SAE 2.01
Rapport de la semaine 4

Sommaire

Ajout de l'option -dca

Le travail de cette semaine demandait à ajouter une option pour l'exécution du programme qui permettrait de générer un DCA lorsque l'option est présente ou de générer un DCC dans le cas contraire.

Pour gérer cette option, dans le fichier PumlDoclet, un booléen isDca a été ajouté et il est passé en paramètre au constructeur de la classe PumlDiagram qui conservera l'information. A partir de ce moment-là, nous avons décidé de laisser fonctionner l'instance de PumlDiagram pleinement en récupérant tous les attributs, méthodes et associations pour qu'ensuite il fasse le tri lors de l'écriture dans le fichier uml selon le type de diagramme voulu.

```
2 usages   Dimitri.T +1
public PumlDiagram(String name, String directory, DocletEnvironment docletEnvironment, boolean isDCA){
    this.name = name;
    this.directory = directory;
    this.docletEnvironment = docletEnvironment;
    this.isDca = isDCA;
    System.out.println("isDca = " + isDCA);
}
```

Constructeur de la classe PumlDiagram

Ajout de la classe GenerateurDiagramme

L'une des nouvelles classes ajoutées est GenerateurDiagramme. Comme son nom l'indique, elle s'occupera de générer le fichier uml. Pour cela, la classe stocke le nom du fichier à créer, le chemin pour y accéder, le nom du package à traiter ainsi que le type de diagramme à produire.

La classe fonctionne de la même façon que lors de la semaine dernière, elle a donc cinq méthodes pour la génération du diagramme qui sont createFile(), initFile(), generateElementsForPuml(elements : Element*), generateLinksFromPuml(associations : Association*) et endFile().

La classe va donc générer le diagramme par plusieurs étapes et les deux méthodes generateElementsForPuml() et generateLinksForPuml() sont les deux seules méthodes à prendre compte le type de diagramme à créer. Comme pour la quatrième semaine aucune association entre les classes n'était attendue, aucune modification n'a été faite du côté de la classe Association.

Séparation de la classe ClassContent

Jusqu'à présent, tous les éléments des packages étaient stockés dans la classe ClassContent, ce qui n'était pas cohérent par rapport au nom puisqu'il y a aussi des énumérations et les interfaces mais aussi parce qu'avec la complexité que prend le projet, les éléments demandent des traitements différents. Pour régler ce problème, une nouvelle classe abstraite ElementContent a été ajoutée ainsi que ses trois classes-enfants EnumContent, InterfaceContent et ClassContent. La classe abstraite reprend les bases de

l'ancienne ClassContent mais ne garde que ce qui est universel pour chaque élément, c'est-à-dire son nom, son type et sa méthode de génération de Contenu. En plus de cela, d'autres méthodes ont été ajoutées mais ne sont pas abstraites, elles retournent le nom ou le type de l'élément.

Chaque classe-enfant d'ElementContent contient une ou deux listes correspondant au type de données qu'il doit stocker : la classe InterfaceContent stocke une liste de méthodes, la classe ClassContent stocke une liste d'attributs et une autre pour les méthodes et EnumContent stocke une liste d'attributs (puisque les instances d'une énumération sont considérés comme des attributs). Comme lors des précédentes semaines, les classes-enfants vont générer leur partie du diagramme avec la méthode genererContenuElement(isDca : Boolean).

Implémentation de la classe Méthode

Même si la classe est présente depuis la deuxième semaine et qu'elle y est déjà implémentée mais volontairement bloquée pour l'utiliser plus tard, nous allons tout de même la présenter. Cette classe est similaire à la classe Attribut, elle stocke un nom, un type de retour et d'autres. Elle a en plus une liste d'attributs qui correspondent à ses paramètres.

Tout comme la classe Attribut, elle peut générer sa partie du diagramme en affichant sa visibilité, ses paramètres, son modificateur et aussi affiché si c'est un constructeur.

La façon dont la classe est instancié se fait lorsqu'un élément est construit. En premier l'élément va chercher les attributs en regardant son ElementKind et il en fera de même pour les méthodes.

Une instance de la classe Methode peut aussi éventuellement être un constructeur, pour cela, un booléen a été créé pour stocker l'information et reconnaître si la méthode est un constructeur si son ElementKind est de type *CONSTRUCTOR*. L'un des problèmes est que le nom de la classe n'est pas celui de son élément, pour résoudre ce problème, l'élément ayant ce constructeur utilise la méthode setName(name : String) qui redéfinit le nom de l'instance.

```
2 usages   TOUPENSE DIMITRI
public Methode(ExecutableElement executableElement){
    this.nom = executableElement.getSimpleName().toString();
    this.type = executableElement.getReturnType();
    this.findModifier(executableElement);
    this.findVisibility(executableElement);
    this.setParameters(executableElement);
    if (executableElement.getKind() == ElementKind.CONSTRUCTOR)
        isConstructor = true;
}
```

Constructeur de la classe Methode

Traitement des informations des attributs et des méthodes

L'un des principes du travail de la semaine a été de gérer les méthodes ainsi que les informations supplémentaires comme les modificateur, les visibilités ainsi que les multiplicités.

Modificateurs et visibilités

Les attributs et méthodes peuvent avoir des modificateurs comme par exemple 'static', 'abstract' ou 'final'. Nous avons décidé de représenter ces modificateurs grâce à une énumération nommée Modificateur. Chaque instance de la classe Methode ou Attribut ne peut en avoir qu'un seul, pour cela un attribut 'modificateur' leur ont été ajouté. Cela leur permet donc d'être différencié lors de la génération du diagramme et d'afficher leur modificateur.

Les visibilités ont aussi le même traitement, ils ont aussi trois valeurs : 'public', 'protected' et 'private'. Ils ont donc eu le droit à une énumération nommée Visibilité et chaque instance de Methode et Attribut ont une instance de cette énumération et aide aussi à les différencier pour la génération.

Pour trouver les modificateurs et les visibilités, une méthode findVisibility(element : Element) et findModifier(element : Element) ont été ajoutées. Elles regardent à l'aide d'une boucle sur les modifcateurs avec .getModifier() ainsi qu'avec un tri si les modificateurs correspondent à l'une des six instances des deux énumérations.

Multiplicités

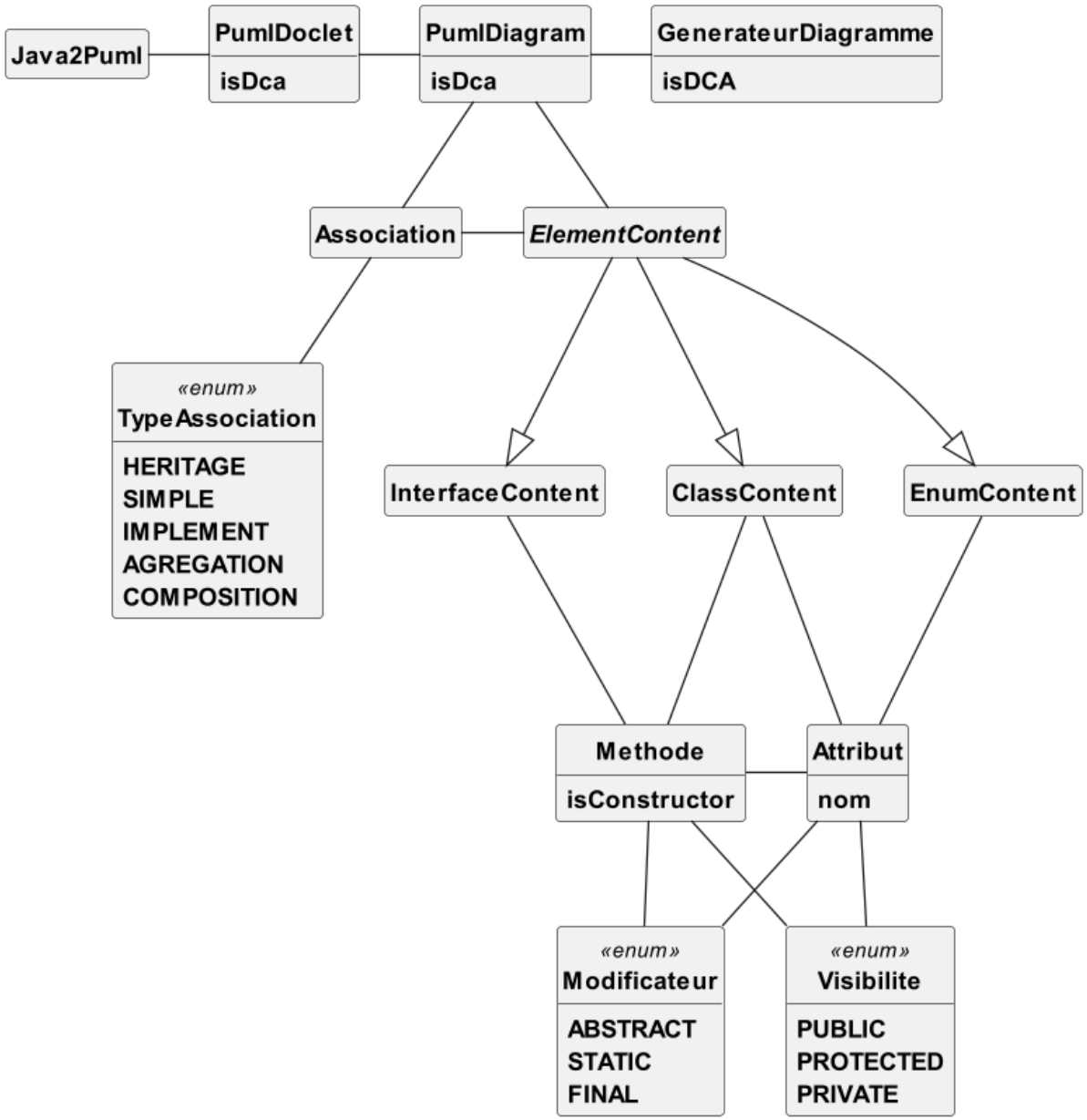
Les multiplicités dans les diagrammes Plant Uml représentent l'ensemble des types stockant plusieurs instances en son sein et sont marqués par le type de variable qu'il stocke suivi d'une étoile.

Pour trouver les attributs et méthodes rentrant dans cette catégorie, il était attendu d'utiliser l'énumération TypeKind de la classe TypeMirror sauf que nous n'avons pas réussi à obtenir un résultat stable. On a alors préféré regarder si le type contenait la chaîne de caractères "java.util" puisque c'est le package contenant l'ensemble des classes rentrant dans le principe de multiplicité. Le problème est qu'il existe d'autres classes dans ce package qui ne rentrent pas dans ce principe, ce qui veut dire que si le package à traiter contient l'une de ces classes, elle serait considéré comme multiple.

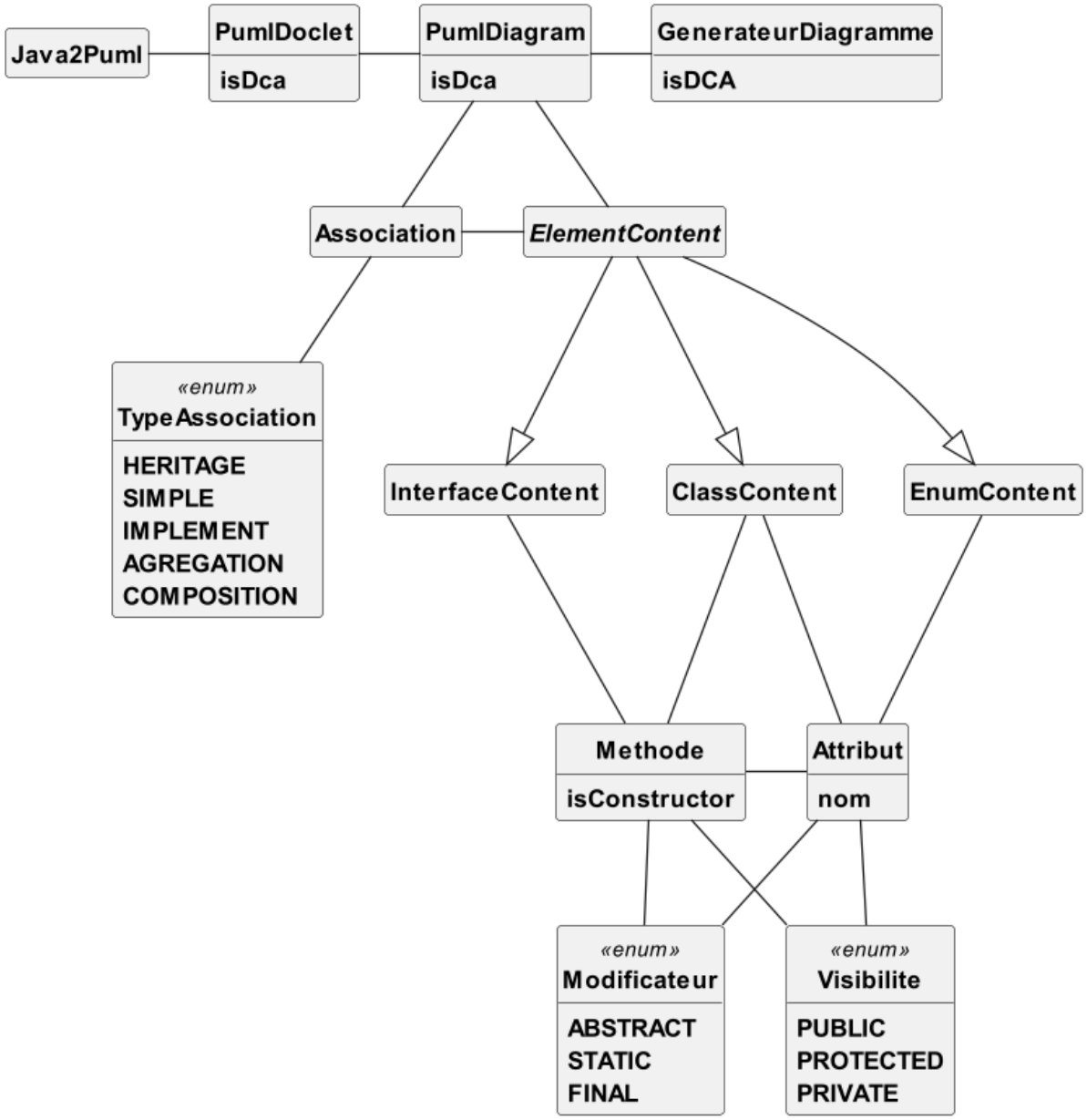
L'un des problèmes que l'on a rencontré avec les multiplicités et les types en général, c'est leur retour de la méthode toString(). En effet, elles ne retournent pas seulement le nom du type mais aussi le nom de leur package. Pour cela, nous avons créé une méthode SubstringType(string : String) qui ne garde que le nom du type.

Mise à jour des diagrammes de classe

Les diagrammes de classes du package ont été mis à jour et des corrections ont été faites par rapport au nom des classes, attributs et méthodes contenant le nom d'association.

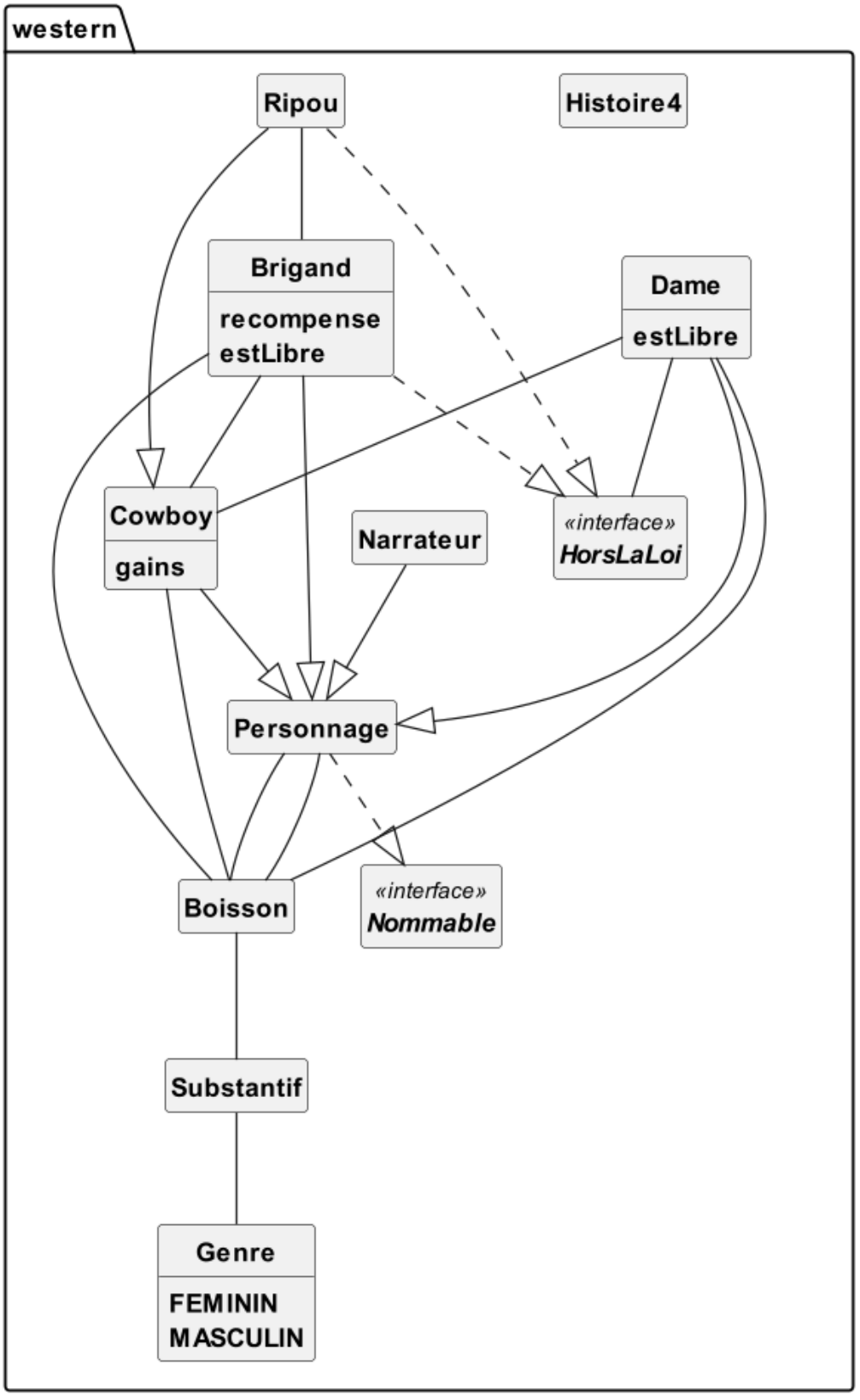


DCA du package pumlFromJava

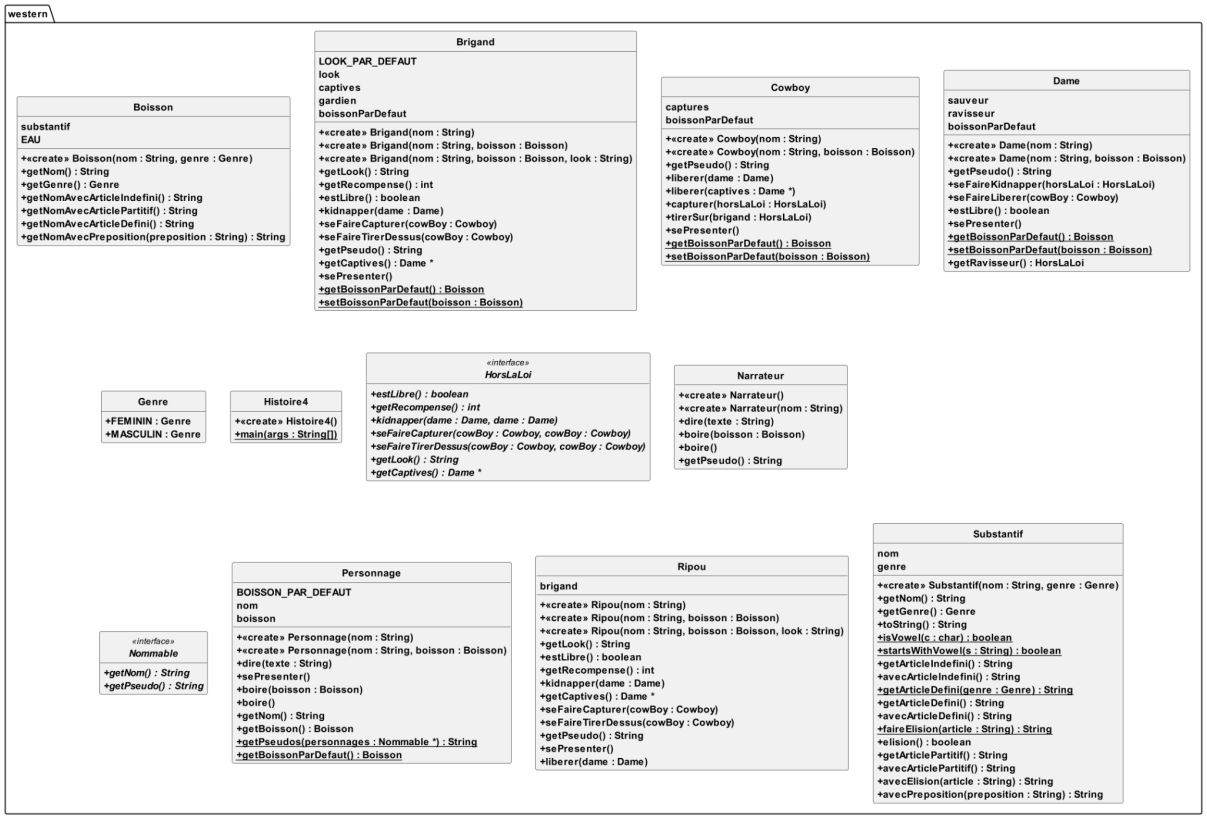


DCA du package pumlFromJava

Résultat de la génération des diagrammes



DCA produit



DCC produit

Objectifs pour la semaine prochaine

Notre objectif pour la semaine sera de faire le travail attendu et en plus de corriger le problème de la reconnaissance des types multiples. Nous réfléchirons aussi à ce que nous ferons des classes Attribut et Méthode à cause de leur ressemblance.