

CCE2 Test Logiciel – M1 ALMA

Durée : 1h30. Documents non autorisés.

7 janvier 2021

Nom : _____

Prénom : _____

Numéro étudiant : _____

	Points	obtenus
Exercice 1	2	
Exercice 2	3.5	
Exercice 3	1.5	
Exercice 4	1.5	
Exercice 5	1	
Exercice 6	2.5	
Exercice 7	1	
Exercice 8	1.5	
Exercice 9	2.5	
Exercice 10	1	
Exercice 11	2	
total	20	

Quelques informations :

- Le sujet comprend 11 exercices, pensez à parcourir le sujet rapidement avant de commencer.
- Le barème est donné à titre indicatif dans la marge droite, mais est susceptible de changer.
- Si la place manque dans une case, utilisez le verso de la feuille et indiquez le.

1 Questions de cours

Exercice 1

—/2

Cochez **vrai** ou **faux**. Une bonne réponse vaut 0.5. Une mauvaise réponse vaut -0.25 . Ne pas mettre de réponse vaut 0.

- (a) Il est impossible de créer un simulacre sans l'aide d'une bibliothèque comme Mockito.
 - ☐ vrai
 - ☐ faux
- (b) La stabilité est le facteur de testabilité concernant la capacité d'un logiciel à fournir les mêmes réponses aux mêmes questions posées plusieurs fois.
 - ☐ vrai
 - ☐ faux
- (c) Une suite de test qui couvre tous les chemins trouve toutes les erreurs.
 - ☐ vrai
 - ☐ faux
- (d) L'analyse de mutation permet de vérifier la bonne couverture des instructions.
 - ☐ vrai
 - ☐ faux

2 Méthodologie structurelle et qualité d'une suite de tests

La classe `SortedIntegerSet` (voir Figure 1) encapsule une liste d'entiers triés et un ensemble d'opérations pour manipuler cette liste. La méthode `containsAll` rend `true` si la liste courante contient tous les éléments de la liste `other` passée en paramètre.

```

1  import java.util.ArrayList;
2  public class SortedIntegerSet {
3      protected ArrayList<Integer> _list;
4
5      public SortedIntegerSet()    {_list = new ArrayList<Integer>();}
6      public int get(int index)    {return _list.get(index);}
7      public int size()            {return _list.size();}
8      public void add(int value)   {_list.add(value); Collections.sort(list);}
9
10     public boolean containsAll(SortedIntegerSet other) {
11         if (other.size() > this.size()) return false;
12         int i = 0;
13         int j = 0;
14         while (i < other.size()) {
15             if (this.get(j) < other.get(i)) {
16                 j++;
17                 if (j >= this.size())
18                     return true;
19             } else if (other.get(i) == this.get(j))
20                 i++;
21             else return false;
22         }
23         return true;
24     }
25 }

```

FIG. 1 : Code de la classe SortedIntegerSet

Exercice 2

—/3.5

Donnez le graphe de flot de contrôle de `containsAll` en utilisant les numéros de ligne pour nommer les sommets. Un sommet peut correspondre à plusieurs lignes.

Exercice 3

—/1.5

Listez l'ensemble des 1-chemins du graphe obtenu, en nommant chaque chemin avec une lettre (a, b, c , etc.).

- (a) remplacer `>` par `>=`
 - (b) remplacer `true` par `false`
 - (c) remplacer `this` par `other`

FIG. 2 : Opérateurs de mutation

Exercice 4

—/1.5

Définissez des scénarios de test pour couvrir tous les 1-chemins de `containsAll`. Précisez pour chaque scénario : les données d'entrée, l'oracle, le(s) chemin(s) couvert(s). S'il n'est pas possible de définir un oracle, indiquez le. Si un chemin ne peut être couvert, indiquez le. **Il n'est pas demandé d'écrire du code JUnit.**

Exercice 5

—/1

Y a-t-il un bug dans cette méthode ? Si oui, lequel et à quelle ligne ?

Exercice 6

—/2.5

La Figure 2 contient trois opérateurs de mutation. Appliqués sur `containsAll`, quels mutants seront produits ? Pour chaque mutant, indiquer quel numéro de ligne est changé et quel opérateur de mutation est utilisé. Nommez chacun de vos mutants M1, M2, M3, etc.

Exercice 7

—/1

Calculez le score de mutation des données de test que vous avez produites lors de l'exercice 4. Détaillez vos calculs en indiquant notamment quel scénario de test tue quel mutant.

3 Testabilité et doublures

Les Figures 4 et 5 fournissent des rappels des APIs de JUnit et de Mockito. Si du code Java est demandé, on suppose que tous les lignes `import` et `import static` ont déjà été écrites.

3.1 Présentation d'un système de domotique

Nous étudions dans cette partie un système dédié à la *domotique*. Ce système met partiellement en œuvre l'architecture MAPE (*Monitor, Analyze, Plan, Execute*) pour réaliser un système de gestion automatisée des volets d'un bâtiment en fonction des mesures de différents capteurs. **Toutes les figures sont situées à la fin du sujet.** La Figure 3 présente l'ensemble des classes et interfaces du système, que nous présentons ici :

HomeAutomation est l'interface définissant les services attendus par l'architecture MAPE : `monitor` pour aller chercher les informations auprès des capteurs, `analyze` pour analyser les données et décider quel état on souhaite atteindre, enfin `execute` pour exécuter les actions nécessaires pour atteindre cet état. Cette interface est implémentée par la classe `HomeAutomationImpl`.

Sensors est l'interface qui définit l'API permettant de lire les valeurs lues par les capteurs à disposition : `readWindDirection` pour la direction du vent, `readWindSpeed` pour la vitesse du vent, `readLuminosity` pour la luminosité. La luminosité est mesurée en *lux*, et la vitesse du vent est mesurée en *km/h*. Cette interface est implémentée par `RealSensors`, qui nécessite l'adresse du port auquel sont branchés les capteurs physiquement.

ShuttersControl est l'interface définissant l'API permettant de contrôler les volets roulant du bâtiment : `close` pour les fermer, `open` pour les ouvrir. Cette interface est implémentée par `RealShuttersControl`, qui nécessite l'adresse du port auquel est branché l'émetteur communiquant avec les volets roulants.

System est la classe qui gère le processus MAPE. Pour simplifier, elle ne contient qu'une méthode `runOnce` qui va effectuer une fois la boucle MAPE en appelant les méthodes de `HomeAutomation` dans le bon ordre.

Exercice 8

—/1.5

On souhaite tester la classe `HomeAutomationImpl`. Quels obstacles doivent être pris en compte ? Pour chaque obstacle, proposez une manière de le résoudre.

Exercice 9

—/2.5

En vous basant si nécessaire sur votre réponse à la question précédente, écrivez un test JUnit pour la méthode `monitor` (dont la spécification est donnée Figure 6) avec ce scénario : la luminosité est 200, la vitesse du vent 12, et la direction du vent `North`.

Exercice 10

—/1

Quel(s) type(s) de doublure nous permettrait de vérifier que la méthode `execute` respecte rigoureusement le comportement décrit dans le diagramme de séquence donné en Figure 7 ? Justifiez votre réponse.

Exercice 11

—/2

Écrivez un/des test(s) JUnit pour vérifier que la méthode `execute` respecte rigoureusement le comportement décrit dans le diagramme de séquence donné en Figure 7.

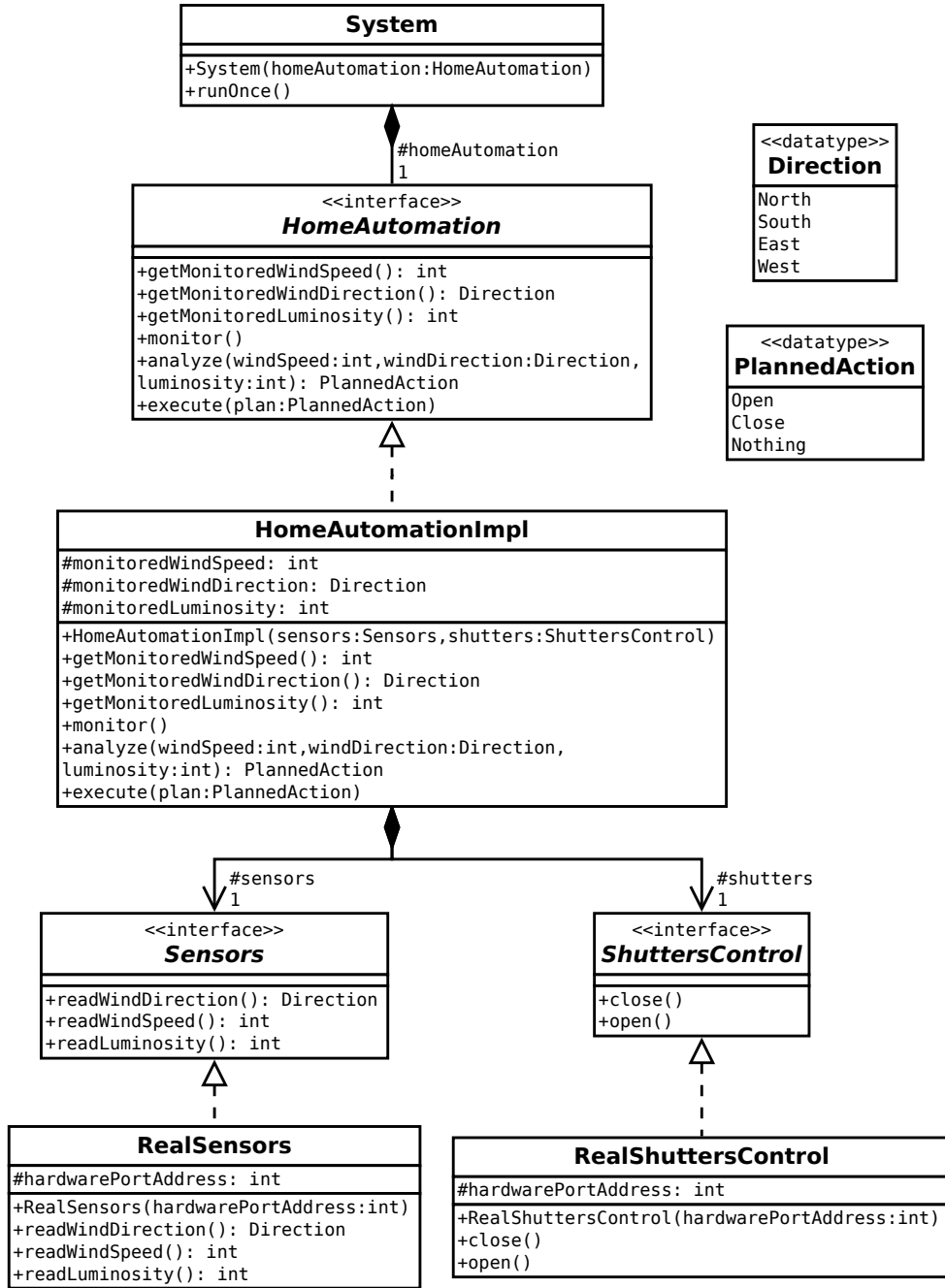


FIG. 3 : Classes et interfaces du système de domotique étudié. Note : *sensors* signifie *capteurs* et *shutters* signifie *volets*.

`assertEquals(Object, Object)` est une assertion que deux objets (ou valeurs) sont égaux (égales).
`assertTrue(boolean)` est une assertion qu'un booléen vaut `true`.
`assertTrue(boolean)` est une assertion qu'un booléen vaut `true`.

FIG. 4 : Rappel d'une partie de l'API de JUnit

- `MyType m = mock(MyType.class)` permet de créer une doublure `m` instance de la classe (ou de l'interface) `MyType`.
- `when(m.someMethod()).thenReturn("some result")` permet de configurer une doublure pour rendre une valeur lors d'un prochain appel à une méthode.
- `verify(m).someMethod()` est une assertion que la méthode `someMethod` a bien été appelée au moins une fois au cours du test.

FIG. 5 : Rappel d'une partie de l'API de Mockito

```
1  /**
2   * This operation retrieves information from all the sensors and store them
3   * for later use. Right after "monitor()" has been called, retrieved
4   * information must be accessible from the three operations
5   * "getMonitoredWindSpeed()", "getMonitoredWindDirection()" and
6   * "getMonitoredLuminosity()".
7   */
8  public void monitor();
```

FIG. 6 : Spécification de l'opération `monitor` de l'interface `HomeAutomation`

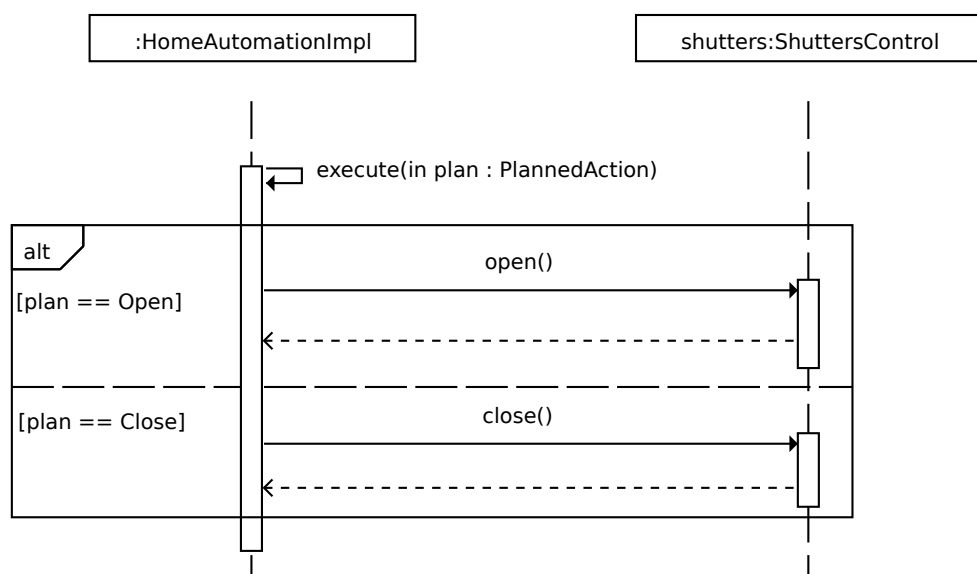


FIG. 7 : Spécification de la méthode `execute`