Projet Hanoi tests et git

1.0

Cyrille François © Ateno Tech

Table des matières

Réalisation et tests d'un jeu de tours de Hanoi

I

A. Tests

Le problème des tours de Hanoï est un jeu de réflexion consistant à déplacer des disques de diamètres différents d'une tour de **départ** à une tour d'**arrivée** en passant par une tour **intermédiaire** et ceci en un minimum de coups, tout en respectant les règles suivantes :

- On ne peut déplacer plus d'un disque à la fois
- On ne peut placer un disque que sur un autre disque plus grand que lui ou sur un emplacement vide

On suppose que cette dernière règle est également respectée dans la configuration de départ (où tous les disques se situent sur la tour de **départ**).

Le nombre de disques est fonction de la hauteur des tours.

La hauteur des tours, le nombre et le diamètre des disques seront paramétrables dans notre implémentation...

Le code du jeux est disponible sur github : https://github.com/CfrancCyrille/hanoi

Test partitionnel et analyse des valeurs limites (tests fonctionnels)

Nous avons créé une méthode *empiler* dans la classe Tour, pour empiler des disques définit par la classe **Disque** dont voici un extrait :

```
1  public class Disque implements Comparable<Disque> {
2    int d;
3    public Disque(int d) {
4        this.d = d;
5    }
6    int compareTo(Disque obj){
7      //[...]
8    }
9  }
```

La fonction *empiler* est déclarée dans l'interface **IPile** et est implémentée dans la classe **Tour** dont voici un extrait :

```
1  public class Tour implements IPile<Disque>{
2    protected int diam(){ //Retourne le diametre de l'élément situé au sommet
3      //[...]
4    }
5    public boolean empiler(Disque d){
6      //[...]
```

```
7       }
8       //[...]
9   }
```

Question 1

Nous n'avons pas encore le code !

Il faut déterminer les **classes d'équivalence** à considérer pour la méthode *empiler*, en remplissant le tableau suivant :

Paramètre d'entrée	Classe valide	Classe invalide
Diamètre de d si la tour est vide (d = diamètre du disque)	**d £] 0,Max_Value]**	**d <= 0** **d > Max_Value**
Diamètre de d si la tour n'est pas vide (s = disque au sommet)	**d £] 0,S]**	**d > S** **d <= 0**

Tableau 1 Classes d'équivalence

Question 2

Déterminer maintenant, par une *approche aux limites*, les données de tests à produire pour la méthode *empiler*, en remplissant le tableau suivant :

Paramètre d'entrée	Classe valide	Classe invalide
Diamètre de d si la tour est vide (d = diamètre du disque)	**d = 1 , d= Max_Value**	**d = 0, d=Max_Value+1**
Diamètre de d si la tour n'est pas vide (s = disque au sommet)	**d = s-1 avec d > 1**	**d = S , d = 0**

Tableau 2 approche aux limites

« *Rappel : le test des valeurs limites n'est pas vraiment une famille de sélection de test, mais une tactique pour améliorer l'efficacité des données de test (DT) produites par d'autres familles. Les erreurs se nichent généralement dans les cas limites, d'où le fait qu'on teste principalement les valeurs aux limites des domaines ou des classes d'équivalence...* »

Sélection de tests boîte blanche (tests structurels)

Soit le programme en langage Java suivant pour la classe **Tour** des Tours de Hanoï :

```
1   public class Tour {
2
3       Queue<Disque> disques=new ArrayDeque<Disque>();
4
5       public int diam(){
6          return this.disques.element().d;
7       }
8
9       public int taille() {
```

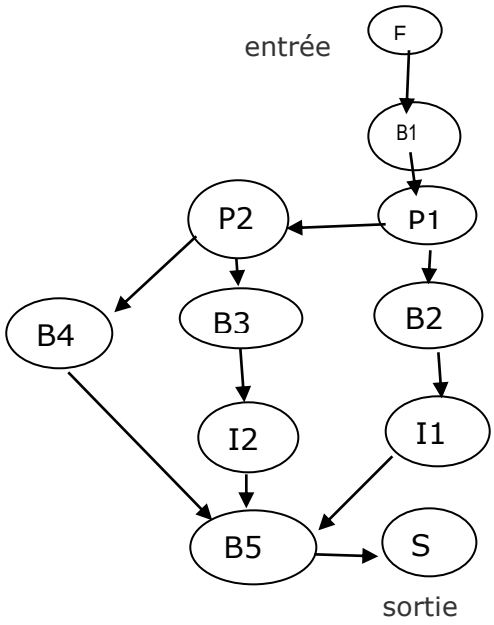
```
        return disques.size();
    }

    boolean empiler(Disque d){
        boolean res=false; // B1
        if(this.disques.isEmpty()){ // P1
            this.disques.offer(d); // B2
            res=true; // I1
        }
        else{
            if( (diam()>d.d) && (taille()<hauteurMax) ){ // P2
                this.disques.offer(d); // B3
                res=true; // I2
            }
            else{
                res=false; // B4
            }
        }
        return res; // B5
    }
}
```

Question 3

Dessiner le graphe de flot de contrôle correspondant à ce programme (en se servant des annotations indiquées en commentaires : // B1, // P1, // B2, // I1, // P2, ...) et donner sa forme algébrique :

>Chemin 1 {B1, P1, B2, I1, B5}
>Chemin 2 {B1, P1, P2, B3, I2, B5}
>Chemin 3 {B1, P1, P2, B4, B5}
>
>Forme Algebrique :
>chemin1+chemin2+chemin3
> B1P1B2I1B5 + B1P1P2B3I2B5 + B1P1P2B4B5
>
>B1P1B5(B2I1 +[P2(B3I2+B4)])
>
>
>



Question 4

Trouver les données de test minimales pour couvrir toutes les instructions (couverture de tous les nœuds du graphe de flot de contrôle)

Reponse : d=1 vérifie tous les cas Possibles

Question 5

Ces données de test assurent-elles la couverture de tous les arcs du graphe ? Sinon ajouter de nouvelles données de test pour couvrir tous les arcs.

Reponse : oui

Question 6

La ligne ci-dessous représente une condition composée (deux expressions booléennes reliées par un ET logique) :

```
(diam()>d.d) && (taille()<hauteurMax)
```

Ainsi, pour que les tests soient complets, il faut évaluer toutes les possibilités de cette conditionnelle, répertoriées dans le tableau ci-dessous :

ET logique	Vrai	Faux
Vrai	Vrai	Faux
Faux	Faux	Faux

Tableau 3 conditionnelle composée

Ajouter au besoin des données de test pour que les tests soient complets.

Tests unitaires en Java avec JUnit

Dans le cadre du jeu des des tours de Hanoï réalisé au TP1 (en Java)

Question 7

Compléter la classe de test **TourTest** (en JUnit) pour tester la méthode empiler de la classe Tour, en considérant les données de test recensés plus haut (tests fonctionnels).

Question 8

Compléter la classe de test **DisqueTest** (en JUnit) pour tester la méthode *compareTo* de la classe Disque.

Question 9

Compléter la classe Hanoi pour résoudre le jeux avec :

- 3 disques
- n disques (nombre paramétrable de disques)