Java : les Tests Unitaires (Application)

Objectif

- Mettre en œuvre les tests unitaires dans un projet à plusieurs classes. Le projet gèrera la conversion de plusieurs devises.

Règles de gestion

- La classe métier **Devise** permet de gérer une devise. Elle stocke son nom et son taux de change par rapport au dollar.
- La classe métier **Convertisseur** offre un service de conversion de valeurs entre plusieurs devises. Pour cela, elle stocke une liste de devises dans une table d'association (voir plus loin) et permet l'ajout de nouvelles devises.

- Une devise est identifiée par un nom de trois lettres (exemples : USD, EUR, GBP).
- Le taux de change d'une devise est obligatoirement positif.
- On ne peut pas ajouter au convertisseur une devise déjà existante.
- Toutes les conversions se font par rapport au dollar US dont le taux de change est 1.

Création du projet et des classes

- Créer un nouveau projet.

- Créer une classe **Devise**. Le code de cette classe est le suivant :

```
public class Devise {

    public Devise (String nom, double tauxChange) {

    }

    public String getNom() {
        return "";
    }

    public double getTauxChange() {
        return 0.0;
    }
}
```

Bien qu'incomplète, il sera possible de tester unitairement cette classe.

Création d'un test unitaire pour le constructeur

Comme vu précédemment, on va écrire un test unitaire afin de valider le bon fonctionnement de cette classe.

- Dans l'arborescence du projet faire un clic droit sur la classe, puis ***Outils – Create Tests***.

- Remplacer la méthode `testSomeMethod()` par celle-ci :

```
public void testConstructeur() {

        String nom = "USD";
        double tauxChange = 1.0;

        Devise devise = new Devise(nom,tauxChange);

        assertEquals(nom,devise.getNom());
        assertEquals(tauxChange,devise.getTauxChange());
}
```

- Tester cette méthode : elle produit un résultat négatif. Compléter la classe **Devise** et exécuter le test unitaire jusqu'à ce qu'il fonctionne.

Test des règles métier

Il faut maintenant tester et valider les règles métier de la classe **Devise**.

La première règle métier indique que le nom d'une devise comporte obligatoirement trois lettres. Si ce n'est pas le cas, la construction de l'objet **Devise** doit échouer, ce qui ce traduit par la levée d'une exception dans le constructeur. Notre test doit donc tenter de créer un objet **Devise** avec un nom incorrect et vérifier que la construction échoue.

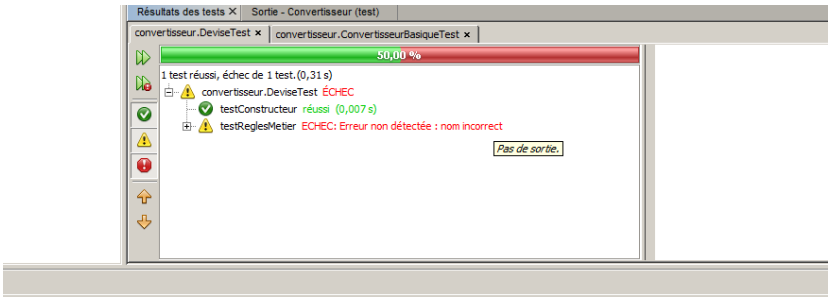
Comment faire en sorte que la levée d'une exception indique le succès du test, et que l'absence d'exception se traduise par un échec ?

- Ajouter à la classe de Test la méthode suivante :

```
public void testReglesMetier() throws Exception {

        Devise devise;
        try {
            devise = new Devise ("EURO",0);
            fail("Erreur non détectée : nom de la devise incorrect");
        }
        catch (Exception e) {
            // erreur attendue
        }
}
```

- Lancer le test unitaire ; le test du constructeur est toujours OK, mais le test de la nouvelle méthode provoque une erreur :



⇨ Aucune règle métier n'est pour l'instant codée ; c'est la méthode `fail()` , héritée de la classe **TestCase**, qui provoque l'échec du test unitaire.

Implémentation de la règle métier : nom de la devise sur 3 caractères

- Modifier le constructeur comme suit :

```
public Devise (String nom, double tauxChange) throws Exception {
    if (nom.length() == 3) {
        this.nom = nom;
        this.tauxChange = tauxChange;
    }
    else {
        throw new Exception ("Le nom " + nom + " est incorrect");
    }
}
```

- Modifier aussi la méthode `testConstructeur()` en ajoutant un `try/catch`.

- Lancer à nouveau le test unitaire : il fonctionne.

⇨ le constructeur a bien levé une exception, retournée à la classe **DeviseTest**, qui a donc géré cette exception dans le `catch` sans exécuter la méthode `fail()` .

Implémentation de la règle métier : le taux de change doit être positif

- Compléter la méthode `testReglesMetier()` de **DeviseTest** pour valider la règle.

- Modifier le constructeur et exécuter le test jusqu'à ce que tout fonctionne.

La classe Convertisseur

Cette classe offre un service de conversion multi-devises.
Les règles métier seront les suivantes :

- On ne peut pas ajouter au convertisseur une devise déjà existante
- Toutes les conversions se font par rapport au dollar US dont le taux de change est 1

Codage de la classe

```
public class Convertisseur {

    public void ajouterDevise (String unNom, double unTauxchange) throws Exception {
        throw new Exception ("Méthode pas encore écrite !");
    }

    public double convertir (double unNombre, String nomDevisedepart,
            String nomDeviseCible) throws Exception {
        throw new Exception ("Méthode pas encore écrite !");
    }
}
```

Chaque méthode lèvera bien sûr une exception.

Note : on va d'abord écrire les tests, puis ensuite compléter la classe.

Création du test unitaire

- Créer la classe ***ConvertisseurTest***.

- On teste d'abord la fonctionnalité d'ajout de devises :

```
public void testAjouterDevise() throws Exception {
        Convertisseur convertisseur = new Convertisseur();

        try {
            convertisseur.ajouterDevise("EUR", 1.36);
            fail ("Erreur non détectée (devise déjà ajoutée");
        }
        catch (Exception e) {
            System.out.println(e.getMessage());
        }
    }
```

On va compléter le test unitaire avant de coder correctement les méthodes de la classe Convertisseur.

- Coder la méthode suivante :

```
    public void testConvertir() throws Exception {

        Convertisseur convertisseur = new Convertisseur();

        convertisseur.ajouterDevise("EUR", 1.36);
        convertisseur.ajouterDevise("USD", 1.0);
        convertisseur.ajouterDevise("GBP", 1.6);

        // conversion d'1 euro en en euros
        double resultat = convertisseur.convertir(1.0, "EUR", "EUR");
        assertEquals(1.0,resultat);

        // conversion d'1 livre GB en dollar US
        resultat = convertisseur.convertir(1.0, "GBP", "USD");
        assertEquals(1.6,resultat);

        // conversion de 20 euros en livres GB
        resultat = convertisseur.convertir(20.0, "EUR", "GBP");
        assertEquals(17.0,resultat);

        try {
            convertisseur.convertir(1.0, "YEN", "USD");
            fail ("erreur non encore détectée : devise inconnue");
        }
        catch (Exception e) {
            System.out.println(e.getMessage());
        }
    }
```

⇨ Le test unitaire ***ConvertisseurTest*** vérifie à présent toutes les fonctionnalités de la classe ***Convertisseur***.

Cependant, on voit que la création d'un convertisseur et l'ajout de devises se répètent dans les 2 méthodes de test.

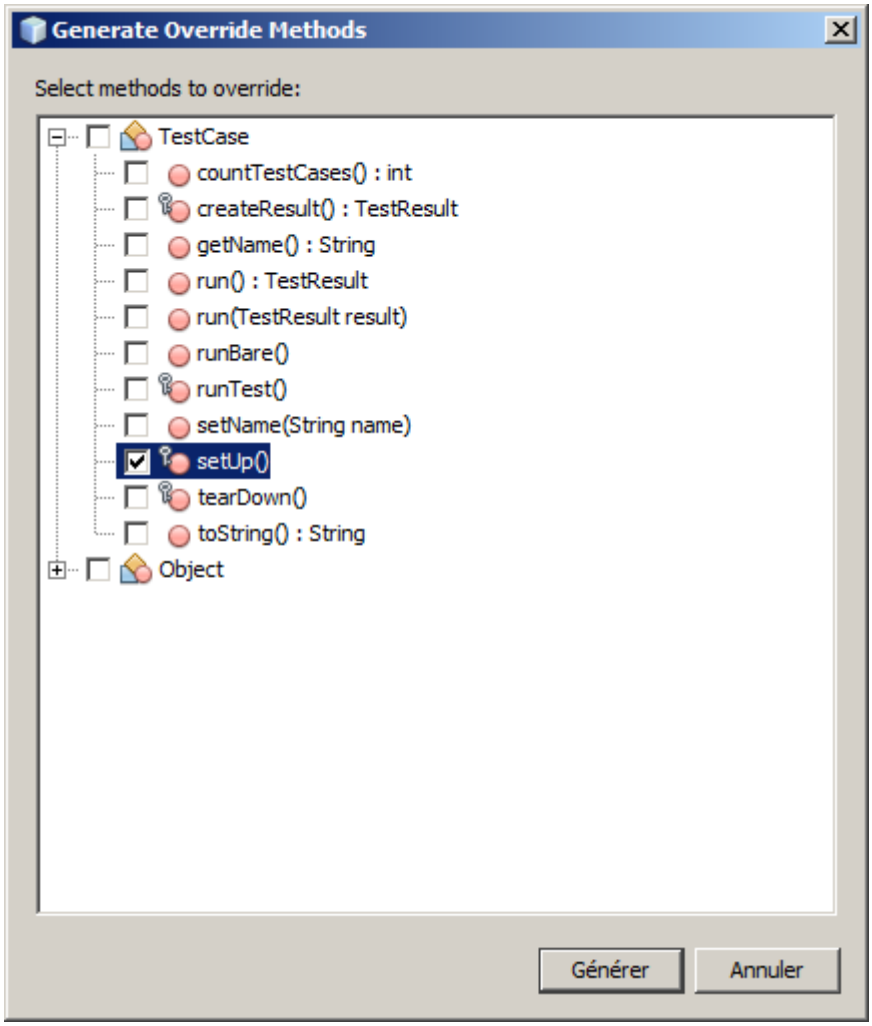
JUnit permet de factoriser le code commun aux méthodes de test.

Optimisation du test unitaire

- Ajouter une propriété dans la classe ***ConvertisseurTest*** :

```
    private Convertisseur convertisseur;
```

- Dans l'éditeur de code de la classe ***ConvertisseurTest***, faire un clic droit sur, puis ***Insérer du code - Override Method*** ; on obtient la boîte de dialogue :



Sélectionner ***setUp()*** , puis ***Générer.*** On obtient le résultat :

```
    @Override
        protected void setUp() throws Exception {
            super.setUp();
        }
```

`@Override` signifie que la surcharge est intentionnelle.

- Cette nouvelle méthode a pour rôle de préparer un test. Elle est appelée par JUnit avant l'exécution de chaque méthode de test. On va l'utiliser pour instancier un convertisseur et lui ajouter une devise.

```
    @Override
        protected void setUp() throws Exception {
            convertisseur = new Convertisseur();
            convertisseur.ajouterDevise("EUR", 1.36);
        }
```

- Modifier les méthodes de test pour supprimer ces lignes de code factorisées.

Le test unitaire est complet. Il reste à le faire réussir en codant la classe ***Convertisseur*** correctement.

Codage des méthodes

- Ajouter un attribut privé à la classe :

```
private HashMap<String,Devise> listeDevises;
```

- Ajouter et coder la méthode suivante dans la classe :

```
public Devise getDevise(String unNomDevise) {

}
```

- Coder le reste de la classe en utilisant les méthodes suivantes de la classe HashMap :

`put(key,value)`	crée un élement en associant la valeur `value` à la clé `key`
`containsKey(key)`	renvoie Vrai si le dictionnaire contient un element de clé `key`
`get(key)`	renvoie la valeur associée à la clé `key`

- Exécuter le test unitaire jusqu'à obtenir un bon fonctionnement de la classe ***Convertisseur***.