Tests unitaires

CREATION DU PROJET

- Créez un projet ***demo-tests-unitaires (Java 8, Maven, GitHub)***
- Modifiez votre fichier pom.xml et ajoutez la dernière version dépendance JUnit :

```
<!-- https://mvnrepository.com/artifact/junit/junit -->
<dependency>
    <groupId>junit</groupId>
    <artifactId>junit</artifactId>
    <version>4.13.2</version>
    <scope>test</scope>
</dependency>
```

EXERCICE 1

Nous allons reprendre l'énumération Saison et testez la méthode qui permet de retrouver une Saison à partir de son libellé.

Etape 1 : créer une classe de tests unitaires appelée SaisonTest dans le même package.
Etape 2 : créer une méthode de tests unitaires avec l'annotation @Test.
Etape 3 : Réalisez un test avec un libellé existant.
Etape 4 : Réalisez un test avec un libellé inexistant.
Etape 5 : Réalisez un test avec un libellé null.

SI UN DES TESTS N'EST PAS PASSANT, METTEZ AU POINT VOTRE METHODE.

EXERCICE 2

Reprenez la classe **Maison** du TP **Java POO J3** et testez là avec une classe de tests unitaires.

Pensez à tester les cas aux limites :

- que se passe t'il si j'ajoute une pièce null à la maison ?
- que se passe t'il si une superficie de pièce est négative ?

Ajoutez des contrôles dans vos méthodes afin de rendre vos classes plus robustes.

PS : Si vous n'avez pas réalisé cet exercice, vous pouvez récupérer le corrigé sur GitHub : https://github.com/DiginamicFormation/java-poo-j3

Les classes à tester sont dans le package **fr.diginamic.maison**

COMMITEZ SUR GITHUB

EXERCICE 2

DISTANCE DE LEVENSHTEIN ET OBJECTIFS DE L'EXERCICE

L'objectif de ce TP est de tester une classe qui fournit un service très utile connu sous le nom de « **calcul de distance de Levenshtein** ».

C'est cette méthode qui permet par exemple à un moteur de recherche de vous proposer des résultats pertinents même si vous faites des erreurs d'orthographe. Supposons que vous tapiez le mot « Jva » au lieu de « Java », le moteur de recherche, ne connaissant pas ce mot, va rechercher tous les mots à une distance de Levenshtein de 1, comme « Java ».

Cet algorithme calcule le **nombre** d'insertions ou suppressions de caractères qu'il faut effectuer pour passer d'un mot à un autre.

Ce **nombre** est ce qu'on appelle **la distance de Levenshtein**.

Exemples de **distance de Levenshtein**:

1) Entre « chat » et « chats » la distance de Levenshtein vaut 1 car il faut :
   - **ajouter** une seule lettre (la lettre s) pour passer du mot « chat » au mot « chats ».
   - ou **supprimer** une seule lettre pour passer du mot « chats » à « chat ».

2) Entre « machins » et « machine » la distance de Levenshtein vaut 1 car il faut **remplacer** une seule lettre pour passer d'un mot à l'autre.

3) Entre « aviron » et « avion » la distance de Levenshtein vaut 1 car il faut **retirer** la lettre r du premier mot (ou ajouter selon le sens).

4) Entre « distance » et « instance » quelle est la distance ?

5) Entre « Chien » et « Chine » quelle est la distance ?

- **Voici la classe *dev.utils.StringUtils* :**

Cette classe a été trouvée sur internet mais nous souhaitons la tester correctement avant de l'intégrer à notre projet.

```java
** Classe qui fournit des services de traitements de chaines de caractères
 * @author DIGINAMIC
 */
public final class StringUtils {

	/** Retourne la distance de Levenshtein entre 2 chaines de caractères
	 * @param lhs chaine 1
	 * @param rhs chaine 2
	 * @return distance
	 */
	public static int levenshteinDistance(CharSequence lhs, CharSequence rhs) {
	    int len0 = lhs.length() + 1;
	    int len1 = rhs.length() + 1;

	    int[] cost = new int[len0];
	    int[] newcost = new int[len0];

	    for (int i = 0; i < len0; i++) {
	       cost[i] = i;
	    }

	    for (int j = 1; j < len1; j++) {
	       newcost[0] = j;

	       for (int i = 1; i < len0; i++) {
	             int match = (lhs.charAt(i - 1) == rhs.charAt(j - 1)) ? 0 : 1;

	             int costReplace = cost[i - 1] + match;
	             int costInsert  = cost[i] + 1;
	             int costDelete  = newcost[i - 1] + 1;

	             newcost[i] = Math.min(Math.min(costInsert, costDelete), costReplace);
	        }

	        int[] swap = cost;
	        cost = newcost;
	        newcost = swap;
	    }
	    return cost[len0 - 1];
	}
}
```

Recopiez cette classe dans votre projet.

Réalisez une **classe** de tests unitaires qui permet de tester la classe ci-dessus en respectant les instructions suivantes :

a) Ne mettez **pas** votre classe de tests dans **src/main/java**.
   a. Pensez à une configuration MAVEN spécifique pour les classes de tests.
b) Couvrez un maximum d'exemples (i.e. cas de tests)
c) Intéressez-vous également à la robustesse de cette classe. Que se passe t'il si on passe à cette classe des paramètres NULL ? Proposez un correctif pour rendre cette classe plus robuste.