***Haute Ecole Léonard de Vinci***

Institut Paul Lambin Session de juin 2021

Examen de SD

B. Lapière – I. Cambron – L. Lecharlier – A. Dupont

Année d’études : 1ère Informatique

Durée de l’examen : 2 h 30

**Ce questionnaire est le questionnaire bleu.**

**Vous trouverez la couleur qui vous a été attribuée dans le document listeEtudiantsCouleurs.**

1. **Liste chaînée (7 points)**

**C’est pour cette question que vous pouvez utiliser votre jokerCorona.**

**Si vous décidez de l’utiliser, ne soumettez pas la classe demandée. Toute classe soumise sera corrigée et cotée. C’est cette cote qui sera prise en compte, même si elle est moins bonne que celle de votre joker !**

Le bal de promo figure depuis longtemps dans la culture américaine. Semblable à un mariage ou des fiançailles, le choix de son ou sa cavalier(e) fait partie d’un rituel.

La classe *ListeBalPromo* va contenir la liste des lycéens inscrits au bal. On peut y découvrir les couples formés pour l’occasion.

On y retrouve une alternance de garçons et de filles. Pour chaque couple, on place d’abord le garçon et ensuite la fille.

On ne peut pas retrouver plusieurs fois un même lycéen dans cette liste.

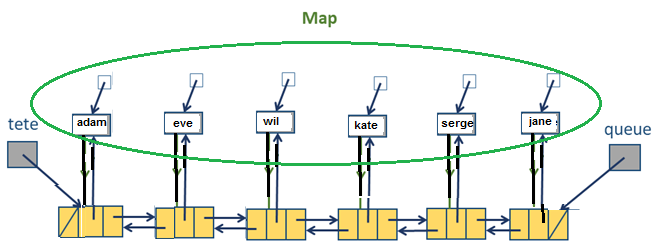
adam ⮀ eve ⮀ william ⮀ kate ⮀ serge ⮀ jane

L’implémentation choisie est une liste doublement chaînée (sans sentinelle) + un map.

La clé sera le lycéen recherché, la valeur associée est le nœud contenant ce lycéen.

Ce map permet d’accéder directement au nœud contenant un lycéen recherché.

Si la recherche échoue, cela signifie que le lycéen n’est pas dans la liste.



La classe *Lyceen* vous est fournie.

Chaque lycéen possède un nom, un sexe et un âge.

Il n’existe pas d’homonyme.

La classe *ListeBalPromo* contient une classe interne Nœud.

Chaque nœud contient un lycéen, une référence vers le nœud précédent et une vers le nœud suivant.

La classe *ListeBalPromo* possède 3 attributs :

Le nœud de tete, le nœud de queue et le map dont l’utilité est donnée ci-dessus.

La classe propose un constructeur qui construit une liste vide.

Les méthodes taille() et estVide() vous sont données.

(La classe contient également un autre constructeur et 2 méthodes de parcours de la liste qui vont servir pour la classe de tests. Ne les modifiez pas.)

**On vous demande de compléter les méthodes donnerNombreLyceensMajeurs(),ontTous16AnsOuPlus()** , **donnerPartenaire()et ajouterCouple().**

Respectez la *JavaDoc* et l’implémentation choisie.

On vous impose 2 contraintes :

La méthode donnerNombreLyceensMajeurs()doit être une méthode itérative et la méthode ontTous16ansOuPlus() doit être une méthode récursive.

Vous pouvez introduire d’autres méthodes que celles présentes.

Pour vos tests, vous pouvez utiliser la classe *TestListeBalPromo*.

1. **EnsembleTrieImpl (5 points)**

Un ensemble trié possède les caractéristiques d’un ensemble mais les éléments y sont triés.

Les éléments qui s’y trouvent sont comparables.

Dans son interface, en plus de toutes les méthodes de l’ensemble, on retrouve des méthodes qui sont intéressantes avec des éléments comparables.

Pour l’examen, nous vous avons fourni une interface *EnsembleTrie* réduite.

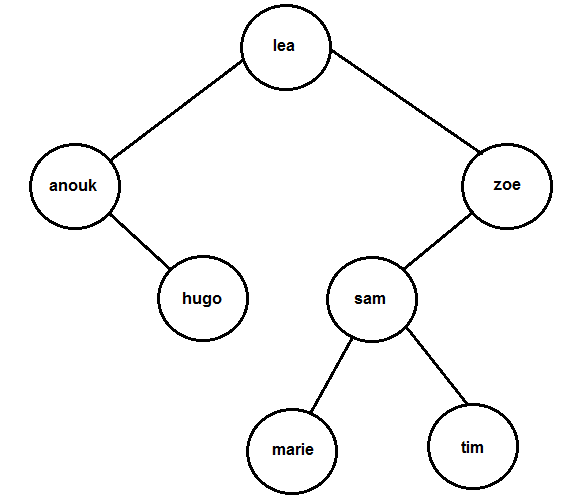
Nous vous demandons d’implémenter cette interface à l’aide d’un arbre binaire de recherche (ABR).

Cet ABR ne peut pas contenir de doublons.

On fait le choix que la descendance **gauche** d’un nœud ne contiendra que des éléments « **inférieurs »** à l’élément de ce nœud et la descendance **droite** d’un nœud ne contiendra que des éléments « **supérieurs »**.

Voici un exemple :

(Les chaînes de caractères sont triées selon l’ordre alphabétique.)

****

Vous allez compléter la classe *EnsembleTrieImpl* qui implémente l’interface *EnsembleTrie*.

Cette classe contient une classe interne *Nœud*.

Elle possède 2 attributs, le nœud racine et la taille.

Elle propose 2 constructeurs dont un va servir pour la classe de tests.

La méthode toString()vous est donnée. Pour l’arbre ci-dessus la méthode renvoie :

[ [ [ ] anouk [ hugo ] ] lea [ [ [ marie ] sam [ tim ] ] zoe [ ] ] ]

Cette classe est générique. Elle ne doit pas tester si ses éléments appartiennent à une classe qui implémente l’interface *Comparable*.

Attention, pour pouvoir utiliser la méthode compareTo(), il faut « caster » l’élément en *Comparable*.

Par exemple, pour vérifier si l’élément1 est « inférieur » à l’élément2 :

if(((Comparable<E>)element1).compareTo(element2)<0) …

Eclipse va vous afficher un avertissement : 

Vous ne devez pas en tenir compte.

Les méthodes taille() et estVide() vous sont données.

**Dans la classe *EnsembleTrieImpl*, vous allez compléter les méthodes min(), ajouter() et predecesseur().**

Complétez ces méthodes en respectant bien la *JavaDoc* (reprise dans l’interface) et les choix d’implémentation imposés.

Vous pouvez ajouter des méthodes.

La classe *TestEnsembleTrieImpl* permet de tester vos méthodes avec l’arbre mis en exemple ci-dessus ainsi que l’arbre vide.

La classe *String* implémente *Comparable*.

1. **API JAVA : Application Brasserie (8 points)**

Nous allons implémenter une application de vente de casiers de bière pour une brasserie.

Celle-ci s’inspire du processus de vente d’une brasserie belge mondialement connue.

L’abbaye de Westvleteren n’a aucune vue commerciale. Sa production de bière est très limitée.

Elle veut que ses clients soient des consommateurs directs (pas de revente en magasin ou autre)

Plusieurs sessions de vente sont organisées par an.

Le nombre de casiers de bière à vendre lors d’une session dépendra de la production de la brasserie.

Pour satisfaire le plus de monde possible, le nombre de casiers que peut acheter un client au cours d’une session de vente est limité à 3.

Les commandes d’une session de vente se font via un magasin en ligne qui ne sera ouvert qu’à un moment bien précis.

Au moment de la vente, le client est placé dans une file d’attente.

Lorsque son tour arrive, il voit le nombre de casiers restants. Il est invité à entrer un nombre de casiers. Si le client avait déjà fait une commande, ce nombre sera ajoute au nombre de casiers déjà commandés.

Le magasin en ligne ferme lorsque tous les casiers sont vendus.

Les clients viendront chercher leurs casiers sur place après fermeture du magasin en ligne.

Implémentation choisie :

Dans le cadre de l’examen, nous n’allons pas introduire de classe *Client.*

Le client sera représenté par son nom (*String*).

La classe *Commande* vous est donnée. Une commande retient le client qui a passé cette commande et le nombre de casiers qu’il a demandé. Il est possible de modifier le nombre de casiers.

Vous allez complétez la classe *SessionDeVente*.

Le constructeur de cette classe et les attributs sont donnés.

Pour gérer la file d’attente, on a fait le choix d’une file (*ArrayDeque<String>*).

Un client peut prendre place dans cette file à condition qu’il ne s’y trouve pas déjà.

On décide d’ajouter l’ensemble (*HashSet<String>*) des clients présents dans la file d’attente. La vérification d’existence est moins coûteuse dans un ensemble que dans une file !

On va retenir toutes les commandes dans une liste (*ArrayList<Commande>*).

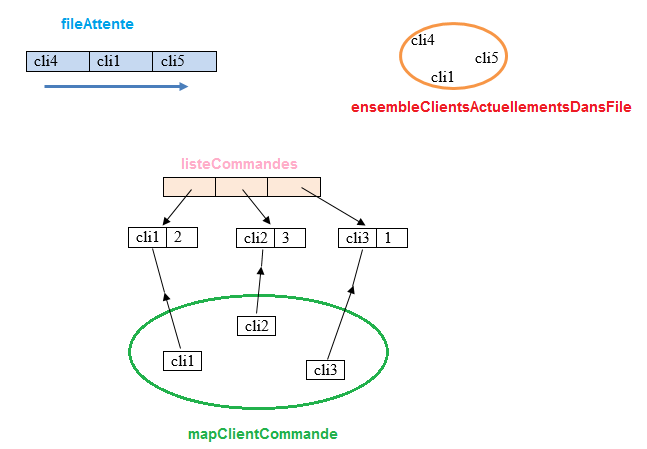
Un *map* (*HashMap<String,Commande>*) permet de retrouver facilement la commande d’un client.

Ne perdez pas de vue que la liste des commandes et le map référencient des mêmes commandes.

Exemple :

Dans cet exemple, on remarque que cli1, cli2 et cli3 ont réservés respectivement 2, 3 et 1 casier.

Actuellement dans la file d’attente on retrouve cli5, cli1 et cli4. Le prochain client qui sera traité sera cli4. Cli1 a pu se retrouver dans la file d’attente car il n’a pas encore commandé les 3 casiers maximum auquel il a droit.



En plus de ces structures de données, des variables et constantes permettant une implémentation efficace figurent parmi les attributs proposés.

Prenez bien connaissance de tous les attributs de la classe avant de vous lancer dans la programmation !

Complétez la classe *SessionDeVente* en respectant bien la *JavaDoc* et les choix d’implémentation imposés ci-dessus.

Si c’est plus facile pour vous, vous pouvez introduire d’autres attributs et des méthodes (*private*)

Utilisez uniquement les méthodes des classes *ArrayList*, *ArrayDeque, HashMap et HashSet* reprises dans le document API\_JAVA.

La classe *GestionSessionDeVente* va vous servir pour tester la classe *SessionDeVente*.

Vous pouvez la modifier.

Ne perdez pas de temps à l’améliorer. Cette classe ne sera pas évaluée.