Rapport tp Analyse statique

Meryam RHADI
Lamyae KHAIROUN

5 novembre 2022

Master 2
Génie logiciel

UE Évolution et Restructuration des Logiciels

Responsable
Djamel Seriai
Bachar Rima

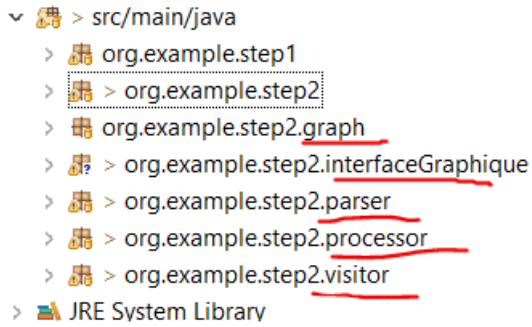
Sommaire

1 Introduction

Dans ce TP, on a fait une application qui permet l'analyse statique du code source d'une autre application java orientée objet, en utilisant le patron de conception **visiteur**, afin d'extraire des informations qui vont nous faciliter la compréhension de l'application.

2 Structure de l'application

Pour encapsulé nos classes dans différente packages distincts pour avoir une architecture bien organisée.



- **processor** : Ce package contient la classe **Processor** qui fait la logique métier de notre application.
- **parser** :Contient la classe **MyParser** qu'on a utilisé pour parser le code le code d'une application java.
- **graph** : Contient les classe **Graph** et **Vertex** qui encapsule le traitement qui concerne la création du graphe d'appel.
- **visitor** : Contient tous les classes visiteurs.
- **interfaceGraphique** : Contient la classe de l'interface graphique.

3 Lien vers github

https://github.com/lamyaeKHAIROUN/AnalyseStatTP1

4 Exécution de l'application

Après l'exécution de notre application on obtient le menu suivant, et on peut soit utiliser la console, ou l'interface graphique (plus détaillée dans la section 6).

```
Main [Java Application] C:\Program Files\Java\jdk-19\bin\javaw.exe  (13 oct. 2022, 22:17:23) [pid: 4412]
************************************ Bienvenue dans l'application d'analyse statique
- Tapez 1 pour continuer au console
- Tapez 2 pour continuer sur l'interface graphique
1
```

Si on tape 1, c'est-à-dire qu'on a choisi la console, donc on aura un autre menu qui nous propose de taper un numéro selon ce qu'on désire afficher parmi l'analyse statique qu'on a effectuée.

```
************************************ Bienvenue dans l'application d'analyse statique de code  **************
- Tapez 1 pour continuer au console
- Tapez 2 pour continuer sur l'interface graphique
1
- Tapez 1 pour afficher le nombre de classes de l'application
- Tapez 2 pour afficher le nombre de lignes de code de l'application
- Tapez 3 pour afficher nombre total de m□thodes de l'application
- Tapez 4 pour afficher nombre total de packages de l'application
- Tapez 5 pour afficher nombre moyen de m□thodes par classe
- Tapez 6 pour afficher nombre moyen de lignes de code  par m□thode
- Tapez 7 pour afficher nombre moyen d'attributs par classe
- Tapez 8 pour afficher les 10% des classes avec le plus grand nbr de m□thode
- Tapez 9 pour afficher les 10% des classes avec le plus grand nbr d'attributs
- Tapez 10 pour afficher les 10% des classes avec le plus grand nbr d'attributs et m□thodes
- Tapez 11 pour afficher les classes ayant un nombre de m□thodes sup□rieur □ X
- Tapez 12 pour afficher les 10% des classes avec le plus grand nbr d'attributs et m□thodes
- Tapez 13 pour Le nombre maximal de param par rapport □ tous les meth de l'appli
- Tapez 0 pour quitter le menu
```

La figure ci-dessus, affiche un exemple de l'affichage de la fonction qui calcul le nombre des classes dans une application donnée.

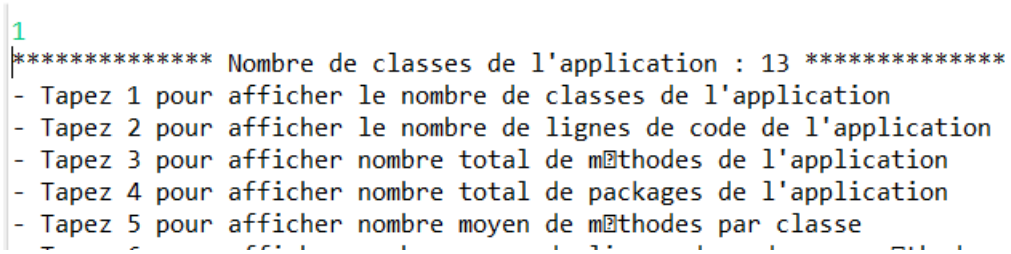


Figure 1. Exemple d'affichage

5 Le graphe d'appel

Pour répondre à cette question, on a crée une structure spécifique pour stocké notre graphe d'appel ; alors on a créé une classe **Graph** qui contient notre structure de données, et une classe **Vertex** qui corresponds aux noeuds, où les noeuds sont les méthodes préfixées par leur type statique du receveur, et les arrêtes correspondent aux invocations des autre méthodes. Pour afficher notre graphe on a utilisé la bibliothèque JGraphT qui permet d'exporter le graphe en format png.
Dans cette image il y'a un extrait de l'image de notre graphe. L'image rentières elle est dans le zip.

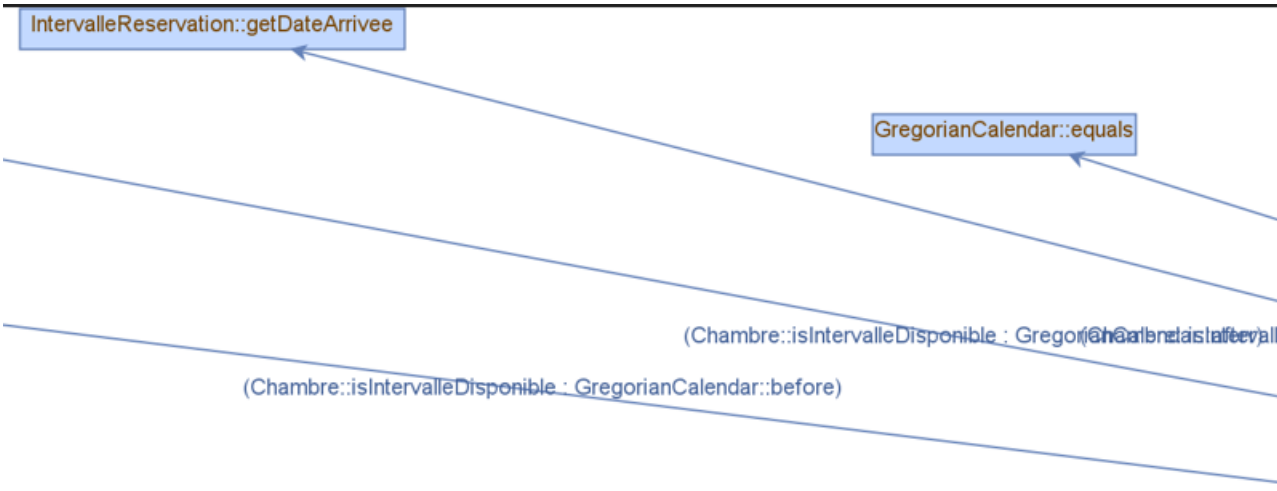


Figure 2. Exemple d'affichage du graphe d'appel

```
Graph d'appel:
Adresse::hashCode: ==> hash
Hotel::equals: ==> getClass  equals
Lit::hashCode: ==> hash
CarteBancaire::equals: ==> getClass  equals
Chambre::equals: ==> getClass  equals  doubleToLongBits
Hotel::addChambre: ==> add
AgenceController::getAllAgences: ==> findAll
Chambre::addLit: ==> add
Client::hashCode: ==> hash
Lit::equals: ==> getClass
CarteBancaire::hashCode: ==> hash
Agence::hashCode: ==> hash
PositionGPS::equals: ==> getClass  doubleToLongBits
Chambre::getNbPlaces: ==> getNbPlaces
Client::equals: ==> getClass  equals
IntervalleReservation::toString: ==> get
Chambre::setCalendrierReservationAVenir: ==> add
Agence::addHotel: ==> add
Chambre::hashCode: ==> hash
```

Figure 3. Exemple d'affichage du graphe d'appel au console

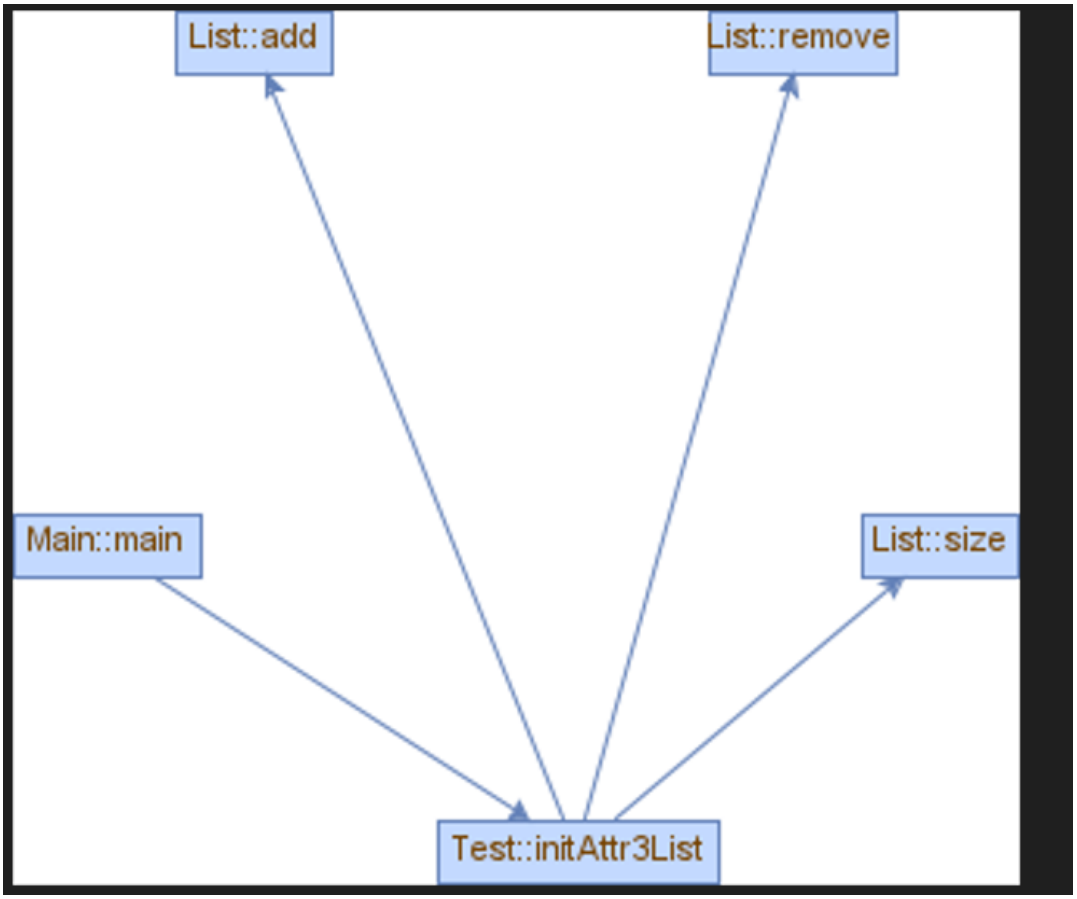


Figure 4. Exemple d'affichage du graphe d'appel d'un petit projet

6 Interface graphique

Pour la création de l'interface graphique, on a utilisé la bibliothèque **Swing**, en utilisant plusieurs butons, chaque bouton permet de répondre à une question du tp, en affichant les informations correspondant en cliquant sur un bouton.

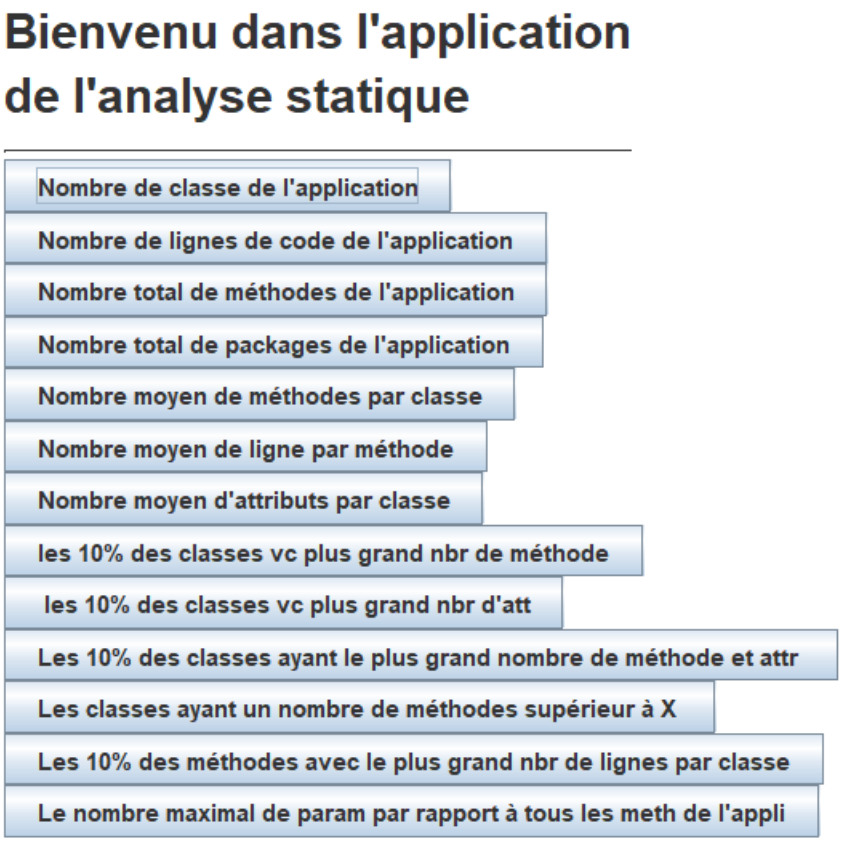


Figure 5. Le menu principale

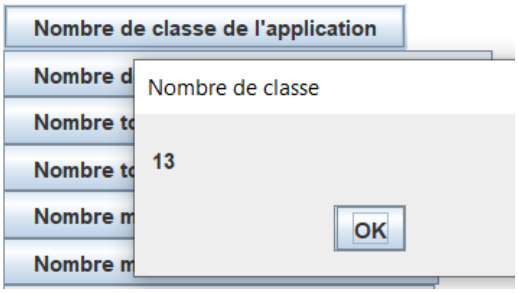


Figure 6. Réponse lors du clique sur bouton 1

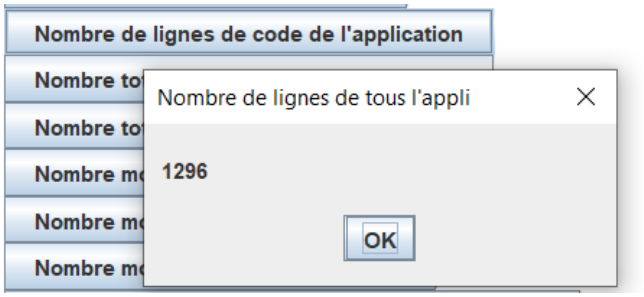


Figure 7. Réponse lors du clique sur bouton 2

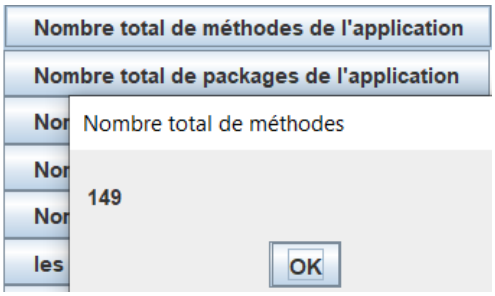


Figure 8. Réponse lors du clique sur bouton 3

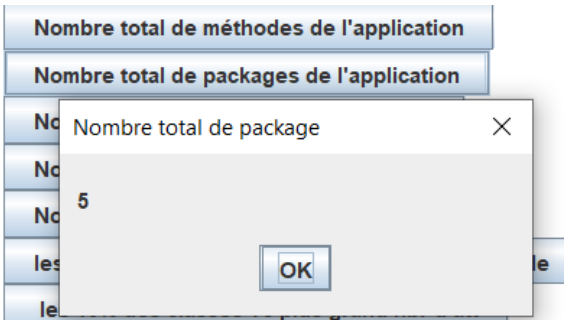


Figure 9. Réponse lors du clique sur bouton 4

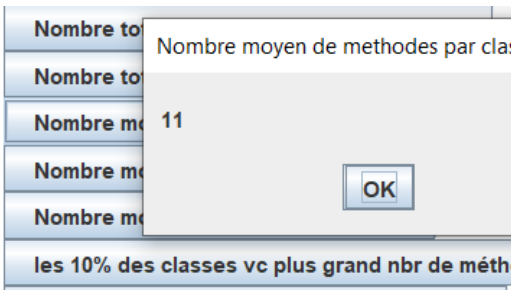


Figure 10. Réponse lors du clique sur bouton 5

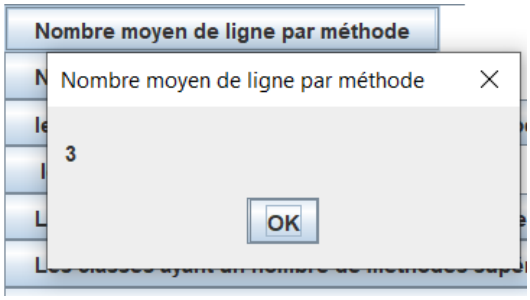


Figure 11. Réponse lors du clique sur bouton 6

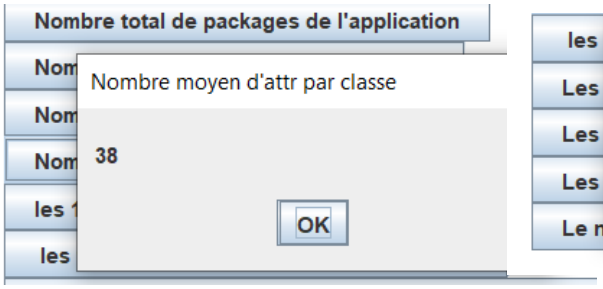


Figure 12. Réponse lors du clique sur bouton 7

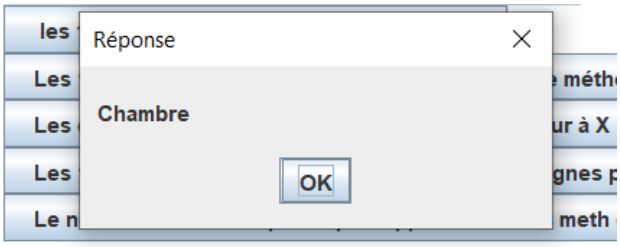


Figure 13. Réponse lors du clique sur bouton 8

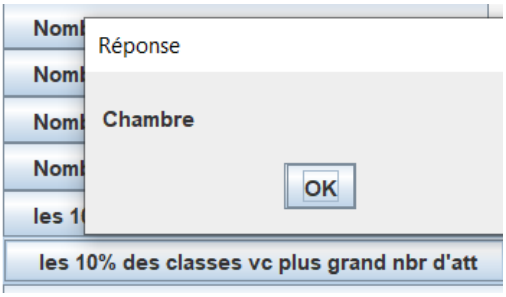


Figure 14. Réponse lors du clique sur bouton 9

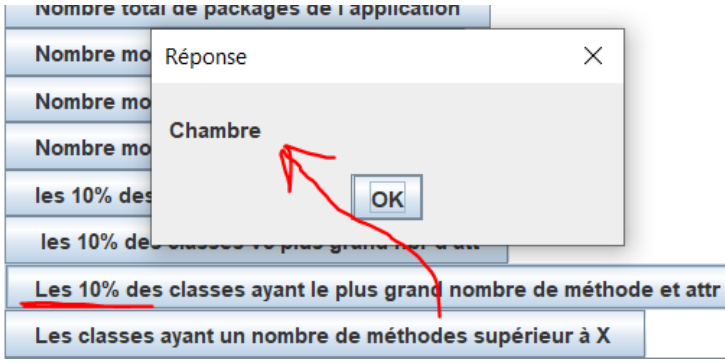


Figure 15. Réponse lors du clique sur bouton 10

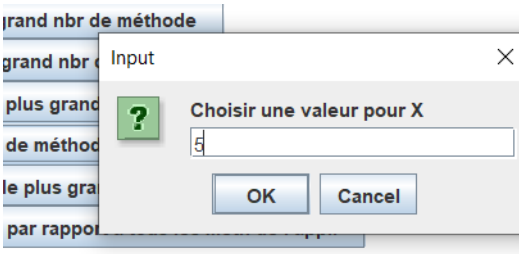


Figure 16. Réponse lors du clique sur bouton 11 - Demande de X

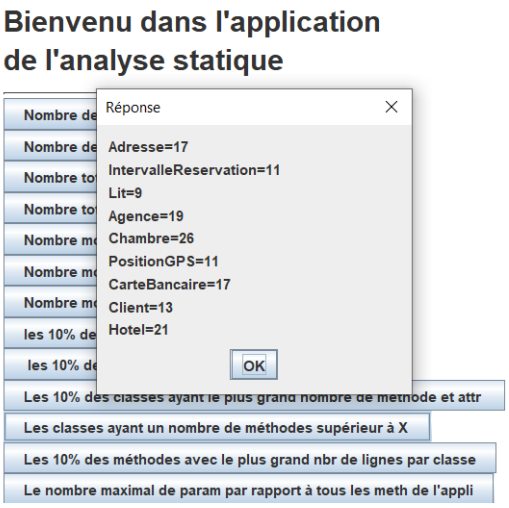


Figure 17. Réponse lors du clique sur bouton 11

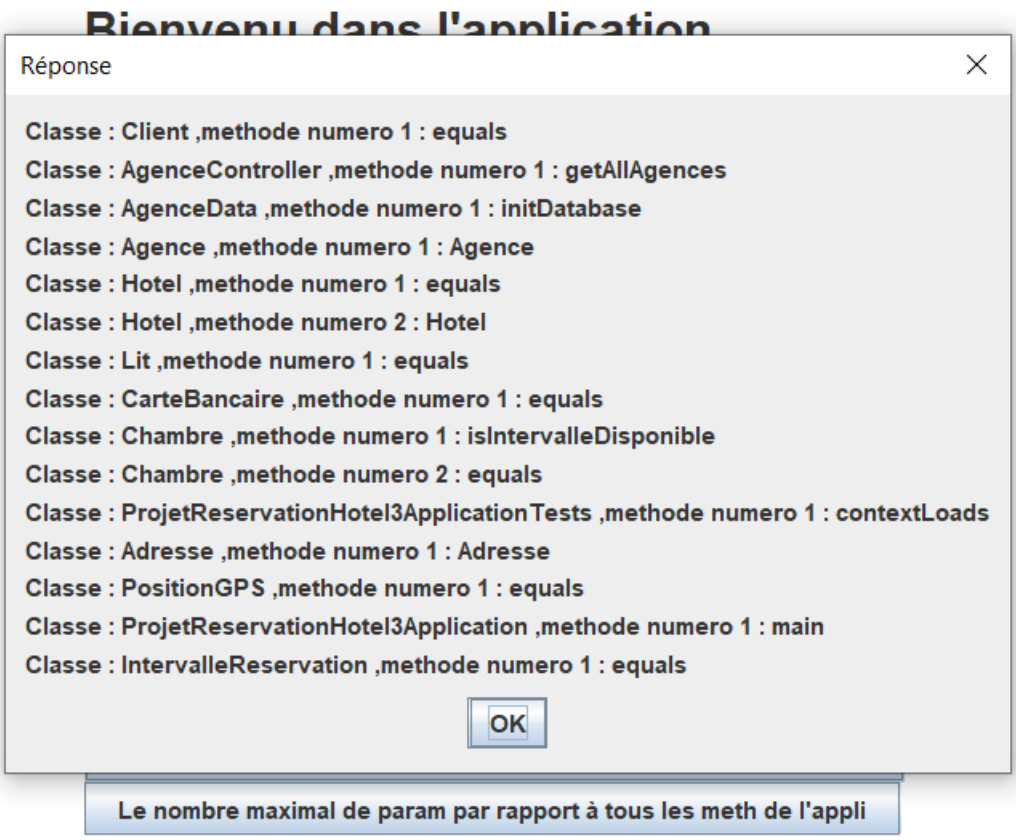


Figure 18. Réponse lors du clique sur bouton 12

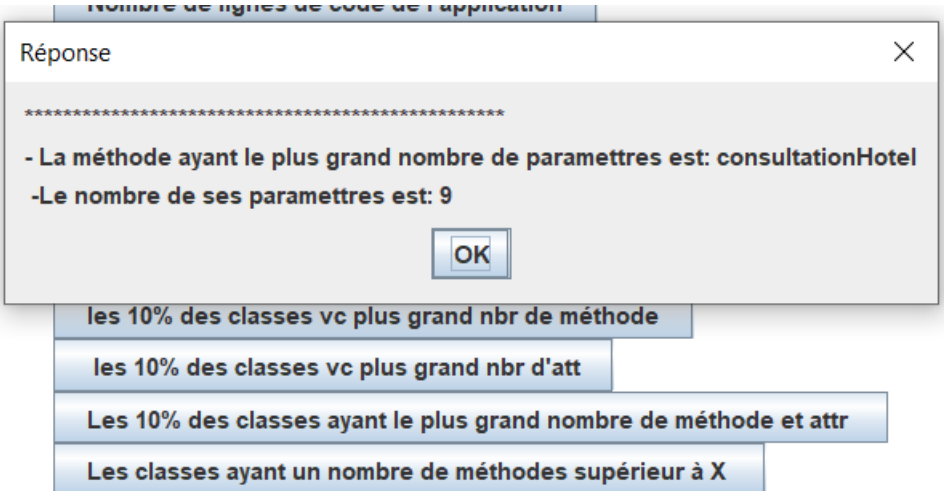


Figure 19. Réponse lors du clique sur bouton 13

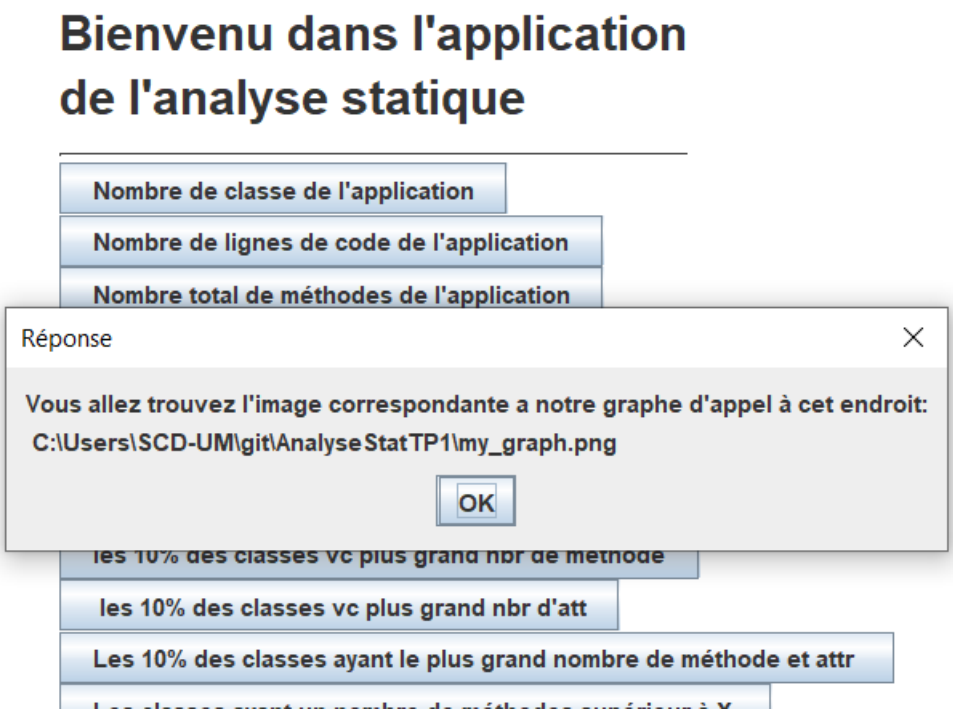


Figure 20. Réponse lors du clique sur bouton 14

7 Tester l'application

Pour tester l'application sur Eclipse,
1)Il faut aller à la classe Main qui se trouve dans le package main et donner le chemin d'accès de l'application que vous voulez analyser en paramètre du constructeur de la classe Processor.
2) Il faut aller dans la classe MyParser qui se trouve dans le package parser–> puis aller dans la variable jrePath et donner le chemin d'accees de jre.
3) Lancer la classe Main.

```java
public static void Menu() throws FileNotFoundException, IOException {
    Scanner sc = new Scanner(System.in);

    int choix = 0;
    Processor processor;
    processor = new Processor("C:\\Users\\SCD-UM\\Downloads\\Comparateur\\Comparateur
    // processor.parseClasses();
    processor.parsePackages();
    InterfaceGraphique i = new InterfaceGraphique();

    // 1
    // processor.parseV();
    while (true) {
```