TP3 JAVA

Concepts abordés

- classes, relation de composition, classes collection: List, Set
- interfaces Comparable et Comparator
- classes imbriquées, classes static, classes anonymes, interface de constantes
- mapping de fonctions

Consignes générales de travail

- Commencez chaque TP dans un nouveau *Projet Java (ici TP3)*
- Créez vos classess dans un **paquetage automobile**
- Écrivez chaque nouvelle classe dans son propre fichier (portant le nom de la classe).
- Pour ce TP, déclarez systématiquement vos attributs en private et vos méthodes en public ou private.
- Définissez systématiquement une méthode toString():String *pour chaque classe* que vous écrivez.
- **Lisez chaque exercice en entier avant de commencer vos réponses.**

I Modéliser un compteur kilométrique

Un compteur kilométrique se compose d'un ***enregistrement totalisateur*** et d'un ***enregistrement partiel***. Ces deux enregistrements conservent les distances parcourues sous la forme d'un nombre réel.

Quand un compteur est créé, la valeur de ces deux enregistrements est toujours nulle.

L'enregistrement partiel peut ***être remis à zéro*** (mais pas à une autre valeur), lorsqu'il atteint 1 000km, il est automatiquement remis à 0.

Outre cette opération, la seule façon de modifier la valeur des deux enregistrements est d'***ajouter des kilomètres*** en roulant.

Définir en Java une classe **Compteur** comportant :

- deux attributs d'instance : **totalisateur** et **partiel**
- **une constante de classe, initialisée à 1 000**
- un c**onstructeur** sans paramètre
- les méthodes publiques d'accès en lecture pour les deux enregistrements
- la méthode publique d'accès en écriture pour l'enregistrement partiel
- une méthode publique **resetPartiel** de remise à zéro du champ partiel
- une méthode publique **add**, qui incrémente les deux champs du nombre de kilomètres (un nombre réel) spécifié en paramètre
- la méthode **toString**, qui retourne une représentation textuelle d'un compteur et de son état conforme à l'exemple suivant :
  compteur = [totalisateur = 500 | partiel = 108]

Tests: activer les tests sur les compteurs dans le programme de tests fourni

```
######## TESTS DES COMPTEURS ########
compteur = [Totalisateur = 0 | Partiel = 0];
compteur = [Totalisateur = 200 | Partiel = 200];
compteur = [Totalisateur = 500 | Partiel = 500];
compteur = [Totalisateur = 500 | Partiel = 0];
compteur = [Totalisateur = 650 | Partiel = 150];
```

II Modéliser un véhicule

Un véhicule se compose d'un ***numéro d'immatriculation***, d'un ***compteur,*** de la ***jauge*** du ***réservoir,*** et d'une ***consommation kilométrique***. Les véhicules seront représentés par des instances de la classe **Vehicule**.

Le ***numéro d'immatriculation*** d'un nouveau véhicule est déterminé par la valeur d'un **registre** commun à tous les véhicules ; la valeur de ce **registre** est automatiquement incrémentée de 1 à chaque création d'un nouveau véhicule. Ce **registre** ne doit pas être accessible en dehors de la classe Vehicule. Le numéro d'immatriculation d'un véhicule ne doit pas être non plus accessible en dehors de la classe, mais sa valeur peut être obtenue par une méthode de lecture à partir de n'importe quelle classe.

Le ***compteur kilométrique*** est celui modélisé dans la partie I. Il est créé lors de l'instanciation d'un véhicule. On obtient une référence à ce compteur par une méthode **getCompteur** de la classe Vehicule.

La ***capacité*** du réservoir est de 50.0 litres pour tous les véhicules et ne varie pas. Elle ne doit pas être accessible de l'extérieur de la classe.

La ***jauge*** du réservoir indique la quantité de carburant qui reste dans le réservoir. Elle propre à chaque véhicule. Elle ne doit pas être accessible de l'extérieur de la classe, mais sa valeur peut être obtenue par une méthode de lecture **getJauge** à partir de n'importe quelle classe. A chaque fois qu'on fait le plein, cette jauge est remise à la valeur de la capacité du réservoir ; il est possible de ne remplir qu'une fraction du réservoir.

La ***consommation*** d'un véhicule (nombre de litres de carburant pour 100 kilomètres parcourus) est propre à chaque véhicule et est fixée à la création du véhicule. Cette consommation ne doit pas être accessible de l'extérieur de la classe.

Les trois opérations que l'on peut faire sur un véhicule sont : ***mettre de l'essence***, ***faire le plein*** et ***rouler*** sur une certaine distance. En roulant on consomme du carburant, donc on ne peut rouler que sur la distance permise par le réservoir et la consommation du véhicule ; le compteur n'est incrémenté que de la distance effectivement parcourue. On décide de rouler jusqu'à ce que le réservoir soit vide, puis on fait le plein.

En fonction des informations ci-dessus, définir en Java une classe **Vehicule** comportant :

- les **attributs de classe, les constantes de classe**
- les **attributs d'instance**
- un **constructeur** ayant pour paramètre la consommation
- des **méthodes d'accès en lecture** aux attributs d'instance, selon le degré d'accessibilité prescrit ci-avant
- les méthodes **mettreDeLessence** en veillant à ne pas dépasser la capacité du réservoir (penser aux exceptions) et **faireLePlein**
- une méthode **rouler**, qui prend en paramètre la distance que l'on souhaite parcourir et qui retourne la distance effectivement parcourue (les distances sont spécifiées par un nombre réel)
- la méthode **toString**, qui retourne une représentation textuelle d'un véhicule et de son état, conforme à l'exemple suivant :
  Véhicule 45 : compteur = [totalisateur = 500 | partiel = 108] ; jauge = 9.00533
- la méthode de comparaison c**ompareTo**, qui compare ce véhicule avec le véhicule spécifié au regard des *numéros d'immatriculation*

Tests: activer les tests sur les véhicules dans le programme de tests fourni

```
######## TESTS DES VEHICULES ########
Vehicule 0 : compteur = [Totalisateur = 0 | Partiel = 0];; jauge = 0.0

Le vehicule 0 a parcouru 0.0
Vehicule 0 : compteur = [Totalisateur = 0 | Partiel = 0];; jauge = 50.0

Le vehicule 0 a parcouru 300.0
Vehicule 0 : compteur = [Totalisateur = 300 | Partiel = 300];; jauge = 34.1

Le vehicule 0 a parcouru 643.39
Vehicule 0 : compteur = [Totalisateur = 943 | Partiel = 300];; jauge = 50.0

Le vehicule 0 a parcouru 200.0
Vehicule 0 : compteur = [Totalisateur = 1143 | Partiel = 143];; jauge = 39.4

Vehicule 0 : compteur = [Totalisateur = 1683 | Partiel = 683];; jauge = 10.77

Vehicule 0 : compteur = [Totalisateur = 1683 | Partiel = 683];; jauge = 50.0

Vehicule 0 : compteur = [Totalisateur = 1943 | Partiel = 943];; jauge = 36.22

Vehicule 0 : compteur = [Totalisateur = 1943 | Partiel = 943];; jauge = 42.22

Votre réservoir a une capacité insuffisante pour mettre 16 d'essence
```

```
Vehicule 0 : compteur = [Totalisateur = 1943 | Partiel = 943];; jauge = 42.22

Vehicule 1 : compteur = [Totalisateur = 0 | Partiel = 0];; jauge = 0.0

0
-1
```

III Modélisation d'un garage

En utilisant une classe d'implémentation de l'**interface Collection**, créer une classe **Garage** permettant comportant un attribut d'instance permettant de stocker les véhicules, une méthode d'ajout d'un véhicule dans le garage et la méthode toString.

Le parcours des véhicules du garage doit être rendu indépendant de la classe d'implémentation choisie à l'aide de l'interface **Iterable**.

A l'aide des interfaces génériques Comparable<T> et Comparator<T> et en utilisant des **classes externes pour les Comparators**, écrire le code nécessaire afin de pouvoir trier les véhicules selon leur *numéro d'immatriculation,* puis selon l'enregistrement *totalisateur* de leur *compteur kilométrique.*

1) Classe d'implémentation: **ArrayList** de l'interface **List**
2) Classe d'implémentation **TreeSet** de l'interface **Set**

Indication: les classes Comparators figurent comme **classes externes à la classe Vehicule.** Elles figurent dans le fichier Vehicule.java.

Tests: activer les tests sur les garages dans le programme de tests fourni

```
######## TESTS DU GARAGE Collection: List ########
[Vehicule 2 : compteur = [Totalisateur = 0 | Partiel = 0];; jauge = 0.0
, Vehicule 3 : compteur = [Totalisateur = 0 | Partiel = 0];; jauge = 0.0
, Vehicule 4 : compteur = [Totalisateur = 0 | Partiel = 0];; jauge = 0.0
, Vehicule 5 : compteur = [Totalisateur = 0 | Partiel = 0];; jauge = 0.0
, Vehicule 6 : compteur = [Totalisateur = 0 | Partiel = 0];; jauge = 0.0
]
[Vehicule 2 : compteur = [Totalisateur = 481 | Partiel = 481];; jauge = 22.57
, Vehicule 3 : compteur = [Totalisateur = 342 | Partiel = 342];; jauge = 28.77
, Vehicule 4 : compteur = [Totalisateur = 167 | Partiel = 167];; jauge = 35.79
, Vehicule 5 : compteur = [Totalisateur = 757 | Partiel = 757];; jauge = 5.33
, Vehicule 6 : compteur = [Totalisateur = 57 | Partiel = 57];; jauge = 47.43
]
Votre réservoir a une capacité insuffisante pour mettre 79 d'essence
Votre réservoir a une capacité insuffisante pour mettre 61 d'essence
Votre réservoir a une capacité insuffisante pour mettre 93 d'essence
Votre réservoir a une capacité insuffisante pour mettre 55 d'essence
[Vehicule 2 : compteur = [Totalisateur = 481 | Partiel = 481];; jauge = 22.57
```

```
, Vehicule 3 : compteur = [Totalisateur = 342 | Partiel = 342];; jauge = 28.77
, Vehicule 4 : compteur = [Totalisateur = 167 | Partiel = 167];; jauge = 35.79
, Vehicule 5 : compteur = [Totalisateur = 757 | Partiel = 757];; jauge = 10.33
, Vehicule 6 : compteur = [Totalisateur = 57 | Partiel = 57];; jauge = 47.43
]

######## TESTS DU GARAGE Collection: List ########
[Vehicule 2 : compteur = [Totalisateur = 0 | Partiel = 0];; jauge = 0.0
, Vehicule 3 : compteur = [Totalisateur = 0 | Partiel = 0];; jauge = 0.0
, Vehicule 4 : compteur = [Totalisateur = 0 | Partiel = 0];; jauge = 0.0
, Vehicule 5 : compteur = [Totalisateur = 0 | Partiel = 0];; jauge = 0.0
, Vehicule 6 : compteur = [Totalisateur = 0 | Partiel = 0];; jauge = 0.0
]
[Vehicule 2 : compteur = [Totalisateur = 57 | Partiel = 57];; jauge = 46.77
, Vehicule 3 : compteur = [Totalisateur = 806 | Partiel = 806];; jauge = 50.0
, Vehicule 4 : compteur = [Totalisateur = 487 | Partiel = 487];; jauge = 8.58
, Vehicule 5 : compteur = [Totalisateur = 229 | Partiel = 229];; jauge = 36.48
, Vehicule 6 : compteur = [Totalisateur = 784 | Partiel = 784];; jauge = 14.73
]
Votre réservoir a une capacité insuffisante pour mettre 32 d'essence
Votre réservoir a une capacité insuffisante pour mettre 15 d'essence
Votre réservoir a une capacité insuffisante pour mettre 83 d'essence
Votre réservoir a une capacité insuffisante pour mettre 72 d'essence
[Vehicule 2 : compteur = [Totalisateur = 57 | Partiel = 57];; jauge = 46.77
, Vehicule 3 : compteur = [Totalisateur = 806 | Partiel = 806];; jauge = 50.0
, Vehicule 4 : compteur = [Totalisateur = 487 | Partiel = 487];; jauge = 8.58
, Vehicule 5 : compteur = [Totalisateur = 229 | Partiel = 229];; jauge = 36.48
, Vehicule 6 : compteur = [Totalisateur = 784 | Partiel = 784];; jauge = 42.73
]


##Tri selon le compteur km totalisateur ##
[Vehicule 2 : compteur = [Totalisateur = 57 | Partiel = 57];; jauge = 46.77
, Vehicule 5 : compteur = [Totalisateur = 229 | Partiel = 229];; jauge = 36.48
, Vehicule 4 : compteur = [Totalisateur = 487 | Partiel = 487];; jauge = 8.58
, Vehicule 6 : compteur = [Totalisateur = 784 | Partiel = 784];; jauge = 42.73
, Vehicule 3 : compteur = [Totalisateur = 806 | Partiel = 806];; jauge = 50.0
]


##Tri selon le no immatriculation ##
[Vehicule 2 : compteur = [Totalisateur = 57 | Partiel = 57];; jauge = 46.77
, Vehicule 3 : compteur = [Totalisateur = 806 | Partiel = 806];; jauge = 50.0
, Vehicule 4 : compteur = [Totalisateur = 487 | Partiel = 487];; jauge = 8.58
, Vehicule 5 : compteur = [Totalisateur = 229 | Partiel = 229];; jauge = 36.48
, Vehicule 6 : compteur = [Totalisateur = 784 | Partiel = 784];; jauge = 42.73
]


######## TESTS DU GARAGE Collection: Set ########
[Vehicule 7 : compteur = [Totalisateur = 0 | Partiel = 0];; jauge = 0.0
, Vehicule 8 : compteur = [Totalisateur = 0 | Partiel = 0];; jauge = 0.0
, Vehicule 9 : compteur = [Totalisateur = 0 | Partiel = 0];; jauge = 0.0
, Vehicule 10 : compteur = [Totalisateur = 0 | Partiel = 0];; jauge = 0.0
, Vehicule 11 : compteur = [Totalisateur = 0 | Partiel = 0];; jauge = 0.0
]
[Vehicule 7 : compteur = [Totalisateur = 877 | Partiel = 877];; jauge = 50.0
```

```
, Vehicule 8 : compteur = [Totalisateur = 654 | Partiel = 654];; jauge = 9.46
, Vehicule 9 : compteur = [Totalisateur = 281 | Partiel = 281];; jauge = 26.14
, Vehicule 10 : compteur = [Totalisateur = 817 | Partiel = 817];; jauge = 1.78
, Vehicule 11 : compteur = [Totalisateur = 628 | Partiel = 628];; jauge = 21.72
]

Votre réservoir a une capacité insuffisante pour mettre 85 d'essence
Votre réservoir a une capacité insuffisante pour mettre 87 d'essence
Votre réservoir a une capacité insuffisante pour mettre 61 d'essence
Votre réservoir a une capacité insuffisante pour mettre 54 d'essence
[Vehicule 7 : compteur = [Totalisateur = 877 | Partiel = 877];; jauge = 50.0
, Vehicule 8 : compteur = [Totalisateur = 654 | Partiel = 654];; jauge = 9.46
, Vehicule 9 : compteur = [Totalisateur = 281 | Partiel = 281];; jauge = 26.14
, Vehicule 10 : compteur = [Totalisateur = 817 | Partiel = 817];; jauge = 1.78
, Vehicule 11 : compteur = [Totalisateur = 628 | Partiel = 628];; jauge = 44.72
]


##Tri selon le compteur km totalisateur ##
[Vehicule 9 : compteur = [Totalisateur = 281 | Partiel = 281];; jauge = 26.14
, Vehicule 11 : compteur = [Totalisateur = 628 | Partiel = 628];; jauge = 44.72
, Vehicule 8 : compteur = [Totalisateur = 654 | Partiel = 654];; jauge = 9.46
, Vehicule 10 : compteur = [Totalisateur = 817 | Partiel = 817];; jauge = 1.78
, Vehicule 7 : compteur = [Totalisateur = 877 | Partiel = 877];; jauge = 50.0
]


##Tri selon le no immatriculation ##
[Vehicule 7 : compteur = [Totalisateur = 877 | Partiel = 877];; jauge = 50.0
, Vehicule 8 : compteur = [Totalisateur = 654 | Partiel = 654];; jauge = 9.46
, Vehicule 9 : compteur = [Totalisateur = 281 | Partiel = 281];; jauge = 26.14
, Vehicule 10 : compteur = [Totalisateur = 817 | Partiel = 817];; jauge = 1.78
, Vehicule 11 : compteur = [Totalisateur = 628 | Partiel = 628];; jauge = 44.72
]
```

IV Compléments: les comparateurs en Classes imbriquées anonymes à l'aide de constantes d'interface

Recopier et renommer le projet des 3 premières questions.

1) Rappel du cours sur les classes statiques imbriquées anonymes

(archive des sources disponibles sur Campus)

Les classes imbriquées et les classes imbriquées statiques sont lourdes à utiliser (voir la partie optionnelle VI), d'où le recours à un sucre syntaxique via des méthodes.

De plus elles nécessitent leur instanciation à chaque appel à la méthode Collections.sort.

Une autre approche bien plus agréable est celle des classes imbriquées anonymes définies à l'intérieur d'une interface de constantes.

Le sucre syntaxique est à forte dose et conduit à un confort de programmation très appréciable.

Interface de constantes

```
// OBSERVER BIEN LA CLASSE ANONYME IMBRIQUEE
// static final ==> constante ==> une et une seule instanciation de ces comparateurs
```

```
package zoo;

import java.util.Comparator;

public interface AnimalComparator {
	static final Comparator<Animal> POIDS_ORDER = new Comparator<Animal>() {
		@Override
		public int compare(Animal o1, Animal o2) {
			int poids1=o1.getPoids();
			int poids2=o2.getPoids();

			if (poids1>poids2)
				return 1;
			else if (poids1<poids2)
				return -1;
			else
				return 0;
```

```
		}
	};

	static final Comparator<Animal> PLAT_ORDER = new Comparator<Animal>() {
		@Override
		public int compare (Animal o1, Animal o2){
			Plat plat1=o1.getPlatPref();
			Plat plat2=o2.getPlatPref();
			return plat1.compareTo(plat2);
		}
	};
}
```

Dans classe Zoo, les 2 méthodes de tri sont définies:

SUCRE SYNTAXIQUE
Tri selon l'ordre naturel des noms, ordre lexicographique, défini par la méthode compareTo de l'interface Comparable

```
/
public void sortNom() {
	Collections.sort(animals);
}
```

Approche utilisant un comparateur statique unique

SUCRE SYNTAXIQUE
Tri selon l'ordre des poids en utilisant le comparateur statique POIDS_ORDER

```
public void sortPoids() {
	Collections.sort(animals, AnimalComparator.POIDS_ORDER);
}
```

SUCRE SYNTAXIQUE
Tri selon l'ordre des plats en utilisant le comparateur statique PLAT_ORDER

```
public void sortPlat() {
	Collections.sort(animals, AnimalComparator.PLAT_ORDER);;
}
```

Sans sucre syntaxique, le code utilisateur s'écrit assez lourdement:

appel à getAnimals ==> l'implementation remonte à la surface...
appel aux constantes d'interface AnimalComparator.POIDS_ORDER et AnimalComparator.PLAT_ORDER

```
	System.out.println("\nTRI PAR NOM");
	Collections.sort(vincennes.getAnimals());
	vincennes.afficher();
	System.out.println("\nTRI PAR POIDS");
	Collections.sort(vincennes.getAnimals(), AnimalComparator.POIDS_ORDER);
```

```
vincennes.afficher();
System.out.println("\nTRI PAR PLAT PREFERE");
Collections.sort(vincennes.getAnimals(), AnimalComparator.PLAT_ORDER);
vincennes.afficher();
```

Avec le sucre syntaxique, le code utilisateur s'écrit très légèrement:

```
System.out.println("\nTRI PAR NOM");
vincennes.sortNom();
vincennes.afficher();
System.out.println("\nTRI PAR POIDS");
vincennes.sortPoids();
vincennes.afficher();
System.out.println("\nTRI PAR PLAT PREFERE");
vincennes.sortPlat();
vincennes.afficher();
```

2) Mettre en oeuvre des comparateurs statiques imbriqués anonymes via une interface de constantes pour comparer les véhicules d'un garage selon l'enregistrement totalisateur de leur compteur kilométrique et selon le niveau de leur jauge.
Ne pas oublier la comparaison selon l'odre naturel, celui des numéros d'immatriculation.

V Application d'un traitement à tous les véhicules d'un garage

1) Rappel du cours sur les classes de mapping

(archive des sources disponibles sur Campus)

```
public interface Fonction<T> {
	abstract public void applyIt (T t);

}

public interface Map<T> {
	abstract public void map (Fonction <T> f);
}


// la classe pour la fonction de vieillissement de tous les animaux
class Vieillissement implements Fonction<Animal>{

	@Override
	public void applyIt(Animal animal) {
		animal.setAge(animal.age + (Animal.LUCIFER-animal.age)) ;
	}
}



//la classe pour la fonction de vieillissement des chats
class Rajeunissement implements Fonction<Animal>{

	@Override
	public void applyIt(Animal animal) {
		if (animal.getClass() != Chat.class) {
			return;
		}
		Chat chat = (Chat)animal;
		if (chat.age < 20)
			chat.setAge(Chat.BEBECHAT);
		else if (chat.age > 20 && chat.age < 30)
			chat.setAge(chat.age - Chat.RAJEUNISSEMENT);
		else
			chat.setAge(Chat.ADOLESCENT);
	}
}
```

2) Mettre en oeuvre des mappings sur tous les véhicules d'un garage afin de remettre tous les compteurs partiels à 0 et afin de faire le plein d'essence si le réservoir contient moins de 10 litres

VI PARTIE OPTIONNELLE

Compléments: les comparateurs en Classes statiques imbriquées

Recopier et renommer le projet des 3 premières questions.

On ne s'intéressera pas ici aux classes imbriquées non statiques dont la mise en oeuvre est complexe.

1) Rappel du cours sur les classes statiques imbriquées

(archive des sources disponibles sur Campus)

```
/**
* UNE PREMIERE APPROCHE est d'imbriquer les comparateurs dans la classe Animal
* La ligne suivante ne peut compiler car il faut appliquer new Comparator() sur
* une instance d'Animal!
* Collections.sort(vincennes.getAnimals(), new PoidsComparator());
*
* Il faudrait donc pouvoir écrire:
* Animal x = new Animal();
* Collections.sort(vincennes.getAnimals(),x.new PoidsComparator());
* Impossible car la classe Animal est abstraite
*
*/


/**
* CLASSE IMBRIQUEE STATIC ==> PAS NECESSAIRE D'INSTANCIER LA CLASSE VEHICULE
*
* classe imbriquée static, elle peut être utilisée sans avoir besoin
* d'instancier la classe qui la contient
* ainsi on peut avoir un PoidsComparator sans instancier pour autant un Animal
*/
static class PoidsComparator implements Comparator <Animal>{

	/**
	* un accesseur statique afin de récupérer une instance de PoidsComparator
	* en dehors de la classe
	* utilisation: Collections.sort(vincennes.getAnimals(),
	*                        Animal.PoidsComparator.getPoidsComparator());
	*/
	public static PoidsComparator getPoidsComparator() {
		return new PoidsComparator();
	}

	@Override
	public int compare(Animal o1, Animal o2) {
		int poids1=o1.getPoids();
		int poids2=o2.getPoids();
```

```
            if (poids1>poids2)
                  return 1;
            else if (poids1<poids2)
                  return -1;
            else
                  return 0;
      }
}

/**
* classe imbriquée static, elle peut être utilisée sans avoir besoin d'instancier
* la classe qui la contient
* ainsi on peut avoir un PlatComparator sans instancier pour autant un animal
*/
static class PlatComparator implements Comparator <Animal>{

      /**
      * un accesseur statique afin de récupérer une instance de PlatComparator
      * en dehors de la classe
      * utilisation: Collections.sort(vincennes.getAnimals(),
      *                         Animal.PlatComparator.getPlatComparator());
      */
      public static PlatComparator getPlatComparator() {
            return new PlatComparator();
      }

      @Override
      public int compare (Animal o1, Animal o2){
            Plat plat1=o1.getPlatPref();
            Plat plat2=o2.getPlatPref();
            return plat1.compareTo(plat2);
      }
}
```

Dans la classe Zoo:

```
/**
 * SUCRE SYNTAXIQUE
 * Tri selon l'ordre naturel des noms, ordre lexicographique, défini par la
 * méthode compareTo de l'interface Comparable
 */
public void sortNom() {
      Collections.sort(animals);
}

/**
 * SUCRE SYNTAXIQUE
 * Tri selon l'ordre des poids, défini par la méthode compareTo de la classe
 * Animal.PoidsComparator
 * Il faut pouvoir utiliser une instance de la classe statique imbriquée
 * Animal.PoidsComparator
 * C'est possible grâce à l'accesseur getPoidsComparator()
 */
public void sortPoids() {
      Collections.sort(animals,
                        Animal.PoidsComparator.getPoidsComparator());
}

/**
 * SUCRE SYNTAXIQUE
 * Tri selon l'ordre des plats, défini par la méthode compareTo de la classe
 * Animal.PlatComparator
 * Il faut pouvoir utiliser une instance de la classe statique imbriquée
 * Animal.PlatComparator
 * C'est possible grâce à l'accesseur getPlatComparator()
 */
public void sortPlat() {
      Collections.sort(animals,
                         Animal.PlatComparator.getPlatComparator());
}
```

Utilisation des classes statiques imbriquées :

```
// sans le sucre syntaxique pour les tris

// ==> écriture lourde pour le code client
// appel à getAnimals ==> l'implementation remonte à la surface...
// appel Animall.PoidsComparator.getPoidsComparator()
// ==> la méthode statique de la classe imbriquée statique !!!!
// appel Animal.PlatComparator.getPlatComparator()
// ==> la méthode statique de la classe imbriquée statique !!!!
System.out.println("\nTRI PAR NOM");
Collections.sort(vincennes.getAnimals());
vincennes.afficher();
System.out.println("\nTRI PAR POIDS");
Collections.sort(vincennes.getAnimals(),
                         Animal.PoidsComparator.getPoidsComparator());
vincennes.afficher();
System.out.println("\nTRI PAR PLAT PREFERE");
Collections.sort(vincennes.getAnimals(),
                         Animal.PlatComparator.getPlatComparator());
vincennes.afficher();


// le sucre syntaxique pour les tris
// ==> code léger pour le code client
System.out.println("\nTRI PAR NOM");
vincennes.sortNom();
vincennes.afficher();
System.out.println("\nTRI PAR POIDS");
vincennes.sortPoids();
vincennes.afficher();
System.out.println("\nTRI PAR PLAT PREFERE");
vincennes.sortPlat();
vincennes.afficher();
```

2) Mettre en oeuvre des comparateurs statiques imbriqués pour comparer les véhicules d'un garage selon les mêmes critères que précédemment.