Nom : Prénom :

Numéro étudiant : Parcours :

Génie Logiciel Avancé — EIDD & M1 LP, GENIAL Examen écrit : correction

Questions à réponses libres et justifications : veillez à ce que vos réponses soient succinctes (brèves et compréhensibles).
Questions à choix multiples : cochez toutes les réponses valides (ne rien cocher si aucun choix n'est valide).

Exercice 1 Généralités (1.5 points)

Le génie logiciel consiste en l'application des méthodes de l'ingénierie au développement logiciel. Expliquez ce que l'on entend par là, plus précisément (une phrase).

Il s'agit de réaliser le développement logiciel dans le cadre d'un projet 1.5 pts

Exercice 2 Méthodes formelles, Testing (3 points)

1- En cours, nous avons vu un exemple de vérification formelle, en utilisant l'outil KeY, d'une fonction de recherche par dichotomie (binary search). Cette fonction prend un tableau a d'entiers triés par ordre croissant et un entier v et retourne l'indice de la case de a contenant v si v est un élément de a, -1 sinon.

Ci-dessous un listing qui montre une annotation KeY rajoutée au début de l'implémentation de la fonction sus-décrite.

```
/*@ public normal_behaviour
      @ requires (\forall int x; (\forall int y; 0 <= x && x < y; a[x] <= a[y]));
      @ ensures ((\exists int x; 0 <= x && x < a.length; a[x] == v) ? a[\result] == v :
            \result == -1);
      @*/
```

— Expliquez la signification des mots-clés **requires** et **ensures** (une phrase chacun)
requires introduit les préconditions 0.5 pt
ensures introduit les postconditions 0.5 pt

— Le bloc **requires** contient une erreur. Corrigez-la en rajoutant une clause à la conjonction $0 <= x \;\&\&\; x < y$.
$0 <= x \;\&\&\; x < y \;\&\&\; y < a.length$ 1 pt

2- En vous basant sur les explications données en cours des annotations **Before** et **After** de JUnit, cochez, parmi les choix suivants, les analogies qui vous semblent pertinentes (bien évidemment, le "correspond" ne remet pas en question les différences fondamentales entre le testing et la vérification formelle). 1 pt

Before (JUnit) correspond à **requires** (KeY) ☑
After (JUnit) correspond à **ensures** (KeY) ☐

Tournez la page...

Exercice 3 UML (2 points)

1- Soient A et B deux classes implémentées en Java. Vous ne disposez que de l'implémentation de B où vous remarquez qu'un *ArrayList* dont les éléments sont de type A figure parmi les attributs. Vous savez par ailleurs qu'il n'y a qu'une seule association entre A et B, que l'on nommera AB.

Que peut-on dire de AB (cochez les réponses valides) ? 1 pt

AB est une agrégation	☐	AB peut être une agrégation	☑
AB peut être une composition	☑	AB est une composition	☐

2- Vous créez désormais une classe C qui hérite de B. Existe-t-il une relation entre A et C ? Justifiez (deux phrases maximum). 1 pt

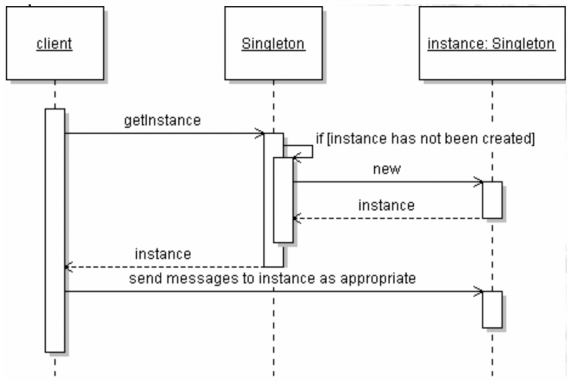
Oui ☐ Non ☐

Justification

Cela dépend de la visibilité de l'attribut de type A dans B : s'il n'est pas *private*, une association implicite existe entre C et A, sinon non

Exercice 4 Design Patterns (1.5 points)

Ci-dessous deux figures relatives au patron de conception *Singleton* vu en cours. Figure 1(a) illustre son diagramme de séquence alors que Figure 1(b) montre un exemple en Java où on essaie d'implémenter un générateur de nombres aléatoires en tant qu'un Singleton.



(a) Diagramme de séquence

```
// random number generator
public class RandomGenerator {
    private static RandomGenerator gen = new
        RandomGenerator();

    public static RandomGenerator getInstance() {
        return gen;
    }

    private RandomGenerator() {}

    ...
}
```

(b) Exemple de code

FIGURE 1 – Singleton

1- L'exemple de code (Figure 1(b)) n'est cependant pas tout à fait conforme au diagramme de séquence générique du patron (Figure 1(a)). Expliquez pourquoi (deux phrases maximum).
Dans le diagramme de séquence, l'instance est créée suivant le premier appel du client, tandis qu'elle existe indépendamment de cet appel dans le code 0.5 pt

2- Expliquez, en français ou en pseudo-code, comment vous auriez fait pour rendre l'exemple du code de la Figure 1(b) conforme au diagramme de séquence générique de la Figure 1(a).
1- initialiser *gen* à *null* 2- créer un bloc *if ... else* dans *getInstance()* dans lequel *gen* n'est instancié que s'il est égal à *null* 1 pt

Fin du sujet. Bon courage.