

Contrôle Continu

24 septembre 2018

Durée : 3 heures

Instructions

- L'examen sera réalisé avec **Eclipse** et **git**.
- Utiliser
 - JDK ≥ 1.8
 - JUnit 5
 - Eclipse Photon
- Le repository à utiliser est https://github.com/CNAMPRO/18_stmn_java.
- Travailler sur **votre** branche.
- Créer un nouveau projet **Eclipse CC1** localisé sous [racine_18_stmn_java]/Exam/CC1, [racine_18_stmn_java] étant le répertoire de votre clone git de https://github.com/CNAMPRO/18_stmn_java.
- Pour chaque exercice, créer un nouveau package portant le nom de l'exercice en minuscule (e.g. **exercice1**) et y inclure toutes les classes créées pour répondre à l'exercice.
- Lorsque des vérifications ou des tests sont demandés dans un exercice, ne **PAS** utiliser de `main ...`
- Ne **PAS** modifier la signature des fonctions/méthodes donnée dans les énoncés.
- À la fin de l'examen, faire un `commit+push` de tout le **code** correspondant à vos réponses.

hint :

```
git add <files>
git commit
git push
```

- Vérifier que vos réponses sont disponibles dans votre branche sur https://github.com/CNAMPRO/18_stmn_java.

Exercice 1

(Barème 15%)

Dans \mathbb{R}^3 , on définit le produit vectoriel de deux vecteurs $\mathbf{u} = (u_x, u_y, u_z)$ et $\mathbf{v} = (v_x, v_y, v_z)$ comme étant :

$$\mathbf{u} \times \mathbf{v} \stackrel{\text{def}}{=} \begin{pmatrix} u_y v_z - u_z v_y \\ u_z v_x - u_x v_z \\ u_x v_y - u_y v_x \end{pmatrix}$$

1. Implémenter le produit vectoriel (en tant que méthode statique d'une classe).

Indications :

- Veiller à traiter **les** cas d'erreurs.
- *hint :*

```
public static double[] crossProduct(double[] v1, double[] v2) throws Exception
```

2. Tester l'implémentation.

- Vérifier que le produit vectoriel de
 - $\mathbf{X} = (1, 0, 0)$ par $\mathbf{Y} = (0, 1, 0)$ est égal à $\mathbf{Z} = (0, 0, 1)$.
 - $\mathbf{Z} = (0, 0, 1)$ par $\mathbf{X} = (1, 0, 0)$ est égal à $\mathbf{Y} = (0, 1, 0)$.
 - $\mathbf{Y} = (0, 1, 0)$ par $\mathbf{Z} = (0, 0, 1)$ est égal à $\mathbf{X} = (1, 0, 0)$.

- Tester **le(s)** cas d'erreurs.

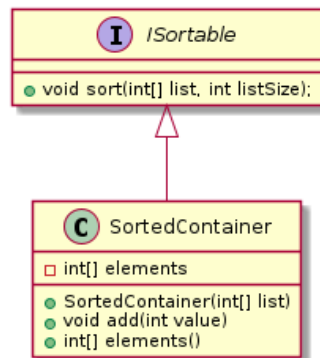
hint :

```
// When using JUnit 5
@Test
public final void test() throws Exception {
    //test code
    assertThrows(Exception.class, () -> {
        // my instruction throwing an exception
        // don't forget final ; at end of line
    });
    //more test code
}
```

Exercice 2

(Barème 20%)

Etudier le diagramme de classe ainsi que la javadoc correspondante ci-dessous :



```

/***** JAVADOC *****/
/**
 * Sorts the list given in input (the input list is modified).
 * @param list
 *   the list to be sorted.
 * @param listSize
 *   effective size of list
 */
public void sort(int[] list, int listSize);

/**
 * Constructs a SortedContainer from the input list.
 * The elements of the constructed SortedContainer are sorted.
 * @param list
 *   list of potentially unsorted elements.
 */
public SortedContainer(int[] list);

/**
 * Adds an element to the SortedContainer.
 * When this function returns, the elements of SortedContainer remain sorted.
 * @param value
 *   the new value to add.
 */
public void add(int value);

/**
 * Get a *copy* of the elements *managed* by this container.
 * Returns an array whose size is the current number of elements in the container.
 */
public int[] elements();

```

La classe `SortedContainer` représente un tableau d'elements **TRIÉS À TOUT INSTANT**.

1. En respectant les indications fournies, implémenter complètement le diagramme de classe.

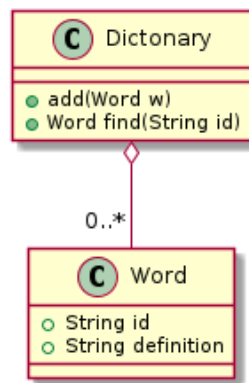
Remarques importantes :

- Ne **PAS** changer le prototype des fonctions/méthodes
 - Ne **PAS** utiliser les classes de la librairie standard Java (e.g. ArrayList, Vector...)
 - Certains éléments nécessaires à l'implémentation n'apparaissent pas dans le diagramme de classe. Ne pas hésiter à ajouter les éléments qui vous semblent indispensables (e.g. attribut...)
 - Le choix de la fonction de tri est laissé libre. On pourra par exemple utiliser un tri à bulles.
 - Pour s'assurer que les éléments de SortedContainer sont triés, on utilisera bien sûr la fonction sort.
2. Tester l'implémentation.
 - Créer un SortedContainer avec la liste {3,2,6,5,4,9,7,8,1} et vérifier que les éléments de l'objet créé sont triés.
 - Ajouter l'élément 4 à l'objet précédent et vérifier que ses éléments restent triés.

Exercice 3

(Barème 25%)

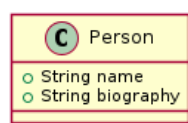
Etudier le diagramme de classe ci-dessous :



```
/**
 * ***** JAVADOC *****
 */
/**
 * Adds a new word to the dictionary
 * @param w
 * the new word to add
 */
public void add(Word w);

/**
 * Search for an entry in the dictionary matching a word
 * @param id
 * id of word.
 * @return the word if found, null otherwise
 */
public Word find(String id);
```

1. Implémenter complètement le diagramme de classe avec les indications fournies. On pourra utiliser les classes de la librairie standard Java.
2. En créant les mots {"meh", "onomatopée"}, {"fonction", "Ensemble d'opérations"} et en les ajoutant à un dictionnaire :
 - Retrouver meh dans le dictionnaire et vérifier que sa définition est onomatopée
 - Vérifier que chien n'est pas dans le dictionnaire.
3. On souhaite faire évoluer la classe Dictionary afin qu'elle puisse gérer autre chose que seulement des objets de la classe Word. En l'occurrence, on souhaiterait qu'elle gère également des objets de la classe Person décrite ci-dessous :



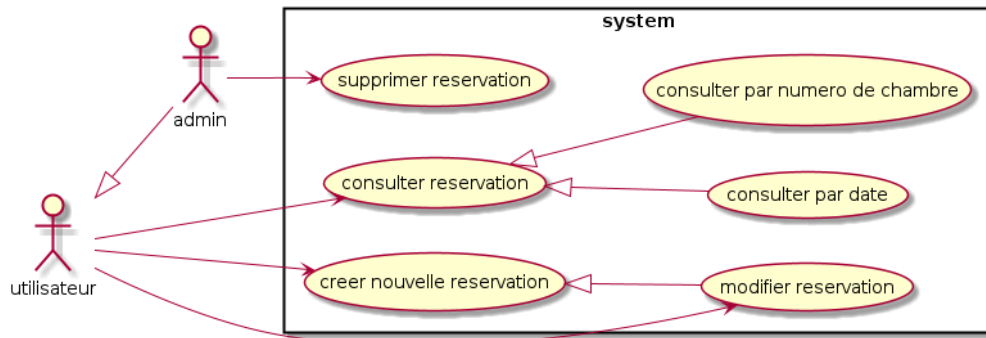
L'objet Dictionary gère donc des objets ayant un *identifiant* et une *description*.

- Proposer une nouvelle implémentation afin de répondre au nouveau besoin. On s'efforcera toutefois de minimiser au strict nécessaire les changements à apporter.
- Ne **PAS** modifier les classes existantes, mais en créer des nouvelles (e.g. `NewDictionary`, `NewWord...`)
- Indications : On pourra par exemple utiliser des **interfaces**.

Exercice 4

(Barème 40%)

On souhaite concevoir et réaliser un logiciel permettant de gérer la réservation de chambres d'hôtel. Les besoins pour un tel logiciel sont exprimés par le diagramme ci-dessous :



On suppose que les réservations pour une chambre se font pour **une journée entière**.

1. Proposer une conception permettant de répondre aux besoins du logiciel.

Indication :

- On s'attachera à ne pas complexifier plus que nécessaire la conception.
- On pourra faire toute hypothèse simplificatrice judicieuse.
- La conception sera décrite dans un document texte, accompagnée éventuellement de schéma (fichier image).

2. Proposer une implémentation.
3. Tester les cas d'utilisations.