Dans le cadre de ce travail, il a été demandé d’utiliser des techniques de tests dites à « [petite échelle](https://studium.umontreal.ca/pluginfile.php/7635827/mod_resource/content/0/Chap8_test.pdf)». Pour déterminer quels jeux de tests à utiliser, l’emphase est mise sur les tests unitaires, ceux-ci sont regroupés en deux catégories : tests de la boite noire et blanche. La distinction entre les deux est remarquable par le fait que les tests de la boite noire se concentrent uniquement aux fonctionnements du programme selon sa spécification, sans se soucier des détails structurels, donc chercher à déterminer si le logiciel fait ce qui est attendu de lui. Contrairement à la boite noire, les tests de la boite blanche se concentrent surtout sur la structure du programme, les détails de sa conception et ce que le programme fait. Il est important de souligner certaines failles de ces deux catégories, mais aussi la raison pour laquelle les deux se complètent l’une et l’autre. En effet, avec la boite noire (BN), il est difficile de détecter où les erreurs sont situées tandis qu’avec la boite blanche (BB), elles sont détectables. En revanche, les omissions ne sont pas détectées dans la BB, mais le sont par la BN. Plusieurs méthodes sont à disposition afin de définir les jeux de tests a utilise dans chacun des cas, pour ce qui est de la boite noire il y a : «

1. La partition des domaines d’entrées en classes d’équivalences
2. Analyse des valeurs frontières » ([chap8](https://studium.umontreal.ca/pluginfile.php/7635827/mod_resource/content/0/Chap8_test.pdf), p36)

Pour ce qui est de la boite blanche il y a 5 conditions/critères soit : «

1. Critère de couverture des instructions
2. Critère de couverture des arcs du graphe de flot de contrôle
3. Critère de couverture des chemins indépendants du graphe de flot de contrôle
4. Critère de couverture des conditions
5. Critère de couverture des i-chemins » ([chap8](https://studium.umontreal.ca/pluginfile.php/7635827/mod_resource/content/0/Chap8_test.pdf), p47)

Des explications détaillées seront abordées dans ce qui suit.

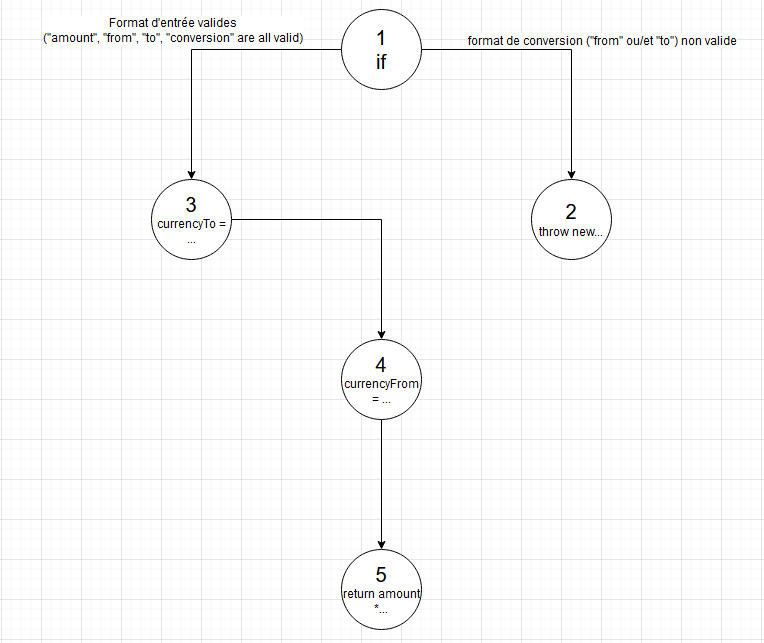
1. Choix des cas de tests :
   1. Cas de tests de la boite noire :

Pour les choix des tests la partition du domaine des entrées en classes d’équivalence a été utilise ainsi que l’analyse des valeurs frontières. Deux domaines ont été identifiés. Le premier implique les devises et le second les valeurs des devises.

* + 1. Pour les devises on s’est assurée que la conversion se faisait à partir de n’importe quelle devise vers une autre devise. Donc, pour N devises il y a N-1 conversion de devises à tester. Ainsi, N\*(N-1) cas de tests ont été effectués.
    2. Pour les valeurs des devises, on s’est assurées de tester des conversions se situant entre 0-10000. Nous avons également testé pour des valeurs >10000 et <0. On peut donc identifier 3 classes d’équivalences pour les tests impliquant la valeur des devises.

Pour ce qui est de l’analyse des valeurs frontières on a vérifié les cas ou la somme est supérieur à 10000, inférieur à 0 et égale à 0, une erreur va être détectée. Il y a aussi les cas ou on essaye de convertir des sommes d’une devise qui ne figure pas parmi les 6 choix, ceux-ci ont été traite.

* 1. Cas de tests de la boite blanche :
     1. Pour le choix de test en ce qui concerne la couverture des instructions, nous avons choisis de faire 2 tests. L’un (insideIf() ) qui rentrera dans la condition « if » et l’autre qui n’y rentrera pas (notInsideIf() ). Il sera ainsi assuré que chaque ligne de code sera exécutée au moins 1 foi lors de ces 2 tests.



* + 1. Pour la couverture des arcs. En faisant le graph de flot de contrôle (GFC) ci-dessus, on remarque qu’encore une fois, il suffit de s’assurer de faire 2 tests, l’un qui rentre dans le « if » ( insideIf2() ) et l’autre qu’y n’y rentrera pas ( notInsideIf2() ) afin de parcourir tous les arcs du graphique. Bien sûr, on peut aussi très bien se contenter des tests de la couverture des instructions car dans notre cas précis, ça reviendrait au même. La seule raison pourquoi nous avons créé de nouveaux tests et pour bien séparer les parties de la boite blanche et se rendre compte que ça reviendrait à faire des tests « identiques » à la fin.
    2. Pour la couverture des chemins indépendants du GFC, nous avons trouvé 2 chemins indépendant : 1-2 et 1-3-4-5 (selon le GFC). Ces 2 chemins reviennent à faire 2 tests, encore une fois, où l’un rentrera dans le « if » et l’autre non. Encore une fois, il est très possible de juste rouler les testes de la couverture des instructions.
    3. Pour la couverture des conditions, il faillait tester tous les cas de la condition « if » que nous avions. Nous nous sommes donc retrouvés à tester le « if » lorsque :
       1. Les 2 valeurs sont valides, ce qui donne

if(!Vrai || !Vrai) == if(Faux || Faux).

* + - 1. La valeur « to » n’est pas valide ce qui donne

if(!Vrai || !Faux) == if(Faux || Vrai)

* + - 1. La valeur « from » n’est pas valide, ce qui donne

if(!Faux || !Vrai) = if(Vrai || Faux)

* + - 1. Les 2 valeurs ne sont pas valides, ce qui donne

if(!Faux || !Faux) == if(Vrai || Vrai)

* + 1. Pour la couverture des i-chemins, puisqu’il n’y avait pas de présence de boucle dans le code, nous avons conclut qu’il n’est pas nécessaire de faire des tests supplémentaires pour cette partie.

1. Description des résultats des tests :
   1. Boite noire

Les résultats des tests fonctionnaient comme prévu (sauf pour les cas où la valeur du montant est >10000 et <0. Se cas sera vue en conclusion car c’est un point important qui « change la donne »). Voici un tableau récapitulatif. Les devises de la colonne de gauche ont été convertit vers les devises de la première rangée (pour plus de détail sur tous les tests, rouler TestCurrencyConvertor.java).

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | USD | CAD | GBP | EUR | CHF | INR | AUD |
| USD | - | 10🡪13.44 | 5000🡪4230.05 | 9999🡪9761.52 | 2034🡪1950.84 | 7500🡪613047.21 | 9054🡪13714.07 |
| CAD | 20🡪14.87 | - | 5701🡪3587.68 | 9954🡪7228.46 | 2001🡪1427.59 | 7365🡪447809.03 | 8654🡪9750.57 |
| GBP | 1🡪1.18 | 1250🡪1986.30 | - | 2500🡪2884.86 | 5000🡪5668.46 | 7500🡪724633.30 | 10000🡪17904 |
| EUR | 1🡪1.02 | 1250🡪1721.32 | 2500🡪2166.48 | - | 5000🡪4912.245 | 7500🡪627961.42 | 10000🡪15515.47 |
| CHF | 1🡪1.04 | 1250🡪1752.07 | 2500🡪2205.18 | 5000🡪5089.32 | - | 7500🡪639179.66 | 10000🡪15792.64 |
| INR | 1🡪0.012 | 1250🡪20.55 | 2500🡪25.875 | 5000🡪59.71 | 7500🡪88.0034 | - | 10000🡪185.3076 |
| AUD | 1🡪0.66 | 1250🡪1109.42 | 2500🡪1396.33 | 5000🡪3222.59 | 7500🡪404732.4 | 10000🡪6332.06S | - |

* 1. Boite blanche

Pour les résultats des tests de la boite blanche, ça fonctionnait comme prévu. Le code roulait normalement et les exceptions sont lancé lorsqu’ils sont attendus. Pour plus de détail, aller directement rouler les tests du fichier TestCurrencyConvertor.java)

1. Observations sur les approches et critères de test :
   1. Boite noire
      1. On remarque que le même test était effectué plusieurs fois, mais sur des valeurs différentes. On a donc procédé à réutiliser le même code pour tester en créant un tableau d’objet TestFacilitator. Cet objet permet d’identifier la valeur attendue ainsi que la devise vers laquelle on convertit. De cette manière en parcourant le tableau, on peut vérifier que la valeur attendue correspond à la valeur testée.
   2. Boite blanche
      1. Nous avons pu remarquer que les tests de plusieurs des couvertures (des instructions, des arcs, des chemins indépendants et des conditions) consistaient à tester les mêmes points dans le code avec plus ou moins de profondeur. Ceci est logique puisqu’appart les instructions simples, il n’y avait qu’une condition « if » qui « complexifiait » le code. Donc pour chacun des cas, il fallait avoir des tests où ça rentre dans le « if » et les autres tests qui n’y rentre pas.
2. Conclusion :

Ce travail nous a permis de voir en détails les différences entre les tests fonctionnels et structurels dit tests de la boite noire et blanche. On peut aussi dire que les deux types de tests se complètent. Nous avons pu constater que les tests de la boites blanches fonctionnent et donnent un résultat qui concorde à la structure du code ( incluant le lancement d’une exception là où c’est supposé), cependant ce n’est pas le cas pour les tests de la boite noir. Avec les tests créés pour respecter les spécifications, nous avons remarqué que le code n’est effectivement pas conforme a ceux-ci. Plus spécifiquement, il n’y a aucun endroit dans le code où l’on vérifie que le montant rentrée en arguments est entre [0, 10000]. Les tests dont les entrées sont >10000 et <0 sont donc accepter par le code, même s’ils ne sont pas supposés être traité selon la spécification. Nous pouvons donc conclure que même si le code, dans sa structure, est bien implémente, nous savons que cette implémentation ne sert pas à répondre aux spécifications fournis.