IFT-3913 Qualité du logiciel et métriques
TP 4 Rapport

1) Test boîte noir

Devises

R : D1{USD}; D2{CAD}; D3{GBP}; D4{EUR}; D5{CHF}; D6{AUD}; D7{INVALIDE}

Jeu de test {USD, CAD, GBP, EUR, CHF, CHF, INVALIDE}

Pour la valeur “INVALIDE”, nous-nous attendons que le système rejette la requête avec un message d’erreur expliquant que la devise entrée n’est pas valide.

Montants

R: D1{0 <= d <= 1 000 000}, D2{d < 0}, D3{d > 1 000 000}

Jeu de test {-9 000, -1, 0, 500 000.5, 1 000 000, 1 000 001, 9 000 000}

Pour les valeurs invalides -9 000, -1, 1 000 001 et 9 000 000, nous-nous attendons que le système rejette la requête avec un message d’erreur expliquant que le montant entré n’est pas valide. Idéalement, le message pourrait être encore plus spécifique et spécifier si le montant entré est inférieur ou supérieur à l’intervalle valide.

2) Test boîte blanche

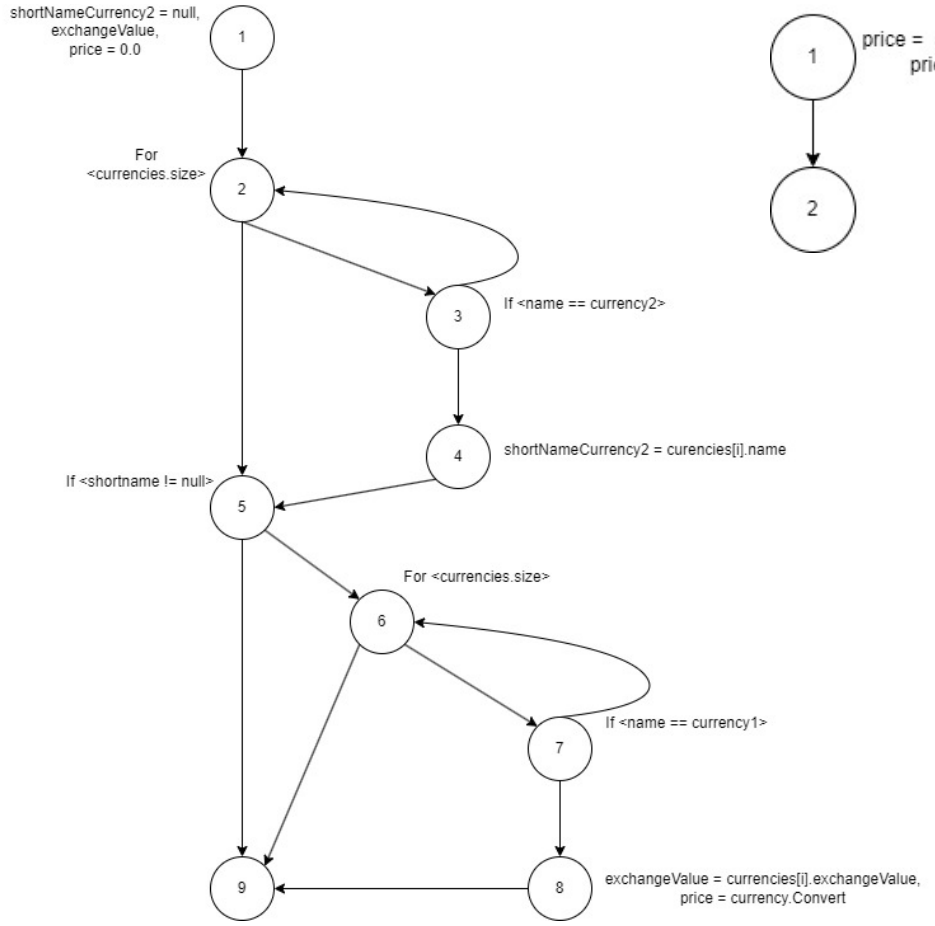
a) Couverture des instructions:

Méthode currencyConverter.MainWindow.convert:
Pour faire en sorte que toute les lignes de code soit exécutées, il faut tout simplement passer 2 devises qui se trouve dans la liste de devises passé à la fonction.
Jeu de test:
{
(USD, EUR, [Currency.USD, Currency.EUR], 50)
}

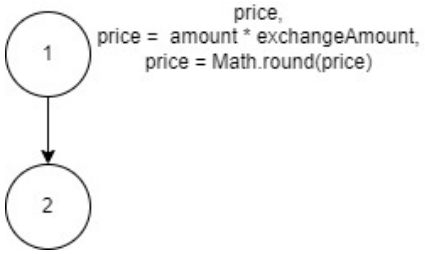
Méthode currencyConverter.Currency.convert
La fonction contient aucune logique conditionnelle, et donc, n’importe quel jeu de test va passer par tout le code de la fonction.
Jeu de test: {(100, 3)}

Pour les prochaines sections, nous allons faire référence aux graphes de flux de contrôle ci-dessous:

currencyConverter.MainWindow.convert



currencyConverter.Currency.convert



b) Critère de couverture du graphe de flux de contrôle:

Méthode currencyConverter.MainWindow.convert

En analysant le graphe, nous observons que nous aurons besoin de 3 tests pour traverser tous les arcs au moins une fois.

Jeu de test: {
(USD, EUR, [], 50),
(USD, EUR, [Currency.GBP, Currency.USD, Currency.EUR], 50),
(USD, EUR, [Currency.EUR], 50)
}

Méthode currencyConverter.Currency.convert

Ce graphe est trivial et n'importe quel jeu de test va traverser tout le graphe.

Jeu de test: {
(100, 3)
}

c) Critère de couverture des chemins indépendants:

Méthode currencyConverter.MainWindow.convert

Selon la complexité cyclomatique du graphe il y a 6 chemins indépendants dans ce graphe.

Voici les chemins:

(1,2,5,9)

(1,2,3,4,5,9)

(1,2,3,4,5,6,9) (Chemin impossible, car currencies doit être vide et non vide en même temps.)

(1,2,3,4,5,6,7,8,9) (Chemin impossible, car on s'attend à ce que currency1 soit à la première position de currencies, en même temps que currency2 à la même position.)

(1,2,3,2,3,4,5,6,9)

(1,2,3,2,3,4,5,6,7,6,9)

Où la répétition des arrêts (2,3) et (6,7) pourrait être modélisée par un autre état.

Jeu de test:
{
(USD, EUR, [], 50),
(USD, EUR, [Currency.EUR où name == null], 50),
(USD, EUR, [Currency.USD, Currency.EUR], 50),
(USD, EUR, [Currency.GBP, Currency.USD, Currency.EUR], 50),
}

Méthode currencyConverter.Currency.convert

Ce graphe a une complexité cyclomatique de 1 et donc on devrait avoir 1 chemin indépendant à tester.

Jeu de test: {
(100, 3)
}

d) Critère de couverture des conditions

En regardant le code des deux méthodes, aucune des deux contiennent des if ou des while à plusieurs conditions à la fois, donc ce critère ne s'applique pas vraiment avec le code fourni. Utiliser le critère en b) revient exactement au même pour le contexte suivant.

e) Critère de couverture des i-chemins

Méthode currencyConverter.MainWindow.convert

Nous avons décidé de choisir n = 4 pour notre borne supérieure d'itération.
Nous allons traiter les deux boucles de manière indépendante.

Boucle 1:

Jeu de test:

{
(USD, EUR, [], 50) // 0 itération,
(USD, EUR, [Currency.EUR], 50) // 1 itération,
(USD, EUR, [Currency.USD, Currency.EUR], 50) // 2 itérations
(USD, EUR, [Currency.USD, Currency.GBP, Currency.EUR], 50) // 3 itérations
(USD, EUR, [Currency.USD, Currency.GBP, Currency.CHF,Currency.EUR], 50) // 4 itérations
(USD, EUR, [Currency.USD, Currency.GBP, Currency.CHF, Currency.CNY, Currency.EUR], 50) // 5 itérations
}

Boucle 2:

Jeu de test:

{
(EUR, USD, [], 50) // 0 itération,
(EUR, USD, [Currency.EUR], 50) // 1 itération,
(EUR, USD, [Currency.USD, Currency.EUR], 50) // 2 itérations
(EUR, USD, [Currency.USD, Currency.GBP, Currency.EUR], 50) // 3 itérations
(EUR, USD, [Currency.USD, Currency.GBP, Currency.AUD,Currency.EUR], 50) // 4 itérations
(EUR, USD, [Currency.USD, Currency.GBP, Currency.CHF, Currency.CNY, Currency.EUR], 50) // 5 itérations
}

Méthode currencyConverter.Currency.convert

Il n'y a pas de boucle donc ce test ne s'applique pas.

Résultats

Pour la méthode currencyConverter.Currency.convert, étant donné que celle-ci était très simple, tous les tests passent. Mais pour la méthode currencyConverter.MainWindow.convert, certains tests ne passent pas, car le code de lance pas d'erreur lorsqu'il le devrait. Par exemple, lorsqu'une devise n'est pas trouvée, la conversion devient impossible et le système devrait refléter la situation selon-nous. Aussi, pour les tests boîte noir, le nom des devises dans la spécification ne correspondait pas aux noms des devises dans l'implémentation.

Pour les tests boîte blanches, nous avons remarqué qu'il y a beaucoup de redondance lorsqu'on utilise tous les critère pour tester une seule et unique méthode. Il également évident que, pour le cas particulier de ce projet, le critère des i-chemins est plutôt inutile, car le nombre d'itérations ne change absolument rien au résultat retourné. Nous croyons que le meilleur critère pour tester ce code était celui du critère des chemins indépendants.