelant la is()méthode de la Matchersclasse. En d’autres termes, nous devons utiliser cette assertion :

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  dix  11  12  13  14  15  16  17  18  19  20  21 | **import** org.junit.jupiter.api.DisplayName;  **import** org.junit.jupiter.api.Nested;  **import** org.junit.jupiter.api.Test;    **import** **static** org.hamcrest.Matchers.is;  **import** **static** org.hamcrest.MatcherAssert.assertThat;    @DisplayName("Write assertions for booleans")  **class** BooleanAssertionTest {        @Nested      @DisplayName("When boolean is true")  **class** WhenBooleanIsTrue {            @Test          @DisplayName("Should be true")  **void** shouldBeTrue() {              assertThat(**true**, is(**true**));          }      }  } |

Si nous voulons vérifier qu'une booleanvaleur est false, nous devons créer notre matcher Hamcrest en appelant la is()méthode de la Matchersclasse. En d’autres termes, nous devons utiliser cette assertion :

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  dix  11  12  13  14  15  16  17  18  19  20  21 | **import** org.junit.jupiter.api.DisplayName;  **import** org.junit.jupiter.api.Nested;  **import** org.junit.jupiter.api.Test;    **import** **static** org.hamcrest.Matchers.is;  **import** **static** org.hamcrest.MatcherAssert.assertThat;    @DisplayName("Write assertions for booleans")  **class** BooleanAssertionTest {        @Nested      @DisplayName("When boolean is false")  **class** WhenBooleanIsFalse {            @Test          @DisplayName("Should be false")  **void** shouldBeFalse() {              assertThat(**false**, is(**false**));          }      }  } |

Passons à autre chose et découvrons comment vérifier qu'un objet est nullou non null.

**Affirmer qu'un objet est nul ou n'est pas nul**

Si nous voulons vérifier qu'un objet est null, nous devons créer notre matcher Hamcrest en appelant la nullValue()méthode de la Matchersclasse. En d’autres termes, nous devons utiliser cette assertion :

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  dix  11  12  13  14  15  16  17  18  19  20  21 | **import** org.junit.jupiter.api.DisplayName;  **import** org.junit.jupiter.api.Nested;  **import** org.junit.jupiter.api.Test;    **import** **static** org.hamcrest.MatcherAssert.assertThat;  **import** **static** org.hamcrest.Matchers.nullValue;    @DisplayName("Writing assertions for objects")  **class** ObjectAssertionTest {        @Nested      @DisplayName("When object is null")  **class** WhenObjectIsNull {            @Test          @DisplayName("Should be null")  **void** shouldBeNull() {              assertThat(**null**, nullValue());          }      }  } |

Si nous voulons vérifier qu'un objet ne l'est pas null, nous devons créer notre matcher Hamcrest en appelant la notNullValue()méthode de la Matchersclasse. En d’autres termes, nous devons utiliser cette assertion :

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  dix  11  12  13  14  15  16  17  18  19  20  21 | **import** org.junit.jupiter.api.DisplayName;  **import** org.junit.jupiter.api.Nested;  **import** org.junit.jupiter.api.Test;    **import** **static** org.hamcrest.MatcherAssert.assertThat;  **import** **static** org.hamcrest.Matchers.notNullValue;    @DisplayName("Writing assertions for objects")  **class** ObjectAssertionTest {        @Nested      @DisplayName("When object is not null")  **class** WhenObjectIsNotNotNull {            @Test          @DisplayName("Should not be null")  **void** shouldNotBeNull() {              assertThat(**new** Object(), notNullValue());          }      }  } |

Nous découvrirons ensuite comment vérifier que deux objets (ou valeurs) sont égaux ou non.

**Affirmer que deux objets ou valeurs sont égaux**

Si nous voulons vérifier que la valeur (ou l'objet) attendue est égale à la valeur (ou l'objet) réelle, nous devons créer notre matcher Hamcrest en appelant la equalTo()méthode de la Matchersclasse. Par exemple, si nous voulons comparer deux Integerobjets, nous devons utiliser cette assertion :

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  dix  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29 | **import** org.junit.jupiter.api.DisplayName;  **import** org.junit.jupiter.api.Nested;  **import** org.junit.jupiter.api.Test;    **import** **static** org.hamcrest.MatcherAssert.assertThat;  **import** **static** org.hamcrest.Matchers.equalTo;    @DisplayName("Writing assertions for objects")  **class** ObjectAssertionTest {        @Nested      @DisplayName("When two objects are equal")  **class** WhenTwoObjectsAreEqual {            @Nested          @DisplayName("When objects are integers")  **class** WhenObjectsAreIntegers {    **private** **final** Integer ACTUAL = 9;  **private** **final** Integer EXPECTED = 9;                @Test              @DisplayName("Should be equal")  **void** shouldBeEqual() {                  assertThat(ACTUAL, equalTo(EXPECTED));              }          }      }  } |

Si nous voulons vérifier que la valeur (ou l'objet) attendue n'est pas égale à la valeur (ou l'objet) réelle, nous devons créer notre matcher Hamcrest en appelant la not()méthode de la Matchersclasse. Par exemple, si nous voulons comparer deux Integerobjets, nous devons utiliser cette assertion :

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  dix  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29 | **import** org.junit.jupiter.api.DisplayName;  **import** org.junit.jupiter.api.Nested;  **import** org.junit.jupiter.api.Test;    **import** **static** org.hamcrest.MatcherAssert.assertThat;  **import** **static** org.hamcrest.Matchers.not;    @DisplayName("Writing assertions for objects")  **class** ObjectAssertionTest {        @Nested      @DisplayName("When two objects aren't equal")  **class** WhenTwoObjectsAreNotEqual {            @Nested          @DisplayName("When objects are integers")  **class** WhenObjectsAreIntegers {    **private** **final** Integer ACTUAL = 9;  **private** **final** Integer EXPECTED = 4;                @Test              @DisplayName("Should not be equal")  **void** shouldNotBeEqual() {                  assertThat(ACTUAL, not(EXPECTED));              }          }      }  } |

Passons à autre chose et découvrons comment écrire des assertions pour les références d'objets.

**Affirmation de références d'objet**

Si nous voulons nous assurer que deux objets font référence au même objet, nous devons créer notre matcher Hamcrest en appelant la sameInstance()méthode de la Matchersclasse. En d’autres termes, nous devons utiliser cette assertion :

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  dix  11  12  13  14  15  16  17  18  19  20  21  22  23  24 | **import** org.junit.jupiter.api.DisplayName;  **import** org.junit.jupiter.api.Nested;  **import** org.junit.jupiter.api.Test;    **import** **static** org.hamcrest.MatcherAssert.assertThat;  **import** **static** org.hamcrest.Matchers.sameInstance;    @DisplayName("Writing assertions for objects")  **class** ObjectAssertionTest {        @Nested      @DisplayName("When two objects refer to the same object")  **class** WhenTwoObjectsReferToSameObject {    **private** **final** Object ACTUAL = **new** Object();  **private** **final** Object EXPECTED = ACTUAL;            @Test          @DisplayName("Should refer to the same object")  **void** shouldReferToSameObject() {              assertThat(ACTUAL, sameInstance(EXPECTED));          }      }  } |

Si nous voulons nous assurer que deux objets ne font pas référence au même objet, nous devons inverser l'attente spécifiée par la sameInstance()méthode en utilisant la not()méthode de la Matchersclasse. En d’autres termes, nous devons utiliser cette assertion :

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  dix  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25 | **import** org.junit.jupiter.api.DisplayName;  **import** org.junit.jupiter.api.Nested;  **import** org.junit.jupiter.api.Test;    **import** **static** org.hamcrest.MatcherAssert.assertThat;  **import** **static** org.hamcrest.Matchers.not;  **import** **static** org.hamcrest.Matchers.sameInstance;    @DisplayName("Writing assertions for objects")  **class** ObjectAssertionTest {        @Nested      @DisplayName("When two objects don't refer to the same object")  **class** WhenTwoObjectsDoNotReferToSameObject {    **private** **final** Object ACTUAL = **new** Object();  **private** **final** Object EXPECTED = **new** Object();            @Test          @DisplayName("Should not refer to the same object")  **void** shouldNotReferToSameObject() {             assertThat(ACTUAL, not(sameInstance(EXPECTED)));          }      }  } |

Si vous souhaitez accéder à du matériel à jour qui vous aide à travailler plus intelligemment et à gagner du temps lorsque vous rédigez des tests avec JUnit 5, vous devriez jeter un œil à mon [cours Introduction à JUnit 5](https://www.cleantestautomation.com/get-started-with-junit-5/) . Il comprend 24 leçons, 47 exercices et 13 quiz.

Nous découvrirons ensuite comment vérifier que deux tableaux sont égaux.

**Affirmer que deux tableaux sont égaux**

Si nous voulons vérifier que deux tableaux sont égaux, nous devons créer notre matcher Hamcrest en appelant la equalTo()méthode de la Matchersclasse. Par exemple, si nous voulons vérifier que deux inttableaux sont égaux, nous devons utiliser cette assertion :

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  dix  11  12  13  14  15  16  17  18  19  20  21  22  23  24 | **import** org.junit.jupiter.api.DisplayName;  **import** org.junit.jupiter.api.Nested;  **import** org.junit.jupiter.api.Test;    **import** **static** org.hamcrest.Matchers.equalTo;  **import** **static** org.hamcrest.MatcherAssert.assertThat;    @DisplayName("Write assertions for arrays")  **class** ArrayAssertionTest {        @Nested      @DisplayName("When arrays contain integers")  **class** WhenArraysContainIntegers {    **final** **int**[] ACTUAL = **new** **int**[]{2, 5, 7};  **final** **int**[] EXPECTED = **new** **int**[]{2, 5, 7};            @Test          @DisplayName("Should contain the same integers")  **void** shouldContainSameIntegers() {              assertThat(ACTUAL, equalTo(EXPECTED));          }      }  } |

Deux tableaux sont considérés comme égaux si :

* Ils sont tous les deux nullou vides.
* Les deux tableaux contiennent les « mêmes » objets ou valeurs. Pour être plus précis, JUnit 5 itère les deux tableaux un élément à la fois et garantit que les éléments trouvés à partir de l'index donné sont égaux.

Passons à autre chose et découvrons comment nous pouvons écrire des assertions pour les listes.

**Affirmation de listes**

Si nous voulons écrire une assertion qui vérifie que la taille d'une liste est correcte, nous devons créer notre matcher Hamcrest en appelant la hasSize()méthode de la Matchersclasse. Par exemple, si l’on veut vérifier que la taille d’une liste est de 2, il faut utiliser cette assertion :

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  dix  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38 | **import** org.junit.jupiter.api.BeforeEach;  **import** org.junit.jupiter.api.DisplayName;  **import** org.junit.jupiter.api.Nested;  **import** org.junit.jupiter.api.Test;    **import** java.util.Arrays;  **import** java.util.List;    **import** **static** org.hamcrest.MatcherAssert.assertThat;  **import** **static** org.hamcrest.Matchers.hasSize;    @DisplayName("Writing assertions for lists")  **class** ListAssertionTest {        @Nested      @DisplayName("When we write assertions for elements")  **class** WhenWeWriteAssertionsForElements {    **private** Object first;  **private** Object second;    **private** List<Object> list;            @BeforeEach  **void** createAndInitializeList() {              first = **new** Object();              second = **new** Object();                list = Arrays.asList(first, second);          }            @Test          @DisplayName("Should contain two elements")  **void** shouldContainTwoElements() {              assertThat(list, hasSize(2));          }      }  } |

Si nous voulons vérifier que la liste contient uniquement les éléments attendus dans l'ordre donné, nous devons créer notre matcher Hamcrest en appelant la contains()méthode de la Matchersclasse. Par exemple, si nous voulons vérifier que notre liste contient les bons éléments dans l’ordre donné, nous devons utiliser cette assertion :

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  dix  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38 | **import** org.junit.jupiter.api.BeforeEach;  **import** org.junit.jupiter.api.DisplayName;  **import** org.junit.jupiter.api.Nested;  **import** org.junit.jupiter.api.Test;    **import** java.util.Arrays;  **import** java.util.List;    **import** **static** org.hamcrest.MatcherAssert.assertThat;  **import** **static** org.hamcrest.Matchers.contains;    @DisplayName("Writing assertions for lists")  **class** ListAssertionTest {        @Nested      @DisplayName("When we write assertions for elements")  **class** WhenWeWriteAssertionsForElements {    **private** Object first;  **private** Object second;    **private** List<Object> list;            @BeforeEach  **void** createAndInitializeList() {              first = **new** Object();              second = **new** Object();                list = Arrays.asList(first, second);          }            @Test          @DisplayName("Should contain the correct elements in the given order")  **void** shouldContainCorrectElementsInGivenOrder() {              assertThat(list, contains(first, second));          }      }  } |

Si nous voulons vérifier que la liste contient uniquement les éléments attendus dans n'importe quel ordre, nous devons créer notre matcher Hamcrest en appelant la containsInAnyOrder()méthode de la Matchersclasse. Par exemple, si nous voulons vérifier que notre liste contient les bons éléments dans n'importe quel ordre, nous devons utiliser cette assertion :

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  dix  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38 | **import** org.junit.jupiter.api.BeforeEach;  **import** org.junit.jupiter.api.DisplayName;  **import** org.junit.jupiter.api.Nested;  **import** org.junit.jupiter.api.Test;    **import** java.util.Arrays;  **import** java.util.List;    **import** **static** org.hamcrest.MatcherAssert.assertThat;  **import** **static** org.hamcrest.Matchers.containsInAnyOrder;    @DisplayName("Writing assertions for lists")  **class** ListAssertionTest {        @Nested      @DisplayName("When we write assertions for elements")  **class** WhenWeWriteAssertionsForElements {    **private** Object first;  **private** Object second;    **private** List<Object> list;            @BeforeEach  **void** createAndInitializeList() {              first = **new** Object();              second = **new** Object();                list = Arrays.asList(first, second);          }            @Test          @DisplayName("Should contain the correct elements in any order")  **void** shouldContainCorrectElementsInAnyOrder() {              assertThat(list, containsInAnyOrder(second, first));          }      }  } |

Si nous voulons nous assurer qu'une liste contient l'élément donné, nous devons créer notre matcher Hamcrest en appelant la hasItem()méthode de la Matchersclasse. Par exemple, si nous voulons vérifier que notre liste contient les Objectéléments stockés dans le champ appelé first, nous devons utiliser cette assertion :

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  dix  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38 | **import** org.junit.jupiter.api.BeforeEach;  **import** org.junit.jupiter.api.DisplayName;  **import** org.junit.jupiter.api.Nested;  **import** org.junit.jupiter.api.Test;    **import** java.util.Arrays;  **import** java.util.List;    **import** **static** org.hamcrest.MatcherAssert.assertThat;  **import** **static** org.hamcrest.Matchers.hasItem;    @DisplayName("Writing assertions for lists")  **class** ListAssertionTest {        @Nested      @DisplayName("When we write assertions for elements")  **class** WhenWeWriteAssertionsForElements {    **private** Object first;  **private** Object second;    **private** List<Object> list;            @BeforeEach  **void** createAndInitializeList() {              first = **new** Object();              second = **new** Object();                list = Arrays.asList(first, second);          }            @Test          @DisplayName("Should contain a correct element")  **void** shouldContainCorrectElement() {              assertThat(list, hasItem(first));          }      }  } |

Si nous voulons nous assurer qu'une liste ne contient pas d'élément, nous devons inverser l'attente spécifiée par la hasItem()méthode en utilisant la not()méthode de la Matchersclasse. En d’autres termes, nous devons utiliser cette assertion :

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  dix  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39 | **import** org.junit.jupiter.api.BeforeEach;  **import** org.junit.jupiter.api.DisplayName;  **import** org.junit.jupiter.api.Nested;  **import** org.junit.jupiter.api.Test;    **import** java.util.Arrays;  **import** java.util.List;    **import** **static** org.hamcrest.MatcherAssert.assertThat;  **import** **static** org.hamcrest.Matchers.hasItem;  **import** **static** org.hamcrest.Matchers.not;    @DisplayName("Writing assertions for lists")  **class** ListAssertionTest {        @Nested      @DisplayName("When we write assertions for elements")  **class** WhenWeWriteAssertionsForElements {    **private** Object first;  **private** Object second;    **private** List<Object> list;            @BeforeEach  **void** createAndInitializeList() {              first = **new** Object();              second = **new** Object();                list = Arrays.asList(first, second);          }            @Test          @DisplayName("Should not contain an incorrect element")  **void** shouldNotContainIncorrectElement() {              assertThat(list, not(hasItem(**new** Object())));          }      }  } |

Si nous voulons vérifier que deux listes sont profondément égales, nous devons créer notre matcher Hamcrest en invoquant la equalTo()méthode de la Matchersclasse. Par exemple, si l’on veut vérifier que deux Integerlistes sont profondément égales, il faut utiliser cette assertion :

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  dix  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27 | **import** org.junit.jupiter.api.DisplayName;  **import** org.junit.jupiter.api.Nested;  **import** org.junit.jupiter.api.Test;    **import** java.util.Arrays;  **import** java.util.List;    **import** **static** org.hamcrest.MatcherAssert.assertThat;  **import** **static** org.hamcrest.Matchers.equalTo;    @DisplayName("Writing assertions for lists")  **class** ListAssertionTest {        @Nested      @DisplayName("When we compare two lists")  **class** WhenWeCompareTwoLists {    **private** **final** List<Integer> ACTUAL = Arrays.asList(1, 2, 3);  **private** **final** List<Integer> EXPECTED = Arrays.asList(1, 2, 3);            @Test          @DisplayName("Should contain the same elements")  **void** shouldContainSameElements() {              assertThat(ACTUAL, equalTo(EXPECTED));          }      }  } |

Deux listes sont considérées comme égales si :

* Ils sont tous les deux nullou vides.
* Les deux listes contiennent les « mêmes » objets ou valeurs. Pour être plus précis, JUnit 5 parcourt les deux listes un élément à la fois et garantit que les éléments trouvés à partir de l'index donné sont égaux.

Nous découvrirons ensuite comment écrire des assertions pour les cartes.

**Affirmation de cartes**

Si nous voulons vérifier qu'une carte contient la clé donnée, nous devons créer notre matcher Hamcrest en appelant la hasKey()méthode de la Matchersclasse. En d’autres termes, nous devons utiliser cette assertion :

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  dix  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36 | **import** org.junit.jupiter.api.BeforeEach;  **import** org.junit.jupiter.api.DisplayName;  **import** org.junit.jupiter.api.Nested;  **import** org.junit.jupiter.api.Test;    **import** java.util.HashMap;  **import** java.util.Map;    **import** **static** org.hamcrest.MatcherAssert.assertThat;  **import** **static** org.hamcrest.Matchers.hasKey;    @DisplayName("Writing assertions for maps")  **class** MapAssertionTest {    **private** **static** **final** String KEY = "key";  **private** **static** **final** String VALUE = "value";    **private** Map<String, String> map;        @BeforeEach  **void** createAndInitializeMap() {          map = **new** HashMap<>();          map.put(KEY, VALUE);      }        @Nested      @DisplayName("When we check if the map contains the given key")  **class** WhenWeCheckIfMapContainsGivenKey {            @Test          @DisplayName("Should contain the correct key")  **void** shouldContainCorrectKey() {              assertThat(map, hasKey(KEY));          }      }  } |

Si nous voulons vérifier qu'une carte ne contient pas la clé donnée, nous devons inverser l'attente spécifiée par la hasKey()méthode en utilisant la not()méthode de la Matchersclasse. En d’autres termes, nous devons utiliser cette assertion :

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  dix  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38 | **import** org.junit.jupiter.api.BeforeEach;  **import** org.junit.jupiter.api.DisplayName;  **import** org.junit.jupiter.api.Nested;  **import** org.junit.jupiter.api.Test;    **import** java.util.HashMap;  **import** java.util.Map;    **import** **static** org.hamcrest.MatcherAssert.assertThat;  **import** **static** org.hamcrest.Matchers.hasKey;  **import** **static** org.hamcrest.Matchers.not;    @DisplayName("Writing assertions for maps")  **class** MapAssertionTest {    **private** **static** **final** String INCORRECT\_KEY = "incorrectKey";  **private** **static** **final** String KEY = "key";  **private** **static** **final** String VALUE = "value";    **private** Map<String, String> map;        @BeforeEach  **void** createAndInitializeMap() {          map = **new** HashMap<>();          map.put(KEY, VALUE);      }        @Nested      @DisplayName("When we check if the map contains the given key")  **class** WhenWeCheckIfMapContainsGivenKey {            @Test          @DisplayName("Should not contain the incorrect key")  **void** shouldNotContainIncorrectKey() {              assertThat(map, not(hasKey(INCORRECT\_KEY)));          }      }  } |

Si nous voulons nous assurer qu'une carte contient la bonne valeur, nous devons créer notre matcher Hamcrest en appelant la hasEntry()méthode de la Matchersclasse. En d’autres termes, nous devons utiliser cette assertion :

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  dix  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36 | **import** org.junit.jupiter.api.BeforeEach;  **import** org.junit.jupiter.api.DisplayName;  **import** org.junit.jupiter.api.Nested;  **import** org.junit.jupiter.api.Test;    **import** java.util.HashMap;  **import** java.util.Map;    **import** **static** org.hamcrest.MatcherAssert.assertThat;  **import** **static** org.hamcrest.Matchers.hasEntry;    @DisplayName("Writing assertions for maps")  **class** MapAssertionTest {    **private** **static** **final** String KEY = "key";  **private** **static** **final** String VALUE = "value";    **private** Map<String, String> map;        @BeforeEach  **void** createAndInitializeMap() {          map = **new** HashMap<>();          map.put(KEY, VALUE);      }        @Nested      @DisplayName("When we check if the map contains the correct value")  **class** WhenWeCheckIfMapContainsCorrectValue {            @Test          @DisplayName("Should contain the correct value")  **void** shouldContainCorrectValue() {              assertThat(map, hasEntry(KEY, VALUE));          }      }  } |

Passons à autre chose et découvrons comment nous pouvons combiner plusieurs matchers Hamcrest.

**Combiner les Matchers Hamcrest**

Nous pouvons maintenant écrire des assertions de base avec Hamcrest. Cependant, nous devons parfois combiner plusieurs matchers Hamcrest. En fait, nous l'avons déjà fait lorsque nous avons inversé l'attente d'un matcher Hamcrest en invoquant la not()méthode de la Matchersclasse.

Ensuite, nous examinerons deux exemples qui démontrent comment nous pouvons combiner les matchers Hamcrest lorsque nous écrivons des assertions pour un Personobjet. Le code source de la Personclasse se présente comme suit :

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  dix  11  12  13  14  15  16  17  18  19  20  21  22  23 | **public** **class** Person {    **private** String firstName;  **private** String lastName;    **public** Person() {}    **public** String getFirstName() {  **return** firstName;      }    **public** String getLastName() {  **return** lastName;      }    **public** **void** setFirstName(String firstName) {  **this**.firstName = firstName;      }    **public** **void** setLastName(String lastName) {  **this**.lastName = lastName;      }  } |

Comme nous pouvons le voir, si nous voulons vérifier qu’une personne porte le nom correct, nous devons nous assurer que l’ Personobjet affirmé porte le nom et le prénom corrects. Lorsque nous écrivons cette assertion, nous devons créer le matcher Hamcrest qui est passé à la assertThat()méthode en utilisant ces Hamcrest Matchers :

* La allOf()méthode de la Matchersclasse renvoie un matcher Hamcrest qui s'attend à ce que l'objet affirmé corresponde à **tous** les matchers Hamcrest spécifiés.
* La hasProperty()méthode de la Matchersclasse renvoie un matcher qui nous permet d'écrire des assertions pour les propriétés de l'objet affirmé.
* La equalTo()méthode de la Matchersclasse renvoie un matcher qui permet de vérifier que la valeur réelle de la propriété est égale à la valeur attendue.

Après avoir rédigé notre assertion, elle se présente comme suit :

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  dix  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33 | **import** org.junit.jupiter.api.BeforeEach;  **import** org.junit.jupiter.api.DisplayName;  **import** org.junit.jupiter.api.Test;    **import** **static** org.hamcrest.MatcherAssert.assertThat;  **import** **static** org.hamcrest.Matchers.allOf;  **import** **static** org.hamcrest.Matchers.equalTo;  **import** **static** org.hamcrest.Matchers.hasProperty;    @DisplayName("Combine multiple assertions")  **class** CombineAssertionsTest {    **private** **static** **final** String FIRST\_NAME = "Jane";  **private** **static** **final** String LAST\_NAME = "Doe";    **private** Person person;        @BeforeEach  **void** createPerson() {          person = **new** Person();          person.setFirstName(FIRST\_NAME);          person.setLastName(LAST\_NAME);      }        @Test      @DisplayName("Should have the correct name")  **void** shouldHaveCorrectName() {          assertThat(person, allOf(                  hasProperty("firstName", equalTo(FIRST\_NAME)),                  hasProperty("lastName", equalTo(LAST\_NAME))          ));      }  } |

D'un autre côté, si nous voulons vérifier qu'une personne a le prénom ou le nom correct, nous devons créer le matcher Hamcrest qui est transmis à la assertThat()méthode en utilisant ces Hamcrest Matchers :

* La anyOf()méthode de la Matchersclasse renvoie un matcher Hamcrest qui s'attend à ce que l'objet affirmé corresponde à **n'importe quel** matcher Hamcrest spécifié.
* La hasProperty()méthode de la Matchersclasse renvoie un matcher qui nous permet d'écrire des assertions pour les propriétés de l'objet affirmé.
* La equalTo()méthode de la Matchersclasse renvoie un matcher qui permet de vérifier que la valeur réelle de la propriété est égale à la valeur attendue.

Après avoir rédigé notre assertion, elle se présente comme suit :

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  dix  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33 | **import** org.junit.jupiter.api.BeforeEach;  **import** org.junit.jupiter.api.DisplayName;  **import** org.junit.jupiter.api.Test;    **import** **static** org.hamcrest.MatcherAssert.assertThat;  **import** **static** org.hamcrest.Matchers.anyOf;  **import** **static** org.hamcrest.Matchers.equalTo;  **import** **static** org.hamcrest.Matchers.hasProperty;    @DisplayName("Combine multiple assertions")  **class** CombineAssertionsTest {    **private** **static** **final** String FIRST\_NAME = "Jane";  **private** **static** **final** String LAST\_NAME = "Doe";    **private** Person person;        @BeforeEach  **void** createPerson() {          person = **new** Person();          person.setFirstName(FIRST\_NAME);          person.setLastName(LAST\_NAME);      }        @Test      @DisplayName("Should have correct first name or last name")  **void** shouldHaveCorrectFirstNameOrLastName() {          assertThat(person, anyOf(                  hasProperty("firstName", equalTo(FIRST\_NAME)),                  hasProperty("lastName", equalTo(LAST\_NAME))          ));      }  } |

De nombreuses méthodes d'usine fournies par la Matchersclasse peuvent prendre un matcher Hamcrest (ou des matchers) comme paramètre de méthode. C'est pourquoi je vous recommande de jeter un œil à [sa documentation](http://hamcrest.org/JavaHamcrest/javadoc/2.2/org/hamcrest/Matchers.html) lorsqu'il semble que vous ne puissiez pas écrire l'assertion dont vous avez besoin. Il y a de fortes chances que vous puissiez l'écrire en combinant plusieurs matchers Hamcrest.

Nous découvrirons ensuite comment fournir un message d'erreur personnalisé qui s'affiche lorsque notre assertion échoue.

**Fournir un message d'erreur personnalisé**

Comme nous nous en souvenons, si nous voulons spécifier un message d'erreur personnalisé qui s'affiche lorsque notre assertion échoue, nous devons transmettre ce message comme premier paramètre de méthode de la assertThat()méthode. Nous pouvons créer ce message d'erreur en utilisant l'une de ces deux options :

* Si le message d'erreur n'a pas de paramètres, nous devons utiliser un Stringlittéral.
* Si le message d'erreur a des paramètres, nous devons utiliser la static format()méthode de la Stringclasse.

Par exemple, si nous voulons créer un message d'erreur qui s'affiche lorsque la liste affirmée ne contient pas l'élément donné, nous devons créer une assertion qui ressemble à ceci :

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  dix  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44 | **import** org.junit.jupiter.api.BeforeEach;  **import** org.junit.jupiter.api.DisplayName;  **import** org.junit.jupiter.api.Nested;  **import** org.junit.jupiter.api.Test;    **import** java.util.Arrays;  **import** java.util.List;    **import** **static** org.hamcrest.MatcherAssert.assertThat;  **import** **static** org.hamcrest.Matchers.hasItem;    @DisplayName("Writing assertions for lists")  **class** ListAssertionTest {        @Nested      @DisplayName("When we write assertions for elements")  **class** WhenWeWriteAssertionsForElements {    **private** Object first;  **private** Object second;    **private** List<Object> list;            @BeforeEach  **void** createAndInitializeList() {              first = **new** Object();              second = **new** Object();                list = Arrays.asList(first, second);          }            @Test          @DisplayName("Should contain a correct element")  **void** shouldContainCorrectElementWithCustomErrorMessage() {              assertThat(String.format(                              "The list doesn't contain the expected object: %s",                              first                      ),                      list,                      hasItem(first)              );          }      }  } |

Il est bon de comprendre que le message d'erreur personnalisé ne remplace pas le message d'erreur par défaut affiché si une assertion échoue. Il s'agit simplement d'un préfixe ajouté au message d'erreur par défaut du matcher Hamcrest utilisé. Au début, cela semble un peu bizarre, mais c'est en fait très utile une fois qu'on s'y est habitué.

Nous pouvons désormais écrire des assertions de base avec Hamcrest, combiner plusieurs correspondants Hamcrest et fournir un message d'erreur personnalisé qui s'affiche lorsqu'une assertion échoue.

Résumons ce que nous avons appris de cet article de blog.

**Résumé**

Cet article de blog nous a appris quatre choses :

* Avant de pouvoir écrire des assertions avec Hamcrest, nous devons nous assurer que la hamcrest-librarydépendance est trouvée à partir du chemin de classe.
* Si nous voulons écrire des assertions avec Hamcrest, nous devons utiliser la assertThat()méthode de la org.hamcrest.MatcherAssertclasse.
* Si nous souhaitons fournir un message d'erreur personnalisé qui s'affiche lorsqu'une assertion échoue, nous devons transmettre ce message d'erreur comme premier paramètre de méthode de la assertThat()méthode.
* Certaines méthodes de la Matchersclasse peuvent prendre un ou plusieurs matchers Hamcrest comme paramètre de méthode. Nous pouvons combiner plusieurs matchers Hamcrest en utilisant ces méthodes.