SAE 2.01
Rapport de la semaine 3

Sommaire

Sommaire **1**
Préface **2**
Code attendu pour la troisième semaine **2**
Moyen d'illustration des associations 2
Reconnaissance des types d'associations 2
Restructuration des classes **3**
Contexte 3
Production du diagramme de séquence 3
Modification des classes 4
Edition des diagrammes **5**
Objectifs pour la quatrième semaine **6**

Préface

Lors de cette semaine, nous avons décidé de reconstruire complètement toutes les classes du projet suite à une remarque d'Adrien Krähenbühl. Le travail effectué cette semaine a été découpé en deux parties étant le code attendu pour la troisième semaine et la restructuration des classes. Il est donc important de noter que la première partie a été faite avant la restructuration et qu'elle n'est donc pas optimisée.

Code attendu pour la troisième semaine

Moyen d'illustration des associations

La troisième semaine de la SAE demande d'ajouter les associations au DCA que nous avions produit lors de la semaine précédente. Pour cela, on a introduit une nouvelle classe ainsi qu'une nouvelle énumération : 'Association' et 'TypeAssociation'. La classe 'Association' contient trois variables contenant tout ce dont une association a besoin, c'est-à-dire les deux éléments à lier ainsi que le type d'association (agrégation, héritage, …). Les deux éléments sont représentés par une instance de la classe 'ClassContent' et le type d'association est une instance de l'énumération 'TypeAssociation'.

Ensuite, lors de la génération du diagramme UML, le nom de la première classe est sorti, suivi de son type d'association retranscrit en une flèche puis le nom de la seconde case.

```
8 usages  TOUPENSE DIMITRI
public class Association {
    3 usages
    private ClassContent element1;
    3 usages
    private ClassContent element2;
    4 usages
    private TypeAssociation typeAssociation = TypeAssociation.SIMPLE;

    no usages  TOUPENSE DIMITRI
    public Association(ClassContent element1, ClassContent element2) {
        this.element1 = element1;
        this.element2 = element2;
    }

    3 usages  TOUPENSE DIMITRI
    public Association(ClassContent element1, ClassContent element2, TypeAssociation typeAssociation) {
        this.element1 = element1;
        this.element2 = element2;
        this.typeAssociation = typeAssociation;
    }
```

Base de la classe 'Association'

Reconnaissance des types d'associations

Pour le DCA, seulement trois types d'associations sont utilisés qui sont les associations simples, ceux qui impliquent un héritage ou ceux impliquant une interface. On a commencé par les associations simples et nous avons simplement regardé pour chacun des

attributs de chaque classe si il correspond ou non à l'une des classes du package. Si un lien a été fait entre deux classes, une nouvelle instance d''Association' est alors créée en prenant en paramètre les deux instances de ClassContent et le type d'association.

Pour reconnaître un héritage, on a en premier regardé si l'élément traité est bien une classe, puis ensuite on l'a converti en un 'TypeElement' pour appeler sa super-classe avec la méthode .getSuperclass(). On a donc comparé pour chaque élément si un autre élément du package correspondait ou non à sa super-classe. On a ensuite utilisé la même méthode d'ajout d'association que pour l'association simple.

Finalement, pour reconnaître les interfaces, on a récupéré toutes les interfaces utilisées par chaque classe et on a regardé si elle correspondait à un élément du package, on a ensuite ajouté l'association comme pour les deux autres associations.

Restructuration des classes

Contexte

Avant le remaniement de toutes les classes, la classe PumlDoclet contenait la grande majorité du code. C'est-à-dire que, par exemple, les classes que nous avions créées comme "ClassContent", "Methode", "Association", "Attribut", "TypeAssociation" et "Visibilite" pour gérer les éléments du package n'étaient que des coquilles vides, sans réelle utilité, contenant juste des attributs qui ne sont pas private et qui ne peuvent pas être gérés par la classe elle-même mais qui sont gérés par PumlDoclet. C'est pourquoi nous avons décidé, sous les conseils de M. Krähenbühl, de refaire l'ensemble des classes après avoir fini le code de la semaine.

Production du diagramme de séquence

Avant de commencer la restructuration, nous avons produit un diagramme de séquence contenant le fonctionnement du doclet. Nous avons découpé le fonctionnement de PumlDiagram en trois parties pour chercher les classes, les liaisons et générer le diagramme que nous avons mis en trois méthodes. Pour la suite de la production du diagramme de séquence, nous avons essayé de faire de la même façon en gardant les fonctionnalités que l'on avait auparavant.

La production du diagramme de séquence nous a permis de mieux représenter ce dont à quoi le programme allait ressembler. On a par la suite ajouté d'autres méthodes pour accéder à certains attributs mais elles n'apparaissent pas dans le diagramme de séquence.

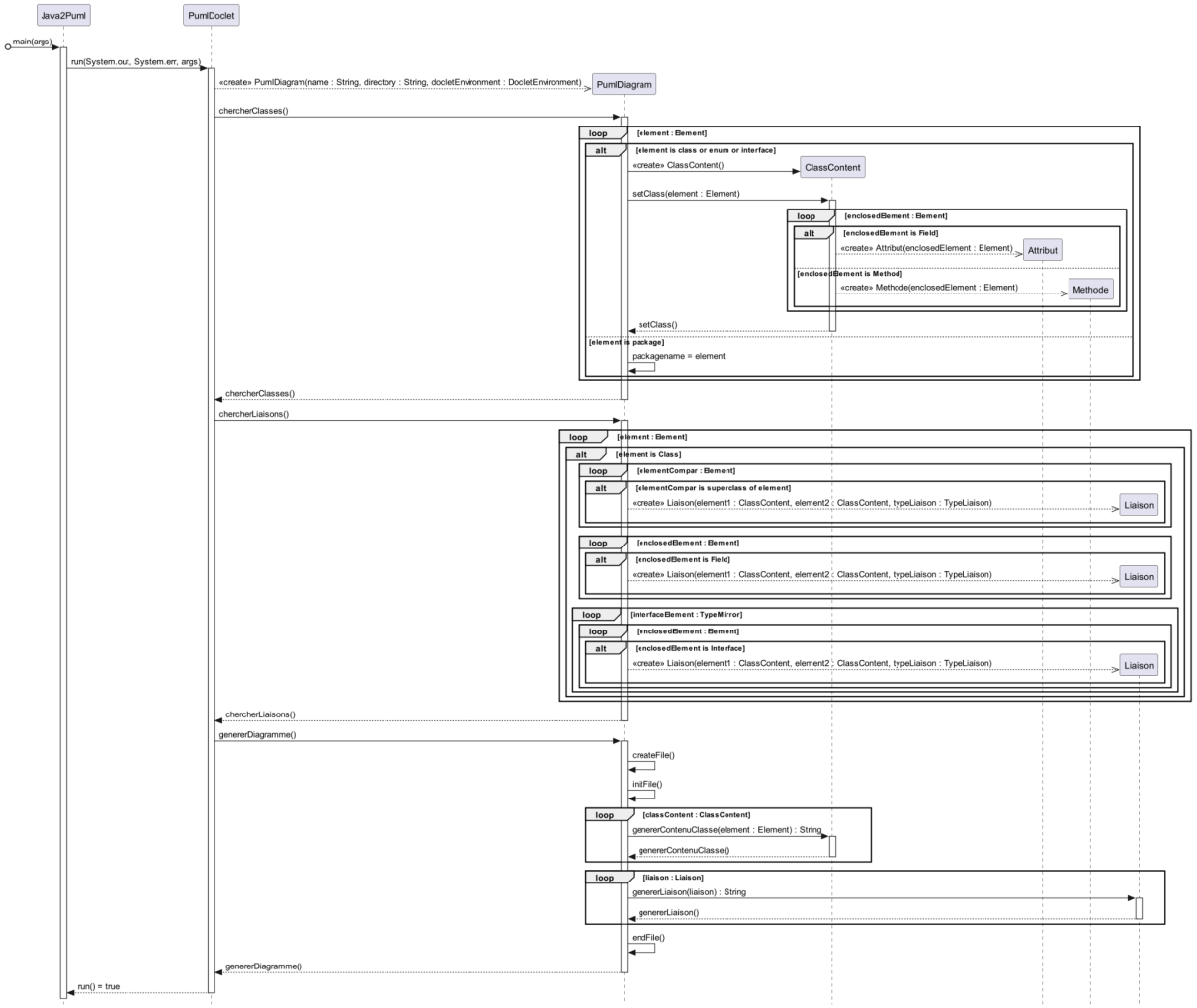


Diagramme de séquence représentant notre objectif

Modification des classes

L'une des plus grosses modifications est que maintenant, les deux classes 'ClassContent' et 'Association' gèrent elles-mêmes leur partie de la génération du diagramme Plant UML.

Dans la classe PumlDiagram, nous avons donc factorisé tout le code concernant la production du diagramme de classe du package Western.

Une méthode "chercherAssociations()" nous permet donc, grâce au package '*javax.lang.model.element*', de pouvoir voir s'il s'agit d'associations simples, d'héritages ou d'implements (ce qui a été expliqué plus tôt).

Les méthodes "chercherClasses()" et "genererDiagramme()" ont été aussi créées pour alléger le code de PumlDoclet. Les méthodes "createFile()", "initFile()" et "endFile" ont été déplacées dans PumlDiagram avec quelques modifications.

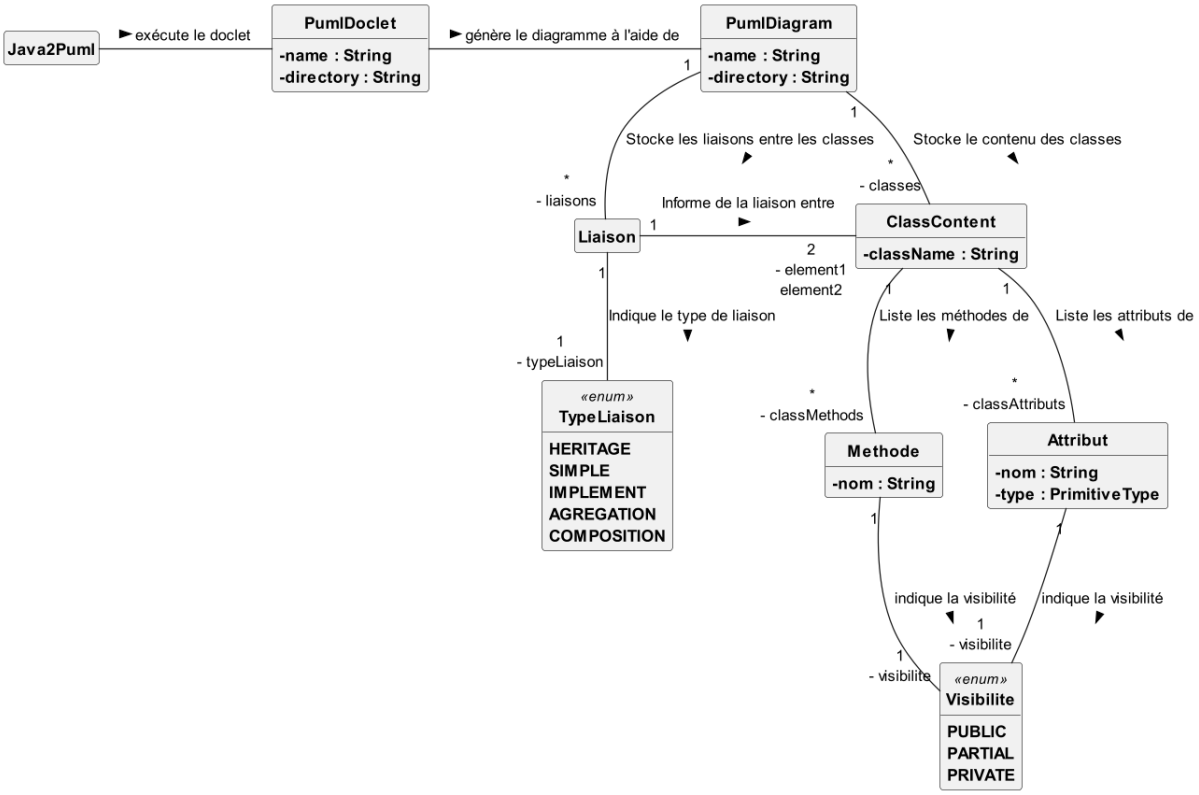
Dans la classe ClassContent, un constructeur a été ajouté, ainsi que plusieurs méthodes pour pouvoir manipuler les attributs de la classe. L'objectif de ces méthodes est essentiellement de récupérer les attributs, leur type et les méthodes pour chaque classe contenue dans le package voulu. Elle permet aussi d'écrire la plupart du contenu du fichier

puml créé par la classe PumlDiagram et donc d'alléger le code de PumlDiagram et de PumlDoclet.

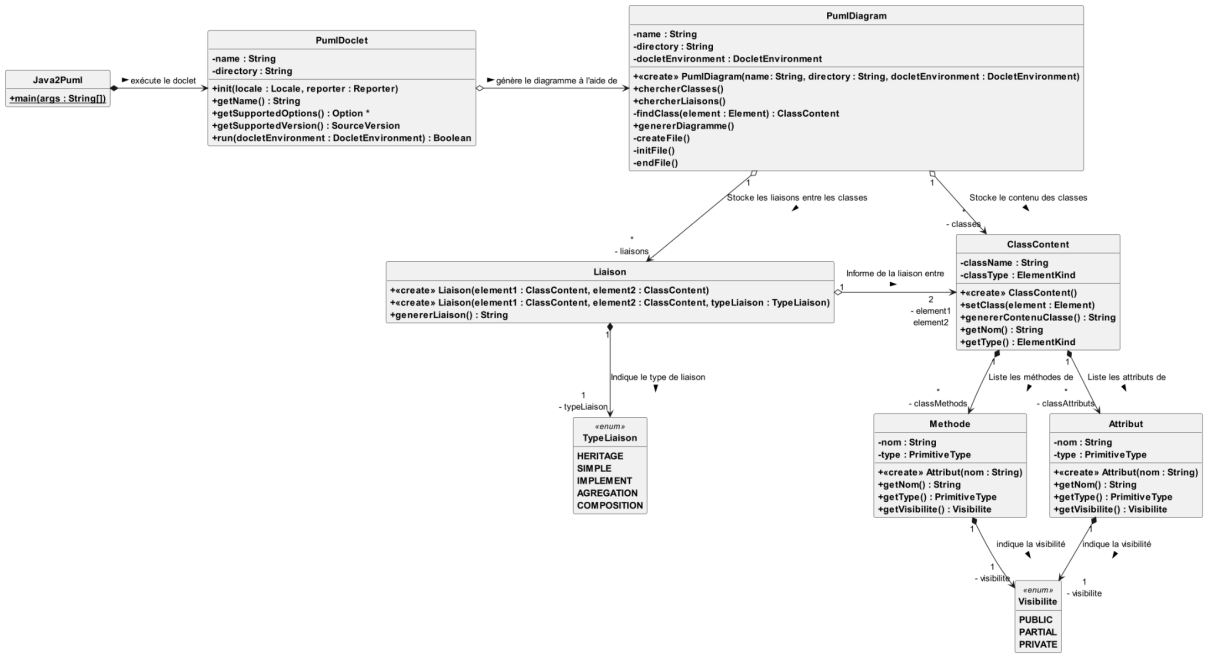
Des méthodes ont été ajoutées dans les classes Attribut et Association pour gérer leurs attributs. En plus de cela, dans cette dernière, la méthode "genererAssociation()" permet d'écrire le code puml relatif aux relations comme le "-" entre deux éléments.

Edition des diagrammes

Suite au travail que l'on a fait cette semaine sur le projet, on a refait notre DCA et notre DCC. Nous avons tenu compte des conseils que l'on a pu nous donner sur les associations sur le DCA.



DCA du package après la restructuration



DCC du package après la restructuration

Objectifs pour la quatrième semaine

Pour la quatrième semaine, en plus du travail demandé, nous nous sommes fixé comme objectif d'utiliser tout le potentiel que git possède en utilisant correctement les branches, c'est-à-dire de directement travailler sur les branches des semaines correspondantes et de faire un merge vers la branche main lorsque la semaine est finie sans supprimer la branche de la semaine. Tout comme pour ce qui a été fait pour la branche 'Restructuration', il est possible que nous commencions à faire de nouvelles branches pour des fonctionnalités.