Aix*Marseille université
Initiative d'excellence
Année 2018-19

Domaine Sciences et Technologies
Licence informatique

Programmation 2 : TD 6
Code UE : SIN3U02

Interfaces et collections

1 Préliminaire sur les listes

L'interface `List<E>` est définie dans le package `java.utilet` est implémenté par la classe `java.util.ArrayList<E>`. L'interface `java.util.List<E>` définit un certain nombre de méthodes :

- `boolean add(E e)` : Appends the specified `element` to the end of this list.
- `boolean contains(Object o)` : Returns `true` if this list contains the specified `element`.
- `E get(int index)` : Returns the element at the specified `position` in this list.
- `boolean isEmpty()` : Returns `true` if this list contains no elements.
- `boolean remove( Object o)` : Removes the first occurrence of the specified element from this list, if it is present.
- `int size()` : Returns the number of elements in this list.

2 L'agence de location

Une agence de location (*rental agency* en anglais) de voitures (*car*) 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`, le troisième sera de type `int` et le dernier de type `double`. Deux voitures sont considérées égales si tous leurs attributs sont égaux.

La classe `Car` contiendra donc les propriétés suivantes :

- `String brand`
- `String model`
- `int productionYear`
- `double dailyRentalPrice`

Elle contiendra aussi le constructeur :

- `public Car(String brand, String model, int productionYear, int dailyrentalPrice)`

ainsi que les méthodes :

- `public String brand()`
- `public String model()`
- `public int productionYear()`
- `public double dailyRentalprice()`
- `public boolean equals(o : Object)`
- `public String toString()`

La classe `RentalAgency` contiendra une propriété `List<Car> cars`, un constructeur `public RentalAgency()` ainsi que les méthodes suivantes (pour le moment) :

- `public void add(Car car)`
- `public void remove(Car car)`
- `public boolean contains(Car car)`

3 Exercices

1. Écrivez la méthode `remove` de la classe `RentalAgency`, qui permet de supprimer une voiture à l'agence.
2. Écrivez la méthode `add` de la classe `RentalAgency`. Si la voiture passée en argument est déjà propriété de l'agence, on prendra garde à ne pas la dupliquer.

3. On souhaite pouvoir sélectionner parmi les voitures à louer toutes les voitures satisfaisant un critère (`criterion`) donné. Programmez une méthode `selectByBrand` dans la classe `RentalAgency` dont le résultat est la liste des véhicules de l'agence dont la marque est la même que celle passée en paramètre de la méthode.
4. De la même façon programmez une méthode `selectByMaxPrice` de la classe `RentalAgency` dont le résultat est la liste des véhicules dont le prix de location à la journée est inférieure au prix passé en paramètre de la méthode.
5. Comparez ces deux méthodes, qu'en pensez-vous ? Imaginez d'autres critères pouvant être sélectionnés.
6. On veut maintenant définir la notion de critère de sélection pour les voitures. Un critère peut être satisfait par une voiture, cela signifie que le critère filtre (ou sélectionne) cette voiture. Définir une interface dans le but (futur) de généraliser les deux méthodes.
7. On pourrait imaginer d'autres applications, où l'on veut filtrer des appartements, des ordinateurs, ... Modifier l'interface pour la rendre générique. Cette interface existe en Java sous le nom `Predicate<T>`.
8. Programmez une classe `MaxPriceCriterion`. Les instances de cette classe sont des critères imposant un prix maximum sur les voitures. Cette classe doit passer avec succès le test suivant :

```
public class MaxPriceCriterionTest {
    @Test
    public void testIsSatisfied() {
        Predicate<Car> cheap = new MaxPriceCriterion(100);
        Car c90 = new Car("Tim","Oleon",2015,90);
        Car c120 = new Car("Tim","Oleon",2016,120);
        assertTrue(cheap.test(c90));
        assertFalse(cheap.test(c120));
    }
}
```

9. Ajoutez une méthode `select` dans la classe `RentalAgency` qui corresponde à :

```
/**
* return the list of cars of this agency that satisfy the specified
* {@code criterion}. The returned cars are then << filtered >> by the criterion.
*
* @param criterion the criterion that the selected cars must satisfy
* @return the list of cars of this agency that satisfy the given criterion
*/
public List<Car> select(Predicate<Car> criterion)
```

   En supposant que `RentalAgency agency` a été initialisée, quelles instructions permettent d'afficher toutes les voitures de cette agence dont le prix est inférieur à 100 ?
10. 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 qui est satisfait lorsque tous les critères qui le composent sont satisfaits. Définissez une classe InterCriterion qui permet de définir des critères par intersection de plusieurs critères. On donnera un constructeur prenant deux critères, et un constructeur prenant une collection de critères.
11. Comment afficher maintenant toutes les voitures de marque "Tim" et de prix inférieur à 100 ?