TP de GL en JAVA

Une séance et demie

Objectifs: Pratiquer le TDD.

1 Contexte

Dans cet exercice, on va programmer de façon simplifiée une banque en utilisant le développement dirigé par les tests (TDD en anglais). La programmation se fera en Java et utilisera GitLab. Afin de vérifier que vous suivez bien la démarche TDD, nous vous demandons, une fois n'est pas coutume de comiter vos modifications, après l'écriture de chaque test, puis quand le test est vert, puis après refactoring. /!\Il faut être bien conscient que c'est uniquement pour les besoins de l'exercice qu'on vous demande de comiter une version du logiciel pour lesquels au moins un test est rouge.

Rappel le TDD consiste à écrire un test. Vu que c'est la première chose que vous écrirez, ce test sera rouge. Ensuite, on écrit le code pour faire passer le test au vert. Puis ensuite, on refactore le code pour qu'il soit *beau*, c'est-à-dire plus lisible, maintenable et évolutif.

Dans cette banque simplifiée, on a des comptes et des comptes épargne qui répondent à des règles différentes.

2 Compte

Dans une première version, on va juste avoir un réel pour représenter l'ensemble des débits et un autre l'ensemble des crédits.

2.1 Premier test - création

Test Ecrire un premier test qui lorsqu'on crée un compte vérifie que le crédit et le débit sont à 0.

Code Ecrire le code qui permet de faire passer ce test au vert.

2.2 Créditer et débiter un compte

Test Ecrire un test qui lorsqu'on crédite un compte vérifie que la variable `crédit` a été incrémentée de la valeur correspondante.

Code Ecrire le code qui permet de faire passer ce test au vert.

Débiter Procéder de façon similaire pour débiter un compte.

2.3 Amélioration

Que se passe-t-il si on crédite ou débite un montant négatif ?

Test Ecrire un test qui permet de vérifier que ce cas n'arrive pas.

Code Ecrire le code qui permet de faire passer ce test au vert.

2.4 Calcul du solde d'un compte

Test Ecrire un test qui aprés avoir créditer ou débiter plusieurs fois un compte vérifie que le solde du compte est bien celui attendu.

Code Ecrire le code qui permet de faire passer ce test au vert.

2.5 Amélioration 2

Afin de pouvoir éditer le relevé de compte, on stocke (par exemple dans des listes) l'ensemble des crédits et des débits effectués sur ce compte. Une fois encore pour simplifier les choses, un débit et un crédit seront ici matérialisés uniquement par un réel. Le solde sera calculé par la somme des crédits moins la somme des débits. On pourra bien évidemment définir des méthodes intermédiraires si nécessaire.

On va historiser les débits et les crédits en stockant chacun des crédits et des débits dans un tableau de réels (`double[]`. Quand le tableau des crédits est plein la somme des crédits est stockée à l'indice 0 dans le tableau et le nouveau crédit à l'indice 1.

Test Ecrire un test qui vérifie que ce mécanisme fonctionne.

Code Ecrire le code qui permet de faire passer ce test au vert.

Modification des autres tests existants Modifier éventuellement les autres tests pour qu'ils passent au vert. Au passage vous essaierai de rendre votre code le plus modulaire et évolutif possible par exemple en le rendant indépendant de la structure de données choisie.

2.6 Amélioration 3

Que se passe-t-il si on fait un crédit ou un débit de 0 ?

Test Ecrire un test qui vérifie qu'un crédit ou un débit de 0 est interdit.

Code Ecrire le code qui permet de faire passer ce test au vert.

2.7 Amélioration 4

Pour éviter le blanchiment d'argent et la fraude fiscale, le montant des débits et des crédits ne peut dépasser 100 000.

Test Ecrire un test qui vérifie qu'un crédit ou un débit ne dépasse pas le montant autorisé.

Code Ecrire le code qui permet de faire passer ce test au vert.

2.8 Amélioration 5

L'utilisation d'un tableau n'est quand même pas la meilleure solution, on va donc utiliser une `ArrayList()`. Normalement, si vous avez bien modifié votre code à la question 2.5 tous vos tests devraient être verts. Si ce n'est pas le cas, modifiez le code pour qu'ils passent au vert.

3 Compte épargne

Un compte épargne est un compte un peu particulier. Il n'est pas autoriser d'avoir un solde négatif sur un compte épargne. Par ailleurs, ce type de compte permet de gagner de l'argent. Il a donc un taux d'intérêt.

3.1 Création d'un compte épargne

Test Ecrire un test qui lorsqu'on crée un compte épargne vérifie que l'ensemble des crédits et des débits associés sont vides.

Code Ecrire le code qui permet de faire passer ce test au vert.

Créditer et solde Procéder de façon similaire pour créditer un compte épargne et calculer le solde. Pour le moment, on ne s'intéresse pas aux débits.

3.2 Débiter un compte épargne

Test Ecrire un premier test qui lorsqu'on vérifie que si on veut débiter un compte épargne d'un montant supérieur au solde, le débit ne s'effectue pas.

Code Ecrire le code qui permet de faire passer ce test au vert.

On réalise que ne rien renvoyer n'est pas une bonne idée. On va donc améliorer notre logiciel pour lever une exception.

Test Ecrire un nouveau test qui lève une exception si on veut débiter un compte épargne d'un montant supérieur à son solde. Bien évidemment, le test précédent ne fonctionne plus. Mettez le en commentaire pour en garder une trace.

Code Ecrire le code qui permet de faire passer ce test au vert.

3.3 Calculer et créditer les intérets

Test calcul des intérêts Ecrire un test qui vérifie que le calcul des intérêts est correct. Ce test implique sans doute la création d'un attribut `interet`. Il faudra alors penser à quand et comment on l'initie.

Code Ecrire le code qui permet de faire passer ce test au vert.

Test échéance Ecrire un test qui vérifie qu'à échéance (c'est-à-dire chaque fois que la méthode `echeance` est appelée, on ne tiendra pas du tout compte de date ou quoi que ce soit, c'est une banque simplifiée !) les intérêts ont été correctement crédités.

Code Ecrire le code qui permet de faire passer ce test au vert.

4 Banque

4.1 Ouverture de banque

Test Ecrire un test qui lorsqu'on crée une banque vérifie que l'ensemble des comptes associés est vide.

Code Ecrire le code qui permet de faire passer ce test au vert.

4.2 Ouverture de comptes et de comptes épargne

Procédez comme jusqu'à présent en commençant par les tests pour programmer l'ouverture de comptes et de compte épargne.

4.3 Créditer / Débiter un compte ou un compte épargne

En considérant qu'on récupère un compte grâce à son numéro qui est son indice dans la liste de comptes, programmez toujours en commençant par les tests les opérations permettant de créditer et de débiter un compte ou un compte épargne.

4.4 Gérer les cas de références à des comptes inexistants

Puisqu'on récupère un compte grâce à son indice dans la liste de comptes, il peut arriver que le compte associé au numéro n'existe pas. Écrivez le ou les tests nécessaires et le code correspondant ensuite.

4.5 Virement

Toujours en commençant par les tests, programmez une méthode qui permet de virer de l'argent d'un compte à un autre. Vous gèrerez le cas où le compte source est un compte épargne ainsi que le cas où l'un des deux comptes n'existe pas.