TP « Generics » Coffre Fort

TP1 :

Vous allez coder une classe nommée « CoffreFort » qui aura pour objectif de stocker n'importe quelle instance d'un objet et de la restituer.

- Cette classe doit prendre un type paramétré nommé <E> (par convention)

- Lors de son instanciation, le coffre-fort se fabrique une clé (utiliser `UUID.randomUUID()`) qui est stockée dans un attribut nommé « key »

- Cette clé ne peut être délivrée qu'une fois et une seule au moyen de la méthode « `getKey()` », sinon une exception `KeyAlreadyGivenException (Exception)` est levée (pour cela, utiliser un attribut de type `boolean :keyHasBeenGiven` initialisé à false)

- Le coffre-fort peut avoir plusieurs états : OPEN ou CLOSED, utiliser un type énuméré simple : `public Enum Etat {OPEN, CLOSED}`

- L'état du coffre est modifié grâce aux méthodes :
  - `void open(UUID k) throws LockerAlreadyOpenException (RuntimeException)` levée si le coffre est déjà ouvert
  - `void close(UUID k) throws LockerAlreadyClosedException (RuntimeException)` levée si le coffre est déjà fermé
  - Ces 2 méthodes vérifient que la clé k passée en paramètre est la bonne pour pouvoir ouvrir/fermer le coffre-fort (autrement dit : l'UUID k passé en paramètre est égal à l'attribut « key » du coffre-fort)

- Quand le coffre est ouvert, il doit permettre de stocker et récupérer un élément :
  - `E getElement()` : récupère l'élément SI (et seulement si) le coffre est ouvert
  - `E putElement(E newElement)` : met un nouvel élément à la place éventuellement d'un élément déjà présent dans le coffre-fort et retourne l'élément qui était présent, le cas échéant, sinon « `null` » ;
  - Quand le coffre est fermé, ces 2 méthodes doivent lever une `LockerNotOpenException (Exception)`

  Le code « client » fourni, qui utilise la classe « CoffreFort », doit fonctionner dans un « main ».

- Coder une batterie de tests unitaires Junit4 qui teste les levées d'exceptions

TP 2 :

Le coffre-fort peut maintenant stocker plusieurs objets, mais de type équivalent, au moyen d'une `List`.

- Le stockage interne se fait au moyen d'une `ArrayList <E>` paramétrée avec les « generics »
- Implémenter « `int size()` » qui renvoie le nombre d'éléments dans le coffre-fort
- Implémenter « `void putElement(E nouvelElement)` » qui ajoute un élément dans le coffre-fort
- Implémenter « `E getElementAt(int index)` » qui renvoie l'élément à l'index donné du coffre-fort
- Ces 2 dernières méthodes doivent lever une « `LockerNotOpenException` » si le coffre-fort n'est pas ouvert

Exemple de structure du projet :