Découverte et prise en main de l'API *fluent* `AssertJ`

Les bonnes pratiques d'écriture d'un test recommandent d'utiliser le pattern AAA.
Le **3ème A** du pattern **AAA** est indispensable dans l'écriture d'un test automatisé car il fait référence à l'**Assertation**, une étape qui consiste à comparer le résultat attendu avec le résultat obtenu afin que le framework de tests automatisés puissent rendre son *verdict* : soit le test passe (et il est VERT), soit le test échoue (et il est ROUGE).

Pour l'instant, vous avez implémenté l'étape d'assertions en utilisant les méthodes de la classe **Assertions** du paquetage **org.junit.jupiter.api.Assertions** de Junit5 à l'image des méthodes qui proposent essentiellement des méthodes **assertEquals** (ou assertNotEquals) surchargées prenant en premier paramètre le résultat attendu et en second paramètre le résultat actuel :

static void	**assertEquals**(**Long** expected, **Long** actual)
static void	**assertEquals**(**Long** expected, **Long** actual, **String** message)
static void	**assertEquals**(**Long** expected, **Long** actual, **Supplier**<**String** > messageSupplier)
static void	**assertEquals**(**Object** expected, **Object** actual)
static void	**assertEquals**(**Object** expected, **Object** actual, **String** message)
static void	**assertEquals**(**Object** expected, **Object** actual, **Supplier**<**String** > messageSupplier)
static void	**assertEquals**(**Short** expected, short actual)
static void	**assertEquals**(**Short** expected, short actual, **String** message)
static void	**assertEquals**(**Short** expected, short actual, **Supplier**<**String** > messageSupplier)
static void	**assertEquals**(**Short** expected, **Short** actual)

Extrait de la javadoc de Junit5, pour visualiser toutes les services offerts par la classe Assertions, rendez-vous à l'adresse suivante 😉
https://junit.org/junit5/docs/current/api/org.junit.jupiter.api/org/junit/jupiter/api/Assertions.html

Maintenant que vous maîtrisez mieux cette étape d'assertion, nous vous proposons de découvrir une nouvelle API qui vous permettra d'écrire l'étape d'assertion de manière plus lisible, et améliorera ainsi la qualité de votre code de test. En effet, écrire des tests sur des collections peut s'avérer fastidieux et complexe avec les méthodes de bases de JUnit, alors qu'avec l'API AssertJ, l'écriture et la lecture des tests deviendra plus aisée, plus fluide (« fluente »), c'est pour cela qu'on qualifie souvent **`AssertJ` de fluent API**

AssertJ est disponible à l'adresse suivante : https://assertj.github.io/doc/
Comme l'indique la page d'accueil, assertJ propose plusieurs modules, nous nous focaliserons pour commencer uniquement sur le **core module**.

AssertJ - fluent assertions java library

Version 1.0

1. AssertJ Overview

AssertJ is composed of several modules:

- A core module to provide assertions for JDK types (String, Iterable, Stream, Path, File, Map...)
- A Guava module to provide assertions for Guava types (Multimap, Optional...)
- A Joda Time module to provide assertions for Joda Time types (DateTime, LocalDateTime)
- A Neo4J module to provide assertions for Neo4J types (Path, Node, Relationship...)
- A DB module to provide assertions for relational database types (Table, Row, Column...)
- A Swing module provides a simple and intuitive API for functional testing of Swing user interfaces

❑ Pour découvrir **AssertJ**, nous vous proposons de commencer par **créer un nouveau projet maven** que vous appelerez **helloassertj** avec pour commencer un **pom** habituel (celui avec la dépendance à Junit5).

❑ Vous allez ensuite créer une classe de tests JUnit 5 nommée **AssertJTest** dans src/test/java/**helloAssertj** à l'aide de **New→JUnit Test Case** qui contiendra un premier test écrit avec JUnit :

```
@Test
void test_hello_with_assertEquals_Junit5() {
      String hello = "hello AssertJ !";
      assertEquals("hello AssertJ !", hello);
}
```

Implémentez ce test et exécutez-le afin qu'il passe AU VERT 😉

2.4. Quick start
2.4.1. Get assertj-core library
Supported Java versions
Android support
Maven
Gradle
Other build tools

❑ Avant de pouvoir écrire un test plus *fluent* faisant appel aux services offerts par AssertJ, il faut commencer par ajouter la **dépendance à cette API** dans votre **pom.xml**
Pour effectuer cela, rendez-vous donc dans la documentation d'AssertJ dans la rubrique **2.4 Quick start** et plus particulièrement dans **Maven** pour copier le bloc dependency à ajouter dans votre pom.xml.
(https://assertj.github.io/doc/#maven)
Après avoir copier/coller ce bloc et sauver votre pom.xml, n'oubliez pas de le updater (sous Eclipse) pour que ces modifications soient bien prises en compte 😉

❑ Vous pouvez maintenant écrire votre premier test avec assertJ :

```
@Test
void test_hello_with_assertThat_AssertJ() {
	String hello = "hello AssertJ !";
	assertThat(hello).isEqualTo("hello AssertJ !");
}
```

Sans oublier l'import qui va bien :

```
import static org.assertj.core.api.Assertions.*;
```

Implémentez ce test et exécutez-le afin qu'il passe AU VERT 😉

Remarques :

⇨ Il est important de noter que **`AssertJ`** ne remplace pas **`JUnit`**.
Vous avez absolument besoin d'utiliser **`JUnit`** pour pouvoir écrire une classe de tests et lancer vos tests unitaire. AssertJ vous fournire des services (méthodes) supplémentaires pour écrire des assertions de manière plus fluide 😉

⇨ La version `JUnit 5` : ***`assertEquals`***`("hello AssertJ !", hello);`
devient avec `AssertJ` : ***`assertThat`***`(hello).isEqualTo("hello AssertJ !");`

Pour la comparaison simple d'un objet, utiliser `JUnit5` ou `AssertJ`, la différence entre **`assertEquals`** et `assertThat` peut paraître minime dans cette exemple. Toutefois utiliser `assertThat` permet d'écrire un code « plus sûr » dans le sens où il permet de comprendre directement que le premier paramètre correspond au résultat obtenu et le deuxième paramètre au réstultat attendu. Quand vous écrivez un test avec `assertEquals`, n'avez-vous pas un doute si le premier paramètre à passer doit être le résultat attendu ou à le résultat obtenu ?
Par contre, le framework **`AssertJ`** va devenir une vraie « pépite » lorsque vous devrez écrire des tests sur des collections de manière fluide comme nous allons le voir…

❑ `AssertJ` et les collections :

⇨ Implémentez le test suivant et exécutez-le afin de vérifier qu'il passe AU VERT 😉

```
@Test
void test_List_String() {
	List<String> list = Arrays.asList("1", "2", "3");
	assertThat(list).contains("1");
}
```

⇨ Ajoutez dans le test précédent l'assertion suivante et exécutez-le à nouveau pour vérifier qu'il passe toujours AU VERT !

```
assertThat(list).doesNotContain("5");
```

⇨ Ajoutez une à une à une chaque assertion et exécutez le test après ajout de chque assertion pour vérifier que ce test passe bien au VERT 😉

```
@Test
void test_List_String() {
	List<String> list = Arrays.asList("1", "2", "3");

	assertThat(list).contains("1");
	assertThat(list).doesNotContain("5");
	assertThat(list).isNotEmpty();
	assertThat(list).startsWith("1");
	assertThat(list).containsSequence("2", "3");
	assertThat(list).doesNotContainSequence("3", "2");
	assertThat(list).containsExactly("1", "2", "3");
}
```

⇨ En fait, lorsqu'un test contient plusieurs assertions sur le même objet, la bonne pratique avec AssertJ consiste à « chainer » ces assertions pour rendre la lecture du test plus fluid. Ecrivez donc maintenant le test suivant à la suite du précédent est exécutez-le pour vérifier qu'il passe bien AU VERT 😉

```
@Test
void test_List_String_best_practice()
{
	List<String> list = Arrays.asList("1", "2", "3");

	assertThat(list).contains("1")
		.doesNotContain("5")
		.isNotEmpty()
		.startsWith("1")
		.containsSequence("2", "3")
		.doesNotContainSequence("3", "2")
		.containsExactly("1", "2", "3");
}
```

⇨ Implémentez également le test suivant et exécutez-le pour vérifier qu'il passed AU VERT 😉

```
@Test
void test_List_Character()
{
	Character someCharacter = 'i';

	assertThat(someCharacter)
	  .isNotEqualTo('a')
	  .inUnicode()
	  .isGreaterThanOrEqualTo('b')
	  .isLowerCase();
}
```

❑ Documentation sur le site officiel de `AssertJ`

A partir de la première page du site officiel d'`AssertJ` : https://assertj.github.io/doc/

⇨ rendez-vous dans la partie **2.1 What is AssertJ Core ?** pour visualiser un bel ensemble d'exemples d'assertions assertJ autour des collections qui vous montre toute la puissance d'utilisation de ce framework **:**

```
// entry point for all assertThat methods and utility methods (e.g. entry)
import static org.assertj.core.api.Assertions.*;

// basic assertions
assertThat(frodo.getName()).isEqualTo("Frodo");
assertThat(frodo).isNotEqualTo(sauron);

// chaining string specific assertions
assertThat(frodo.getName()).startsWith("Fro")
                           .endsWith("do")
                           .isEqualToIgnoringCase("frodo");

// collection specific assertions (there are plenty more)
// in the examples below fellowshipOfTheRing is a List<TolkienCharacter>
assertThat(fellowshipOfTheRing).hasSize(9)
                               .contains(frodo, sam)
                               .doesNotContain(sauron);

// as() is used to describe the test and will be shown before the error message
assertThat(frodo.getAge()).as("check %s's age", frodo.getName()).isEqualTo(33);

// exception assertion, standard style ...
assertThatThrownBy(() -> { throw new Exception("boom!"); }).hasMessage("boom!");
// ... or BDD style
Throwable thrown = catchThrowable(() -> { throw new Exception("boom!"); });
assertThat(thrown).hasMessageContaining("boom");

// using the 'extracting' feature to check fellowshipOfTheRing character's names
assertThat(fellowshipOfTheRing).extracting(TolkienCharacter::getName)
                               .doesNotContain("Sauron", "Elrond");

// extracting multiple values at once grouped in tuples
assertThat(fellowshipOfTheRing).extracting("name", "age", "race.name")
                               .contains(tuple("Boromir", 37, "Man"),
                                         tuple("Sam", 38, "Hobbit"),
                                         tuple("Legolas", 1000, "Elf"));

// filtering a collection before asserting
assertThat(fellowshipOfTheRing).filteredOn(character -> character.getName().contains("o"))
                               .containsOnly(aragorn, frodo, legolas, boromir);

// combining filtering and extraction (yes we can)
assertThat(fellowshipOfTheRing).filteredOn(character -> character.getName().contains("o"))
                               .containsOnly(aragorn, frodo, legolas, boromir)
                               .extracting(character -> character.getRace().getName())
                               .contains("Hobbit", "Elf", "Man");

// and many more assertions: iterable, stream, array, map, dates, path, file, numbers, predicate, optional ...
```

⇨ **Les parties 2.5 Core Assertion Guide et 2.6 Extending Assertions** expliquent à l'aide de nombreux exemples les possibilités offertes par ce framework. Laissez toujours un onglet sur votre navigateur ouvert sur cette partie (**https://assertj.github.io/doc/#assertj-core-assertions-guide**) qui pourra grandement vous aider dans l'écriture de vos tests AssertJ 😉

⇨ sinon, comme indiqué au début de la partie **2.AssertJ**, la javadoc dassertj-core est consultable à l'adresse suivante : https://www.javadoc.io/doc/org.assertj/assertj-core
(un conseil, laissez également cet onglet ouvert 😉)

❑ Zoom sur AssertJ et l'extraction des propriétés des objets d'une collection

Les méthodes **`extracting`** (et **`flatExtracting`**) permettent d'extraire une ou plusieurs propriétés (attributs) des objets d'une collection soit via une fonction soit via le nom de la propriété.
En effet, face à une liste d'objet complexe, il est parfois plus simple de tester un sous-ensemble des propriétés des objets de la liste plutôt que les objets en entier.

⇨ Avant d'écrire un nouveau test, nous allons ajouter du code de production. Dans **`src/main/java`**, ajoutez donc les classes suivantes :

```
public enum Sport {
	Handball,Judo;
}

public class SportMan {
	private final String firstName;
	private final String lastName;
	private final Sport sport;

	public SportMan(String firstName, String lastName, Sport sport) {
		this.firstName = firstName;
		this.lastName = lastName;
		this.sport = sport;
	}

	public String whoAmI() {
		return this.firstName + " " + this.lastName + " - " + this.sport;

	}

	public String getFirstName() {
		return firstName;
	}

	public String getLastName() {
		return lastName;
	}

	public Sport getSport() {
		return sport;
	}
	// ne pas oublier de générer hashcode et equals 😉
}
```

⇨ Revenir dans le code de test (**`src/test/java`**) et ajoutez une classe de test **`SportManTest`**.
Récupérez le contenu de la classe **`SportManTest`** à partir du gist suivant : https://unil.im/testassertj
(URL réelle : https://gist.github.com/iblasquez/ad3a030b508f534e32c8babc358bba87)

Exécutez la classe **`SportManTest`** pour vérifier que tous les tests passent AU VERT 😉
Prendre le temps de lire les code de la classe **`SportManTest`** et de comprendre les tests écrits, en vous aidant si nécessaire de la documentation donné par le site officiel de **`AssertJ`** présentée précédemment. Vous pouvez modifier les tests ou en ajouter d'autres si le cœur vous en dit 😉

❑ Un petit exercice du tutoriel en ligne de Vogella …

Rendez-vous maintenant sur le tutoriel de Vogella sur AssertJ qui est disponible à l'adresse suivante :
https://www.vogella.com/tutorials/AssertJ/article.html

⇨ Un peu de documentation …

Lire les parties 1 à 4

1. Introduction to AssertJ
2. Using AssertJ
3. IDE configuration
4. Usage of AssertJ

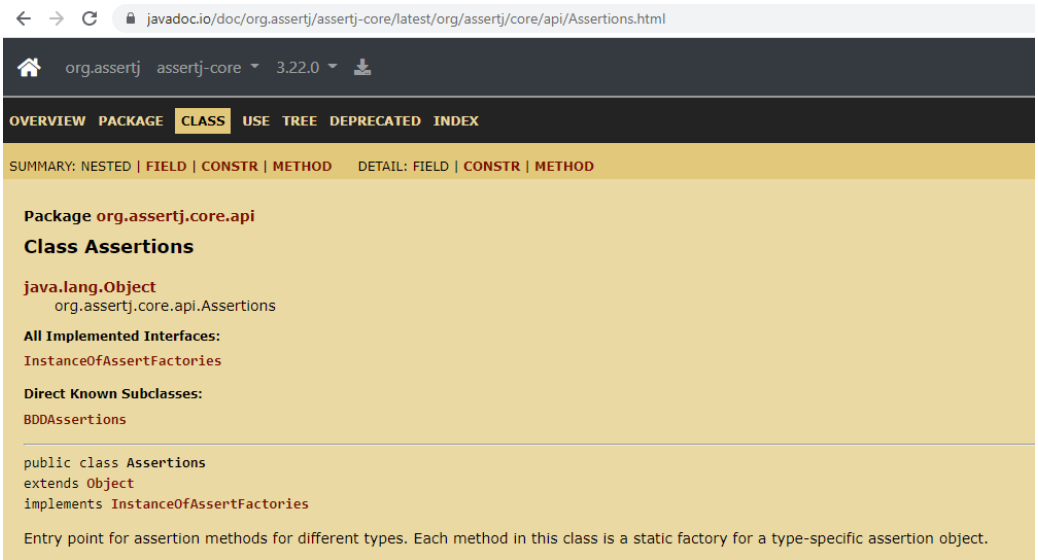
⇨ Rendez-vous dans la partie 5 : Using AssertJ matchers in unit tests

✓ Dans le projet **`helloassertJ`**, créez maintenant dans **le code de test** (`src/test/java`), une classe **`AssertJExamplesTests`** dans laquelle vous copie-collerez le code de la même de la classe de même nom dans la partie 5 du tutoriel de Vogella :
https://www.vogella.com/tutorials/AssertJ/article.html#using-assertj-matchers-in-unit-tests

✓ Exécutez le fichier, et assurez-vous que tous les tests échouent et sont AU ROUGE !!!

✓ Ajoutez en tête de fichier un import à tous les service d'**`AssertJ`** :

```
import static org.assertj.core.api.Assertions.*;
```



⇒ assurez-vous bien d'avoir à porter de main la javadoc de la classe Asserions du paquetage … pour savoir quelle méthode vous pouvez utiliser 😉 :
https://www.javadoc.io/doc/org.assertj/assertj-core/latest/org/assertj/core/api/Assertions.html

✓ En utilisant les services offerts par AssertJ, faites passer AU VERT les test UN à UN, de manière à ce que tous les tests de ce fichier de tests passent AU VERT !
Le contenu du test que vous devez écrire se trouve bien évidemment dans l'annotation `@Display` de chaque test,
Indice : pour la méthode de test `everyItemGreaterThan1` allez voir du côté `allMatch` 😉

✓ Une fois que tous vos tests passent AU VERT, vous pouvez améliorer la qualité de code au niveau de vos imports en utilisant l'IDE et notamment sous Eclipse le menu suivant :
`Source → Organize Imports` qui va réorganiser vos imports pour ne laisser que les import nécessaires au bon fonctionnement du projet. En effet, si vous remontez au début de votre fichier de test, vous ne verrez plus : `import static org.assertj.core.api.Assertions.*;`
qui importe tout avec l'*, mais vous n'importerez plus que ce qui est nécessaire au bon fonctionnement de votre code.

Tips « Bonne pratique » :

N'hésitez pas à faire des `Organize Imports` pour améliorer la qualité de votre code 😉

⇨ Retour vers `Tolkien` …

✓ Dans le projet **`helloassertJ :`**
→ dans **le code de production** (`src/main/java`), un paquetage **`helloassertj.tolkien`**
Dans lequel , vous ajouterez l'énumération **`Race`** et la classe **`Tolkien`** disponible dans le gist suivant : https://unil.im/tolkien
(https://gist.github.com/iblasquez/8bdd3e28a08ae3e2dfec6c84cab156e5)
→ dans **le code de test** (`src/test/java`), dans un paquetage **`helloassertj.tolkien,`**
ajoutez le fichier test **TolkienChecksWithAssertJTests** également disponible dans le même gist que précédemment. Exécutez ce fichier de test afin de constater que les tests échouent biien 😉

✓ En utilisant les services offerts par **AssertJ**, faites passer AU VERT les test UN à UN, de manière à ce que tous les tests de ce fichier de tests passent AU VERT !

✓ Une fois que tous vos tests passent AU VERT, améliorez la qualité de code de votre fichier test en organisant les import afin de garder uniquement les imports nécessaire au bon fonctionnement du code.

❑ Pour en savoir plus sur `AssertJ` ….

→ **Site officiel :**
https://assertj.github.io/doc/
https://www.javadoc.io/doc/org.assertj/assertj-core
→ **Tutoriels :**
https://www.baeldung.com/introduction-to-assertj
https://www.vogella.com/tutorials/AssertJ/article.html
https://www.petrikainulainen.net/programming/testing/junit-5-tutorial-writing-assertions-with-assertj/
https://pieces-of-code.com/guide/quickstart/assertj.htm (en français)