R4.02 – Qualité Logicielle (JUnit 5)

Devoir sur table

1. Question de cours (4 points)

Question 1 : Quelle est la différence entre plusieurs tests unitaires dans un même `TestCase` en JUnit 4 et plusieurs tests dans une méthode `assertAll` en JUnit 5 ?

Question 2 : Quelle est la différence entre les méthodes :

1. `assertTimeout(Duration.ofSeconds(1), () -> {f.compute(42);})`
2. `assertTimeoutPreemptively(Duration.ofMillis(1), () -> {f.compute(42);})`

Question 3 : Sachant que `IllegalArgumentException` est la super classe de `NumberFormatException`, que se passe-t-il avec cette assertion ?

```
assertThrows(IllegalArgumentException.class, () -> {
	Integer.parseInt("One");
});
```

Question 4 : Sachant que `IllegalFormatException` est une class sœur de `IllegalArgumentException`, que se passe-t-il avec cette assertion ?

```
assertThrows(IllegalFormatException.class, () -> {
	Integer.parseInt("One");
});
```

2. Tests en JUnit 5 d'une classe MyString (10 points)

Soit le début d'une classe `MyString` donnée à la page suivante, écrire les tests unitaires et le code java des fonctionnalités suivantes :

1. Convertir tous les caractères de la chaîne en majuscule,
2. Obtenir le nombre d'occurrence d'un caractère donné de la chaîne,
3. Obtenir le nombre de voyelles (le Y sera considéré comme consonne) de la chaîne,
4. Obtenir le premier mot de la chaîne.

Consignes :

- Faire précéder chacune des méthodes de test par une balise `@DisplayName` décrivant par une phrase l'intention du test ou du groupe de tests unitaires.
- Si vous avez besoin d'un groupe de tests, utilisez une méthode `assertAll`.
- Il sera noté la cohérence entre vos tests et votre code java, puisque c'est vous qui décidez du comportement « détaillé » de chacune des 4 méthodes.

```java
public class MyString {

    private StringBuffer sb;
    public MyString(String s) {
      this.sb = new StringBuffer(s); }
    public String getString() {
      return this.sb.toString(); }
    public void setString(String s) {
      this.sb = new StringBuffer(s); }
}
```

3. AssertIterableEquals sur une file (6 points)

Il s'agit de tester l'itérateur d'une file d'éléments générique. Une file est composée de cellules d'une référence sur la première et dernière cellule de la file. Ci-dessous, le squelette de la classe File et de son *inner class* implémentant un itérateur :

```java
public class File<E> implements Iterable<E> {

	private Cellule premier;
	private Cellule dernier;

	private class Cellule {
		private E element;
		private Cellule suivant;
		public Cellule(E element, Cellule suivant) {
			this.element = element;
			this.suivant = suivant;
		}
	}

	public File() {…}
	public void enfiler(E element) {…}
	private class MyIterateurFile implements Iterator<E> {
		…
		public boolean hasNext() {…}
		public E next() {…}
	}

	public Iterator<E> iterator() { return new MyIterateurFile(); }
}
```

Question 1 : écrire un test unitaire en JUnit 5 montrant que l'itérateur fonctionne.

Question 2 : en vous inspirant de ce code, sauriez-vous écrire l'itérateur ?

```java
public int nbElements() {
	int nbElements = 0;
	Cellule curseur = this.premier;
	while (curseur != null) {
		nbElements +=1;
		curseur = curseur.suivant;
	}
	return nbElements;
}
```