Examen de Test et Vérification
2018–2019

— Durée : 2h

— Type : ExamManager

— Rendu : une archive contenant les sources Java à déposer dans le dossier `rendu-test-verif`, et, pour les réponses aux autres questions, un fichier PDF à déposer dans le même dossier, ou une copie papier à rendre aux surveillants

— Tous documents autorisés.

— Toutes vos affaires (sacs, vestes, *etc.*) doivent être placées à l'avant de la salle.

— Aucun téléphone ne doit se trouver sur vous ou à proximité, même éteint.

— Les déplacements et les échanges ne sont pas autorisés.

— Aucune question ne peut être posée aux enseignants, posez des hypothèses en cas de doute.

Test (10 points)

Tests en boîte noire

Une droite dans un plan affine de dimension 2 est déterminée par une équation cartésienne de la forme $ax+by+c=0$, où a, b et c sont des constantes telles que $(a,b) \neq (0,0)$. Soit une méthode `int droites(double a1, double b1, double c1, double a2, double b2, double c2)` permettant de déterminer si les deux droites passées en paramètre sont confondues, strictement parallèles, perpendiculaires ou non. Pour cela, la méthode renvoie l'une des constantes suivantes :

— `DROITES_CONFONDUES` si les deux droites ont une infinité de points communs ($a_1 * b_2 = a_2 * b_1$ et $a_1 * c_2 = a_2 * c_1$),

— `DROITES_PARALLÈLES` si les deux droites n'ont aucun point commun ($a_1 * b_2 = a_2 * b_1$ et $a_1 * c_2 \neq a_2 * c_1$),

— `DROITES_PERPENDICULAIRES` si les deux droites se coupent en formant un angle droit ($a_1 * b_2 \neq a_2 * b_1$ et $a_1 * a_2 = -b_1 * b_2$),

— `DROITES_SÉCANTES` sinon ($a_1 * b_2 \neq a_2 * b_1$ et $a_1 * a_2 \neq -b_1 * b_2$).

Si pour une droite donnée $(a,b) = (0,0)$, une exception de type `IllegalArgumentException` est levée.

1. Donnez un ensemble complet de cas de test pour tester la méthode. (3pts)
2. Écrivez, dans une classe de test JUnit, les méthodes de test correspondant aux cas de test (pour compiler ces tests, une version minimale de la méthode testée est requise). (2pts)

Tests en boîte blanche

Soit la méthode suivante :

```
1  int foobar(int x, int y, int z) {
2    int k = 0;
3    for(int i=0; i < z; i++) {
4      for(int j=0; j < y; j++) {
5        if (i < x) {
6          k++;
7        } else {
8          k--;
9        }
10     }
11   }
12   return k;
13 }
```

1. Construisez le graphe de flot de contrôle de la méthode, en indiquant à quoi correspondent chaque nœud et chaque arête dans le code. (1pt)
2. Calculez le nombre cyclomatique et énumérez tous les chemins élémentaires du graphe correspondants/à tester. (2pts)
3. Donnez un ensemble minimal de cas de test qui couvre tous les chemins élémentaires du graphe. (2pts)

Vérification (10 points)

Soit la méthode suivante qui renvoie un nouveau tableau dont le contenu est l'inverse du tableau non vide passé en paramètre. Par exemple, `inverser([1,-2,3,-4])` renvoie le tableau `[-4,3,-2,1]`.

```
1  int[] inverser(int[] t) {
2    int[] rev = new int[t.length];
3    for(int i=0; i < t.length; i++) {
4      rev[i] = t[t.length - i];
5    }
6    return rev;
7  }
```

Une faute a été introduite dans cette méthode. Utilisez l'outil de vérification Why3 [1] (ou Why2) pour prouver la correction totale de la méthode.

1. Écrivez la pré-condition de la méthode. (1pt)
2. Écrivez la post-condition de la méthode. (3pts)
3. Écrivez l'invariant de la boucle. (3pts)
4. Écrivez la fonction variant de la boucle. (1pt)
5. Identifiez et corrigez la faute. (1pt)
6. Afin d'augmenter encore davantage la confiance du programmeur, est-il utile d'ajouter des tests pour cette méthode ? Si oui, lesquels ? Si non, expliquez pourquoi ? (1pt)

1. Avant de lancer l'outil de vérification Why3, commencez par exécuter la commande `why3 config --detect` afin qu'il détecte les prouveurs installés (*e.g.*, Alt-Ergo).