**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)

Pour Currency, on va toujours exécuter toutes les conditions, donc on peut mettre n’importe quel domaine de valeurs continues. D’ailleurs, on remarque que le montant peut être négatif, contrairement à la spécification de la boite noire.

Pour la couverture des arcs de flots, on retrouve le domaine précédent, en plus de devoir tester 2 autres domaines : une devise1 qui ne se retrouve pas dans devises, et une devise2 qui ne se retrouve également pas dans devises.

Pour Currency, on n’a pas d’if, donc on n’a pas besoin de tester les arcs de flots car le domaine serait pareil.

Pour la couverture des arcs de flots indépendants, il ne fait pas de sens de tester car nous aurions à tester les mêmes domaines que pour les arcs de flots. En effet, si on passe par des chemins indépendants on aura les domaines : 1er if false, 1er if vrai, 3eme if false, 1er if vrai, 3eme if vrai. Donc, on ne fera pas de tests séparés pour la couverture des arcs de flots indépendants.  
  
Pour la couverture des conditions, c’est un peu comme pour la couverture des arcs de flots indépendants (vu plus haut). Les cas ont déjà étés testé, cela serait superflu de retester pour arrivé aux mêmes résultat qu’obtenu précedement.

Pour la couverture des i-chemins,