

Université De Montpellier
Faculté Des Sciences



Niveau : Master 2

Module : Évolution et restructuration des logiciels

HAI913I

Rapport de TP N°2 Partie 2 : Analyse statique et dynamique

Supervisé par :
M. Abdelhak-Djamel Seriali
M. Marianne Huchard

Réalisé par :
AHMED Kaci
YANIS Allouch

2021/2022

Table des matières

1	Travail réalisé	3
2	Conclusion	10
3	Annexe	10
3.1	Ressources utilisées	10
	Références bibliographiques	11

Introduction

Dans ce TP, ayant pour thème l'analyse statique de programmes, nous allons réaliser une application permettant d'analyser le code source d'un programme java en se basant sur son [arbre de syntaxe abstraite](#) (AST) que nous pouvons visualiser au travers d'un [AST View](#) au sein de l'IDE Eclipse et que nous manipuleront en utilisant [Eclipse JDT ASTParser](#) dont le fonctionnement se base sur le patron de conception [Visiteur](#).

Lien du dépôt GitLab du [projet](#) est le suivant :
https://gitlab.com/kaciahmed3/tp2_hai913i

1 Travail réalisé

L'application réaliser prends en argument le chemin vers le dossier contenant le programme java à analysé.

Pour éviter certain bug générer par [java swing](#), il est préférable d'ajouter la ligne du [Listing 1](#) suivante en argument de la VM.

```
1 -Djava.awt.headless=false
```

Listing 1 – Argument à passé pour la JVM

L'image en [Figure 1](#) renseigne un exemple de configuration permettant d'exécuter l'application.

Après l'exécution de l'application on obtient le menu sur la [Figure 2](#) suivante :

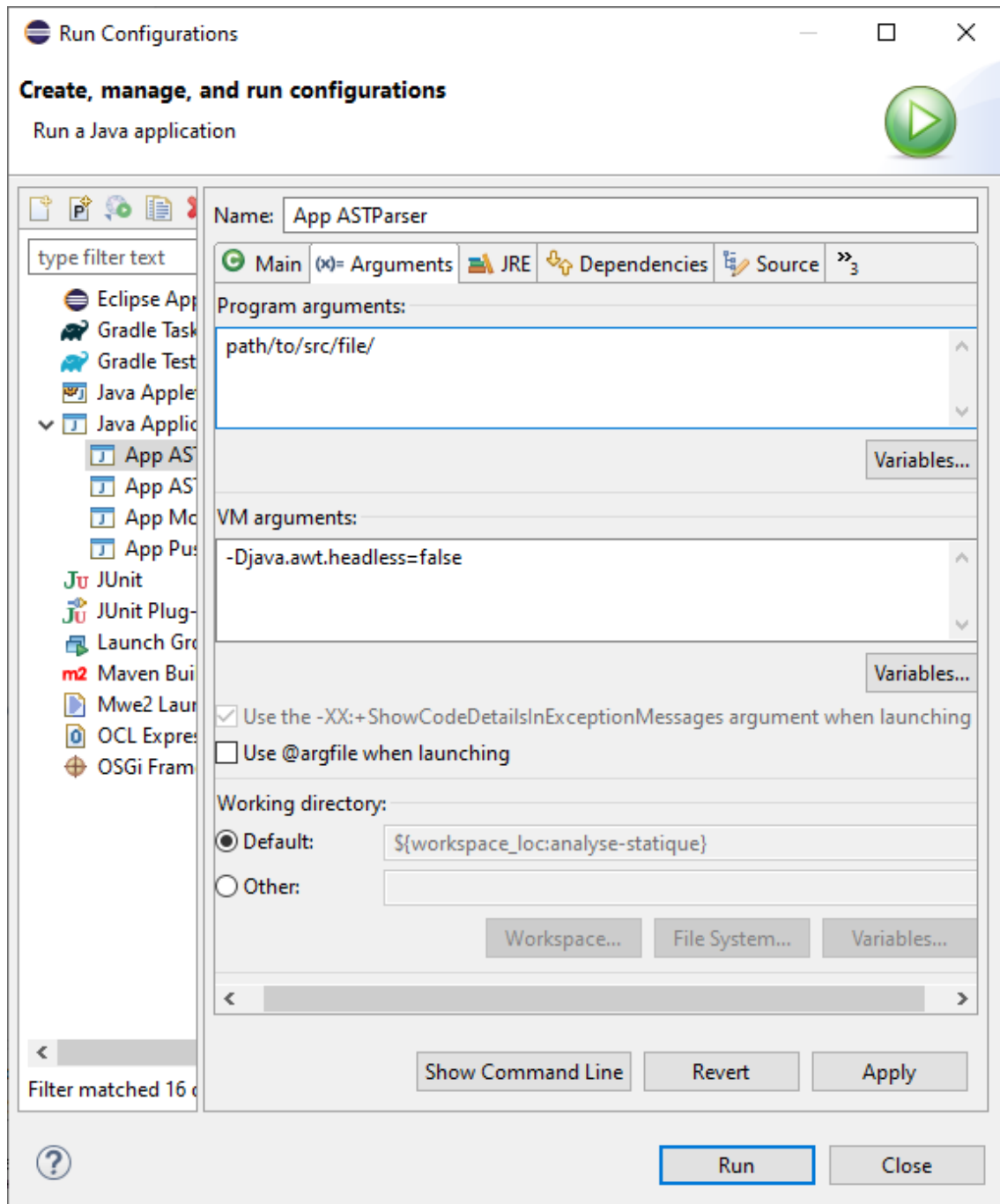
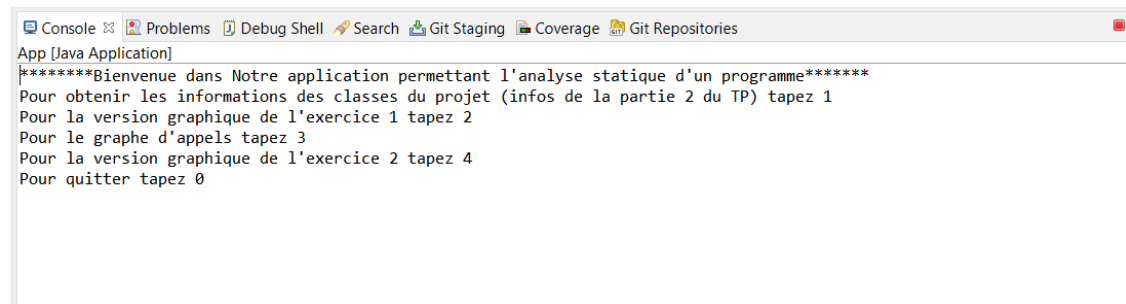


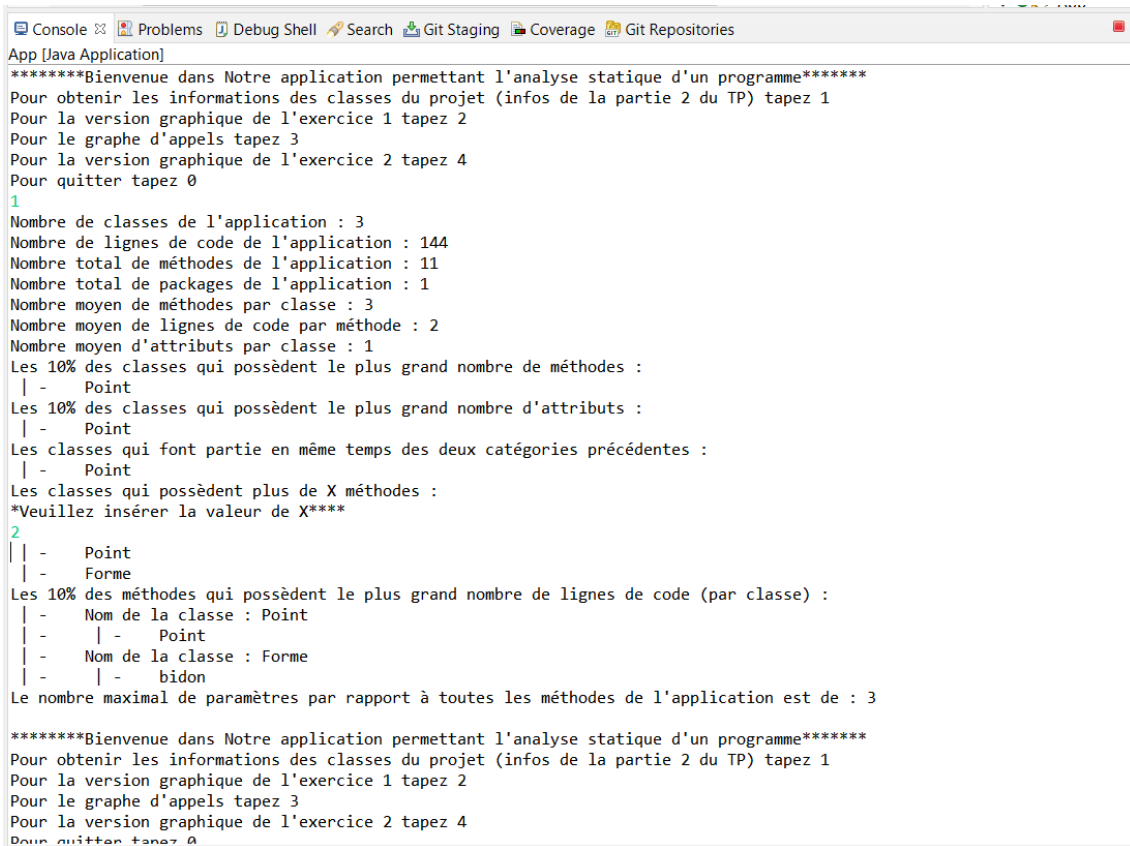
FIGURE 1 – Capture d'écran «Run Configuration» d'Eclipse



```
App [Java Application]
*****Bienvenue dans Notre application permettant l'analyse statique d'un programme*****
Pour obtenir les informations des classes du projet (infos de la partie 2 du TP) tapez 1
Pour la version graphique de l'exercice 1 tapez 2
Pour le graphe d'appels tapez 3
Pour la version graphique de l'exercice 2 tapez 4
Pour quitter tapez 0
```

FIGURE 2 – Capture d’écran du lancement de l’application

En tapant sur 1, on obtient l’affichage en [Figure 3](#) des informations statistiques en interagissant avec la console.



```
App [Java Application]
*****Bienvenue dans Notre application permettant l'analyse statique d'un programme*****
Pour obtenir les informations des classes du projet (infos de la partie 2 du TP) tapez 1
Pour la version graphique de l'exercice 1 tapez 2
Pour le graphe d'appels tapez 3
Pour la version graphique de l'exercice 2 tapez 4
Pour quitter tapez 0
1
Nombre de classes de l'application : 3
Nombre de lignes de code de l'application : 144
Nombre total de méthodes de l'application : 11
Nombre total de packages de l'application : 1
Nombre moyen de méthodes par classe : 3
Nombre moyen de lignes de code par méthode : 2
Nombre moyen d'attributs par classe : 1
Les 10% des classes qui possèdent le plus grand nombre de méthodes :
| - Point
Les 10% des classes qui possèdent le plus grand nombre d'attributs :
| - Point
Les classes qui font partie en même temps des deux catégories précédentes :
| - Point
Les classes qui possèdent plus de X méthodes :
*Veuillez insérer la valeur de X****
2
| | - Point
| | - Forme
Les 10% des méthodes qui possèdent le plus grand nombre de lignes de code (par classe) :
| - Nom de la classe : Point
| - | - Point
| - Nom de la classe : Forme
| - | - bidon
Le nombre maximal de paramètres par rapport à toutes les méthodes de l'application est de : 3
*****Bienvenue dans Notre application permettant l'analyse statique d'un programme*****
Pour obtenir les informations des classes du projet (infos de la partie 2 du TP) tapez 1
Pour la version graphique de l'exercice 1 tapez 2
Pour le graphe d'appels tapez 3
Pour la version graphique de l'exercice 2 tapez 4
Pour quitter tapez 0
```

FIGURE 3 – Capture d'écran de d'interaction sur console avec l'application pour l'obtention des informations statistiques

En tapant sur 2, on obtient l'affichage en [Figure 4](#) des informations statistiques en interagissant avec une interface graphique.

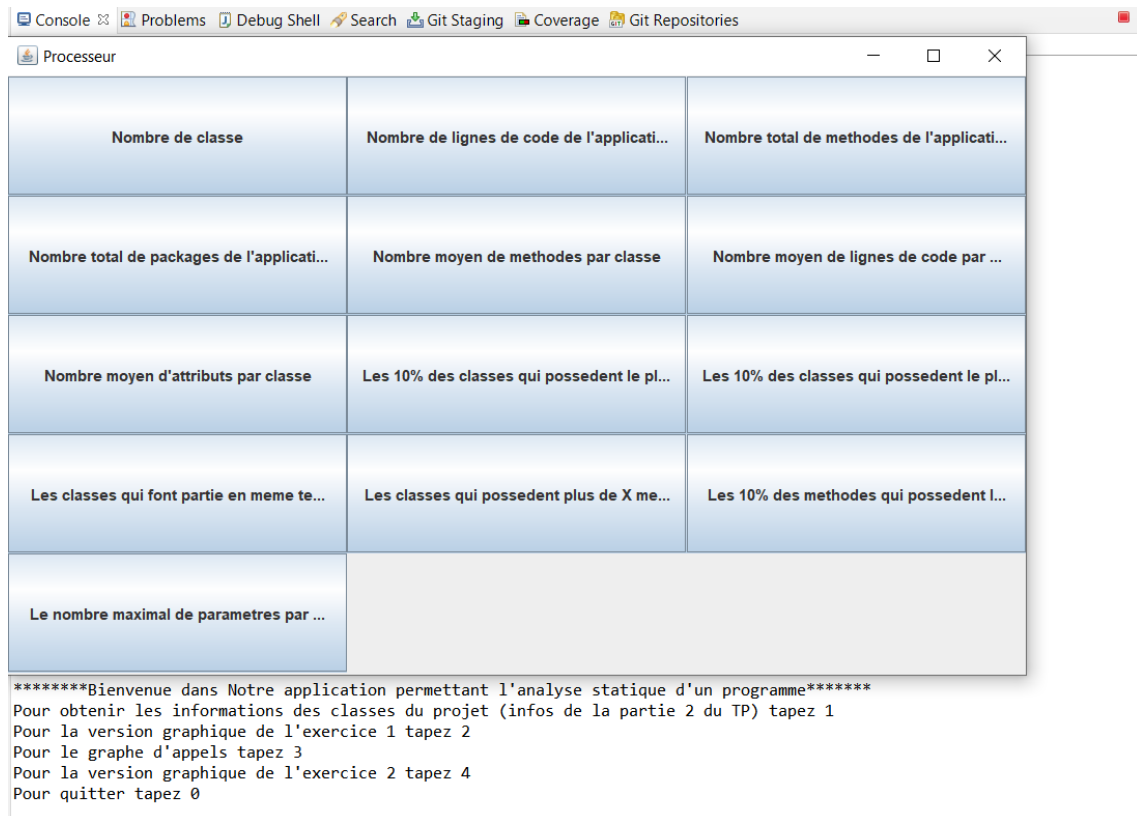


FIGURE 4 – Capture d'écran de l'interface graphique d'interaction pour l'obtention des informations statistiques

En tapant sur 3, on obtient l'affichage en [Figure 5](#) des informations concernant le graphe d'appel en interagissant avec la console.


```
App [Java Application]
*****Bienvenue dans Notre application permettant l'analyse statique d'un programme*****
Pour obtenir les informations des classes du projet (infos de la partie 2 du TP) tapez 1
Pour la version graphique de l'exercice 1 tapez 2
Pour le graphe d'appels tapez 3
Pour la version graphique de l'exercice 2 tapez 4
Pour quitter tapez 0
3
Le graphe d'appels est :
Methode : bidon
Methode(s) sortie : []
Methode(s) entree : []
Methode : bidon2
Methode(s) sortie : []
Methode(s) entree : []
Methode : bidon3
Methode(s) sortie : []
Methode(s) entree : []
Methode : getX
Methode(s) sortie : [getY, getEdgeID]
Methode(s) entree : []
Methode : setX
Methode(s) sortie : []
Methode(s) entree : []
Methode : getY
Methode(s) sortie : [getEdgeID]
Methode(s) entree : [getX]
Methode : setY
Methode(s) sortie : []
Methode(s) entree : []
Methode : getEdgeID
Methode(s) sortie : []
Methode(s) entree : [getX, getY]
Methode : setEdgeID
Methode(s) sortie : []
Methode(s) entree : []

*****Bienvenue dans Notre application permettant l'analyse statique d'un programme*****
Pour obtenir les informations des classes du projet (infos de la partie 2 du TP) tapez 1
```

FIGURE 5 – Capture d'écran de l'affichage sur console du graphe d'appel des méthodes

En tapant sur 4, on obtient l'affichage en [Figure 6](#) des informations concernant le graphe d'appel sous forme d'une image graphique.

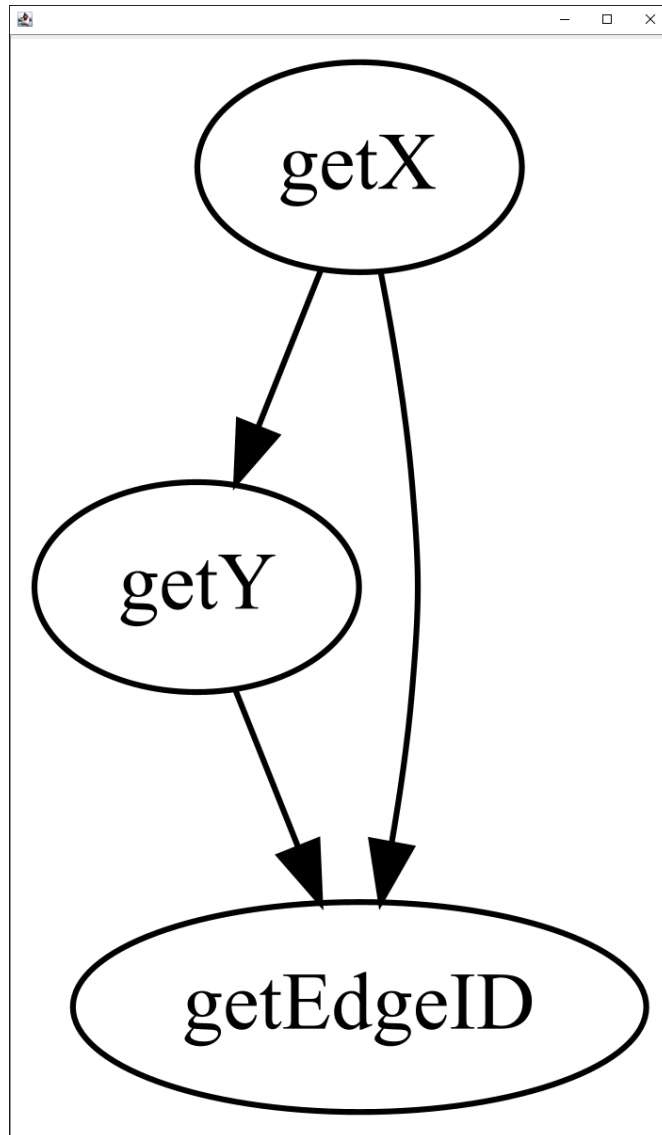


FIGURE 6 – Capture d’écran de l’affichage graphique du graphe d’appel des méthodes

2 Conclusion

Dans ce TP nous avons utilisé Eclipse JDT ASTParser pour réaliser une application permettant l'analyse statique de programme. En se basant sur cette dernière, nous avons calculé un ensemble d'informations statistiques qui pourront par la suite être utilisé pour l'implémentation de métriques. En outre, nous avons construit le graphe d'appels des méthodes du code analysé. Enfin, nous offrons à l'utilisateur la possibilité d'interagir avec l'application via une console ou une interface graphique.

3 Annexe

3.1 Ressources utilisées

1. Documentation de l'API de ASTParser, etc. :
 - <https://help.eclipse.org/latest/topic/org.eclipse.jdt.doc.isv/reference/api/org/eclipse/jdt/core/dom/package-summary.html>.
2. Syntaxe de visualisation de graphe :
 - <https://graphviz.org/>,
 - <https://dreampuf.github.io/GraphvizOnline/>.
3. Transformation Dot au format PNG en Java :
 - <https://github.com/nidi3/graphviz-java>.
4. Interface graphique :
 - <https://www.javatpoint.com/java-swing>.
5. Affichage d'image dans une interface graphique Swing :
 - <https://www.dummies.com/programming/java/how-to-write-java-code-to-show-an-image-on-the-screen/>.

Références bibliographiques

- [1] Abdelhak-Djamel SERIAI. *Évolution et maintenance des systèmes logiciels*. Paris : Hermès Science publications Lavoisier, 2014. ISBN : 9782746245549.