TP4-IFT3913

Wenhao Xu, 20150702
Manping Li, 968527

1. Test boite noire

1. partition du domaine des entrées en classes d'équivalence:
Il y a quatre parameters dans le méthod "convert(double amount, String from, String to, CurrencyConversion conversion)".Idéallement,on fait la partition sur ces quatre paramètres et prend la permutation de ces partitions.À cause de la contrainte du temps limité,nous avons choisi de fixer trois paramètres qui sont dans le demaine valide et de faire la partition sur le domaine du quatième paramètre.

Pour les paramètres "from" et "to", une classe d'équivalence pour les valeurs d'entrée valides est:
D_1 = {USD, CAD, EUR, GBP, CHF, INR, AUD}
Le domain de est tout les entrées qui ne sont pas dans D1. Nous avons choisi une classe d'équivalence pour les valeurs d'entrée invalides est:
D_2 = {FJD, "", null}

Pour le paramètre "amount",le domain D'=R.
une classe d'équivalence pour les valeurs d'entrée valides est:
D'_1={$0 \leq d \leq 10000$}
une classe d'équivalence pour les valeurs d'entrée invalides inférieures à l'intervalle est:
$D'_2 = \{d < 0\}$
une classe d'équivalence pour les valeurs d'entrée invalides supérieures à l'intervalle est:
$D'_3 = \{d > 10000\}$

Premièrement,nous avons fixer le paramètre "amount" dans le demaine valide et mettre un des deux paramètres parmi "to" et "from" dans le domaine nonvalide,nous avons les jeux de tests suivant:
T1={(100,null,"CAD",conversion)}=ParseException;
T2={(100,"CAD",null,conversion)}=ParseException;
T3={(100,"","CAD",conversion)}=ParseException;
T4={(100,"CAD","",conversion)}=ParseException;
T5={(100,"EHR","FJD",conversion)}=0.0
T6={(100,"FJD","EUR",conversion)}=ArithmeticException;
Le code a réussi les tests de T1 à T4,mais a échué T5 et T6.Le code n'a pas réussi le T5,car "FJD" ne fait pas partie de la domaine valide D_1. Le méthode convert a quand même trouvé le taux échange pour "FJD" converti 100 "EUR" à "FJD".Ceci n'est pas équal à 0.0. Le code n'a pas réussi le T6,car si "FJD" est l'entrée du paramètre "From",qui est dans dénominateur de l'équation:amount*(curencyTo/curencyFrom).Si curencyFrom=0,il y aura une exception arithmetic.Mais le programme a trouvé le taux de change pour "FJD",ceci n'est pas correct.

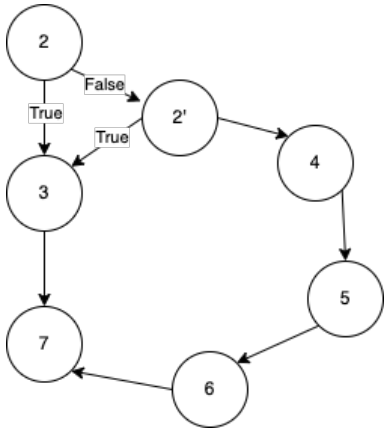
Ensuite,nous avons testé le paramètre "amount" en utilisant l'approches de partition du domaine des entrées en classes d'équivalence,nous avons les tests suivants:
T7={(-5,"USD","USD",conversion)}=0.0;
T8={(20000,"EUR","EUR",conversion)}=0.0;
T9={(100,"CAD","CAD",conversion)}=100;
T7 et T8 sont échués car -5 et 20000 sont dans le domaine de nonvalide,le valeur du méthod doivent équal à 0.0 mais le code n'a pas de contraint sur le validité du domaine selon la spécification.T9 est réussi car 100 est dans le domaine valide.
Finallement,nous avons utilisé l'approchee d'analyse des valeurs frontières pour tester le méthode "convert".Un jeu de test valide serait:{ -5, -1, 0, 100, 10000,10001, 20000}

Nous avons économisé deux tests pour les valeur -5 et 20000 qui sont déja fait dans les test précédant. nous avons les tests suivants:
T10={(0,"AUD","GBP",conversion)}=0.0;
T11={(-1,"AUD","GBP",conversion)}=0.0;
T12={(10000,"CHF","INR",conversion)}>0.0;
T13={(10001,"CHF","INR",conversion)}=0.0;
T14={(100,"AUD","GBP",conversion)}>0.0;
T11,T13 sont échués car le code n'a pas mis le contraint de demaine sur le paramètre "amount",-1 et 10001 ne sont pas dans le domaine de valide mais le code a quand même calculé les résultats qui ne sont pas correctes. Tout les autres test sont réussi.

2. Test boite blanc

```
1  public static double convert(double amount, String from, String to, CurrencyConversion
       conversion) throws ParseException{
2          if(!conversion.getRates().containsKey(to)||!conversion.getRates().containsKey(from))
3                  throw new ParseException("Not correct format currency"+ "", 0);
4          double curencyTo = conversion.getRates().get(to);
5          double curencyFrom = conversion.getRates().get(from);
6          return amount*(curencyTo/curencyFrom);
7  }
```

Le graphe de flot de contrôle est le suivant:



Nous avons utilisé 5 critères de sélection de jeux de test pour faire le test boite blanc:
(1)Couverture des instructions:
D4={(amount,from,to,conversion)|!conversion.getRates().containsKey(to)=true}
D5={(amount,from,to,conversion)|!conversion.getRates().containsKey(to)=false&&!conversion.getRates().containsKey(from)=true}
D6={(amount,from,to,conversion)|!conversion.getRates().containsKey(to)=false&&!conversion.getRates().containsKey(from)=false}
Pour couvrir toutes les noeuds,nous avons fait deux tests:
T15={(100,"AUD","CAD",conversion)}>0.0;
T16={(100,NULL,"CAD",conversion)}=ParseException;
T17={(100,"CAD",NULL,conversion)}=ParseException;
Tous les tests sont réussi.
(2) couverture des arcs du graphe de flot de contrôle:
Arc:2→3→7⇒!conversion.getRates().containsKey(to)=true
Arc:2→2'→3→7⇒!conversion.getRates().containsKey(to)=false&&!conversion.getRates().containsKey(from)=true
Arc:2→3→4→5→6→7⇒!conversion.getRates().containsKey(to)=false&&!conversion.getRates().containsKey(from)=false
Pour couvrir toutes les arcs,nous avons fait trois tests:
T18={(100,"USD","CAD",conversion)}>0.0;
T19={(100,NULL,"CAD",conversion)}=ParseException;
T20={(100,"CAD",NULL,conversion)}=ParseException;

Tous les tests sont réussi.
(3)couverture des chemins indépendants du graphe de flot de contrôle:
Parceque V(G) du GFC est 3,nous avons trois chemin indépendant:
Chemin1=(2,3,7);
Chemin2=(2,2',3,7);
Chemin3=(2,2',4,5,6,7);
Pour couvrir toutes les chemins,nous avons fait trois tests:
T21={(100,"USD","CAD",conversion)}>0.0;
T22={(100,"cad","CAD",conversion)}=ParseException;
T23={(100,"CAD","cad",conversion)}=ParseException;
Tous les tests sont réussi.
(4)couverture des conditions:
nous avons décomposé la condition composé à deux conditions simples,le code est suivant:

```
public static double convert(double amount, String from, String to, CurrencyConversion conversion) throws ParseException{
    if(!conversion.getRates().containsKey(to)){
        throw new ParseException("Not correct format currency"
                + "", 0);
    }else{
        if(!conversion.getRates().containsKey(from)){
            throw new ParseException("Not correct format currency"
                    + "", 0);
        }else{
            double curencyTo = conversion.getRates().get(to);
            double curencyFrom = conversion.getRates().get(from);
            return amount*(curencyTo/curencyFrom);

        }
    }
}
```

Pour couvrir toutes les conditions simples,nous avons fait trois tests:
T24={(100,"USD","CAD",conversion)}>0.0;
T25={(100,"","CAD",conversion)}=ParseException;
T26={(100,"CAD","",conversion)}=ParseException;
Tous les tests sont réussi.Il faut noter que il y a 4 combinaisions de valeurs de condition 1 et condition 2.Nous avons testé trois car le compilateur de java,si condition1 est faux,il aura une ParseException.Le condition 2 est faux n'a jamais été testé.

(5)couverture des i-chemins:
Car il n'y a pas de boucle dans le méthode "convert",c'est pas pertinant de utiliser ce critère.

3. Conclusion

On a 6 test échec sur tests boite noire, et 0 échec sur tests boite blanc.
ceci du au fait que le test boite noire est basé sur la spécification donc sur l'entrée des données et test boite blanc est plus concentré sur des logiques s'il y a des erreurs qui n'a pas êtres considère.
On voit que le méthode "convert" n'a pas prise en comptre le domaine valides de la spécification.Les constraints sur les paramètre "to","from","amount" ne marche pas.