Le vecteur et la pile implémentés via des tables

Exercices à soumettre

A Implémentation de l'interface *PileDeCaracteres*

A1 Complétez le document *A1*.
La solution de cet exercice se trouve sur moodle dans le dossier *solutions*. La solution est là pour vérifier vos réponses après avoir terminé l'exercice ! Le document s'appelle *A1Sol*.

A2 Implémentez l'interface *PileDeCaracteres* :
Complétez la classe *PileDeCaracteresImpl* et testez-la avec la classe *TestPileDeCaracteresImpl.*
La classe *TestPileDeCaracteresImpl* reprend les mêmes tests qu'à l'exercice précédent.
Respectez bien la *JavaDoc*. Celle-ci est présente dans l'interface *PileDeCaracteres*.

B Applications utilisant la classe *PileImpl*

Vous trouverez sur moodle, dans le dossier *classes Java Plus*, la classe *PileImpl* implémentée. Copiez cette classe ainsi que l'interface *Pile* dans votre répertoire *src*.

La classe *PileImpl* est similaire à la classe que vous venez d'écrire mais c'est une **classe générique** Elle peut être utilisée pour des objets de différents types.
E est le type générique. Lors de l'instanciation de cette classe, on doit lui donner une valeur, qui doit être une classe (ou une interface).
Par exemple : `pile = new PileImpl<Integer>` pour une pile d'entiers.

B1 Cabines d'essayage

Y a-t-il une cabine de libre ?

On vous demande de participer à l'informatisation d'une attribution automatique de cabines d'essayage.
Chaque cabine porte un numéro. **La numérotation de ces cabines commence à 1.**
Le client se verra attribuer automatiquement un numéro d'une cabine libre.

Complétez la classe *CabinesDEssayage*.

Un des attributs de cette classe est une pile d'entiers.
Cette pile contient les numéros des cabines libres.
Elle permet une attribution efficace (peu coûteuse) d'une cabine libre. C'est la cabine qui se trouve au sommet de la pile qui est attribuée.
Vous utiliserez la classe *PileImpl* fournie.

L'autre attribut est une table de booléens. Elle permet de tester la cohérence du système.
En effet, il doit être impossible de libérer une cabine qui n'était pas occupée.
Elle permet de savoir, pour chaque cabine, si elle est libre ou occupée.
true → la cabine est occupée false → la cabine est libre

Voici un exemple pour vous aider à mieux comprendre le rôle des attributs :
Le magasin compte 10 cabines.
Les cabines 1, 3, 6, 8, 9 et 10 sont occupées.
Les cabines 2, 4, 5 et 7 sont libres.
Le prochain client se verra attribuer la cabine 5.

5
2
7
4

Pile avec les numéros des cabines libres

0	1	2	3	4	5	6	7	8	9
T	F	T	F	F	T	F	T	T	T

Table des occupations

Testez cette classe avec la classe *TestCabinesDEssayage* fournie.

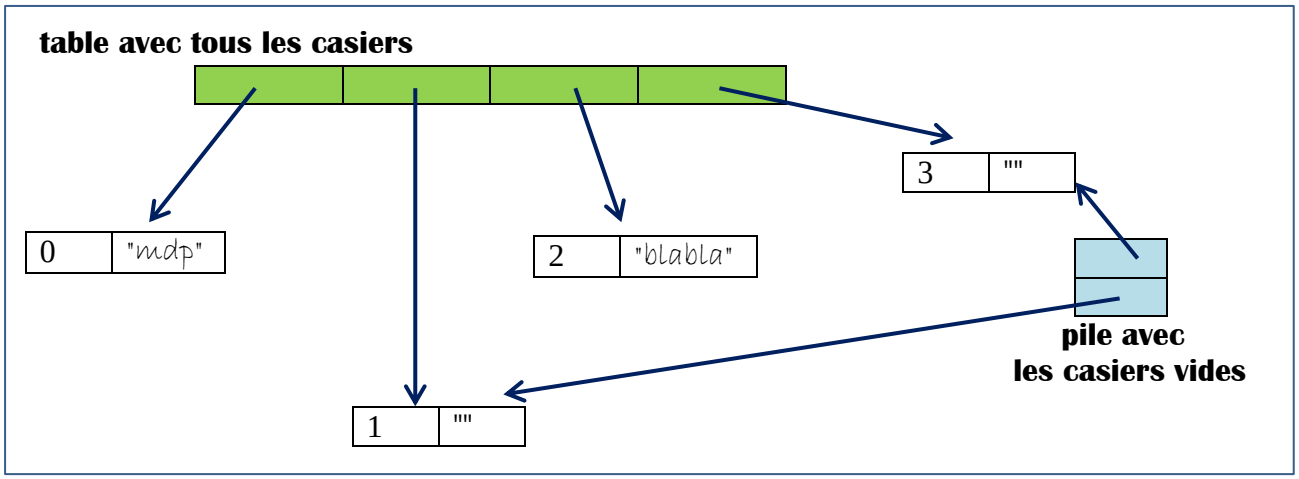
B2 Consigne de gare

a) La gare décide de s'équiper d'un nouveau système informatisé pour sa consigne.
Des nouveaux casiers (numérotés en continu à partir de 0) vont être installés dans la gare pour permettre aux voyageurs de déposer leurs bagages.
Le voyageur payera un forfait de 2 euros pour l'occupation d'un casier quel que soit le temps d'occupation.
Le voyageur qui désire occuper un casier ira à la borne informatisée.
Si un casier est libre, le voyageur sera invité à payer les 2 euros et à introduire un mot de passe de son choix.
Les casiers étant tous identiques, le numéro du casier lui sera attribué automatiquement et la porte de ce casier pourra être fermée.
Lorsque le voyageur désirera récupérer ses bagages, il se présentera à la borne informatisée. Il introduira le numéro du casier et son mot de passe. Si le mot de passe est correct le casier s'ouvrira automatiquement. Il est libéré. (Si le voyageur désire à nouveau l'utiliser, il devra refaire une demande et payer 2 euros.)

Vous allez compléter la classe *Consigne.*
Tous les casiers sont sauvegardés dans une table. Cette table contient des objets de la classe *Casier*. L'indice correspond au numéro de casier. Un casier est reconnu par ce numéro et retient un éventuel mot de passe.

Pour gérer le problème d'attribution d'un casier libre, elle utilise une pile d'objets de la classe *Casier*. Celle-ci contiendra les casiers libres.
Vous utiliserez la classe *PileImpl* fournie.

Voici un schéma pour vous aider à mieux visualiser un objet de la classe *Consigne* :
Dans cet exemple, la consigne contient 4 casiers. 2 de ces 4 casiers sont occupés.



Le constructeur reçoit en paramètre le nombre de casiers et va créer les différents casiers. Il va placer chaque casier simultanément dans la pile **et** la table.
Ne perdez pas de vue que la pile et la table référencient des mêmes objets !!!

Testez votre implémentation grâce à la classe *TestConsigne.*

C Implémentation de l'interface Vecteur

C1 Complétez la classe *VecteurDeCaracteresImpl* et testez une méthode à la fois avec la classe *TestVecteurDeCaracteresImpl.*
Respectez bien la *JavaDoc*. Celle-ci est présente dans l'interface.

D Applications utilisant la classe *PileImp*l et la classe *VecteurImpl*

Vous trouverez sur moodle, dans le dossier *classes Java Plus*, l'interface *Vecteur* et la classe *VecteurImpl.* Copiez-les dans le répertoire *src.*

D1 Salle d'exposition

Le responsable d'une salle d'exposition d'œuvres d'art demande une application informatique qui devrait permettre aux visiteurs de consulter aisément son catalogue.
Chaque œuvre d'art exposée porte un numéro. Il s'agit du numéro de son emplacement.
La numérotation commence à 0
Le visiteur pourra obtenir des informations sur une œuvre en tapant son numéro.
Le nombre d'œuvres exposées n'est pas limité.
La suppression d'une œuvre a comme effet de libérer l'emplacement.
Un emplacement libéré n'est pas supprimé.
On crée un nouvel emplacement que si tous les emplacements déjà existants sont occupés.

La classe *Emplacement* est donnée.
Un emplacement contient un numéro et éventuellement une œuvre d'art (String).
L'œuvre est à null, si l'emplacement est vide.

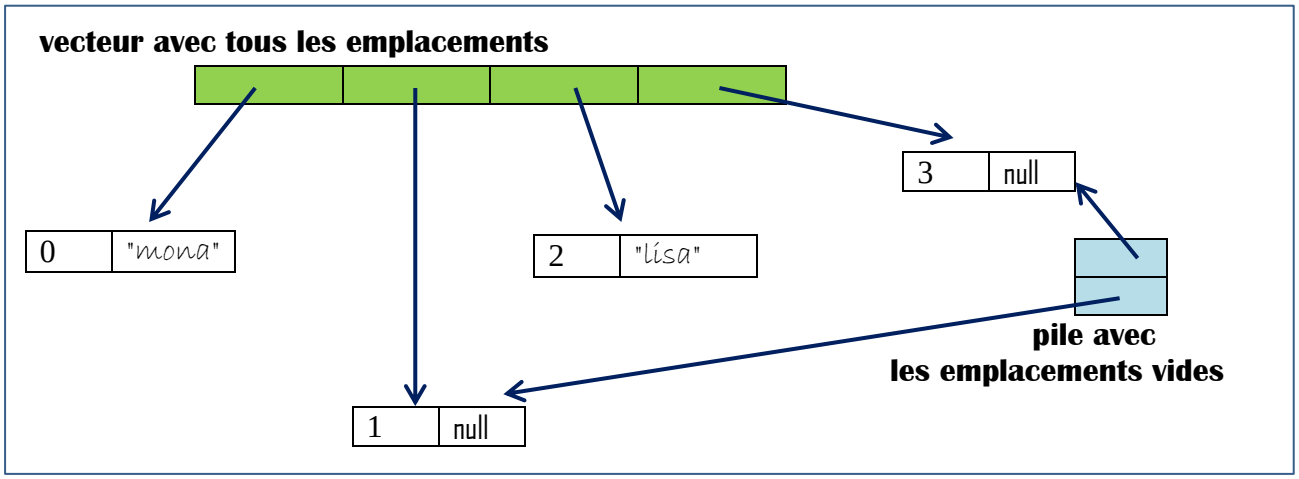
Complétez la classe *SalleExposition*.

Les emplacements sont enregistrés dans un vecteur.
Le numéro de l'emplacement correspond au rang.

Au départ, le vecteur est vide.

La méthode `ajouterOeuvre()` ajoute une œuvre.
Elle l'ajoutera, de préférence, dans un emplacement qui s'est libéré.
Si tous les emplacements existants sont occupés, un nouvel emplacement est créé et ajouté en fin de vecteur.
Cette méthode sera peu coûteuse en utilisant une pile ! Cette pile va contenir les numéros des emplacements libérés.
Vous utiliserez les classes *PileImpl* et *VecteurImpl* fournies.

Voici un schéma pour vous aider à mieux visualiser un objet de la classe *SalleExposition* :
Dans cet exemple, la salle d'exposition contient 4 emplacements dont 2 sont occupés.



Ne perdez pas de vue que la pile et le vecteur référencient des mêmes objets !!!

Complétez la classe *GestionSalleExposition.*
Via un menu, le responsable doit pouvoir ajouter, consulter ou supprimer une œuvre.
Il doit aussi pouvoir afficher le nombre d'emplacements, le nombre d'œuvres et toutes les œuvres présentes.
Il vous reste à écrire les options de consultation et de suppression d'une œuvre (via le numéro).
Cette classe va vous permettre de tester la classe *SalleExposition*.

Exercice défi

Un classique !

Ecrivez la méthode `verificationParenthesage(String texte)` de la classe *Parenthesage*. Cette méthode vérifie si le texte passé en paramètre est correct pour les parenthèses, accolades et crochets :

Toute parenthèse, toute accolade ou tout crochet ouvrant sera suivi d'une parenthèse ou d'une accolade ou d'un crochet fermant du même type!
Toute parenthèse, accolade ou crochet fermant est précédé d'une parenthèse, d'une accolade ou d'un crochet ouvrant du même type!

Voici quelques exemples :
(a (b [c] d { e } f) g) → OK
a)b(c) → KO
[a (b] c) → KO

Cette méthode ne parcourt qu'une seule fois le texte.
Elle nécessite l'utilisation d'une pile!

Remarque:
Une chaîne de caractères est une table de caractères!
Pour connaître la taille de la chaîne de caractères `texte` → `texte.length()`
Pour connaître le caractère de rang i de la chaîne de caractères `texte` → `texte.charAt(i)`

Exercices supplémentaires

B2
b) La gare décide de s'équiper de casiers de 3 formats différents (notés : 1, 2 et 3)
(Le tarif dépendra du format choisi :
Format 1 : 1 euros
Format 2 : 2 euros
Format 3 : 5 euros)
Complétez la classe *Consigne3Formats*.

Elle propose les mêmes opérations que la classe *Consigne.*

Cependant, le constructeur de la classe reçoit en paramètre un tableau précisant le format de chaque casier.

Par exemple :

3	1	1	2	1	1	1	2

Il y a 8 casiers :

Casier 0 : format 3
Casier 1 : format 1
Casier 2 : format 1
Casier 3 : format 2
…

Les méthodes `attribuerCasierLibre()` et `resteUnCasierLibre()` possèdent un paramètre supplémentaire : le format du casier.
Prévoyez une *IllegalArgumentException* si le format n'est pas valide.
La classe *TestConsigne3Formats* vous est fournie. Elle doit être en partie complétée.

c) La gare décide de s'équiper de casiers de 10 formats différents.
Ecrivez une classe *Consigne10Formats*.
Cette classe contiendra un tableau de 10 piles !