Examen de Test et Vérification
2016–2017

- 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)

Soit la méthode `permutation` suivante qui renvoie `true` si le tableau d'entiers passé en paramètre est sans doublon (*i.e.*, sans apparition multiple d'un élément) et si tous ses éléments sont compris entre 0 et $N-1$, où N est la taille du tableau, `false` sinon. Par exemple, `permutation([1, 3, 4])` retourne `false`, alors que `permutation([1, 0, 2])` retourne `true`.

```
1  public static boolean permutation(int[] t) {
2    for (int i = 0; i < t.length - 1; i++) {
3      if (t[i] < 0 || t[i] >= t.length)
4        return false;
5      for (int j = i + 1; j < t.length; j++)
6        if (t[i] == t[j])
7          return false;
8    }
9    return true;
10 }
```

1. Construisez le graphe de flot de contrôle de la méthode `permutation`, en décomposant les conditionnelles. (2pts)
2. Listez les sommets et les arêtes parcourus par les données de test suivantes :
   (a) `t1 = [1, 3, 4]` (0.5pt)
   (b) `t3 = [1, 0, 2]` (0.5pt)
3. Écrivez, dans une classe de test JUnit, les méthodes de test correspondant aux données de test de la question précédente. (1pt)
4. Rajoutez les tests nécessaires pour couvrir tous les arcs du graphe. (1pt)
5. Modifiez la méthode `permutation` de sorte que si le tableau passé en paramètre est `null` un exception de type `IllegalArgumentException` soit levée, puis ajoutez dans la classe de test, la méthode de test correspondante. (1pt)
6. La méthodes `permutation` est erronée.
   (a) Identifiez la faute. (0.5pt)
   (b) Si possible, identifiez un cas de test qui n'exécute pas la faute. (0.5pt)
   (c) Si possible, identifiez un cas de test qui exécute la faute, mais n'aboutit pas à un état d'erreur. (0.5pt)

(d) Si possible, identifiez un cas de test qui entraîne une erreur, mais pas une défaillance. (0.5pt)

(e) Identifiez un cas de test qui provoque une défaillance. (0.5pt)

(f) Corrigez la faute. (0.5pt)

7. Vos tests sont-ils suffisants ? Autrement dit, pouvez-vous écrire une méthode `permutation` qui passe vos tests mais qui soit fausse ? Si oui, donnez un exemple d'une telle méthode et écrivez le(s) cas de test à ajouter pour éviter ce problème. Si non, expliquez pourquoi. (1pt)

Vérification (10 points)

Utilisez l'outil de vérification Why3 [1] pour prouver la correction totale de la méthode `permutation`, une fois celle-ci corrigée.

1. Écrivez la pré-condition du programme. (1pt)
2. Écrivez la post-condition du programme. (3pts)
3. Écrivez l'invariant de la première boucle (3pts)
4. Écrivez la fonction variant de la première boucle. (0.5pt)
5. Écrivez l'invariant de la seconde boucle (2pts)
6. Écrivez la fonction variant de la seconde boucle. (0.5pt)

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).