Noms : Stefan Roman, Simo Hakim
Titre : TP4 – Currency Converter

1 Tests boîte noire

1.1 currencyConverter.Currency.convert

On créer les cas de tests pour vérifier si tous les types de devises sont existantes.

On partition le domaine selon les spécifications du logiciel.

$$D_1 = \{0 \leq d \leq 1000000\}$$
$$D_2 = \{d < 0\}$$
$$D_3 = \{d > 1000000\}$$
$$T = \{-1000000, 500000, 2000000\}$$

On ajoute les valeurs frontières au jeu de tests déterminer par la partition du domaine des entrées en classes d'équivalence.

$$D_1 = \{0 \leq d \leq 1000000\}$$
$$D_2 = \{d < 0\}$$
$$D_3 = \{d > 1000000\}$$
$$T = \{-1000000, -1, 0, 500000, 1000000, 1000001, 2000000\}$$

1.2 currencyConverter.MainWindow.convert

Les choix d'entrées pour les tests se feront en ordre d'apparition pour les arguments de la fonction. Les arguments sont les suivants :

$$(String\ currency1, String\ currency2, ArrayList < Currency > \ currencies, Double\ amount)$$

Donc, nous commençons par une partition du domaine des entrées en classe d'équivalence pour les valeurs discrètes.

Pour partitionner le domaine en classes d'équivalence, nous allons élaborer le domaine complet.

$$\{\text{"USD", "GBP", "EUR", "CHF", "CAD", "AUD"}\}$$

On partitionne le domaine en deux classes : les valeurs d'entrée valides et les valeurs d'entrée invalides. Soit C_{11} pour les valeurs valides, C_{12} pour les valeurs invalides.

$$C_{11} = \{\text{"USD", "GBP", "EUR", "CHF"}\}$$
$$C_{12} = \{\text{CAD, "AUD"}\}$$

Finalement nous avons un jeu de test valide : $C_1 = \{\text{"USD", "CAD"}\}$

Les mêmes procédures sont répétées pour le deuxième argument de la fonction mais nous changeons la valeur valide choisie. Ainsi le jeu de test valide est le suivant : $C_2 = \{\text{"EUR", "AUD"}\}$

Depuis ces ensembles nous pouvons faire les combinaisons suivantes :

$$T = \{(\text{"USD", "EUR"}), (\text{"USD", "AUD"}), (\text{"CAD", "EUR"}), (\text{"CAD", "AUD"})\}$$

Noms : Stefan Roman, Simo Hakim
Titre : TP4 – Currency Converter

Nous avons comme 3e argument une liste de valeurs. Si les valeurs pour "CAD" et "AUD" ne sont pas initialisées nous aurons des retours inattendus pour les 3 derniers tuples choisis. Ainsi il y a 2 classes pour la liste de devises, le domaine D_1 qui est celui par défaut de l'application et le domaine Cs_2 qui est le même que Cs_1 avec l'ajout de "CAD" et "AUD".

$$Cs_1 = \{\text{"USD"}, \text{"GBP"}, \text{"EUR"}, \text{"CHF"}\}$$
$$Cs_2 = \{\text{"USD"}, \text{"GBP"}, \text{"EUR"}, \text{"CHF"}, \text{"CAD"}, \text{"AUD"}\}$$

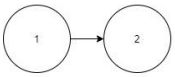
Le montant à convertir devrait être entre 0 et 1 million inclusivement pour être accepté. De cette manière, nous voyons qu'il devrait y avoir une analyse des valeurs frontières aussi pour cet argument. Ainsi, le domaine du montant *A* est le suivant :

$$A = \{-1000000, -1{,}0, 500000{,}1000000{,}1000001{,}2000000\}$$

On peut regarder la partition du domaine pour la fonction précédente pour voir les étapes suivis pour arriver à ce résultat.

2 Tests boîte blanche avec diagrammes de flux

2.1 currencyConverter.Currency.convert



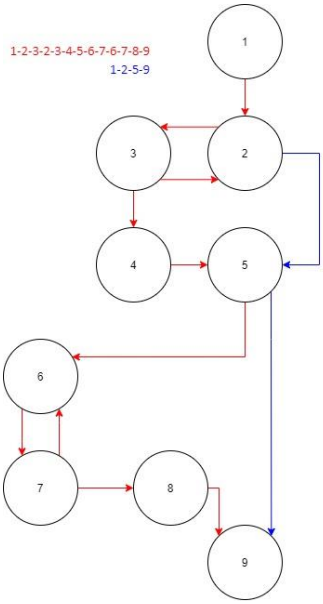

```
// Convert a currency to another
public static Double convert(Double amount, Double exchangeValue) {
    Double price;
1   price = amount * exchangeValue;
    price = Math.round(price * 100d) / 100d;

2   return price;
}
```

Dans ce cas il y a qu'un seul chemin à couvrir peu importe les critères.

2.2 currencyConverter.MainWindow.convert

La fonction est représentée ci-dessous sous la forme d'un diagramme de flux ainsi que les différents chemins à parcourir. Le chemin 5-6 implique qu'il est impossible de former un chemin 6-9.




```
// Find the short name and the exchange value of the second currency
public static Double convert(String currency1, String currency2, ArrayList<Currency> currencies, Double amount) {
    String shortNameCurrency2 = null;
1   Double exchangeValue;
    Double price = 0.0;

    // Find shortname for the second currency
2   for (Integer i = 0; i < currencies.size(); i++) {
3       if (currencies.get(i).getName() == currency2) {
            shortNameCurrency2 = currencies.get(i).getShortName();
4           break;
        }
    }

    // Find exchange value and call convert() to calcul the new price
5   if (shortNameCurrency2 != null) {
6       for (Integer i = 0; i < currencies.size(); i++) {
7           if (currencies.get(i).getName() == currency1) {
                exchangeValue = currencies.get(i).getExchangeValues().get(shortNameCurrency2);
8               price = Currency.convert(amount, exchangeValue);
                break;
            }
        }
    }

9   return price;
}
```

2.2.1 Critère de couverture des instructions

Pour que chaque instruction de la fonction est exécutée il faut une liste currencies qui contient les devises currency1 et currency2 existantes et un montant à convertir de 0 à 1 million inclusivement.

2.2.2 Critère de couverture des arcs du graphe de flot de contrôle

Premièrement, on choisit le chemin le plus complexe en premier qui accède au maximum d'instructions.
Chemin : 1-2-3-2-3-4-5-6-7-6-7-8-9Deuxièmement, on prend un chemin différent au nœud le plus près du départ.
Chemin : 1-2-5-9

2.2.3 Critère de couverture des chemins indépendants du graphe de flot de contrôle

Il est clair que le critère de couverture des arcs du graphe de flot de contrôle couvre tous les chemins indépendants du graphe de flot contrôle. Ainsi, on maintient les mêmes chemins.

2.2.4 Critère de couverture des conditions

Il n'y a pas de conditions composées alors ce critère est inutile pour tester cette fonction.

2.2.5 Critère de couverture des i chemin

Pour tester les boucles visibles dans le diagramme flux, il est suffisant de prendre les cas d'une boucle qui fera 0 itérations et une boucle qui fera 1 à n itérations où n est la taille de la liste currencies. Ceci est suffisant car les boucles agissent de la même manière peu-importe le nombre d'itérations car ces boucles servent de lookup de la liste currencies et donc n'ont pas d'effet cumulatif. Alors le cas où la boucle n'est jamais visitée est le seul autre type de cas.

Finalement les cas de tests pour les 2 chemins sont les suivants :

$$\{(currency1, currency2, currencies, amount)|^{currencies \ni currency1, currency2}_{0 \leq amount \leq 1000000}\}$$

$$\{(currency1, currency2, currencies, amount)|^{currencies \not\ni currency1, currency2}_{0 \leq amount \leq 1000000}\}$$

3 Tests invalides

Certaines assertions dans les tests boîte noire ne devraient pas avoir passés. Étant donné que les tests boîte noir visent à tester la fonctionnalité de l'application, il est impératif que les tests valident bien que l'application fonctionne selon les spécifications. Nous concluons que le code source de l'application ne contient pas de vérifications pour s'assurer que les devises à convertir sont existantes et que le montant est contenu entre $[0,\ 1000000]$.

4 Tests annexes

Nous avons rajouté des tests afin de mettre en évidence le manque de logique du code source pour respecter toutes les spécifications. "testConvertInvalidAmountInvalidCurrency, testConvertValidAmountInvalidCurrency" étant les cas avec une devise incorrecte, ne sont pas traités et échouent. Ce dernier ne gère pas les cas où la devise de destination est invalide d'une manière spécifique.

Une autre erreur est mise en avant par "testConvertInvalidAmountValidCurrency" qui marche malgré le fait que les montants testés sont hors des bornes supposées être tolérés.

Noms : Stefan Roman, Simo Hakim
Titre : TP4 – Currency Converter

A l'opposé, les tests testConvertZeroAmount, testConvertEmptyCurrencyList et testConvertNoCurrencies passent, la logique du code prenant en compte ces cas aberrants comme démontré dans ce rapport.