Rapport TP 3 - IFT3913

Loïc Daudé Mondet
20243814 – Programmes d'échanges - 1er c.(Échange)

Alaa edlin Yacoub

16/12/2022

I – Tests boîte noire

Définition des tests

On défini les classes d'équivalence comme les cas pour lesquels le programme doit se comporter à peu près de la même manière.

Devises

On peut dans un premier temps tester ce qu'il se passe en faisant varier les devises. Pour ce faire on identifie les classes d'équivalence suivantes :

- deux valeur dans la liste {USD, CAD, GBP, EUR, CHF, INR, AUD}
- une valeur appartient à liste et l'autre non
- les deux valeur n'appartiennent pas à cette liste

On peut choisir par exemple le jeu de test :

- USD, CAD
- USD, ZZZ
- XXX, ZZZ

On peut également considérer les classes où :

- une valeur n'appartient pas à la liste {USD, CAD, GBP, EUR, CHF, INR, AUD}, mais est quand même présente dans le fichiers json avec une valeur non valide
- les deux valeurs n'appartiennent pas à la liste {USD, CAD, GBP, EUR, CHF, INR, AUD}, mais sont quand même présentent dans le fichiers json
- un mélange entre une valeur dans la liste {USD, CAD, GBP, EUR, CHF, INR, AUD} et une autre n'appartenant à cette liste, mais présente dans le fichier json

On a alors le jeu de test suivant :

- NGN, ZZZ
- NGN, ALL
- USD, NGN

Il n'y a pas de valeurs frontières car les variables sont nominales.
Normalement d'après la spécification seul le cas {USD, CAD} devrait renvoyer une valeur.

Montants

Pour les montant on identifie les classes suivantes :

- une valeur dans le domaine [0, 10000]
- une valeur en dessous de ce domaine
- une valeur au dessus de ce domaine
- une valeur frontière à la borne inférieure de ce domaine
- une valeur frontière à la borne supérieure de ce domaine

Ce qui donne le jeu de test suivant :

- 1000
- -1000
- 11000
- 0
- 10000

On s'attend à ce que la méthode convert() renvoie la valeur correctement convertie avec les taux de change présent dans le fichier json quand la classe d'équivalence est celle dans le domaine. Idem pour les valeurs frontières. D'après la spécification elle ne doit pas accepter de valeurs en dehors du domaine, on peut donc s'attendre à une `IllegalArgumentException` quand il s'agit des classes d'équivalence correspondant aux valeurs en dehors du domaine.

Total

Il y a donc 6 tests pour les devises et 5 tests pour les montants soit un total de 11 tests à effectuer.

Résultats

```
Results :

Failed tests:   ua.karatnyk.TestCurrencyConvertor.testCurrencyTestBothInJson(): ParseExceptio
n error was expected ==> Expected java.text.ParseException to be thrown, but nothing was thro
wn.
  ua.karatnyk.TestCurrencyConvertor.testCurrencyTestDomainAndJson(): ParseException error was
 expected ==> Expected java.text.ParseException to be thrown, but nothing was thrown.
  ua.karatnyk.TestCurrencyConvertor.testAmountExtremeLow(): IllegalArgumentException error wa
s expected ==> Expected java.lang.IllegalArgumentException to be thrown, but nothing was thro
wn.
  ua.karatnyk.TestCurrencyConvertor.testAmountExtremeUp(): IllegalArgumentException error was
 expected ==> Expected java.lang.IllegalArgumentException to be thrown, but nothing was throw
n.

Tests run: 11, Failures: 4, Errors: 0, Skipped: 0
```

On remarque que 4 tests sur les 11 échouent, la méthode se comporte bien dans les plages définies par ses domaines, mais en plus elle accepte des valeurs en dehors : des devises en dehors de la liste mais dans le fichier json et des montants en dehors de l'intervalle [0,10000]. La spécification n'est donc pas respectée.

II – Tests boîte blanche

On décide d'effectuer un critère de couverture des conditions car cela permet de trouver les erreurs les plus subtiles en évaluant au moins une fois toutes les composantes des conditions Voici le code de `convert()`

```
public final class CurrencyConvertor {

    12 usages   Loïc Daudé Mondet
    public static double convert(double amount, String from, String to, CurrencyConversion conversion) throws ParseException{
        if(!conversion.getRates().containsKey(to)|| !conversion.getRates().containsKey(from))
            throw new ParseException("Not correct format currency"
                    + "", 0);
        double curencyTo = conversion.getRates().get(to);
        double curencyFrom = conversion.getRates().get(from);
        return amount*(curencyTo/curencyFrom);

    }
}
```

On identifie la condition :
`if(!conversion.getRates().containsKey(to)||!conversion.getRates().containsKey(from))`
Il faut donc un jeu de test ou on a :

- !conversion.getRates().containsKey(to) = vrai et !conversion.getRates().containsKey(from) = vrai
- !conversion.getRates().containsKey(to) = vrai et !conversion.getRates().containsKey(from) = faux
- !conversion.getRates().containsKey(to) = faux et !conversion.getRates().containsKey(from) = vrai
- !conversion.getRates().containsKey(to) = faux et !conversion.getRates().containsKey(from) = faux

Les appels aux méthodes getRates et containsKey permettent de vérifier si la clef to (ou from) appartient au dictionnaire extrait du fichier json contenant les taux de conversion. Ces clefs sont les identifiants des devises comme USD ou CAD.

On se rend compte qu'on a déjà testé ces conditions lors des tests en boite blanche.

Il en ressort que si la méthode ne respecte pas la spécification demandée c'est car il manque des conditions, notamment vérifier que le montant est bien dans le domaine et que les devises à convertir appartiennent bien à notre liste réduite.