CHAMROUK Laïla
MAJDOUL Kaoutar
M2 GL

TP2 Partie 1 : Évolution et Restructuration des Logiciels - Analyse statique et dynamique

Le code se trouve principalement dans le fichier Parser.java

Exercice 1

1. Nombre de classes dans l'application

Nous avons le code suivant :

```
public static void main(String[] args) throws IOException {
  ArrayList<String> listClassInApp=new ArrayList<String>();
  ArrayList<File> javaFiles = listJavaFilesForFolder(folder,
listClassInApp);

//...

  System.out.println("\n ---1. NOMBRE DE CLASSES DANS
L'APPLICATION---");
  int nbOfClassInApp = listClassInApp.size();
  System.out.println(nbOfClassInApp);
}
```

La fonction `listJavaFilesForFolder(final File folder, ArrayList<String> listClass)` permet de prendre en paramètre une liste `listClass` qui va contenir les noms des classes qui sont rencontrées au fur et à mesure de la lecture des fichiers java (les fichiers java étant les noms des classes).
Une fois instanciée, nous pouvons appeler la fonction `size()` sur la liste `listClassInApp` qui a été passé en paramètre, et ainsi obtenir le nombre de classes.

2. Nombre de lignes de code dans l'application

Nous avons le code suivant :

```
public static void main(String[] args) throws IOException {
  long lines = 0;

  for (File fileEntry : javaFiles) {
    String content = FileUtils.readFileToString(fileEntry);
    //qu2
```

```
    lines += content.lines().count();
  }
  System.out.println("\n ---2. NOMBRE DE LIGNES DE CODE DANS
L'APPLICATION---");
  System.out.println(lines);
}
```

Pour compter le nombre de lignes total dans l'application, nous utilisons la variable `content` qui contient le code d'un fichier sous forme de chaîne de caractères. Nous pouvons ensuite appeler `lines().count()` sur cette variable pour compter le nombre de lignes dans un fichier. Puis nous ajoutons ce chiffre obtenu à lines, qui contiendra le nombre total de lignes dans l'application.

3. Nombre total de méthodes de l'application

Nous avons le code suivant:

```
public static void main(String[] args) throws IOException {
  //qu3
  int nbOfMethodInClass = 0;
  ArrayList<String> listMethodInApp=new ArrayList<String>();

  for (File fileEntry : javaFiles) {
    CompilationUnit parse = parse(content.toCharArray());
    //qu3
    nbOfMethodInClass = navigateMethodInfo(parse, listMethodInApp);
  }
  System.out.println("\n ---3. NOMBRE TOTAL DE METHODES DANS
L'APPLICATION---");
  int nbOfMethodInApp = listMethodInApp.size();
  System.out.println(nbOfMethodInApp);
}
```

La méthode `navigateMethodInfo` prend en paramètre une liste `listMethodInApp` qui va contenir le nom de toutes les méthodes de l'application.
Une fois instanciée, il suffit de récupérer la taille de cette liste pour obtenir le nombre de méthodes dans l'application.

4. Nombre total de packages de l'application

Nous avons le code suivant:

```
public static void main(String[] args) throws IOException {
  //qu4
  HashMap<String, Integer> listPackageInApp = new HashMap<String,
```

```
Integer>();

  for (File fileEntry : javaFiles) {
    CompilationUnit parse = parse(content.toCharArray());
    //qu4
    navigatePackageInfo(parse, listPackageInApp);
  }
  System.out.println("\n ---4. NOMBRE TOTAL DE PACKAGES DANS
L'APPLICATION---");
  int nbOfPackagesInApp = listPackageInApp.keySet().size();
  System.out.println(nbOfPackagesInApp);
}
```

Pour cette solution, nous avons dû créer un `PackageDeclarationVisitor.java` spécifiquement pour les packages.
La méthode `navigatePackageInfo` prend en paramètre une hashmap `listPackageInApp` qui va contenir les paires (`nomPackage`, `noOfPackage`). Nous avons choisi une hashmap au lieu d'une liste car nous nous sommes rendu compte qu'au moment de renvoyer la taille de la liste, celle-ci comptait les doubles. L'utilisation d'une hashmap permet de s'assurer de l'unicité des clés lors du comptage, et donc permet de retourner le bon nombre de packages une fois appelée avec la fonction `size()`.
Il suffit donc de récupérer la taille de cette hashmap pour obtenir le nombre de packages dans l'application.

5. Nombre moyen de méthodes par classe

Nous avons le code suivant:

```
public static void main(String[] args) throws IOException {
  System.out.println("\n ---5. NOMBRE MOYEN DE METHODES PAR CLASSE---");
  System.out.println(nbOfMethodInApp/nbOfClassInApp);
}
```

Pour cette question, nous avons utilisé les résultats des questions 3 et 1. En effet, le nombre moyen de méthodes par classe est le résultat de la division entre le nombre de méthodes total et le nombre de classes de l'application.

6. Nombre moyen de lignes de code par méthode

Nous avons le code suivant:

```
public static void main(String[] args) throws IOException {
  //qu6
  ArrayList<Integer> listLinesInMethod=new ArrayList<Integer>();
```

```
  for (File fileEntry : javaFiles) {
    CompilationUnit parse = parse(content.toCharArray());
    //qu6
    numberOfLineInMethod(parse, listLinesInMethod);
  }
  System.out.println("\n ---6. NOMBRE MOYEN DE LIGNES DE CODE PAR
METHODE---");
  int c=0;
  for(int nbOfLineInMethod : listLinesInMethod) {
    c+=nbOfLineInMethod;
  }
  System.out.println(c/nbOfMethodInApp);
}
```

La fonction `numberOfLineInMethod` prend en paramètre une liste `listLinesInMethod` qui va contenir le nombre de lignes pour chaque méthode.
Une fois l'appel fait, il suffit de récupérer les valeurs de la liste et de les ajouter dans une variable afin d'obtenir le nombre de lignes total pour les méthodes à travers l'application.
Enfin , pour obtenir une moyenne, nous divisons cette variable par le nombre de méthodes dans l'application.

7. Nombre moyen d'attributs par classes

Nous avons le code suivant:

```
public static void main(String[] args) throws IOException {
  //qu7
  int nbOfVariablesInClass = 0;
  ArrayList<Integer> listNbOfVariables=new ArrayList<Integer>();

  for (File fileEntry : javaFiles) {
    CompilationUnit parse = parse(content.toCharArray());
    //qu7
    nbOfVariablesInClass = navigateVariableInfo(parse,
listNbOfVariables);
  }
  System.out.println("\n ---7. NOMBRE MOYEN D'ATTRIBUTS PAR CLASSE---");
  int d = 0;
  for(int nbOfVariables : listNbOfVariables) {
    d+=nbOfVariables;
  }
  System.out.println(d/nbOfClassInApp);
}
```

La fonction `navigateVariableInfo` prend en paramètre une liste `listNbOfVariables` qui va contenir le nombre de variable pour chaque méthodes visitées.

Une fois appelé, il suffit de récupérer la liste d'entiers et de les ajouter dans une variable afin d'obtenir le nombre total de variables, qu'on divise ensuite par le nombre de classes. On obtient ainsi une moyenne.

8. Les 10% de classes qui possèdent le plus grand nombre de méthodes

Nous avons le code suivant:

```
public static void main(String[] args) throws IOException {
  //qu8
  HashMap<String, Integer> nbOfMethodPerClass = new HashMap<String,
Integer>();
  for (File fileEntry : javaFiles) {
    CompilationUnit parse = parse(content.toCharArray());
    //qu8
    nbOfMethodPerClass.put(fileEntry.getName(), nbOfMethodInClass);
  }
  System.out.println("\n ---8. LES 10% DE CLASSES QUI ONT LE PLUS DE
METHODES---");
  HashMap<String, Integer> nbOfMethodPerClassSorted =
sortByValue(nbOfMethodPerClass);
  ArrayList<String> listKey1=new ArrayList<String>();
  float tenpcNbMethodClass = (float)(nbOfClassInApp*10)/100;
  int i=0;
  for (String name: nbOfMethodPerClassSorted.keySet()) {
    if(i<Math.ceil(tenpcNbMethodClass)) {
      String key = name.toString();
      listKey1.add(key);
      String value = nbOfMethodPerClassSorted.get(name).toString();
      System.out.println("La classe " + key + " a " + value + "
méthode(s).");
      i++;
    }
  }
}
```

Pour cette question, nous avons utilisé la question 3 afin d'obtenir le nombre de méthodes par classe. Nous mettons cette information dans une hashmap `nbOfMethodPerClass` avec le nom de la classe afin d'obtenir les couples (`nomClasse`, `nbOfMethod`). Nous trions également cette hashmap suivant les valeurs afin d'obtenir une liste `nbOfMethodPerClassSorted`.
Une fois cet appel effectué, pour chaque clé de cette hashmap, nous effectuons un affichage de la méthode et du nombre de lignes tant qu'elle fait partie des 10%.

CHAMROUK Laïla
MAJDOUL Kaoutar
M2 GL

9. Les 10% de classes qui possèdent le plus grand nombre d'attributs

Nous avons le code suivant:

```
public static void main(String[] args) throws IOException {
  //qu9
  HashMap<String, Integer> nbOfVariablesPerClass = new HashMap<String,
Integer>();

  for (File fileEntry : javaFiles) {
    CompilationUnit parse = parse(content.toCharArray());
    //qu9
    nbOfVariablesPerClass.put(fileEntry.getName(),
nbOfVariablesInClass);
  }
  System.out.println("\n ---9. LES 10% DE CLASSES QUI ONT LE PLUS
D'ATTRIBUTS---");
  HashMap<String, Integer> nbOfVariablesPerClassSorted =
sortByValue(nbOfVariablesPerClass);
  ArrayList<String> listKey2=new ArrayList<String>();
  float tenpcNbVariableClass = (float)(nbOfClassInApp*10)/100;
  int j=0;
  for (String name: nbOfVariablesPerClassSorted.keySet()) {
    if(j<Math.ceil(tenpcNbVariableClass)) {
      String key = name.toString();
      listKey2.add(key);
      String value = nbOfVariablesPerClassSorted.get(name).toString();
      System.out.println("La classe " + key + " a " + value + "
variable(s).");
      j++;
    }
  }
}
```

Même principe que la question précédente, cette fois-ci, nous utilisons la question 7 pour obtenir le nombre d'attributs par classe.

10. Les classes qui font partie des deux catégories précédentes

Nous avons le code suivant:

```
public static void main(String[] args) throws IOException {
  System.out.println("\n ---10. LES CLASSES QUI FONT PARTIE DES 2
CATEGORIES PRECEDENTES EN MÊME TEMPS---");
  ArrayList<String> compareKeys=new ArrayList<String>(listKey1);
```

```
  compareKeys.retainAll(listKey2);
  System.out.println(compareKeys);
}
```

Pour cette question, nous avons préalablement enregistré les résultats des deux précédentes questions dans des listes distinctes. Il nous suffit donc de comparer ces deux listes et de renvoyer les objets communs.

11. Les classes qui possèdent plus de X méthodes

Nous avons le code suivant:

```
public static void main(String[] args) throws IOException {
  System.out.println("\n ---11. LES CLASSES QUI ONT PLUS DE X METHODES
(ici 5)---");
  for (String name: nbOfMethodPerClassSorted.keySet()) {
    String key = name.toString();
    String value = nbOfMethodPerClassSorted.get(name).toString();
    if(Integer.parseInt(value) > 5) {
      System.out.println("La classe " + key + " a " + value + "
méthode(s).");
    }
  }
}
```

Pour cette question, nous utilisons la liste `nbOfMethodPerClassSorted` de la question 8. Il nous suffit de conditionner l'affichage par un if avec une valeur choisie.

12. Les 10% des méthodes qui possèdent le plus grand nombre de lignes de code (par classe).

Nous avons le code suivant:

```
public static void main(String[] args) throws IOException {
  //qu12
  ArrayList<String> question12=new ArrayList<String>();

  for (File fileEntry : javaFiles) {
    //... ici nous récupérons le nombre de lignes par method et nous le
stockons dans une liste question12, qui va contenir les affichages
  }
  System.out.println("\n ---12. LES 10% DE METHODES QUI ONT LE PLUS
GRAND NOMBRE DE LIGNES DE CODE (par classes)---");
    for(int q12=0; q12<question12.size(); q12++) {
      System.out.println(question12.get(q12));
```

```
    }
}
```

Une fois la liste `question12` remplie avec les affichages qui permettent de connaître les méthodes contenant le plus grand nombre de méthodes par classe, nous affichons chacun de ceux-ci.

13. Le nombre maximal de paramètres

Nous avons le code suivant:

```
public static void main(String[] args) throws IOException {
  //qu13
  long nbParamInClass = 0;
  ArrayList<Integer> listNbMaxParam=new ArrayList<Integer>();

  for (File fileEntry : javaFiles) {
    //qu13
    nbParamInClass = navigateParamInfo(parse);
    listNbMaxParam.add((int) nbParamInClass);
  }
  System.out.println("\n ---13. NOMBRE MAXIMAL DE PARAMETRES PARMI
TOUTES LES METHODES DE L'APPLICATION---");
  System.out.println(Collections.max(listNbMaxParam));
}
```

La fonction `navigateParamInfo` renvoie le nombre de paramètres dans une classe. Nous stockons cette valeur dans une liste `listNbMaxParam`. Une fois complète, nous récupérons les entiers de cette liste et nous affichons la valeur max.

Une fois le programme `Parser.java` exécuté, voilà le résultat renvoyé en console :

CHAMROUK Laïla
MAJDOUL Kaoutar
M2 GL

```
 ---1. NOMBRE DE CLASSES DANS L'APPLICATION---
8

 ---2. NOMBRE DE LIGNES DE CODE DANS L'APPLICATION---
204

 ---3. NOMBRE TOTAL DE METHODES DANS L'APPLICATION---
31

 ---4. NOMBRE TOTAL DE PACKAGES DANS L'APPLICATION---
3

 ---5. NOMBRE MOYEN DE METHODES PAR CLASSE---
3

 ---6. NOMBRE MOYEN DE LIGNES DE CODE PAR METHODE---
4

 ---7. NOMBRE MOYEN D'ATTRIBUTS PAR CLASSE---
2

 ---8. LES 10% DE CLASSES QUI ONT LE PLUS DE METHODES---
La classe Personne.java a 8 méthode(s).

 ---9. LES 10% DE CLASSES QUI ONT LE PLUS D'ATTRIBUTS---
La classe TestTableau.java a 4 variable(s).

 ---10. LES CLASSES QUI FONT PARTIE DES 2 CATEGORIES PRECEDENTES EN MÊME TEMPS---
[]

 ---11. LES CLASSES QUI ONT PLUS DE X METHODES (ici 5)---
La classe Personne.java a 8 méthode(s).
La classe CPile.java a 7 méthode(s).

 ---13. NOMBRE MAXIMAL DE PARAMETRES PARMI TOUTES LES METHODES DE L'APPLICATION---
2

 ---12. LES 10% DE METHODES QUI ONT LE PLUS GRAND NOMBRE DE LIGNES DE CODE (par classes)---
Dans le fichier ElementAvecPriorite.java
La méthode priorite contient 1 ligne(s) de code.

Dans le fichier FileAttente.java
La méthode sort contient 14 ligne(s) de code.

Dans le fichier Personne.java
La méthode priorite contient 5 ligne(s) de code.

Dans le fichier CPile.java
La méthode toString contient 7 ligne(s) de code.

Dans le fichier IPile.java
La méthode sommet contient 1 ligne(s) de code.

Dans le fichier TestPile.java
La méthode main contient 16 ligne(s) de code.

Dans le fichier Tableau.java
La méthode triBulle contient 14 ligne(s) de code.

Dans le fichier TestTableau.java
La méthode main contient 14 ligne(s) de code.
```

CHAMROUK Laïla
MAJDOUL Kaoutar
M2 GL

Exercice 2

Nous avons le code suivant :

```
public static void main(String[] args) throws IOException {
  //qu13
  long nbParamInClass = 0;
  ArrayList<Integer> listNbMaxParam=new ArrayList<Integer>();

  for (File fileEntry : javaFiles) {
    //ex2
    printMethodInvocationInfo(parse);
  }
}
```

La methode `printMethodInvocationInfo` permet d'afficher les noms des méthodes ainsi que les méthodes appelées. C'est ainsi que l'arbre est affiché, avec chacune des méthodes dans l'arborescence et les appels.

Une fois exécuté, le code renvoie le résultat suivant :

CHAMROUK Laïla
MAJDOUL Kaoutar
M2 GL

```
 ---GRAPHE D'APPEL---
Method name: priorite Return type: int
Method name: FileAttente Return type: null
Method name: entre Return type: void
                |___ method entre invoc method add
Method name: sort Return type: A
                |___ method sort invoc method isEmpty
                |___ method sort invoc method size
                |___ method sort invoc method priorite
                |___ method sort invoc method get
                |___ method sort invoc method priorite
                |___ method sort invoc method get
                |___ method sort invoc method get
                |___ method sort invoc method remove
Method name: estVide Return type: boolean
                |___ method estVide invoc method isEmpty
Method name: toString Return type: String
Method name: Personne Return type: null
Method name: Personne Return type: null
Method name: getNom Return type: String
Method name: setNom Return type: void
Method name: getAge Return type: int
Method name: setAge Return type: void
Method name: toString Return type: String
Method name: priorite Return type: int
Method name: CPile Return type: null
Method name: estVide Return type: boolean
                |___ method estVide invoc method size
Method name: empile Return type: void
                |___ method empile invoc method addFirst
Method name: depile Return type: A
                |___ method depile invoc method removeFirst
Method name: nbElement Return type: int
                |___ method nbElement invoc method size
Method name: sommet Return type: A
                |___ method sommet invoc method element

Method name: toString Return type: String
                |___ method toString invoc method toString
Method name: estVide Return type: boolean
Method name: empile Return type: void
Method name: depile Return type: A
Method name: nbElement Return type: int
Method name: sommet Return type: A
Method name: main Return type: void
                |___ method main invoc method println
                |___ method main invoc method empile
                |___ method main invoc method println
                |___ method main invoc method depile
                |___ method main invoc method println
                |___ method main invoc method println
                |___ method main invoc method empile
                |___ method main invoc method empile
                |___ method main invoc method empile
                |___ method main invoc method empile
                |___ method main invoc method println
                |___ method main invoc method println
                |___ method main invoc method nbElement
Method name: Tableau Return type: null
Method name: triBulle Return type: void
                |___ method triBulle invoc method compareTo
Method name: toString Return type: String
                |___ method toString invoc method toString
Method name: main Return type: void
                |___ method main invoc method println
                |___ method main invoc method println
                |___ method main invoc method triBulle
                |___ method main invoc method println
                |___ method main invoc method println
                |___ method main invoc method println
                |___ method main invoc method triBulle
                |___ method main invoc method println
```