**IFT 3913 TP4**

Par Hugo Carrier - 20197563 et Maggie Robert - 20182443

16 décembre 2022

# Test boites noires

## Montant

Le domaine de montant est de 0 à 5000. Nous avons donc pris un nombre plus petit : -5000, un nombre dans le domaine :5000 et un nombre plus grand 50000.

Nous avons aussi tester les frontières 0 et 10 000 de plus que le cas critique d’un très petit chiffre : 0.000001;

## Devise

Les devises viennent en paire la devise de départ et la devise à convertie vers. Normalement nous devrions choisir une devise dans le domaine et programme et une devise hors du domaine du programme. Dans notre cas, nous avons fait la combinaison entre les différentes possibilités. D’abord nous avons choisi une des devises valide : CAD et USD. Nous avons ensuite choisi de devises hors programme : JPY (le yen japonais) et le BZD (monnaies du Brésil). Nous avons formé différentes combinaisons. [Valide, Valide; Valide, Non valide; Non Valide, Valide; Non Valide, Non Valide]. Nous avons ajouté la combinaisons CAD vers CAD afin de tester une autre combinaison valide mais qui est un cas limite et aussi un cas avec une devise qui n’existe pas de tout.

## Tests

Nous avons donc cas 13 cas. Nous avons tous les montant tester avec le CAD USD (donc six tests) puis cinq autres tests pour tester les devises avec un montant valide.

Nous attendons que la fonction *convert*() retourne seulement cinq valeurs valides. Le changement de CAD à USD avec un montant de 5000, CAD à CAD avec un montant de 5000 et les 3 tests avec les frontières. Seulement ces deux cas respect les spécifications. Les autres tests ont des valeurs qui ne respecte les spécifications. Ainsi, on s’attend à un retour d’exception. Si on n’a pas d’exception, cela veut dire que les spécifications ne sont pas respectées et qu’il faut donc revoir le code ou les spécifications. Même si le montant a été bien convertie, il faut quand même renvoyer l’erreur. En entreprise, les spécifications sont très importantes car ils peuvent parfois cacher de l’information. Par exemple. Si avoir un montant négatif est impossible, on veut recevoir une erreur et on ne veut surtout pas passer cette valeur dans le système.

## Résultats des tests

Tristement, les tests n’ont pas tous passé. La méthode répond plus large que les spécifications. C’est-à-dire, selon nos résultats obtenus, elle prend en compte plus de devises que prévu, accepte les nombres négatifs et les nombres plus grand que 10 000.

Lorsqu’on est dans les spécifications, elle envoie les bonnes réponses, même pour les frontières.

# Tests boites blanches

### 2.1 Critère de couverture des instructions

Pour le critère de couverture des instructions, on voit que les tests déjà écrits dans la partie précédente couvrent tous les instructions de la fonction *convert*(). Cependant, pour les cas où on lance une exception dans la fonction, nous pouvons être plus précis en prenant compte de la structure du code. Alors nous avons ajouté un test *whenGivenNonExistentCurrency\_thenThrowsCorrectException*() qui test si le type d’exception lancé est le type d’exception spécifié par le code est si le message d’erreur est aussi celui spécifié par le code. Ce test, ainsi que le test *whenConversionIsValid\_thenReturnsCorrectConversion*() défini dans la partie précédente couvrent toutes les instructions de la fonction *convert*().

**2.2 Autres critères**

A picture containing diagram

Description automatically generated

Figure 1: GFC de la fonction convert() avec les lignes de code

Comme nous pouvons voir dans Figure 1, le GFC de la fonction *convert*() n’a aucune boucle, alors le critère de couverture des i-chemins n’a pas du sens pour cette fonction. Puisque *convert*() est très simple, les critères de couverture des arcs de GFC, des chemins indépendants de GFC et des conditions aboutissent tous aux mêmes chemins à tester : (8, 9-10) et (8, 11, 12, 13). En analysant ces deux chemins, ils peuvent être couvrir aussi par les deux tests mentionnés en 2.1.

**2.3 Résultats des tests**

Les deux tests passent, qui indique que le code de la fonction est exécuté comme attendu étant donnée sa structure.

# Conclusion

Les tests boites noires nous permet de tester si une fonction répond correctement à sa spécification et les tests boites blanches nous permet de vérifier que le code fait ce qu’on pense qu’il fait selon sa structure. Avec la fonction testée *convert*() comme exemple, nous pouvons voir que pour les petites fonctions où il y a peu d’instructions, les tests boites noire et blanches souvent se chevauchent. Ceci signifie qu'il faut écrire moins de tests pour obtenir une couverture de code adéquate et démontre une autre raison pour laquelle il est préférable de coder des fonctions courtes et précises en Java.