Projet Hanoi tests et git

1.0

CANDAPPANE Vincent
ESIREM 4A

Table des matières

Réalisation et tests d'un jeu de tours de Hanoi

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      //[...]
```

```
    }
    //[...]
}
```

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)	$d \in]\ 0\ ;\ MAX\_VALUE\ [$	$d \leq 0$ $d > MAX\_VALUE$
Diamètre de d si la tour n'est pas vide (s = disque au sommet)	$d \in]\ 0\ ;\ s\ [$	$d \geq s$ $d \leq 0$

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)	$d = 1$ $d = MAX\_VALUE$	$d = 0$
Diamètre de d si la tour n'est pas vide (s = disque au sommet)	$d = 1$ $d = s - 1$ avec $s > 1$	$d = s$ $d = 0$

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 :

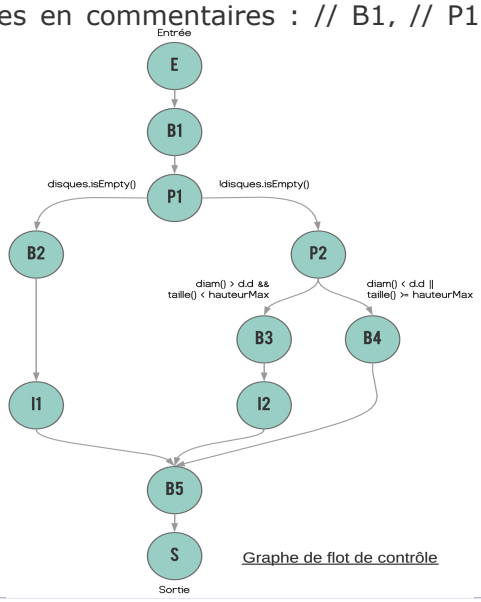
```
public class Tour {

    Queue<Disque> disques=new ArrayDeque<Disque>();

    public int diam(){
       return this.disques.element().d;
    }

    public int taille() {
```

```
10          return disques.size();
11      }
12
13      boolean empiler(Disque d){
14          boolean res=false; // B1
15          if(this.disques.isEmpty()){ // P1
16              this.disques.offer(d); // B2
17              res=true; // I1
18          }
19          else{
20              if( (diam()>d.d) && (taille()<hauteurMax) ){{ // P2
21                  this.disques.offer(d); // B3
22                  res=true; // I2
23              }
24              else{
25                  res=false; // B4
26              }
27          }
28          return res; // B5
29      }
30  }
```

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 :

>
>
>
>
>
>
>
>
>
>
>
>



Graphe de flot de contrôle

Forme algébrique :

= B1.P1.B2.I1.B5 + B1.P1.P2.B3.I2.B5
+ B1.P1.P2.B4.B5
= B1.P1.B5(B2.I1.P2.B3.I2 + P2.B4)
= B1.P1.B5[B2.I1 + P2(B3.I2 + 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) **Voir fin du document .pdf**

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. **Voir fin du document .pdf**

Question 6

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

```
1   (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. **Voir fin du document .pdf**

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). **Voir code**

Question 8

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

Question 9

Compléter la classe Hanoi pour résoudre le jeux avec : **Voir code**

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

Eléments de réponse :

- Question 4

Cas 1 : La tour est vide (P1 vrai) -> Placement de disque autorisé
Cas 2 : La tour n'est ni vide ni pleine (P1 faux, P2 vrai) -> Placement de disque autorisé
Cas 3 : La tour est pleine (P1 faux, P2 faux) -> Placement de disque non autorisé

- Question 5

Oui 👍 Ces données de tests assurent la couverture de tous les arcs du graphe

- Question 6

À la question N°4, nous avions pu énumérer trois données minimales couvrant la totalité des arcs du cercle. Cependant, la condition P2 est composé de deux conditions (expressions booléennes) reliées par un ET logique. Voici les données de tests pour que les tests (tous les cas possibles) soient tous vérifiés et complets :

Vrai && Vrai	d.d = diam() -1 && taille() = hauteurMax - 1
Vrai && Faux	d.d = diam() -1 && taille() = hauteurMax
Faux && Vrai	d.d = diam() && taille() = hauteurMax - 1
Faux && Faux	d.d = diam() && taille() = hauteurMax

De cela, en découle les tests suivants qui sont à présents fonctionnels après rectification de bugs :
- Test(1) Vrai : Empiler dans tour vide
- Test(2) Faux : Empiler sur disque plus petit
- Test(3) Vrai : Empiler sur disque plus grand
- Test(4) Faux : Empiler sur disque de la même taille
- Test(5) Faux : Empiler correctement mais dans tour pleine