# SAE 2.01
# Rapport de la semaine 5

## Sommaire

# Modifications des classes existantes

Le travail de la cinquième semaine consiste globalement à générer les associations entre les classes du diagramme. Comme pour la troisième semaine on devait aussi utiliser des associations entre les classes, on a décidé de mettre à jour le fonctionnement de la classe Association.

En premier, le travail de la semaine demande de gérer les multiplicités ainsi que les rôles des associations. Pour cela, la classe Association s'est vu ajouter deux attributs mult1 et mult2 contenant une chaîne de caractère correspondant à la multiplicité de l'une des extrémités de l'association.

Pour gérer les rôles, mis à part pour les dépendances, seulement les agrégations et les compositions ont besoin d'un rôle personnalisé. Cette partie a été gérée avec l'ajout de l'attribut attributLié qui va aider pour afficher le texte du rôle et obtenir sa visibilité.

De plus, pour éviter les doublons dans les associations, un booléen isPumlVisible a été ajouté. Elle aide à définir si l'association doit être affichée dans le diagramme ou non.

En plus de ces attributs, des méthodes get/set ont été ajoutées pour manipuler les instances de la classe en toute sécurité.

```
1 usage  TOUPENSE DIMITRI
public void setMult1(String mult1) { this.mult1 = mult1; }
2 usages  TOUPENSE DIMITRI
public void setMult2(String mult2) { this.mult2 = mult2; }
1 usage  TOUPENSE DIMITRI
public void setAttributLié(Attribut attribut){this.attributLié = attribut;}
1 usage  TOUPENSE DIMITRI
public boolean getPumlVisibilite(){return this.isPumlVisible;}
1 usage  TOUPENSE DIMITRI
public void setToInvisible(){this.isPumlVisible = false;}
1 usage  new *
public void IncrementationMult(){
    if(!this.mult2.equals("*"))
        this.mult2 = Integer.toString( i: Integer.valueOf(this.mult2)+1);
}
no usages  TOUPENSE DIMITRI
public void setType(TypeAssociation typeAssociation){
    this.typeAssociation = typeAssociation;
}
```

Ensemble des méthodes ajoutées dans Association.java

# Génération des associations

Même si le travail que l'on a fait lors de la troisième semaine était toujours fonctionnel dans l'ensemble, nous avons préféré complètement le refaire pour qu'il corresponde mieux à la structure actuelle du projet et pour mieux intégrer les modifications de la classe Association.

On a programmé la reconnaissance des agrégations/compositions, des interfaces, des héritages ainsi que la reconnaissance des dépendances. Nous avons décidé de regrouper les agrégations et compositions ensemble parce qu'elles sont assez similaires et que nous préférons tout de même afficher les compositions d'une certaine manière même si elles sont compliquées à prouver.

La reconnaissance des agrégations/compositions s'est seulement faite en regardant les types des attributs des instances de ClassContent. On a comparé le type des attributs avec le nom de chaque élément du package et nous avons gardé seulement celles qui correspondent.

Les associations d'héritages se sont faites en regardant si le type de l'élément traité a une super classe ou non. On a juste vérifié par la suite que le super élément n'était pas nul et qu'il n'est pas le même élément que celui que l'on traite.

On a ensuite regardé pour les interfaces utilisées. Pour cela, on a converti l'instance d'ElementContent traité en son équivalent de type Element puis on a regardé s'il a des interfaces avec la méthode getInterfaces(). On a ensuite ajouté l'association entre l'élément et l'interface correspondante.

La dernière association possible est la dépendance, elle apparaît seulement si aucune autre association n'est possible. Pour cela, on a traité les méthodes des éléments qui peuvent en contenir, c'est-à-dire les classes et les interfaces. On a alors inspecté le type des méthodes puis celui des paramètres.

Rapidement, un problème s'est posé puisqu'il peut y avoir de nombreuses dépendances entre deux mêmes éléments. On a alors ajouté une méthode dans PumlDiagram qui s'appelle ajoutAssociation(associationCanditate : Association) qui va vérifier que l'association n'est pas la même et qu'il n'y a pas deux fois une dépendance entre deux éléments.

```
1 usage   TOUPENSE DIMITRI +1 *
public void chercherAssociations() {
    //Recherche des agrégations
    for (ElementContent elementContent : elements) {
        Element element = findElementFromElementContent(elementContent);

        //Recherche d'agrégation/composition
        if (elementContent.classType == ElementKind.CLASS) {
            ClassContent classContent = (ClassContent) elementContent;
            for(Attribut attribut : classContent.getAttributs()) {
                if(!attribut.getType().toString().equals("void")){
                    for(ElementContent elementContentCompar : elements){
                        if(SubstringType(attribut.getType().toString()).equals(elementContentCompar.className)) {
                            attribut.setToInvisible();
                            Association associationAgreg = new Association(elementContent, elementContentCompar, TypeAssociation.AGREGATION);
                            associationAgreg.setAttributLié(attribut);
                            associationAgreg.setMult1("1");
                            if(attribut.getType().toString().contains("java.util"))
                                associationAgreg.setMult2("*");
                            else
                                associationAgreg.setMult2("1");
                            this.ajoutAssociation(associationAgreg);
                        }
                    }
                }
            }
        }
    }
```

Début de la méthode de recherche des associations

# Filtrage des associations pour la génération

La génération des associations se fait dans la classe Association et est appelé dans GenerateurDiagramme avec la méthode generateLinksForPuml(associations : Association *). Pour cela, on a écrit dans le fichier du paramètre d'exécution -out pour chaque association en générant la chaîne de caractère correspondant à l'association.

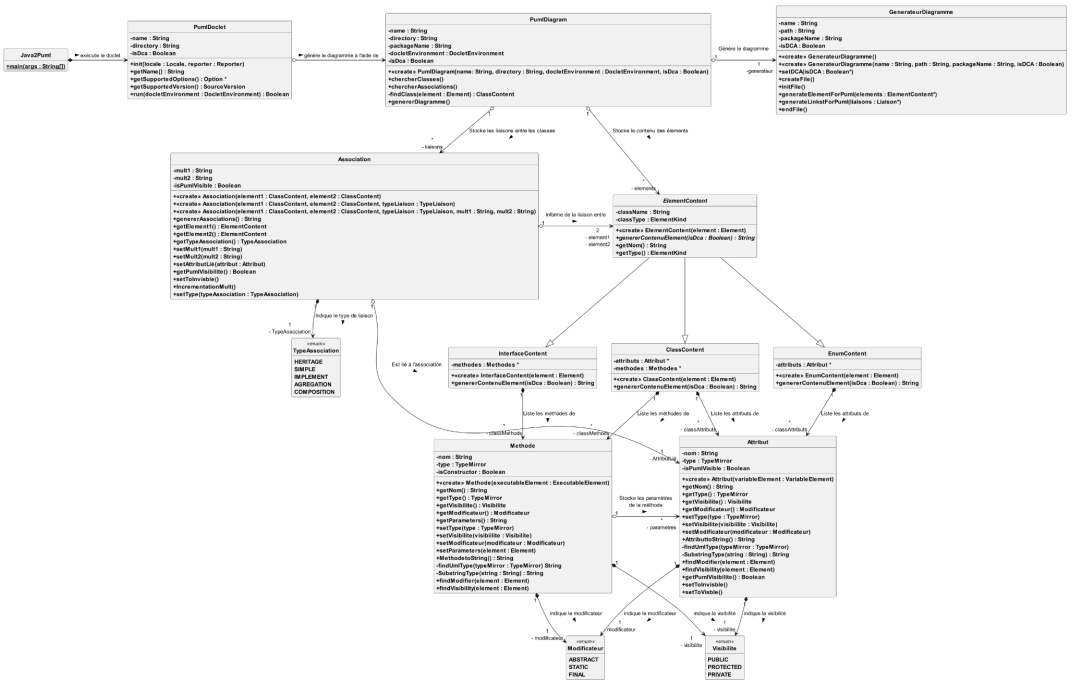
Les instances de la classe Association va donc traiter ses propres informations pour former la chaîne de caractère voulu. Il va par exemple traiter les types de flèches à créer en regardant le type d'association ou encore générer ou non les rôles si l'association en a.

```java
1 usage  TOUPENSE DIMITRI *
public String genererAssociation(boolean isDca) {
    if(this.isPumlVisible) {
        String associationString = "";

        if (isDca || !isDca && typeAssociation != TypeAssociation.DEPENDANCE)
            associationString = "\n" + element1.getNom() + " ";

        //Ajout de la multiplicité du premier élément
        if (mult1 != null)
            associationString += " \"" + mult1 + "\" ";

        //Ajout du début de la flèche selon le type d'association
        if (!isDca) {
            if (typeAssociation == TypeAssociation.AGREGATION)
                associationString += "o";
            else if (typeAssociation == TypeAssociation.COMPOSITION)
                associationString += "*";
        }

        //Ajout du corps de la flèche
        if (isDca || !isDca && typeAssociation != TypeAssociation.DEPENDANCE) {
            for (int i = 0; i < 2; i++) {
                if ((i & 2) == 0) {
                    if (typeAssociation == TypeAssociation.IMPLEMENT)
                        associationString += ".";
                    else
                        associationString += "-";
                }
            }
        }
```

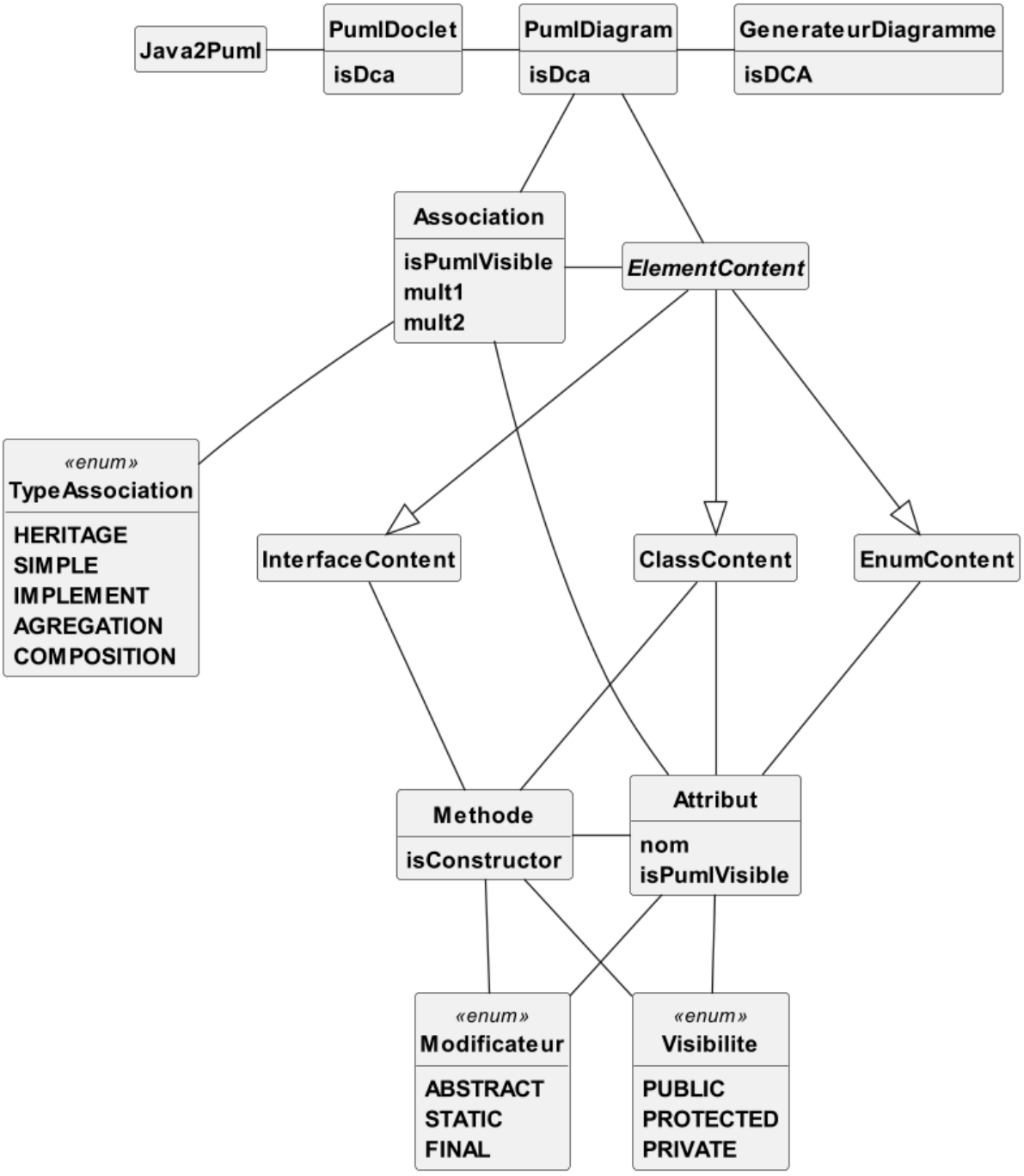
Début de la méthode de génération des associations dans Association.java

# Mise à jour des DCA et DCC

Les diagrammes de classes du package ont été mis à jour en suivant les mêmes méthodes que les semaines précédentes.

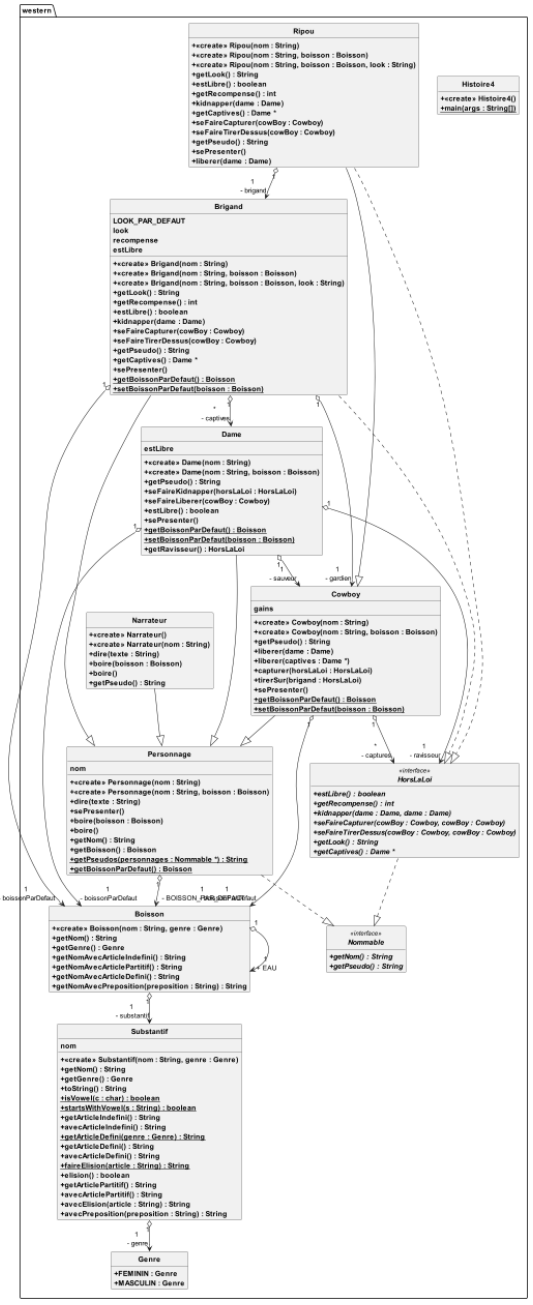



DCC du package PumlFromJava

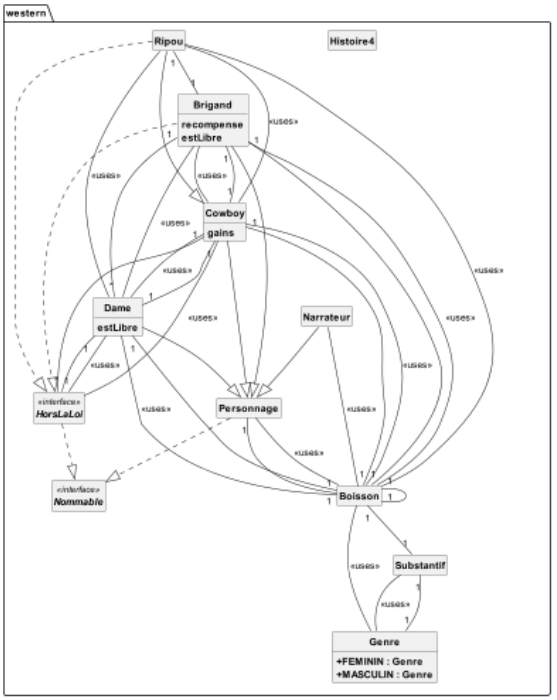



DCC du package PumlFromJava

# Diagrammes de classes générés




DCC produit sur le package western




DCA produit sur le package western

# Objectifs pour la dernière semaine

Pour la dernière semaine, nous ferons en plus du travail demandé, une méthode pour mieux organiser les classes dans le diagramme produit pour qu'il soit plus agréable visuellement. Nous réglerons aussi l'affichage des méthodes qui sont déjà appelés dans leur super-classe.