Formation Bonnes Pratiques

TP 02 - Tests Unitaires avec JUnit

Tests unitaires avec JUnit

Exercice 1

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.

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 ?

Création du projet

- Créez un projet ***demo-tests-unitaires***.
- Synchronisez votre projet avec un projet GitHub
- Passez à la page suivante pour débuter le TP
- Ajoutez/Modifiez la dépendance vers la librairie *JUnit* :

pom.xml

```
        <dependency>
            <groupId>junit</groupId>
            <artifactId>junit</artifactId>
            <version>4.12</version>
            <scope>test</scope>
        </dependency>
```

Exercice 1

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

```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.

Exercice 2

Reprenez la classe Maison et testez là avec une classe de tests unitaires.
Pensez à tester les cas aux limites : que se passe t'il si j'ajoute null plutôt qu'une pièce ?

Exercice 3

Mise en place des tests unitaires sur le projet pizzeria

1) Réalisez la classe de tests unitaires de la classe PizzaMemDao.

2) Réalisez les classes de tests unitaires de vos classes de services.

   - Problème : comment traiter le cas de la classe Scanner qui attend que l'utilisateur saisisse une information ? N'oubliez pas qu'une classe de tests unitaires doit être automatisable, ce qui signifie qu'elle doit pouvoir s'exécuter de nuit sur une plateforme dédiée.

   - Solution : il faut utiliser une « rule », afin de modifier le comportement du Scanner.

   - Mise en œuvre : il est nécessaire d'utiliser une librairie appelée « system-rules » qui propose des « Rules » pour JUnit.

pom.xml

```
<dependency>
	<groupId>com.github.stefanbirkner</groupId>
	<artifactId>system-rules</artifactId>
	<version>1.16.0</version>
	<scope>test</scope>
</dependency>
```

- Analyser les exemples ci-dessous avant de vous lancer dans les développements

Exemple d'une classe qui utilise un Scanner :

```
import java.util.Scanner;

/** Service de calculs interactifs */
public class CalculService {

	/** Demande à l'utilisateur un nombre et renvoie le carré de ce nombre */
	public int calculerCarre(Scanner scanner){

		System.out.println("Veuillez saisir un chiffre:");
		String valeur = scanner.next();
		return Integer.parseInt(valeur);
	}

	/** Demande à l'utilisateur 2 nombres et renvoie la multiplication */
	public int calculerMult(Scanner scanner){

		System.out.println("Veuillez saisir un 1er chiffre:");
		String val1 = scanner.next();

		System.out.println("Veuillez saisir un 2nd chiffre:");
		String val2 = scanner.next();

		return Integer.parseInt(val1) * Integer.parseInt(val2);
	}
}
```

Et, ci-dessous, la classe de tests unitaires associée qui utilise une rule spécifique pour modifier le comportement du Scanner.

La classe **TextFromStandardInputStream** modifie **System.in** afin de pouvoir injecter du comportement via la méthode **provideLines**:

```
import org.junit.Rule;
import org.junit.Test;
import org.junit.contrib.java.lang.system.TextFromStandardInputStream;
import static org.junit.contrib.java.lang.system.TextFromStandardInputStream.*;

public class CalculServiceTest {

	/** Création d'une "Rule" qui va permettre
	 * de substituer le System.in utilisé par le Scanner
	 * par un mock: systemInMock */
	@Rule
	public TextFromStandardInputStream systemInMock = emptyStandardInputStream();

	@Test
	public void testCalculerCarre() {

		// J'alimente le mock avec la valeur 8
		systemInMock.provideLines("8");

		CalculService serv = new CalculService();
		int value = serv.calculerCarre(new Scanner(System.in));

		assertEquals(64, value);
	}

	@Test
	public void testCalculerMult() {

		// J'alimente le mock avec la valeur 8
		systemInMock.provideLines("8", "7");

		CalculService serv = new CalculService();
		int value = serv.calculerMult(new Scanner(System.in));

		assertEquals(56, value);
	}
}
```