***Haute Ecole Léonard de Vinci***

Institut Paul Lambin Session de septembre 2020

Examen de SD

R. Baroni - A. Legrand – L. Lecharlier – A. Dupont

Année d’études : 1ère Informatique

Durée de l’examen : 2 h 30

1. **Liste chaînée (8 points)**

Pour la fête nationale, tous les enfants du village sont invités à s’habiller aux couleurs nationales : soit tout en noir, soit tout en jaune, soit tout en rouge. A un moment de la journée, ils se rassembleront par couleur afin de constituer un drapeau belge humain.

Tous les enfants qui veulent participer à cette activité doivent s’inscrire.

Vous allez complétez la classe *FeteNationale* qui permet la gestion de la liste des enfants participants.

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

Un enfant sera identifié par son nom (String).

Voici l’implémentation qui a été choisie :

Les enfants sont placés dans une liste.

Tous les enfants d’une même couleur se suivent.

L’ordre dans lequel les enfants d’une même couleur apparaissent n’a pas d’importance.

Elle contient toujours au minimum un enfant noir, un enfant jaune et un enfant rouge.

Elle ne contient pas de doublon.

Cette liste est implémentée via pointeurs.

Cette liste est doublement chaînée sans sentinelle.

La classe *FeteNationale* possède une classe interne *Noeud*.

Chaque nœud possède un « nom » et référencie le nœud précédent et le nœud suivant.

La classe Nœud a comme attributs :

Noeud precedent

String nom

Noeud suivant

Pour permettre de respecter l’ordre des couleurs dans la liste, il faut pouvoir accéder directement à la bonne couleur. Ceci va se faire grâce à une référence du nœud qui contient le premier enfant portant cette couleur.

Pour permettre un ajout efficace, le nouvel inscrit va prendre cet emplacement.

La référence du nœud qui contient le premier enfant portant la couleur est donc le dernier enfant inscrit pour cette couleur !

La classe *FeteNationale* possède donc les 3 attributs :

Noeud premierNoir

Noeud premierJaune

Noeud premierRouge

On va placer tous les enfants inscrits dans un *map*.

La clé est le nom de l’enfant et la valeur, le nœud qui le contient.

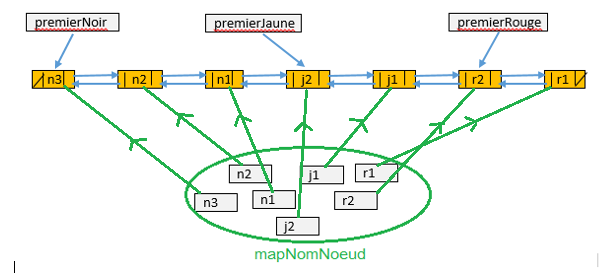
Ce *map* permet de vérifier l’existence d’un enfant dans la liste mais surtout d’y accéder directement !

La classe *FeteNationale* possède donc l’attribut supplémentaire :

HashMap<String, Noeud> mapNomNoeud

Ex d’une liste :

La liste contient 7 enfants, 3 noirs, 2 jaunes et 2 rouges.



Le constructeur et la méthode toString() sont donnés.

Le constructeur reçoit en paramètre un enfant de chaque couleur.

**Dans la classe *FeteNationale*, vous allez compléter les méthodes** ajouterParticipant() et donnerCouleur().

Respectez la *JavaDoc* et l’implémentation choisie.

Pour vos tests, utilisez la classe *TestFeteNationale*.

1. **Arbres binaires (5 points)**

Cette question est consacrée aux expressions arithmétiques (ex. 3 + -(4 \* 5)). Chaque expression arithmétique contient uniquement des **opérateurs** unaires (-, par ex -3), des **opérateurs** binaires (+, -, \*, / ) et des **opérandes** entiers composés d’un seul chiffre(0, 1, 2, 3, 4, 5, 6, 7, 8, 9).

Dans cet exercice, une expression est :

* soit une **expression simple** qui contient un opérande entier (ex. 3),
* soit une **expression composée** **unaire** qui contient l’opérateur unaire - et une sous-expression (ex. -(4 \* 5) où (4 \* 5) est la sous-expression).
* soit une **expression composée** **binaire** qui contient un opérateur binaire, une sous-expression de gauche et une sous-expression de droite (ex. 3 + -(4 \* 5) où 3 est la sous-expression de gauche et -(4 \* 5) est la sous-expression de droite).

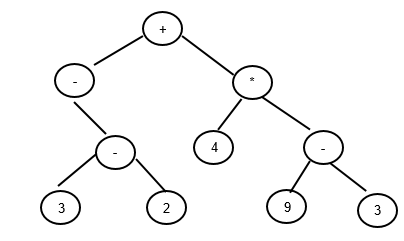
La classe *ExpressionArithmetique* héritede la classe *ArbreDeCaracteres*.

Elle est telle que nous avons abordée durant les cours et représente une expression arithmétique sous forme d’un arbre binaire de caractères.

Notez que dans le cas d’une expression **composée unaire,** c’est le fils droit de l’arbre qui contient la sous-expression.

Exemple d’expression :

exp1 = -( 3 – 2 ) + (4 \* (9 – 3)

****

**Dans le cadre de cet examen, vous pouvez supposer que l’arbre est correct. C’est-à-dire qu’il contient bien une expression arithmétique !**

**Dans la classe *ExpressionArithmetique*, vous allez compléter les méthodes** nombreMoins()et entiersTousDifferents().

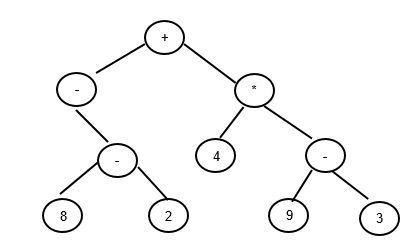
Ces méthodes seront récursives.

Vous pouvez ajouter des méthodes telles que les deux méthodes récursives nombreMoinsUnaire() et nombreMoinsBinaire()".

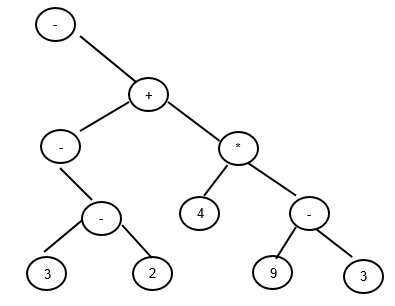
Suivez bien la *JavaDoc* et les consignes données.

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

exp2 = -( 8 – 2 ) + (4 \* (9 – 3)



exp3 = -( -( 3 – 2 ) + (4 \* (9 – 3))



1. **API JAVA : Application Brocante (7 points)**

Le responsable des réservations désire informatiser l’attribution des différents emplacements de la brocante annuelle du village.

Les emplacements sont répartis dans les rues, mais également dans le parking du grand magasin, le parking de la gare, la cours de récréation de l’école, …

N’importe qui peut exposer. Cependant les emplacements dans les rues seront prioritairement attribués aux riverains.

Chaque habitation a reçu un code permettant de directement réserver l’emplacement qui se trouve devant elle. Ces codes riverains ont été déposés dans les différentes boîtes aux lettres.

Tous les emplacements sont numérotés.

La numérotation des emplacements commence à 0.

Les petits numéros correspondent aux emplacements pour les riverains.

Pour chaque emplacement réservé, on retiendra le nom de la personne qui a fait la demande.

Il n’y a pas de limite en nombre d’emplacements demandés.

L’attribution des emplacements se déroule en 2 phases.

Phase 1

Lors de cette phase, il y a deux types d’attribution : « pour les riverains » et « pour tous ».

C’est lors de cette phase que le riverain donnera le code qu’il a reçu afin de pouvoir réserver l’emplacement devant son habitation.

La réservation des autres emplacements est automatique. On ne peut pas les choisir.

Lors de cette 1ère phase, ce sont uniquement les emplacements non attribués aux riverains qui peuvent être réservés automatiquement.

On attribuera les numéros selon l’ordre croissant.

Par exemple, si les numéros pour les riverains vont de 0 à 45, Ce sera d’abord l’emplacement 46 qui sera attribué automatiquement, ensuite l’emplacement 47 et ainsi de suite.

N’importe qui a le droit de réserver ces autres emplacements, y compris les riverains !

Phase 2

Lors cette phase, tous les emplacements prévus pour les riverains mais non enregistrés seront également automatiquement distribués. Lors de cette phase, on commencera par attribuer les emplacements non confirmés par les riverains. Les emplacements libres sont donc attribués selon l’ordre croissant des numéros.

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 *Brocante* et *GestionBrocante*.

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

L’emplacement sera représenté par son numéro (*int*). La numérotation commence à 0.

Le constructeur de la classe *Brocante* reçoit comme paramètres le nombre total d’emplacements et une table avec les codes pour les emplacements riverains.

Voici un exemple de table de codes:

0 1 2 3

|  |  |  |  |
| --- | --- | --- | --- |
| 0000 | 1234 | 2222 | 4321 |

Il y a 4 emplacements riverains.

Pour réserver l’emplacement 0, le riverain encodera le code 0000, pour l’emplacement 1, 1234 et ainsi de suite.

Pour obtenir directement le numéro de l’emplacement correspondant à un code, un *map* va être créé. A chaque code, lui sera associé son numéro d’emplacement.

C’est le constructeur qui s’occupera de créer ce *map*.

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

Chaque case correspond à un emplacement.

Le constructeur crée cette table.

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.

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

**Lors de la phase 1**, pour optimaliser l’attribution d’un emplacement « pour tous », on retiendra dans un attribut le prochain numéro à attribuer.

Exemple :

Voici une brocante qui propose 10 emplacements dont les 4 premiers sont réservés pour les riverains.

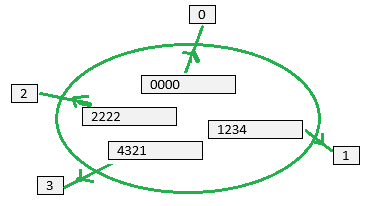
Chacune des 4 habitations a reçu un code (cfr table des codes présentée ci-dessus). A chaque code correspond un numéro d’emplacement.

Pour l’instant seul un riverain a confirmé son emplacement. Nico a dû introduire le code 2222 pour confirmer l’emplacement 2.

3 emplacements « pour tous » ont été réservés automatiquement. Ce sont les emplacements 4, 5 et 6.

Le prochain numéro d’emplacement qui sera automatiquement attribué en phase 1 sera le 7.

mapCodes



tableEmplacements

0 1 2 3 4 5 6 7 8 9

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| *null* | *null* | « nico» | *null* | « sam » | « sam » | *« léa »* | *null* | *null* | *null* |

premEmplacementAuto = 7

Lors de la phase 2, les emplacements « riverains » non enregistrés sont également attribués de façon automatisée.

Pour ce faire, tous les numéros des emplacements encore libres seront placés dans une file.

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

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 0 | 1 | 3 | 7 | 8 | 9 |

fileEmplacementsLibres

Les emplacements seront attribués dans l’ordre de cette file : d’abord 0, puis 1, puis 3 et ainsi de suite.

Pour cette file, 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.

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* et *HashMap* reprises dans le document API\_JAVA.

La classe *GestionBrocante* va vous servir pour tester la classe *Brocante*.

Vous pouvez la modifier.

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