***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 rouge.**

**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 !**

Un tournoi de tennis de double mixte est organisé pour les étudiants de l’IPL.

La classe *ListeTournoiDoubleMixte* va contenir la liste des étudiants inscrits au tournoi. 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 étudiant dans cette liste.

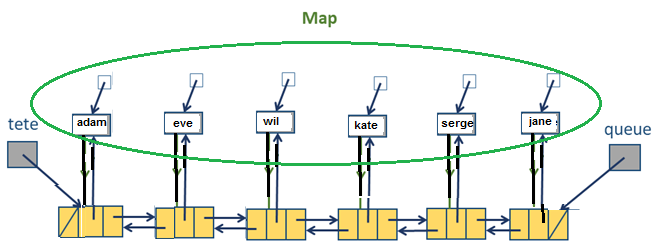
adam ⮀ eve ⮀ william ⮀ kate ⮀ serge ⮀ jane

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

La clé sera l’étudiant recherché, la valeur associée est le nœud contenant cet étudiant.

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

Si la recherche échoue, cela signifie que l’étudiant n’est pas dans la liste.



La classe *Etudiant* vous est fournie.

Chaque étudiant possède un nom, un sexe, une section et une année d’étude.

Il n’existe pas d’homonyme.

La classe *ListeTournoiDoubleMixte* contient une classe interne Nœud.

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

La classe *ListeTournoiDoubleMixte* 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 donnerNombreEtudiantsBloc(),sontTousEnInfo()** , **donnerPartenaire()et ajouterCouple().**

Respectez la *JavaDoc* et l’implémentation choisie.

On vous impose 2 contraintes :

La méthode donnerNombreEtudiantsBloc()doit être une méthode itérative et la méthode sontTousEnInfo() doit être une méthode récursive.

Vous pouvez introduire d’autres méthodes que celles présentes.

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

1. **ABRSDImpl (5 points)**

Pour l’examen, intéressons-nous à un arbre binaire de recherche(ABR) sans doublon (SD).

Les éléments qui s’y trouvent sont des éléments comparables.

On peut trouver de nombreuses fonctionnalités intéressantes avec des éléments comparables.

Pour l’examen, nous vous avons fourni une interface ABRSD.

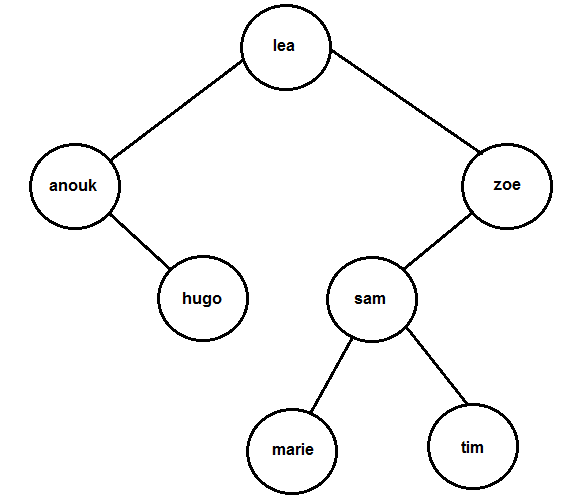
Nous vous demandons d’implémenter cette interface.

Cet ABRSD 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 *ABRSDImpl* qui implémente l’interface *ABRSD*.

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 *ABRSDImpl*, vous allez compléter les méthodes elementLePlusPetit(), insere() et elementPrecedent().**

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 *TestABRSDImpl* 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 Take-away (8 points)**

Vous allez implémenter une application de réservation du menu du dimanche midi pour un restaurant.

Depuis le début de la crise sanitaire, ce restaurant propose tous les dimanches un menu 3 services à emporter. Ces menus remportent beaucoup de succès. Compte tenu de ses infrastructures, le nombre de menus à emporter est limité.

Ce restaurant voudrait mettre en place un système de réservation en ligne. Son but est d’éviter une file sur le trottoir qui commence de plus en plus tôt le dimanche matin et d’avoir des déçus.

Pour contenter le plus de clients possible, il compte limiter le nombre de menu, pour un dimanche bien précis, à 4 par client.

Nous allons nous intéresser aux réservations des menus d’un dimanche en particulier.

Les réservations se font en ligne via son site, le mercredi précédent ce dimanche à 19h.

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

Lorsque son tour arrive, il voit le nombre de menus restants. Il est invité à entrer un nombre de menus. Si le client avait déjà fait une réservation, ce nombre sera ajouté au nombre de menus déjà réservés.

Le magasin en ligne ferme lorsque tous les menus sont réservés.

Les clients viendront chercher leurs menus sur place le dimanche matin.

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 *Reservation* vous est donnée. Une réservation retient le client qui a passé cette réservation et le nombre de menus qu’il a demandé. Il est possible de modifier le nombre de menus.

Vous allez complétez la classe *TakeAway*.

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 la 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. La vérification d’existence est moins coûteuse dans un ensemble que dans une file !

On va retenir toutes les réservations dans une liste (*ArrayList<Reservation>*).

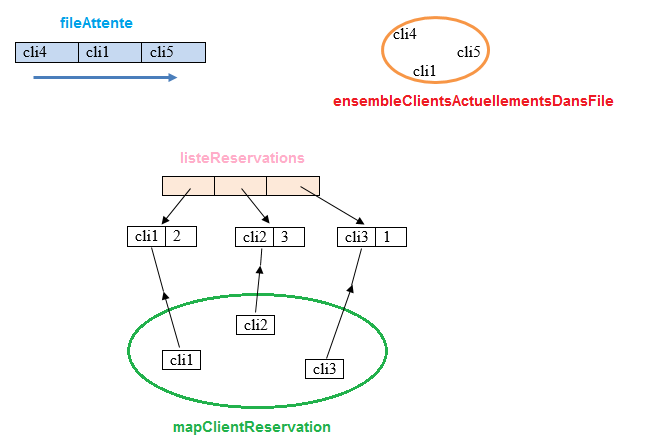
Un *map* (*HashMap<String,Reservation>*) va permettre de retrouver facilement la réservation d’un client.

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

Exemple :

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

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 réservé les 4 menus 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 *TakeAway* 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 *GestionTakeAway* va vous servir pour tester la classe *TakeAway*.

Vous pouvez la modifier.

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