TP N°2 *Associations unidirectionnelles en UML et listes en Java*

Objectifs

- UML : les associations unidirectionnelles dans un diagramme de classes
- JAVA : Lire de la documentation et manipuler des listes

Exercice 1 *Lecture de documentation*

Rends toi sur la documentation de `ArrayList` pour répondre aux questions
`https://docs.oracle.com/javase/7/docs/api/index.html?java/util/ArrayList.html`

1.1 Dans l'extrait de code ci-contre, identifie les lignes qui provoquent nécessairement une erreur à la compilation en raison d'un appel à une méthode qui n'existe pas ou qui est mal utilisée.

```
1  List<Pokemon> pokedex = new ArrayList<>();
2  ...
3  pokedex.add(etourmi);
4  pokedex.add(chenipan, 4);
5  pokedex.add(3, abo);
6  pokedex.set(feurisson);
7  pokedex.set(bulbizarre, 12);
8  pokedex.set(4, griknot);
9  pokedex.get(feurisson);
10 pokedex.get(12);
11 pokedex.pop(abo);
12 pokedex.pop(3);
13 pokedex.remove(dardagnan);
14 pokedex.remove(2);
15 pokedex.removeRange(3, 5);
```

1.2 Dans l'extrait de code suivant, précise le type des variables `mystereX`

```
mystere1 = pokedex.subList(1, 4);
mystere2 = pokedex.size();
mystere3 = pokedex.get(2);
mystere4 = pokedex.add(hydragon);
mystere5 = pokedex.remove(dardagnan);
```

Exercice 2 *Des magasins*

On veut gérer les magasins d'une ville. Ces magasins peuvent être ouverts le lundi et/ou le dimanche.

```
public class Magasin {
    private String nom;
    private boolean ouvertLundi;
    private boolean ouvertDimanche ;

    public Magasin(String nom, boolean lundi, boolean dimanche) {
    }

    // les getters

    @Override
    public String toString() {
        ...
    }
}
```

```java
import java.util.ArrayList ;
import java.util.List ;

public class Ville {
    private String nom;
    private List<Magasin> magasins;
    public Ville(String nom) {
    }
    public void ajouteMagasin(String nom, boolean lundi, boolean dimanche) {
    }
    public List<Magasin> ouvertsLeLundi() {
        ...
    }
    @Override
    public String toString() {
        ...
    }
}
```

2.1 Recopie ces codes en remplaçant les `...` par du code qui ne fait rien, mais qui est syntaxiquement correct et permet de compiler. Vérifie que tes classes compilent sans erreur.
Note : Pour le moment, ces classes ne font RIEN.

2.2 Crée une classe `ExecutableMagasins` contenant la modélisation de la situation suivante :

- Le magasin Fleurus est ouvert le lundi et fermé le dimanche.
- La ville de `Trainou` contient les magasins suivants :
  - Le magasin BeauMagasin qui est ouvert et le lundi et le dimanche.
  - Le magasin Venir fermé le lundi et le dimanche
  - Le magasin Magnifique fermé le lundi et ouvert le dimanche
  - Le magasin Bauchamp ouvert le lundi et fermé le dimanche.

2.3 Dans ton exécutable, ajoute du code de façon à afficher la liste des magasins de Trainou puis complète le code des classes `Magasin` et `Ville`.

2.4 Complète ton exécutable de façon à appeler au moins une fois chacune des méthodes des classes `Magasin` et `Ville` pour vérifier que ton code est correct.

Exercice 3 *Plan de table*
Code les classes correspondant à l'exercice 3 du TD2.

Exercice 4 *La fraternité de l'anneau*
Code les classes correspondant à l'exercice 4 du TD2.