



## M2 INFORMATIQUE AIGLE

**HMIN306**  
ÉVOLUTION ET RESTRUCTURATION

---

# Rapports de TPs

(TPs 1, 2, 3 et 4)

---

Bachar Rima,  
Amandine Paillard

17 janvier 2020

# Table des matières

<b>1 TP1 : Généralités</b>	<b>3</b>
1.1 Exercice 1 : Compréhension des concepts liés à l'évolution / maintenance des logiciels	3
1.2 Exercice 2 : Outils de maintenance / évolution . . . . .	5
1.2.1 Checkstyle . . . . .	5
1.2.2 CodeCity : Introduction à la métaphore « <i>Software City</i> » . . . . .	18
1.2.3 JSCity : Une implémentation orientée-JavaScript de la métaphore <i>Software City</i> . . . . .	23
1.3 Exercice 3 : Analyse d'approches en maintenance et évolution logicielle . . . . .	28
<b>2 TP2 - Analyse statique et dynamique</b>	<b>31</b>
2.1 Parties 1 et 2 - AST d'Eclipse JDT . . . . .	31
2.1.1 Le modèle Java ( <i>Java Model</i> ) du JDT . . . . .	31
2.1.2 L'AST du JDT . . . . .	32
2.1.3 Les processeurs utilisant l'AST du JDT . . . . .	37
2.1.4 Le graphe d'appel statique . . . . .	38
2.2 Partie 3 - Étude de l'outil Spoon . . . . .	39
2.2.1 Introduction . . . . .	39
2.2.2 Installation en tant qu'un <i>plugin</i> Maven . . . . .	39
2.2.3 Le métamodèle de Spoon . . . . .	40
2.2.4 Les références . . . . .	41
2.2.5 Le processus standard d'utilisation de Spoon . . . . .	42
2.2.6 Afficher le modèle Spoon d'un code source Java . . . . .	42
2.2.7 Les <i>factories</i> . . . . .	42
2.2.8 Les <i>getters/setters</i> standards . . . . .	43
2.2.9 Les filtres . . . . .	43

2.2.10	Les <i>queries</i>	43
2.2.11	Les processeurs	45
2.2.12	Les <i>launchers</i>	46
2.2.13	Instrumentation de code et traces d'appels	47
2.2.14	Le graphe d'appel dynamique	50
<b>3</b>	<b>TP3 - Compréhension de programmes</b>	<b>53</b>
3.1	Exercice 1 : Graphe de couplage entre classes	53
3.2	Exercice 2 : Identification de Modules	54
3.2.1	Construction du <i>cluster</i> hiérarchique	54
3.2.2	Partitionnement	55
3.3	Exercice 3 : Spoon pour Identification de Modules	56
<b>4</b>	<b>TP4 - Réingénierie des logiciels</b>	<b>57</b>
4.1	Exercice 1 - Extraction de la variabilité	57
4.2	Exercice 2 - Transformation du code source pour supprimer les dépendances OO	58

# Chapitre 1

## TP1 : Généralités

### 1.1 Exercice 1 : Compréhension des concepts liés à l'évolution / maintenance des logiciels

Cet exercice consiste à mettre en relation différents concepts dans un modèle UML. Les concepts entrants en jeu sont liés au domaine de l'évolution et de la maintenance logicielle et sont nombreux. En voici quelques uns :

**Maintenance** : « *la modification d'un logiciel, après sa livraison, afin de corriger des défaillances, d'améliorer sa performance ou d'autres attributs ou de l'adapter suite à des changements d'environnements* » IEEE 610.12, 1993. L'activité qui regroupe à la fois l'ajout de fonctionnalités à un logiciel, son adaptation aux changements de contexte et aux nouvelles technologies, l'amélioration de ses performances, la correction des erreurs.

**Évolution** : les « *changements apportés à un logiciel pour s'adapter à un nouvel environnement et/ou besoin.* » (IEEE, DOI : 10.1109/ICCASM.2010.5620014, <https://ieeexplore.ieee.org/document/5620014>).

**Réingénierie** : il s'agit de la restructuration ou réécriture de tout ou partie d'un système sans changer sa fonctionnalité.

**Rétro ingénierie** : analyser un programme sans spécifications originales pour le comprendre à un niveau supérieur d'abstraction. D'après l'IEEE, cette notion est la « *création de modèles ou autres documentations à partir du code* » (DOI : 10.1109/ITNG.2010.189, <https://ieeexplore.ieee.org/document/5501482>).

**Adaptation** : réaliser les différents changements à faire pour que le logiciel subsiste dans son environnement. Il s'agit principalement de changements technologiques pour que le produit ne soit pas dépassé ou non compatible.

**Analyse statique** : analyser un logiciel sans l'exécuter pour en prévenir les comportements non désirables. Permet, entre autre, de trouver des erreurs de programmation, de conception et une baisse de qualité du code.

**Analyse dynamique** : vérifier que l'exécution du logiciel se déroule normalement, selon ses spécifications. Cela permet également de trouver des problèmes liés à l'utilisation de la mémoire ou du processeur.

**Dette technique** : c'est le retard pris dans la programmation d'un projet par manque de respect de la conception et des normes définies. Il faudra rattraper (rembourser) ce retard tout le long de la durée de vie du projet.

**Extraction d'architecture** : processus visant à analyser un logiciel pour en déduire son architecture.

**Extraction de modèles** : processus visant à analyser un logiciel pour en extraire un modèle le renseignant. Cette technique est utilisée notamment pour l'ingénierie dirigée par les modèles.

**Slicing** : processus de simplification d'un programme à l'aide d'un critère (une ou plusieurs lignes de code du programme initial par exemple). Cela est utilisé notamment pour le débogage.

**Refactoring** : processus de restructuration du code source d'un logiciel existant, sans changer son comportement externe (donc y ajouter des fonctionnalités ou corriger les éventuelles bogues) Son objectif est d'améliorer la lisibilité d'un code et le rendre plus générique.

**Migration** : faire la transition d'un logiciel d'une plateforme vers une autre.

**Restructuration** : une forme de la maintenance perfective d'un logiciel, transformant le système d'une représentation à une autre en conservant, entre les deux, le même comportement et un même niveau d'abstraction. Ces objectifs sont l'amélioration de la maintenabilité afin de faciliter certaines activités en faisant partie (l'ajout de nouvelles fonctionnalités, la correction de bogues non détectées préalablement).

**Correction de bugs** : processus de détection et de correction des déficiences d'un logiciel empêchant son comportement correct. Son objectif est d'assurer la cohérence entre les spécifications du logiciel et son implantation.

**Impact des changements** : décrire comment mener, à un coût efficace, une analyse complète des impacts d'un changement dans un logiciel en analysant sa structure et son contenu. Les principaux objectifs liés à ce concept sont :

- détermination de la portée d'un changement pour établir un plan et planter le travail,
- développement d'estimations justes de ressources nécessaires pour effectuer le travail,
- analyse des coûts / bénéfices du changement demandé,
- communication de la complexité d'un changement donné.

**Localisation de feature** : l'identification dans le code source des parties correspondant à une *feature* spécifique. Une *feature* est une fonctionnalité du logiciel définie par ses spécifications et accessible aux développeurs et utilisateurs. Ce concept est une des activités les plus fréquentes en maintenance/évolution et est utilisé dans le cadre de l'analyse d'impact des changements.

**Intégration continue** : phase de maintenance préventive (dans le processus de développement logiciel), relié à l' *extreme programming*, consistant d'un ensemble de pratiques utilisé en génie logiciel permettant de vérifier à chaque modification du code source que le résultat des modifications ne produit pas de régression dans l'application développée. 2. objectifs : détecter et corriger les erreurs pendant la phase de développement du logiciel.

**DevOps** : ensemble de pratiques en génie logiciel combinant le développement logiciel et les opérations de la technologie de l'information. Son objectif est de raccourcir le temps entre l'intégration d'un changement d'un système et sa mise en production, tout en assurant la bonne qualité des livrables (*features, fixes, updates*). Le DevOps est réalisé par l'utilisation de *toolchains* pertinents aux catégories suivantes :

- codage : outils pour le développement du code source, sa revue, et sa gestion, etc,
- construction : outils d'intégration continue, etc,
- tests : outils de test continu.

**Compréhension de logiciel** : compréhension du code source. Réalisable par différents processus tels que l'examen et compréhension d'un système, l'acquisition de connaissances sur un programme informatique ou le développement de modèles mentaux de l'architecture, du sens et du comportement des logiciels. Ce concept a plusieurs objectifs tels que la récupération d'informations de haut niveau (modèles) sur un système, incluant sa structure (composants et leurs interrelations), ses fonctionnalités (opérations effectuées et composants cibles), son comportement dynamique (transformation de l'entrée en sortie), sa raison d'être (le processus de conception et les décisions prises dans ce processus), sa construction, ses modules, sa documentation et ses suites de tests...

**Code review** : phase de maintenance préventive (dans le processus de développement logiciel) dans laquelle les développeurs, *peer-reviewers* et les testeurs se mettent ensemble pour revoir le code dans son entièreté. L'objectif est de détecter et corriger les erreurs pendant la phase de développement du logiciel. Ce concept se réalise en lisant le code source ligne par ligne afin de détecter les éventuels défauts, s'assurer de la cohérence globale du logiciel, s'assurer de la qualité des commentaires, s'assurer de l'adhérence aux standards de programmation adoptés.

**Software mining** : application de la découverte de connaissances dans le domaine de la modernisation de logiciels en examinant les artefacts logiciels existants. Il s'agit d'une technique qui sert dans la rétro-ingénierie.

**Redocumentation** : processus de rétro-ingénierie utilisé pour générer de la documentation pour un système hérité qui en manque, en se basant sur les artefacts logiciels existants. L'objectif est de minimiser les efforts de rétro-ingénierie et faciliter la compréhension du logiciel.

Les définitions précédentes aident à organiser les concepts entre eux tels qu'ils sont présentés dans le diagramme UML en figure (1.1). Ce dernier est également disponible dans l'archive du rendu.

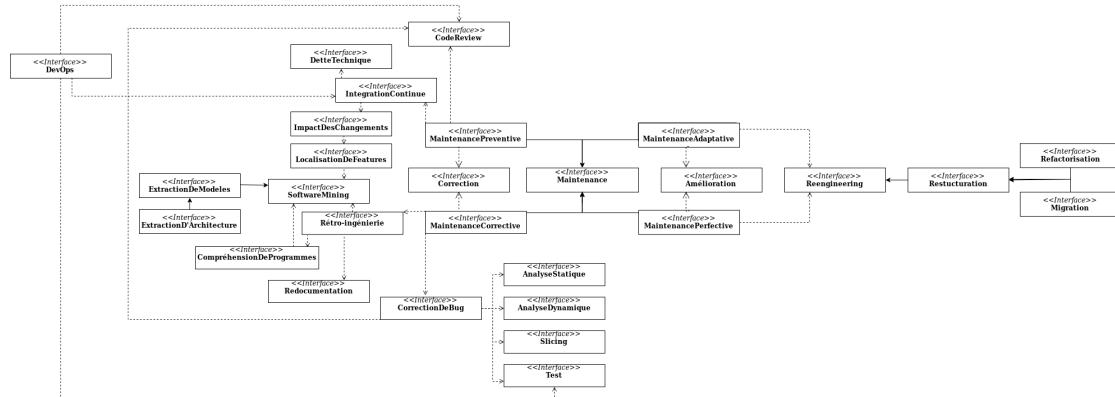


FIGURE 1.1 – Modèle UML montrant les concepts liés au domaine de l'évolution/maintenance de logiciel ainsi que leurs relations.

## 1.2 Exercice 2 : Outils de maintenance / évolution

Notre choix s'est porté sur **Checkstyle** et **JSCity**.

### 1.2.1 Checkstyle

#### Description

**Checkstyle** est un outil permettant aux développeurs de respecter des normes de programmation. Cet outil se base sur l'analyse statique et vérifie que la syntaxe du code produit respecte des règles définies. Ces règles peuvent être des conventions publiées par des organismes reconnus comme Google ou Sun, mais elles peuvent aussi être des règles personnelles.

De plus, **Checkstyle** peut aussi être utilisé pour étudier d'autres concepts comme une mauvaise qualité dans la conception ou dans le développement d'une méthode. Dans ce TP, nous nous sommes concentrés sur la correction syntaxique.

#### Installation

Il y a plusieurs options pour utiliser cet outil.

- Installé comme *plug-in* dans un IDE (IntelliJ ou Eclipse) ;
- Configuré comme dépendance d'un projet Maven (*cf. Listing 1.1*) ;

— Installé <sup>1</sup> et utilisé en ligne de commande.

```
1 <plugin>
2   <groupId>org.apache.maven.plugins</groupId>
3   <artifactId>maven-checkstyle-plugin</artifactId>
4   <version>${checkstyle-maven-plugin.version}</version>
5   <configuration>
6     <configLocation>checkstyle.xml</configLocation>
7   </configuration>
8   <executions>
9     <execution>
10       <goals>
11         <goal>check</goal>
12       </goals>
13     </execution>
14   </executions>
15 </plugin>
```

Listing 1.1 – Dépendance Maven pour installer Checkstyle.

### Exemple d'utilisation avec les conventions de Java Google et Sun

Une fois le *plug-in* installé, nous pouvons ouvrir le menu de **Checkstyle** (*cf. Figure 1.2*) ainsi que la perspective associée. Dans ce menu (accessible au chemin : **Windows > Preferences > Checkstyle**) nous pouvons choisir les règles à suivre pour notre projet. Sur la capture écran on remarque celles de Google et de Sun. Plus tard nous verrons qu'il est également possible de créer de nouvelles conventions de code en appuyant sur le bouton **New**.

---

1. <https://sourceforge.net/projects/checkstyle/>

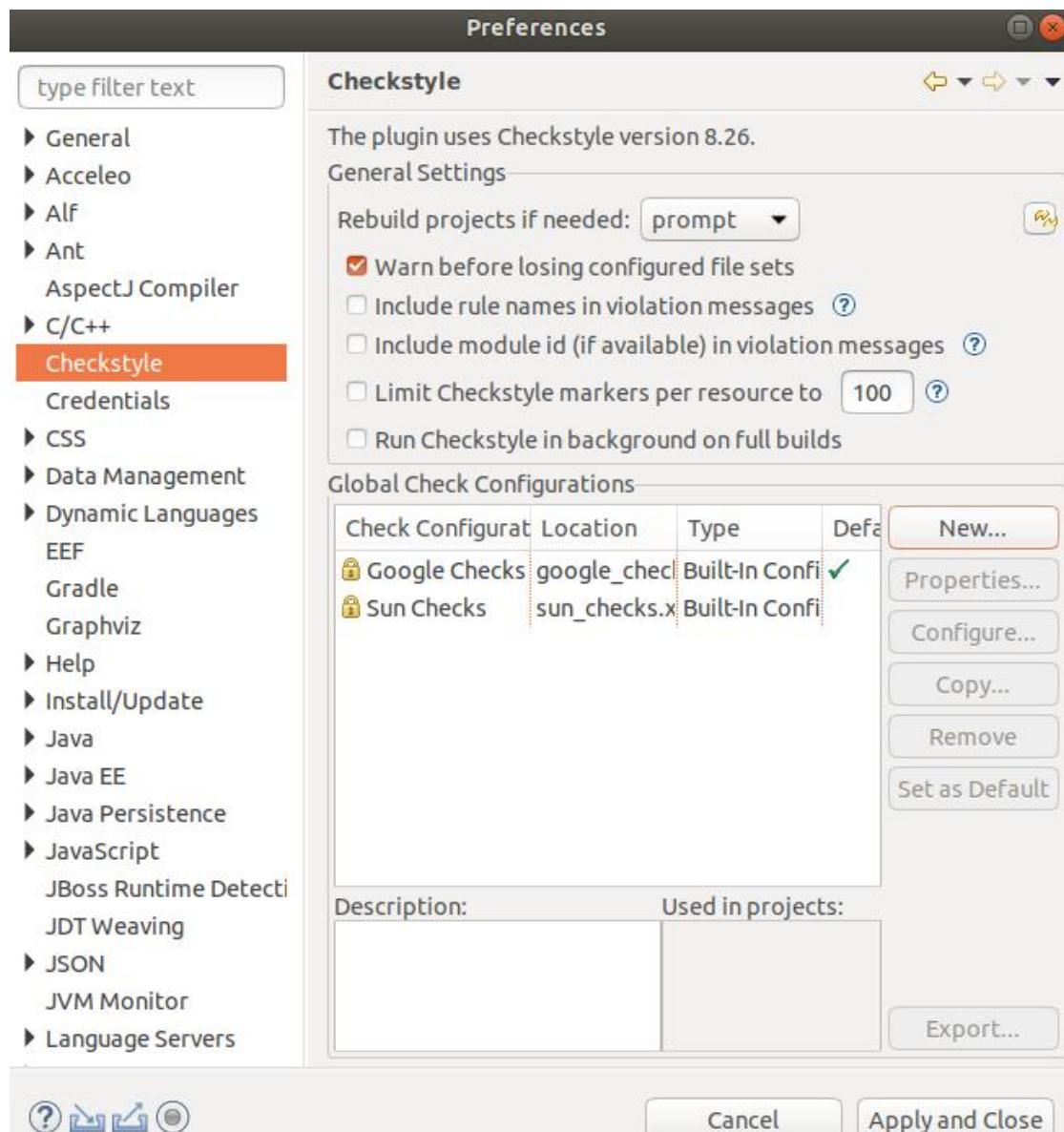


FIGURE 1.2 – Menu de Checkstyle.

Une fois les conventions à suivre choisies (dans un premier temps celles de Google), pour réaliser l'analyse de Checkstyle, il faut faire un clic droit dans le Package Explorer du projet à analyser puis cliquer sur le menu Checkstyle et enfin cliquer sur Check Code with Checkstyle. L'analyse obtenue est visualisable de deux manières différentes.

Checkstyle violation type	Occurrences
'X' à la colonne X devrait être sur la même ligne que la prochaine ligne.	2
'X' doit être séparé de la déclaration précédente.	14
La première phrase de la Javadoc doit se terminer avec un point.	20
Il y a une espace de trop avant 'X'.	11
Commentaire au niveau d'indentation X au lieu de X, l'indentation est incorrecte.	3
'X' au niveau d'indentation X n'est pas indenté correctement.	399
La ligne contient un caractère tabulation.	1408
Le commentaire Javadoc à la colonne X ne peut être analysé.	1
La classe de haut niveau X doit résider dans son propre fichier.	3
<b>Le fils de 'X' au niveau d'indentation X n'est pas indenté correctement.</b>	<b>501</b>
Il manque une espace avant 'X'.	73
Il manque une espace après 'X'.	91
La ligne excède X caractères (trouvé X).	97
Une ligne vide doit être suivie par une balise <p> sur la ligne suivante.	1
Mauvais ordre lexicographique pour l'import 'X'. Il devrait être 'Y'.	4
'X' devrait être sur une nouvelle ligne.	18
Chaque déclaration de variable doit faire l'objet d'une instruction.	5
Commentaire Javadoc manquant.	17
Il y a une espace de trop après 'X'.	1
'X' à la colonne X devrait avoir un saut de ligne après.	12
'X' à la colonne X devrait être seul sur sa ligne.	12
L'instruction 'X' devrait utiliser des accolades ('{' et '}').	18
La balise Javadoc doit avoir une description non vide.	2

FIGURE 1.3 – Affichage de la liste de toutes les violations observées.

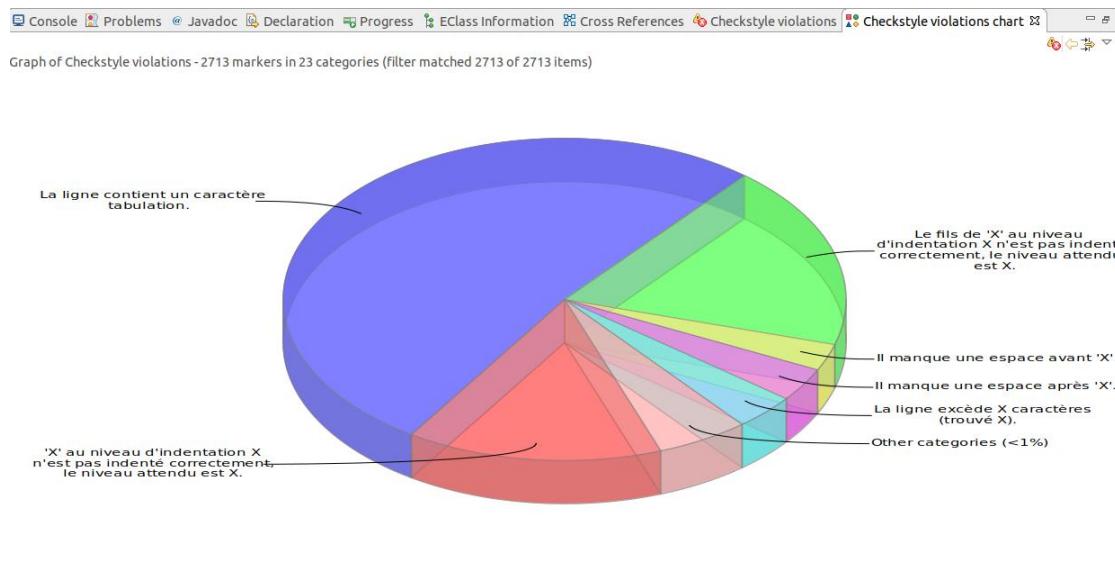


FIGURE 1.4 – Visualisation graphique selon les types de violation.

Les violations illustrées sont celles de la convention de Google pour Java concernant le code réalisé. Nous avons ensuite essayé d'appliquer les conventions de Sun sur le même projet. On remarque que ces dernières sont moins nombreuses.

Checkstyle violation type	Occurrences
Il y a une espace de trop avant 'X'.	11
Il manque une espace avant 'X'.	73
Il manque une espace après 'X'.	103
Les classes utilitaires ne doivent pas avoir de constructeur public.	5
Instruction vide.	1
'X' devrait être sur une nouvelle ligne.	18
Balise Javadoc X manquante pour 'X'.	13
Commentaire Javadoc manquant.	65
Il y a une espace de trop après 'X'.	7
La classe X devrait être déclarée finale.	1
'X' à la colonne X devrait avoir un saut de ligne après.	12
Le nom 'X' n'est pas conforme à l'expression 'X'.	5
Le fichier package-info.java est manquant.	8
Balise Javadoc @return manquante.	5
Le paramètre X devrait être final.	69
Il manque un caractère NewLine à la fin du fichier.	1
La ligne excède X caractères (trouvé X).	230
Le fichier contient des caractères de tabulation (ce n'est qu'une première occurrence).	20
Line has trailing spaces.	231
'X' masque un attribut.	10
La variable 'X' devrait être privée et avoir des accesseurs.	10
La première ligne de la Javadoc doit se terminer avec un point.	20
Chaque déclaration de variable doit faire l'objet d'une instruction.	5

FIGURE 1.5 – Affichage de la liste de toutes les violations observées sur le même projet avec les conventions de Sun.

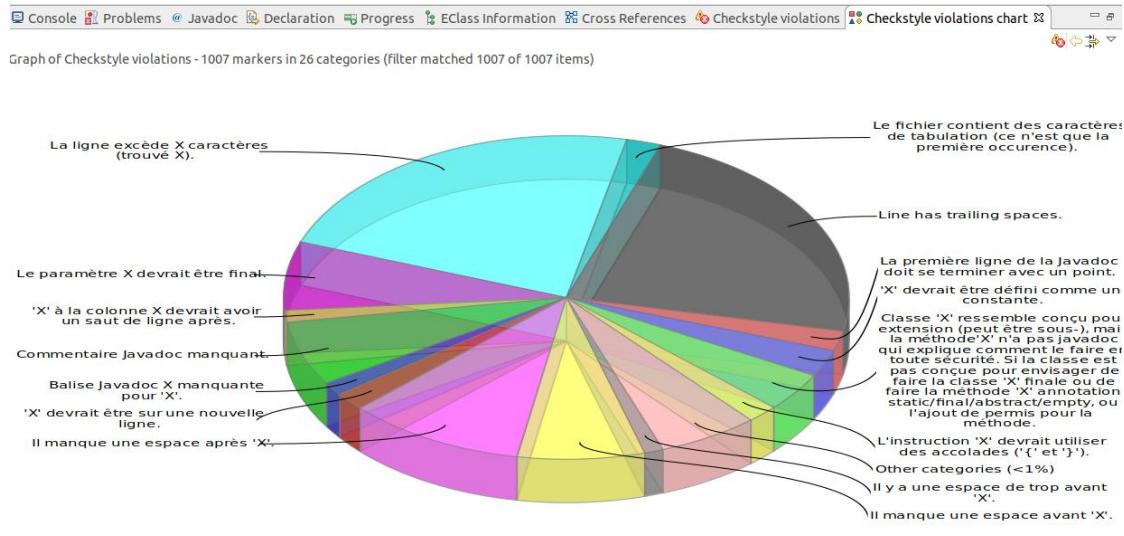


FIGURE 1.6 – Visualisation graphique selon les types de violation sur le même projet avec les conventions de Sun.

## Correction de la syntaxe

En plus d'une vue générale des violations des règles, il est possible de voir toutes les violations pour une règle précise. En reprenant la figure 1.6, on peut accéder à la liste de toutes les erreurs concernant la règle « Il y a un espace en trop avant X » en cliquant sur cette dernière sur le graphe. On obtient alors une liste comme celle en figure 1.7

Resource	In Folder	Line	Message
② <a href="#">Couple.java</a>	/HMIN306_Serial/src/couple	81	Il y a une espace de trop avant ';
② <a href="#">CouplingParser.java</a>	/HMIN306_Serial/src/couple	183	Il y a une espace de trop avant ';
② <a href="#">CouplingParser.java</a>	/HMIN306_Serial/src/couple	288	Il y a une espace de trop avant ';
② <a href="#">CouplingParser.java</a>	/HMIN306_Serial/src/couple	288	Il y a une espace de trop avant ';
② <a href="#">CouplingParser.java</a>	/HMIN306_Serial/src/couple	290	Il y a une espace de trop avant ';
② <a href="#">CouplingParser.java</a>	/HMIN306_Serial/src/couple	290	Il y a une espace de trop avant ';
② <a href="#">CouplingParser.java</a>	/HMIN306_Serial/src/couple	350	Il y a une espace de trop avant ';
② <a href="#">CouplingParser.java</a>	/HMIN306_Serial/src/couple	350	Il y a une espace de trop avant ';
② <a href="#">CouplingParser.java</a>	/HMIN306_Serial/src/couple	352	Il y a une espace de trop avant ';
② <a href="#">CouplingParser.java</a>	/HMIN306_Serial/src/couple	352	Il y a une espace de trop avant ';
② <a href="#">CouplingParser.java</a>	/HMIN306_Serial/src/couple	118	Il y a une espace de trop avant ';

FIGURE 1.7 – Liste des violations de la règle « Il y a un espace en trop avant X ».

Lorsque l'on clique sur une violation, **Checkstyle** nous amène à la ligne où l'erreur a lieu pour que nous puissions la corriger.

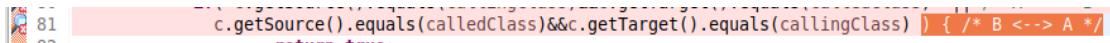


FIGURE 1.8 – Le problème vient de l'espace supplémentaire après la condition.

**Checkstyle** soulève beaucoup de manques de respect des conventions de programmations de Sun et de Google. Nous avons voulu corriger toutes les violations soulevées par **Checkstyle** sur un projet. Pour cela nous avons pris un projet plus petit, distribué par notre enseignant pour avoir une base de code lors du TP suivant. Voici les erreurs soulevées par **Checkstyle**.

Checkstyle violation type	Occurrences
② Le nom 'X' n'est pas conforme à l'expression 'X'.	9
② Le fichier contient des caractères de tabulation (ce n'est qu'	6
② Line has trailing spaces.	19
② Le fichier package-info.java est manquant.	1
② Le paramètre X devrait être final.	11
② La variable 'X' devrait être privée et avoir des accesseurs.	5
② Import inutilisé - X.	7
② Commentaire Javadoc manquant.	25
② Il y a une espace de trop après 'X'.	2
② Classe 'X' ressemble conçu pour extension (peut être sous-)	12
② La ligne excède X caractères (trouvé X).	22
② Les classes utilitaires ne doivent pas avoir de constructeur	1

FIGURE 1.9 – Liste des violations soulevées sur un petit projet.

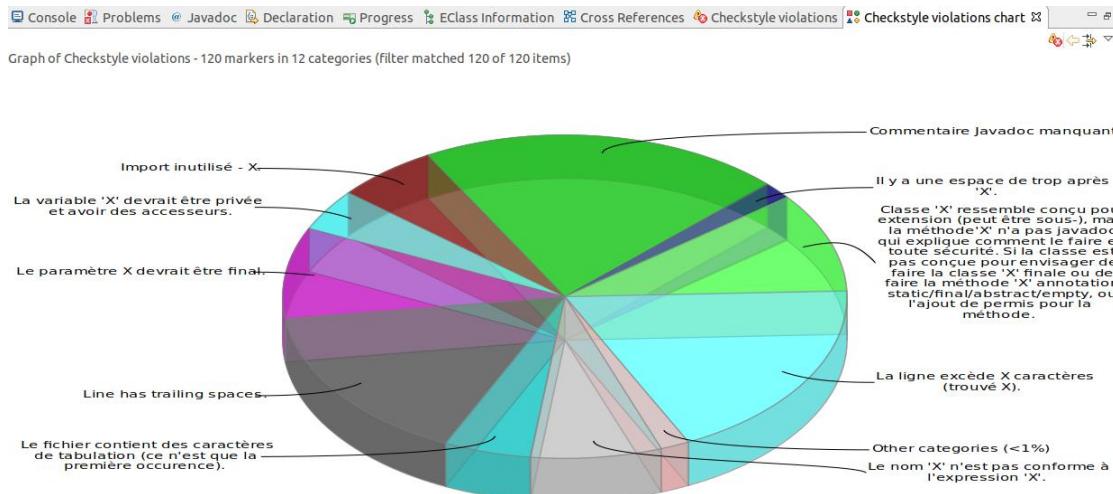


FIGURE 1.10 – Visualisation graphique des différentes règles violées sur un petit projet.

Details of Checkstyle violation "La variable 'X' devrait être privée et avoir des accesseurs." - 5 occurrences			
Resource	In Folder	Line	Message
FieldAccessVisitor.java	/step2/src/step2	12	La variable 'fields' devrait être privée et avoir des accesseurs.
MethodDeclarationVisi	/step2/src/step2	10	La variable 'methods' devrait être privée et avoir des accesseurs.
MethodInvocationVisi	/step2/src/step2	11	La variable 'methods' devrait être privée et avoir des accesseurs.
MethodInvocationVisi	/step2/src/step2	12	La variable 'superMethods' devrait être privée et avoir des accesseurs.
TypeDeclarationVisit	/step2/src/step2	10	La variable 'types' devrait être privée et avoir des accesseurs.

FIGURE 1.11 – Zoom sur une règle.

Après avoir corrigé toutes les erreurs de syntaxe soulevées par **Checkstyle**, celui-ci nous informe qu'il ne trouve aucune erreur.

Overview of Checkstyle violations - 0 markers in 0 categories (filter matched 0 of 0 items)	
Checkstyle violation type	Occurrences

FIGURE 1.12 – Affichage obtenu après correction.

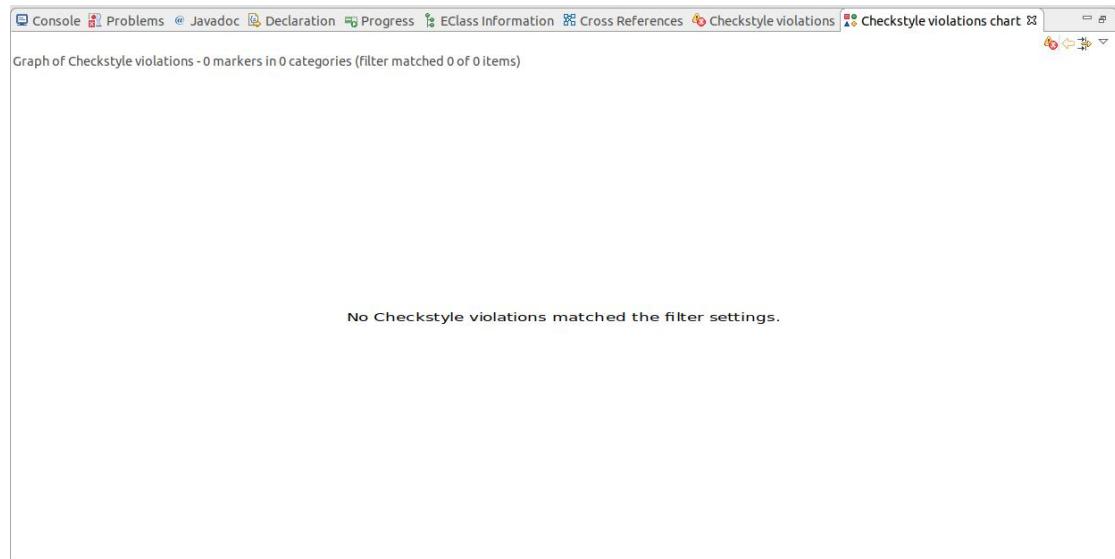


FIGURE 1.13 – Affichage « graphique » obtenu après correction

### Exemple d'utilisation : créer ses propres règles

Jusqu'à présent nous avons utilisé des règles et des conventions de codage réalisées par des organismes officiels. Cependant, **Checkstyle** permet également de définir ses propres règles. C'est ce que nous avons fait avec un exemple trivial. Admettons qu'une équipe de développement peu précautionneuse ait une seule exigence : que tous les attributs d'instance des classes aient un nom commençant par **toto\_** suivi d'un numéro<sup>2</sup>. Il va donc falloir définir une règle dans une nouvelle configuration.

2. Dans la réalité une telle règle serait ridicule mais il s'agit ici d'un exemple.

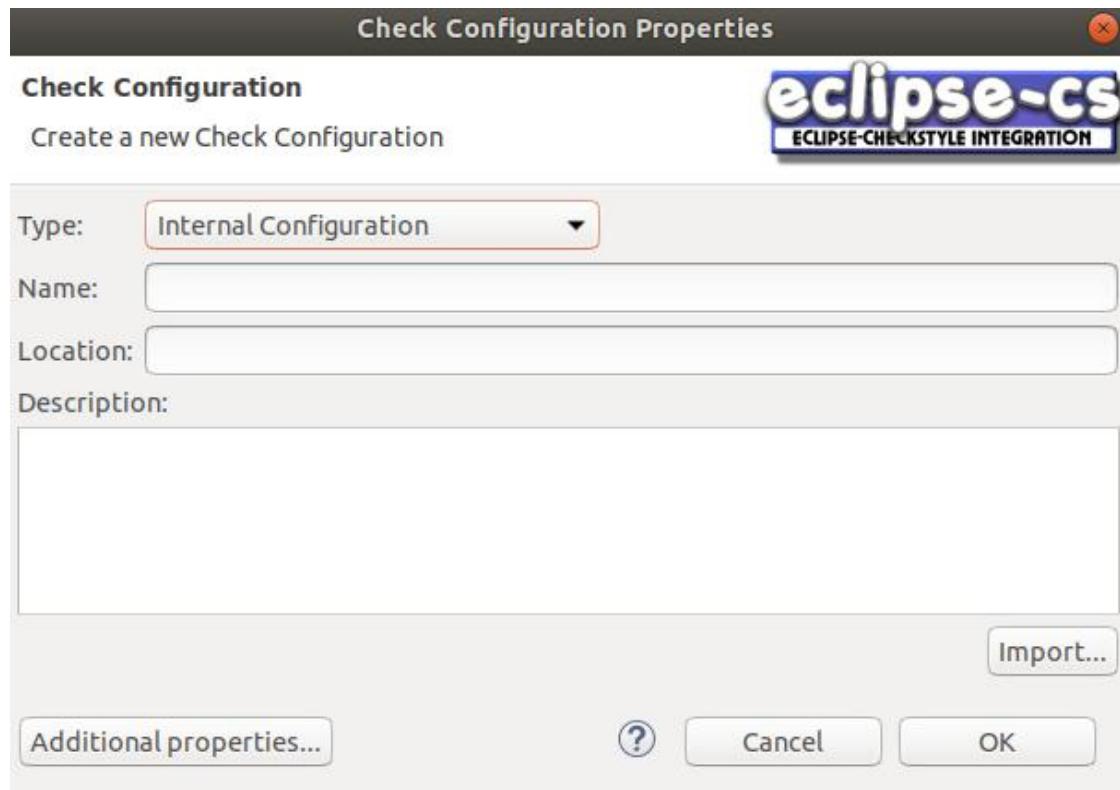


FIGURE 1.14 – Fenêtre de création d'une nouvelle norme.

Une fois le nom de la norme choisié, il est temps de définir des règles. Dans notre cas, nous ne voulons accepter qu'un certain format pour le nom des attributs d'instance d'une classe.

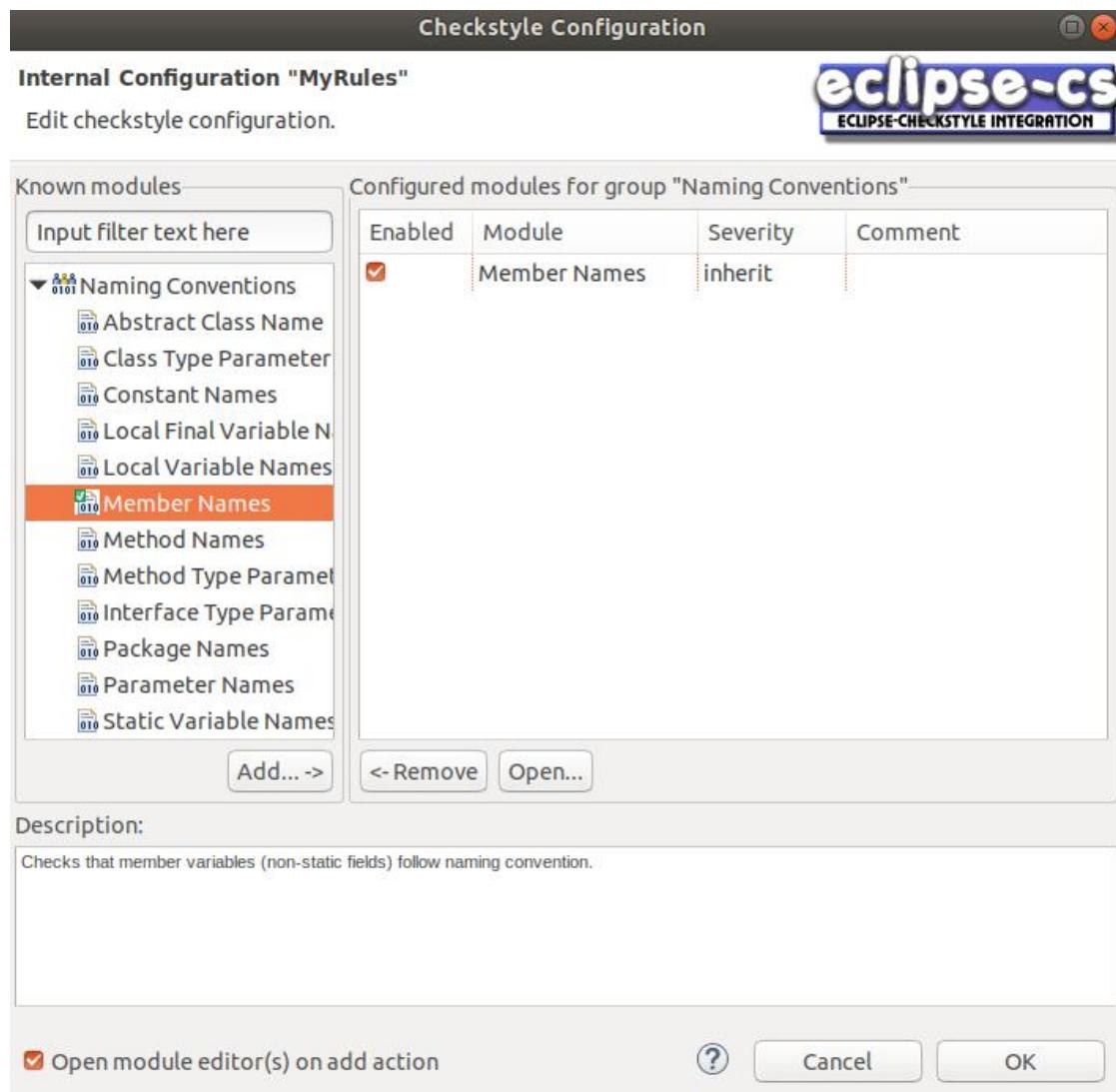


FIGURE 1.15 – Interface permettant de choisir des règles à créer.



FIGURE 1.16 – Nouvelle règle et valeur admise.



FIGURE 1.17 – Exemple de valeur non permise par la règle.

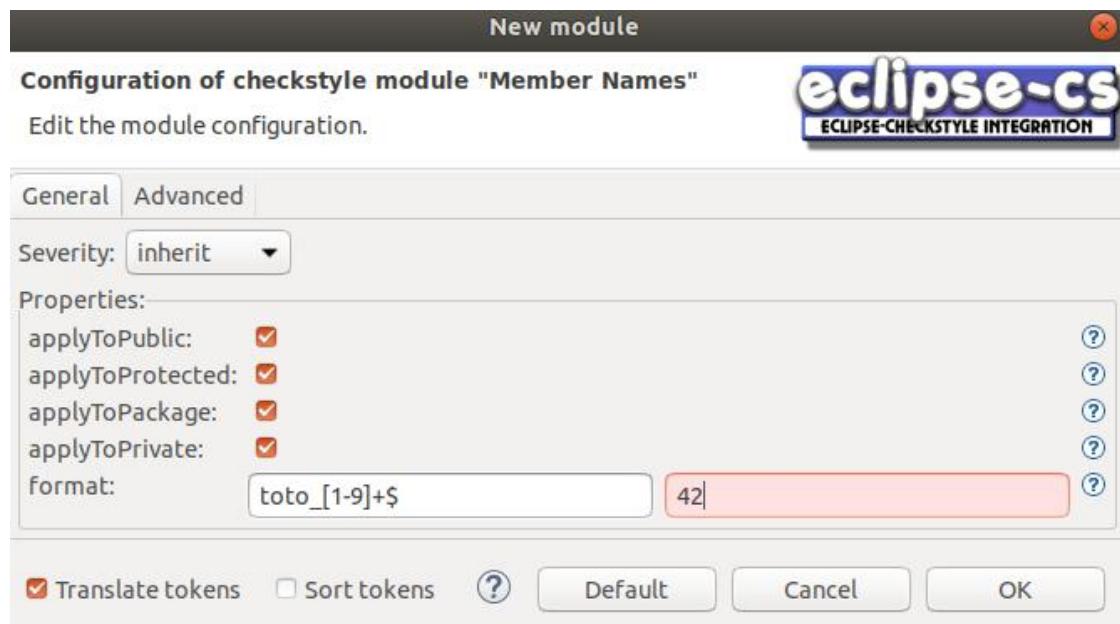


FIGURE 1.18 – Exemple de valeur non autorisée par la règle.

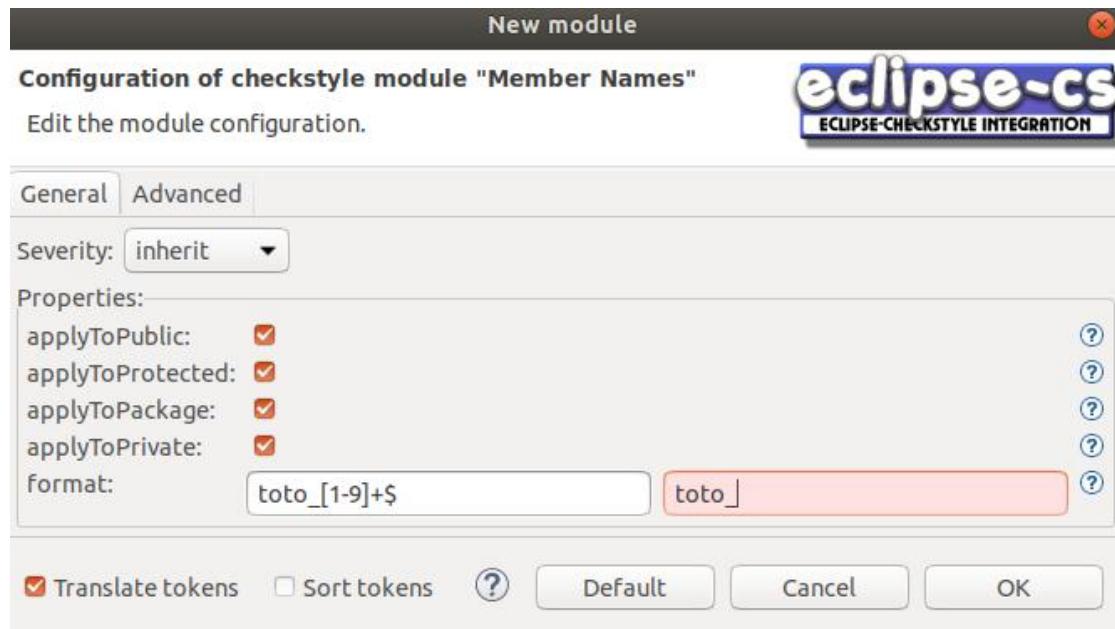


FIGURE 1.19 – Exemple de valeur non autorisée par la règle.

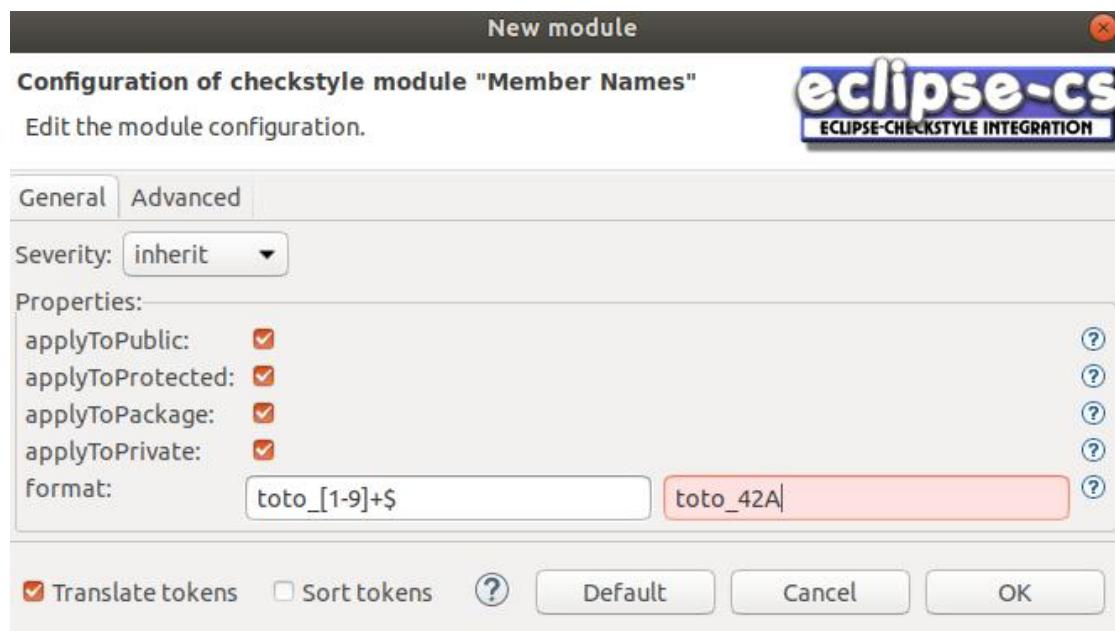


FIGURE 1.20 – Exemple de valeur non autorisée par la règle.

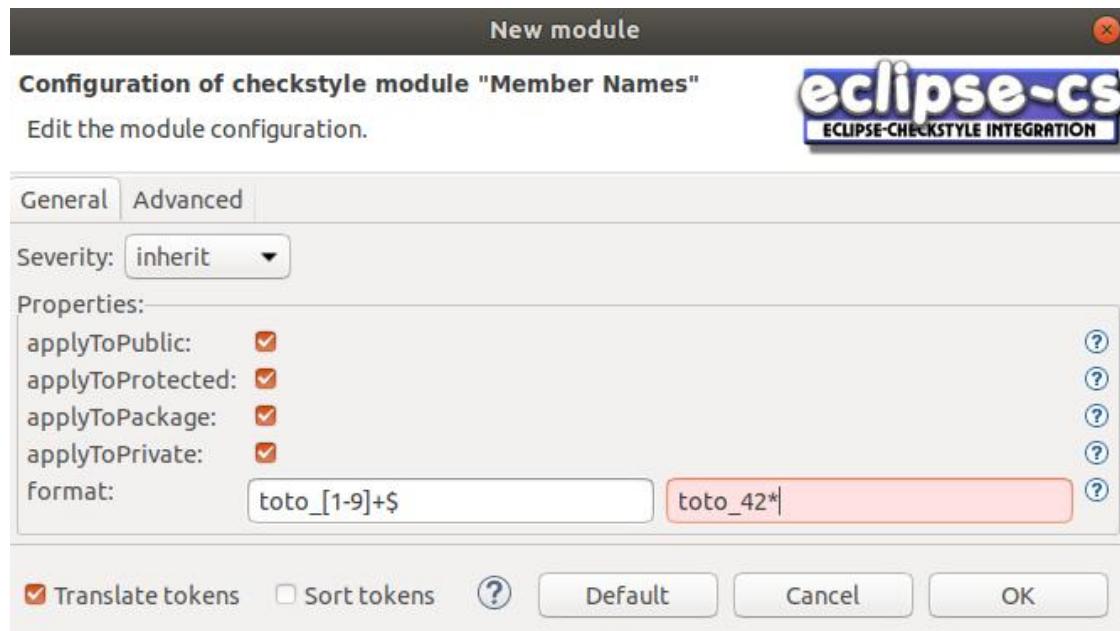


FIGURE 1.21 – Exemple de valeur non autorisée par la règle.

La nouvelle norme étant créée, il est temps de la tester sur le même projet précédent (nommé **step2**).

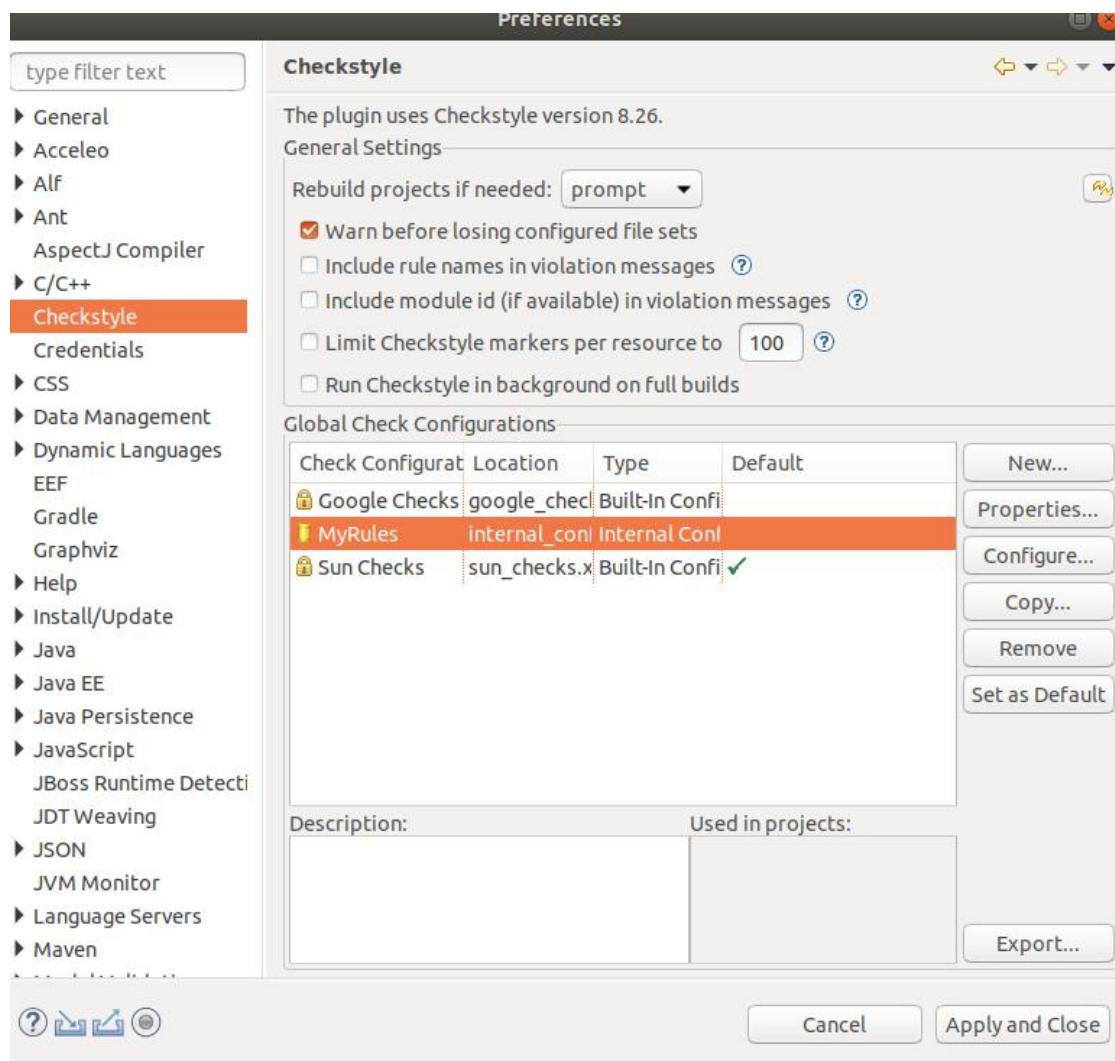


FIGURE 1.22 – La norme est bien créée, il faut la sélectionner pour l'appliquer.

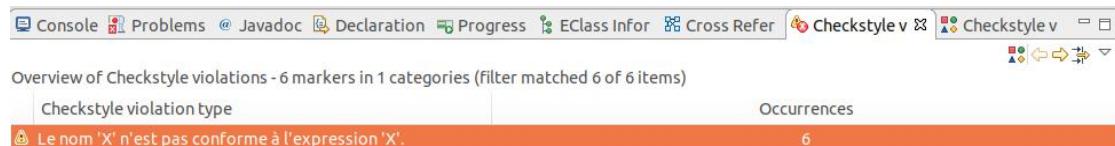


FIGURE 1.23 – La règle est bien prise en compte.

Details of Checkstyle violation "Le nom 'X' n'est pas conforme à l'expression 'X'." - 6 occurrences			
Resource	In Folder	Line	Message
FieldAccessVisitor.java	/step2/src/step	15	Le nom 'fields' n'est pas conforme à l'expression 'toto_[1-9]+\$'.
MethodDeclarationVis	/step2/src/step	14	Le nom 'methods' n'est pas conforme à l'expression 'toto_[1-9]+\$'.
MethodInvocationVis	/step2/src/step	15	Le nom 'methods' n'est pas conforme à l'expression 'toto_[1-9]+\$'.
MethodInvocationVis	/step2/src/step	21	Le nom 'superMethods' n'est pas conforme à l'expression 'toto_[1-9]+\$'.
TypeDeclarationVisit	/step2/src/step	14	Le nom 'types' n'est pas conforme à l'expression 'toto_[1-9]+\$'.
VariableDeclarationF	/step2/src/step	13	Le nom 'variables' n'est pas conforme à l'expression 'toto_[1-9]+\$'.

FIGURE 1.24 – Notre norme n'ayant qu'une seule règle, il est normal que Checkstyle ne remarque que celle-ci.

## Conclusion

Nous avons vu comment utiliser **Checkstyle** pour suivre une norme de programmation, cet outil nous a permis de corriger l'ensemble d'un petit projet et nous a donné l'occasion d'en apprendre plus sur les conventions d'organismes comme Sun ou Google<sup>3</sup>. Enfin, nous avons pu voir comment définir sa propre convention de programmation.

**Checkstyle** est un outil puissant et utile pour veiller à ce que le code produit par une équipe de développement (donc constituée de plusieurs personnes) soit uniforme. Nous l'avons trouvé intéressant et souhaitons continuer à l'utiliser dans le futur.

### 1.2.2 CodeCity : Introduction à la métaphore « *Software City* »

Pour bien comprendre la philosophie de conception et d'utilisation de **JSCity**, un résumé de l'outil **CodeCity**, sur lequel se base **JSCity** dans sa conception et implémentation, s'avère primordial.

Dans la suite de cette partie nous résumons les aspects de **CodeCity** essentiels pour la compréhension de **JSCity**. Enfin, nous terminons par un résumé de **JSCity**, son installation et un guide de son utilisation.

#### Introduction

**CodeCity** est un outil multiplateforme interactif pour la visualisation 3D et l'analyse des logiciels orienté-objet indépendants des langages et des plateformes d'implémentation. Il permet la visualisation d'un logiciel à partir de modèles abstraits le décrivant, plutôt qu'à partir de son code source. Il a été introduit pour faciliter la compréhension, l'analyse, la rétro-ingénierie et le suivi de l'évolution des systèmes orienté-objet à grande échelle, en tant qu'une approche basée sur la métaphore « **ville logicielle** »(*software city*). Cette approche tire parti de l'aspect synthétique de la visualisation logicielle pour atteindre ses objectifs, notamment lorsque le système dévoile une quantité considérable d'informations à traiter, afin d'estimer sa complexité logicielle.

#### Les aspects de la métaphore *Software City*

La conception basée sur la métaphore *software city* décrit un système orienté-objet de la manière suivante :

**système** : ville ;  
**paquetages** : arrondissements ;  
**classes** : bâtiments.

En outre, les propriétés visuelles des artefacts de la ville reflètent des mesures logicielles propres au système, décrites de la manière suivante :

**le nombre de méthodes d'une classe<sup>4</sup>** : hauteur du bâtiment ;  
**le nombre d'attributs d'une classe<sup>5</sup>** : longueur et largeur de la base du bâtiment ;  
**le niveau d'imbrication d'un paquetage** : la saturation de la couleur de l'arrondissement (*e.g.* les paquetages avec un niveau d'imbrication assez profond sont colorés en bleu foncé, alors que ceux ayant un niveau d'imbrication relativement superficiel le sont en bleu clair)

---

3. Ces derniers préconisent par exemple que des espaces soient utilisés au lieu des tabulations pour l'indentation de code.

4. **NOM** : Number of Methods

5. **NOA** : Number of Attributes

Cette approche permet ainsi non seulement d'avoir un aperçu de la structure du système étudié ainsi que de sa magnitude.

## Composition et Interaction

La fenêtre de visualisation du logiciel (*cf. Figure 1.25*) est composée d'une fenêtre de navigation interactive et d'un panel informatif (*à droite*), affichant les informations de l'élément sélectionné dans la fenêtre de navigation. Une ville logicielle peut être explorée par la souris ou le clavier dans une perspective panoramique permettant de *zoomer* en avant/arrière et de graviter autour de la ville. De plus, on peut se déplacer entre ses artefacts et les survoler sur les plans horizontal et vertical.

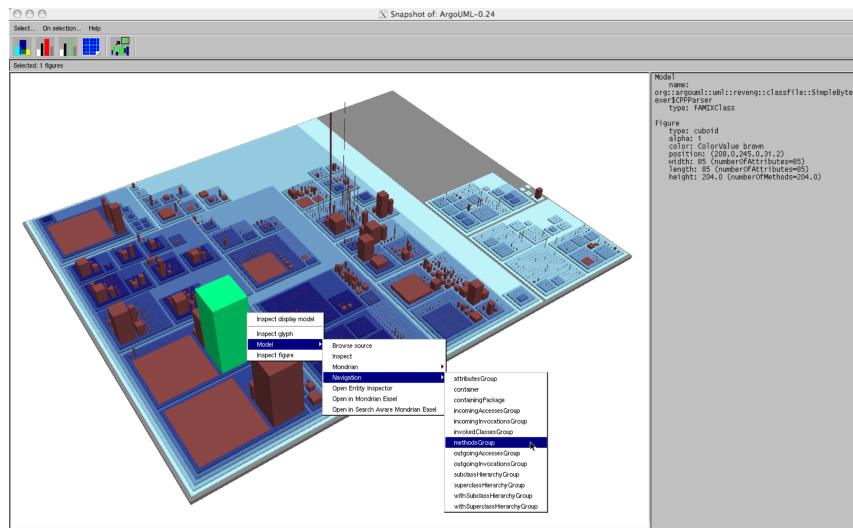


FIGURE 1.25 – Ville logicielle du système Java ArgoUML

Par ailleurs, l'interaction avec les artefacts peut se faire selon les modes d'interaction suivants :

**interaction directe** : le menu contextuel affiché sur un élément lors de sa sélection ;

**interaction indirecte** : un mécanisme de requêtes **absolu** (*e.g.* sur les interfaces, classes racines) ou **relatif** à la sélection en cours (*e.g.* les classes invoquant des méthodes parmi celles sélectionnées).

## Détection des dissonances conceptuelles

Outre la visualisation logicielle et le calcul des métriques, **CodeCity** permet d'évaluer la qualité logicielle du système du point de vue de sa conception, à travers des stratégies de détection identifiant des dissonances conceptuelles dans son design. La représentation des dissonances conceptuelles se base (*encore une fois*) sur la métaphore *software city*, pour visualiser leur localisation et distribution au sein du système.

Effectivement, en s'inspirant des cartes de maladies, **Code City** affecte une couleur vive spécifique à chaque type de dissonance conceptuelle, et des nuances de gris aux entités non-touchées par la dissonance (*cf. Figure 1.26*).

### *Remarque.*

Les types de dissonances conceptuelles et les stratégies de détection ne seront pas abordées ici, étant au delà de la portée de ce TP.

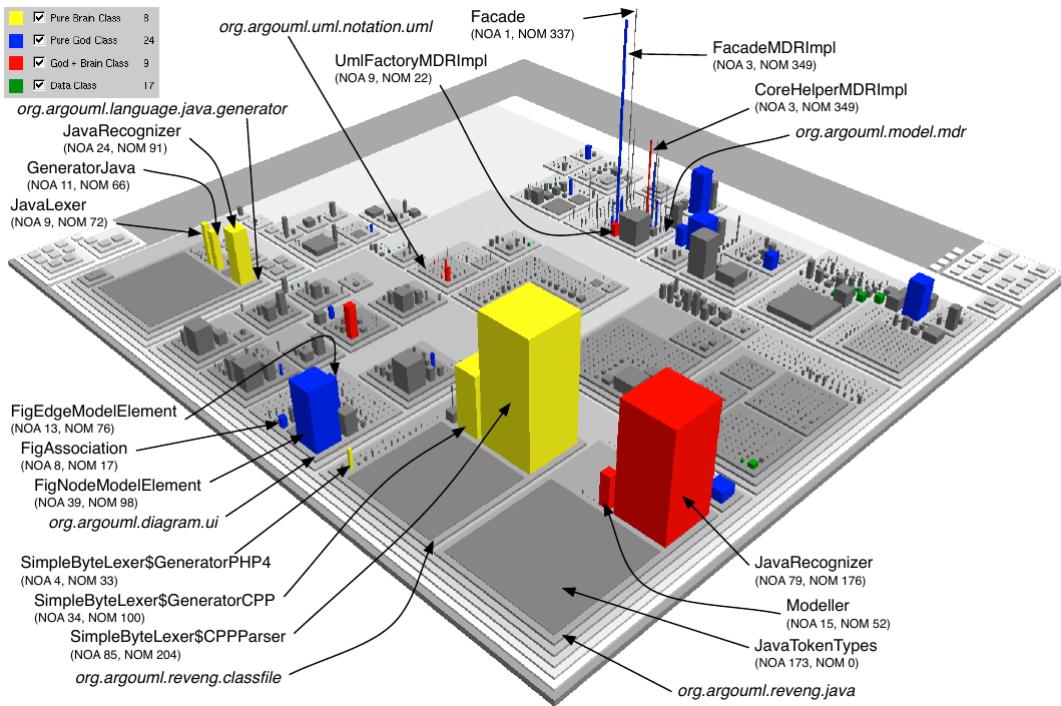


FIGURE 1.26 – Visualisation des dissonances conceptuelles du système Java ArgoUML

### Suivi de l'évolution d'un logiciel

CodeCity permet de suivre l'évolution d'un logiciel selon trois perspectives : **Age Map**, **Time Travel** et **Timeline**.

Le mode **Age Map** désigne une carte utilisant les couleurs et les distances entre les artefacts pour indiquer leur ancienneté (*cf.* Figures 1.27 et 1.28) tels que :

- les artefacts les plus récents/anciens sont colorés avec des nuances de couleurs claires/foncées ;
- les artefacts les plus récents sont empilés sur les artefacts les plus anciens ;
- les artefacts chronologiquement contemporains sont localisés les uns à côté des autres.

En outre, le mode **Time Travel** permet de visualiser les différentes versions des artefacts à différentes échelles de précision (*cf.* Figures 1.29 et 1.30).

Enfin, le mode **Timeline** permet de visualiser les transitions entre les différentes versions des artefacts (*cf.* Figure 1.31).

### Ensemble des outils utilisés

CodeCity est écrit en Smalltalk et construit sur le *framework Moose*<sup>6</sup>. Il permet de tirer partie de la grosse volumétrie d'informations structurées et détaillées offerte par les modèles des logiciels conformes au métamodèle **FAMIX**<sup>7</sup> implémenté par Moose, mais aussi de l'ensemble extensif des mesures logicielles calculées par celui-ci. Par conséquent, CodeCity permet la visualisation de systèmes orientés-objet écrits en des langages conformes à **FAMIX**, incluant Smalltalk, C++ et Java.

6. une plateforme libre et *open-source* pour l'analyse de données et de logiciels, construite en Pharo, une implémentation open-source du langage de programmation Smalltalk

7. un métamodèle de logiciels orientés-objet indépendant de tout langage de programmation

# Age map interpretation

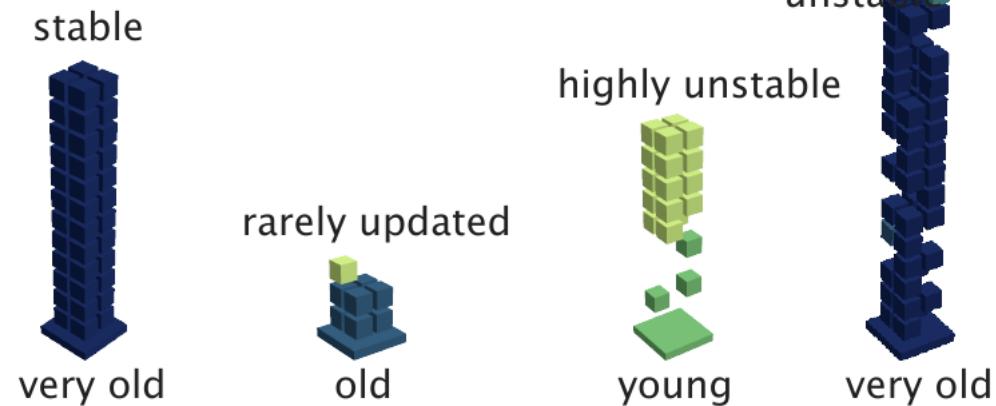
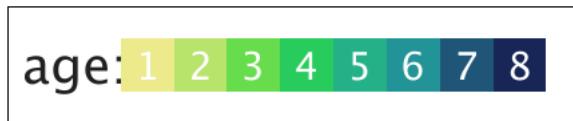


FIGURE 1.27 – Exemple de l’Age Map des méthodes d’une classe

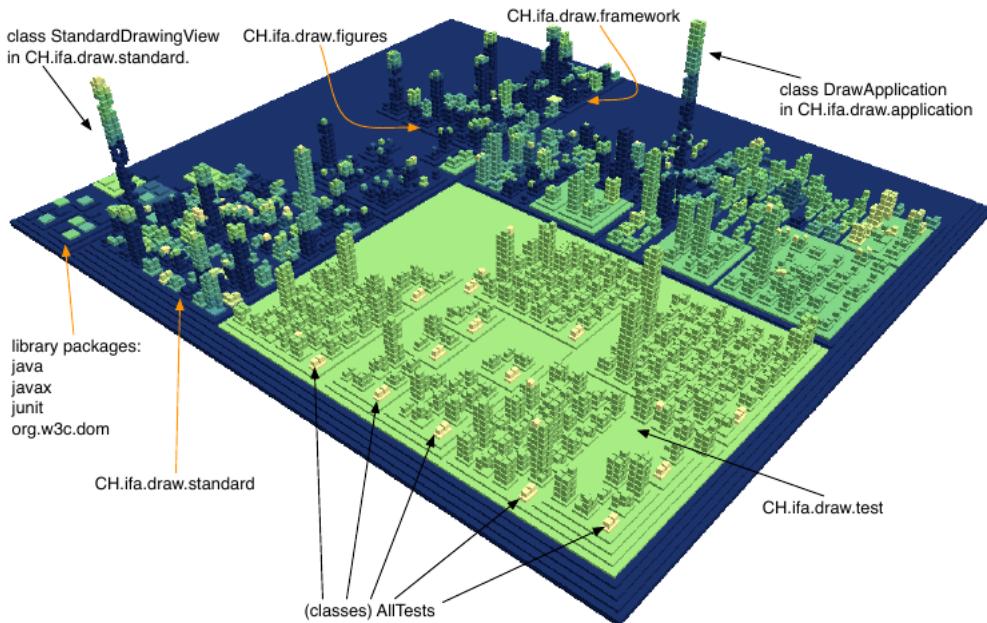


FIGURE 1.28 – Exemple de l’Age Map d’un système

Pour les logiciels Smalltalk, **CodeCity** construit leurs modèles **FAMIX** en utilisant **Moose**. Toutefois, les logiciels C++/Java nécessitent l’outil standalone **iPlasma** avant de construire leurs modèles **FAMIX**. **iPlasma** permet ainsi de *parse* leur code source, calculer les mesures logicielles dessus, en construire un modèle propre qui sera exporté en **MSE**<sup>8</sup> et importer ce dernier dans **Moose**.

À partir du modèle **FAMIX** d’un logiciel, **CodeCity** construit un modèle visuel, applique des *mappings* sur ses figures et les prépare pour le moteur de rendu. Le rendu des villes 3D

8. le format d’échange de modèles et métamodèles défini par Moose

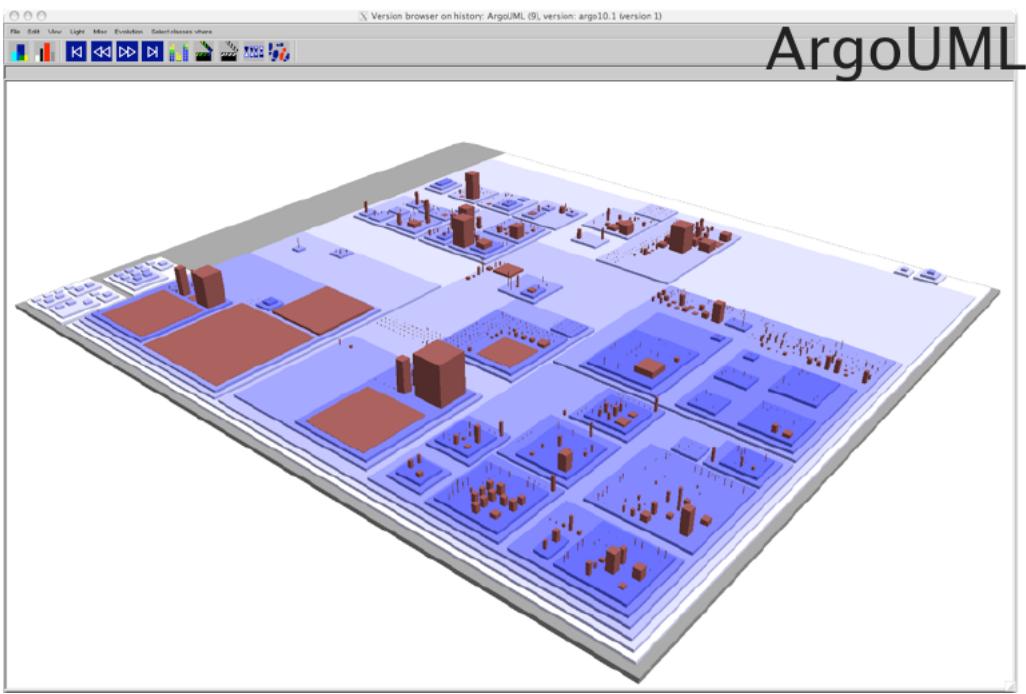


FIGURE 1.29 – Exemple de **Time Travel** d'ArgoUML à une échelle grossière

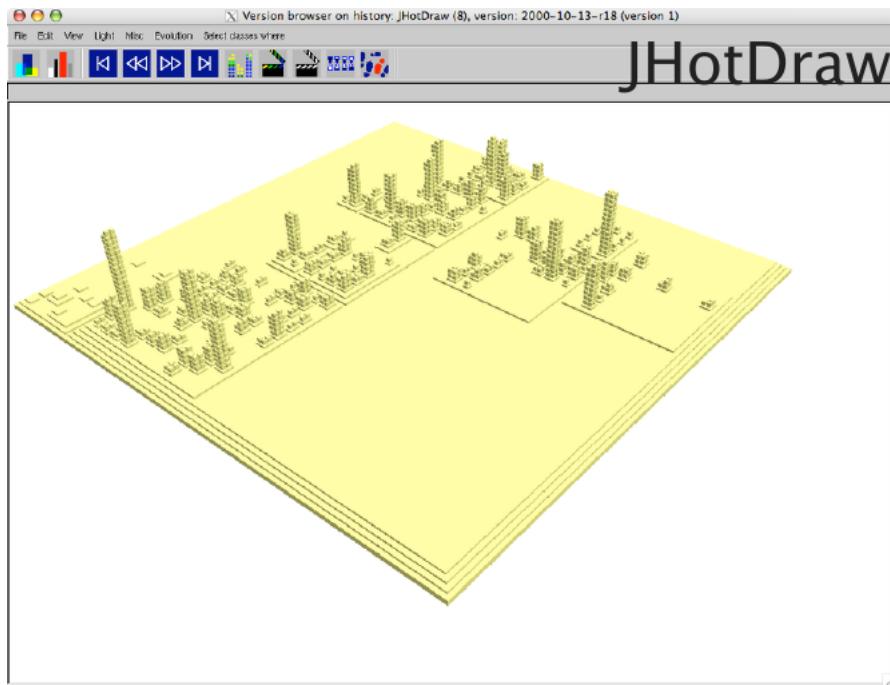


FIGURE 1.30 – Exemple de **Time Travel** de JHotDraw à une échelle fine

interactives est enfin effectué par une implémentation OpenGL du *framework Jun*<sup>9</sup>.

9. un framework libre et *open-source* pour la gestion du graphisme 3D et/ou des objets multimédia

# Timeline

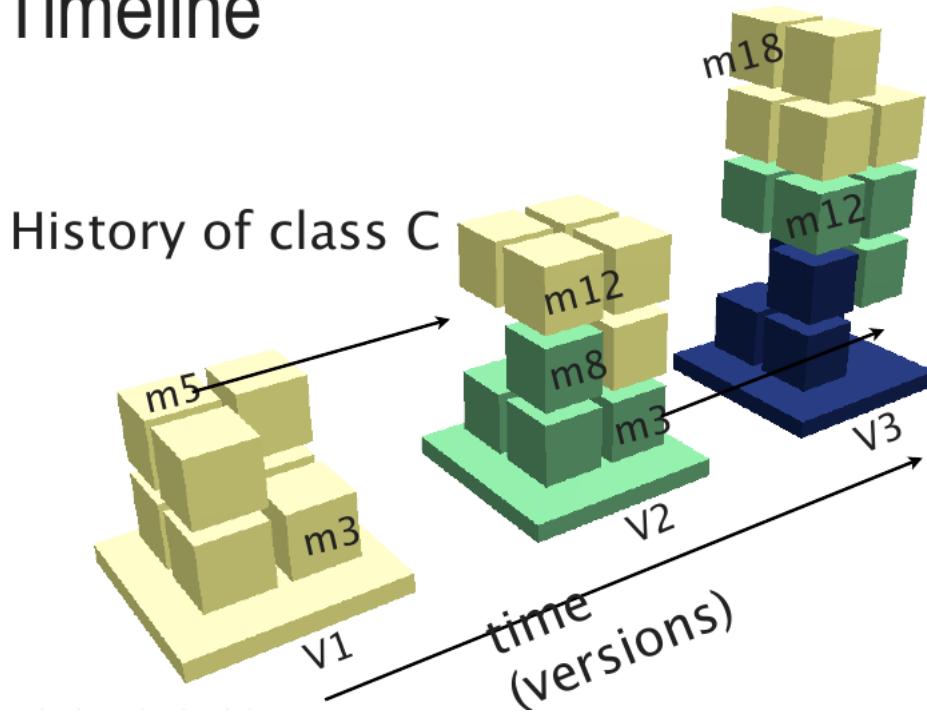


FIGURE 1.31 – Exemple de **Timeline** des méthodes d'une classe

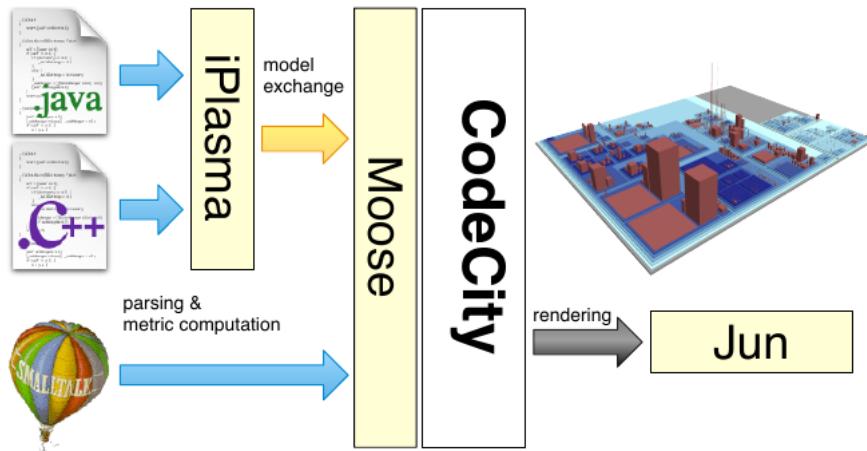


FIGURE 1.32 – La chaîne d'outils de **CodeCity**

### 1.2.3 JSCity : Une implémentation orientée-JavaScript de la métaphore *Software City*

#### Introduction

JSCity est un outil interactif de visualisation 3D et d'analyse de logiciels créés en JavaScript. Il s'agit d'une implémentation orientée-JavaScript minimalistique de la métaphore *software city*, notamment pour le **paradigme fonctionnelle** de JavaScript, utilisant la bibliothèque JavaScript `three.js` pour le graphisme 3D.

## Les aspects de la métaphore « *Software City* »

La conception basée sur la métaphore *software city* décrit un système en JavaScript de la manière suivante :

**système JavaScript** : ville ;  
**dossiers** : arrondissements ;  
**fichiers** : sous-arrondissements ;  
**fonctions** : bâtiments ;  
**fonctions internes à une fonction** : bâtiments empilés sur le bâtiment désignant la fonction encapsulante ;  
**fonctions nommées** : bâtiments en bleu ;  
**fonctions anonymes** : bâtiments en vert.

En outre, les propriétés visuelles des artefacts de la ville reflètent des mesures logicielles propres au système, décrites de la manière suivante :

**le nombre de lignes de code source d'une fonction**<sup>10</sup> : hauteur du bâtiment désignant la fonction.  
**le nombre de variables d'une fonction**<sup>11</sup> : longueur et largeur de la base du bâtiment désignant la fonction.

## Installation

L'installation de JSCity requiert une installation préalable de `Node.js`<sup>12</sup> et d'un serveur `MySQL` (e.g. MySQL, MariaDB<sup>13</sup>, Postgresql, ...).

Une fois les prérequis en place, il suffit de télécharger le fichier `zip` depuis le dépôt GitHub<sup>14</sup> du logiciel et extraire le dossier.

## Première utilisation

Suite au téléchargement du `zip` et l'extraction du dossier, un exemple d'une ville logicielle peut être directement visualisé en suivant les étapes suivantes :

1. vérifier que le serveur `MySQL` est actif, sinon le démarrer.
2. ouvrir la console.
3. se déplacer vers le dossier `path-to-jscity-folder/sql/` du logiciel.
4. ouvrir une session CLI de `MySQL`.
5. exécuter le script `schema.sql`, qui créera la base de données « `jscity` » et remplira les tables avec les informations du système exemple à visualiser.
6. modifier les entrées nécessaires (cf. Listing 1.2) dans le fichier `path-to-jscity-folder/js/config.json` afin de se connecter à la base de données pour permettre le rendu de la ville.
7. se déplacer vers le dossier `path-to-jscity-folder/js/` du logiciel.
8. démarrer l'application avec `node : node server.js`.
9. accéder à l'**URL** <http://localhost:8888> dans le navigateur.
10. sélectionner l'option "Example City" désignant le système exemple depuis la liste déroulante "Select the System" dans la barre horizontale en haut de la fenêtre.

---

10. **LOC** : Lines of Code

11. **NOV** : Number of Variables

12. <https://nodejs.org/en/>

13. <https://mariadb.org/download/>

14. <https://github.com/aserg-ufmg/JSCity>

11. attendre la fin du rendu de la ville.
12. commencer la visualisation et la navigation des artefacts de la ville logicielle générée (cf. [Figure 1.33](#))

```

1 ...
2 {
3   "host": "localhost",
4   "user": "<utilisateur_mysql>",
5   "password": "<mot_de_passe>",
6   "database": "jscity"
7 }
8 ...

```

Listing 1.2 – Les entrées du fichier config.json à modifier

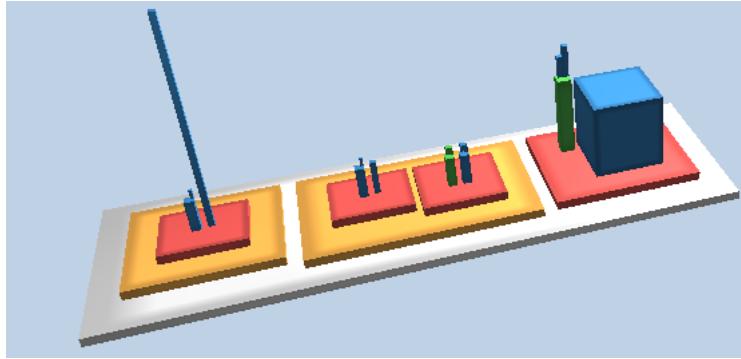


FIGURE 1.33 – Exemple d'une ville d'un système JavaScript avec JSCity

## Navigation et Interaction

Le mode de visualisation par défaut des villes est le mode "Orbital" fournissant les contrôles suivants :

1. enfoncez le clic gauche de la souris et déplacer celle-ci pour graviter autour de la ville.
2. utiliser la molette de la souris pour *zoomer* en avant/arrière.
3. utiliser les touches fléchées du clavier pour se déplacer dans les plans horizontal et vertical.

Un autre mode visualisation peut être utilisé pour naviguer la ville en première personne. Pour s'en servir, il suffit de sélectionner l'option « **First Person** » de la liste déroulante « **Control Type** » dans la barre horizontale en haut de la fenêtre. Ce mode fournit les contrôles suivants :

1. utiliser les touches fléchées (ou les touches W A S D) pour se déplacer dans le plan horizontal ou *zoomer* en avant/arrière.
2. utiliser les touches +, -, et \* respectivement pour augmenter/diminuer la vitesse de déplacement <sup>15</sup> ou retourner à la vitesse standard.

Par ailleurs, les artefacts de la ville peuvent être survolés pour afficher leurs mesures associées.

## Générer une ville logicielle pour son système JavaScript

Pour générer une ville logicielle pour son système JavaScript, les étapes suivantes doivent être effectuées :

1. mettre le dossier racine du projet dans un dossier parent (*e.g. systems/*)
2. copier le fichier `path-to-jscity-folder/js/backend/generator.js` dans le dossier racine du projet `systems/your-project/`

<sup>15</sup>. la vitesse de déplacement est affichée comme valeur du champ « "Speed" » de la barre horizontale en haut de la fenêtre

3. copier les fichiers `path-to-jscity-folder/js/config.json` et le dossier `path-to-jscity-folder/js/lib` dans le dossier parent `systems/` du projet.
4. vérifier que le serveur **MySQL** est actif, sinon le démarrer.
5. se déplacer vers le dossier `path-to-jscity-folder/js/` du logiciel.
6. démarrer l'application avec `Node.js` : `node server.js`
7. ouvrir une autre fenêtre de la console.
8. se déplacer vers le dossier racine du projet `systems/your-project/`
9. générer toutes les informations à insérer dans la base de données pour créer le modèle de la ville du projet, en parcourant tous les dossiers et sous-dossiers du projet et traitant les fichiers source JavaScript : `node generator.js systems/your-project -c "City Name"`
10. accéder à l'**URL** `http://localhost:8888` dans le navigateur.
11. sélectionner l'option **City Name** désignant la ville générée de votre système depuis la liste déroulante **Select the System** dans la barre horizontale en haut de la fenêtre.
12. attendre la fin du rendu de la ville.
13. commencer la visualisation et la navigation des artefacts de la ville logicielle générée de votre système.

**Remarque.**

Il faut prendre soin à ne générer que les artefacts provenant du code source propre au projet, et ignorer les fichiers source pouvant désigner des bibliothèques minifiées, des échantillons de code (*sample codes*), ... ne faisant pas partie de la structure du projet. Une bonne pratique sera ainsi de cibler explicitement le chemin du dossier de code source lors de la génération des informations des artefacts : `node generator.js systems/your-project/src/ -c "City Name"`

Par ailleurs, parfois un fichier source du système étudié peut contenir une fonction dont la définition n'est fermée que dans un autre fichier, engendrant des erreurs de *parsing*. Ce cas n'est malheureusement pas géré par **JSCity** ; il faut penser à réunir les fonctions dispersées, voire éviter cette pratique complètement dans les projets qu'on souhaite visualiser/analyser.

Enfin, voici quelques exemples de villes logicielles générées pour des projets JavaScript *open-source*.

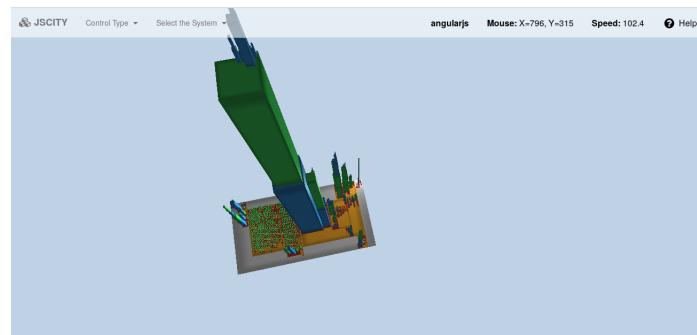


FIGURE 1.34 – La ville logicielle d'**Angularjs** avec **JSCity**

## Conclusion

**CodeCity** est un outil multiplateforme interactif pour la visualisation 3D et l'analyse des logiciels orientés-objet écrits en des langages conformes au métamodèle **FAMIX**, incluant Smalltalk, C++ et Java. Il permet la visualisation d'un logiciel à partir de modèles abstraits le décrivant, plutôt qu'à partir de son code source. Il est utilisé pour faciliter la compréhension, l'analyse, la rétro-ingénierie et le suivi de l'évolution des systèmes orienté-objet à grande échelle, en tant qu'une approche basée sur la métaphore *software city*.

**JSCity** est un outil interactif de visualisation 3D et d'analyse de logiciels créés en JavaScript. Il s'agit d'une implémentation orientée-JavaScript minimalistique de la métaphore *software city*,



FIGURE 1.35 – La ville logicielle du logiciel JavaScript **Prettier** avec **JSCity**

notamment pour le **paradigme fonctionnelle** de JavaScript, utilisant la bibliothèque JavaScript **three.js** pour le graphisme 3D.

Dans cette partie nous avons vu comment installer **JSCity**, comment générer une ville logicielle pour les systèmes JavaScript écrit dans le paradigme fonctionnel du langage et comment naviguer et interagir avec les villes logicielles créées. Nous l'avons trouvé comme outil incontournable pour la visualisation et la compréhension logicielle et que nous continuerons à utiliser dans le futur, afin d'estimer la complexité de nos projets et/ou de projets tiers que nous utiliserons.

### 1.3 Exercice 3 : Analyse d'approches en maintenance et évolution logicielle

**Choix :** *Bug or Not ? Bug Report Classification Using N-Gram IDF<sup>16</sup>.*

Dans cette partie du TP, nous discutons d'un modèle de classification automatique des rapports de bogues, basé sur une approche **n-grammes IDF**. Les rapports de bogues (*Bug reports*) sont utilisés par diverses tâches d'ingénierie logicielle, en particulier pour la maintenance logicielle (*e.g.* affectation de priorité et de sévérité des bogues, tri des bogues à traiter, …).

Toutefois, l'expérience prouve qu'environ la moitié des rapports de bogues sont souvent mal classés (*i.e.* un bogue signalé qui, en réalité, ne l'est pas) et par suite non fiables. Ceci est dû notamment à une mauvaise manipulation d'un **BTS<sup>17</sup>**, utilisé pour effectuer d'autres tâches que la trace de bogues (*e.g.* l'ajout de features, *refactoring*, …). La solution naïve consiste en l'inspection manuelle des rapports de bogues tracés par un **BTS**, une tâche assez chronophage révélant l'intérêt *a posteriori* de disposer d'une solution automatique.

Dans le cadre de cette problématique, TERDCHANAKUL *et al.* ont étudié l'état de l'art des solutions proposées pour la classification automatique des rapports de bogues. Parmi ces solutions, ils ont trouvé une solution basée sur la fouille de textes à l'échelle d'un mot (*a word-grain text-based classification*).

Par ailleurs, ils ont étudié une autre approche plus performante basée sur la fouille de données, consistant à effectuer une **modélisation thématique<sup>18</sup>** (*topic modeling*) des rapports, en appliquant les techniques de modélisation **LDA<sup>19</sup>** et **HDP<sup>20</sup>**. Bien que cette dernière solution soit assez performante et répandue dans divers applications de l'ingénierie logicielle (*e.g. clustering* de documents, localisation concept/*feature*, analyse d'évolution logicielle, recherche, …), elle se voit limitée par l'absence d'une approche systématique recommandée pour la sélection des thèmes.

Enfin, ils ont adressé une approche hybride combinant la fouille de textes et la fouille de données par l'application de la technique "*data grafting*", mais se sont également aperçus de ses limitations.

Par conséquent, TERDCHANAKUL *et al.* proposent dans cet article un modèle de classification basé sur les **n-grammes IDF**. D'une part, **IDF** (**Inverse Document Frequency**) désigne une mesure statistique révélant la quantité d'information fournie et souvent utilisé comme mesure de poids lors de l'extraction d'informations. D'autre part, **n-grammes** est une extension théorique d'**IDF** permettant de gérer des phrases de *n* termes (au lieu d'un seul mot à la fois avec **IDF**).

Pour construire leurs modèles, les chercheurs adoptent la méthodologie suivante (*cf. Figure 1.36*) :

**Parsing :** analyse syntaxique des rapports de bogues.

**Text Preprocessing :** pré-traitements des rapports de bogues (*e.g.* suppression des caractères réservés par les langages de programmation).

**Applying N-Gram IDF :** utilisation de l'outil **ngweight<sup>21</sup>** définissant un schéma d'attribution de poids **n-grammes** sur les rapports pré-traités, afin obtenir un dictionnaire des termes-clés.

**Feature extraction :**

1. filtrage des termes n'apparaissant qu'une seule fois dans un rapport ;

---

16. [https://www.researchgate.net/publication/320883291\\_Bug\\_or\\_Not\\_Bug\\_Report\\_Classification\\_Using\\_N-Gram\\_IDF](https://www.researchgate.net/publication/320883291_Bug_or_Not_Bug_Report_Classification_Using_N-Gram_IDF)

17. Bug Tracking System

18. une approche consistant à trouver les thèmes communs aux documents analysés.

19. Latent Dirichlet Allocation

20. Hierarchical Dirichlet Process

21. <https://github.com/iwnsew/ngweight>

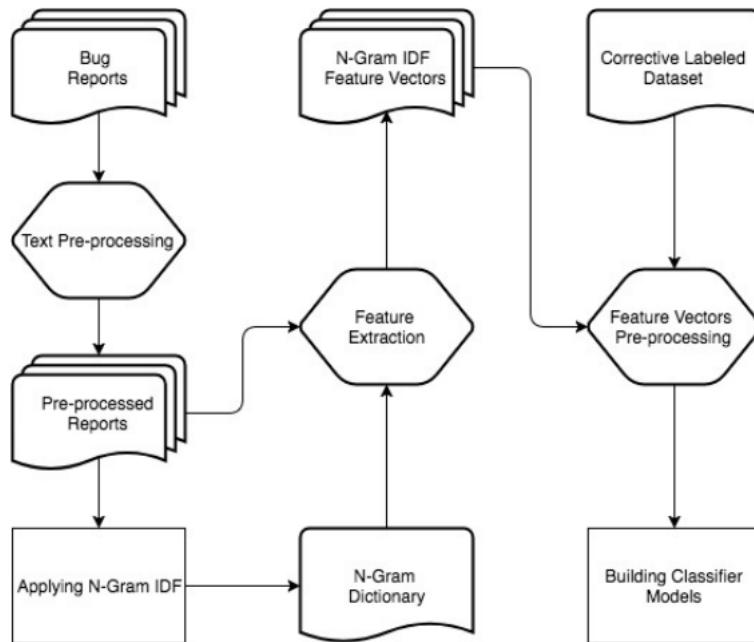


FIGURE 1.36 – Aperçu du processus de construction du modèle de classification

2. création des vecteurs de *features* à partir du dictionnaire des termes et le corpus des rapports pré-traités, en calculant la fréquence de chaque terme dans chaque rapport.

**Feature Vector Preprocessing :** filtrage des *features* par la suppression des termes ayant un impact minimal sur la classification, en utilisant les techniques de sélection de *features* suivantes :

**Correlation-based Feature Selection :** sélection du sous-ensemble des *features* fortement corrélées à la classification mais non corrélées entre elles, en utilisant la librairie Weka<sup>22</sup>, appliquée sur une **validation croisée sur 10 itérations** (*10-fold cross-validation setup*).

**Chi-squared stats :** sélection des *features* par l’application d’une méthode statistique mesurant le niveau du *fitting* entre les *features* et la classification, en utilisant la librairie python **scikit-learn**<sup>23</sup>, appliquée lors de la construction des ensembles d’entraînement et de test.

**Build Classifier Models :** utilisation d’une combinaison de techniques de pré-traitement et de classification de données différentes, notamment la **régression logistique** (*Logistic regression*) et les **forêts d’arbres décisionnels** (*Random Forest*).

Pour entraîner et tester leurs modèles, les chercheurs collectent trois jeux de données depuis le **BTS JIRA**<sup>24</sup> utilisé par les projets *open-source* (**HTTPClient**, **Jackrabbit**, **Lucene**), provenant du laboratoire les accueillant. Un 4<sup>e</sup> jeu de données (**Cross Project**) est construit par la combinaison des trois jeux de données précédents.

Pour évaluer leurs modèles, les chercheurs utilisent la **moyenne harmonique** (*F-score*), combinant les mesures **”rappel”** et **”précision”**. De plus, ils préparent deux environnements d’évaluation différents :

1. un environnement d’ensembles obtenus par **validation croisée sur dix itérations**<sup>25</sup> : il s’agit de partitionner chaque jeu de données en dix partitions aléatoires de tailles égales, telles que, pour chaque itération, neuf sont utilisées pour l’entraînement et une pour le test, de manière à avoir utilisé toutes les partitions exactement une seule fois pour le test au cours du processus. Enfin ils rapportent la valeur moyenne du *F-score*.

22. <https://www.cs.waikato.ac.nz/ml/weka/>

23. <https://scikit-learn.org/stable/>

24. <https://www.atlassian.com/software/jira>

25. *10-fold cross-validation*

2. un environnement d'ensembles d'entraînement et de test basé sur un **partitionnement chronologique** : il s'agit d'utiliser les 90% des rapports les plus anciens comme ensemble d'entraînement et les 10% les plus récents comme ensemble de test. Enfin ils rapportent la valeur du *F-score*.

De plus, les chercheurs construisent des modèles de classification automatique obtenus par une **modélisation thématique**, afin de pouvoir comparer les performances des deux approches.

Les expérimentations ont permis de démontrer que les techniques de sélection des *features* permettent de passer de [58.000; 530.000] **n-grammes** à [40; 200] **n-grammes** influants sur la classification. De plus, ils prouvent que l'approche par **n-grammes IDF** permet de séparer les rapports de bogues des autres demandes. De fait, cette approche permet d'obtenir de meilleurs résultats en comparaison avec ceux obtenus via la **modélisation thématique**, et ceci pour les deux environnements d'évaluation adoptés.

Quelques nuances sont néanmoins à apporter à leurs résultats :

- les labels de leurs jeux de données dépendent des précédentes recherches, souvent obtenues par inspection manuelle basée sur une perspective individuelle non systématique.
- tous les projets analysés sont *open-source*, utilisent le même **BTS** (*i.e.* JIRA) et sont écrits en **Java**.

Dans la suite de leurs recherches, TERDCHANAKUL *et al.* veulent améliorer la performance de leur modèle et l'étendre à un corpus multi-classes. En outre, ils souhaitent généraliser leur modèle à des jeux de données provenant d'autres projets, utilisant différents **BTS** et écrits en différents langages de programmation. Enfin, ils souhaitent étendre la portée de leurs modèles au-delà des **BTS**, pour inclure d'autres activités d'ingénierie logicielle dans leur analyse.

# Chapitre 2

## TP2 - Analyse statique et dynamique

### 2.1 Parties 1 et 2 - AST d'Eclipse JDT

#### 2.1.1 Le modèle Java (*Java Model*) du JDT

Tout projet Java peut être représenté en interne par un modèle léger et tolérant aux pannes, désignant l'arbre enraciné par le dossier du projet contenant la totalité de ses éléments.

Le modèle obtenu n'est pas si riche et verbeux qu'un modèle représenté par l'AST du JDT, mais offre quand même un ensemble de traitements de base, tel que la visualisation de la vue **Package Explorer** d'Eclipse.

L'ensemble des concepts définissant les différents éléments du modèle Java sont localisés dans le plugin `org.eclipse.jdt.core`.

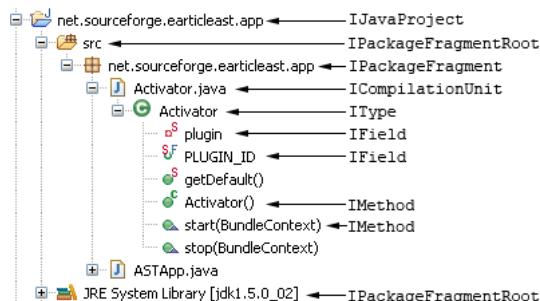


FIGURE 2.1 – Un exemple d'un projet visualisé dans Package Explorer grâce au modèle Java du JDT

```
1 // getting the root workspace
2 IWorkspaceRoot root = ResourcesPlugin.getWorkspace().getRoot();
3
4 // getting the project "someJavaProject" from the root workspace
5 IProject project = root.getProject("someJavaProject");
6
7 // opening the java project
8 project.open(null /*IProgressMonitor*/);
9
10 // getting the java project handle
11 IJavaProject javaProject = JavaCore.create(project);
12
```

```

13 // getting a type in the java project
14 IType lwType = javaProject.findType("some.package.somewhere.Type");
15
16 // getting the compilation unit corresponding to the type
17 ICompilationUnit lwCompilationUnit = lwType.getCompilationUnit();

```

Listing 2.1 – Exemple de récupération d'une unité de compilation désignant un type dans le projet

### 2.1.2 L'AST du JDT

L'AST du JDT fournit une API permettant de modifier, créer, lire et supprimer du code source indirectement en manipulant ses noeuds, en traitant en entrée des unités de compilation désignant du code source (i.e. `ICompilationUnit` du modèle Java). Il s'agit d'un outil de base utilisés par plusieurs fonctionnalités d'Eclipse IDE (*refactoring, quick fix, quick assist, ...*).

Le flux de travail lors de l'utilisation de l'AST peut être résumé de la manière suivante :

1. fournir du code source en entrée sous forme d'un fichier Java ou un tableau de caractères (`char[]`);
2. parser le code source en utilisant une instance de `ASTParser` pour obtenir un AST et éventuellement des informations calculées supplémentaires (i.e *bindings*) sur ses noeuds;
3. manipuler l'AST directement ou indirectement à travers une instance de `ASTRewrite`;
4. appliquer les changements de l'AST au code source d'origine via l'interface *wrapper* du code source `IDocument`.

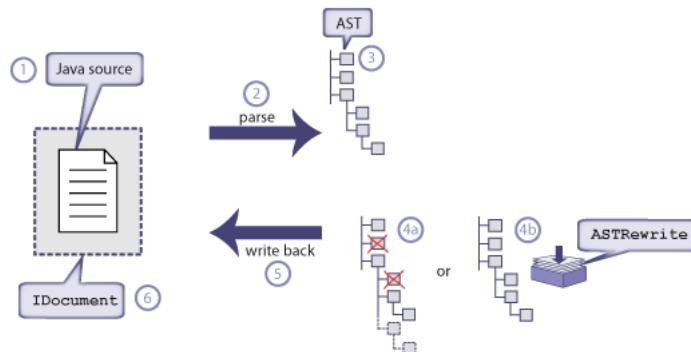


FIGURE 2.2 – Le flux de travail avec l'AST du JDT

#### Quelques classes de base

**ASTNode** : superclasse de tous les noeuds de l'AST.

**ASTParser** : classe définissant un parseur de code source et fournissant un AST.

**TypeDeclaration** : un noeud désignant la déclaration d'un type (i.e. une classe, une interface, une énumération).

**MethodDeclaration** : un noeud désignant la déclaration d'une méthode.

**VariableDeclaration** : un noeud désignant la déclaration d'une variable.

**SimpleName** : un noeud désignant n'importe quel `String` ne désignant pas un mot-clé Java, `true`, `false`, ou `null`.

**ASTVisitor** : la superclasse de toutes les classes implémentant le patron de conception *Visitor*, afin de visiter les noeuds de l'AST.

## Flux de travail avec les visiteurs des nœuds de l'AST

**preVisit(node)** : visiter le nœud générique de l'AST (*i.e.* une instance de la classe `ASTNode`) avant de visiter son type spécifique.  
**visit(concreteNode)** : visiter le nœud spécifique de l'AST (*i.e.* une instance d'une sous-classe d'`ASTNode`).  
**récursion** : visiter les noeuds enfants du nœud visité, si la méthode `visit()` retourne `true`.  
**endVisit(concreteNode)** : terminer la visite du nœud spécifique de l'AST.  
**postVisit(node)** : terminer la visite du nœud générique de l'AST.

```
1  public class MethodDeclarationVisitor extends ASTVisitor {  
2  
3     private ArrayList<MethodDeclaration> methods = new ArrayList<>();  
4  
5     @Override  
6     public boolean visit(MethodDeclaration node) {  
7         methods.add(node);  
8         return super.visit(node);  
9     }  
10  
11    public ArrayList<MethodDeclaration> getMethods(){return methods;}  
12 }
```

Listing 2.2 – Exemple d'un visiteur collectant les déclarations des méthodes d'une classe

## Les propriétés structurelles d'un nœud de l'AST

Les propriétés structurelles d'un nœud de l'AST sont des métadonnées sur le nœud accessibles via des descripteurs, instances de la classe abstraite `StructuralPropertyDescriptor` ou de l'une de ses spécialisations `SimplePropertyDescriptor`, `ChildPropertyDescriptor` ou `ChildListPropertyDescriptor`, selon la valeur de la propriété.

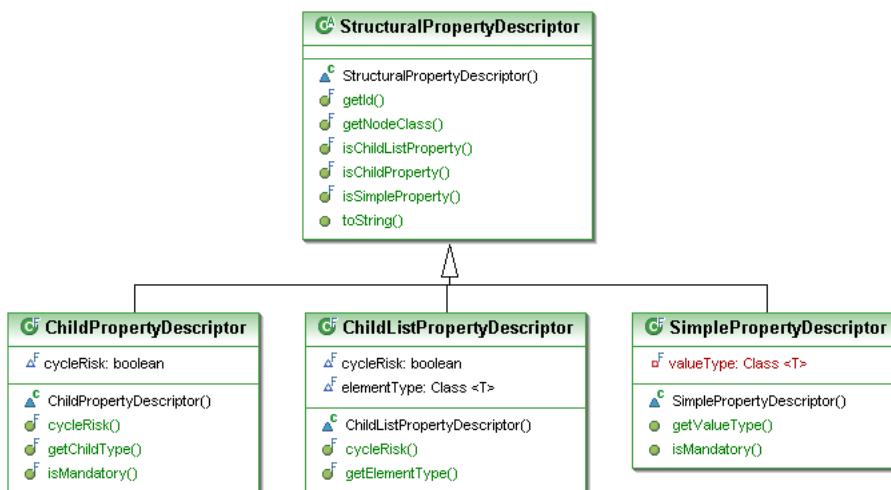


FIGURE 2.3 – Hiérarchie des descripteurs des propriétés structurelles d'un nœud de l'AST

## Récupérer des informations sur les nœuds de l'AST

On peut récupérer des informations sur les nœuds de l'AST :

- en accédant à des méthodes dédiées pour les noeuds spécifiques : *e.g.* les méthodes `getName()` et `thrownExceptions()` pour récupérer le nom et la liste des exceptions levées, respectivement, d'un nœud désignant une déclaration d'une méthode (*i.e.* une instance de la classe `MethodDeclaration`).
- en utilisant des méthodes génériques d'accès aux propriétés structurelles des noeuds : *e.g.* les méthodes `getStructuralProperty(propertyDescriptor)` et `structuralPropertiesForType()` retournant, respectivement, la valeur d'une propriété structurelle pointée par le descripteur « `propertyDescriptor` » et la liste des propriétés structurelles du noeud générique de l'AST courant.



FIGURE 2.4 – Exemple des propriétés structurelles d'une déclaration de méthode

### Les *bindings* d'un nœud de l'AST

Des informations sur certains noeuds de l'AST peuvent être calculées via le mécanisme des liaisons (*bindings*). Il s'agit d'heuristiques exécutées lors de la construction de l'AST via le parseur pour calculer des informations diverses.

En pratique, certaines classes définissant des noeuds spécifiques de l'AST disposent au moins de la méthode `resolveBinding()` retournant le *binding* du type du noeud. D'autres classes disposent davantage de méthodes de résolution de liaisons diverses, telles que la classe `MethodInvocation` désignant l'invocation d'une méthode et disposant de la méthode `resolveMethodBinding()` retournant le *binding* de la déclaration de la méthode invoquée.

```
1 int i = 7;
2 System.out.println("Hello!");
3 int x = i * 2;
```

Listing 2.3 – Exemple d'un bout de code sur lequel la résolution des liaisons est activée

#### Remarque.

Le service de résolution des *bindings* doit être explicitement activé lors de la création du parseur avant de parser le code source, via l'invocation de la méthode du parseur `setResolveBindings(true)`.

### Modification des noeuds de l'AST

Parfois nous aurons besoin d'introduire des changements au code source analysé à travers l'AST. Pour ce faire, soit on effectue des modifications directement sur l'AST, soit on délègue la gestion des modifications à une instance de `ASTRewrite`.



FIGURE 2.5 – Exemple du *binding* de la référence de la variable *i* à la ligne 3 du listing 2.3

```

1 // creating an instance of rewrite on the CompilationUnit "unit"'s AST
2 ASTRewrite rewrite = ASTRewrite.create(unit.getAST());
3
4 VariableDeclarationStatement statement =
5     createNewVariableDeclarationStatement(
6 manager, ast); // creating a new variable declaration statement
7
8 // getting the first reference index of the block in which to add the
9 // statement
10 int firstReferenceIndex = getFirstReferenceIndex(manager, block);
11
12 // recover the list of statements of the block to rewrite
13 ListRewrite statementsListRewrite = rewrite.getListRewrite(
14 block, Block.STATEMENTS_PROPERTY);
15
16 // inserting the statement into the list of statements at the 1st reference
17 // index
18 statementsListRewrite.insertAt(statement, firstReferenceIndex, null);

```

Listing 2.4 – Exemple d'ajout d'un noeud à un AST via ASTRewrite

```

1 ASTRewrite rewrite = ASTRewrite.create(unit.getAST());
2 // renaming the name of the method invocation
3 rewrite.set(methodInvocation, MethodInvocation.NAME_PROPERTY, newName, null);
4
5 // same effect using a different method
6 rewrite.replace(methodInvocation.getName(), newName, null);

```

Listing 2.5 – Exemple de modification d'un noeud de l'AST via ASTRewrite

```

1 // enable modification recording at the root of the AST
2 unit.recordModifications();
3 // ...
4 VariableDeclarationStatement statement =
5     createNewVariableDeclarationStatement(
6 manager, ast); // creating a new variable declaration statement
7
8 // getting the first reference index of the block in which to add the
9 // statement
10 int firstReferenceIndex = getFirstReferenceIndex(manager, block);

```

```

10 // adding the statement at the beginning of the block
11 block.statements().add(firstReferenceIndex, statement);

```

Listing 2.6 – Exemple de modification directe d'un noeud de l'AST

### Répercuter les modifications des nœuds de l'AST au code source

Les changements effectués sur l'AST sont répercutés au code source d'origine par le biais de l'interface `IDocument` et ses implémentations.

```

1 /*
2 * document: the source code file parsed by ASTParser
3 * options: source code formatter options, (null for default options)
4 */
5
6 // invoked on the ASTRewrite instance if used
7 TextEdit rewriteAST(IDocument document, Map options);
8
9 // invoked on the CompilationUnit if the AST is directly modified
10 TextEdit rewrite(IDocument document, Map options);

```

Listing 2.7 – méthodes de répercussion des modifications de l'AST au code source

```

1 // get a file buffer manager
2 ITextFileBufferManager bufferManager =
3     FileBuffers.getTextFileBufferManager();
4
5 // get the path of the source file "unit" (CompilationUnit)
6 IPath path = unit.getJavaElement().getPath();
7
8 try{
9     /* connect a path of a file buffer manager
10    * after this call, the document of the file described by "path"
11    * can be obtained and modified
12    */
13    bufferManager.connect(path, null);
14
15    // retrieve the text file buffer
16    ITextFileBuffer textFileBuffer = bufferManager.getTextFileBuffer(path);
17
18    // ask the buffer for a working copy of the document
19    IDocument document = textFileBuffer.getDocument();
20
21    /*rewrite changes of AST into the document*/
22
23    // commit changes to the underlying file
24    textFileBuffer.commit(null /*ProgressMonitor*/, false /*Overwrite*/);
25
26 } catch (Exception e){
27     /*handle exception*/
28 }
29
30 finally {
31     /* disconnect the file buffer manager from the path
32     * after this call, the document of the file described by "path"
33     * should no longer be modified
34     */
35     bufferManager.disconnect(path, null);
36 }

```

Listing 2.8 – Exemple générique de répercussion des modifications d'un AST au code source

```

1 CompilationUnit astRoot = ...; // current compilation unit
2
3 // creating an instance of rewrite on the CompilationUnit "astRoot"'s AST
4 ASTRewrite rewrite = ASTRewrite.create(astRoot.getAST());
5
6 // retrieving the body of the first method of the first class in the unit
7 Block block = ((TypeDeclaration) astRoot.types().get(0))
8   .getMethods()[0].getBody();
9
10 // retrieve the list of statements of the body
11 ListRewrite listRewrite = rewrite.getListRewrite(block,
12       Block.STATEMENTS_PROPERTY);
13
14 // creating a comment
15 Statement placeHolder = rewrite.createStringPlaceholder("//mycomment",
16 ASTNode.EMPTY_STATEMENT);
17
18 // inserting the comment prior to the method's body
19 listRewrite.insertFirst(placeHolder, null);
20
21 // retrieving the text-based modifications introduced to the AST of "astRoot"
22 TextEdit textEdits = rewrite.rewriteAST(document, null);
23
24 // committing the changes to the document
25 textEdits.apply(document);

```

Listing 2.9 – Exemple d'ajout d'un commentaire au début du corps d'une méthode

### 2.1.3 Les processeurs utilisant l'AST du JDT

Dans le cadre de ce TP, nous avons repris le code de la classe utilisant l'`ASTParser` qui nous a été fourni et nous l'avons restructuré pour avoir une version orientée-objet dans la classe `Parser`. `Parser` est utilisée par tous les processeurs cherchant à visiter les noeuds d'un AST pour effectuer des tâches d'analyse spécifiques. En effet, nous avons défini un processeur de base `BaseProcessor` utilisant une instance de `Parser` et factorisant sa configuration. Ce processeur est spécialisé par tous les processeurs spécifiques.

Pour la première partie du TP2, nous avons défini le processeur `InfoProcessor` définissant des méthodes permettant de visiter les noeuds des classes, de leurs attributs, de leurs méthodes et des méthodes invoquées dans ces dernières et d'extraire les informations requises (cf. Figure ??).

Pour la deuxième partie du TP2, nous avons défini le processeur `StatsProcessor` définissant des méthodes permettant d'extraire les statistiques requises sur les éléments d'un projet (cf. Figure 2.7).

Les deux processeurs sont testés sur un projet contenant des *sample codes* illustrant différents patrons de conception (`Composite`, `Visitor`, `Singleton`, ...).

```

1 Class: composite.Song
2 Superclass: SongComponent
3 Attributes:
4   private String songName
5   private String bandName
6   private int releaseYear
7 Methods:
8   @Override public String Song::getSongName()
9   @Override public String Song::getBandName()
10  @Override public int Song::getReleaseYear()
11  @Override public void Song::displaySongCompone
12
13 Class: composite.DiscJockey
14 Superclass: N/A
15 Attributes:
16   private SongComponent songList
17 Methods:
18   public SongComponent DiscJockey::getSongList()
19   public void DiscJockey::displaySongList()
20
21 Class: composite.SongListGenerator
22 Superclass: N/A
23 Attributes:
24 Methods:
25   public static void SongListGenerator::main(Str
26
27 Class: composite.SongComponent

```

```

1 Class: composite.Song
2 Superclass: SongComponent
3 Methods:
4   @Override public String Song::getSongName()
5   @Override public String Song::getBandName()
6   @Override public int Song::getReleaseYear()
7   @Override public void Song::displaySongComponentInfo() invokes:
8     StringBuilde
9       Song::getSongName()
10      StringBuilde
11        Song::getBandName()
12        StringBuilde
13          Song::getReleaseYear()
14          System.out::println(buf.toString())
15        StringBuilde
16
17 Class: composite.DiscJockey
18 Superclass: N/A
19 Methods:
20   public SongComponent DiscJockey::getSongList()
21   public void DiscJockey::displaySongList() invokes:
22     songList::displaySongComponentInfo()
23
24 ...

```

(a) Aperçu des résultats orientés-classes obtenus par (b) Aperçu des résultats orientés-méthodes obtenus par  
InfoProcessor InfoProcessor

```

1 Nombre de packages : 7
2 Nombre de classes : 36
3 Nombre de méthodes : 115
4 Lignes de code : 1510
5 Moyenne méthodes/classe : 3.1944444444444446
6 Moyenne attributs/classe : 1.0833333333333333
7 Moyenne lignes/méthode : 13.130434782608695
8 nombre maximal de paramètres par rapport à toutes les méthodes : 3
9
10 10% classes avec plus grand nombre de méthodes :
11    state.ATMMachine
12    visitor.TaxVisitor
13    factory.EnemyShip
14
15 10% classes avec plus grand nombre d'attributs :
16    state.ATMMachine
17    observer.StockObserver
18    singleton.Singleton
19
20 10% classes avec plus grand nombre de méthodes et d'attributs :
21    state.ATMMachine
22
23 classes avec plus que 3 méthodes :
24   composite.Song
25   composite.SongComponent
26   composite.SongGroup
27   singleton.Singleton
28   visitor.TaxVisitor
29 ...
30
31 10% des méthodes qui possèdent le plus grand nombre de statements par classe :
32   adapter.EnemyRobot : smashWithHands
33   observer.GrabStocks : main
34   state.HasCard : ejectCard
35   visitor.TaxHolidayVisitor : TaxHolidayVisitor
36   composite.Song : displaySongComponentInfo
37   factory.RocketEnemyShip : RocketEnemyShip
38 ...

```

FIGURE 2.7 – Aperçu des résultats obtenus par StatsProcessor

#### 2.1.4 Le graphe d'appel statique

Pour construire le graphe d'appel statique d'un projet, nous avons défini tout d'abord un graphe d'appel de base `AbstractCallGraph` définissant l'interface commune à tous les graphes d'appel.

Le graphe d'appel statique est modélisé dans la classe `StaticCallGraph`, héritant l'interface définie par `AbstractCallGraph` et utilisant une instance de `BaseProcessor` pour définir les traitements nécessaires pour la construction du graphe d'appel statique en visitant les noeuds de l'**AST** obtenus par le parser de `BaseProcessor`. Pour ce faire, nous avons adopté une approche récursive consistant à construire le graphe d'appel statique de chaque unité de compilation et de réaliser leur union pour obtenir le graphe d'appel statique de tout le projet. Chaque appel est associé à un nombre désignant le nombre de fois que cet appel est réalisé statiquement dans le code du projet (*cf.* Figure 2.8).

```

1   factory.EnemyShipTesting::doStuffEnemy:
2   |---> enemy::followHeroShip (1 fois)
3   |---> enemy::enemyShipShoots (1 fois)
4   |---> enemy::displayEnemyShip (1 fois)
5   state.HasCard::requestCash:
6   |---> System.out::println (1 fois)
7   adapter.EnemyTank::driveForward:
8   |---> System.out::println (1 fois)
9   |---> generator::nextInt (1 fois)
10  state.CorrectPin::requestCash:
11  |---> this.context::getCashInMachine (3 fois)
12  |---> this.context::setATMState (1 fois)
13  |---> this.context::setCashInMachine (1 fois)
14  |---> System.out::println (2 fois)
15  |---> this::ejectCard (1 fois)
16  |---> this.context::getNoCashState (1 fois)
17  observer.StockGrabber::register:
18  |---> observers::add (1 fois)
19  observer.StockObserver::update:
20  |---> observer.StockObserver::displayPrices (1 fois)
21  visitor.Necessity::accept:
22  |---> visitor::visit (1 fois)
23  visitor.TaxVisitor::visit:
24  |---> visitor.TaxVisitor::computeTax (3 fois)
25  singleton.GetTheTiles::run:
26  |---> instance::getLetterslist (1 fois)
27  |---> instance::getTiles (1 fois)
28  |---> System::identityHashCode (1 fois)
29  |---> Singleton::getInstance (1 fois)
30  |---> System.out::println (4 fois)

```

FIGURE 2.8 – Aperçu des résultats obtenus par `StatsProcessor`

Les graphe d'appel statique est testé sur le même projet précédent.

## 2.2 Partie 3 - Étude de l'outil Spoon

### 2.2.1 Introduction

Spoon est une librairie *open-source* permettant :

1. l'analyse et la transformation de code source Java ;
2. l'analyse de bytecode Java après sa décompilation ;
3. la transpilation (*e.g.* Java → JavaScript)
4. ...

### 2.2.2 Installation en tant qu'un *plugin* Maven

Créer un nouveau projet Maven sous Eclipse. Afin d'utiliser Spoon, il faut ajouter les lignes suivantes dans le fichier de gestion des dépendances Maven `pom.xml` :

```
<dependencies>
```

```

2 <dependency>
3   <groupId>fr.inria.gforge.spoon</groupId>
4   <artifactId>spoon-core</artifactId>
5   <version>8.0.0</version>
6 </dependency>
7 </dependencies>

```

### 2.2.3 Le métamodèle de Spoon

Spoon fournit un métamodèle Java à grain fin permettant d'accéder à n'importe quel élément en lecture/écriture. Tous les éléments du métamodèle sont modélisés par des interfaces dont les noms commencent par `Ct` (*Compile-time*) dans une hiérarchie d'héritage multiple. Ils peuvent être répartis en trois catégories :

**éléments structurels** : les éléments de déclarations d'un programme (*e.g. classes, interfaces, énumérations, variables, méthodes, variables, ...*) (cf. Figure 2.9).

**éléments exécutables** : les éléments exécutables de Java (*e.g. les corps de méthodes/constructeurs, invocations de méthodes/constructeurs, les statements, les expressions, ...*) (cf. Figure 2.10).

**éléments de référence** : des éléments désignant des références de type (cf. Figure 2.11).

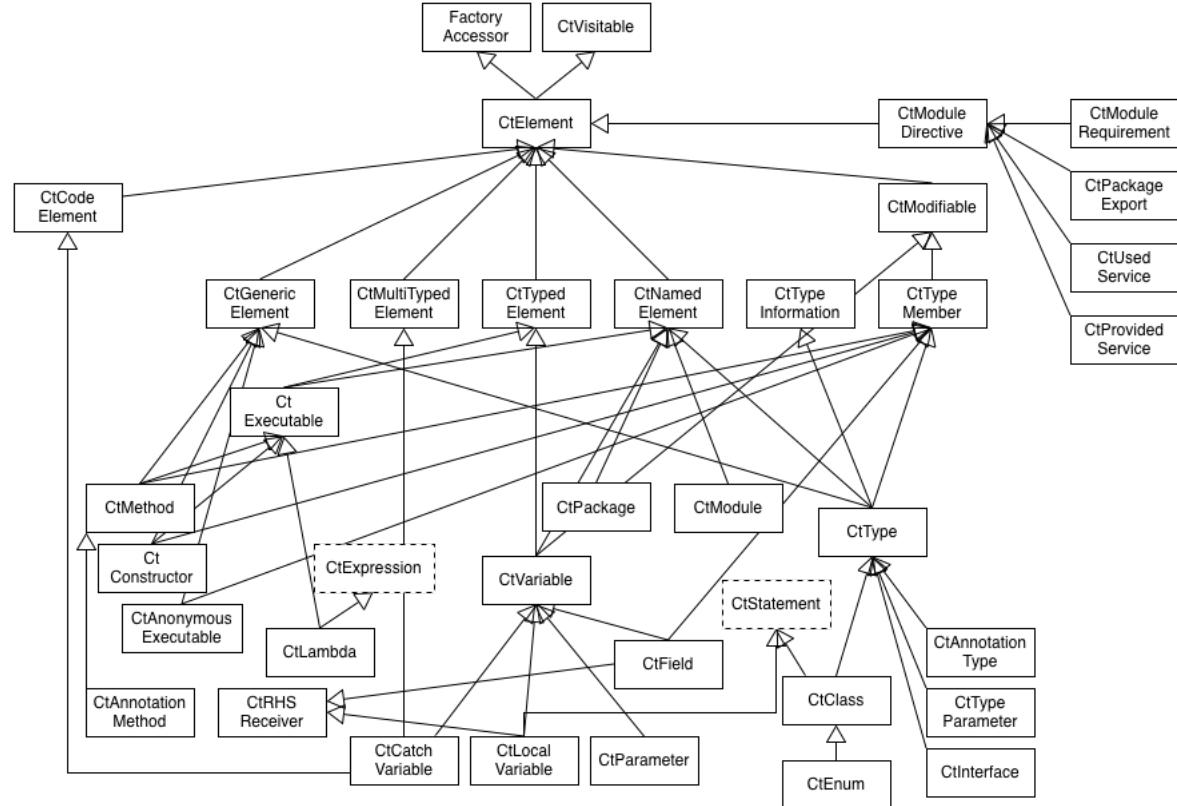


FIGURE 2.9 – Éléments structurels du métamodèle Spoon

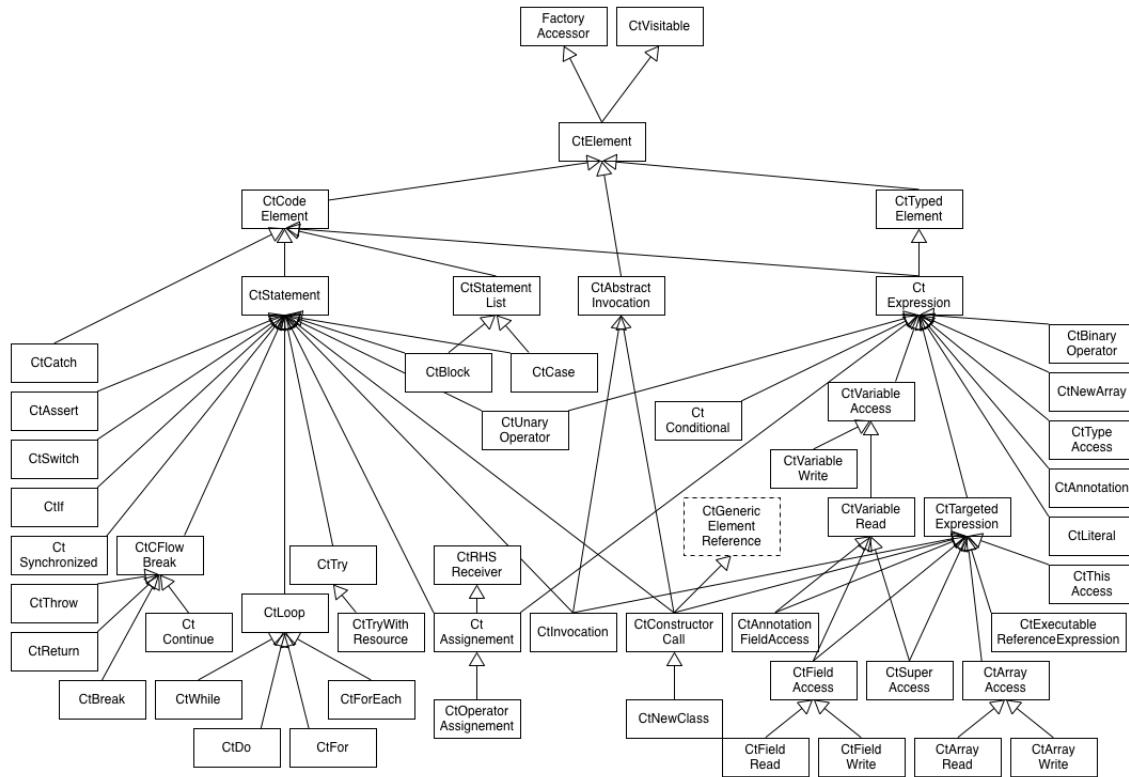


FIGURE 2.10 – Éléments exécutables du métamodèle Spoon

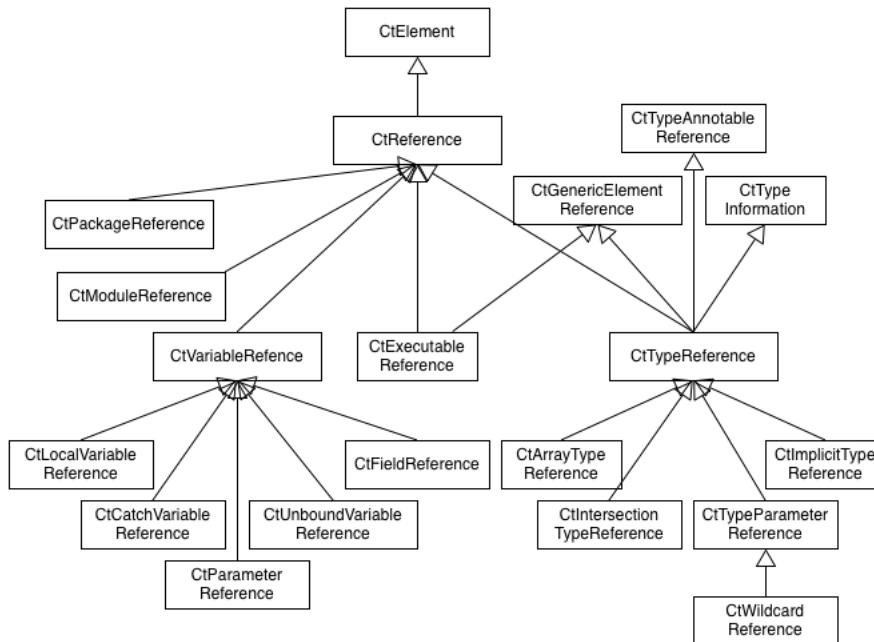


FIGURE 2.11 – Éléments de référence du métamodèle Spoon

## 2.2.4 Les références

Le mécanisme de référencement de type désigne le référencement d'éléments qui ne sont pas forcément réifiés au sein du métamodèle, comme ils peuvent appartenir à des bibliothèques tierces. Ceci permet une flexibilité lors de la construction/modification du modèle du programme en cours d'analyse.

Par exemple, référencer un objet de type `String` ne désigne pas le modèle compilable de `String.java`, étant donné que le code source de `String.java` ne fait pas partie (*en général*) du programme en cours d'analyse.

Pour récupérer la référence d'un type cible et le type ciblé d'une référence, on utilise, respectivement, les méthodes `CtType#getType()` et `CtTypeReference#getTypeDeclaration()`.

La résolution des références se fait lors de la construction du modèle et ne cible que les éléments dont le code source est fourni en entrée de `Spoon`. Cette résolution est faible puisque les cibles des références ne doivent pas nécessairement exister au préalable.

### 2.2.5 Le processus standard d'utilisation de Spoon

1. construire le modèle `Spoon` du projet à analyser ;
2. analyser et effectuer des *queries* sur les parties pertinentes du projet ;
3. transformer le code source qui doit être transformé ;
4. fournir un code source transformé du projet.

### 2.2.6 Afficher le modèle Spoon d'un code source Java

```
java -cp <spoon-jar> spoon.Launcher -i <class>.java --gui --noclasspath
```

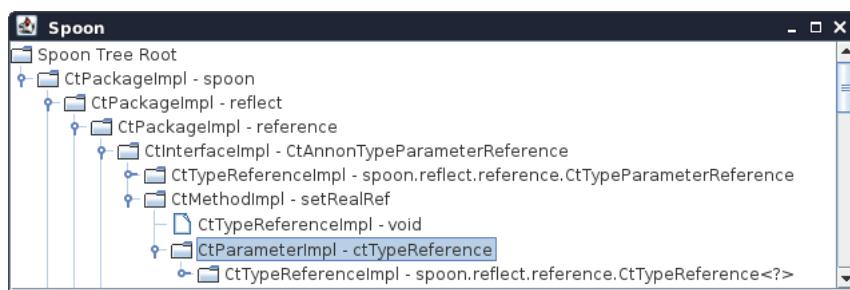


FIGURE 2.12 – Exemple d'un modèle Spoon d'une unité de code Java

### 2.2.7 Les *factories*

Lors de la conception et l'implémentation des transformations, nous aurons besoin de créer des implémentations des interfaces d'éléments fournies, les initialiser et les ajouter au modèle construit. Pour ce faire, `Spoon` offre une hiérarchie d'usines (**Factories**) où chaque usine est destinée à la création de noeuds spécifiques du modèle `Spoon`.

L'interface d'usine de base `Factory` fournit des points d'accès aux usines spécialisées telles que :

**Class()** : fournit l'accès à l'usine `ClassFactory` spécialisée pour l'usinage des classes.

**Constructor()** : fournit l'accès à l'usine `ConstructorFactory` spécialisée pour l'usinage des constructeurs.

**Field()** : fournit l'accès à l'usine `FieldFactory` spécialisée pour l'usinage des attributs.

**Method()** : fournit l'accès à l'usine `MethodFactory` spécialisée pour l'usinage des méthodes.

**Code()** : fournit l'accès à l'usine `CodeFactory` spécialisée pour l'usinage des éléments exécutables.

**Type()** : fournit l'accès à l'usine `TypeFactory` spécialisée pour l'usinage des types.

En outre, elle fournit des méthodes d'usinage génériques telles que :

`createClass()` : une méthode générique permettant de créer un nœud vide désignant une classe.

`createField()` : une méthode générique permettant de créer un nœud vide désignant un attribut.

`createMethod()` : une méthode générique permettant de créer un nœud vide désignant une méthode.

## 2.2.8 Les *getters/setters* standards

En utilisant la réflexion, Spoon permet de récupérer/modifier différents nœuds du modèle en employant des *getters/setters* appropriés pour chaque type de nœud, utilisant des critères de recherche/modification différents.

Par exemple, pour récupérer les constructeurs, les méthodes et les attributs d'une classe, nous utiliserons, respectivement, `CtClass#getConstructors()`, `CtType#getMethods()` et `CtType#getFields()`.

D'autre part, si nous souhaitons récupérer un attribut par son nom, nous utiliserons `CtType#getField(String name)`, par exemple.

## 2.2.9 Les filtres

Le principe des filtres est de récupérer des noeuds satisfaisant des prédictats bien définis. Certaines implémentations de filtres sont fournies par Spoon, telles que les filtres de types (`TypeFilter<T> typeClass`) et d'annotations (`AnnotationFilter<? extends java.lang.Annotation> typeAnnotation`). Toutefois, une implémentation personnalisable peut être fournie par l'utilisateur par l'extension de la classe `AbstractFilter<E extends CtElement>`, superclasse abstraite de tous les filtres Spoon. Cette classe implémente l'interface `Filter<E extends CtElement>` et il suffit de fournir une implémentation de sa seule méthode, `boolean matches(E element)`.

### Exemple

```
1 // collecting all assignments of a method body
2 list1 = methodBody.getElements(new TypeFilter(CtAssignment.class));
3
4 // collecting all deprecated classes
5 list2 = rootPackage.getElements(new AnnotationFilter(Deprecated.class));
6
7 // a custom filter to select all public fields
8 list3 = rootPackage.filterChildren(
9     new AbstractFilter<CtField>(CtField.class){
10         @Override
11         public boolean matches(CtField field){
12             return field.getModifiers().contains(ModifierKind.PUBLIC);
13         }
14     }).list();
```

Listing 2.10 – Exemple d'utilisation de filtres Spoon

## 2.2.10 Les *queries*

À partir de Spoon 5.5, on a introduit les *queries*, désignant un mécanisme de filtrage plus sophistiqué que les filtres classiques. En particulier, elles peuvent être enchaînées, réutilisées sur différents noeuds et rédigées par le biais de lambdas.

Lors de l'enchaînement des *queries* :

- si le résultat d'une *query* est non nul, il sera passé en entrée de la *query* suivante dans la chaîne.
- si le résultat d'une *query* est itérable, chacun de ses éléments constituera une entrée différente à la *query* suivante.

Il existe différentes manières d'évaluation d'une *query*, mais la plus courante étant celle renvoyant la liste des résultats de toutes les *queries* d'une chaîne via la méthode `CtQuery#list()`.

## Exemple

```
1 // collecting all class names
2 list = myPackage
3     .map((CtClass c) -> c.getSimpleName())
4     .list();
5
6 // collecting all deprecated classes
7 list2 = rootPackage
8     .filterChildren(new AnnotationFilter(Deprecated.class))
9     .list();
10
11 // creating a custom filter to select all public fields using Java 8 lambdas
12 list3 = rootPackage
13     .filterChildren((CtField field) -> field.getModifiers()
14         .contains(ModifierKind.PUBLIC))
15     .list();
16
17 // a query which processes non-deprecated methods of deprecated classes
18 list4 = rootPackage
19     .filterChildren((CtClass cls) ->
20         cls.getAnnotation(Deprecated.class) != null)
21     .map((CtClass cls) -> cls.getMethods())
22     .map((CtMethod<?> method) ->
23         method.getAnnotation(Deprecated.class) == null)
24     .list();
25
26 // reusing a query
27 CtQuery q = Factory
28     .createQuery()
29     .map((CtClass cls) -> c.getSimpleName());
30 String cls1Name = q.setInput(Class1).list().get(0);
31 String cls2Name = q.setInput(Class2).list().get(0);
32
33 // prints each deprecated element
34 rootPackage
35     .filterChildren(new AnnotationFilter(Deprecated.class))
36     .forEach((CtElement e) -> System.out.println(e));
37
38 // returns the first deprecated element
39 CtElement firstDeprecated =
40     rootPackage
41     .filterChildren(new AnnotationFilter(Deprecated.class))
42     .first();
```

Listing 2.11 – Exemple d'utilisation des queries Spoon

### Remarque.

Spoon dispose d'autres outils pertinents lors de l'interaction avec le modèle d'un code source, mais qui ne seront pas abordés dans ce tutoriel étant au delà de la portée du TP, tels que les *scanners*, les *iterateurs*, les *paths*, les *roles*, les *templates* et les *patterns*.

## 2.2.11 Les processeurs

Pour définir des méthodes d'analyse et de transformation de code source, employant les différents outils d'interaction avec le modèle **Spoon** vu précédemment, **Spoon** propose la notion de **processeur**, une classe encapsulant ces différents traitements.

Tous les processeurs héritent de la classe de base de processeurs `AbstractProcessor<E extends CtElement>`, et permettent de traiter et d'analyser individuellement des types de nœuds spécifiques du modèle **Spoon**.

En interne, le traitement des éléments est effectué par le biais du *pattern Visitor* appliqué aux éléments du modèle **Spoon**, où chaque élément définit une implémentation de la méthode `accept()` en vue d'être visité par un visiteur.

### Processus standard d'utilisation d'un processeur Spoon

1. définir un processeur étendant `AbstractProcessor<E extends CtElement>` traitant un nœud de type spécifique du modèle ;
2. éventuellement définir un prédictat de sélection des nœuds à traiter via la méthode `boolean isToBeProcessed (E candidate)` ;
3. définir le traitement des éléments sélectionnés dans la méthode `void process(E element)` ;
4. éventuellement arrêter le processus de traitement explicitement n'importe où dans le code du processeur défini, en invoquant la méthode `public void interrupt()`.

### Exemples

```
1 public class CtCommentProcessor extends AbstractProcessor<CtComment> {
2
3     @Override
4     public boolean isToBeProcessed(CtComment candidate){
5         // process only javadoc comments
6         return candidate.getCommentType() == CtComment.CommentType.JAVADOC;
7     }
8
9     @Override
10    public void process(CtComment comment){
11        // process the comment
12    }
13}
```

Listing 2.12 – Exemple d'un processeur de commentaires Spoon

```
1 public class CatchProcessor extends AbstractProcessor<CtCatch> {
2     /*attributes*/
3     // empty catch clauses
4     List<CtCatch> emptyCatches = new ArrayList<>();
5
6     @Override
7     public boolean isToBeProcessed(CtCatch candidate){
8         // process only empty catch clauses
9         return candidate.getBody().getStatements().isEmpty();
10    }
11
12    @Override
13    public void process(CtCatch element){
14        getFactory()
15            .getEnvironment()
```

```

16     .report(this, Level.WARN, "empty catch clause"
17     + element.getPosition().toString());
18
19     emptyCatches.add(element);
20 }
21 }
```

Listing 2.13 – Exemple d'un processeur de clauses catch vides

### 2.2.12 Les *launchers*

La classe **Launcher** modélise un *launcher* CLI pour la construction du modèle Spoon d'un code source, ainsi que son traitement, affichage et sa compilation, en utilisant le *builder* natif d'Eclipse JDT.

Le *launcher* permet de spécifier l'ensemble des processeurs à appliquer sur des fichiers de code source en entrée, comme il peut être utilisé pour traiter directement un **String** en entrée, désignant le code source d'une classe, en invoquant la méthode statique **Launcher.parseClass(String code)**.

Par ailleurs, des *launchers* dédiés plus spécifiques peuvent être utilisés :

**IncrementalLauncher** : effectuer des *builds* incrémentaux en utilisant un cache.

**MavenLauncher** : effectuer un *build* à partir d'un fichier de dépendances d'un projet Maven **pom.xml**.

**JarLauncher** : construire un modèle de code source à partir d'un fichier **.jar** en utilisant un décompilateur pour décompiler le *bytecode* du *jar*.

### Exemples

```

1  public class App {
2      public static void main(String[] args) {
3          // Exemple 1 : Les méthodes d'une classe
4          CtClass l = Launcher.parseClass(
5              "class A {
6                  void m() { System.out.println("Hello, World!"); }
7              }
8          );
9          Set methods = l.getAllMethods();
10         for(Object o: methods.toArray())
11             System.out.println(o.toString());
12     }
13 }
```

Listing 2.14 – Exemple de construction d'un modèle Spoon d'une classe en directe

```

1 *Méthodes de la classe App et celles héritées d'Object*/
2 =====
3 public native final Class<?> getClass() {}
4 public final void wait(long arg0, int arg1) throws InterruptedException {}
5 public final void wait(long arg0) throws InterruptedException {}
6 public final void wait() throws InterruptedException {}
7 public native final void notifyAll() {}
8 public boolean equals(Object arg0) {}
9 private static native void registerNatives() {}
10 public native final void notify() {}
11 void m() {
12     System.out.println("Hello, World!");
13 }
14 public native int hashCode() {}
```

```

15 protected void finalize() throws Throwable {}
16 public String toString() {}
17 protected native Object clone() throws CloneNotSupportedException {}

```

Listing 2.15 – Résultat du listing 2.14

```

1 String[] args = {
2     "-i", "src/main/java/spoon/test",
3     "-o", "target/spooned"
4 };
5
6 Launcher launcher = new Launcher();
7 CommentProcessor commentP = new CommentProcessor();
8 CatchProcessor catchP = new CatchProcessor();
9 launcher.addProcessor(commentP);
10 launcher.addProcessor(catchP);
11
12 launcher.setArgs(args);
13 launcher.run();

```

Listing 2.16 – Exemple d'utilisation d'un Launcher pour appliquer plusieurs processeurs à un projet

### 2.2.13 Instrumentation de code et traces d'appels

Dans le cadre de ce TP, nous avons construit un projet Maven dépendant de `spoon-core 8.0.0.jar` pour effectuer des tâches d'analyse dynamique sur des codes source. Au sein du projet, nous avons défini un processeur Spoon `ToStringGenerator` pour traiter les classes d'un projet quelconque de la manière suivante :

1. filtrer les attributs ayant des *getters* et leurs *getters*;
2. instrumenter les méthodes faisant appel à ces *getters*;
3. si la classe ne déclare pas une implémentation de la méthode `toString()`, celle-ci sera générée automatiquement en utilisant les *getters* et les attributs filtrés.

L'instrumentation des méthodes est *offline* et est effectuée par un *logger* personnalisé composant une instance statique du *logger* de l'**API Java** pour le *logging* (*i.e.* `java.util.logging.Logger`). En outre, les traces d'appel sont formatées :

**java.util.logging.SimpleFormatter** : un formateur de texte simple, prédéfini par l'**API** du *logging* pour les messages à *logger*;  
**HTMLFormatter** : un formateur personnalisé permettant de formater du texte dans des pages **HTML**.

Afin de tester notre processeur, nous avons défini un projet de test contenant les classes suivantes (*cf.* Figure 2.13) :

**Empty** : une classe vide ne définissant pas d'attributs propres, mais définissant une méthode `display()` qui affiche un message indiquant que la classe est vide ;

**Closed** : une classe ayant des attributs propres mais ne définissant aucun *getter* dessus. La classe définit une méthode `callClosedAttrK()` d'accès à l'attribut `closedAttrK` ( $K \in \mathbb{N}^*$ ) sans passer par un *getter*, et une méthode `callAllClosedAttr()` invoquant les méthodes précédentes pour tous les attributs ;

**AlreadyHasToString** : une classe ayant des attributs propres, avec leurs *getters* correspondant, mais ayant déjà sa propre implémentation de la méthode `toString()`. La classe

<b>Empty</b>	<b>AlreadyHasToString</b>
+ display()	- someAttr1: String - someAttr2: String
<b>Closed</b>	<b>HasGettersButNoToString</b>
- closedAttr1: String - closedAttr2: String	- someAttr1: String - someAttr2: String
+ callClosedAttr1() + callClosedAttr2() + callAllClosedAttr()	+ getSomeAttr1(): String + setSomeAttr1(String someAttr1) + getSomeAttr2(): String + setSomeAttr2(String someAttr2) + callGetterAttr1() + callGetterAttr2() + callAllGetters() + toString(): String
<b>HasGettersButNoToString</b>	<b>HasSomeGettersButNoToString</b>
- someAttr1: String - someAttr2: String	- someAttr1: String - someAttr2: String - closedAttr: String
+ getSomeAttr1(): String + setSomeAttr1(String someAttr1) + getSomeAttr2(): String + setSomeAttr2(String someAttr2) + callAllGetters()	+ getSomeAttr1(): String + setSomeAttr1(String someAttr1) + getSomeAttr2(): String + setSomeAttr2(String someAttr2) + callClosedAttr() + callSomeGetters()

FIGURE 2.13 – Classes de test de `ToStringGenerator`

définit une méthode `callGetterAttrK()` pour accéder à un attribut `someAttrK` ( $K \in \mathbb{N}^*$ ) via son *getter* correspondant, et une méthode `callAllGetters()` invoquant les *getters* de tous les attributs ;

**HasSomeGettersButNoToString** : une classe ayant des attributs propres n'ayant pas tous de *getters* correspondant, et n'ayant pas une implémentation de `toString()`. La classe définit la méthode `callClosedAttr()` pour accéder à son attribut n'ayant pas un *getter* et la méthode `callSomeGetters()` pour accéder aux attributs ayant leurs propres *getters* via leur *getters*.

**HasGettersButNoToString** : une classe ayant des attributs propres ayant tous de *getters* correspondant, mais n'ayant pas une implémentation de `toString()`. La classe définit la méthode `callAllGetters()` pour accéder à tous les attributs via leurs *getters*.

**Main** : la classe définissant les différents scénarios d'exécution en manipulant des instances des classes dessus.

Selon la définition de `ToStringGenerator()`, seules les classes `HasSomeGettersButNoToString` et `HasGettersButNoToString` auront une implémentation de `toString()` générée à partir de leurs attributs ayant des *getters* (cf. Listing 2.17).

```

1 // generated toString() for class
2     to_string_generator.data.HasGettersButNoToString:
3 /* Automatically generated by Spoon */
4 @java.lang.Override
5 public java.lang.String toString() {
6     java.lang.StringBuffer buf = new java.lang.StringBuffer();
7     buf.append("someAttr1:\n");
8     buf.append("=====\\n");
9     buf.append(getSomeAttr1()+"\\n\\n");
10    buf.append("someAttr2:\\n");
11    buf.append("=====\\n");
12    buf.append(getSomeAttr2()+"\\n\\n");
13}

```

```

14     return buf.toString();
15 }
16 // generated toString() for class
17     to_string_generator.data.HasSomeGettersButNoToString:
18 /* Automatically generated by Spoon */
19 @java.lang.Override
20 public java.lang.String toString() {
21     java.lang.StringBuffer buf = new java.lang.StringBuffer();
22     buf.append("someAttr1:\n");
23     buf.append("=====\\n");
24     buf.append(getSomeAttr1()+"\\n\\n");
25
26     buf.append("someAttr2:\\n");
27     buf.append("=====\\n");
28     buf.append(getSomeAttr2()+"\\n\\n");
29
30     return buf.toString();
}

```

Listing 2.17 – implémentations de `toString()` générées automatiquement pour les classes `HasGettersButNoToString` et `HasSomeGettersButNoToString`

Par ailleurs, toutes les méthodes invoquant des *getters* seront instrumentées par l’insertion d’instructions *sensors* consistant à *logger* l’invocation par `SpoonLogger` dans un fichier texte et un fichier **HTML** dans le dossier `log/` associé à chaque projet instrumenté par le processeur.

Un exemple d’une méthode qui sera instrumentée par le processeur est la méthode `HasGettersButNoToString::callAllGetters` (*cf.* Listing 2.18)

```

1  public void callAllGetters() {
2      java.lang.System.out.println("called all attributes with getters:");
3      my_spoon.logger.SpoonLogger.info(
4          "to_string_generator.data.HasGettersButNoToString::getSomeAttr1()"
4          " invoked from"
4          " to_string_generator.data.HasGettersButNoToString::callAllGetters()");
5      java.lang.System.out.println("calling someAttr1 with getter:" +
5          "getSomeAttr1()");
6      my_spoon.logger.SpoonLogger.info(
7          "to_string_generator.data.HasGettersButNoToString::getSomeAttr2()"
7          " invoked from"
7          " to_string_generator.data.HasGettersButNoToString::callAllGetters()");
8      java.lang.System.out.println("calling someAttr2 with getter:" +
8          "getSomeAttr2()");
9  }

```

Listing 2.18 – instrumentation de la méthode `HasGettersButNoToString::callAllGetters`

Enfin, les traces d’exécution obtenues par le *logging* du projet de test peuvent être visualisées dans la figure 2.14.

```

1 Jan 16, 2020 1:48:06 PM my_spoon.logger.SpoonLogger info
2 INFO: to_string_generator.data.HasGettersButNoToString::getSomeAttr1() invoked from
* to_string_generator.data.HasGettersButNoToString::callAllGetters()
3 Jan 16, 2020 1:48:06 PM my_spoon.logger.SpoonLogger info
4 INFO: to_string_generator.data.HasGettersButNoToString::getSomeAttr2() invoked from
* to_string_generator.data.HasGettersButNoToString::callAllGetters()
5 Jan 16, 2020 1:48:06 PM my_spoon.logger.SpoonLogger info
6 INFO: to_string_generator.data.HasSomeGettersButNoToString::getSomeAttr1() invoked from
* to_string_generator.data.HasSomeGettersButNoToString::callSomeGetters()
7 Jan 16, 2020 1:48:06 PM my_spoon.logger.SpoonLogger info
8 INFO: to_string_generator.data.HasSomeGettersButNoToString::getSomeAttr2() invoked from
* to_string_generator.data.HasSomeGettersButNoToString::callSomeGetters()

```

(a) Log textuel par SpoonLogger

Thu Jan 16 13:48:06 CET 2020 /n /n /n /n

LogLevel	Time	LogMessage
INFO	Jan 16,2020 13:48	to_string_generator.data.HasGettersButNoToString::getSomeAttr1() invoked from to_string_generator.data.HasGettersButNoToString::callAllGetters()
INFO	Jan 16,2020 13:48	to_string_generator.data.HasGettersButNoToString::getSomeAttr2() invoked from to_string_generator.data.HasGettersButNoToString::callAllGetters()
INFO	Jan 16,2020 13:48	to_string_generator.data.HasSomeGettersButNoToString::getSomeAttr1() invoked from to_string_generator.data.HasSomeGettersButNoToString::callSomeGetters()
INFO	Jan 16,2020 13:48	to_string_generator.data.HasSomeGettersButNoToString::getSomeAttr2() invoked from to_string_generator.data.HasSomeGettersButNoToString::callSomeGetters()

(b) Log HTML par SpoonLogger

FIGURE 2.14 – Logs par SpoonLogger pour les classes traitées par ToStringGenerator

### 2.2.14 Le graphe d'appel dynamique

Pour construire le graphe d'appel dynamique d'un projet, nous avons ciblé les classes contenant une méthode `main()` définissant des exécutions particulières du code du projet. En particulier, l'ajout des invocations au graphe doit se faire à la volée par le biais d'instructions *sensors* instrumentant la méthode `main()` d'une classe et toutes les méthodes invoquées dedans (*si leurs déclarations sont disponibles*).

Le graphe d'appel dynamique est modélisé dans la classe `DynamicCallGraph` qui reprend l'interface définie par le graphe d'appel de base `AbstractCallGraph` définissant l'interface commune à tous les graphes d'appel et utilisée dans la première partie du TP2. Toutefois, nous l'avons réadaptée pour que ses membres soient statiques. Cette réadaptation découle du fait que l'insertion des instructions *sensors* est plus facile et moins verbeuse avec `Spoon` depuis un contexte statique que depuis un contexte d'instance. Ainsi il suffit d'insérer les instructions *sensors* dans `main()` et dans les méthodes invoquées dedans et de terminer l'instrumentation par l'insertion d'une instruction d'affichage du graphe d'appel dynamique obtenu dans la méthode `main()` instrumentée. L'instrumentation des méthodes est *offline* et est effectuée par le biais d'un processeur `Spoon DynamicCallGraphProcessor` utilisant des filtres et des méthodes d'accès aux noeuds de l'`AST` obtenu pour un projet.

Afin de tester notre processeur, nous avons défini un projet de test contenant les classes suivantes (*cf. Figure 2.15*) :

**Person** : une classe définissant une interface de base d'une personne. Une personne peut posséder zéro/plusieurs maison(s) si elle a 21 ans ou plus.

**Address** : une classe définissant une interface de base d'une adresse.

**House** : une classe définissant une interface de base d'une maison. Une maison a exactement une adresse.

**Main** : la classe définissant les différents scénarios d'exécution en manipulant des instances des classes dessus.

Pour créer plusieurs scénarios d'exécution, nous utilisons `java.util.Random` pour générer des nombres aléatoires désignant les âges des personnes, ce qui entraînera des modifications sur

leur capacité de posséder une maison et par conséquent sur le chemin d'exécution choisi dans la méthode `main()` du projet. Les méthodes des classes sont implémentées de manière à pouvoir tester plusieurs localisations différentes des méthodes invoquées, dont l'insertion des instructions *sensors* dépendera.

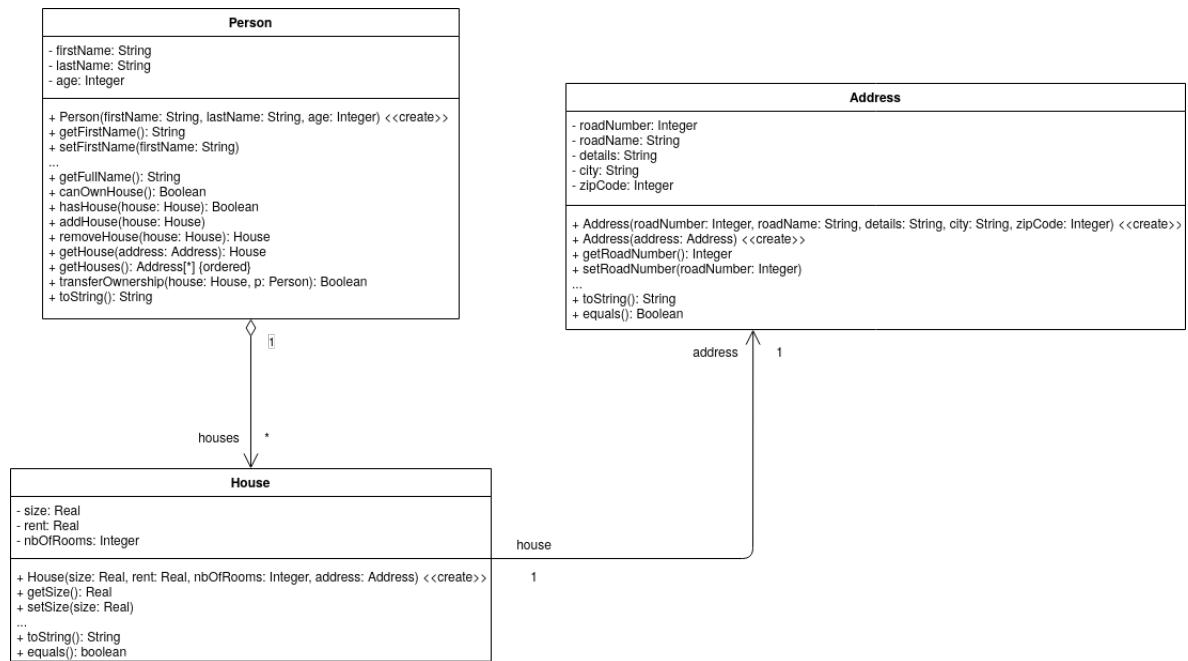


FIGURE 2.15 – Classes de test de DynamicCallGraphProcessor

Ainsi, toutes les méthodes invoquées dans `main()` et les méthodes invoquées dedans seront instrumentées par l'insertion d'instructions *sensors* consistant à ajouter leurs invocations dans la liste des invocations de `DynamicCallGraph`.

Un exemple d'une méthode qui sera instrumentée par le processeur est la méthode `Person::addHouse` (*cf.* Listing 2.19)

```

1  public void addHouse(dynamic_callgraph.data.House house) {
2      my_spoon.callgraph.DynamicCallGraph.addInvocation(
3          "dynamic_callgraph.data.Person::addHouse",
4              "dynamic_callgraph.data.Person::canOwnHouse");
5      my_spoon.callgraph.DynamicCallGraph.addInvocation(
6          "dynamic_callgraph.data.Person::addHouse",
7              "dynamic_callgraph.data.Person::hasHouse");
8      if (canOwnHouse() && (!hasHouse(house))) {
9          my_spoon.callgraph.DynamicCallGraph.addInvocation(
10             "dynamic_callgraph.data.Person::addHouse",
11                 "java.util.List::add");
12         this.houses.add(house);
13     }
14 }
```

Listing 2.19 – instrumentation de la méthode Person::addHouse

Enfin, un aperçu du graphe d'appel dynamique obtenu du projet de test peut être visualisé dans la figure 2.16.

#### *Remarque.*

Le code source du TP2 est disponible sur [https://github.com/anonbnr/hmin306\\_TP/tree/master/tp2/src](https://github.com/anonbnr/hmin306_TP/tree/master/tp2/src).

```
1  dynamic_callgraph.data.Main::main:  
2      ---> java.util.Random::nextInt (3 fois)  
3      ---> dynamic_callgraph.data.Person::transferOwnership (3 fois)  
4      ---> dynamic_callgraph.data.Person::addHouse (2 fois)  
5      ---> java.io.PrintStream::println (6 fois)  
6      ---> dynamic_callgraph.data.Person::canOwnHouse (3 fois)  
7  dynamic_callgraph.data.Person::transferOwnership:  
8      ---> dynamic_callgraph.data.Person::hasHouse (3 fois)  
9      ---> dynamic_callgraph.data.Person::getFullName (3 fois)  
10     ---> java.io.PrintStream::println (3 fois)  
11     ---> dynamic_callgraph.data.Person::getAge (1 fois)  
12     ---> dynamic_callgraph.data.Person::canOwnHouse (1 fois)  
13  dynamic_callgraph.data.Person::addHouse:  
14      ---> dynamic_callgraph.data.Person::hasHouse (2 fois)  
15      ---> java.util.List::add (2 fois)  
16      ---> dynamic_callgraph.data.Person::canOwnHouse (2 fois)
```

FIGURE 2.16 – Aperçu des invocations du graphe d’appel dynamique obtenu du projet de test

# Chapitre 3

## TP3 - Compréhension de programmes

Le but de ce TP est d'étudier les dépendances entre des parties d'un programme. Dans le cadre de l'évolution des logiciels, connaître les dépendances au sein d'un projet est une façon d'en extraire de l'information. Ces informations peuvent être présentées sous la forme de modèles mentaux, d'architectures ou de comportements et sont capitales pour comprendre le programme.

### 3.1 Exercice 1 : Graphe de couplage entre classes

Ici, la dépendance va être étudiée entre deux classes données d'un programme. Pour ce faire on se base sur une certaine métrique, la *mesure de couplage*. Cette métrique est définie comme le « *nombre de relations (relation = appel) entre les couples de méthodes appartenant respectivement aux deux classes en question ( $A.m_i$  et  $B.m_j$ ) / nombre de toutes les relations (binaires) entre les couples de méthodes appartenant respectivement à n'importe quelles des classes de l'application analysée.* », soit en quelque sorte la fréquence d'appel entre deux classes données sur tous les appels existant dans le projet.

Par exemple, si un petit programme comptabilise 35 appels de méthodes entre toutes les classes le composant et que les classes A et B s'appellent 7 fois entre elles, le score de couplage de (A,B) sera de  $\frac{7}{35}$ . Un appel entre les classes A et B est défini comme l'appel d'une méthode mB (définie dans la classe B) dans la classe de A ou l'appel d'une méthode mA (définie dans la classe A) dans la classe B.

Pour implémenter ceci dans le TP, nous avons commencé par compter le nombre d'appels faits entre toutes les classes de l'application. On obtient ainsi le dividende de la métrique de couplage. Ensuite, on parcourt les deux classes étudiées (celles dont on veut connaître le couplage) et on compte tous les appels faits entre elles. Un exemple de ceci est disponible en figure 3.1.

`ArgoJFontChooser<- [10/28152] ->JList`

FIGURE 3.1 – Mesure de couple entre les classes `ArgoJFontChooser` et `JList` d'ArgoUML v0.34.

Ces métriques ont ensuite été utilisées dans un graphe de couplage. Ce graphe précise la métrique de couplage pour toutes les classes constituant un projet. De fait, il suffit d'étendre l'algorithme précédemment évoqué à toutes les classes constituant le programme. Un exemple des résultats obtenus est disponible en figure 3.2<sup>1</sup>.

1. Toutefois, les sources d'ArgoUML étant conséquentes, les illustrations montrées ici ne seront pas entières

```

ArgoJFontChooser<-[8/28152]->Container
ArgoJFontChooser<-[2/28152]->JScrollPane
ArgoJFontChooser<-[10/28152]->JList
ArgoJFontChooser<-[2/28152]->GraphicsEnvironment
ArgoJFontChooser<-[1/28152]->DefaultListModel
ArgoJFontChooser<-[2/28152]->ListSelectionModel
ArgoJFontChooser<-[16/28152]->Integer
ArgoJFontChooser<-[4/28152]->AbstractButton
SettingsTabAppearance<-[11/28152]->Container
SettingsTabAppearance<-[11/28152]->JComboBox
SettingsTabAppearance<-[6/28152]->JLabel

```

FIGURE 3.2 – Extrait de l'affichage console renseignant le graphe de couplage entre les classes du projet ArgoUML v0.34.

## 3.2 Exercice 2 : Identification de Modules

Tout au long de cet exercice, on souhaite identifier des modules au sein des sources d'un programme. Avoir ces modules permet dans un contexte de compréhension des programmes, par exemple d'établir un modèle de correspondance conceptuel. Ceci peut être utile dans le cas d'une migration du système.

On cherche donc, grâce à la mesure de similarité définie auparavant (métrique de couplage) à regrouper les éléments du programme (ici les classes) entre eux en *clusters*. À partir de ces *clusters* on cherche à identifier les composants les constituant.

### 3.2.1 Construction du *cluster* hiérarchique

L'algorithme utilisé pour cette partie est disponible en figure 3.3. Il a été implémenté dans notre TP à partir de la mesure de couplage décrite précédemment. On retrouve le résultat de la construction de la hiérarchie de *clusters* ci-dessous. L'algorithme a été expérimenté sur le code réalisé pour le TP et sur le code d'ArgoUML v0.34, cependant le processus obtenu avec ArgoUML v0.34 n'a pas pu être récupéré car il s'agit d'un projet trop conséquent (la sortie console a été effacée à de multiples reprises, l'exécution a pris plus d'une demi-heure contre moins d'une minute pour notre projet).

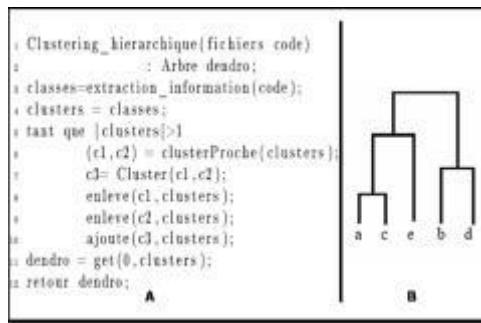


FIGURE 3.3 – Algorithme de regroupement hiérarchique des classes d'une application.

Dans le listing 2.1, on remarque qu'il y a plusieurs *clusters* finaux : au vu du travail demandé, nous avons pris le choix de ne pas réunir deux *clusters* s'ils ne sont pas couplés.

La trace totale de l'exécution de l'algorithme est disponible en annexe 4.2.

Listing 3.1 – Clusters finaux obtenus lors de la construction d'un *cluster* hiérarchique à partir des classes de notre projet.

```

Final clusters :
Cluster [classes=[MethodBodyLineNumberReverseComparator], couplingScore=0]
Cluster [classes=[ClassAttributeNumberReverseComparator], couplingScore=0]
Cluster [classes=[ClassMethodNumberReverseComparator], couplingScore=0]
Cluster [classes=[MethodParamNumberComparator], couplingScore=0]
Cluster [classes=[TestSon], couplingScore=0]
Cluster [classes=[Test], couplingScore=0]
Cluster [classes=[AA, BB, CC], couplingScore=6]
Cluster [classes=[Main, VariabilityParser, ClassInfo, MethodInfo, CallGraph,
StatsParser, Parser, MethodDeclarationVisitor, CouplingParser, Spoon,
Cluster, Couple, MethodInvocationVisitor, TypeDeclarationVisitor,
FieldDeclarationVisitor, PackageDeclarationVisitor], couplingScore=72]
```

### 3.2.2 Partitionnement

L'algorithme utilisé pour cette partie est disponible en figure 3.4. Ce dernier a été implémenté dans notre TP à l'aide du *cluster* hiérarchique précédemment construit. Comme indiqué précédemment, réaliser un tel exemple est assez chronophage. Par souci de simplicité, l'algorithme est appliqué sur le code réalisé dans le cadre de ce TP. On retrouve le résultat de cette exécution dans le *listing* suivant, ainsi que l'exécution totale en annexe 4.2.

```

: Selection_clusters(arbre dendro)
:           : Partition R;
: Pile parcoursClusters;
: empile(racine(dendro),parcoursClusters);
: tant que l'vide(parcoursClusters)
:   Cluster pere=depile(parcoursClusters)
:   Cluster f1=fils1(pere,dendro);
:   Cluster f2=fils2(pere,dendro);
:   si S(pere)>moyenne(S(f1,f2))
:     ajouter(pere,R);
:   sinon
:     empile(f1,parcoursClusters);
:     empile(f2,parcoursClusters);
: retour R;
```

FIGURE 3.4 – Algorithme d'identification des groupes de classes couplées.

Listing 3.2 – Partition obtenue sur les classes de notre projet.

```

Final partition :
Cluster [classes=[Main, VariabilityParser, ClassInfo, MethodInfo, CallGraph,
StatsParser, Parser, MethodDeclarationVisitor, CouplingParser, Spoon, Cluster,
Couple, MethodInvocationVisitor, TypeDeclarationVisitor,
FieldDeclarationVisitor, PackageDeclarationVisitor], couplingScore=72]
Cluster [classes=[Main, VariabilityParser, ClassInfo, MethodInfo, CallGraph,
StatsParser, Parser, MethodDeclarationVisitor, CouplingParser, Spoon, Cluster,
Couple, MethodInvocationVisitor, TypeDeclarationVisitor,
FieldDeclarationVisitor], couplingScore=71]
Cluster [classes=[Main, VariabilityParser, ClassInfo, MethodInfo, CallGraph,
StatsParser, Parser, MethodDeclarationVisitor, CouplingParser, Spoon, Cluster,
Couple, MethodInvocationVisitor, TypeDeclarationVisitor], couplingScore=69]
Cluster [classes=[CallGraph, StatsParser, Parser, MethodDeclarationVisitor,
CouplingParser, Spoon, Cluster, Couple, MethodInvocationVisitor,
TypeDeclarationVisitor], couplingScore=35]
Cluster [classes=[Parser, MethodDeclarationVisitor, CouplingParser, Spoon,
Cluster, Couple, MethodInvocationVisitor, TypeDeclarationVisitor],
couplingScore=29]
```

```

Cluster [classes=[CallGraph, StatsParser], couplingScore=2]
Cluster [classes=[Parser, MethodDeclarationVisitor], couplingScore=2]
Cluster [classes=[CouplingParser, Spoon, Cluster, Couple,
MethodInvocationVisitor, TypeDeclarationVisitor], couplingScore=25]
Cluster [classes=[AA, BB, CC], couplingScore=6]
Cluster [classes=[CouplingParser, Spoon, Cluster, Couple,
MethodInvocationVisitor], couplingScore=22]
Cluster [classes=[AA, BB], couplingScore=3]
Cluster [classes=[Main, VariabilityParser, ClassInfo, MethodInfo],
couplingScore=32]
Cluster [classes=[CouplingParser, Spoon, Cluster, Couple], couplingScore=19]
Cluster [classes=[Spoon, Cluster, Couple], couplingScore=14]
Cluster [classes=[Spoon, Cluster], couplingScore=7]
Cluster [classes=[Main, VariabilityParser, ClassInfo], couplingScore=27]
Cluster [classes=[VariabilityParser, ClassInfo], couplingScore=20]

```

### 3.3 Exercice 3 : Spoon pour Identification de Modules

Les mêmes exercices, à savoir le graphe de couplage, le *cluster* hiérarchique et les groupes de classes couplées ont été réalisés à l'aide de Spoon. Les mêmes algorithmes que ceux précédemment présentés ont été utilisés mais ce qui change ici est que les informations d'appel dans le code ont été trouvées grâce à des **Queries**.

# Chapitre 4

## TP4 - Réingénierie des logiciels

### 4.1 Exercice 1 - Extraction de la variabilité

Le but de cet exercice est de pouvoir comparer deux versions différentes du développement du même logiciel. Idéalement il aurait fallu le tester sur un projet de plus grande ampleur comme ArgoUML. Cependant après avoir essayé notre programme sur ce logiciel, nous avons constaté que l'exécution prenait un temps certain. Par souci de simplicité nous avons donc exécuté nos analyses sur le code réalisé dans le cadre de ce TP ainsi que sur une version altérée de ce code. Le but de ces altérations étant de rendre rendre les deux projets suffisamment différents pour pouvoir appliquer nos scénarios d'analyse (héritage, nouvelles méthodes...).

Les scénarios ainsi étudiés sont :

- Identifier les classes communes et variables entre deux versions d'un logiciel en se basant sur le nom de la classe<sup>1</sup> ;
- Identifier les classes communes et variables entre deux versions d'un logiciel en se basant sur la liste des méthodes de la classe ;
- Identifier les classes communes et variables entre deux versions d'un logiciel en se basant sur la déclaration d'héritage de la classe ;
- Identifier les classes communes et variables entre deux versions d'un logiciel en se basant sur la déclaration des exceptions de la classe ;
- Identifier les classes communes et variables entre deux versions d'un logiciel en se basant sur les différentes implémentations des méthodes d'une classe donnée ;

Le programme réalisé implémente chacune de ces fonctionnalités et permet à l'utilisateur de choisir quel type de comparaison il souhaite effectuer comme illustré en figure 4.1. Enfin, un exemple d'exécution de comparaison des classes par leur déclaration d'exception est disponible dans le *listing* suivant.

```
Comparisons will be on this project.Which information do you want?  
1. Common and variant classes (same classes name).  
2. Common and variant classes (different methods).  
3. Common and variant classes (different heritancy).  
4. Common and variant classes (different exceptions).  
5. Different methods for a given class.  
0 To quit.
```

FIGURE 4.1 – Choix entre les différentes méthodes de comparaison.

---

1. Les versions suivantes se basent également sur le nom de la classe

Listing 4.1 – Résultat de la comparaison en suivant la méthode de comparaison des classes par leur déclaration d'exception (en plus de leur nom).

Following are common classes .  
Parser  
Main  
ClassInfo  
MethodBodyLineNumberReverseComparator  
ClassAttributeNumberReverseComparator  
ClassMethodNumberReverseComparator  
MethodParamNumberComparator  
MethodInvocationVisitor  
FieldDeclarationVisitor  
TypeDeclarationVisitor  
PackageDeclarationVisitor  
MethodDeclarationVisitor  
StatsParser  
TestSon  
AA  
BB  
CC  
Test  
CallGraph  
Spoon  
Couple  
Cluster  
CouplingParser

Following are different classes .

VariabilityParser  
MethodInfo  
MethodInfo

## 4.2 Exercice 2 - Transformation du code source pour supprimer les dépendances OO

Cette question d'ouverture n'a pas été abordée dans le cadre de notre TP.

# Annexes du TP3

## Exercice 2 - *Clustering* hiérarchique

Trace complète de l'exécution de l'algorithme de *clustering* hiérarchique sur le code réalisé pour ces TPs.

```
Original classes are put in clusters :  
Cluster [classes=[Parser], couplingScore=0]  
Cluster [classes=[VariabilityParser], couplingScore=0]  
Cluster [classes=[Main], couplingScore=0]  
Cluster [classes=[MethodInfo], couplingScore=0]  
Cluster [classes=[ClassInfo], couplingScore=0]  
Cluster [classes=[MethodBodyLineNumberReverseComparator], couplingScore=0]  
Cluster [classes=[ClassAttributeNumberReverseComparator], couplingScore=0]  
Cluster [classes=[ClassMethodNumberReverseComparator], couplingScore=0]  
Cluster [classes=[MethodParamNumberComparator], couplingScore=0]  
Cluster [classes=[MethodInvocationVisitor], couplingScore=0]  
Cluster [classes=[FieldDeclarationVisitor], couplingScore=0]  
Cluster [classes=[TypeDeclarationVisitor], couplingScore=0]  
Cluster [classes=[PackageDeclarationVisitor], couplingScore=0]  
Cluster [classes=[MethodDeclarationVisitor], couplingScore=0]  
Cluster [classes=[StatsParser], couplingScore=0]  
Cluster [classes=[TestSon], couplingScore=0]  
Cluster [classes=[AA], couplingScore=0]  
Cluster [classes=[BB], couplingScore=0]  
Cluster [classes=[CC], couplingScore=0]  
Cluster [classes=[Test], couplingScore=0]  
Cluster [classes=[CallGraph], couplingScore=0]  
Cluster [classes=[Spoon], couplingScore=0]  
Cluster [classes=[Couple], couplingScore=0]  
Cluster [classes=[Cluster], couplingScore=0]  
Cluster [classes=[CouplingParser], couplingScore=0]
```

Creation of hierarchical clusters :

```
Fusion of : Cluster [classes=[VariabilityParser], couplingScore=0] and Cluster [classes=  
Clusters :  
Cluster [classes=[Parser], couplingScore=0]  
Cluster [classes=[Main], couplingScore=0]  
Cluster [classes=[MethodInfo], couplingScore=0]  
Cluster [classes=[MethodBodyLineNumberReverseComparator], couplingScore=0]  
Cluster [classes=[ClassAttributeNumberReverseComparator], couplingScore=0]  
Cluster [classes=[ClassMethodNumberReverseComparator], couplingScore=0]  
Cluster [classes=[MethodParamNumberComparator], couplingScore=0]  
Cluster [classes=[MethodInvocationVisitor], couplingScore=0]  
Cluster [classes=[FieldDeclarationVisitor], couplingScore=0]  
Cluster [classes=[TypeDeclarationVisitor], couplingScore=0]
```

```

Cluster [classes=[PackageDeclarationVisitor], couplingScore=0]
Cluster [classes=[MethodDeclarationVisitor], couplingScore=0]
Cluster [classes=[StatsParser], couplingScore=0]
Cluster [classes=[TestSon], couplingScore=0]
Cluster [classes=[AA], couplingScore=0]
Cluster [classes=[BB], couplingScore=0]
Cluster [classes=[CC], couplingScore=0]
Cluster [classes=[Test], couplingScore=0]
Cluster [classes=[CallGraph], couplingScore=0]
Cluster [classes=[Spoon], couplingScore=0]
Cluster [classes=[Couple], couplingScore=0]
Cluster [classes=[Cluster], couplingScore=0]
Cluster [classes=[CouplingParser], couplingScore=0]
Cluster [classes=[VariabilityParser, ClassInfo], couplingScore=20]

```

Fusion of : Cluster [classes=[Main], couplingScore=0] and Cluster [classes=[Variabilit]

Clusters :

```

Cluster [classes=[Parser], couplingScore=0]
Cluster [classes=[MethodInfo], couplingScore=0]
Cluster [classes=[MethodBodyLineNumberReverseComparator], couplingScore=0]
Cluster [classes=[ClassAttributeNumberReverseComparator], couplingScore=0]
Cluster [classes=[ClassMethodNumberReverseComparator], couplingScore=0]
Cluster [classes=[MethodParamNumberComparator], couplingScore=0]
Cluster [classes=[MethodInvocationVisitor], couplingScore=0]
Cluster [classes=[FieldDeclarationVisitor], couplingScore=0]
Cluster [classes=[TypeDeclarationVisitor], couplingScore=0]
Cluster [classes=[PackageDeclarationVisitor], couplingScore=0]
Cluster [classes=[MethodDeclarationVisitor], couplingScore=0]
Cluster [classes=[StatsParser], couplingScore=0]
Cluster [classes=[TestSon], couplingScore=0]
Cluster [classes=[AA], couplingScore=0]
Cluster [classes=[BB], couplingScore=0]
Cluster [classes=[CC], couplingScore=0]
Cluster [classes=[Test], couplingScore=0]
Cluster [classes=[CallGraph], couplingScore=0]
Cluster [classes=[Spoon], couplingScore=0]
Cluster [classes=[Couple], couplingScore=0]
Cluster [classes=[Cluster], couplingScore=0]
Cluster [classes=[CouplingParser], couplingScore=0]
Cluster [classes=[Main, VariabilityParser, ClassInfo], couplingScore=27]

```

Fusion of : Cluster [classes=[Spoon], couplingScore=0] and Cluster [classes=[Cluster],

Clusters :

```

Cluster [classes=[Parser], couplingScore=0]
Cluster [classes=[MethodInfo], couplingScore=0]
Cluster [classes=[MethodBodyLineNumberReverseComparator], couplingScore=0]
Cluster [classes=[ClassAttributeNumberReverseComparator], couplingScore=0]
Cluster [classes=[ClassMethodNumberReverseComparator], couplingScore=0]
Cluster [classes=[MethodParamNumberComparator], couplingScore=0]
Cluster [classes=[MethodInvocationVisitor], couplingScore=0]
Cluster [classes=[FieldDeclarationVisitor], couplingScore=0]
Cluster [classes=[TypeDeclarationVisitor], couplingScore=0]
Cluster [classes=[PackageDeclarationVisitor], couplingScore=0]
Cluster [classes=[MethodDeclarationVisitor], couplingScore=0]
Cluster [classes=[StatsParser], couplingScore=0]
Cluster [classes=[TestSon], couplingScore=0]
Cluster [classes=[AA], couplingScore=0]
Cluster [classes=[BB], couplingScore=0]
Cluster [classes=[CC], couplingScore=0]

```

```

Cluster [classes=[Test], couplingScore=0]
Cluster [classes=[CallGraph], couplingScore=0]
Cluster [classes=[Couple], couplingScore=0]
Cluster [classes=[CouplingParser], couplingScore=0]
Cluster [classes=[Main, VariabilityParser, ClassInfo], couplingScore=27]
Cluster [classes=[Spoon, Cluster], couplingScore=7]

Fusion of : Cluster [classes=[Spoon, Cluster], couplingScore=7] and Cluster [classes=]

Clusters :
Cluster [classes=[Parser], couplingScore=0]
Cluster [classes=[MethodInfo], couplingScore=0]
Cluster [classes=[MethodBodyLineNumberReverseComparator], couplingScore=0]
Cluster [classes=[ClassAttributeNumberReverseComparator], couplingScore=0]
Cluster [classes=[ClassMethodNumberReverseComparator], couplingScore=0]
Cluster [classes=[MethodParamNumberComparator], couplingScore=0]
Cluster [classes=[MethodInvocationVisitor], couplingScore=0]
Cluster [classes=[FieldDeclarationVisitor], couplingScore=0]
Cluster [classes=[TypeDeclarationVisitor], couplingScore=0]
Cluster [classes=[PackageDeclarationVisitor], couplingScore=0]
Cluster [classes=[MethodDeclarationVisitor], couplingScore=0]
Cluster [classes=[StatsParser], couplingScore=0]
Cluster [classes=[TestSon], couplingScore=0]
Cluster [classes=[AA], couplingScore=0]
Cluster [classes=[BB], couplingScore=0]
Cluster [classes=[CC], couplingScore=0]
Cluster [classes=[Test], couplingScore=0]
Cluster [classes=[CallGraph], couplingScore=0]
Cluster [classes=[CouplingParser], couplingScore=0]
Cluster [classes=[Main, VariabilityParser, ClassInfo], couplingScore=27]
Cluster [classes=[Spoon, Cluster, Couple], couplingScore=14]

Fusion of : Cluster [classes=[CouplingParser], couplingScore=0] and Cluster [classes=]

Clusters :
Cluster [classes=[Parser], couplingScore=0]
Cluster [classes=[MethodInfo], couplingScore=0]
Cluster [classes=[MethodBodyLineNumberReverseComparator], couplingScore=0]
Cluster [classes=[ClassAttributeNumberReverseComparator], couplingScore=0]
Cluster [classes=[ClassMethodNumberReverseComparator], couplingScore=0]
Cluster [classes=[MethodParamNumberComparator], couplingScore=0]
Cluster [classes=[MethodInvocationVisitor], couplingScore=0]
Cluster [classes=[FieldDeclarationVisitor], couplingScore=0]
Cluster [classes=[TypeDeclarationVisitor], couplingScore=0]
Cluster [classes=[PackageDeclarationVisitor], couplingScore=0]
Cluster [classes=[MethodDeclarationVisitor], couplingScore=0]
Cluster [classes=[StatsParser], couplingScore=0]
Cluster [classes=[TestSon], couplingScore=0]
Cluster [classes=[AA], couplingScore=0]
Cluster [classes=[BB], couplingScore=0]
Cluster [classes=[CC], couplingScore=0]
Cluster [classes=[Test], couplingScore=0]
Cluster [classes=[CallGraph], couplingScore=0]
Cluster [classes=[Main, VariabilityParser, ClassInfo], couplingScore=27]
Cluster [classes=[CouplingParser, Spoon, Cluster, Couple], couplingScore=19]

Fusion of : Cluster [classes=[Main, VariabilityParser, ClassInfo], couplingScore=27] and Cluster [classes=]

Clusters :
Cluster [classes=[Parser], couplingScore=0]
Cluster [classes=[MethodBodyLineNumberReverseComparator], couplingScore=0]
Cluster [classes=[ClassAttributeNumberReverseComparator], couplingScore=0]

```

```

Cluster [classes=[ClassMethodNumberReverseComparator], couplingScore=0]
Cluster [classes=[MethodParamNumberComparator], couplingScore=0]
Cluster [classes=[MethodInvocationVisitor], couplingScore=0]
Cluster [classes=[FieldDeclarationVisitor], couplingScore=0]
Cluster [classes=[TypeDeclarationVisitor], couplingScore=0]
Cluster [classes=[PackageDeclarationVisitor], couplingScore=0]
Cluster [classes=[MethodDeclarationVisitor], couplingScore=0]
Cluster [classes=[StatsParser], couplingScore=0]
Cluster [classes=[TestSon], couplingScore=0]
Cluster [classes=[AA], couplingScore=0]
Cluster [classes=[BB], couplingScore=0]
Cluster [classes=[CC], couplingScore=0]
Cluster [classes=[Test], couplingScore=0]
Cluster [classes=[CallGraph], couplingScore=0]
Cluster [classes=[CouplingParser, Spoon, Cluster, Couple], couplingScore=19]
Cluster [classes=[Main, VariabilityParser, ClassInfo, MethodInfo], couplingScore=32]

```

Fusion of : Cluster [classes=[AA], couplingScore=0] and Cluster [classes=[BB], couplingScore=0]

Clusters :

```

Cluster [classes=[Parser], couplingScore=0]
Cluster [classes=[MethodBodyLineNumberReverseComparator], couplingScore=0]
Cluster [classes=[ClassAttributeNumberReverseComparator], couplingScore=0]
Cluster [classes=[ClassMethodNumberReverseComparator], couplingScore=0]
Cluster [classes=[MethodParamNumberComparator], couplingScore=0]
Cluster [classes=[MethodInvocationVisitor], couplingScore=0]
Cluster [classes=[FieldDeclarationVisitor], couplingScore=0]
Cluster [classes=[TypeDeclarationVisitor], couplingScore=0]
Cluster [classes=[PackageDeclarationVisitor], couplingScore=0]
Cluster [classes=[MethodDeclarationVisitor], couplingScore=0]
Cluster [classes=[StatsParser], couplingScore=0]
Cluster [classes=[TestSon], couplingScore=0]
Cluster [classes=[CC], couplingScore=0]
Cluster [classes=[Test], couplingScore=0]
Cluster [classes=[CallGraph], couplingScore=0]
Cluster [classes=[CouplingParser, Spoon, Cluster, Couple], couplingScore=19]
Cluster [classes=[Main, VariabilityParser, ClassInfo, MethodInfo], couplingScore=32]
Cluster [classes=[AA, BB], couplingScore=3]

```

Fusion of : Cluster [classes=[CouplingParser, Spoon, Cluster, Couple], couplingScore=19]

Clusters :

```

Cluster [classes=[Parser], couplingScore=0]
Cluster [classes=[MethodBodyLineNumberReverseComparator], couplingScore=0]
Cluster [classes=[ClassAttributeNumberReverseComparator], couplingScore=0]
Cluster [classes=[ClassMethodNumberReverseComparator], couplingScore=0]
Cluster [classes=[MethodParamNumberComparator], couplingScore=0]
Cluster [classes=[FieldDeclarationVisitor], couplingScore=0]
Cluster [classes=[TypeDeclarationVisitor], couplingScore=0]
Cluster [classes=[PackageDeclarationVisitor], couplingScore=0]
Cluster [classes=[MethodDeclarationVisitor], couplingScore=0]
Cluster [classes=[StatsParser], couplingScore=0]
Cluster [classes=[TestSon], couplingScore=0]
Cluster [classes=[CC], couplingScore=0]
Cluster [classes=[Test], couplingScore=0]
Cluster [classes=[CallGraph], couplingScore=0]
Cluster [classes=[Main, VariabilityParser, ClassInfo, MethodInfo], couplingScore=32]
Cluster [classes=[AA, BB], couplingScore=3]
Cluster [classes=[CouplingParser, Spoon, Cluster, Couple, MethodInvocationVisitor], couplingScore=32]

```

Fusion of : Cluster [classes=[AA, BB], couplingScore=3] and Cluster [classes=[CC], couplingScore=0]

```

Clusters :
Cluster [classes=[Parser], couplingScore=0]
Cluster [classes=[MethodBodyLineNumberReverseComparator], couplingScore=0]
Cluster [classes=[ClassAttributeNumberReverseComparator], couplingScore=0]
Cluster [classes=[ClassMethodNumberReverseComparator], couplingScore=0]
Cluster [classes=[MethodParamNumberComparator], couplingScore=0]
Cluster [classes=[FieldDeclarationVisitor], couplingScore=0]
Cluster [classes=[TypeDeclarationVisitor], couplingScore=0]
Cluster [classes=[PackageDeclarationVisitor], couplingScore=0]
Cluster [classes=[MethodDeclarationVisitor], couplingScore=0]
Cluster [classes=[StatsParser], couplingScore=0]
Cluster [classes=[TestSon], couplingScore=0]
Cluster [classes=[Test], couplingScore=0]
Cluster [classes=[CallGraph], couplingScore=0]
Cluster [classes=[Main, VariabilityParser, ClassInfo, MethodInfo], couplingScore=32]
Cluster [classes=[CouplingParser, Spoon, Cluster, Couple, MethodInvocationVisitor], couplingScore=6]
Cluster [classes=[AA, BB, CC], couplingScore=6]

```

Fusion of : Cluster [classes=[CouplingParser, Spoon, Cluster, Couple, MethodInvocationVisitor], couplingScore=6]

```

Clusters :
Cluster [classes=[Parser], couplingScore=0]
Cluster [classes=[MethodBodyLineNumberReverseComparator], couplingScore=0]
Cluster [classes=[ClassAttributeNumberReverseComparator], couplingScore=0]
Cluster [classes=[ClassMethodNumberReverseComparator], couplingScore=0]
Cluster [classes=[MethodParamNumberComparator], couplingScore=0]
Cluster [classes=[FieldDeclarationVisitor], couplingScore=0]
Cluster [classes=[PackageDeclarationVisitor], couplingScore=0]
Cluster [classes=[MethodDeclarationVisitor], couplingScore=0]
Cluster [classes=[StatsParser], couplingScore=0]
Cluster [classes=[TestSon], couplingScore=0]
Cluster [classes=[Test], couplingScore=0]
Cluster [classes=[CallGraph], couplingScore=0]
Cluster [classes=[Main, VariabilityParser, ClassInfo, MethodInfo], couplingScore=32]
Cluster [classes=[AA, BB, CC], couplingScore=6]
Cluster [classes=[CouplingParser, Spoon, Cluster, Couple, MethodInvocationVisitor], couplingScore=6]

```

Fusion of : Cluster [classes=[Parser], couplingScore=0] and Cluster [classes=[MethodInfo], couplingScore=32]

```

Clusters :
Cluster [classes=[MethodBodyLineNumberReverseComparator], couplingScore=0]
Cluster [classes=[ClassAttributeNumberReverseComparator], couplingScore=0]
Cluster [classes=[ClassMethodNumberReverseComparator], couplingScore=0]
Cluster [classes=[MethodParamNumberComparator], couplingScore=0]
Cluster [classes=[FieldDeclarationVisitor], couplingScore=0]
Cluster [classes=[PackageDeclarationVisitor], couplingScore=0]
Cluster [classes=[StatsParser], couplingScore=0]
Cluster [classes=[TestSon], couplingScore=0]
Cluster [classes=[Test], couplingScore=0]
Cluster [classes=[CallGraph], couplingScore=0]
Cluster [classes=[Main, VariabilityParser, ClassInfo, MethodInfo], couplingScore=32]
Cluster [classes=[AA, BB, CC], couplingScore=6]
Cluster [classes=[CouplingParser, Spoon, Cluster, Couple, MethodInvocationVisitor], couplingScore=6]
Cluster [classes=[Parser, MethodDeclarationVisitor], couplingScore=2]

```

Fusion of : Cluster [classes=[CallGraph], couplingScore=0] and Cluster [classes=[StatsParser], couplingScore=0]

```

Clusters :
Cluster [classes=[MethodBodyLineNumberReverseComparator], couplingScore=0]
Cluster [classes=[ClassAttributeNumberReverseComparator], couplingScore=0]
Cluster [classes=[ClassMethodNumberReverseComparator], couplingScore=0]
Cluster [classes=[MethodParamNumberComparator], couplingScore=0]

```

```

Cluster [classes=[FieldDeclarationVisitor], couplingScore=0]
Cluster [classes=[PackageDeclarationVisitor], couplingScore=0]
Cluster [classes=[TestSon], couplingScore=0]
Cluster [classes=[Test], couplingScore=0]
Cluster [classes=[Main, VariabilityParser, ClassInfo, MethodInfo], couplingScore=32]
Cluster [classes=[AA, BB, CC], couplingScore=6]
Cluster [classes=[CouplingParser, Spoon, Cluster, Couple, MethodInvocationVisitor, T]
Cluster [classes=[Parser, MethodDeclarationVisitor], couplingScore=2]
Cluster [classes=[CallGraph, StatsParser], couplingScore=2]

Fusion of : Cluster [classes=[Parser, MethodDeclarationVisitor], couplingScore=2] and Clusters :
Cluster [classes=[MethodBodyLineNumberReverseComparator], couplingScore=0]
Cluster [classes=[ClassAttributeNumberReverseComparator], couplingScore=0]
Cluster [classes=[ClassMethodNumberReverseComparator], couplingScore=0]
Cluster [classes=[MethodParamNumberComparator], couplingScore=0]
Cluster [classes=[FieldDeclarationVisitor], couplingScore=0]
Cluster [classes=[PackageDeclarationVisitor], couplingScore=0]
Cluster [classes=[TestSon], couplingScore=0]
Cluster [classes=[Test], couplingScore=0]
Cluster [classes=[Main, VariabilityParser, ClassInfo, MethodInfo], couplingScore=32]
Cluster [classes=[AA, BB, CC], couplingScore=6]
Cluster [classes=[CallGraph, StatsParser], couplingScore=2]
Cluster [classes=[Parser, MethodDeclarationVisitor, CouplingParser, Spoon, Cluster, C]

Fusion of : Cluster [classes=[CallGraph, StatsParser], couplingScore=2] and Cluster [c
Clusters :
Cluster [classes=[MethodBodyLineNumberReverseComparator], couplingScore=0]
Cluster [classes=[ClassAttributeNumberReverseComparator], couplingScore=0]
Cluster [classes=[ClassMethodNumberReverseComparator], couplingScore=0]
Cluster [classes=[MethodParamNumberComparator], couplingScore=0]
Cluster [classes=[FieldDeclarationVisitor], couplingScore=0]
Cluster [classes=[PackageDeclarationVisitor], couplingScore=0]
Cluster [classes=[TestSon], couplingScore=0]
Cluster [classes=[Test], couplingScore=0]
Cluster [classes=[Main, VariabilityParser, ClassInfo, MethodInfo], couplingScore=32]
Cluster [classes=[AA, BB, CC], couplingScore=6]
Cluster [classes=[CallGraph, StatsParser, Parser, MethodDeclarationVisitor, CouplingP

Fusion of : Cluster [classes=[Main, VariabilityParser, ClassInfo, MethodInfo], couplingScore=0]
Clusters :
Cluster [classes=[MethodBodyLineNumberReverseComparator], couplingScore=0]
Cluster [classes=[ClassAttributeNumberReverseComparator], couplingScore=0]
Cluster [classes=[ClassMethodNumberReverseComparator], couplingScore=0]
Cluster [classes=[MethodParamNumberComparator], couplingScore=0]
Cluster [classes=[FieldDeclarationVisitor], couplingScore=0]
Cluster [classes=[PackageDeclarationVisitor], couplingScore=0]
Cluster [classes=[TestSon], couplingScore=0]
Cluster [classes=[Test], couplingScore=0]
Cluster [classes=[AA, BB, CC], couplingScore=6]
Cluster [classes=[Main, VariabilityParser, ClassInfo, MethodInfo, CallGraph, StatsPar

Fusion of : Cluster [classes=[Main, VariabilityParser, ClassInfo, MethodInfo, CallGraph], couplingScore=0]
Clusters :
Cluster [classes=[MethodBodyLineNumberReverseComparator], couplingScore=0]
Cluster [classes=[ClassAttributeNumberReverseComparator], couplingScore=0]
Cluster [classes=[ClassMethodNumberReverseComparator], couplingScore=0]
Cluster [classes=[MethodParamNumberComparator], couplingScore=0]
Cluster [classes=[PackageDeclarationVisitor], couplingScore=0]

```

```

Cluster [classes=[TestSon], couplingScore=0]
Cluster [classes=[Test], couplingScore=0]
Cluster [classes=[AA, BB, CC], couplingScore=6]
Cluster [classes=[Main, VariabilityParser, ClassInfo, MethodInfo, CallGraph, StatsParser], couplingScore=0]

Fusion of : Cluster [classes=[Main, VariabilityParser, ClassInfo, MethodInfo, CallGraph, StatsParser], couplingScore=0]
Clusters :
Cluster [classes=[MethodBodyLineNumberReverseComparator], couplingScore=0]
Cluster [classes=[ClassAttributeNumberReverseComparator], couplingScore=0]
Cluster [classes=[ClassMethodNumberReverseComparator], couplingScore=0]
Cluster [classes=[MethodParamNumberComparator], couplingScore=0]
Cluster [classes=[TestSon], couplingScore=0]
Cluster [classes=[Test], couplingScore=0]
Cluster [classes=[AA, BB, CC], couplingScore=6]
Cluster [classes=[Main, VariabilityParser, ClassInfo, MethodInfo, CallGraph, StatsParser], couplingScore=0]

Process done.

Final clusters :
Cluster [classes=[MethodBodyLineNumberReverseComparator], couplingScore=0]
Cluster [classes=[ClassAttributeNumberReverseComparator], couplingScore=0]
Cluster [classes=[ClassMethodNumberReverseComparator], couplingScore=0]
Cluster [classes=[MethodParamNumberComparator], couplingScore=0]
Cluster [classes=[TestSon], couplingScore=0]
Cluster [classes=[Test], couplingScore=0]
Cluster [classes=[AA, BB, CC], couplingScore=6]
Cluster [classes=[Main, VariabilityParser, ClassInfo, MethodInfo, CallGraph, StatsParser, Parser, MethodDeclarationVisitor, CouplingParser, Spoon, Cluster, Couple, MethodInvocationVisitor, TypeDeclarationVisitor, FieldDeclarationVisitor, PackageDeclarationVisitor], couplingScore=72]

```

## Exercice 3 - Partition

Here is the partitionnement process :

```

Partition construction
Original classes are put in clusters :
Cluster [classes=[Parser], couplingScore=0]
Cluster [classes=[VariabilityParser], couplingScore=0]
Cluster [classes=[Main], couplingScore=0]
Cluster [classes=[MethodInfo], couplingScore=0]
Cluster [classes=[ClassInfo], couplingScore=0]
Cluster [classes=[MethodBodyLineNumberReverseComparator], couplingScore=0]
Cluster [classes=[ClassAttributeNumberReverseComparator], couplingScore=0]
Cluster [classes=[ClassMethodNumberReverseComparator], couplingScore=0]
Cluster [classes=[MethodParamNumberComparator], couplingScore=0]
Cluster [classes=[MethodInvocationVisitor], couplingScore=0]
Cluster [classes=[FieldDeclarationVisitor], couplingScore=0]
Cluster [classes=[TypeDeclarationVisitor], couplingScore=0]
Cluster [classes=[PackageDeclarationVisitor], couplingScore=0]
Cluster [classes=[MethodDeclarationVisitor], couplingScore=0]
Cluster [classes=[StatsParser], couplingScore=0]
Cluster [classes=[TestSon], couplingScore=0]
Cluster [classes=[AA], couplingScore=0]
Cluster [classes=[BB], couplingScore=0]
Cluster [classes=[CC], couplingScore=0]
Cluster [classes=[Test], couplingScore=0]
Cluster [classes=[CallGraph], couplingScore=0]
Cluster [classes=[Spoon], couplingScore=0]

```

```

Cluster [classes=[Couple], couplingScore=0]
Cluster [classes=[Cluster], couplingScore=0]
Cluster [classes=[CouplingParser], couplingScore=0]

```

Creation of hierarchical clusters :

Fusion of : Cluster [classes=[VariabilityParser], couplingScore=0] and Cluster [classes=[Main], couplingScore=0]

Clusters :

```

Cluster [classes=[Parser], couplingScore=0]
Cluster [classes=[Main], couplingScore=0]
Cluster [classes=[MethodInfo], couplingScore=0]
Cluster [classes=[MethodBodyLineNumberReverseComparator], couplingScore=0]
Cluster [classes=[ClassAttributeNumberReverseComparator], couplingScore=0]
Cluster [classes=[ClassMethodNumberReverseComparator], couplingScore=0]
Cluster [classes=[MethodParamNumberComparator], couplingScore=0]
Cluster [classes=[MethodInvocationVisitor], couplingScore=0]
Cluster [classes=[FieldDeclarationVisitor], couplingScore=0]
Cluster [classes=[TypeDeclarationVisitor], couplingScore=0]
Cluster [classes=[PackageDeclarationVisitor], couplingScore=0]
Cluster [classes=[MethodDeclarationVisitor], couplingScore=0]
Cluster [classes=[StatsParser], couplingScore=0]
Cluster [classes=[TestSon], couplingScore=0]
Cluster [classes=[AA], couplingScore=0]
Cluster [classes=[BB], couplingScore=0]
Cluster [classes=[CC], couplingScore=0]
Cluster [classes=[Test], couplingScore=0]
Cluster [classes=[CallGraph], couplingScore=0]
Cluster [classes=[Spoon], couplingScore=0]
Cluster [classes=[Couple], couplingScore=0]
Cluster [classes=[Cluster], couplingScore=0]
Cluster [classes=[CouplingParser], couplingScore=0]
Cluster [classes=[VariabilityParser, ClassInfo], couplingScore=20]

```

Fusion of : Cluster [classes=[Main], couplingScore=0] and Cluster [classes=[VariabilityParser], couplingScore=0]

Clusters :

```

Cluster [classes=[Parser], couplingScore=0]
Cluster [classes=[MethodInfo], couplingScore=0]
Cluster [classes=[MethodBodyLineNumberReverseComparator], couplingScore=0]
Cluster [classes=[ClassAttributeNumberReverseComparator], couplingScore=0]
Cluster [classes=[ClassMethodNumberReverseComparator], couplingScore=0]
Cluster [classes=[MethodParamNumberComparator], couplingScore=0]
Cluster [classes=[MethodInvocationVisitor], couplingScore=0]
Cluster [classes=[FieldDeclarationVisitor], couplingScore=0]
Cluster [classes=[TypeDeclarationVisitor], couplingScore=0]
Cluster [classes=[PackageDeclarationVisitor], couplingScore=0]
Cluster [classes=[MethodDeclarationVisitor], couplingScore=0]
Cluster [classes=[StatsParser], couplingScore=0]
Cluster [classes=[TestSon], couplingScore=0]
Cluster [classes=[AA], couplingScore=0]
Cluster [classes=[BB], couplingScore=0]
Cluster [classes=[CC], couplingScore=0]
Cluster [classes=[Test], couplingScore=0]
Cluster [classes=[CallGraph], couplingScore=0]
Cluster [classes=[Spoon], couplingScore=0]
Cluster [classes=[Couple], couplingScore=0]
Cluster [classes=[Cluster], couplingScore=0]
Cluster [classes=[CouplingParser], couplingScore=0]
Cluster [classes=[Main, VariabilityParser, ClassInfo], couplingScore=27]

```

Fusion of : Cluster [classes=[Spoon] , couplingScore=0] and Cluster [classes=[Cluster] , couplingScore=0]  
Clusters :

```
Cluster [classes=[Parser] , couplingScore=0]
Cluster [classes=[MethodInfo] , couplingScore=0]
Cluster [classes=[MethodBodyLineNumberReverseComparator] , couplingScore=0]
Cluster [classes=[ClassAttributeNumberReverseComparator] , couplingScore=0]
Cluster [classes=[ClassMethodNumberReverseComparator] , couplingScore=0]
Cluster [classes=[MethodParamNumberComparator] , couplingScore=0]
Cluster [classes=[MethodInvocationVisitor] , couplingScore=0]
Cluster [classes=[FieldDeclarationVisitor] , couplingScore=0]
Cluster [classes=[TypeDeclarationVisitor] , couplingScore=0]
Cluster [classes=[PackageDeclarationVisitor] , couplingScore=0]
Cluster [classes=[MethodDeclarationVisitor] , couplingScore=0]
Cluster [classes=[StatsParser] , couplingScore=0]
Cluster [classes=[TestSon] , couplingScore=0]
Cluster [classes=[AA] , couplingScore=0]
Cluster [classes=[BB] , couplingScore=0]
Cluster [classes=[CC] , couplingScore=0]
Cluster [classes=[Test] , couplingScore=0]
Cluster [classes=[CallGraph] , couplingScore=0]
Cluster [classes=[Couple] , couplingScore=0]
Cluster [classes=[CouplingParser] , couplingScore=0]
Cluster [classes=[Main, VariabilityParser , ClassInfo] , couplingScore=27]
Cluster [classes=[Spoon , Cluster] , couplingScore=7]
```

Fusion of : Cluster [classes=[Spoon, Cluster] , couplingScore=7] and Cluster [classes=[Cluster] , couplingScore=0]  
Clusters :

```
Cluster [classes=[Parser] , couplingScore=0]
Cluster [classes=[MethodInfo] , couplingScore=0]
Cluster [classes=[MethodBodyLineNumberReverseComparator] , couplingScore=0]
Cluster [classes=[ClassAttributeNumberReverseComparator] , couplingScore=0]
Cluster [classes=[ClassMethodNumberReverseComparator] , couplingScore=0]
Cluster [classes=[MethodParamNumberComparator] , couplingScore=0]
Cluster [classes=[MethodInvocationVisitor] , couplingScore=0]
Cluster [classes=[FieldDeclarationVisitor] , couplingScore=0]
Cluster [classes=[TypeDeclarationVisitor] , couplingScore=0]
Cluster [classes=[PackageDeclarationVisitor] , couplingScore=0]
Cluster [classes=[MethodDeclarationVisitor] , couplingScore=0]
Cluster [classes=[StatsParser] , couplingScore=0]
Cluster [classes=[TestSon] , couplingScore=0]
Cluster [classes=[AA] , couplingScore=0]
Cluster [classes=[BB] , couplingScore=0]
Cluster [classes=[CC] , couplingScore=0]
Cluster [classes=[Test] , couplingScore=0]
Cluster [classes=[CallGraph] , couplingScore=0]
Cluster [classes=[CouplingParser] , couplingScore=0]
Cluster [classes=[Main, VariabilityParser , ClassInfo] , couplingScore=27]
Cluster [classes=[Spoon , Cluster , Couple] , couplingScore=14]
```

Fusion of : Cluster [classes=[CouplingParser] , couplingScore=0] and Cluster [classes=[Cluster] , couplingScore=0]  
Clusters :

```
Cluster [classes=[Parser] , couplingScore=0]
Cluster [classes=[MethodInfo] , couplingScore=0]
Cluster [classes=[MethodBodyLineNumberReverseComparator] , couplingScore=0]
Cluster [classes=[ClassAttributeNumberReverseComparator] , couplingScore=0]
Cluster [classes=[ClassMethodNumberReverseComparator] , couplingScore=0]
Cluster [classes=[MethodParamNumberComparator] , couplingScore=0]
Cluster [classes=[MethodInvocationVisitor] , couplingScore=0]
Cluster [classes=[FieldDeclarationVisitor] , couplingScore=0]
```

```

Cluster [classes=[TypeDeclarationVisitor], couplingScore=0]
Cluster [classes=[PackageDeclarationVisitor], couplingScore=0]
Cluster [classes=[MethodDeclarationVisitor], couplingScore=0]
Cluster [classes=[StatsParser], couplingScore=0]
Cluster [classes=[TestSon], couplingScore=0]
Cluster [classes=[AA], couplingScore=0]
Cluster [classes=[BB], couplingScore=0]
Cluster [classes=[CC], couplingScore=0]
Cluster [classes=[Test], couplingScore=0]
Cluster [classes=[CallGraph], couplingScore=0]
Cluster [classes=[Main, VariabilityParser, ClassInfo], couplingScore=27]
Cluster [classes=[CouplingParser, Spoon, Cluster, Couple], couplingScore=19]

```

Fusion of : Cluster [classes=[Main, VariabilityParser, ClassInfo], couplingScore=27] and Cluster [classes=[AA], couplingScore=0]

```

Clusters :
Cluster [classes=[Parser], couplingScore=0]
Cluster [classes=[MethodBodyLineNumberReverseComparator], couplingScore=0]
Cluster [classes=[ClassAttributeNumberReverseComparator], couplingScore=0]
Cluster [classes=[ClassMethodNumberReverseComparator], couplingScore=0]
Cluster [classes=[MethodParamNumberComparator], couplingScore=0]
Cluster [classes=[MethodInvocationVisitor], couplingScore=0]
Cluster [classes=[FieldDeclarationVisitor], couplingScore=0]
Cluster [classes=[TypeDeclarationVisitor], couplingScore=0]
Cluster [classes=[PackageDeclarationVisitor], couplingScore=0]
Cluster [classes=[MethodDeclarationVisitor], couplingScore=0]
Cluster [classes=[StatsParser], couplingScore=0]
Cluster [classes=[TestSon], couplingScore=0]
Cluster [classes=[AA], couplingScore=0]
Cluster [classes=[BB], couplingScore=0]
Cluster [classes=[CC], couplingScore=0]
Cluster [classes=[Test], couplingScore=0]
Cluster [classes=[CallGraph], couplingScore=0]
Cluster [classes=[CouplingParser, Spoon, Cluster, Couple], couplingScore=19]
Cluster [classes=[Main, VariabilityParser, ClassInfo, MethodInfo], couplingScore=32]

```

Fusion of : Cluster [classes=[AA], couplingScore=0] and Cluster [classes=[BB], couplingScore=0]

```

Clusters :
Cluster [classes=[Parser], couplingScore=0]
Cluster [classes=[MethodBodyLineNumberReverseComparator], couplingScore=0]
Cluster [classes=[ClassAttributeNumberReverseComparator], couplingScore=0]
Cluster [classes=[ClassMethodNumberReverseComparator], couplingScore=0]
Cluster [classes=[MethodParamNumberComparator], couplingScore=0]
Cluster [classes=[MethodInvocationVisitor], couplingScore=0]
Cluster [classes=[FieldDeclarationVisitor], couplingScore=0]
Cluster [classes=[TypeDeclarationVisitor], couplingScore=0]
Cluster [classes=[PackageDeclarationVisitor], couplingScore=0]
Cluster [classes=[MethodDeclarationVisitor], couplingScore=0]
Cluster [classes=[StatsParser], couplingScore=0]
Cluster [classes=[TestSon], couplingScore=0]
Cluster [classes=[CC], couplingScore=0]
Cluster [classes=[Test], couplingScore=0]
Cluster [classes=[CallGraph], couplingScore=0]
Cluster [classes=[CouplingParser, Spoon, Cluster, Couple], couplingScore=19]
Cluster [classes=[Main, VariabilityParser, ClassInfo, MethodInfo], couplingScore=32]
Cluster [classes=[AA, BB], couplingScore=3]

```

Fusion of : Cluster [classes=[CouplingParser, Spoon, Cluster, Couple], couplingScore=19] and Cluster [classes=[AA, BB], couplingScore=3]

```

Clusters :
Cluster [classes=[Parser], couplingScore=0]

```

```

Cluster [classes=[MethodBodyLineNumberReverseComparator], couplingScore=0]
Cluster [classes=[ClassAttributeNumberReverseComparator], couplingScore=0]
Cluster [classes=[ClassMethodNumberReverseComparator], couplingScore=0]
Cluster [classes=[MethodParamNumberComparator], couplingScore=0]
Cluster [classes=[FieldDeclarationVisitor], couplingScore=0]
Cluster [classes=[TypeDeclarationVisitor], couplingScore=0]
Cluster [classes=[PackageDeclarationVisitor], couplingScore=0]
Cluster [classes=[MethodDeclarationVisitor], couplingScore=0]
Cluster [classes=[StatsParser], couplingScore=0]
Cluster [classes=[TestSon], couplingScore=0]
Cluster [classes=[CC], couplingScore=0]
Cluster [classes=[Test], couplingScore=0]
Cluster [classes=[CallGraph], couplingScore=0]
Cluster [classes=[Main, VariabilityParser, ClassInfo, MethodInfo], couplingScore=32]
Cluster [classes=[AA, BB], couplingScore=3]
Cluster [classes=[CouplingParser, Spoon, Cluster, Couple, MethodInvocationVisitor], c

```

Fusion of : Cluster [classes=[AA, BB], couplingScore=3] and Cluster [classes=[CC], couplingScore=0]

Clusters :

```

Cluster [classes=[Parser], couplingScore=0]
Cluster [classes=[MethodBodyLineNumberReverseComparator], couplingScore=0]
Cluster [classes=[ClassAttributeNumberReverseComparator], couplingScore=0]
Cluster [classes=[ClassMethodNumberReverseComparator], couplingScore=0]
Cluster [classes=[MethodParamNumberComparator], couplingScore=0]
Cluster [classes=[FieldDeclarationVisitor], couplingScore=0]
Cluster [classes=[TypeDeclarationVisitor], couplingScore=0]
Cluster [classes=[PackageDeclarationVisitor], couplingScore=0]
Cluster [classes=[MethodDeclarationVisitor], couplingScore=0]
Cluster [classes=[StatsParser], couplingScore=0]
Cluster [classes=[TestSon], couplingScore=0]
Cluster [classes=[Test], couplingScore=0]
Cluster [classes=[CallGraph], couplingScore=0]
Cluster [classes=[Main, VariabilityParser, ClassInfo, MethodInfo], couplingScore=32]
Cluster [classes=[CouplingParser, Spoon, Cluster, Couple, MethodInvocationVisitor], c
Cluster [classes=[AA, BB, CC], couplingScore=6]

```

Fusion of : Cluster [classes=[CouplingParser, Spoon, Cluster, Couple, MethodInvocationVisitor], couplingScore=6] and Cluster [classes=[AA, BB, CC], couplingScore=6]

Clusters :

```

Cluster [classes=[Parser], couplingScore=0]
Cluster [classes=[MethodBodyLineNumberReverseComparator], couplingScore=0]
Cluster [classes=[ClassAttributeNumberReverseComparator], couplingScore=0]
Cluster [classes=[ClassMethodNumberReverseComparator], couplingScore=0]
Cluster [classes=[MethodParamNumberComparator], couplingScore=0]
Cluster [classes=[FieldDeclarationVisitor], couplingScore=0]
Cluster [classes=[PackageDeclarationVisitor], couplingScore=0]
Cluster [classes=[MethodDeclarationVisitor], couplingScore=0]
Cluster [classes=[StatsParser], couplingScore=0]
Cluster [classes=[TestSon], couplingScore=0]
Cluster [classes=[Test], couplingScore=0]
Cluster [classes=[CallGraph], couplingScore=0]
Cluster [classes=[Main, VariabilityParser, ClassInfo, MethodInfo], couplingScore=32]
Cluster [classes=[AA, BB, CC], couplingScore=6]
Cluster [classes=[CouplingParser, Spoon, Cluster, Couple, MethodInvocationVisitor], c

```

Fusion of : Cluster [classes=[Parser], couplingScore=0] and Cluster [classes=[MethodInvocationVisitor], couplingScore=0]

Clusters :

```

Cluster [classes=[MethodBodyLineNumberReverseComparator], couplingScore=0]
Cluster [classes=[ClassAttributeNumberReverseComparator], couplingScore=0]
Cluster [classes=[ClassMethodNumberReverseComparator], couplingScore=0]

```

```

Cluster [classes=[MethodParamNumberComparator], couplingScore=0]
Cluster [classes=[FieldDeclarationVisitor], couplingScore=0]
Cluster [classes=[PackageDeclarationVisitor], couplingScore=0]
Cluster [classes=[StatsParser], couplingScore=0]
Cluster [classes=[TestSon], couplingScore=0]
Cluster [classes=[Test], couplingScore=0]
Cluster [classes=[CallGraph], couplingScore=0]
Cluster [classes=[Main, VariabilityParser, ClassInfo, MethodInfo], couplingScore=32]
Cluster [classes=[AA, BB, CC], couplingScore=6]
Cluster [classes=[CouplingParser, Spoon, Cluster, Couple, MethodInvocationVisitor, T]
Cluster [classes=[Parser, MethodDeclarationVisitor], couplingScore=2]

```

Fusion of : Cluster [classes=[CallGraph], couplingScore=0] and Cluster [classes=[StatsParser]]

```

Clusters :
Cluster [classes=[MethodBodyLineNumberReverseComparator], couplingScore=0]
Cluster [classes=[ClassAttributeNumberReverseComparator], couplingScore=0]
Cluster [classes=[ClassMethodNumberReverseComparator], couplingScore=0]
Cluster [classes=[MethodParamNumberComparator], couplingScore=0]
Cluster [classes=[FieldDeclarationVisitor], couplingScore=0]
Cluster [classes=[PackageDeclarationVisitor], couplingScore=0]
Cluster [classes=[TestSon], couplingScore=0]
Cluster [classes=[Test], couplingScore=0]
Cluster [classes=[Main, VariabilityParser, ClassInfo, MethodInfo], couplingScore=32]
Cluster [classes=[AA, BB, CC], couplingScore=6]
Cluster [classes=[CouplingParser, Spoon, Cluster, Couple, MethodInvocationVisitor, T]
Cluster [classes=[Parser, MethodDeclarationVisitor], couplingScore=2]
Cluster [classes=[CallGraph, StatsParser], couplingScore=2]

```

Fusion of : Cluster [classes=[Parser, MethodDeclarationVisitor], couplingScore=2] and Cluster [classes=[CallGraph, StatsParser]]

```

Clusters :
Cluster [classes=[MethodBodyLineNumberReverseComparator], couplingScore=0]
Cluster [classes=[ClassAttributeNumberReverseComparator], couplingScore=0]
Cluster [classes=[ClassMethodNumberReverseComparator], couplingScore=0]
Cluster [classes=[MethodParamNumberComparator], couplingScore=0]
Cluster [classes=[FieldDeclarationVisitor], couplingScore=0]
Cluster [classes=[PackageDeclarationVisitor], couplingScore=0]
Cluster [classes=[TestSon], couplingScore=0]
Cluster [classes=[Test], couplingScore=0]
Cluster [classes=[Main, VariabilityParser, ClassInfo, MethodInfo], couplingScore=32]
Cluster [classes=[AA, BB, CC], couplingScore=6]
Cluster [classes=[CallGraph, StatsParser], couplingScore=2]
Cluster [classes=[Parser, MethodDeclarationVisitor, CouplingParser, Spoon, Cluster, C]

```

Fusion of : Cluster [classes=[CallGraph, StatsParser], couplingScore=2] and Cluster [classes=[Parser, MethodDeclarationVisitor, CouplingParser, Spoon, Cluster, CallGraph, StatsParser]]

```

Clusters :
Cluster [classes=[MethodBodyLineNumberReverseComparator], couplingScore=0]
Cluster [classes=[ClassAttributeNumberReverseComparator], couplingScore=0]
Cluster [classes=[ClassMethodNumberReverseComparator], couplingScore=0]
Cluster [classes=[MethodParamNumberComparator], couplingScore=0]
Cluster [classes=[FieldDeclarationVisitor], couplingScore=0]
Cluster [classes=[PackageDeclarationVisitor], couplingScore=0]
Cluster [classes=[TestSon], couplingScore=0]
Cluster [classes=[Test], couplingScore=0]
Cluster [classes=[Main, VariabilityParser, ClassInfo, MethodInfo], couplingScore=32]
Cluster [classes=[AA, BB, CC], couplingScore=6]
Cluster [classes=[CallGraph, StatsParser, Parser, MethodDeclarationVisitor, CouplingP

```

Fusion of : Cluster [classes=[Main, VariabilityParser, ClassInfo, MethodInfo], couplingScore=32] and Cluster [classes=[CallGraph, StatsParser, Parser, MethodDeclarationVisitor, CouplingParser, Main, VariabilityParser, ClassInfo, MethodInfo]]

```

Cluster [classes=[MethodBodyLineNumberReverseComparator], couplingScore=0]
Cluster [classes=[ClassAttributeNumberReverseComparator], couplingScore=0]
Cluster [classes=[ClassMethodNumberReverseComparator], couplingScore=0]
Cluster [classes=[MethodParamNumberComparator], couplingScore=0]
Cluster [classes=[FieldDeclarationVisitor], couplingScore=0]
Cluster [classes=[PackageDeclarationVisitor], couplingScore=0]
Cluster [classes=[TestSon], couplingScore=0]
Cluster [classes=[Test], couplingScore=0]
Cluster [classes=[AA, BB, CC], couplingScore=6]
Cluster [classes=[Main, VariabilityParser, ClassInfo, MethodInfo, CallGraph, StatsParser], couplingScore=0]

Fusion of : Cluster [classes=[Main, VariabilityParser, ClassInfo, MethodInfo, CallGraph, StatsParser], couplingScore=0]
Clusters :
Cluster [classes=[MethodBodyLineNumberReverseComparator], couplingScore=0]
Cluster [classes=[ClassAttributeNumberReverseComparator], couplingScore=0]
Cluster [classes=[ClassMethodNumberReverseComparator], couplingScore=0]
Cluster [classes=[MethodParamNumberComparator], couplingScore=0]
Cluster [classes=[PackageDeclarationVisitor], couplingScore=0]
Cluster [classes=[TestSon], couplingScore=0]
Cluster [classes=[Test], couplingScore=0]
Cluster [classes=[AA, BB, CC], couplingScore=6]
Cluster [classes=[Main, VariabilityParser, ClassInfo, MethodInfo, CallGraph, StatsParser], couplingScore=0]

Fusion of : Cluster [classes=[Main, VariabilityParser, ClassInfo, MethodInfo, CallGraph, StatsParser], couplingScore=0]
Clusters :
Cluster [classes=[MethodBodyLineNumberReverseComparator], couplingScore=0]
Cluster [classes=[ClassAttributeNumberReverseComparator], couplingScore=0]
Cluster [classes=[ClassMethodNumberReverseComparator], couplingScore=0]
Cluster [classes=[MethodParamNumberComparator], couplingScore=0]
Cluster [classes=[TestSon], couplingScore=0]
Cluster [classes=[Test], couplingScore=0]
Cluster [classes=[AA, BB, CC], couplingScore=6]
Cluster [classes=[Main, VariabilityParser, ClassInfo, MethodInfo, CallGraph, StatsParser], couplingScore=0]
Process done.

Final clusters :
Cluster [classes=[MethodBodyLineNumberReverseComparator], couplingScore=0]
Cluster [classes=[ClassAttributeNumberReverseComparator], couplingScore=0]
Cluster [classes=[ClassMethodNumberReverseComparator], couplingScore=0]
Cluster [classes=[MethodParamNumberComparator], couplingScore=0]
Cluster [classes=[TestSon], couplingScore=0]
Cluster [classes=[Test], couplingScore=0]
Cluster [classes=[AA, BB, CC], couplingScore=6]
Cluster [classes=[Main, VariabilityParser, ClassInfo, MethodInfo, CallGraph, StatsParser], couplingScore=0]

Father weight : 72
Average of sons weight : 35.0
We add father to partition.
Partition(s) :
Cluster [classes=[Main, VariabilityParser, ClassInfo, MethodInfo, CallGraph, StatsParser], couplingScore=0]

Father weight : 71
Average of sons weight : 34.0
We add father to partition.
Partition(s) :
Cluster [classes=[Main, VariabilityParser, ClassInfo, MethodInfo, CallGraph, StatsParser], couplingScore=0]
Cluster [classes=[Main, VariabilityParser, ClassInfo, MethodInfo, CallGraph, StatsParser], couplingScore=0]

Father weight : 69
Average of sons weight : 33.0

```

We add father to partition.

Partition(s) :

```
Cluster [classes=[Main, VariabilityParser, ClassInfo, MethodInfo, CallGraph, StatsParser, StatsCalculator], name=Cluster]
Cluster [classes=[Main, VariabilityParser, ClassInfo, MethodInfo, CallGraph, StatsParser, StatsCalculator], name=Cluster]
Cluster [classes=[Main, VariabilityParser, ClassInfo, MethodInfo, CallGraph, StatsParser, StatsCalculator], name=Cluster]
```

Father weight : 35

Average of sons weight : 15.0

We add father to partition.

Partition(s) :

Father weight : 29

Average of sons weight : 13.0

We add father to partition.

Partition(s) :

```
Cluster [ classes=[Main, VariabilityParser, ClassInfo, MethodInfo, CallGraph, StatsParser]
Cluster [ classes=[Main, VariabilityParser, ClassInfo, MethodInfo, CallGraph, StatsParser]
Cluster [ classes=[Main, VariabilityParser, ClassInfo, MethodInfo, CallGraph, StatsParser]
Cluster [ classes=[CallGraph, StatsParser, Parser, MethodDeclarationVisitor, CouplingParser]
Cluster [ classes=[Parser, MethodDeclarationVisitor, CouplingParser, Spoon, Cluster, Configuration]
```

Father weight : 2

Average of sons weight : 0.0

We add father to partition.

Partition(s) :

```
Cluster [classes=[Main, VariabilityParser, ClassInfo, MethodInfo, CallGraph, StatsParser], couplingScore=1]
Cluster [classes=[Main, VariabilityParser, ClassInfo, MethodInfo, CallGraph, StatsParser], couplingScore=1]
Cluster [classes=[Main, VariabilityParser, ClassInfo, MethodInfo, CallGraph, StatsParser], couplingScore=1]
Cluster [classes=[CallGraph, StatsParser, Parser, MethodDeclarationVisitor, CouplingParser], couplingScore=1]
Cluster [classes=[Parser, MethodDeclarationVisitor, CouplingParser, Spoon, Cluster, Configuration], couplingScore=1]
Cluster [classes=[CallGraph, StatsParser], couplingScore=2]
```

Father weight : 2

Average of sons weight : 0.0

We add father to partition.

Partition(s) :

```
Cluster [classes=[Main, VariabilityParser, ClassInfo, MethodInfo, CallGraph, StatsParser], couplingScore=2]
Cluster [classes=[Main, VariabilityParser, ClassInfo, MethodInfo, CallGraph, StatsParser], couplingScore=2]
Cluster [classes=[Main, VariabilityParser, ClassInfo, MethodInfo, CallGraph, StatsParser], couplingScore=2]
Cluster [classes=[CallGraph, StatsParser, Parser, MethodDeclarationVisitor, CouplingParser], couplingScore=2]
Cluster [classes=[Parser, MethodDeclarationVisitor, CouplingParser, Spoon, Cluster, Configuration], couplingScore=2]
Cluster [classes=[CallGraph, StatsParser], couplingScore=2]
Cluster [classes=[Parser, MethodDeclarationVisitor], couplingScore=2]
```

Father weight : 25

Average of sons weight : 11.0

We add father to partition.

Partition(s) :

```
Cluster [classes=[Main, VariabilityParser, ClassInfo, MethodInfo, CallGraph, StatsParser], couplingScore=2]
Cluster [classes=[Main, VariabilityParser, ClassInfo, MethodInfo, CallGraph, StatsParser], couplingScore=2]
Cluster [classes=[Main, VariabilityParser, ClassInfo, MethodInfo, CallGraph, StatsParser], couplingScore=2]
Cluster [classes=[CallGraph, StatsParser, Parser, MethodDeclarationVisitor, CouplingParser], couplingScore=2]
Cluster [classes=[Parser, MethodDeclarationVisitor, CouplingParser, Spoon, Cluster, ClassInfo], couplingScore=2]
Cluster [classes=[CallGraph, StatsParser], couplingScore=2]
Cluster [classes=[Parser, MethodDeclarationVisitor], couplingScore=2]
```



```
Cluster [classes=[AA, BB, CC], couplingScore=6]
Cluster [classes=[CouplingParser, Spoon, Cluster, Couple, MethodInvocationVisitor], c
Cluster [classes=[AA, BB], couplingScore=3]
Cluster [classes=[Main, VariabilityParser, ClassInfo, MethodInfo], couplingScore=32]
```

Father weight : 19

Average of sons weight : 7.0

We add father to partition.

Partition(s) :

```
Cluster [classes=[Main, VariabilityParser, ClassInfo, MethodInfo, CallGraph, StatsParser], couplingScore=1]
Cluster [classes=[Main, VariabilityParser, ClassInfo, MethodInfo, CallGraph, StatsParser], couplingScore=1]
Cluster [classes=[Main, VariabilityParser, ClassInfo, MethodInfo, CallGraph, StatsParser], couplingScore=1]
Cluster [classes=[CallGraph, StatsParser, Parser, MethodDeclarationVisitor, CouplingParser], couplingScore=1]
Cluster [classes=[Parser, MethodDeclarationVisitor, CouplingParser, Spoon, Cluster, ClassInfo], couplingScore=1]
Cluster [classes=[CallGraph, StatsParser], couplingScore=2]
Cluster [classes=[Parser, MethodDeclarationVisitor], couplingScore=2]
Cluster [classes=[CouplingParser, Spoon, Cluster, Couple, MethodInvocationVisitor, Type], couplingScore=1]
Cluster [classes=[AA, BB, CC], couplingScore=6]
Cluster [classes=[CouplingParser, Spoon, Cluster, Couple, MethodInvocationVisitor], couplingScore=1]
Cluster [classes=[AA, BB], couplingScore=3]
Cluster [classes=[Main, VariabilityParser, ClassInfo, MethodInfo], couplingScore=32]
Cluster [classes=[CouplingParser, Spoon, Cluster, Couple], couplingScore=19]
```

Father weight : 14

Average of sons weight : 3.0

We add father to partition.

Partition(s) :

```
Cluster [classes=[Main, VariabilityParser, ClassInfo, MethodInfo, CallGraph, StatsParser], couplingScore=1]
Cluster [classes=[Main, VariabilityParser, ClassInfo, MethodInfo, CallGraph, StatsParser], couplingScore=1]
Cluster [classes=[Main, VariabilityParser, ClassInfo, MethodInfo, CallGraph, StatsParser], couplingScore=1]
Cluster [classes=[CallGraph, StatsParser, Parser, MethodDeclarationVisitor, CouplingParser], couplingScore=1]
Cluster [classes=[Parser, MethodDeclarationVisitor, CouplingParser, Spoon, Cluster, Couple], couplingScore=1]
Cluster [classes=[CallGraph, StatsParser], couplingScore=2]
Cluster [classes=[Parser, MethodDeclarationVisitor], couplingScore=2]
Cluster [classes=[CouplingParser, Spoon, Cluster, Couple, MethodInvocationVisitor, Type], couplingScore=1]
Cluster [classes=[AA, BB, CC], couplingScore=6]
Cluster [classes=[CouplingParser, Spoon, Cluster, Couple, MethodInvocationVisitor], couplingScore=1]
Cluster [classes=[AA, BB], couplingScore=3]
Cluster [classes=[Main, VariabilityParser, ClassInfo, MethodInfo], couplingScore=32]
Cluster [classes=[CouplingParser, Spoon, Cluster, Couple], couplingScore=19]
Cluster [classes=[Spoon, Cluster, Couple], couplingScore=14]
```

Father weight : 7

Average of sons weight : 0.0

We add father to partition.

Partition(s) :

```
Cluster [classes=[Main, VariabilityParser, ClassInfo, MethodInfo, CallGraph, StatsParser], couplingScore=1]
Cluster [classes=[Main, VariabilityParser, ClassInfo, MethodInfo, CallGraph, StatsParser], couplingScore=1]
Cluster [classes=[Main, VariabilityParser, ClassInfo, MethodInfo, CallGraph, StatsParser], couplingScore=1]
Cluster [classes=[CallGraph, StatsParser, Parser, MethodDeclarationVisitor, CouplingParser], couplingScore=1]
Cluster [classes=[Parser, MethodDeclarationVisitor, CouplingParser, Spoon, Cluster], couplingScore=1]
Cluster [classes=[CallGraph, StatsParser], couplingScore=2]
Cluster [classes=[Parser, MethodDeclarationVisitor], couplingScore=2]
Cluster [classes=[CouplingParser, Spoon, Cluster, Couple, MethodInvocationVisitor, Type], couplingScore=1]
Cluster [classes=[AA, BB, CC], couplingScore=6]
Cluster [classes=[CouplingParser, Spoon, Cluster, Couple, MethodInvocationVisitor], couplingScore=1]
Cluster [classes=[AA, BB], couplingScore=3]
Cluster [classes=[Main, VariabilityParser, ClassInfo, MethodInfo], couplingScore=32]
Cluster [classes=[CouplingParser, Spoon, Cluster, Couple], couplingScore=19]
```



```
CouplingParser , Spoon , Cluster , Couple , MethodInvocationVisitor ,  
TypeDeclarationVisitor] , couplingScore=35]  
Cluster [classes=[Parser , MethodDeclarationVisitor , CouplingParser , Spoon ,  
Cluster , Couple , MethodInvocationVisitor , TypeDeclarationVisitor] ,  
couplingScore=29]  
Cluster [classes=[CallGraph , StatsParser] , couplingScore=2]  
Cluster [classes=[Parser , MethodDeclarationVisitor] , couplingScore=2]  
Cluster [classes=[CouplingParser , Spoon , Cluster , Couple ,  
MethodInvocationVisitor , TypeDeclarationVisitor] , couplingScore=25]  
Cluster [classes=[AA, BB, CC] , couplingScore=6]  
Cluster [classes=[CouplingParser , Spoon , Cluster , Couple ,  
MethodInvocationVisitor] , couplingScore=22]  
Cluster [classes=[AA, BB] , couplingScore=3]  
Cluster [classes=[Main , VariabilityParser , ClassInfo , MethodInfo] ,  
couplingScore=32]  
Cluster [classes=[CouplingParser , Spoon , Cluster , Couple] , couplingScore=19]  
Cluster [classes=[Spoon , Cluster , Couple] , couplingScore=14]  
Cluster [classes=[Spoon , Cluster] , couplingScore=7]  
Cluster [classes=[Main , VariabilityParser , ClassInfo] , couplingScore=27]  
Cluster [classes=[VariabilityParser , ClassInfo] , couplingScore=20]
```