IFT 3913: Travail pratique 4

Pour le 8 Décembre 2023 à 23:59

Professeur: Michalis Famelis

Killian Gervais & Gabriel Hazan

Tâche 1

D'après la spécification donnée, on a deux types d'entrées : les devises (*currencies*) et les montants (*amounts*). Soit les devises du programme P_C défini sur {USD, CAD, GBP, EUR, CHF, AUD} avec C = Currencies = {USD, CAD, GBP, EUR, CHF, AUD, JPY, INR, NZD}, cette dernière pourrait être agrandie au besoin, et les montants du programme P_A défini sur [0, 1 000 000], l'intervalle des valeurs valides, avec A = Amounts = Réels positifs.

Il y a deux classes d'équivalences pour les devises :

- Les valeurs d'entrées valides : $C_1 = P_C$ = {USD, CAD, GBP, EUR, CHF, AUD}
- Les valeurs d'entrées invalides : C_2 = {JPY, IRN, NZD}

On choisit une valeur de chaque classe, pour pouvoir tester le cas ou l'une des devises est valide tandis que l'autre ne l'est pas, ainsi qu'une seconde valeur de devise valide pour tester les conversion de montant pour créer notre jeu de test T_C = {USD, EUR, JPY}.

Pour les montants, il y a trois classes d'équivalences, $a \in \mathbb{R}$:

- Les valeurs appartenant à P_A : $A_1 = \{0 \leq a \leq 1\ 000\ 000\}$
- Les valeurs invalides inférieures à P_A : $A_2 = \{a < 0\}$
- Les valeurs invalides supérieure à P_A : $A_3 = \{a > 1\ 000\ 000\}$

On choisit choisit une valeur "typique" dans chaque classe d'équivalence et plusieurs aux bornes de celles-ci pour créer le jeu de test $T_A = \{-2\ 222\ 222, -0.09, 0, 500\ 000, 1\ 000\ 000, 1\ 000\ 000.01, 2\ 222\ 222\}$.

Si une entrée est invalide, on s'attend à ce que le code ne s'arrête pas, il est donc capable de s'adapter à une mauvaise entrée fournie par l'utilisateur, que ce soit pour une devise ou un montant. On s'attend a ce que ce soit la classe MainWindow qui gère les entrées invalides. Puisque, selon la spécifiation, le type des données retournées par ces méthodes sont des double positifs on s'attend à recevoir "-1.0" comme résulat lors d'une entrée invalide.

Tâche 2

On va suivre la méthode des 5 critères de sélection de jeux de tests de la méthode boîte blanche dans le but de créer nos jeux de tests, lorsque ceux-ci sont applicables.

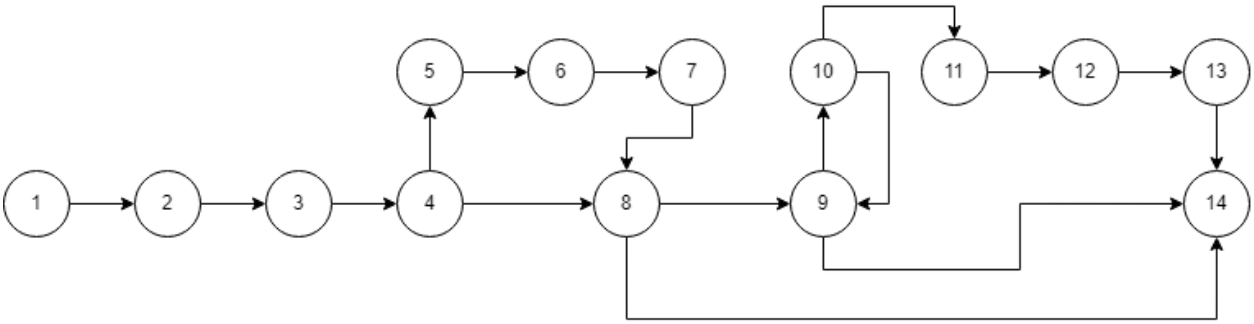
Commençons par la méthode *convert* de la classe MainWindow :

Couverture des instructions : Puisque les boucles utilisées sont des *for* il suffit que $currencies.size() > 0$ pour que les boucles soient exécutées au moins une fois, avec *currencies* la liste des devises acceptées. Les structures conditionnelles sont toutes des *if*, leur code est exécuté si leur condition est vraie. De plus, on note que le premier et le troisième sont dans les boucles *for*. Le code du premier *if* est exécuté quand *currency2* est dans la liste des devises, même chose pour le code du troisième mais avec *currency1* cette fois. Le code du dernier, qui est en fait le second *if* de la méthode, est exécuté lorsque le code du premier l'a été, ils partagent donc la même classe du domaine. D'après cela, on obtient la classe d'équivalence suivante :

$D_1 = \{(currency1, currency2) \mid currencies.size() > 0\ \&\ currency1, currency2 \in currencies\}$

Avec celle-ci, tout le code est exécuté au moins une fois. On peut donc prendre le jeu de test : {"US Dollar", "Euro"}, ces deux valeurs étant définies dans la classe Currency.

Couverture des arcs du graphe de flot de contrôle : On obtient le graphe de flot contrôle suivant



Les numéros correspondent à la ligne de code qui n'est pas un commentaire correspondante dans la méthode, 1 la première ligne après le bracket d'ouverture. On déjà énoncé les conditions qui font que les *if* et les boucles *for* sont exécutés ci-haut. On veut cependant ajouter les cas qui font que leur conditions sont fausses. D'où :
$D_1 = \{(currency1,\ currency2) \mid currencies.size() > 0\ \&\ currency1,\ currency2 \in currencies\}$
$D_2 = \{curr.1,\ curr.2) \mid currencies.size() = 0\}$
$D_3 = \{(currency1,\ currency2) | currencies.size() > 0,\ currency1 \notin currencies\ \&\ currency2 \in currencies\}$
$D_4 = \{(currency1,\ currency2) | currencies.size() > 0,\ currency1 \in currencies\ \&\ currency2 \notin currencies\}$

Il n'y a pas le cas où les deux devises ne sont pas dans la liste, puisque si la deuxième ne l'est pas alors la partie du code qui vérifie si la première se trouve dans la liste n'est pas exécutée. On peut donc prendre le jeu de test suivant : {{("US Dollar", "Euro") | $currencies = \emptyset$}, {("US Dollar", "Euro"), ("Australian Dollar", "US Dollar"), ("Euro", "US Dollar") | "US Dollar", "Euro" $\in$ *currencies*}}, puisque le dollar australien n'est pas défini dans Currency tandis que les deux autres le sont.

Couverture des chemins indépendants du graphe de flot de contrôle : La compléxité cyclomatique est $V(G) = e - n + 2 = 18 - 14 + 2 = 1 + d = 1 + 5 = 6$. On définit maintenant une base de 6 chemins indépendants du graphe de flot :

- 1-2-3-4-8-14 : la liste des devises est vide, $D_1 = \{(currency1,\ currency2) \mid currencies = \emptyset\}$
- 1-2-3-4-5-4-8-14 : $D_2 = \{(currency1,\ currency2) \mid currency2 \notin currencies,\ currency1 \in currencies\}$
- 1-2-3-4-5-6-7-8-9-10-9-14 : $D_3 = \{(curr.1,\ curr.2) \mid curr.2 \in currencies,\ curr.1 \notin currencies\}$
- 1-2-3-4-5-6-7-8-9-10-11-12-13-14 : $D_4 = \{(curr.1,\ curr.2) \mid curr.2,\ curr.1 \in currencies\}$

On a utilisé tous les arcs du graphe et il ne reste donc plus de chemins linéairement indépendants. Le jeu de test créer à l'étape précédente répondait déjà à ces critères, on le conserve donc tel quel.

Couverture des conditions : Il n'y a pas de conditions composées dans cette méthode. Ce critère n'est donc pas applicable.

Couverture des i-chemins : Les deux boucles sont des boucles simples. Il n'y a pas un grand intêret à faire l'ensemble des cas proposées par ce critère puisque le nombre d'itération n'impact pas le résultat. Les cas intéressant sont : on saute la boucle, ce qui se fait lorsque *currencies.size() = 0*, on effectue m itérations, ce qui arrive lorsque les devises sont dans la liste, la valeur de m n'est pas importante du moment que $0 < m < n$, et on effectue n itérations, ce qui "arrive" lorsque l'une des devises n'est pas dans la liste. Or, le jeux de test conçu répond déjà à tout cela. Notre jeu de test le plus couvrant est donc :
T = {{("US Dollar", "Euro") | $currencies = \emptyset$}, {("US Dollar", "Euro"), ("Australian Dollar", "US Dollar"), ("Euro", "US Dollar") | "US Dollar", "Euro" $\in$ *currencies*}}

Et maintenant, la méthode *convert* de la classe Currency :

Couverture des instructions : Tout ensemble de paires de nombres réels positifs va permettre à chaque instruction d'être exécutée au moins une fois. Un jeu de test valide est donc : $T = \{(1234.56, 789)\}$

Couverture des arcs du graphe de flot de contrôle : On obtient le graphe de flôt de controle suivant :

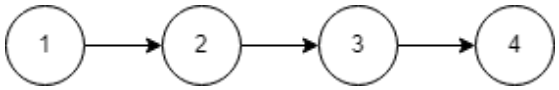


FIGURE 1 – Flot de contrôle de la méthode *convert* de la classe Currency

Le critère n'est pas pertient puisque l'on a une simple séquence d'instructions sans branchements de contrôles conditionnels. On peut garder le même jeu de test qu'à l'étape précédente.

Couverture des chemins indépendants du graphe de flot de contrôle : Ici encore, le critère est applicable mais peu pertinent puisque $V(G) = e - n + 2 = 3 - 4 + 2 = 1 + d = 1 + 0 = 1$. Il y a donc un unique chemin que l'on suit déjà avec le jeu de test que l'on a définit.

Couverture des conditions : Non applicable puisqu'il n'y a pas de conditions dans cette méthode.

Couverture des i-chemins : Non applicable puisqu'il n'y a pas de boucles dans cette méthode.

Tâche 3

On aurait du ajouter une seconde valeur de devise invalide afin de tester le cas de conversion une devise invalide vers une devise invalide

Pour la methode boite noire puisque l'on ne connait pas l'implementation des methodes on a assume que c'est dans la classe MainWindow que la validation des erreurs se fait pusique l'on s'attend a ce que celle-ci utilise la classe Currency. On s'attendait donc a ce que *Currency.convert* convertise toutes paires de nombres reels et c'est pourquoi on a utilise les meme valeurs numeriques que pour les tests de *MainWindow.convert*, soit le jeu de test T_A definit dans la tache 1, on ne se soucit pas des taux de changes puisqu'ils ne font pas partis de la specification. Les resultats des tests sont ce qui etait attendu. Pour tester *MainWindow.convert* on a utilise T_C et T_A puisque celle-ci requiere les deux types d'entrees enonce par la specification. On a convertie la version raccourci des noms des devises a leur noms coomplets dans les tests puisque l'on a vu que l'interface utilise ces derniers. On a assume que lorsqu'un montant ou une des devises ne repondait pas a la specification la methode retourne "-1.0". Ce n'est visiblement pas le cas ce qui fait que seuls les tests dont le montant et les deux devises sont valide n'echouent pas tandis que tous les autres test echouent. N'ayant pas acces au code on ne peut dire pourquoi le resultat attendu n'est pas celui que l'on veut avec cette methode. On a teste le comportement avec l'une des devises qui est invalide tandis que l'autre l'est (nous n'avons pas donne d'importance au montant puisque le comportement attendu ne change pas en fonction de celui-ci), les deux devises sont valides et le montant l'est aussi, les deux devises sont valides mais le montant ne l'est pas.

Pour la methode boite blanche on a utilise les jeux de tests que nous avons definit dans la tache 2. Pour *MainWindow.convert*, tous les criteres etaient applicables sauf celui des conditions puisqu'il n'y a pas de conditions composees dans la methode. Les cas couverts par nos tests sont : onsaute les boucles, on effectue m iterations, une des devises n'est pas dans la liste des devises. Les autres cas ne sont pas plus interessants pour nous (justification dans la tache 2). Cette fois-ci tous les tests passent puisque l'on sait maintenant

comment la methode fonctionne. Pour *Currency.convert*, seuls le critere de couverture des instructions est vraiment pertinent, puisque la methode est tres simple. On peut tout de meme applique la plupart sauf les criteres de couverture des conditions et des i-chemins (precision dans tache 2). On obtient du critere de couverture des instructions un jeu de test tres simple qui nous permet de confirmer que la methode marche comme attendue avec notre test. On a pas besoin de plus d'un cas puisque l'on sait que cela est generalisable a l'ensemble des paires de reels positifs.

La methode boite noire a l'incovenient que des tests que l'on pense bons ne le sont pas puisque l'on ne sait pas exactement comment le programme est code mais elle est rapide est facile. En comparaison, la methode boite blanche est beaucoup plus longue est complexe. Elle permet cependant d'obtenir une couverture maximale du code avec de meilleurs resultats et moins de tests a ecrire.