

TD2 : Codification de logique en Java

Ce TD a pour objectif de vous faire pratiquer, écrire en java des algorithmes abordés au cours de logique l'an passé ainsi que de revoir certaines notions java (générique, foreach, ...).

Cet énoncé provient de l'examen de septembre 2008 du cours de logique et technique de programmation.

En annexe vous sont fournies les classes *ArbreBinaire* et *NoeudBinaire* à compléter.

1 UN PATCHWORK NUMÉRIQUE

On vous demande d'écrire le code de la classe *Patchwork* qui contient au moins comme attributs privés :

```

morceau : tableau [ L ] [ C ] d'entiers
L : entier qui est le nombre de lignes du tableau morceau
C : entier qui est le nombre de colonnes du tableau morceau
    
```

1. Écrivez le constructeur de la classe

ayant comme paramètre un tableau d'entiers à deux dimensions. Le tableau doit être un tableau rectangulaire non nul, mais peut être vide. Le constructeur renverra une exception *IllegalArgumentException* dans le cas contraire.

L'entier contenu dans chaque élément du tableau morceau représente une couleur. Un ensemble d'éléments contigus ayant la même couleur (donc la même valeur) s'appelle une tache. Il faut au moins deux éléments contigus de même couleur pour constituer une tache (*un point de couleur isolé ne constitue pas une tache*). On suppose que le constructeur de cette classe place aléatoirement des valeurs dans le tableau « morceau » créant donc probablement un ensemble de taches. Des taches de même couleur peuvent bien évidemment apparaître dans ce patchwork. Il est possible également que le tableau construit ne contienne que des points de couleur isolés et donc aucune tache. En fait, le nombre de taches est au minimum 0, et au maximum $(L * C) \text{ DIV } 2$

2. Écrivez la méthode `compterTaches`

```
méthode compterTaches ( ) → entier
```

compte et retourne le nombre de taches contenues dans le tableau morceau.

Il est évident que cette méthode appelle une méthode privée réursive que vous devez également écrire. Par souci de modularité vous pouvez également écrire d'autres méthodes.

Pour rappel, l'élément $[a][b]$ est contigu à l'élément $[x][y]$ si ces deux éléments sont bien dans le tableau et si :

$$(|a-x| == 1 \text{ ET } b == y) \text{ OU } (|b-y| == 1 \text{ ET } a == x)$$

Il va de soi qu'il est interdit de modifier le tableau morceau.

3. Écrivez les tests Junit pour le constructeur et la méthode `compterTaches`.

4. Documentez votre classe.

Exemple : pour le tableau ci-dessous, `compterTaches ()` retournerait 14

	0	1	2	3	4	5	6	7
0	5	5	4	4	4	8	8	9
1	2	5	4	4	4	8	9	8
2	2	2	5	3	4	8	9	8
3	2	2	2	3	3	3	9	4
4	5	1	1	1	3	3	4	4
5	7	5	4	1	3	5	4	4
6	7	5	5	1	6	5	5	5
7	7	5	5	9	6	5	6	6
8	7	7	7	4	5	1	6	6

2 COMPARAISON D'ARBRES BINAIRES

Soit les deux classes suivantes fournies :

```

classe NoeudBinaire <T>
public :
constructeur NoeudBinaire ( valeur : T )
// construit un noeud avec nul dans filsGauche et filsDroit
méthode getValeur ( ) □ T
méthode setValeur ( valeur : T )
méthode getGauche ( ) □ NoeudBinaire
méthode setGauche ( fils:NoeudBinaire )
méthode getDroit ( ) □ NoeudBinaire
méthode setDroit ( fils:NoeudBinaire )
fin classe
    
```

```

classe ArbreBinaire <T>
public :
constructeur ArbreBinaire ( ) // construit un arbre vide (nul pour racine)
méthode getRacine ( ) □ NoeudBinaire <T>
méthode setRacine ( rac : NoeudBinaire <T> )
fin classe
    
```

1. Ajoutez les méthodes equals

La méthode `equals` de l'arbre retourne vrai si les 2 arbres sont identiques (même structure et mêmes valeurs dans les nœuds de même position).

La méthode `equals` d'un nœud retourne vrai si les deux nœuds sont identiques (même valeur et mêmes fils).

2. Utilisez Junit pour tester vos deux méthodes

3. Générez la javadoc et un fichier .jar de vos classes (qu'on puisse tester ;-)