Auteurs : Christian Lungescu (20079725), Malik Abada (20173066) **TP3 du cours IFT-3913**

Partie 1

Note : Dans notre rapport, **la tache 3 est inclue dans l'ensemble du rapport.** De plus, le « range » est la plage des valeurs acceptables pour une variable**.** Pour les tests de **currencyConverter.MainWindow.convert(String, String, ArrayList, Double**), si on utilise les noms tels que "**USD**", tous les tests ratent. On ne peut pas savoir pourquoi sans avoir accès au code, donc pas pour la boite noire. Toutefois, on a accès à l'interface. En observant l'interface, on peut remarquer que les "**USD**" sont représentés comme "**US Dollar**". En testant avec ces noms-là, les tests fonctionnent. On a donc utilisé ces noms, pour pouvoir tester les autres éléments aussi, mais la spécification n'est de base pas respectée car on ne peut pas utiliser les noms courts donnés dans la spécification. Toutefois, nous avons représenté les noms en version courte dans les jeux de test quand même.

Pour le test boite noire, n'ayant accès qu'à la spécification sans le code, on ne peut qu'hypothétiser que la manière dont les noms sont codés ou utilisés est mauvaise, ou que le code utilise la version longue et n'utilise pas du tout la version courte.

Tests pour la méthode « currencyConverter.MainWindow.convert(String curr1, String curr2, currenciesSupported, montant) »

Tout d'abord, voici des définitions des arguments de cette méthode :

- curr1 : Le nom (String) de la monnaie que l'on souhaite convertir à une autre (curr2 dans ce cas-là).
- curr2 : Le nom (String) de la monnaie dont on souhaite convertir curr1.
- currenciesSupported : Des objets représentant des types de monnaies supportés par le programme de conversion.
- montant : Le montant dont on souhaite convertir de la monnaie curr1 a la monnaie curr2.

On remarque qu'il y a 3 variables qui ne sont pas fixes (constantes) dans cette méthode : **curr1**, **curr2** et **montant**.
On peut les partitionner en différentes classes d'équivalences pertinentes (NOTE : on peut regrouper curr1 et curr2 pour créer leurs classes d'équivalences):
Note : Pour les classes d'équivalences du **« montant »**, on les partitionne en fonction de leur range et des valeurs frontières de ce range car, comme on l'a vu en cours, c'est à ces points-là qu'on peut trouver le plus des failles. Pour les classes d'équivalence de (curr1,curr2), on trouve tous les cas possibles de paires valeurs pour s'assurer de couvrir toutes les « edge cases ».

Voilà le tableau des classes d'équivalences pour la variable « Montant » :

Classes d'equivalences	**Valeur(s) possible(s) (montant)**
Montant inferieur au range [0, 1 000 000]	-1000
Montant dans les frontières inferieur du range [0, 1 000 000]	-0.01 , 0 , 0.01 , etc..
Montant dans le range [0, 1 000 000]	500
Montant dans les frontières superieur du range [0, 1 000 000]	1 000 000 ,1000 001
Montant superieur au range [0, 1 000 000]	2 000 000

Voici le tableau des classes d'équivalences pour les variables « curr1 » et « curr2 » :

Classes d'equivalences	**Valeurs possible (curr1,curr2)**
curr1 et curr2 EXISTENT dans « currenciesSupported »	("US Dollar", "Euro")
curr1 EXISTE dans « currenciesSupported » mais pas curr2	("US Dollar", "Algerian dinar")
curr2 EXISTE dans « currenciesSupported » mais pas curr1	("Algerian dinar", "US Dollar")
curr1 et curr2 N'EXISTENT PAS dans « currenciesSupported »	("Algerian dinar", "Algerian dinar")
curr1 et curr2 sont indentiques	("US Dollar", "US Dollar")

Maintenant, en faisant le produit cartésien des valeurs deux ensembles des classes d'équivalences trouvées (**amount** x **(curr1,curr2)**) en haut, on peut obtenir un jeu de test qui couvre toutes les valeurs frontières de la fonction :

Note : on doit ajouter la constante currenciesSupported à tous les éléments de notre jeu de tests et réorganiser aussi l'ordre des variables pour qu'il soit un format du genre –> (curr1, curr2, currenciesSupported, montant)

Jeux de test pour la méthode « currencyConverter.MainWindow.convert(String curr1, String curr2, currenciesSupported, montant) :

Tests	Résultat attendu	Raison possible d'un résultat invalide (-1)
("US Dollar", "Euro", currenciesSupported, -1000)	-1	Montant négatif situé en dehors du range valide
("US Dollar", "Algerian dinar", currenciesSupported, -1000)	-1	Montant négatif situé en dehors du range valide
("Algerian dinar", "US Dollar", currenciesSupported, -1000)	-1	Montant négatif situé en dehors du range valide
("Algerian dinar", "Algerian dinar", currenciesSupported, -1000)	-1	Montant négatif situé en dehors du range valide
("US Dollar", "US Dollar", currenciesSupported, -1000)	-1	Montant négatif situé en dehors du range valide
("US Dollar", "Euro", currenciesSupported, -0.01)	-1	Montant négatif situé en dehors du range valide
("US Dollar", "Algerian dinar", currenciesSupported, -0.01)	-1	Montant négatif situé en dehors du range valide
("Algerian dinar", "US Dollar", currenciesSupported, -0.01)	-1	Montant négatif situé en dehors du range valide
("Algerian dinar", "Algerian dinar", currenciesSupported, -0.01)	-1	Montant négatif situé en dehors du range valide
("US Dollar", "US Dollar", currenciesSupported, -0.01)	-1	Montant négatif situé en dehors du range valide
("US Dollar", "Euro", currenciesSupported ,500)	465	
("US Dollar", "Algerian dinar", currenciesSupported, 500)	-1	D'après la spécification, "Algerian dinar" (curr2) n'est pas supporté dans le code.
("Algerian dinar", "US Dollar", currenciesSupported, 500)	-1	D'après la spécification, "Algerian dinar" (curr1) n'est pas supporté dans le code.
("Algerian dinar", "Algerian dinar", currenciesSupported, 500)	-1	D'après la spécification, "Algerian dinar" (curr1 et curr2) n'est pas supporté dans le code.
("US Dollar", "US Dollar", currenciesSupported ,500)	500	
("US Dollar", "Euro", currenciesSupported ,1 000 000)	930 000	
("US Dollar", "Algerian dinar", currenciesSupported, 1 000 000)	-1	D'après la spécification, "Algerian dinar" (curr2) n'est pas supporté dans le code.
("Algerian dinar", "US Dollar", currenciesSupported, 1 000 000)	-1	D'après la spécification, "Algerian dinar" (curr1) n'est pas supporté dans le code.
("Algerian dinar", "Algerian dinar", currenciesSupported, 1 000 000)	-1	D'après la spécification, "Algerian dinar" (curr1 et curr2) n'est pas supporté dans le code.
("US Dollar", "US Dollar", currenciesSupported ,1 000 000)	1 000 000	
("US Dollar", "Euro", currenciesSupported ,2 000 000)	1	Montant trop large situé en dehors du range valide
("US Dollar", "Algerian dinar", currenciesSupported, 2 000 000)	-1	Montant trop large situé en dehors du range valide
("Algerian dinar", "US Dollar", currenciesSupported, 2 000 000)	-1	Montant trop large situé en dehors du range valide
("Algerian dinar", "Algerian dinar", currenciesSupported, 2 000 000)	-1	Montant trop large situé en dehors du range valide
("US Dollar", "US Dollar", currenciesSupported ,2 000 000)	-1	Montant trop large situé en dehors du range valide

Tests pour la méthode « currencyConverter. Currency.convert(Double montant, Double tauxEchange)»

Tout d'abord, voilà des définitions des argument de cette méthode :

- montant : Le montant (double) dont on souhaite convertir d'une monnaie à une autre.
- tauxEchange : Le taux d'échange (double) permettant de passer d'une monnaie à une autre.s

On peut les partitionner en différentes classes d'équivalences pertinentes :
Note : Pour les classes d'équivalences du **« montant »**, on les partitionne en fonction de leur range et des valeurs frontières de ce range car, comme on l'a vu en cours, c'est à ces points là qu'on peut trouver le plus des failles. Pour les classes d'équivalences de **« tauxEchange »**, on les sépare en trois classes dont une seule est valide, le taux d'échange positif. En effet, pour obtenir un montant positif d'une conversion de monnaie en fonction d'un taux d'échange, il faudrait aussi que le taux d'échange soit positif. Un taux d'échange dans ce cas-là ne peut être ni négatif ni nul (0), car un montant ne peut être négatif.

Voila le tableau des classes d'équivalences pour la variable « Montant » :

Classes d'equivalences	Valeur(s) possible(s) (montant)
Montant inferieur au range [0, 1 000 000]	-1000
Montant dans les frontières inferieur du range [0, 1 000 000]	-0.01 , 0 , 0.01 , etc..
Montant dans le range [0, 1 000 000]	500
Montant dans les frontières superieur du range [0, 1 000 000]	1 000 000 ,1000 001, etc..
Montant superieur au range [0, 1 000 000]	2 000 000

Voici le tableau des classes d'équivalences pour la « tauxEchange » :

Classes d'equivalences	Valeurs possible (tauxEchange)
Un taux d'échange négatif	-2
Un taux d'échange nul	0
Un taux d'échange positif	1.5

Maintenant, en faisant le produit cartésien des valeurs des deux ensembles de classes d'équivalences trouvés (**amount** x **(curr1,curr2)**) en haut, on peut obtenir un jeu de tests qui couvrent toutes les valeurs frontières de la fonction :

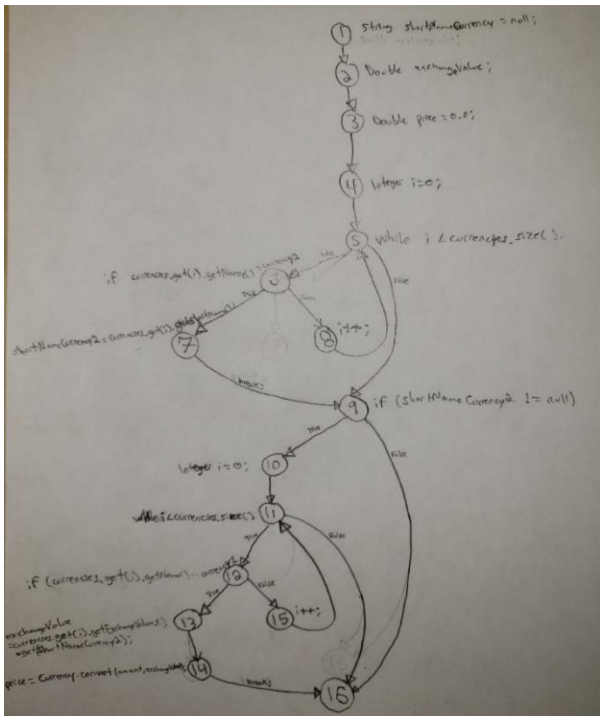
Jeux de test pour la méthode « currencyConverter. Currency.convert(Double montant, Double tauxEchange) » :

Tests	Résultat attendu	Raison possible d'un résultat invalide (-1)
(-1000, -2)	-1	Montant négatif situé en dehors du range valide et taux d'échange négatif
(-1000 , 0)	-1	Montant négatif situé en dehors du range valide et taux d'échange nulle
(-1000 , 1.5)	-1	Montant négatif situé en dehors du range valide
(-0.01, -2)	-1	Montant négatif situé en dehors du range valide et taux d'échange négatif
(-0.01 , 0)	-1	Montant négatif situé en dehors du range valide et taux d'échange nulle
(-0.01 , 1.5)	-1	Montant négatif situé en dehors du range valide
(500, -2)	-1	Pas possible d'avoir un taux d'échange négatif
(500 , 0)	-1	Pas possible d'avoir un taux d'échange nul
(500 , 1.5)	750	
(1 000 000, -2)	-1	Pas possible d'avoir un taux d'échange négatif
(1 000 000, 0)	-1	Pas possible d'avoir un taux d'échange nul
(1 000 000, 1.5)	1 500 000	
(2 000 000, -2)	-1	Montant trop large en dehors du range et pas possible d'avoir un taux d'échange négatif
(2 000 000, 0)	-1	Montant trop large en dehors du range et pas possible d'avoir un taux d'échange nul
(2 000 000, 1.5)	750	Montant trop large en dehors du range

Partie 2

Methode currencyConverter.MainWindow.convert(String currency1, String currency2, ArrayList<Currency> currencies, Double amount) Pour cette méthode, nous appliquons le critère de couverture des chemins indépendants du graphe de flot de contrôle. Nous n'avons pas besoin d'appliquer le critère de couverture des conditions, car il n'y a pas de conditions composées.

Traçons le graphe de flot de contrôle de la méthode en considérant les blocs **for (Integer i = 0; i < currencies.size(); i++)** comme des blocs **while** avec trois instructions séparées. **Integer i = 0;**, puis **while (i < currencies.size())**, puis **i++;** comme dernière instruction de la boucle :



Il y a deux **while** (en fait deux **for**) et trois if. La complexité cyclomatique est 6. V(G) = 2 + 3 + 1 = 6. Cela borne le nombre de chemins pour le critère de couverture des chemins indépendants. Représentons les **ArrayList** par des crochets fléchés <a,b,c>. **ArrayList** sera représenté par <>. Pour rouler une boucle for (représentée comme while) 0 fois, la taille de **ArrayList currencies** doit être 0. Pour rouler 1 fois, la taille doit être 1. Ainsi de suite.

Chemin 1 : 1-2-3-4-5-9-16. Pour ce chemin, **(i < currencies.size())** est **False** la première fois et **if (shortNameCurrency2 != null)** est **False**. currencies doit donc être vide.
Jeu de test : USD, EUR, <>, 21
Résultat attendu : 0.0, car **price** ne sera jamais modifié (price est modifié au nœud 14).
Résultat obtenu : 0.0, tel qu'attendu.

Chemin 2 : 1-2-3-4-5-6-8-5-9-16. Pour ce chemin, **(i < currencies.size())** est **True** la première fois et **False** la deuxième fois (on roule la boucle une fois), **if (currencies.get(i).getName() == currency2)** est **False** et **if (shortNameCurrency2 != null)** est **False**. **currencies** ne doit pas contenir les valeurs de **currency1** et **currency2** et doit avoir une taille de 1.
Jeu de test : USD, EUR, <CHF>, 42536
Résultat attendu : 0.0, car **price** ne sera jamais modifié (price est modifié au nœud 14).
Résultat obtenu : 0.0, tel qu'attendu.

Chemin 3 : 1-2-3-4-5-6-7-9-10-11-12-15-11-16. Pour ce chemin, **(i < currencies.size())** est **True** la première fois et **False** la deuxième fois dans le cas de la deuxième boucle (la première boucle roule une seule fois car puis un **break;** cause la sortie de la boucle), i**f (currencies.get(i).getName() == currency2)** est **False**, **if (shortNameCurrency2 != null)** est **True** et if **(currencies.get(i).getName() == currency1)** est **False**. Ainsi, **currencies** doit contenir la valeur de **currency2**, mais pas de **currency1**, et avoir une taille de 1.

Jeu de test : USD, EUR, <EUR>, 12
Résultat attendu : 0.0, car **price** ne sera jamais modifié (price est modifié au nœud 14).
Résultat obtenu : 0.0, tel qu'attendu.

Chemin 4 : 1-2-3-4-5-6-7-9-10-11-12-13-14-16. Ce chemin est comme le chemin 3, mais **if (currencies.get(i).getName() == currency1)** est **True** plutôt que **False**. Les deux boucles ne roulent qu'une seule fois, car un **break;** cause une sortie de boucle dans les deux cas. Ainsi, **currencies** doit contenir la valeur de **currency2** en première position pour que la première boucle roule une seule fois, puis la valeur de **currency1** en deuxième position. La taille doit être 2.
Jeu de test : USD, EUR, <EUR, USD>, 1000
Résultat attendu : 930, car **price est modifié au nœud 14 et la valeur de exchangeValue** de USD pour EUR est **0.93**.
Résultat obtenu : 930, tel qu'attendu.

La représentation vectorielle des chemins est :
Chemin 1 = {1,1,1,1,1,0,0,0,1,0,0,0,0,0,0,1}
Chemin 2 = {1,1,1,1,1,1,0,1,1,0,0,0,0,0,0,1}
Chemin 3 = {1,1,1,1,1,1,1,0,1,1,1,1,0,0,1,1}
Chemin 4 = {1,1,1,1,1,1,1,0,1,1,1,1,1,1,0,1}

Si le rang d'une matrice est égal au nombre de ses colonnes, on dit qu'elle est "de plein rang" et cela implique que toutes ses colonnes sont linéairement indépendantes. En formant une matrice dont les colonnes sont les vecteurs de nos 4 chemins et en utilisant Wolfram Alpha pour en calculer le rang, on obtient bien un rang de 4. Les chemins sont donc linéairement indépendants.

Ainsi, tous les 1-chemins du graphe sont parcourus au moins une fois. Nous considérons qu'il n'est pas nécessaire d'appliquer le critère de couverture des i-chemins, car nos deux boucles sont des boucles for, avec un **i=0**, modifié seulement par incrémentation **i++,** donc qui n'a pas d'autres opérations qui agissent dessus tel que des divisions, etc., et les boucles n'ont pas de borne supérieure fixe **n**, mais plutôt une borne qui varie en fonction de la taille du paramètre **currencies**, que l'on contrôle.

---

currencyConverter.Currency.convert(Double amount, Double exchangeValue)

Le seul critère dont l'application a du sens dans ce cas-ci est le critère de couverture des instructions. Les autres critères ne sont pas nécessaires (est sont en fait déjà couverts la couverture des instructions), car il n'y a pas de **while**, **for**, **if**, etc. Ainsi, le graphe de flot de contrôle aurait 3 nœuds et deux arcs, donc un seul chemin possible. On veut que chaque instruction soit exécutée au moins une fois. En fait, elles le seront peu importe la valeur de **amount** et **exchangeValue**. Nous pouvons donc choisir n'importe quelles valeurs.

Jeu de test : 100, 1.4
Résultat attendu : 140, car 100 * 1.4 = 140 et arrondi à deux décimales demeure 140.
Résultat obtenu : 140, tel qu'attendu.

Tel que mentionnée dans la partie 1, les noms de la spécification ne fonctionnent pas, on a donc utilisé la version longue des noms), mais dans les jeux de test, on écrit la version courte. On sait maintenant, en ayant accès au code, que les versions longues des noms équivalent aux **String shortName** et les version courtes équivalent aux **String name**. Pour le test boite blanche, nous avons accès au code et pouvons remarquer qu'en utilisant **String shortName** plutôt que **String name** pour **String currency1** et **String currency2**, il y a un problème, car if **(currencies.get(i).getName() == currency2)** (et même chose pour **currency1**) compare le **String name** des éléments de **currencies** à la valeur de **currency2** qui, selon la spécification, devrait utiliser le **shortName**. Il faudrait donc modifier le code pour permettre la comparaison des **shortName**.