Exercice 1
Effectuer les modifications dans le précédent TP en rajoutant un constructeur qui initialise tous les champs des différentes classes `Vehicule`, `Voiture`, `Camion` et `Autocar (si ce n'est pas déjà fait)`.

Exercice 2
1. Créer les classes `Agence`, `Bureau` et `Garage` permettant la représentation des bureaux de location et garages **ainsi que les méthodes demandées en TD**. On utilisera un tableau de références sur des garages pour représenter l'ensemble des garages connus par un bureau de location. Un bureau pouvant connaître 4 garages ou moins, nous utiliserons un tableau de 4 références sur des garages en utilisant la valeur de référence particulière `null` pour marquer les « cases vides » du tableau.
2. Écrire une (des) méthode(s) `afficherXXX()` produisant des affichages tels que :
```
Bureau de location – Angers – 0241123456 – 4 adm – 5 comm
Garage – Angers sud - 0241234567 – 1 adm – 6 meca
```

Exercice 3
1. Ajouter aux classes représentant les véhicules et les agences des méthodes `toString()` retournant l'équivalent de ce qui est affiché par `afficherXXX()` pour les agences et renvoyant la même chose que les méthodes `infoXXX()` pour les véhicules.
2. Modifier les méthodes `afficher()` pour utiliser `toString()`, et vérifier qu'il est possible d'utiliser des objets véhicule ou agence comme paramètre de `System.out.println()`.
3. Écrire un programme de test qui crée un bureau de location et trois garages, ajoute ces garages au bureau de location, et affiche les différents objets créés par un `System.out.println()`.
4. Écrire une méthode `categorie()` qui retourne un entier représentant la catégorie de l'agence. Les agences de catégorie 1 sont celles employant moins de 10 personnes, les agences de catégorie 2 sont celles en employant 10 à 19, et les agences de catégorie 3 sont celles en employant 20 ou plus. Cette méthode ne devra être définie que dans une seule classe.
5. Écrire une méthode `compareTo()` prenant une agence en paramètre qui compare la taille de cette agence avec l'agence courante : cette méthode devra retourner (une référence sur) celle qui emploie le plus de personnel.
6. Écrire une méthode `afficherGarages()` (dans la classe représentant les bureaux) prenant comme paramètre un entier `cat` et affichant les garages, liés à ce bureau, dont la catégorie est supérieure ou égale à `cat`. L'affichage des caractéristiques des garages devra être précédé d'une ligne décrivant ce qui est affiché, comme sur l'exemple suivant :
```
Garages de catégorie >= 2 du Bureau de location - Angers - 0241123456 - 4 adm - 5 comm
Garage - Angers est - 0241456789 - 5 adm - 10 meca
```

Exercice 4
Écrire la classe représentant l'`Entreprise` de location dans son entier comme elle est proposée dans l'exercice 2 du TD3. La classe `Ensemble` sera ici aussi représentée par un tableau de références.

Constructeurs et Appel du constructeur de la super-classe

Rappel : lorsqu'on définit une classe C on peut définir des constructeurs avec ou sans paramètres pour initialiser les objets créés.

Tout constructeur d'une classe autre que la classe Object (voir plus loin) fait appel soit à un constructeur de la super-classe, soit à un constructeur de la même classe. Cet appel doit être la première instruction dans la définition du constructeur.

On utilise la syntaxe suivante: `super(arguments)` (où arguments peut être vide ou non) pour appeler un constructeur de la superclasse, et `this(arguments)` pour appeler un constructeur de la même classe.

Si aucun constructeur de la super-classe ou de la même classe n'est appelé explicitement, le compilateur génère implicitement un appel au constructeur par défaut `super()` qui est sans paramètre (il faut qu'il y ait un tel constructeur dans la super classe sinon une erreur est détectée).

```
class Sc extends C
{
 public Sc(int a, String s){
   super(a); // On suppose que C a un constructeur à un paramètre
entier
                                          // initialisation propre à
Sc
                                       }
}
```

Appel de la super méthode

Quand une méthode est redéfinie dans une sous-classe, il est souvent nécessaire d'appeler la méthode masquée (méthode de même signature de la super-classe). Cet appel peut être fait n'importe où dans le code de le méthode de la sous-classe. L'appel se fait en utilisant la pseudo-variable `super` qui, comme `this` repère l'objet sur lequel la méthode est en train d'être exécutée, mais qui est du type de la super-classe.

```
class Sc extends C{
 public methode(int a) // redéfinition d'une méthode définie dans C
 {
  // traitements
  super.methode(a);
  // traitements
 }
}
```

Tableaux

En Java, on ne manipule pas directement des tableaux (comme en C ou en Pascal), mais des références sur des tableaux. Ainsi la déclaration `int[] tab;` (la syntaxe `int tab[]; est aussi acceptée`) déclare une référence sur un tableau d'entiers. Pour pouvoir utiliser un tableau, il faut donc le créer (de la même façon qu'il faut instancier une classe pour utiliser un objet, et non pas simplement déclarer une référence sur un objet) en précisant sa taille : `tab = new int[20];`. Ceci crée un tableau de 20 entiers, ces entiers étant accessibles par `tab[0]` .. `tab[19]`. Contrairement au langage C, il est possible de connaître la taille d'un tableau, en accédant à l'attribut `length` d'un tableau (`System.out.println(tab.length);` affiche 20). Attention : cet attribut est particulier et n'est accessible qu'en lecture (`tab.length = 25;` est interdit).

L'exemple ci-dessous déclare une référence sur tableau d'entiers, qui repère un tableau de 20 entiers, ce tableau contenant les valeurs 0 à 19.

```
int []tab = new int[20];
for (int i=0; i<tab.length; i++)

                                        tab[i] = i;
```

Référence null

`null` est une valeur de référence particulière (non typée, pouvant donc être utilisée avec tout type de référence) qui ne repère aucun objet. Quand une référence est créée, elle est implicitement initialisée à `null`. Il est évidemment interdit d'accéder à des attributs ou d'appeler des méthodes à partir d'une référence `null`.

La classe Object : racine de l'arbre d'héritage

Si une classe est déclarée comme n'ayant aucune super-classe, elle admet implicitement `Object` (classe abstraite) comme super-classe. La classe Object fait partie du paquetage `java.lang`, qui contient les classes et les interfaces les plus centrales du langage.

Méthode toString()

Dans `Object`, la méthode `public String toString()` est déclarée; elle retourne une chaîne de caractères représentant l'objet : sa classe et son adresse. Lorsqu'un objet quelconque est passé comme paramètre à `System.out.println()`, c'est le résultat de l'appel à `toString()` sur cet objet qui est affiché. Afin de pouvoir afficher simplement l'état des objets du programme, on redéfinit souvent la méthode `toString()` dans les classes que l'on crée. Beaucoup de classes de l'API Java redéfinissent aussi la méthode `toString()`.