**Numéro 1.**

*MainWindow:*

Tout d’abord, il nous est possible de remarquer qu’en fait, les devises à utiliser pour la méthode ne sont pas celles mentionnées dans l’énoncé, mais bien leurs noms complets. Donc, avec les noms de devises abrégées, on ne trouvera jamais les devises dans le ArrayList, et ce dernier retournera toujours une valeur de 0.0. En ce qui concerne les montants négatifs, cela est similaire, le test ne sera pas concluant, car à la base, nos « bonnes » devises nous retourneront toujours 0.0, car ce sont elles le problème et non pas le montant. Il serait donc pratique d’utiliser une vérification afin d’indiquer qu’elle est le problème avec les devises afin que l’utilisateur ne refasse pas la même erreur.

D’un autre côté, même si nous avions des devises qui sont dans l’énoncé, lorsque l’on teste avec une variable mauvaise, selon nous (p. ex. valeur de -1) on se retrouve quand même avec un problème. Ce dernier étant que nous aurions une valeur de retour alors qu’on ne devrait pas, à aucun moment dans le code, il y a une vérification afin de s’assurer que les valeurs sont supérieures ou égales à 0.

*Currency:*

Pour Currency.convert, on a qu’une seule restriction, le montant négatif. Donc, on a :

Domaine double = [0, infini[   
1er classe : dans domaine   
2eme classe : plus petit que domaine   
jeu de test : [300, -300]   
valeurs frontières : -1, 0

On a les jeux de tests suivants : {0, 0} {-1,0} {0,-1}

**Numéro 2.**

En ce qui concerne la couverture des instructions, pour MainWindow, nous avons 2 instructions spécifiques à exécuter : le 1er (if (currencies.get(i).getName() == currency2))et le 3eme(currencies.get(i).getName() == currency1) if. Le 2eme (if (shortNameCurrency2 != null)) n’a pas d’importance puisque ce dernier est inclut dans le premier. En d’autres mots, si on va dans le 1er, on ira toujours dans le 2eme. Avec ses observations, on peut conclure qu’on a besoin que d’un seul jeu de test qui contient une seule classe, c’est-à-dire la classe avec le domaine suivant :

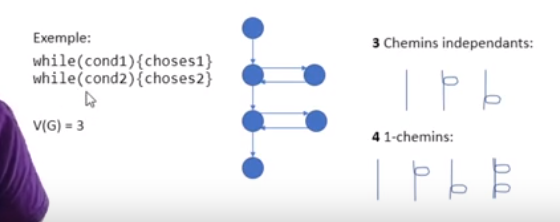
D1={(devise1,devise2, ArrayList<Currency>devises , montant) | devises.contains(devise1), devises.contains(devise2)}

Donc, en analysant le domaine, on arrive à la conclusion qu’un jeu de test valide serait : {("Japanese Yen", "Chinese Yuan Renminbi", Currency.init(), 100.0)}

Testez les deux méthodes en utilisant les 5 critères de sélection de jeux de test quand il fait du sens : couverture des instructions, couverture des arcs du graphe de flot de contrôle, couverture des chemins indépendants du graphe de flot de contrôle, couverture des conditions, couverture des i-chemins.

TEST BOITE BLANCHE  
-Teste en terme du contenu de structure interne, de la conception, on teste ce que prog fait

-Couverture des instructions : Jeu de test qui execute chaque instruction de P au moins une fois.

-Couverture arc du graphe de contrôle : on s’interesse aux branchements de contrôle conditionnels.  
-Chaque arc du graphe traversee au moins une fois. (if , vrai et faux)  
  
-Couverture chemins independants : on parcourt tous les 1-chemin du graphe de flot de contrôle   
Chemin = nœud de depart jusqu’à un nœud terminal   
1-chemin : chemin parcourant boucles 0 ou 1 fois  
chemin independant : chemin qui par parcourt au moins 1 nouvel arc   
  
1. Construire graphe de flot de contrôle   
2. Determiner complexite cyclomatique   
3.Determiner ensemble base B de chemins independants dans graphe  
4. Construire jeu de test qui permettra execution de chaque chemin de ensemble B  
  
Pour base B : une base qui a V(G)(taille de complexite cyclomatique) chemins   
  


4 1-chemins : 0 true, 1 true 2 fois, 2 true

3 chemins independants : 0 true, 1 true 2 fois (le 2 true n’est pas indep des 2 autreS car il est compose des 2 chemines true)

CHAPITRE 8 :   
Validation = bon produit, repondu aux besoins  
Verification = produit construit correctement, exigences de specification repondues

Supposez que la spécification du « Currency Converter » exige qu’il doit : • Convertir des montants entre les devises suivantes : USD, CAD, GBP, EUR, CHF, CNY. • Seulement accepter des montants non-négatives. Testez les deux méthodes en utilisant les approches de partition du domaine des entrées en classes d’équivalence et d’analyse des valeurs frontières.

Pour les types de devises, on a 2 classes : les devises permises et non permises. Pour le montant, on a également 2 classes : les montants situé entre 0 et infini, et les montants inferieurs à 0. On n’a pas à considérer les montants supérieurs à infini. Finalement, on a le ArrayList contenant les devises avec 2 classes, les ArrayList de devises et tout autre valeur qui n’est pas un ArrayList de devises.  
  
Donc, pour la première méthode, on a les jeux de tests suivants : (USD, JPY, ArrayList<Currency>, 0} {JPY, USD, ArrayList<Currency>, 0}, {USD, CAD, ArrayList<Currency>, -1} {USD, CAD, ArrayList<Double>, 0> . On ne considère pas le cas

Domaine de 1er = USD CAD GBP EUR CHF CNY  
Classe équivalence : USD = Americain CAD = canadien, GBP = british pound  
EUR = euro CHF = Suisse franc , CNY = Chinese Yuan  
1er classe : Devises permises  
2eme classe : Devises non permises   
Jeu de test valide : {USD, JPY} , usd est valide, pas JPY

Domaine 2eme = [0, infini[  
1er classe : dans domaine   
2eme classe : plus petit que domaine   
jeu de test : [300, -300] valeurs frontieres : -1, 0

2. Pour la couverture des instruction, pour MainWindow, on a 2 instructions spécifiques à exécuter : le 1er et le 3eme if. Le 2eme n’a pas d’importance car si on va dans le 1er, on ira toujours dans le 2eme. Avec ceci, on n’a besoin que d’un jeu de test qui contient une seule classe, c’est-à-dire la classe avec le domaine suivant : D1={(devise1,devise2, ArrayList<Currency>devises , montant) | devises.contains(devise1), devises.contains(devise2)}

Donc, un jeu de test valide serait : {("Japanese Yen", "Chinese Yuan Renminbi", Currency.init(), 100.0)}