Université de Montréal

Rapport TP4

Par Lilou Blanchette

Matricule : 20188851

et

William D'Anjou

Matricule: 20188213

Département d'informatique et recherche opérationnelle

Faculté des arts et des sciences

Travail présenté à Michalis Famelis

Dans le cadre du cours IFT-3913

IFT3913 - Qualité du logiciel et métriques

Rapport:

Tests pour la boîte noire

Nous avons créé des tests en utilisant l'approche de partition du domaine des entrées en classes d'équivalence et l'approche des valeurs frontières.

Nous avons déterminé, en regardant la méthode convert de la classe Currency.java, que le domaine D des valeurs d'entrée sont de type Double et P est défini sur [0, 1 000 000] (selon la spécification).

Nous avons donc définit 3 classes d'équivalence (comme dans les notes de cours):

$D_1 = \{0 \leq d \leq 1\,000\,000\}$

$D_2 = \{d < 0\}$

$D_3 = \{d > 1\,000\,000\}$

Donc, nous testons pour des valeurs qui sont dans l'intervalle et des valeurs qui sont supérieures et inférieures à l'intervalle.

Le jeu de test que nous avons choisi pour la méthode convert de la classe Currency.java est :

T = {-500, -1, 0, 500 000, 1 000 000, 1 000 001}

Donc nous pouvons voir que les valeurs -500 et -1 sont utilisées pour tester classes d'équivalences inférieures à l'intervalle (D_2) et pour tester les valeurs frontières. Ensuite, les valeurs 0 et 1 000 000 sont utilisées pour s'assurer que les valeurs à la limite de l'intervalle sont acceptées (D_1), ici nous parlons aussi des valeurs frontières. Après quoi, nous testons la valeur 500 000, qui vient de nouveau tester la classe d'équivalence D_1, pour s'assurer qu'une valeur aléatoire dans l'intervalle est acceptée. Cette valeur correspond aussi à une valeur typique pour une analyse des valeurs frontières. Pour terminer, nous testons la valeur 1 000 001, qui est supérieur à notre intervalle, afin de tester la classe d'équivalence D_3 et est une valeur frontière.

Pour tester cette méthode, nous utilisons l'hypothèse que si une "mauvaise" valeur est entrée dans la fonction, celle-ci devrait retourner la valeur nulle (null). Notre méthode *testCorrectAmount,* utilisée pour faire les tests de ce jeu de test, nous retourne 3 erreurs. En effet, les valeurs -500, -1 et 1 000 001 ne devraient rien nous retourner, car celles-ci sont hors de l'intervalle, selon la spécification qui nous a été donnée. Toutefois, c'est trois cas retourne une valeur autre que nulle, ce qui signifie que toutes les trois ont été acceptées par la méthode convert. Il y aurait donc une erreur d'implémentation si l'on souhaite faire respecter cette spécification.

Pour la méthode *convert* de la classe MainWindow, nous avons déterminé que le domaine D est tous les noms des devises mentionnées dans la spécification. Donc, nous avons
D = {US Dollar, Canadian Dollar, British Pound, Euro, Swiss Franc, Australian Dollar}

Pour tester la méthode *convert*, nous avons créée la méthode *currencyCheck* qui a pour but de créer toutes les différentes paires de devises possible, car la méthode *convert* prend comme paramètre deux devises pour convertir celles-ci. Cette méthode est donc utilisée afin de créer notre jeu de tests. C'est dans la méthode *testCurrencyConvertingList* que les paires de devises de *currencyCheck* sont utilisées et que l'on teste chacune d'entre elles. Pour les autres paramètres nécessaires à la méthode *convert*, nous envoyons toujours la même ArrayList de Currency, qui est celle initialisée grâce à Currency.init(). De plus, le montant envoyé n'est pas très important, car si les devices ne sont pas trouvés, alors le montant retourné sera égal à 0. On envoie donc toujours 1 comme montant.

Nous testons deux classes d'équivalences, soit des entrées valides et invalides.

L'ArrayList de *Currency* initialisé à partir de la méthode de la classe *Currency*, retourne des devices avec les pays suivants : {US Dollar, Euro, British Pound, Swiss Franc, Chinese Yuan Renminbi, Japanese Yen}. Toutefois selon la spécification, le programme devrait pouvoir accepter les devises suivantes:

{US Dollar, Canadian Dollar, British Pound, Euro, Swiss Franc, Australian Dollar}

Donc, puisque les devises {Canadian Dollar, Australian Dollar} ne sont pas dans celles initialisées dans l'ArrayList, les tests contenant une de ces devises ne réussiront pas. Cependant, si les paires de devises contiennent exclusivement ces devises :

{US Dollar, British Pound, Euro, Swiss Franc}, alors le test sera réussi.

Test pour la boîte blanche
Pour l'implémentation de test "boîte blanche", nous avons regardé de quelle façon les méthodes à tester étaient implémentées dans le programme fourni. Nous avons supposé qu'elle fonctionnait correctement et ne devrait jamais renvoyer d'erreur. Les tests que nous avons ajoutés servent à s'assurer que les méthodes conservent le même fonctionnement même après une mise à jour de celle-ci. Ainsi, pour les deux méthodes, tous nos tests de boîte blanche se font avec succès sans signaler d'erreurs.

Première méthode (*currencyConverter.MainWindow.convert()*)
Pour tester cette méthode, nous avons choisi différentes combinaisons d'entrées afin de couvrir chaque cas possible, et donc de tester tous les branchements réalisables.

1. On essaie toutes les combinaisons de conversion de devises contenues dans *Currency.init()*
2. On essaie une conversion lorsque la première devise n'est pas présente dans la liste des devises passée à la fonction.
3. On essaie une conversion lorsque la seconde devise n'est pas présente dans la liste des devises passée à la fonction
4. On essaie une conversion lorsque la liste de devises passée à la fonction est vide.
5. On essaie lorsque les deux devises ne sont pas présente dans la liste passée à la fonction

Dans les cas 2,3,4 et 5, on s'attend à recevoir une valeur de "0" peu importe le montant inscrit puisque le programme renvoie "0" comme valeur par défaut lorsque celui-ci n'arrive pas à trouver la pair de conversion.
Dans nos tests, nous avons utilisé la valeur "1" pour effectuer nos tests, la méthode *currencyConverter.MainWindow.convert()* appelle *currencyConverter.Currency.Convert()* lorsqu'elle tente d'effectuer une conversion, puisque nous testons également cette méthode, nous n'avons pas choisi de tester avec différentes valeurs de conversion.

Seconde méthode (*currencyConverter.Currency.Convert()*):
Nous avons choisi un ensemble de valeurs qui se situe dans les valeurs qu'un Double en java peut prendre. Ces valeurs peuvent être positives tout comme négatives puisque la méthode qui a été définie ne tient pas compte du signe du nombre qu'on entre en paramètre. Nous avons fait ce choix de valeurs puisque celui-ci couvre l'ensemble du domaine des Doubles, il n'y a pas de cas spéciaux dans cette méthode à prendre en compte. Il n'y a pas de branchement ni de boucle "for" ou "while" et donc à chaque appel de fonction, il n'y a qu'un seul chemin que l'on peut emprunter. Nous nous sommes également assuré de de grande valeurs (*ex: Double.MAX_VALUE * Double.MAX_VALUE*) fonctionne correctement.

Ensemble choisi:
{-500,-1, 0, 500000, 1000000, 1000001, 0.003, 5.4, 1.5987435, 6}