TD Location de voitures

Préliminaire sur les listes

L'**interface** `List` est définie dans le paquetage `java.util`. Elle permet, sans surprise, de définir des données représentant des listes. En java les listes sont typées, c'est-à-dire que tous les éléments d'une liste doivent être d'un même type. Ce type est précisé en paramètre du type `List`. Ainsi on utilise la notation : `List<E>` pour créer une liste dont les éléments sont tous de type `E`. `E` doit être remplacé (prendre pour valeur) n'importe quel type java valide.

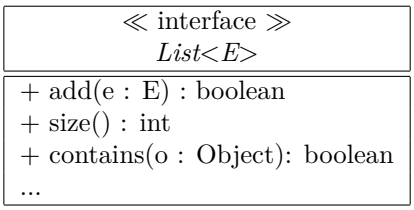
La classe `ArrayList`

Dans cet exercice nous pourrons utiliser la classe `java.util.ArrayList<E>` qui implémente l'interface `List<E>`. Pour créer une liste on peut donc écrire :

- `List<String> l = new ArrayList<String>();` pour créer une référence `l` représentant une liste ne contenant que des chaînes de caractères
- `List<Recyclable> trashcan = new ArrayList<Recyclable>();` ppour créer une liste `trashcan` dont les éléments sont de type `Recycable`, on peut alors ranger dans cette liste des objets `Paper` ou `Bottle`[1]

Quelques méthodes

L'interface `java.util.List` définit un certain nombre de méthodes. En voici quelques-unes suffisantes pour cet exercice (on devine facilement leur rôle) . Nous aurons l'occasion d'en découvrir d'autres ultérieurement.



Le type `E` qui apparaît dans la méthode `add` est le même que le type choisi lors de la création de la liste. Pour la référence `l` ci-dessus la signature de la méthode `add`[2] devient donc alors :

```
public boolean add(String e)
```

Parcours d'une liste

Enfin pour itérer sur une liste et appliquer une même opération à chacun de ses élements on peut utiliser la syntaxe « à la *for-each* » suivante :

```
List<Recyclable> trashcan = new ArraytList<Recycbale>();
...    // ajouts d'éléments dans la liste
for(Recyclable r :  trashcan) {
   // r prend successivement la valeur de chacun des éléments de la liste
   r.recycle();
}
```

Le type `List` sera revu plus en détail ultérieurement.

[1] cf. cours
[2] le booléen permet de savoir si l'élément `e` a pu être ou non ajouté à la liste

L'agence de location

(Les classes/interfaces de cet exercice seront à ranger dans un paquetage `agency`.)

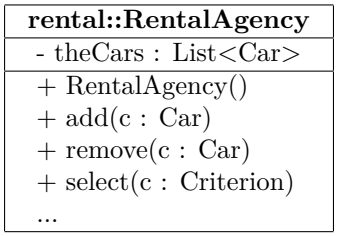
Une agence de location (*rental agency*) de voitures offre à ses clients la possibilité de choisir la voiture louée en fonction de différents critères.
Les voitures sont définies par une marque (*brand*), un nom de modèle (*model*), une année de production (*production year*) et un prix de location à la journée (*daily rental price*). Pour simplifier les deux premiers paramètres seront des objets de la classe `String` et les deux derniers seront des `int`. Deux voitures sont considérées égales si tous leurs attributs sont égaux.

Q 1 . Donner le diagramme UML de la classe `Car` pour laquelle on souhaite disposer des méthodes habituelles `equals` et `toString`.

Il est possible de sélectionner parmi les voitures à louer toutes les voitures satisfaisant un critère (*criterion*) donné.
On définit l'interface `Criterion` ainsi :

```
public interface Criterion {
  /**
   * @param c the car that must verify the criterion
   * @return true if and only id the car c complies with this criterion.
   * c is said to satisfy this criterion.
  */
  public boolean isSatisfiedBy(Car c);
}
```

On suppose une classe `RentalAgency` définie (au minimum) ainsi :



Q 2 . Donnez le code de la méthode `remove` de la classe `RentalAgency` qui permet de supprimer une voiture à l'agence. Cette méthode doit déclencher une exception `UnknownCarException` si la voiture n'est pas gérée par l'agence.

On suppose la classe d'exception `UnknownCarException` définie dans le paquetage `agency`.

Q 3 . Donnez des lignes de code qui, en supposant que la référence `agency` est de type `RentalAgency` et a été initialisée, supprime une voiture de l'agence.

Ces lignes de code affichent un message sur la sortie standard rendant compte du résultat de l'opération : selon que la voiture a pu ou non être supprimée de l'agence.

Q 4 . Donnez le code de la méthode `select` dont le résultat est la liste des véhicules, de l'attribut `allCars`, qui satisfont le critère `c` passé en paramètre.

Q 5 . Donnez le code d'une classe `PriceCriterion` qui est un critère satisfait par toutes les voitures dont le prix est inférieur à un prix fixé à la construction du critère.

Q 6 . Donnez le code d'une classe `BrandCriterion` qui est un critère satisfait par toutes les voitures d'une marque donnée. La marque est précisée à la construction du critère.

Q 7 . En supposant que la référence `agency` est de type `RentalAgency` et a été initialisée, donnez la ou les lignes de code permettant d'afficher toutes les voitures de cette agence dont le prix est inférieur à `100`.

Q 8 . On peut naturellement souhaiter faire des intersections de critères, ce qui revient à appliquer le **et** logique entre les critères . On obtient alors un nouveau critère satisfait si tous les critères qui le composent sont satisfaits.

Définissez une classe `InterCriterion` qui permet de définir des critères par intersection de **deux** critères. Les deux critères dont on fait l'intersection sont précisés à la construction.

Q 9 . Donnez la ou les lignes de code permettant de créer un critère intersection d'un critère pour la marque `"Timoleon"` et d'un critère pour un prix inférieur à `100`.