**Partie 1**

**Gestion des montants non-valides :**

Si l'utilisateur entre un montant qui n'est pas un nombre, affiche un message d'erreur et demande une nouvelle saisie.

Si le montant est négatif, demande une nouvelle saisie en expliquant que le montant doit être positif.

**Gestion des devises non-valides :**

Si l'utilisateur entre une devise qui n'est pas reconnue, affiche un message d'erreur et demande une nouvelle saisie.

**Pour le montant de l’argent j’ai choisi 3 classes d’équivalence**

Une classe pour la valeur appartenant dans les valeurs valides (10)

Une classe pour la valeur qui est inferieur pour les valeurs

d’entrées invalides inférieures à l’intervalle.(-50)

Une classe d’équivalence pour les valeurs d’entrées invalides supérieures à l’intervalle. (5 000 000)

**Pour le type de monnaie j’ai choisi 2 classes**

Une classe d’équivalence pour les valeurs d’entrée valides. (USD, CAD)

Une autre classe d’équivalence pour les valeurs d’entrées invalides (ABC , 123)

**La valeur frontière pour le montant on va aussi tester la valeur -1, 0 , 1 000 000 et 1000 001**

**Partie 2**

currencyConverter.Currency.convert(Double, Double)

**Couverture des instructions:** Il suffit d'utiliser des données normales pour que la méthode exécute toutes les instructions, par exemple (10, 1.5).

**Couverture des arcs du graphe de flot de contrôle:** Comme cette méthode n’a pas d’instruction conditionnelle (if, while), le test est exactement le même que le premier.

**Couverture des chemins indépendants du graphe de flot de contrôle:** …

**Couverture des conditions:** …

**Couverture des i-chemins:** même raison, on a seulement besoin d’utiliser des données normales, tester 2ou 3 fois suffit.

currencyConverter.MainWindow.convert(String, String, ArrayList, Double)

**Couverture des instructions:** Il suffit d'utiliser des données normales pour que la méthode exécute toutes les instructions, par exemple ("USD", "JPY", Currency.init() , 100). Normalement, les instructions devraient exécuter au moint une fois.

**Couverture des arcs du graphe de flot de contrôle:** Pour chaque instruction « if », on teste la condition est vraie et celui où elle est fausse :

* Le cas normal : ("USD", "JPY", Currency.init() , 100)
* Le cas où currency2 n’est pas dans currencies : ("USD", "CAD", Currency.init() , 100). Dans ce cas , shortNameCurrency2 est alors null, on saute la prochaine instruction if.
* Le cas où currency1 n’est pas dans currencies : ("CAD","USD", Currency.init() , 100).

On a alors parcouru tous les arcs du graphe de flot de contrôle.

**Couverture des chemins indépendants du graphe de flot de contrôle :** Le graphe ressemble à :

**A diagram of a diagram

Description automatically generated with medium confidence**

V(G)=3

On a donc exactement les mêmes tests que ‘couverture des arcs du graphe de flot de contrôle’.

Le cas normal (1-2-3-5-6-8), le cas où currency2 n’est pas dans currencies (1-2-4-8), le cas où currency1 n’est pas dans currencies mais currency2 est dedans (1-2-3-5-7-8).

**Couverture des conditions:** On doit s’assurer que toutes les conditions sont calculées au moins une fois.

Donc, on a 4 tests : Le cas normal (currency1 et currency2 sont dans currencies), le cas où currency2 n’est pas dans currencies (mais currency1 oui) , le cas où currency1 n’est pas dans currencies mais currency2 est dedans, et le cas où currency1 et currency2 sont absents dans currencies.

Mais, lors de l’exécution, le deuxième et le dernier sont exactement même, logiquement.

**Couverture des i-chemins:** On a besoin de tester non seulement si currency est dans currencies, mais aussi tester la boucle de « for » :

On ajoute quatre tests, qui peuvent se combiner avec les tests de couverture des conditions :

* Le cas où currencies == [] //on saute la boucle
* Le cas où currencies == [Currency("US Dollar", "USD")] //1 iterations
* Le cas où currencies == [Currency("Euro", "EUR"),Currency("US Dollar", "USD")] //2 iterations
* Le cas où currencies == [Currency.init()] //n iterations