Contrôle Continu

6 novembre 2017

Instructions

- L'examen sera réalisé avec **Eclipse**.
- Créer un nouveau projet dans **Eclipse** que l'on nommera **Exam1-<votre_nom>**.
- 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` ...
- À la fin de l'examen, créer une archive (*e.g.* `.tgz`, `.zip`) du répertoire correspondant au projet **Exam1-<votre_nom>**

Exercice 1

On définit le produit scalaire de deux vecteurs $\mathbf{v}$ et $\mathbf{w}$ de $\mathbb{R}^n$, $n \geq 1$ comme étant :

$$\mathbf{v} \cdot \mathbf{w} \stackrel{\text{def}}{=} \sum_{i=1}^{n} v_i w_i.$$

1. Implémenter le produit scalaire (en tant que méthode statique d'une classe).
   Indications :
   - Veiller à traiter **les** cas d'erreurs.
   - *hint :*

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

2. Tester l'implémentation.
   - Vérifier que le produit scalaire de $\mathbf{v} = (1,2,3)$ par $\mathbf{w} = (4,5,6)$ est égal à 32.
   - Tester **les** cas d'érreurs.
     hint :

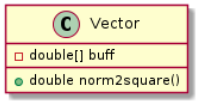
     ```
     @Rule
     public ExpectedException expected = ExpectedException.none();

     @Test
     public final void test() throws Exception {
       //test code
       expected.expect(Exception.class);
       //more test code
     }
     ```

3. On définit le carré de la norme euclidienne d'un vecteur $\mathbf{v}$ comme étant :

   $$||\mathbf{v}||^2 \stackrel{\text{def}}{=} \mathbf{v} \cdot \mathbf{v}$$

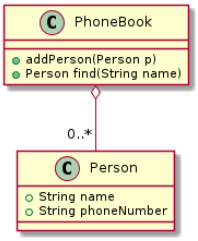
   La classe `Vector` est définie par le diagramme ci-dessous :

   - Implémenter la classe `Vector`
     Indications : Ré-utiliser bien sûr l'implémentation précédente du produit scalaire.
   - Soient $\mathbf{v} = (1,2,3)$ et $\mathbf{w} = (2,3,1)$. Vérifier que $||\mathbf{v}||^2 = ||\mathbf{w}||^2$

Exercice 2

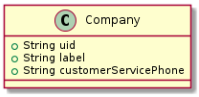
Etudier le diagramme de classe ci-dessous :




```
/******** JAVADOC **************/
/**
* Adds a new person to the phone book
* @param p
*  the new person to add. A new entry is created even if the p already existed in the phone book.
*/
public void addPerson(Person p);

/**
* Search for an entry in the phone book matching a name
* @param name
*  name of the person to find.
* @return the person if found, null otherwise
*/
public Person find(String name);
```

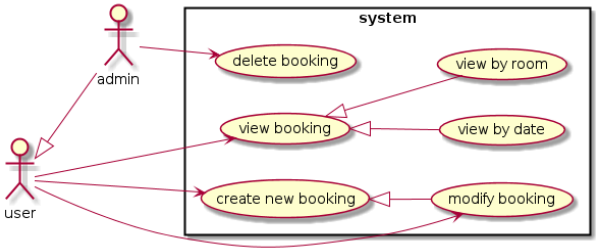
1. Implémenter complètement le diagramme de classe avec les indications fournies.
2. En créant les personnes {Joe, 04.94.10.20.01}, {Alan, 04.94.10.20.02}, {Mandy, 04.94.10.20.03} et en les ajoutant à un répertoire :
   - Vérifier que le numéro d'Alan est le 04.94.10.20.02.
   - Vérifier que Bob n'est pas dans le répertoire.
3. On souhaite faire évoluer la classe `PhoneBook` afin qu'elle puisse gérer autre chose que seulement des objets de la classe `Person`. En l'occurence, on souhaiterait qu'elle gère également des objets de la classe `Company` décrite ci-dessous :

   Pour les objets de la classe `Company`, on stocke dans l'objet `PhoneBook`, le `label` et le `customerServicePhone`.
   - Proposer une nouvelle implémentation afin de répondre au nouveau besoin (on créera de nouvelles classes, sans modifier celles existantes).
     Indications : On pourra par exemple utiliser des **interfaces**.

Exercice 3

On souhaite concevoir et réaliser un logiciel permettant de gérer la réservation de créneaux horaires pour une salle. Les besoins pour un tel logiciel sont exprimés par le diagramme ci-dessous :



Pour des raisons de simplicité, on suppose que les réservations pour une salle 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.
2. Proposer une implémentation.
3. Tester les cas d'utilisations.