HashMap et autres

Pour les exercices de cette semaine, vous allez utiliser les classes de l'API JAVA.
Nous vous conseillons d'utiliser uniquement les méthodes reprises dans le document API_JAVA !

Exercices obligatoires

A Application Parking

A1 Le personnel de la société X a le droit de placer sa voiture dans le parking de la société. Pour éviter les indésirables, l'entrée du parking est munie d'une barrière.
La société décide d'équiper la barrière d'un détecteur de plaques. Celle-ci ne s'ouvrira que pour les voitures autorisées.

La classe *EnsembleVoituresAutorisees* que vous avez écrite la semaine dernière possède les méthodes :

```
boolean retirerVoiture(Voiture voiture)
boolean ajouterVoiture(Voiture voiture)
boolean voitureAcceptee(Voiture voiture)
```

Le système mis en place détecte uniquement des numéros de plaque.
Des objets de type « Voiture » doivent être crées avec des propriétaires bidons pour pouvoir utiliser les méthodes de retrait et de vérification.
Cette classe devrait plutôt posséder les méthodes suivantes :

```
boolean retirerVoiture(String plaque)
boolean ajouterVoiture(String plaque, Proprietaire proprietaire)
boolean voitureAcceptee(String plaque)
```

On peut même ajouter la méthode :

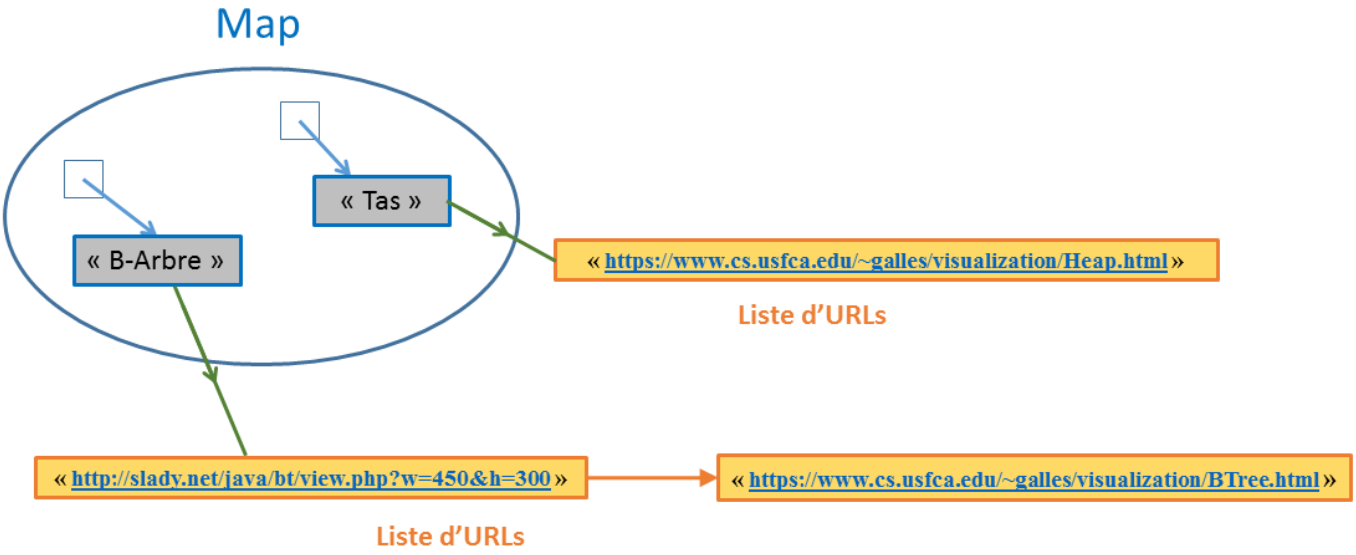
```
Proprietaire donnerProprietaire(String plaque)
```

L'utilisation de ces méthodes est plus naturelle.

Complétez la classe *EnsembleVoituresAutorisees*.
Comme structure de données, utilisez un objet de la classe *HashMap*.
Testez la classe avec la classe *TestEnsembleVoituresAutorisees*.

B DicoSD

Pour le cours de SD, on aimerait bien disposer d'un dico qui associerait des URLs vers des sites web intéressants à chaque structure de données rencontrée.



Dans cet exemple, le site web conseillé pour le Tas est :

https://www.cs.usfca.edu/~galles/visualization/Heap.html

Ceux conseillés pour le B-Arbre sont :

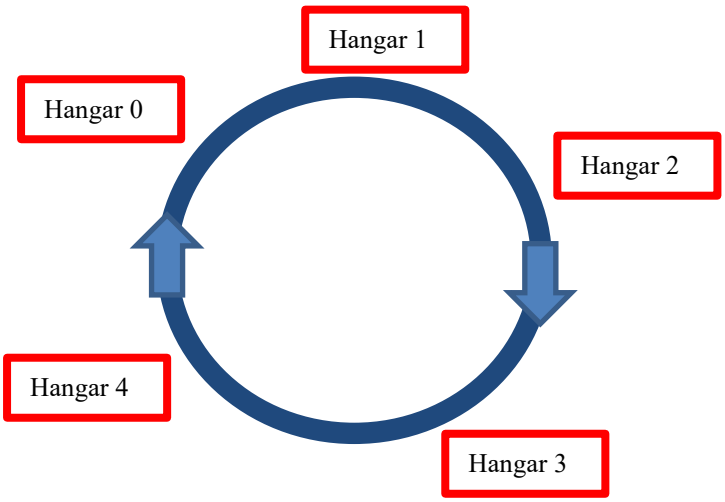
http://slady.net/java/bt/view.php?w=450&h=300
https://www.cs.usfca.edu/~galles/visualization/BTree.html

L'utilisation d'un « map » semble évidente.
A chaque structure de données (*String*) lui sera associée une liste d'URLs de sites internet (*String*).
Vous utiliserez les classes *HashMap* et *LinkedList* de Java.
Complétez la classe *DicoSD* et testez-la avec la classe *TestDicoSD*.

C Application Entrepôt

L'exercice proposé se veut évolutif.
Essayez d'aller le plus loin possible dans les améliorations proposées.
Pour les plus téméraires, il y a même un défi 💣

C1 Un entrepôt met à disposition plusieurs hangars. Ceux-ci sont disposés le long d'une route circulaire à sens unique.
Les hangars sont proposés à des sociétés qui peuvent en réserver un ou plusieurs.



L'application que vous allez programmer gère cet entrepôt.

Elle est composée de plusieurs classes :

La classe *Hangar*
Chaque hangar possède un numéro et la société qui l'occupe.
La numérotation des hangars commence à 0.
Si le hangar n'est pas occupé, la société est à `null`.

La classe *Societe*
Il s'agit de conserver les informations utiles de la société au sein de cette application.
Chaque société possède un numéro et un nom. Le numéro est un identifiant unique.
Elle va retenir d'autres informations, comme, l'ensemble des hangars qu'elle occupe.
Cette classe doit être en partie complétée.

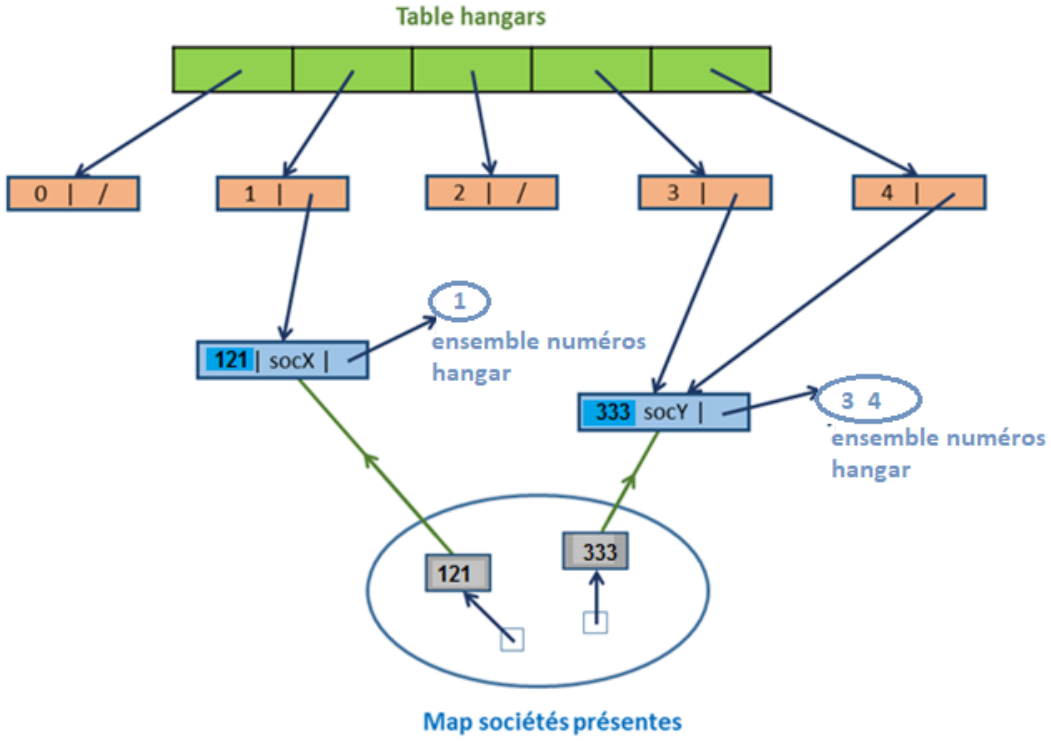
La classe Entrepôt

Tous les hangars sont sauvegardés dans une table. Cette table contient des objets de la classe *Hangar*. L'indice correspond au numéro du hangar.

Un dictionnaire non trié (*map*) va permettre de retrouver directement les informations liées à une société. La clé est le numéro de société et la valeur, la société correspondante.
Ce *map* contient toutes les sociétés présentes.

Pensez à mettre un attribut `nombreHangarsLibres`. Cet attribut permet de savoir directement si une attribution peut être faite.

Voici un schéma pour vous aider à mieux visualiser un objet de la classe *Entrepot* :
Dans cet exemple, l'entrepôt contient 5 hangars. 3 de ces 5 hangars sont occupés par 2 sociétés.



Le constructeur reçoit en paramètre le nombre d'hangars. Cette information permet de créer les différents hangars et de les placer dans la table.

La méthode `Societe getSociete(int numeroSociete)` renvoie la société dont le numéro est passé en paramètre ou null si aucune société présente porte ce numéro.

La méthode `int attribuerHangar(int numeroSociete, String nomSociete)` attribue un hangar à la société passée en paramètre
Si l'attribution a pu se faire, la société est enregistrée comme présente (si pas encore présente) et le hangar lui est ajouté.

Pour, entre autres, permettre une **bonne répartition** des hangars occupés, voici comment se fait l'attribution d'un hangar :
C'est le numéro de la société qui va déterminer le numéro du hangar à attribuer.
La société 0 → 0, la société 1 → 1, … , la société 5 → 0 (si 5 hangars), la société 6 → 1, ….
Si le hangar est occupé, c'est le hangar le plus proche après celui-ci, qui sera attribué. Les hangars sont disposés de façon circulaire ! Après le dernier hangar, on retrouve le hangar 0.
Cette classe doit être en partie complétée.

La classe *TestAttributionHangar.*

Cette classe suit un scénario de tests. Elle utilise les différentes méthodes à compléter de la classe *Societe* et de la classe *Entrepot*. Elle permet de détecter pas mal de bugs.
Les 2 classes doivent être entièrement complétées pour permettre le bon fonctionnement de cette classe.
Utilisez cette classe avant de vous lancer la classe *GestionEntrepot*.

La classe *GestionEntrepot*

Elle propose le menu suivant :

```
1 : attribuer un hangar
2 : lister les hangars d'une société
```

C'est dans cette classe (et uniquement dans celle-ci) que se font les interactions avec un utilisateur.

Cette classe utilise la classe *MonScanner*.
Prenez connaissance de cette classe en lisant le document word qui porte son nom.

Cette classe doit être en partie complétée.

C2 Le menu propose (en plus) :

```
3 : libérer un hangar
```

Pensez à ajouter la méthode `boolean libererHangar(int numeroHangar)` dans la classe *Entrepot*.
(N'oubliez pas la *JavaDoc*)

C3 Le gestionnaire de l'entrepôt décide d'équiper l'entrée d'une barrière automatique.
Seuls les véhicules enregistrés auront le droit d'entrer.
Une société qui possède au moins un hangar devra donner les numéros des plaques d'immatriculation de ses véhicules.

Le menu propose (en plus) :

```
4 : ajouter un véhicule à une société
5 : vérifier une plaque d'immatriculation
```

On voudrait que la méthode `boolean estAutorisee(String numPlaque)` que vous allez ajouter dans la classe *Entrepot*, soit en **O(1)**.

Le choix de(s) structure(s) de données que vous allez ajouter est très important !
Commencez par compléter le schéma ci-dessus.
La société X possède les véhicules 1XXX000 et 1XXX001
La société Y possède les véhicules 1YYY000, 1YYY001 et 1YYY002

DISCUTEZ de votre choix avec un professeur, avant de le programmer.

L'ajout de ces options va demander l'ajout de méthodes, mais également de revoir certaines que vous avez écrites.
Une société qui n'a plus d'hangar, ne peut plus rentrer dans l'entrepôt …

Exercice supplémentaire

C4 Le menu propose (en vue de travaux) :

```
6 : mettre momentanément un hangar (non utilisé) hors service/en service
```

Exercice défi ❤️

C5 Amélioration du point 2 du menu :

Exécutez la classe *GestionEntrepot* en prenant le fichier de commande *commandesDefi*.

```
private static MonScanner scanner = new MonScanner( file: "commandesDefi.txt");
```

Remarquez que les numéros des hangars n'apparaissent pas toujours par ordre croissant :

```
Entrez le numero de la societe : 3
Voici les hangars de cette societe : [16, 3]
Entrez le numero de la societe : 4
Voici les hangars de cette societe : [4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 17]
```

On voudrait que ces numéros apparaissent toujours par ordre croissant.
Ne programmez pas vous-même une méthode de tri ! Prenez quelque chose qui existe.