IFT 3913 - Qualité du logiciel et métriques

TP 3 - TEST

Rapport de tests

BACH William 20127144

LUNICA Albert 20101201

Notre objectif est de tester les 2 méthodes “convert” provenant des classes *Currency* et *CurrencyConverter*.

Dans *CurrencyConverter,* on transmet à la fonction 2 chaines de caractère contenant le nom de chaque devise (avec la devise vers laquelle convertir en position 2), ainsi qu’une liste des différentes devises supportées et enfin le montant d’argent à convertir.

La classe *Currency* implémente le calcul de conversion qui consiste à multiplier le montant d’argent à convertir par le taux de change correspondant aux 2 devises (ex : convertir des Euros en Dollars -> EURUSD).

Tests à boîte noire

Voyons la sélection des tests a boite noire pour chaque approche différente (*Partition du domaine des entrées en classe d'équivalence* et *Analyse des valeurs frontières*) :

Partition du domaine des entrées en classe d'équivalence

La spécification du *CurrencyConverter* précise que seules ces devises sont supportées : {USD, CAD, GBP, EUR, CHF, CNY}.

Cette spécification nous permet de définir 4 classes d'équivalence pour les paramètres [currency1, currency2] qui sont des chaînes de caractère correspondant au nom de la

devise voulue. On considère le domaine des entrées Devises et P défini sur Dp = {USD, CAD, GBP, EUR, CHF, CNY}.

D1 = { currency1 **in** Dp AND currency2 **in** Dp } (devrait retourner un résultat valide).

D2 = { currency1 **in** Dp AND currency2 **not in** Dp } (devrait retourner un message d'erreur).

D3 = { currency1 **not in** Dp AND currency2 **in** Dp } (devrait retourner un message d'erreur).

D4 = {currency1 **not in** Dp AND currency2 **not in** Dp } (devrait retourner un message d'erreur).

Le paramètre [currencies] n'a pas besoin de ce test car elle constitue la représentation du domaine d'entrée (liste de devises, cad objets *Currency*).

Quant au dernier paramètre [amount], il est précisé que c'est un nombre dans le domaine D des Doubles non-négatifs (D= [0, 1.8 * 10^308]). Ce domaine est trop important pour pouvoir tester des valeurs supérieures à D mais nous pouvons établir tout de même 2 classes d'équivalence :

D1' = { amount **>=** 0 } (devrait retourner un résultat valide).

D2' = { amount **<** 0 } (devrait retourner un message d'erreur).

Ce partitionnement nous permet d'élaborer un jeu de test pour la méthode "convert" implémentée par *CurrencyConverter* :

T = { [currency1 : USD, currency2 : EUR, amount : 10],

[currency1 : CHF, currency2 : DEM, amount : 10],

[currency1 : DZD, currency2 : CNY, amount : 10],

[currency1 : AFN, currency2 : XAF, amount : 10], -----> Test de [currency1, currency2]

[currency1 : USD, currency2 : EUR, amount : -47],

[currency1 : USD, currency2 : EUR, amount : 36] } -------> Test de [amount]

La méthode “convert” implémentée dans *Currency* est différente car elle ne prend que 2 paramètres : des Doubles non-négatifs représentant le montant à convertir et le taux de change.

A la manière de la première méthode, établissons les classes d'équivalences :

D1 = { amount **>=** 0 AND exchangeValue **>=** 0 } (devrait retourner un résultat valide).

D2 = { amount **>=** 0 AND exchangeValue **<** 0 } (devrait retourner un message d’erreur).

D3 = { amount **<** 0 AND exchangeValue **>=** 0 } (devrait retourner un message d’erreur).

D4 = { amount **<** 0 AND exchangeValue **<** 0 } (devrait retourner un message d’erreur).

T2 = { [amount : 50, exchangeValue : 1.34],

[amount : 34, exchangeValue : -2],

[amount : -81, exchangeValue : 1.8],

[amount : -43, exchangeValue : -4] }

Interpretation des resultats

Selon les tests “testMainWindowConvert” (1, 2, 3 et 4) la fonction MainWindow.convert() ne respecte pas les specifications. Effectivement, la fonction ne gère pas les exceptions lorsque le montant entré est négatif et à la place elle retourne un montant négatif. De plus, lorsque la dévise ne se trouve pas dans la liste des devises reconnues par défaut, la fonction retourne un montant nul et n’envoye pas un message d’erreur.

Analyse des valeurs frontières

Étant donné que le domaine des entrées pour les paramètres [currency1, currency2] pour la méthode de *CurrencyConverter* est une liste non-ordonnée de valeurs, nous ne pouvons

pas trouver de “valeurs frontières" donc nous gardons les mêmes classes d'équivalence que pour l’approche de partition.

Le paramètre “amount” dans les classes d'équivalence D1 = {x >= 0 } et D2 = {x < 0} possède les valeurs frontière ‘0’ qui devrait retourner un résultat valide et ‘-0.1’ qui devrait retourner un message d’erreur. Nous rajoutons ces 2 cas dans le jeu de test final et nous obtenons :

T = { [currency1 : USD, currency2 : EUR, amount : 10],

[currency1 : CHF, currency2 : DEM, amount : 10],

[currency1 : DZD, currency2 : CNY, amount : 10],

[currency1 : AFN, currency2 : XAF, amount : 10], -----> Test de [currency1, currency2]

[currency1 : USD, currency2 : EUR, amount : -47],

[currency1 : USD, currency2 : EUR, amount : -0.1], *new test*

[currency1 : USD, currency2 : EUR, amount : 0], *new test*

[currency1 : USD, currency2 : EUR, amount : 36] } -------> Test de [amount]

Encore une fois, pour la méthode de *Currency* le jeu de test est similaire à celui de la première approche mais nous rajoutons des cas de test pour les valeurs frontière des paramètres [amount, exchangeValue] (= 0, =-0.1) :

T2 = { [amount : 50, exchangeValue : 1.34],

[amount : 0, exchangeValue : 1.34], *new test*

[amount : 50, exchangeValue : 0], *new test*

[amount : 34, exchangeValue : -2],

[amount : 34, exchangeValue : -0.1], *new test*

[amount : -81, exchangeValue : 1.8],

[amount : -0.1, exchangeValue : 1.8], *new test*

[amount : -43, exchangeValue : -4] }

Interpretation des resultats

Lors des tests, la fonction retourne un nombre positif à la limite positive inférieure et retourne un nombre négatif à la limite négative supérieure. Ceci est contre les spécifications, car lors de la limite negative supérieure, il faut envoyer un message d'erreur à la place.

Tests à boîte blanche

Pour les tests a boite blanche, nous allons produire des jeux de test uniquement pour la méthode de *CurrencyConverter*. En effet, la méthode de *Currency* ne contient aucune instruction conditionnelle et les tests structurels sont donc inutiles.

Nous pouvons également noter que dans le graphe, les instructions conditionnelles "for" consistent à parcourir la liste du paramètre "currencies", le domaine des entrées Dp pour les paramètres [currency1, currency2].

Couverture des instructions

Le programme de la classe *CurrencyConverter* contient 3 instructions conditionnelles "if". Si ces conditions sont respectées dans les 3 cas, le programme exécute chaque instruction.

Les 3 instructions "if" sont reliées à l'appartenance des paramètres [currency1, currency2] à la liste de devises [currencies]. Si les devises "currency1" et "currency2" sont référencées dans la liste "currencies", alors les instructions à l'intérieur du 'if' sont exécutées une unique fois.

Ainsi, si nous passons en paramètre des devises appartenant au domaine des entrées Dp (= {USD, CAD, GBP, EUR, CHF, CNY, JPY}), nous sommes certains que le programme va exécuter toutes les instructions.

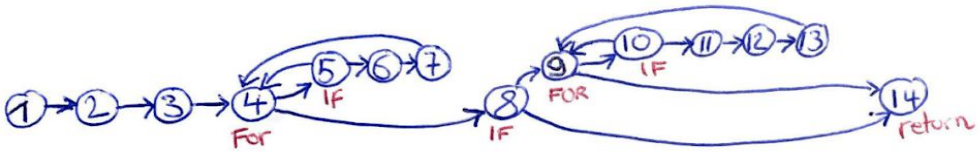
Afin de tester grâce à l'approche "Couverture des instructions", nous pouvons donc nous inspirer des classes d'équivalence produites par le partition des domaines des entrées dans la première partie :

T = { [currency1 : USD, currency2 : EUR, amount : 10], } ---- *toutes les instructions exécutées*

Interpretation des resultats

Couverture des arcs du graphe de flux de contrôle

Voici le graphe de flux de contrôle de la méthode de *CurrencyConverter*.



Pour chaque instruction "if", on teste le cas où la condition est vraie et celui où elle est fausse. Ici, l'instruction conditionnelle "for" n'est pas comptée car elle parcourt la liste [currencies].

Dans les 3 cas, la condition "if" n'est respectée que si les paramètres [currency1, currency2] font partie du domaine des entrees Dp. Nous pouvons donc nous inspirer des tests a boite noire pour trouver un jeu de test.

Si le premier "if" n'est pas respecté, alors le 3e "if" à l'intérieur de la 2e boucle ne sera pas exécuté car le 2e "if" n'est par conséquent pas respecté et on passe a la fin du programme.

T = { [currency1 : USD, currency2 : EUR, amount : 10], *les 3 'if' sont respectés*

[currency1 :DEM, currency2 : JPY, amount : 10], *if 1+2 respecte / if 3 non respecte*

[currency1 : AFN, currency2 : XAF, amount : 10] } *aucun if n'est respecté*

Interpretation des resultats

Les tests de couverture, montrent que peut importe la validité des trois conditions “if”, la fonction MainWindow.convert() retourne un nombre. Ceci n’est pas valide selon les spécifications, puisqu’il doit avoir un message d’erreur lorsque la dévise n’existe pas.

Couverture des chemins indépendants du graphe de flux de contrôle

Déterminons la complexité cyclomatique V(G) du graphe de flux de contrôle.

V(G) = #prédicats + 1

= 3 + 1

= 4 (on ne compte pas les prédicats “for” qui sont exécutés à chaque fois).

On sait que si la première condition “if” n’est pas respectée, alors les 2 prochains “if” ne peuvent pas être respectés.

Nous ne pouvons donc pas inclure un cas de test qui respecte les “if” 2 et 3 sans respecter le premier “if”.

Avec ces conditions, nous ne pouvons que produire 3 chemins indépendants dans le graphe (aucun if respecté, if 1+3 respectés, tous les if respectés) et nous pouvons utiliser les mêmes cas de test que dans la dernière approche.

T = { [currency1 : AFN, currency2 : XAF, amount : 10], *aucun if n’est respecté*

[currency1 : CHF, currency2 : DEM, amount : 10], *if 1+2 respecte / if 3 non respecte*

[currency1 : USD, currency2 : EUR, amount : 10] } *les 3 ‘if’ sont respectés*

Ces cas de test couvrent également tous les “i-chemins” possibles du graphe.

Nous n’avons donc pas besoin de documenter l’approche “Couverture des i-chemins du graphe” ni “Couverture des conditions” (nous n’avons pas de condition composée).

Interpretation des resultats

Comme il est évident de le voir, ces tests sont les mêmes tests que lors de la couverture des arcs du graphe de flux de contrôle, mais dans un ordre different. Vu que la cohérence des tests est indépendente de l'ordre de ceux-ci, il est valide de constater que la même intérprétation s'applique dans ce cas-ci. En résumé, ces tests montrent que la spécification de la fonction convert de MainWindow n'est pas respectée, car aucun message d'erreur n'est envoyé lorsqu'un des 'if' n'est pas respecté et la fonctionne retourne un nombre incohérent.

Conclusion

Pour conclure, les tests éffectués dénotent une mauvaise intérprétation des spécifications et une implémentation naïve de la fonction de conversion, sans ténir compte de aucune exception ou erreur probable de la part de l'utilisateur.

D'un côté, les tests à boîte noire nous montrent qu'il manque une gestion des erreurs de signe ou de fausse dévise. En effet, la fonction retourne zéro lorsqu'une mauvaise dévise est introduite, ce qui peut être mélangé avec un montant nul par l'utilisateur. De plus, elle retourne un nombre négatif lorsqu'un montant négatif est introduit. Ces deux rémarques indiquent une mauvaise précision de la fonction à cause de son manque de gérer les faux positifs et un non respect de la spécification exigée.

De l'autre côte, les tests à boîte blanche nous révélent qu'il manque une gestion des exceptions lorsque les conditions "if" ne sont pas satisfaites. Effectivement, lorsqu'une ou plusieurs de ces conditions ne sont pas satisfaites, la fonction continue son execution et retourne une valeur numérique illogique au contexte d'échange de dévises. Encore une fois, ceci indique un manque de précision et un non respect des spécifications exigées.

En fin de compte, pour corriger ces erreurs, une gestion des exceptions doit être introduite lorsque le signe du montant est négatif ou lorsque la dévise ne fait pas partie des dévises reconnues par le système. Ceci réduira le nombre de faux positifs et augmentera la précision de la convérsion.