***Haute Ecole Léonard de Vinci***

Institut Paul Lambin Session de juin 2020

Examen de SD

R. Baroni - A. Legrand – 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 (8 points)**

Une école primaire possède plusieurs classes.

Chaque classe a un instituteur.

Vous allez complétez la classe *Ecole* qui permet la gestion des différentes classes.

Dans le cadre de l’examen, nous n’allons pas introduire de classe *Eleve*, *Instituteur*, *Classe* ou autres.

Les élèves et les instituteurs seront identifiés par leurs prénoms (String).

Une classe sera identifiée par un code (String). Ex : 1A, 1B, 2A, …

Voici l’implémentation qui a été choisie :

Les élèves et instituteurs sont placés dans une liste.

Cette liste est implémentée via pointeurs.

Cette liste est simplement chaînée sans sentinelle.

Elle ne peut pas être vide. Elle contient au minimum toujours les instituteurs.

Elle ne contient pas d’homonymes.

Voici comment y sont placés les élèves :

Tous les élèves d’une même classe se suivent.

Pour chaque classe, on commence par trouver son instituteur et ensuite les élèves qui la composent.

Au sein de chaque classe, les élèves y apparaitront du plus récent (dernier ajouté) au plus ancien.

Pour permettre de respecter cet ordre, il faut pouvoir accéder directement à une classe. Ceci va se faire grâce à une référence du nœud qui contient l’instituteur.

Ex d’une école:

L’école contient 4 élèves répartis dans 3 classes : 1A, 2B et 6A.

La classe 1A a comme instituteur 1Ainstit et contient 2 élèves : 1Ael1 et 1Ael2.

La classe 2B a comme instituteur 2Binstit et contient 1 élève : 2Bel1.

La classe 6A a comme instituteur 6Ainstit et contient 1 élève : 6Ael1.

Voici la liste :

1Ainstit 1Ael2 1Ael1 2Binstit 2Bel1 6Ainstit 6Ael1

Après ajout d’un élève dans la classe 1A (1Ael3) la liste devient:

1Ainstit 1Ael3 1Ael2 1Ael1 2Binstit 2Bel1 6Ainstit 6Ael1

L’élève qui vient d’être ajouté est placé directement après son instituteur.

La classe *Ecole* contient une classe interne *Nœud*.

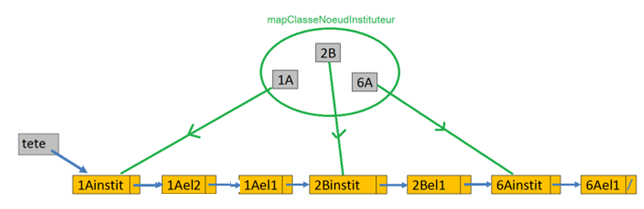
Chaque nœud possède un « élève » ou un « instituteur » et référencie le nœud suivant.

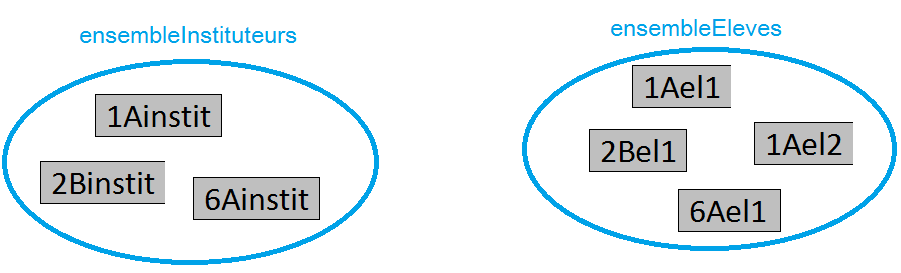
La classe *Ecole* possède comme attribut le nœud de tête de la liste et un *map* qui associe à chaque classe, le nœud de la liste qui contient son instituteur.

Pour optimiser certaines méthodes, la classe *Ecole* contient encore 2 autres attributs :

L’ensemble des élèves et l’ensemble des instituteurs.

Voici le schéma de l’école donnée en exemple ci-dessus :

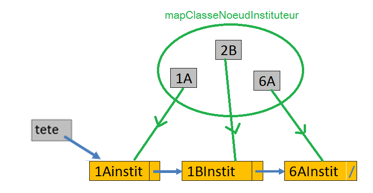


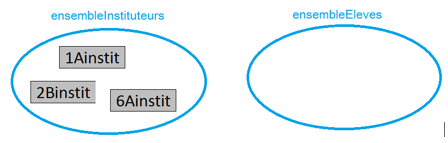


Le constructeur et la méthode toString() sont donnés.

Le constructeur, grâce à une table passée en paramètre, construit une liste avec les instituteurs.

Voici le schéma de l’école après appel du constructeur (avant ajouts d’élèves):





**Dans la classe *Ecole*, vous allez compléter les méthodes** estEleve(), estInstituteur(), estClasse(), estHomonyme(), ajouterEleve(), supprimerEleve()et donnerClasse().

Respectez la *JavaDoc* et l’implémentation choisie.

Pour vos tests, utilisez la classe *TestEcole*.

1. **Arbres binaires (5 points)**

Chaque instance de la classe *ExpressionArithmetique* mémorise une expression arithmétique sous forme d’un arbre binaire de caractères.

Une expression arithmétique contient des opérateurs binaires ( +, - , \* , / ) et des constantes entières.

Exemple d’expression :

exp1 : ( ( 3 – 2 ) + ( 4 \* ( 9 / 3 ) ) )

Remarques importantes :

* On se limitera à des valeurs positives, codées sur un seul chiffre (de 0 à 9 donc).
* Les opérations seront toujours binaires : x+y, x-y, x\*y, x/y. Pas de –x tout seul.
* Vous pouvez supposer que l’arbre est correct : il contient bien une expression arithmétique !

**Dans la classe *ExpressionArithmetique*, vous allez compléter les méthodes** nombreOperations(), uniquementDes()et nombreEntiersDifferents().

Ces méthodes seront récursives.

Vous pouvez ajouter des méthodes.

La classe *TestExpressionArithmetique* permet de tester vos méthodes avec l’arbre mis en exemple ci-dessus ainsi que les 2 arbres donnés à la page suivante :

exp2 : ( 4 + ( ( 8 \* ( 4 - 1 ) ) + 4 ) - 1)

exp3 : ( ( 3 + 2 ) + ( 4 + ( 9 + 3 ) ) )

1. **API JAVA : Application Fancy fair (7 points)**

C’est bientôt la fancy fair et son traditionnel spectacle des enfants!

Le responsable des réservations désire informatiser l’attribution des différentes places pour le spectacle.

Toutes les places sont numérotées. La numérotation des places commence à 0. Plus on s’éloigne de la scène, plus les numéros sont grands.

Les premières rangées seront prioritairement attribuées aux enfants. Un nombre limité de places leur sera réservé.

L’attribution des places se déroule en 2 phases.

Lors des préventes (phase 1), on peut réserver des places attribuées aux enfants et des places attribuées aux adultes.

Les rangées pour les enfants vont se remplir au fur et à mesure, sans trou, en partant de la place n°0.

Par contre les adultes, auront la possibilité de choisir leurs places.

Lors de la réservation, une personne pourra réserver plusieurs places en son nom. Pour chaque place réservée, on retiendra uniquement le nom de la personne qui a fait la demande. Il n’y a pas de limite en nombre de places demandées.

Le jour même de la fancy fair (phase 2), toutes les places (enfant et adulte) non attribuées seront automatiquement distribuées. On commencera par attribuer les places libres en partant de la première rangée et ainsi de suite.

Lors de cette phase, on ne peut plus choisir sa place et on ne fera plus de distinction entre place enfant et place adulte. Des adultes pourraient se retrouver dans une rangée d’enfants et inversement.

On attribuera une place à la fois lors de cette phase.

L’application doit enregistrer/gérer les différentes réservations.

Voici l’implémentation qui a été choisie :

Dans le cadre de l’examen, nous n’allons pas introduire d’autres classes que les classes *FancyFair* et *GestionFancyFair*.

La personne qui demande des places sera représentée par un *String*.

La place sera représentée par son numéro (*int*). La numérotation commence à 0.

Le constructeur de la classe *FancyFair* reçoit comme paramètres le nombre total de places et le nombre de places réservées pour les enfants.

Les réservations seront enregistrées dans une table de *String*.

Chaque case correspond à une place.

Au départ, cette table ne contiendra que des null.

On y placera, au fur et à mesure, les noms des personnes qui font les demandes.

Pour délimiter dans la table les places « enfant » des places « adultes », on décide d’introduire un attribut qui retient le numéro de la première place pour adulte.

Ex :

tableDesPlaces

0 1 2 3 4 5 6 7 8 9

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| « sam» | « nico» | null | null | « sam » | « sam » | null | « nico» | null | null |

premPlaceAdulte = 3

Il y a 10 places, 3 sont réservées pour les enfants

Les places 0, 1 et 2 sont donc réservées pour les enfants, les suivantes pour les adultes.

Les places 0, 4 et 5 sont attribuées à sam.

Les places 1 et 7 sont attribuées à nico.

Les autres places sont libres.

La phase d’attribution des places est un attribut de la classe. Elle est initialisée à 1.

Lors de la phase 1, pour optimaliser l’attribution d’une place enfant, on retiendra dans un attribut le dernier numéro attribué à un enfant.

Dans l’exemple :

derPlaceEnfantAttribuee = 1

Lors de la phase 2, les places sont attribuées de façon automatisée.

Pour ce faire, les numéros des places libres seront placés dans une file.

Dans l’exemple, pour débuter la phase 2, voici le contenu de cette file:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 2 | 3 | 6 | 8 | 9 |

filePlacesLibres

Les places seront attribuées dans l’ordre de cette file : d’abord 2, puis 3, 6, etc.

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

Complétez la classe FancyFair 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* et *ArrayDeque* reprises dans le document API\_JAVA.

La classe *GestionFancyFair* va vous servir pour tester la classe *FancyFair*.

Vous pouvez la modifier.

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