ISET SFAX	DEPARTEMENT TECHNOLOGIES DE L'INFORMATIQUE

ATELIER 1 : TEST UNITAIRE AVEC JUNIT

Matière : ATELIER TEST ET QUALITE LOGICIELLE	**Niveau :** DSI 2
Enseignante : MME. MANEL BEN SALAH	

Exercice 1 : Calculatrice

1. Sous Eclipse, créez un nouveau projet java
2. Nommez votre projet **calculator** en laissant les options par défaut puis cliquez sur OK.
3. Ajoutez un package org.math dans le répertoire src.
4. Ajoutez la classe **FourOpCalculator** dans le répertoire org.math.

```
package org.math;

public class FourOpCalculator {
	public int add(int a, int b)
	{
		return a + b;
	}

	public int sub(int a, int b)
	{
	return a - b;
	}

	public int mul(int a, int b)
	{
	return a * b;
	}

	public int div(int a, int b)
	{
	return a / b;
	}
}
```

5. Ajoutez un nouveau dossier de sources (source Folder) nommé **test** au même niveau d'arborescence que src par New | Source Folder.
6. Ajoutez une classe de test en suivant les étapes nécessaires.
7. Terminer la classe de test selon la structure suivante en utilisant plusieurs scénarios de tests:

```java
package org.math;
import static org.junit.jupiter.api.Assertions.*;
import org.junit.jupiter.api.Test;

class FourOpCalTest {

	@Test
	void testAdd() {
		fail("Not yet implemented");
	}

	@Test
	void testSub() {
		fail("Not yet implemented");
	}

	@Test
	void testMul() {
		fail("Not yet implemented");
	}

	@Test
	void testDiv() {
		fail("Not yet implemented");
	}
}
```

Exercice 2 : Gestion des comptes bancaire simplifié

Un compte bancaire (simplifié) est défini par le solde disponible sur le compte. Les opérations de manipulation minimales seront :

- Initialiser un compte bancaire
- Accéder au solde d'un compte bancaire
- Créditer un compte bancaire
- Débiter un compte bancaire

On étendra également la classe avec :

- Un constructeur permettant d'initialiser le solde.
- Une méthode afficher qui affiche l'objet.
- Une méthode virerVers qui effectue un virement vers un autre compte bancaire.

1. Réaliser une implémentation en langage Java de la classe précédemment spécifiée.
2. Ecrire un programme JUnit pour les tests unitaires de la classe **Compte**

Bien entendu il vous est conseillé d'écrire les tests au fur et à mesure que vous implémenterez les méthodes de la classe Compte. Vous pouvez même, une fois les spécifications de votre classe Compte effectuées

a. Ecrire le squelette de la classe **Compte** (en mettant un corps vide pour chacune de méthodes)
b. Générer le code du programme JUnit pour les tests unitaires,

c. Ecrire les différents cas de test
d. Complétez le code de la classe **Compte**, en effectuant à chaque étape l'exécution des tests unitaires.

Exercice 3 : Spécification de la collection

On considère une classe **Collection** dont les instances stockent des couples clé-valeur. Les clés sont des instances d'Object, de même que les valeurs.

- On supposera que les clés et les valeurs ne sont jamais null.
- Une valeur peut être associée à deux clés distinctes.
- Une collection a une capacité fixée à l'initialisation, strictement supérieure à 1, qu'on peut agrandir quand la collection est plein (et seulement dans ce cas).
- On peut ajouter et retirer des couples clé-valeur d'une collection, tester si une clé est présente dans une collection, ainsi que retrouver dans ce cas la valeur associée.
- Si on tente de retirer une clé absente d'une collection, rien ne se passe.
- Si on ajoute un couple clé-valeur pour une clé qui existe déjà, l'ancien couple est écrasé.
- Si on ajoute un couple clé-valeur à une collection plein, l'exception **DebordementCollection** (sous-classe de **ErreurCollection**) est signalée.
- Si on cherche la valeur d'une clé absente, l'exception **ErreurCollection** est signalée, de même que si on applique une méthode dans un état où elle est interdite.

Quand une collection est pleine, et uniquement dans ce cas, on peut la redimensionner en fixant une capacité plus grande. Quand la collection n'est pas vide, et uniquement dans ce cas, on peut vider la collection en supprimant tous les éléments, sans changer sa capacité. Parmi les autres fonctionnalités, on peut savoir si une collection est vide, connaître sa capacité ainsi que son nombre courant de couples clé-valeur.

Le squelette de la classe Java Collection est la suivante :

```
Public class Collection {
     Public Collection (int n) throws ErreurCollection {}
     public void ajouter (Object C, Object O) throws ErreurCollection {}
     public void retirer (Object C){}
     public void raz() throws ErreurCollection {}
     public void redimensionner(int nouv) throws ErreurCollection {}
     public boolean present (Object C) {}
     public Object valeur(Object C) throws ErreurCollection {}
     public boolean est Vide(){}
     public int taille (){}
     public int capacite (){}
}
```

À partir de la spécification de la classe Collection donnée, écrire un ensemble de tests JUnit pour cette classe permettant de valider la spécification déjà expliquée auparavant. Pour chaque test, vous préciserez obligatoirement en commentaire l'objectif du test et le résultat attendu. Penser à tester aussi bien les cas qui doivent réussir que les cas qui doivent lever une exception: l'objectif est de couvrir un maximum de cas différents parmi les cas possibles. Pensez également aux cas aux limites.

Exercice 4 : Gestion des Étudiants d'une Université

On souhaite développer et tester une application en mode console permettant de gérer les étudiants dans une université. Cette application doit offrir une gestion complète des étudiants, des matières enseignées, des formations proposées, ainsi que des groupes d'étudiants. Elle doit également permettre de calculer la moyenne générale des étudiants pour chaque matière.

Classes à implémenter :

- **Etudiant** : caractérisé par un numéro CIN, un nom, un prénom, une classe et une formation.
- **Formation** : caractérisée par un identifiant unique et une liste de matières.
- **Groupe** : liste d'étudiants appartenant à une même formation.
- **Matière** : caractérisée par un identifiant et un nom.
- **EtudiantExistException** : exception personnalisée à lever lorsqu'on tente d'ajouter un étudiant déjà existant dans la liste.

Travails demandés :

1. Développer les classes nécessaires pour gérer les étudiants, les formations, les matières et les groupes

2. Implémenter une classe pour la gestion du groupe d'étudiants, leurs affectations et le calcul des moyennes.
3. Ecrire des tests unitaires permettant de tester le code écrit en insistant sur les fonctionnalités de base.

A titre indicatif :

- **Ajouter un étudiant :** Vérifier qu'un étudiant est correctement ajouté à une formation.
- **Ajouter un étudiant existant :** Vérifier que l'exception personnalisée EtudiantExistException est levée.
- **Affecter une formation à un étudiant :** Vérifier que la formation est bien affectée à l'étudiant.
- **Calcul de la moyenne générale par étudiant :** Vérifier l'exactitude de la moyenne calculée selon les notes attribuées.
- **Lister les étudiants par groupe :** Vérifier que la liste des étudiants du groupe correspond bien à la formation associée.