Bianca Bica (20161056)
Chaima Boussora (20159909)

Le 16 décembre 2022
Rapport - TP4

Tests boîte noire :

Les tests de la boîte noire nous permettent d'évaluer les résultats sur la base de la spécification seulement. En d'autres mots, on teste ce que le programme est supposé faire. C'est des tests de type *fonctionnels*. Dans le code du *Currency Convertor*, on exige que la spécification soit la suivante : nous voulons convertir des montants uniquement entre les devises USD, CAD, GBP, EUR, CHF, INR, AUD et seulement accepter des montants entre 0 et 10 000. Nous avons utilisé les approches de partition du domaine des entrées en classe d'équivalence et l'analyse des valeurs frontières. Avant d'implémenter les tests pour la méthode *convert()*, il a fallu tout d'abord déterminer les domaine D des valeurs d'entrées pour notre spécification. Nous avons deux types de variables : discrètes et continues.

Pour les variables discrètes, c'est-à-dire les devises, nous avons établi le domaine D comme étant les devises qui se retrouvent dans le fichier *OfflineFixer.json*. Notre programme P est alors défini sur les devises mentionnées ci-haut. Nous avons deux classes d'équivalence, une pour les valeurs d'entrées valides et une pour les valeurs d'entrées invalides :

D1 = {USD, CAD, GBP, EUR, CHF, INR, AUD}
D2 = {**D**\D1}

Quant aux variables continues, c'est-à-dire les montants entre 0 et 10 000, nous avons défini D comme étant les entiers où P est défini sur [0, 10 000]. Nous avons 3 classes d'équivalence :

D1 = $\{0 \leq d \leq 10\ 000\}$
D2 = $\{d < 0\}$
D3 = $\{d > 10\ 000\}$

Les jeux de tests pour les devises sont les suivants :

Pour le cas D1 :
{(FROM : CAD, TO : INR), (FROM : USD, TO : EUR)}

Pour le cas D2:

{(FROM : CAD, TO : HNL), (FROM : RSD, TO : CAD), (FROM: KES, TO: SLL)}

Les jeux de tests pour les montants sont les suivants:

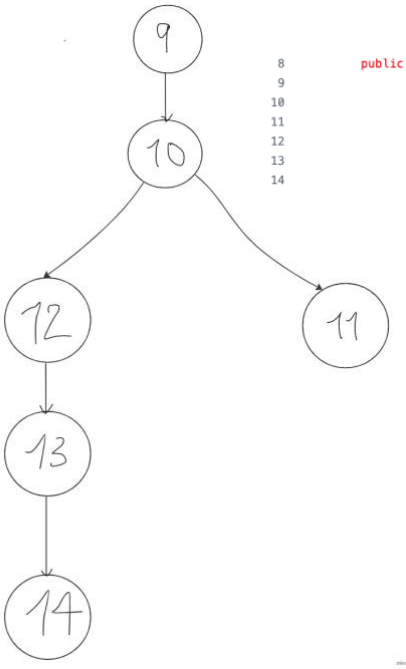
Valeurs typiques : 5000, - 20 000, 10 001, 20 000
Valeurs frontières : 0, 10 000, -1

Pour les jeux de tests D2 (devises) et D2/D3 (montants), nos tests sont configurés de sorte qu'ils retournent une *Exception* en se basant bien évidemment sur les conditions de la spécification. Si les conditions sont respectées, la conversion se fait correctement. Ces tests nous garantissent une

bonne couverture des domaines d'entrée de notre fonction. Ceux-ci permettent également de détecter des oublis par rapport à la spécification.

Tests boîte blanche :

Les tests boîte blanche diffèrent des tests boîte noire en type et en objectif. Les tests boîte blanche sont de type structurel et testent le programme en tenant compte de la structure interne. Cette fois-ci, on teste ce que le programme fait. Parmi les 5 critères de sélection de jeux de test, nous avons seulement choisi le critère de couverture des arcs du graphe de flot de contrôle et le critère de couverture des conditions. Contrairement aux tests boîte noire qui évaluent le comportement fonctionnel, nous avons choisi le deuxième critère de couverture (B) qui s'intéresse aux branchements de contrôle conditionnels dans le programme. Pour ce faire, il faut utiliser le graphe de flot de contrôle.




```
8   public static double convert(double amount, String from, String to, CurrencyConversion conversion) throws ParseException {
9           String[] listConvert = {"USD", "CAD", "GBP", "EUR", "CHF", "INR", "AUD"};
10          if (!(amount >= 0 && amount <= 10000) || !Arrays.stream(listConvert).anyMatch(from::equals)|| !Arrays.stream(listConvert).anyMatch(to::equals))
11                  throw new ParseException("Not correct format currency" + "", 0);
12          double curencyTo = conversion.getRates().get(to);
13          double curencyFrom = conversion.getRates().get(from);
14          return amount * (curencyTo / curencyFrom);
```

Les instructions correspondantes sont placées en photo ci-dessus afin de faciliter la compréhension du graphe. Nous avons choisi un jeu de test tel que lorsqu'on exécute P sur d, chaque arc du graphe de flot de contrôle de P est traversé au moins une fois.

Nous avons fait 2 tests :

1- Pour l'exécution des branches 9-10-11 :
   {Amount : 20 000 000, from : CAD, to : USD}
   Le montant ne respecte pas la condition du *if*, donc retourne une *exception.*
2- Pour l'exécution des branches 9-10-12-13-14 :
   {Amount : 20, from : CAD, to : USD}
   Nous avons mis des valeurs qui respectent les conditions du *if* afin de couvrir les autres instructions de la fonction.

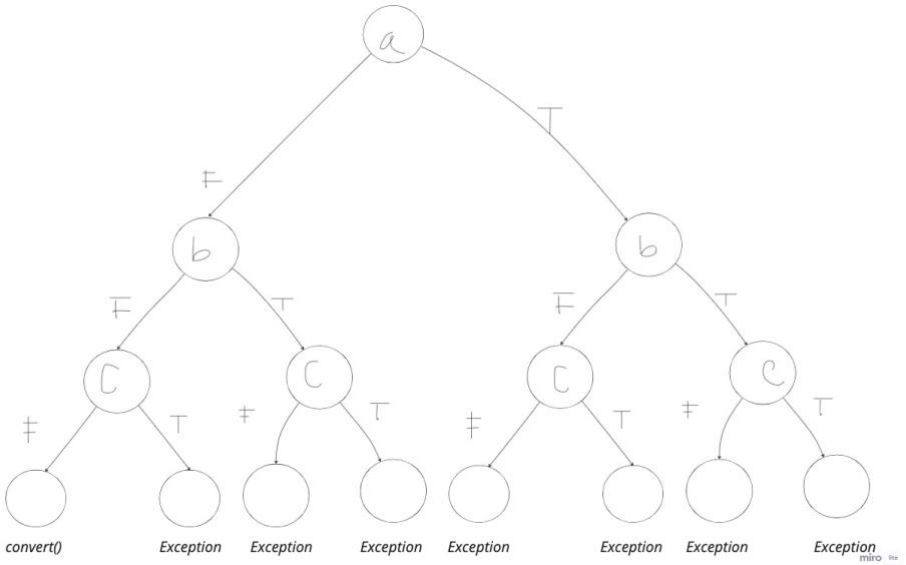
Bianca Bica (20161056)
Chaima Boussora (20159909)

Le 16 décembre 2022
Rapport - TP4

Nous avons choisi le critère de couverture des conditions (D) puisque celui-ci traverse toutes les valeurs possibles des conditions composées. Dans notre cas, nous avons une condition composée avec une disjonction de 3 clauses. Il est donc important de s'assurer que tous les cas sont traités pour éviter les diverses erreurs possibles. Nous avons couvert toutes les valeurs possibles de a, b et c (voir graphique ci-dessous) en leur attribuant une valeur « true » ou « false ».

```
if (!(amount >= 0 && amount <= 10000) || !Arrays.stream(listConvert).anyMatch(from::equals)||
!Arrays.stream(listConvert).anyMatch(to::equals))
```

Voici un graphe visuel qui démontre toutes les valeurs possibles que peuvent prendre nos clauses:



Comme on peut l'observer, toutes les valeurs (excepté FFF, puisque c'est celles-ci qui respectent la condition du *if*) vont retourner une exception. Ces tests se retrouvent à la ligne 96 de la classe *TestCurrencyConvertor.java*.

En conclusion, tous les tests que nous avons écrits passent et fournissent une bonne couverture des divers cas du code, autant pour la partie fonctionnelle que la partie structurelle.