LOG3430 – Méthodes de tests et validation de logiciels

TP1- Codage d'une application

Auteurs

Thierno Barry 1550237
Tomas Costanzo 1578677

Automne 2016
20 Septembre 2016

Dans ce premier laboratoire du cours de LOG430, nous avons developpé un module JAVA qui permet la construction d'une liste chaînée. Techniquement , nous avons developpé des methodes qui permettent de construire la suite chainée en appliquant les operations ensemblistes suivantes : union, intersection, difference, symmetric difference, is subset, is superset; et par la suite fournir des méthodes de gestion de la liste chaînée(comme un ajout d'element ,un retrait ,etc…). Dans la suite du rapport ,nous allons expliquer comment nous avons concu le module dans une section (manuel de conception) et comment l'utiliser dans une section (manuel d'utilisation).

Manuel d'utilisation

Un fichier main a été ajouter dans l'application. Pour utiliser les fonctions et interagir avec le module., suivre les étapes suivantes :

1. Créer 2 ensembles

```
ArrayList<Integer> liste1 = new ArrayList<Integer>();
ArrayList<Integer> liste2 = new ArrayList<Integer>();
```

2. Ajouter des éléments aux ensembles

   Par exemple, ajouter {1,2} au premier ensemble et {2,3} au second

```
liste1.add(1);
liste1.add(3);

liste2.add(2);
liste2.add(3);
```

3. Construire la liste chainée

   Invoquer le constructeur de la liste chaînée :

   Il prend comme premier argument une des opérations listés dans l'introduction, en second le premier ensemble et enfin le second ensemble

   EX : Nous allons appliquer l'opération d'union aux liste1 et liste2

```
SuiteChainee chaineTest = new SuiteChainee("union", liste1, liste2);
```

4. Récupérer la liste chainée

   Après avoir construit la liste chainée elle est accessible via une fonction get().

   Par exemple :

chaineTest.get();

NB : Noter que les méthodes appliquées sur la chaine ne sont pas accessibles directement via chaineTest ; utiliser la méthode get() ou l'autre option est de faire

Chaine chaine = chaineTest.get();

NB : Nous utilisions la première option dans la suite des étapes

5. Afficher le contenu de la liste chaînée
   Une fonction utilitaire qui affiche les ensembles contenues dans la liste
   chaineTest.get().print();

6. Effectuer une opération de gestion sur la liste chaînée
   Plusieurs méthodes sont applicables (voir liste en annexe) pour la gestion de la suite chaînée, mais pour ne pas alourdir le rapport nous allons juste prendre une fonction en exemple, l'utilisation est la même pour les autres

   Exemple : ajouter un élément a l'ensemble add(ensemble)
   Création Dun nouveau ensemble tout d'abord
   *ArrayList<Integer> liste3= **new** ArrayList<Integer>();*
   liste3.add(89);
   liste3.add(85);
   Ensuiite ajout de l'ensemble à la suite
   chaineTest.get().add(liste3);

Dans le souci de faciliter la correction, ou de voir rapidement tous les résultats de chaque méthode du module, nous avons ajouté un fichier seedmain.java, qui lorsqu'on le roule imprime à la console les résultats produits par toutes les méthodes.

Manuel de conception

Pour ce qui en est de la conception, nous avons 2 classes et 1 interfaces :

Interface(s)

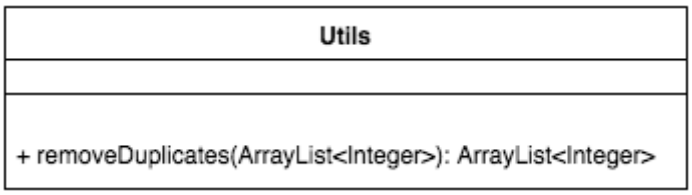
IChaine est une interface, elle possède donc une liste de toutes les fonctions (méthodes de gestion supportés) .

Classe(s)

La classe Chaine implémente l'interface IChaine; par conséquent elle implémente toutes les méthodes de gestion supportés. La chaine est en quelque sorte un tableau qui contient des tableaux.
La classe suiteChainee est celle avec laquelle nous allons interagir. Elle possède comme attribut une liste de type Chaine. Et elle est construite en appliquant une des opérations ensemblistes citées en introduction. Chaque opération ensembliste est représentée par une méthode de la classe et son implémentation est fait via un petit algorithme. Les commentaires dans le code de ces fonctions expliquent clairement l'algorithme utilisé pour chacune d'entre elles.

La figure en annexe représente un diagramme de classe qui décrit chaque classe ou interface et l'interaction entre les différents du composant du module.

Aussi, pour éviter la répétition de code, nous avons ajouté la classe Utils qui possède une seule méthode, cette méthode prend un ArrayList et enlevé tous les valeurs qui sont dupliquées, ensuite il retourne la nouvelle liste.

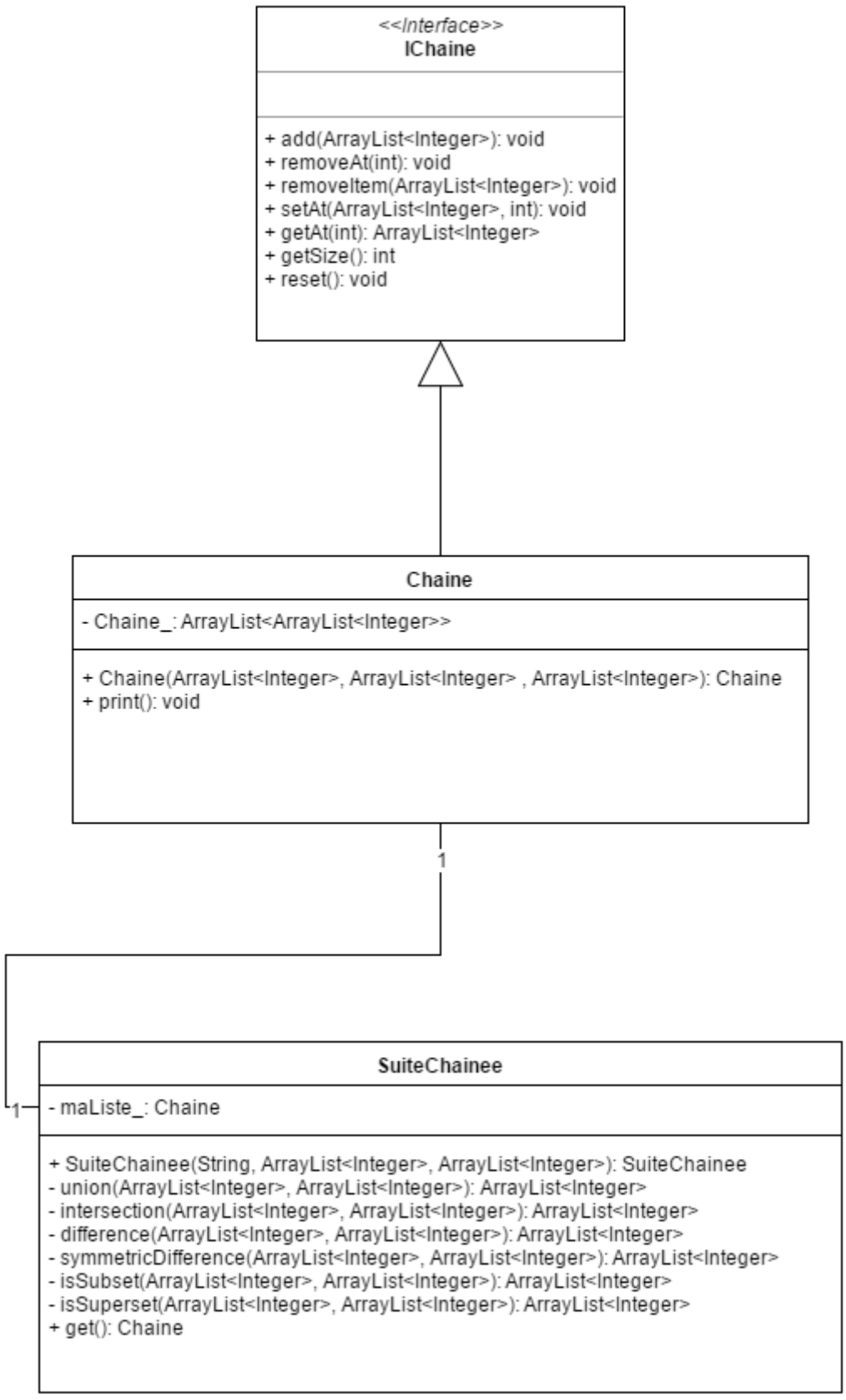


Conclusion

Ce travail pratique nous a permis d'approfondir nos connaissances sur l'utilisation et la manipulation des structures de données en langage JAVA

Annexes

Diagramme de classe


<<Interface>>
IChaine
+ add(ArrayList<Integer>): void
+ removeAt(int): void
+ removeItem(ArrayList<Integer>): void
+ setAt(ArrayList<Integer>, int): void
+ getAt(int): ArrayList<Integer>
+ getSize(): int
+ reset(): void
Chaine
- Chaine_: ArrayList<ArrayList<Integer>>
+ Chaine(ArrayList<Integer>, ArrayList<Integer> , ArrayList<Integer>): Chaine
+ print(): void
1
1
SuiteChainee
- maListe_: Chaine
+ SuiteChainee(String, ArrayList<Integer>, ArrayList<Integer>): SuiteChainee
- union(ArrayList<Integer>, ArrayList<Integer>): ArrayList<Integer>
- intersection(ArrayList<Integer>, ArrayList<Integer>): ArrayList<Integer>
- difference(ArrayList<Integer>, ArrayList<Integer>): ArrayList<Integer>
- symmetricDifference(ArrayList<Integer>, ArrayList<Integer>): ArrayList<Integer>
- isSubset(ArrayList<Integer>, ArrayList<Integer>): ArrayList<Integer>
- isSuperset(ArrayList<Integer>, ArrayList<Integer>): ArrayList<Integer>
+ get(): Chaine

Liste des methodes de gestion de la chaine[i]
add(ensemble) : Ajout d'un ensemble à la chaine.

removeAt(position) : Supprime l'ensemble de position "position" de la chaine
removeItem(ensemble) : Supprime l'ensemble " ensemble " de la chaine

setAt(ensemble, position) : changer le contenu de l'ensemble à la position "position" par "ensemble".

getAt(position) : retourne l'ensemble à la position "position".

getSize() : retourne la longueur de la chaine.

reset() : remet la chaine à vide.

---

[i] Tire de l'ennoncé du laboratoire