Application utilisant les classes de l'API Java

L'exercice proposé se veut évolutif.
Dans un premier temps, vous implémenterez une version préliminaire.
Il s'agit d'une ancienne question d'examen.
Vous soumettrez cette version préliminaire.
Un professeur testera cette version préliminaire en votre présence et vous fera un feed-back.
Vous passerez ensuite à la version de base et ensuite à ses améliorations.
Beaucoup d'améliorations demandent d'introduire une structure de données supplémentaire.
Pensez à compléter (modifier) le schéma de la version de base.
Essayez d'aller le plus loin possible dans les améliorations proposées.
L'idéal serait d'arriver à l'amélioration 3 comprise.
Vous soumettrez la version à laquelle vous êtes arrivé.

Lors de vos tests, pour éviter de devoir chaque fois encodé toute une séquence de commande identique, on vous conseille de les encoder dans un fichier. La classe *MonScanner* permet de passer de l'encodage via fichier à l'encodage manuel. Le document *CommandesAPartirDUnFichier* **vous explique comment procéder.**

Bon travail !

Avant de commencer cet exercice, veuillez relire le document *API_JAVA* avec les structures de données présentées à ce cours, ainsi que le diagramme des hiérarchies *Collection* et *Map*.
Contrainte : utilisez uniquement les méthodes reprises dans le document *API_JAVA* !

Application Brocante

VERSION PRELIMINAIRE

L'organisateur d'une brocante désire informatiser l'attribution des différents emplacements (qui ont tous la même taille).

L'attribution des emplacements se déroule en 2 phases.

Lors de la phase 1, seuls les riverains pourront faire des réservations.
Ils auront la possibilité de choisir les emplacements.
Cependant, ils ne pourront pas réserver plus de 3 emplacements.

Lors de la phase 2, n'importe qui peut demander des emplacements.
Il n'y a plus de limite en nombre.
Un riverain qui aurait déjà réservé 3 emplacements lors de la phase 1, peut donc encore en réserver d'autres lors de cette phase.
Mais lors de cette phase, les emplacements sont attribués de façon automatisée.
On ne peut plus les choisir.

L'organisateur aimerait bien garder une trace des attributions.

Voici l'implémentation qui a été choisie :

Pour la version de base, nous n'allons pas introduire de classe *Exposant* et de classe *Emplacement*.
L'exposant sera représenté par un *String*.
L'emplacement sera représenté par son numéro (*int*). La numérotation des emplacements commence à 0.

Le constructeur de la classe *Brocante* reçoit comme paramètres le nombre d'emplacements et une table (*String*) avec les riverains.

Ex :

« riv1 »	« riv2 »	« riv3 »

Les attributions seront enregistrées dans une table de *String*.
Chaque case correspond à un emplacement.

Au départ, cette table ne contiendra que des *null*.
On y placera, au fur et à mesure des réservations, les exposants.

Ex :

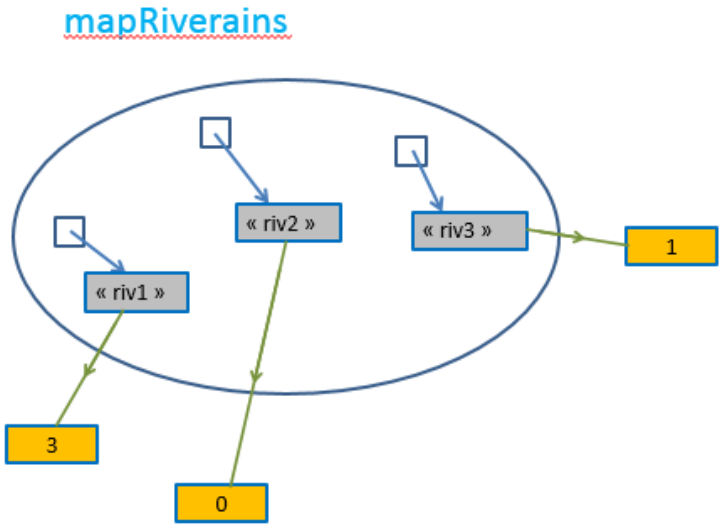
tableEmplacements

« riv1 »	« riv1 »	null	null	« riv3 »	null	« riv1 »	null

Les emplacements 0, 1 et 6 sont attribués à riv1.
L'emplacement 4 est attribué à riv3.
Les autres emplacements sont libres.

La phase est un attribut de la classe. Elle est initialisée à 1.

Lors de la phase 1, le nombre maximum de réservations est de 3 par riverain.
Il faut retenir, pour chacun de ceux-ci, son nombre de réservations.
C'est un *map* qui retiendra ces nombres :



Pour ce *map*, vous utiliserez un objet de la classe *HashMap<String,Integer>*.

Lors de la phase 2, les emplacements sont attribués de façon automatisée.
Pour ce faire, les numéros des emplacements libres seront placés dans une pile.

Dans l'exemple, pour débuter la phase 2, voici le contenu de la pile :

tableEmplacements

« riv1 »	« riv1 »	null	null	« riv3 »	null	« riv1 »	null

7
5
3
2

pileEmplacementsLibres

Pour cette pile, vous utiliserez un objet de la classe *ArrayDeque<Integer>*

Complétez la classe *Brocante* en respectant bien la *JavaDoc* et les choix d'implémentation imposés ci-dessus.
Utilisez uniquement les méthodes des classes *HashMap* et *ArrayDeque* reprises dans le document API_JAVA.

Complétez la classe *GestionBrocante* qui permet la gestion de la brocante.
Les menus proposés sont très simples mais vont s'étoffer au fur et à mesure des améliorations que vous allez apporter.

Conseil :

Pour faciliter le débogage, faites appel à la méthode `toString()`:
Dans la classe *GestionBrocante* : `System.out.println(brocante.toString()) ;`

Dans la classe *Brocante* : `System.out.println(toString()) ;`

Placez ces affichages « un peu partout ».
Vous les enlèverez quand toute l'application sera au point.

Dans un premier temps, faites bien apparaître toutes les structures de données dans la méthode `toString()` !
Celles-ci peuvent apparaître de façon simple :

```
aRenvoyer += Arrays.toString(tableReservations) ;
aRenvoyer += mapRiverains.toString() ;
etc…
```

Vous modifierez la méthode `toString()` pour faire apparaître un bel affichage de la brocante quand toute l'application sera au point.

AVANT DE PASSER A LA VERSION DE BASE, soumettez la classe Brocante via moodle.

VERSION DE BASE

Ecrivez une classe *Emplacement* et une classe *Exposant*.
Un emplacement est composé d'un numéro et d'un exposant. S'il n'y a pas d'exposant, l'exposant est mis à *null*.
Un exposant a comme attributs un nom, une adresse mail, un numéro de GSM, …

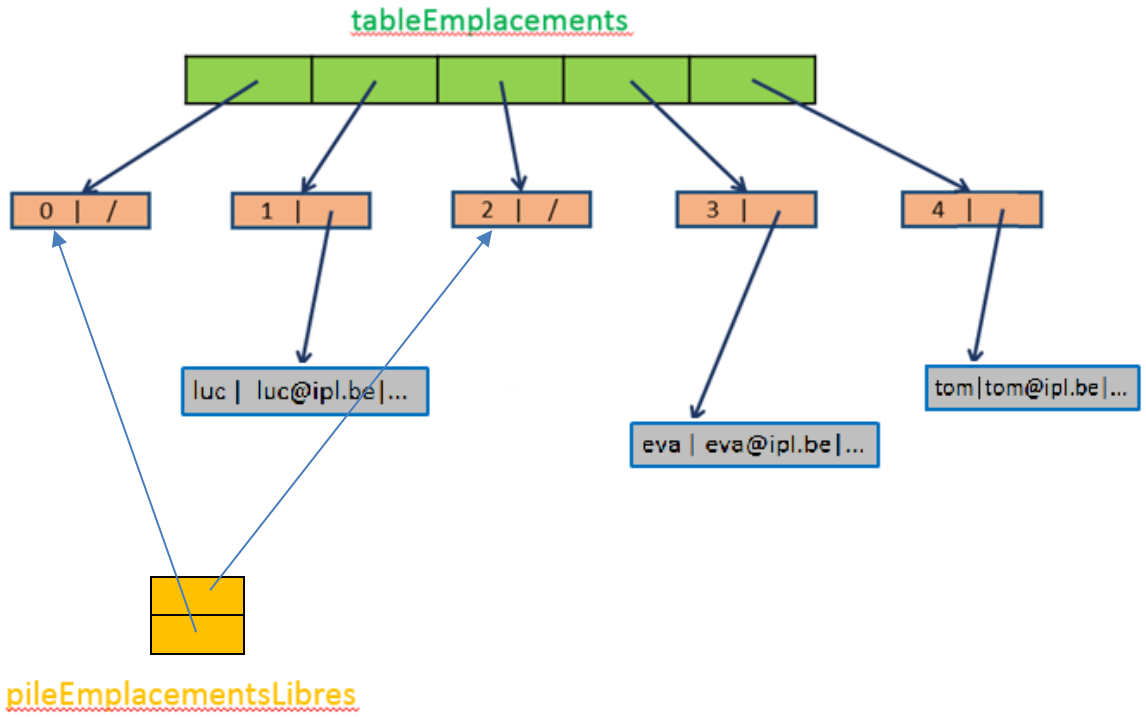
Dans la classe *Brocante*, apportez les modifications suivantes :

Modifiez les attributs :
La table ne contient plus des *String* mais des emplacements.
La pile contient des emplacements.

Modifiez le constructeur :
C'est le constructeur qui s'occupe de créer les différents emplacements et de les placer dans la table.

Modifiez la méthode `changerPhase()` :
La pile contient des emplacements.
Ne perdez pas de vue que la pile et la table référencient des mêmes objets !!!

Exemple :



Modifiez les signatures des méthodes de réservation et adaptez-les :

```
boolean reserver(Exposant exposant,int numeroEmplacement)
int attribuerAutomatiquementEmplacement(Exposant exposant)
```

Ajoutez quelques méthodes qui vont permettre d'éviter des encodages inutiles :

```
boolean estUnRiverain(String nom)
int nombreEmplacementsRiverain(String nom) // pour la phase 1 uniquement
boolean estLibre(int numeroEmplacement) //
boolean emplacementLibre() // il y a encore des emplacements de libre ?
```

Modifiez la classe *GestionBrocante*.
Veillez à ne pas demander l'encodage des coordonnées d'un demandeur si sa demande ne peut être aboutie.
Cependant, pour l'instant, si un demandeur demande plusieurs emplacements, il devra chaque fois introduire ses coordonnées. L'amélioration n°1 et un nouveau *map* permettra d'éviter cela.

AMELIORATION 1

On aimerait bien, pour un exposant distrait, lui permettre de retrouver les numéros de ses emplacements.

Pour éviter de devoir parcourir tous les emplacements, vous allez ajouter un *map* qui associe à un nom, un exposant.
Vous allez également ajouter à chaque exposant, la liste de ses emplacements.
Le *map* va contenir tous les exposants.
Il permet de retrouver les numéros des emplacements d'un exposant, mais permet beaucoup plus !
Il permet de vérifier l'existence d'un exposant via son nom et de trouver ses coordonnées,
Un parcours du *map* permet de lister tous les brocanteurs. Dans cette liste, chaque brocanteur n'y apparaîtra qu'une fois ! Étant donné la nature d'un *map*, il y a unicité de la clé.
Si on parcourt la table des emplacements pour dresser la liste des brocanteurs, un brocanteur qui a fait plusieurs réservations s'y trouverait plusieurs fois.
Attention, un *map*, ne propose pas d'itérateur, mais possède une méthode qui renvoie l'ensemble des valeurs. Cet ensemble peut être itéré.

```
return mapExposants.values().iterator();
```

Pour vous aider à visualiser ceci, commencez par compléter le schéma de la version de base.

Dans la classe *Exposant*, ajoutez une liste retenant ses emplacements (*LinkedList<Emplacement>*).

Ne mettez pas la méthode :

```
LinkedList<Emplacement> getListeEmplacements(){
      return listeEmplacements;
}
```

Mais plutôt la méthode :

```
Iterator<Emplacement> tousLesEmplacements(){
      return listeEmplacements.iterator() ;
}
```

Dans la classe *Brocante :*
Ajoutez un *map* (*HashMap<String,Exposant>*).
Ajoutez les méthodes

```
Exposant getExposant(String nom)
Iterator<Exposant> tousLesExposants()
```

Dans la classe *GestionBrocante :*

Améliorez l'attribution d'un emplacement : il est inutile de demander les coordonnées de quelqu'un qui se trouve dans le *map* des exposants !

Ajoutez deux points au menu :

3 : Consulter un exposant via son nom
4 : Lister tous les exposants

AMELIORATION 2

Permettez des désistements : un exposant peut libérer un emplacement qui lui a été attribué.

Ajoutez cette option dans la classe *GestionBrocante.*

Revoyez les classes *Exposant* et *Brocante* afin de maintenir les différentes structures de données cohérentes après libération d'un emplacement.

AMELIORATION 3

L'organisateur voudrait proposer des emplacements de différents types :
Les emplacements n'ont pas tous la même taille. Il y en des petits (S), des moyens (M) et des grands (L).
Il y a aussi des emplacements particuliers, par exemple, des stands de restauration (R)

En phase 1, le riverain continue à réserver les emplacements qu'il désire (max 3) via leurs numéros.
En phase 2, le demandeur devra préciser le type d'emplacement qu'il veut. L'attribution est automatique.

Dans la classe *Emplacement*, ajoutez un attribut type (char) et faites les modifications nécessaires.

Dans la classe *Brocante* apportez les modifications suivantes :

Pour l'attribution automatique des emplacements, une pile ne sera pas suffisante.
Il faut une pile par type d'emplacement.
Remplacez la pile par un map de piles.
La clé = un type (*Character*)
La valeur = une pile (*ArrayDeque<Emplacement>*).
Vous utiliserez donc un objet de la classe HashMap<Character,ArrayDeque<Emplacement>>
Pour vous aider à visualiser tout ceci, commencez par modifier le schéma de la version de base.

Modifiez le constructeur :
Il ne reçoit plus en paramètre le nombre d'emplacements mais une table (de caractères) avec les types des emplacements.

Voici un exemple de table passée en paramètre :

'S'	'R'	'L'	'S'	'A'	'S'	'X'	'M'

La brocante contient 8 emplacements.
L'emplacement 0 est un emplacement de taille S.
L'emplacement 1 est un stand de restauration.
L'emplacement 2 est un emplacement de taille L.
L'emplacement 3 est un emplacement de taille S.
L'emplacement 4 est un stand d'animation pour enfant.
…

Revoyez la méthode `changerPhase()` qui remplit les piles.

Modifiez les signatures de certaines méthodes et adaptez-les :

```
boolean emplacementLibre(char type)
int attribuerAutomatiquementEmplacement(char type, Exposant exposant)
```

Ces méthodes lancent une `IllegalArgumentException` si le type n'est pas reconnu.
.

Modifiez La classe *GestionBrocante.*
En phase 2, lors d'une réservation, il faudra maintenant demander le type d'emplacement.

AMELIORATION 4

Permettez à l'organisateur d'ajouter de nouveaux emplacements.

Dans la classe *Brocante*, remplacez la table des emplacements par un vecteur (*ArrayList*) et modifiez les différentes méthodes en conséquence.
Vous devrez aussi ajouter de nouvelles méthodes !

Dans la classe *GestionBrocante*, pensez à ajouter un point au menu permettant l'ajout d'emplacement

AMELIORATION 5

Un emplacement de taille L, peut être divisé en deux emplacements de taille S, s'il n'y a plus d'emplacement de type S disponible.
Un emplacement de taille L peut donc contenir 2 exposants différents.

Dans la classe *Emplacement*, on ne trouvera plus un exposant, mais 2 exposants.

Apportez les modifications nécessaires.

AMELIORATION 6

Prévoyez les payements et la possibilité pour l'organisateur de récupérer les adresses mails des exposants qui ne sont pas en ordre de payement.

AMELIORATION 7

Pour les plus courageux ! Cette amélioration demande vraiment beaucoup de changements et beaucoup d'initiatives.

Lorsqu'il n'y a plus de place pour un type d'emplacement, l'organisateur voudrait que les exposants, qui le souhaitent, soient placés dans une file d'attente.

Il vous faut ajouter une file d'attente par type d'emplacement → *map* de *file* d'attente
Un demandeur, n'est pas un exposant, mais il faut garder ses coordonnées. On pourrait imaginer une classe *Demandeur*. Mais un demandeur peut être un exposant en attente d'un emplacement supplémentaire. Pour éviter de la redondance d'information, on vous suggère de ne pas à ajouter une classe *Demandeur*, mais d'accepter l'existence d'exposant sans emplacement.
Que faire lors d'un désistement, si des demandeurs sont en attente ? A vous de trouver une belle solution pour gérer cette situation.