1. Rappels généraux

Tous les TD et TP se dérouleront sur **Linux**.

Chaque semaine vous allez devoir rendre une partie de votre travail sur Moodle. Ça sera généralement la partie centrale d'un projet. Ces documents seront évalués dans le cadre de la note de TP. Ainsi, il n'y aura pas un TP noté final, mais de multiples travaux à faire dans le cadre d'un contrôle continu. Les TP ne doivent pas être fait à la maison, seulement pendant les séances, en présence d'un enseignant.

Ce système de notation repose sur l'honnêteté collective. Il vous est interdit de copier des sources entre vous. Si des tricheries sont constatées, on sera obligé de revenir à des devoirs sur table et des contrôles de TP sur table également. À vous de voir ce que vous préférez.

Cette semaine, on va apprendre à utiliser AssertJ, une très bonne librairie pour écrire des tests unitaires en Java.

2. Découverte de AssertJ

2.1. Présentation

Dans cette partie, on va apprendre à écrire des assertions avec AssertJ qui repose sur JUnit5. Le but des assertions est de vérifier que telle variable contient la valeur attendue, ou se conforme à une propriété voulue. Dans ce cours, AssertJ a été choisi contre Hamcrest et Truth parce qu'il est à la fois plus simple et plus complet. Le problème principal de Hamcrest, c'est le typage générique des *matchers*, très complexe en Java. Le problème de Truth, c'est le manque de documentation et la non-fluidité des assertions.

Pour vous convaincre que AssertJ écrabouille la concurrence en ce qui concerne la documentation et qu'il est tout à fait complet, consultez ce projet AssertJ sur GitHub qui illustre AssertJ sur un très grand nombre d'exemples. Consultez par exemple ces exemples de base et ces exemples sur les tableaux.

Une autre grande qualité de AssertJ est la clarté des messages d'erreurs. Voici un exemple :

```
assertThat(23).isCloseTo(20, withinPercentage(5));
```

Il affiche ceci, qui est extrêmement clair et suggère même la valeur qui ferait réussir le test :

```
java.lang.AssertionError:
Expecting actual:
  23
to be close to:
  20
by less than 5% but difference was 15.0%.
(a difference of exactly 5% being considered valid)
```

Le principe général de AssertJ est d'écrire des instructions commençant par `assertThat(résultat)` suivies d'une chaîne d'affirmations sur ce résultat. Par exemple, le résultat est égal à ceci, est proche de cela, contient ces choses, etc. Les affirmations dépendent du type du résultat : type simple, objet, tableau, etc. et sont détaillées dans la suite.

2.2. Mise en œuvre

Pour fixer les idées, on passe à l'action. Il y a une configuration et des importations à faire.

1. Créez un nouveau projet Maven dans le *workspace* Eclipse :
   - GroupID : `fr.iutlan.ql`
   - ArtifactID : `tp2`
2. Remplacez le fichier `pom.xml` par ce fichier pom.xml (enregistrer la cible sous... et changez le nom). Il est plus simple que celui du TP1, car on ne créera pas de fichier *jar*.
3. Mettez à jour le projet Maven.

Normalement, le projet doit être bien reconnu par Eclipse et les différents dossiers de `src`, `main` et `test` bien créés.

4. Dans le dossier `src/main/java`, créez le *package* `fr.iutlan.ql.tp2` et une classe nommée `DataProvider`.
5. Remplacez cette classe par DataProvider.java (enregistrer la cible sous...).
6. Dans le dossier `src/test/java`, créez le *package* `fr.iutlan.ql.tp2` et cette classe `TestsAssertions` :

```java
package fr.iutlan.ql.tp2;

import static org.assertj.core.api.Assertions.*;
import static org.junit.jupiter.api.Assertions.*;
import org.assertj.core.data.*;
import org.junit.jupiter.api.*;

public class TestsAssertions {

    @Test
    void testEntier23egal() {
        // ARRANGE
        int nombre = DataProvider.getVingtTrois();

        // ASSERT
        assertThat(nombre).isEqualTo(23);
    }

    @Test
    void testEntier23proche() {
        // ARRANGE
        int nombre = DataProvider.getVingtTrois();

        // ASSERT
        assertThat(nombre).isCloseTo(20, withinPercentage(20));
    }

    @Test
    void testEntier23entre() {
```

```
        // ARRANGE
        int nombre = DataProvider.getVingtTrois();

        // ASSERT
        assertThat(nombre).isBetween(20, 30);
    }
}
```

6. Lancez l'exécution de la classe de test en tant que « JUnit Test ». Vous devez voir le résultat en vert, 0 failures.
7. Dans `DataProvider`, changez la valeur retournée par `getVingtTrois()` en 24, réessayez les tests (1 échec), puis 28 (2 échecs), puis 35 (3 échecs). C'est ainsi que votre travail sera noté, en changeant les valeurs retournées par `DataProvider` et en vérifiant que vos tests sont conformes.

Dans un vrai projet, si on a besoin de vérifier plusieurs propriétés simultanément, on peut regrouper les assertions ainsi :

```
    @Test
    void testsMultiplesEntier38() {
        // ARRANGE
        int nombre = DataProvider.getTrenteHuit();

        // ASSERT
        assertThat(nombre)
            .isEqualTo(38)
            .isCloseTo(40, withinPercentage(10))
            .isBetween(30, 50);
    }
```

AssertJ permet d'enchaîner les assertions de manière fluide. En Java, une API à chaînage de méthode (*fluent API*) permet d'enchaîner les méthodes dans des appels successifs : `objet.methode1(param1).methode2(param2).methode3(param3);`. C'est rendu possible parce que chacune de ces méthodes retourne `this`, c'est à dire l'objet du début.

Dans le cas de AssertJ, l'objet du début est `assertThat(valeur)`, et les méthodes visent à comparer cette valeur à quelque chose qu'on attend. L'assertion échoue quand l'une des comparaisons échoue.

3. Présentation de AssertJ

Voici une petite liste des méthodes que propose AssertJ.

La documentation complète est sur cette page. Elle est assez volumineuse, mais très claire, il faut un peu chercher pour trouver ce qu'on veut. Vous avez aussi une gigantesque collection d'exemples rangés par catégories.

Une première chose à savoir, c'est qu'il faut utiliser la complétion de code. Tapez `assertThat(chose).` et l'éditeur vous propose tout ce qu'on peut mettre après. Ajoutez quelques lettres de plus, et ça sera filtré par ces lettres.

3.1. Assertions sur des nombres

3.1.1. Validité

Pour les `float` et `double` :

- `assertThat(nombre).isNaN()`
  NaN veut dire *not a number*. C'est un symbole qui signale une erreur avec la valeur.
- `assertThat(nombre).isNotNaN()` vraie si le nombre est valide
- `assertThat(nombre).isInfinite()`
- `assertThat(nombre).isNotInfinite()` vraie si le nombre n'est pas l'infini

3.1.2. Égalité stricte

- `assertThat(nombre1).isEqualTo(nombre2)`
- `assertThat(nombre1).isIn(nombre2, nombre3, nombre4, ...)`
- `assertThat(nombre1).isIn(iterable du même type)`

3.1.3. Égalité proche

Il y a plusieurs méthodes similaires dont celles-ci :

- `assertThat(nombre1).isCloseTo(nombre2, byLessThan(tolerance))`
  vraie si $|nombre1 - nombre2| < tolerance$
- `assertThat(nombre1).isCloseTo(nombre2, within(tolerance))`
  vraie si $|nombre1 - nombre2| \leq tolerance$
- `assertThat(nombre1).isCloseTo(nombre2, withinPercentage(tolerance))`
  vraie si $|nombre1 - nombre2| \leq tolerance$ *en* %

3.1.4. Signe

- `assertThat(nombre1).isNotZero()`
- `assertThat(nombre1).isPositive()` vraie si $nombre1 > 0$
- `assertThat(nombre1).isNotPositive()` vraie si $nombre1 \leq 0$
- `assertThat(nombre1).isNegative()` vraie si $nombre1 < 0$
- `assertThat(nombre1).isNotNegative()` vraie si $nombre1 \geq 0$

3.1.5. Bornes

- `assertThat(nombre1).isBetween(nombre2, nombre3)`
- `assertThat(nombre1).isLessThan(nombre2)`
- `assertThat(nombre1).isLessThanOrEqualTo(nombre2)`
- `assertThat(nombre1).isGreaterThanOrEqualTo(nombre2)`
- `assertThat(nombre1).isGreaterThan(nombre2)`

3.1.6. Prédicat

Un prédicat Java est une lambda ou une méthode qui prend un paramètre du même type que le nombre et qui retourne un booléen. Par exemple, `n -> (n < 10)` et `n -> Math.abs(n) > 10`

sont des prédicats écrits sous forme de lambda. Et `public boolean estPair(int n) { return n % 2 == 0; }` est un prédicat écrit sous la forme d'une méthode.

- `assertThat(nombre1).matches(this::estPair, "le nombre doit être pair")` vraie si le prédicat (version *methode*) retourne `true`, sinon la description est affichée et le test échoue. L'écriture `this::methode` est appelée une référence de méthode. C'est une sorte d'objet mais de type méthode, qu'on peut passer comme un paramètre.
- `assertThat(nombre1).matches(n -> n % 2 == 0, "le nombre doit être pair")` vraie si le prédicat (version *lambda*) retourne `true`, et la description est affichée dans le cas contraire. La lambda est une référence de méthode mais anonyme.

3.2. Assertions sur des chaînes

Voir la documentation. Il y a une multitude de méthodes dont voici un petit échantillon.

3.2.1. Caractéristiques

- `assertThat(chaine).isNullOrEmpty()`
- `assertThat(chaine).isEmpty()`
- `assertThat(chaine).isNotEmpty()`
- `assertThat(chaine).hasSize(longueur)`
- `assertThat(chaine).hasSizeBetween(longueur1, longueur2)`
- `assertThat(chaine).isBlank()`
- `assertThat(chaine).isNotBlank()`
- `assertThat(chaine).isLowerCase()`
- `assertThat(chaine).isUpperCase()`

3.2.2. Contenu

- `assertThat(chaine).startsWith(souschaine)`
- `assertThat(chaine).endsWith(souschaine)`
- `assertThat(chaine).contains(souschaine1, souschaine2, ...)`
- `assertThat(chaine).containsAnyOf(souschaine1, souschaine2, ...)`
- `assertThat(chaine).doesNotContain(souschaine1, souschaine2, ...)`
- `assertThat(chaine).hasLineCount(nombre)` dans le cas où la chaîne contient plusieurs lignes séparées par des `\n`
- `assertThat(chaine).matches(expression régulière)` se souvenir de grep pour les écrire, ou consulter ce tutoriel.

3.3. Assertions sur des objets

3.3.1. Nullité

La variable `chose` est comparée à `null`.

- `assertThat(chose).isNull()`
- `assertThat(chose).isNotNull()`

3.3.2. Classe et variables membres

- `assertThat(chose).isInstanceOf(Classe1.class)`
  vraie si `chose` est de la classe `Classe1` ou d'une de ses sous-classes
- `assertThat(chose).isExactlyInstanceOf(Classe1.class)`
  vraie si `chose` est directement de la classe `Classe1`
- `assertThat(chose).hasFieldOrProperty("nom")`
  vraie si `chose` possède une variable membre `nom` (*field* = variable privée ou protégée, *property* = variable publique avec accesseurs).

3.3.3. Égalité

- `assertThat(chose1).isSameAs(chose2)`
  vraie si *chose1* est physiquement le même objet que *chose2*, par exemple quand on a affecté une variable avec la valeur d'une autre. Le test se fait en comparant les adresses en mémoire des deux objets, donc sans passer par aucune de leurs méthodes.
- `assertThat(chose1).isEqualTo(chose2)`
  vraie si l'appel Java `chose1.equals(chose2)` retourne `true`. Cette assertion utilise la méthode `equals` de `chose1`. En général, cette méthode retourne `true` quand les objets représentent la même valeur. Souvent le test est fait en comparant les *hachages* des objets (un calcul rapide sur le contenu de l'objet qui donne un entier identifiant). Il faut vraiment approfondir la question pour savoir quelle comparaison est faite.
- `assertThat(chose1).usingRecursiveComparison().isEqualTo(chose2)`
  vraie si *chose1* possède les mêmes valeurs membres que *chose2* en comparant toutes les variables membres y compris celles des objets inclus.
- `assertThat(chose1).usingRecursiveComparison()`
  `.ignoringFields("membre1",...).isEqualTo(chose2)`
  vraie si *chose1* possède les mêmes valeurs membres que *chose2* sans considérer les variables membres indiquées dans l'énumération.
- `assertThat(chose1).usingRecursiveComparison()`
  `.ignoringActualNullFields().isEqualTo(chose2)`
  vraie si les membres non null de chose1 (*actual*) sont égaux à ceux de chose2 (*expected*)
- `assertThat(chose1).usingRecursiveComparison()`
  `.ignoringExpectedNullFields().isEqualTo(chose2)`
  vraie si les membres non null de chose2 (*expected*) sont égaux à ceux de chose1 (*actual*)

Il y a de très nombreuses possibilités `comparingOnlyFields(...)`, `withComparatorForFields(cmp, ...)`. Consulter la documentation.

3.4. Assertions sur des collections

La documentation présente plusieurs catégories d'assertions sur les collections (tableaux, listes, ensembles, itérables...)

3.4.1. Cardinalité

- `assertThat(collection).isEmpty()`
- `assertThat(collection).isNotEmpty()`
- `assertThat(collection).hasSize(longueur)`

3.4.2. Contenu

- `assertThat(collection).containsNull()`
  vraie si la collection contient la valeur `null`
- `assertThat(collection).doesNotContainNull()`
  vraie si la collection ne contient pas la valeur `null`
- `assertThat(collection).contains(valeur1, valeur2, ...)`
  vraie si la collection contient au moins une fois chaque valeur indiquée
- `assertThat(collection).doesNotContain(valeur1, valeur2, ...)`
  vraie si la collection contient ne contient aucune des valeurs indiquées
- `assertThat(collection).containsOnly(valeur1, valeur2, ...)`
  vraie si la collection contient toutes et uniquement les valeurs indiquées, dans n'importe quel ordre et éventuellement en plusieurs exemplaires
- `assertThat(collection).containsOnlyOnce(valeur1, valeur2, ...)`
  vraie si la collection contient exactement les valeurs indiquées une seule fois et rien d'autre, mais dans un ordre quelconque
- `assertThat(collection).containsExactly(valeur1, valeur2, ...)`
  vraie si la collection contient exactement les valeurs indiquées dans cet ordre et rien d'autre
- `assertThat(collection).containsExactlyInAnyOrder(valeur1, valeur2, ...)`
  vraie si la collection contient exactement les valeurs indiquées mais dans n'importe quel ordre, et rien d'autre

La dernière est une variante de `containsOnlyOnce`.

3.4.3. Assertions sur les éléments isolés

Des assertions peuvent être déposées sur des éléments précis, désignés par leur index.

- `assertThat(collection).first().`*suite*
  applique la suite sur le premier élément de la liste
- `assertThat(collection).last().`*suite*
  applique la suite sur le dernier élément de la liste
- `assertThat(collection).element(index).`*suite*
  applique la suite sur l'élément n°index, les indices commencent à 0.

Par exemple, `assertThat(liste).element(1).isNotNull();` est vraie si le 2e élément de la liste n'est pas `null`.

3.4.4. Assertions sur tous les éléments

- `assertThat(collection).allSatisfy(elem -> { assertions sur elem })`
  vraie si les assertions sont vraies sur chacun des éléments de la collection
- `assertThat(collection).anySatisfy(elem -> { assertions sur elem })`
  vraie si les assertions sont vraies sur au moins un des éléments de la collection
- `assertThat(collection).noneSatisfy(elem -> { assertions sur elem })`
  vraie si les assertions ne sont jamais vraies sur les éléments de la collection

Par exemple, `assertThat(collection).anySatisfy(elem -> assertThat(elem).isNotNull());` est vraie s'il y a au moins un élément différent de `null` dans la liste.

3.4.5. Assertions sur une partie de la collection

Pour limiter les assertions à une partie de la liste définie par un prédicat :

- `assertThat(liste).filteredOn(elem -> condition sur elem)`
  `.xxxSatisfy(elem -> assertions sur elem)`
  on peut utiliser `allSatisfy`, `anySatisfy`, `noneSatisfy`.

Attention, `filteredOn` demande une condition, c'est à dire une simple comparaison, pas une assertion.

3.5. Assertions sur les fichiers

Les assertions suivantes utilisent AssertJ et demandent les importations suivantes :

```
import org.assertj.core.util.Files;
import java.io.File;
import java.nio.charset.Charset;
import java.nio.charset.StandardCharsets;
```

3.5.1. Tester l'existence ou l'absence d'un fichier

Voici comment vérifier qu'un fichier existe ou pas :

```
File file1 = new File("je_dois_exister.txt");
assertThat(file1).exists();

File file2 = new File("/non.non.non");
assertThat(file2).doesNotExist();
```

3.5.2. Vérifier la nature d'un fichier

En Java, tout élément d'un système de fichiers (*file system*) est représenté par une instance de `File`. Ça peut être un fichier, un dossier, ou un fichier spécial (lien, tube nommé, etc).

```
File file = new File("chose");
assertThat(file).isDirectory();
assertThat(file).isFile();
```

3.5.3. Vérifier qu'on peut lire ou écrire un fichier

Voici les assertions AssertJ :

```
File file = new File("fichier.txt");
assertThat(file).isReadable();
assertThat(file).isWritable();
```

3.5.4. Vérifier la taille d'un fichier

La taille d'un fichier est spécifiée par un entier long (suffixe `L` après les chiffres) :

```
File file = new File("fichier.txt");
long taille_attendue = ...L;        // mettre un long
assertThat(file).size().isEqualTo(taille_attendue);
```

La méthode `size()` appliquée à un fichier permet d'enchaîner des assertions sur des entiers longs.

3.5.5. Vérifier le contenu d'un fichier

Si le fichier n'est pas trop gros et qu'on peut mettre son contenu dans une chaîne : 

```
File file = new File("je_dois_contenir_ceci.txt");
assertThat(file).usingCharset(StandardCharsets.UTF_8).hasContent("ceci");
assertThat(file).content(StandardCharsets.UTF_8).startsWith("ce").endsWith("ci");
```

La dernière assertion, `assertThat(file).content(StandardCharsets.UTF_8)` est très générale. On peut enchaîner toutes les assertions voulues sur le contenu en tant que chaîne.

Si le fichier est plus gros, alors il faut lire son contenu morceau par morceau et vérifier chaque morceau. Par exemple, lire ligne par ligne : 

```
import org.assertj.core.util.Files;

File file = new File("je_dois_contenir_tout_ca.txt");
for (String line: Files.linesOf(file, StandardCharsets.UTF_8)) {
    assertThat(line).hasSizeBetween(5, 40).startsWith("Data:");
}
```

On peut aussi écrire des assertions globales sur un fichier : 

```
assertThat(Files.linesOf(file, StandardCharsets.UTF_8))
    .element(2).startsWith("Informations");
assertThat(Files.linesOf(file, StandardCharsets.UTF_8)).noneSatisfy(
    line -> assertThat(line).isEmpty()      // lambda appliquée sur chaque ligne
);
```

3.5.6. Supprimer un fichier

La classe `File` de Java possède de nombreuses méthodes, dont `delete()` : 

```
File file = new File("a_supprimer.txt");
file.delete();
```

4. Exercices

Après avoir vu tout AssertJ, on revient à l'écriture de tests unitaires.

👉 Téléchargez TestsAssertions.java et mettez-le à la place de `TestsAssertions.java` côté tests. C'est un fichier que vous devez compléter.

ATTENTION: veillez à ne pas changer les noms des packages, des classes et des méthodes. Tout travail de *refactoring* que vous imposez en ne respectant pas les consignes vous coûtera des points.

4.0.1. Assertions sur des nombres

☛ Complétez les méthodes de test pour vérifier ces informations :

- `testEntierM28negatif` : vérifie que -28 est négatif.
- `testEntierM28proche` : vérifie que -28 est proche de -30 à 2 près.
- `testEntier198multiple9` : vérifie que 198 est multiple de 9 à l'aide d'un prédicat. Comment faire ? Il faut penser à la division euclidienne (entière) de 198 par 9, et regarder si le reste est nul. Le reste d'une division par 9 est obtenu directement par n % 9. Donc le prédicat s'écrit `n -> n%9 == 0`.
- `testEntier198dansListe` : vérifie que 198 fait partie de la liste {23, 127, 198, 223}.
- `testReelM314valide` : vérifie que -3.14 est un nombre valide (pas `NaN` qui veut dire *not a number*).
- `testReelM314entre` : vérifie que -3.14 est compris entre -3 et -4 (piège),
- `testReelM314nonEntier` : vérifie que -3.14 est un nombre non entier. C'est à dire qu'il y a des chiffres non nuls après la virgule. Comment faire ce dernier test ? Un nombre réel comme 2.0 possède une partie entière égale à lui-même. La partie entière de n est `Math.floor(n)`. Il suffit donc de comparer n et sa partie entière pour savoir si n est non-entier.
- `testInfini` : vérifie que le nombre est infini.

4.0.2. Assertions sur des chaînes

☛ Complétez les méthodes de test pour vérifier ces informations :

- `testBonjourLongueur7` : vérifie que la chaîne a une longueur de 7.
- `testBonjourContientUr` : vérifie que la chaîne contient `"ur"`,
- `testBonjourContientOuEtOn` : vérifie que la chaîne contient `"ou"` et `"on"`.
- `testBonjourContientOiOuOn` : vérifie que la chaîne contient `"oi"` ou `"on"` (au moins l'un d'eux).
- `testBonjourContientNiOiNiOp` : vérifie que la chaîne ne contient ni `"oi"` ni `"op"`.
- `testBonjourContientAucunEspace` : vérifie que la chaîne ne contient pas d'espaces. Cherchez dans la documentation la méthode qui exprime cela.
- `testBonjourCommenceMajuscule` : vérifie que la chaîne commence par une majuscule. Il faut regarder si la chaîne correspond à l'expression régulière `"^[A-Z].*"` (attention, retapez ce motif de zéro, ne le copiez pas du pdf, car le caractère `^` est différent). Lisez la documentation.

4.0.3. Assertions sur des objets

☛ Complétez les méthodes de test pour vérifier ces informations :

- `testPersonneP1classe` : vérifie que la variable `p1` est une instance de `Personne` ou d'une sous-classe.
- `testPersonneP1classestricte` : vérifie que la variable `p1` est une instance de `Personne` mais pas d'une sous-classe.
- `testPersonneP1pasnull` : vérifie que la variable `p1` ne vaut pas`null`.
- `testPersonneP1diffP2\` : vérifie que la variable `p1` n'est pas égal, en tant qu'objet, à `p2` (que ce sont deux instances distinctes).
- `testPersonneP1commeP2` : vérifie que la variable `p1` possède les mêmes valeurs de champs (variables membres) que `p2`. Il faut utiliser une comparaison récursive.

- `testPersonneP1P3memeprenom` : vérifie que les variables `p1` et `p3` représentent la même chose si on considère leurs variables membres sauf le nom.
- `testPersonneP1commeP4` : vérifie que `p1` et `p4` représentent la même chose si on considère leurs variables membres qui ne sont pas nulles côté `p4` (valeurs « attendue »).
- `testPersonneP4commeP5` : vérifie que `p4` et `p5` représentent la même chose si on considère leurs variables membres qui ne sont pas nulles (valeurs « attendue » et « observée »).

4.0.4. Assertions sur des collections

☛ Complétez les méthodes de test pour vérifier ces informations :

- `testListeNonVide` : vérifie que la liste n'est pas vide.
- `testListeContientTout` : vérifie que la liste contient la chaîne `"tout"`.
- `testListe4eElement` : vérifie que le 4e élément de la liste est la chaîne `"le"`.
- `testListeContientNull` : vérifie que la liste contient au moins une valeur `null`.
- `testListeNeContientPasVide` : vérifie que la liste ne contient pas de chaîne vide.
- `testListeNeContientPasQueNull` : vérifie que la liste ne contient pas que des `null`.
- `testListePasChainesPlusLonguesQue8` : vérifie que la liste ne contient pas de chaînes plus longues que 8 caractères. Il y a plusieurs difficultés :
  - D'abord, `assertThat(liste).hasSizeLessThanOrEqualTo(8)` ne vérifie pas la longueur des éléments, mais la longueur de la liste. Donc il faut employer un opérateur comme `allSatisfy` pour appliquer une assertion sur chacun des éléments de la liste.
  - Mais la liste contient un `null`, et donc l'assertion `hasSizeLessThanOrEqualTo` échoue sur cet élément. Il faut préalablement filtrer la liste pour ne garder que les non-nulls. Consultez la partie théorique pour voir comment appliquer une assertion sur une partie de la liste.
  - Et il y a un autre problème. La liste, étant une `List<Object>`, peut ne pas contenir que des `String`, et donc il faut convertir chaque élément testé en chaîne avec `asString()` avant de faire le test `hasSizeLessThanOrEqualTo`.

☛ Vérifiez ce dernier test en allongeant progressivement l'un des mots à plus de 8 caractères dans `DataProvider.getListe()`.

- `testListeContientUniquementDesStrings` : vérifie que liste ne contient que des instances de `String` ou `null`.
  - Il y a sûrement plusieurs façons de faire ça. Une suggestion est de vérifier que tous les éléments satisfont une *lambda*.
  - Cette *lambda* est une assertion qui exprime qu'un élément est soit une instance de `String`, soit `null`. C'est cette *lambda* qui est difficile à écrire. Elle doit vérifier une disjonction. Regardez la documentation de `satisfiesAnyOf`. On lui fournit une liste de plusieurs autres *lambda* exprimant des assertions dont une au moins doit réussir. La difficulté est donc qu'il y a deux *lambda* imbriquées dans une troisième. Aidez-vous de cette réponse sur StackOverflow.
  - Une autre manière consiste à affirmer qu'il ne doit pas y avoir autre chose que des `String` dans la liste, sachant que la méthode `isNotInstanceOf` échoue pour la valeur `null`.

☛ Vérifiez ce dernier test en ajoutant un entier à la liste dans `DataProvider.getListe()`.

- `testListeNeContientPasDeDoublons` : vérifie que la liste ne contient pas d'élément en

double (ou plus). Vous devrez chercher dans la documentation une assertion qui interdit les doublons. Comment traduit-on *doublon* en anglais ?

- `testListeElementsParmi` : vérifie que la liste est uniquement composée d'éléments parmi {"le", "monde", 17, "me", "dit", 23, "bonjour", "tout", "le", "temps", null}. Une méthode citée dans la documentation permet de faire le travail très facilement, mais il faut comprendre comment on la détourne pour faire cette vérification.

4.0.5. Assertions sur des fichiers

☛ Complétez les méthodes de test pour vérifier ces informations :

- `testFichierSansException` : vérifie qu'on peut récupérer le nom d'un fichier qui est créé s'il n'existe pas. Cette méthode est fournie toute faite, car un peu spéciale.
- `testFichierEstUnFichierLisible` : vérifie que la chose dont on récupère le nom est bien un fichier et qu'il est lisible.
- `testFichierPasVide` : vérifie que le fichier n'est pas vide.
- `testFichierContientEntre3et6lignes` : vérifie que le fichier contient entre 3 et 6 lignes.
- `testFichierContientLignesCommençantParLigne` : vérifie que le fichier contient uniquement des lignes qui commencent par `"ligne "`.
- `testFichierContientLignesLigneNombre` : vérifie que le fichier contient uniquement des lignes qui correspondent au patron "ligne <chiffres...>". Vous devrez consulter la documentation de la classe `Pattern`. Cette classe permet de construire une expression régulière par `Pattern.compile("expression")`. Cette méthode retourne une instance de `Pattern` qu'on fournit à une assertion AssertJ, à vous de trouver laquelle. L'expression régulière doit exprimer que les lignes commencent par `"ligne "` suivi d'un ou plusieurs chiffres, au moins un.

5. Rendu du TP

Vous devrez remettre votre travail à la fin de chaque séance. C'est ainsi que c'est noté, en contrôle continu.

Ouvrez un terminal bash dans le dossier de votre TP, là où il y a le fichier pom.xml, puis tapez ceci :

```
mvn clean
cd ..
tar cfvz tp2.tgz tp2
```

Puis déposez le fichier `tp2.tgz` dans la zone de dépôt du TP2 sur Moodle R4.02 Qualité de développement.

ATTENTION votre travail est personnel. Si vous copiez pour quelque raison que ce soit le travail d'un autre, vous serez tous les deux pénalisés par un zéro.

En cas de souci quelconque, envoyez un mail à `pierre.nerzic@univ-rennes1.fr` pour expliquer le problème.