Université De La Rochelle

Master ICONE 1ère année

TP Qualité du logiciel

Réaliser par :
Mohammed BENAOU
Walid CHERKAOUI

2017-2018

Table des matières

1 Correction de code **3**
1.1 Calcul de l'efferent/afferent coupling pour les classes 8
1.2 Complexité cyclomatique 8
1.3 Lack Of Cohesion Of Methods 8

2 Code source **9**
2.1 La classe Somme 9
2.2 La classe Arc 9
2.3 La classe Node 11
2.4 La classe Graphe 13
2.5 La classe TPGraphe 22

Table des figures

1 le résultat du première analyse 3
2 Erreur de la concaténation des chaînes de caractère 4
3 le résultat final analyse 7
4 le résultat final analyse 8

1 Correction de code

Après la préparation d'environnement de travail de Une fois le serveur SonarQube lance et le scanner applique sur notre projet, on retrouve des informations plus detaillees sur la qualite de notre projet avec les remarques et conseils de Sonar directement depuis l'interface web.
Ci-après la capture d'écran du tableau de bord de l'analyse du projet :

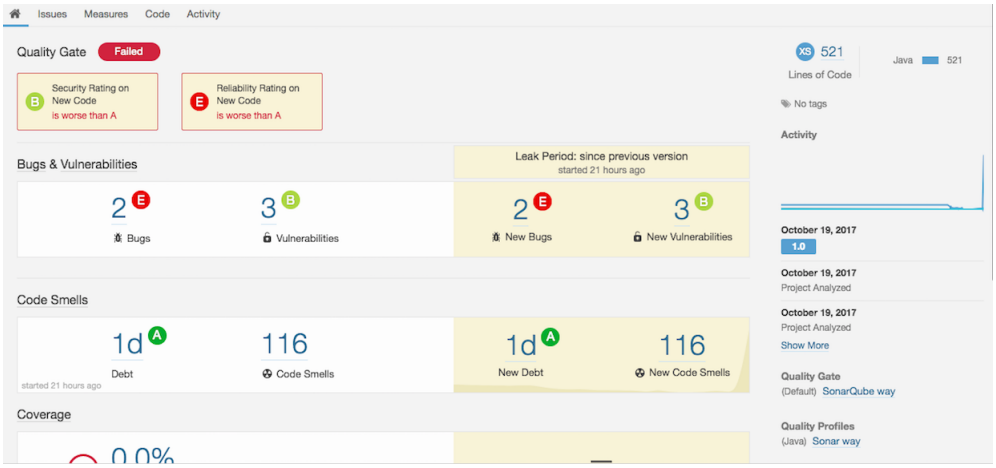
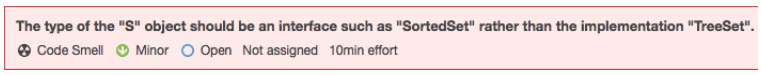


FIGURE 1 – le résultat du première analyse

on catégorise les erreurs à corriger :

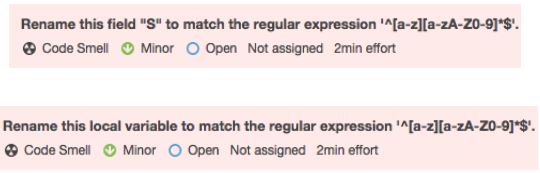
- Erreur de Typage de variable :



Solution Proposée :

```
1     public SortedSet sources(boolean visu) {
2    TreeSet sources = new TreeSet();
3    for (Iterator I = treeSet.iterator(); I.hasNext();) {
4      Node node = (Node) I.next();
5      if (node.pred().isEmpty()) {
6
7        sources.add(node);
8        if (visu)
9          node.setShape("box");
10     }
11   }
12   return sources;
13 }
```

- Erreur par rapport à l'intitulé de la variable :



  Par Exemple : Node N => Node PremierNode

- Erreur de la mauvaise utilisation des fonctions :

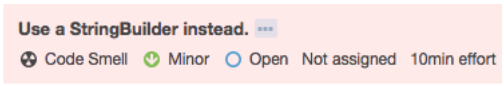
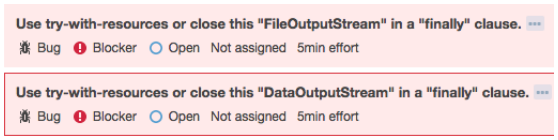


FIGURE 2 – Erreur de la concaténation des chaînes de caractère

Solution Proposée :

```
1  nodes += " " + node.toString();
```

- Erreur d'absence d'exception



Solution Proposée : Mettre le traitement dans un bloc de type (TryTryfinallycatch

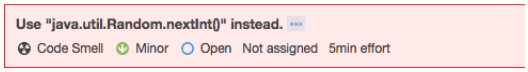
```
1     try{
2                         FileOutputStream fich = new FileOutputStream(
                              filename);
3                         DataOutputStream out = new DataOutputStream(
                              fich);
4                        try{
5
6
7
8     out.writeBytes("digraph G {\n");
9     String nodes= "";
10    String arcs = "";
11    // parcours de l'ensemble de Node
12    for (Iterator I = graph.treeSet.iterator(); I.hasNext();) {
13      Node node = (Node) I.next();
14      nodes = node.toDot();
15      SortedSet succ = node.succ();
16      // parcours de l'ensemble des successeurs de N
```

```
17        for (Iterator J = succ.iterator(); J.hasNext();) {
18          Arc originalArc = (Arc) J.next();
19
20          String arc = originalArc.toString();
21          String str = "";
22          String label = "";
23          if (originalArc.label().length() != 0)
24            label = "label=" + originalArc.label() + ",";
25          String color = "color=" + originalArc.color();
26          str += arc + " [" + label + color + "]\n";
27                                  arcs = new StringBuilder(arcs).append(
                                        " ").append(str).toString();
28        }
29      }
30                          }finally{
31                              fich.close();
32                          }
33              }catch(Exception e){
34                 Logger l=Logger.getLogger("Graphe");
35                   l.log(Level.INFO,e.toString());
36                                  }
37                      }
```

- Erreur 'block de code vide '

Solution proposée : Supprimer le block vide

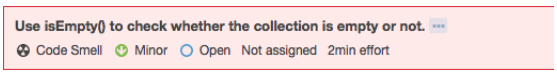
- Erreur de la mauvaise utilisation de la classe



Solution Proposée :

```
1  Random r =new Random();
2  int id =r.nextInt(10*nb);
```

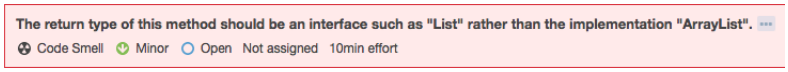
- Erreur de test si la variable est vide :



Solution Proposée :

```
1  if (node.pred().isEmpty()) {
2          sources.add(node);
3          if (visu)
4            node.setShape("box");
5        }
```

- Erreur de type de retour de la méthode



Solution Proposée :

```
1 public List triTopologique(boolean visu)
```

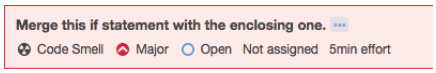
- Erreur de complexité de la méthode :

Solution Proposée : Faire sortir la boucle de la méthode (créer une nouvelle méthode contenant cette boucle) pour diminuer la complexité

```
1  public List triTopologique(boolean visu) {
2    SortedSet sources = this.sources(visu);
3    ArrayList choisis = new ArrayList();
4    Random r= new Random();
5    while (!sources.isEmpty()) {
6      // choix d'un Node dans sources
7      int nb = r.nextInt(sources.size());
8      if (nb == 0)
9        nb++;
10     Node nX = null;
11     for (Iterator I = sources.iterator(); I.hasNext();) {
12       if (nb == 1)
13         nX = (Node) I.next();
14       else
15         I.next();
16       nb--;
17     }
18     sources.remove(nX);
19     choisis.add(nX);
20     // maj des sources par parcours des succ de x
21     if(nX != null) {
22       sourcesBis(nX, choisis, sources, visu);
23     }
24   }
25   return choisis;
26 }
27 public void sourcesBis(Node nX ,List choisis, SortedSet sources ,
       boolean visu) {
28   for (Iterator I = nX.succ().iterator(); I.hasNext();) {
29     Arc aX = (Arc) I.next();
30     Node nV = aX.to();
31     boolean ajouty = true;
32     for (Iterator J = nV.pred().iterator(); J.hasNext();) {
33       Arc vA = (Arc) J.next();
34       Node zN = vA.from();
35
36       if (!choisis.contains(zN))
```

```
37              ajouty = false;
38          }
39          if (ajouty) {
40            sources.add(nV);
41            if (visu)
42              aX.setColor("red");
43          }
44        }
45    }
```

- Erreur d'emboîtement de condition non nécessaire :



Solution Proposée : encapsuler les deux conditions dans une seule condition

```
1       if (dexiemeNode.id()>premierNode.id() && choice > 5) {
2               Arc arc = new Arc(premierNode, dexiemeNode);
3               int il = arc.from().id();
4               int jl = arc.to().id();
5               if (graphe.containsNode(il) && graphe.containsNode(jl)
                    && !graphe.containsArc(il, jl)) {
6                   arc.from().addSucc(arc);
7                   arc.to().addPred(arc);
8             }
9           }
```

le résultat d'analyse de la qualité de notre projet :

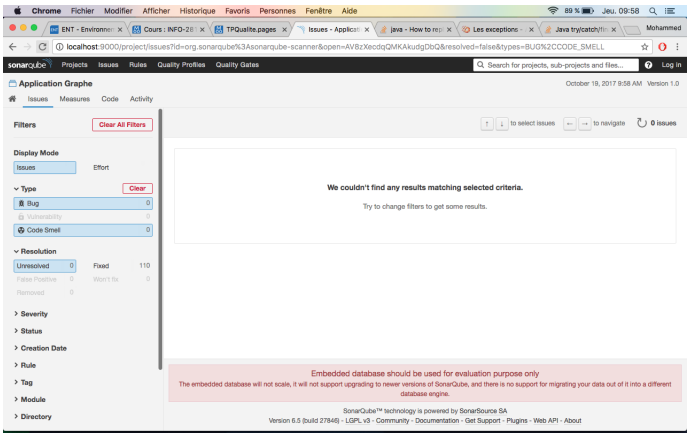


FIGURE 3 – le résultat final analyse

Les détails de l'analyse SonarQube

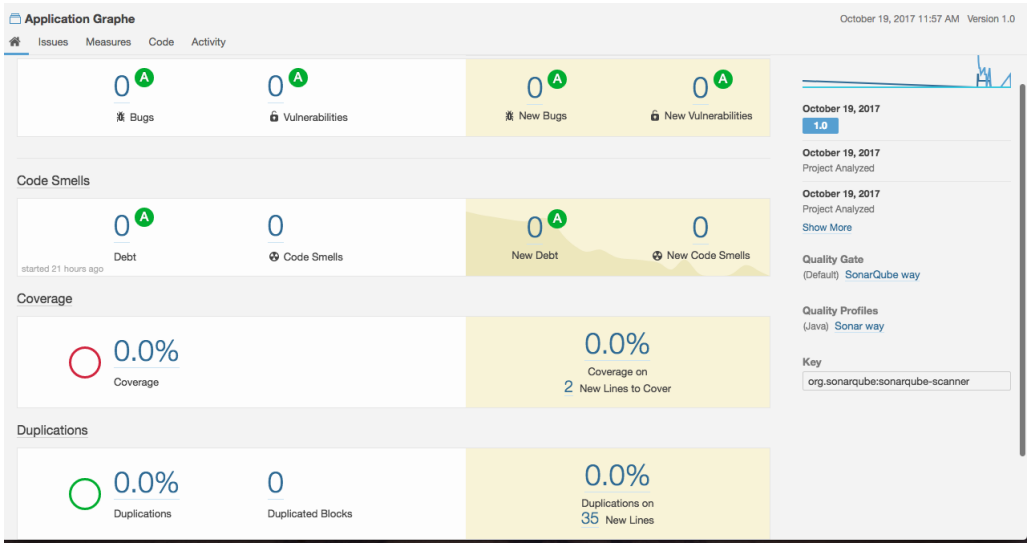


FIGURE 4 – le résultat final analyse

Niveau 3 : Calcul des métriques

1.1 Calcul de l'efferent/afferent coupling pour les classes

La classe	Afferent coupling	Efferent coupling	Instabilité
Arc	2	1	1/3
Graphe	0	2	1
Comparable	2	0	0
somme	0	0	0
Node	2	1	1/3
TPGraphe	0	1	1

TABLE 1 –

1.2 Complexité cyclomatique

Pour calculer la complexité cyclomatique , on calcule le nombre de chemins possible dans le graphe :
Donc c'est le nombre de condition pour chaque méthode plus le cas échéant
Deux conditions sur le main et le cas échéant donc la complexité cyclomatique égale à 3

1.3 Lack Of Cohesion Of Methods

LCOM =1-(le nombre de méthodes de la classe appelant un champs donné)/ le nombre de méthode dans la classe*le nombre de champ d'instance Pour la classe Graphe :
Sum(MF) :27

M :25
F :4
CC= 1-Sum(MF)/M*F = 0.27

2 Code source

Voilà le code du projet après les modifications apportées.

2.1 La classe Somme

```
1  package metrique;
2
3  public class Somme {
4    public int addition(int a,int b){
5      return a+b;
6    }
7  }
```

2.2 La classe Arc

```
1  package metrique;
2  /*****************************************************/
3  import java.util.*;
4  public class Arc implements Comparable {
5      /** Node origine de l'Arc **/
6      private Node from;
7      /** Node origine de l'Arc **/
8      private Node to;
9      /** Arc valu **/
10     private double value;
11     /** attributs dot **/
12     private String color;
13     private String label;
14
15     /** constructeurs **/
16     Arc (Node from, Node to) {
17   this.from = from;
18   this.to = to;
19   this.color = "black";
20   this.label = "";
21   this.value = -1;
22     }
23     /** accesseur du Node origine **/
24     public Node from (){
25   return this.from;
26     }
27     /** accesseur du Node extremite **/
28     public Node to (){
29   return this.to;
30     }
31     // mthodes d'accs aux attributs
```

```
32      public double value (){
33    return this.value;
34      }
35      public void setValue (double v){
36    this.value = v;
37    this.label=Double.toString(v);
38      }
39      /** mthodes d' accs  a un attribut **/
40      public String color (){
41    return this.color;
42      }
43      /** mthodes d' accs  a un attribut **/
44      public void setColor (String c){
45    this.color = c;
46      }
47      /** mthodes d' accs  a un attribut **/
48      public String label (){
49    return this.label;
50      }
51      /** mthodes d' accs  a un attribut **/
52      public void setLabel (String l){
53    this.label = l;
54      }
55      /** methode de comparaison necessaire pour implementer Comparable
56    Permet de rechercher un Arc dans un TreeSet **/
57      public int compareTo (Object compareObj) {
58        Arc compare = (Arc) compareObj;
59    if (this.from.id() == compare.from().id() && this.to.id() == compare.to
          ().id())
60        return 0;
61    if (this.from.id() < compare.from().id() ||
62        this.from.id() == compare.from().id() && this.to.id() < compare.to
              ().id())
63        return -1;
64    return 1;
65      }
66
67      @Override
68      public boolean equals(Object obj)
69      {
70        return super.equals(obj);
71      }
72
73      @Override
74      public int hashCode()
75      {
76        return super.hashCode();
77      }
78
79      // methode d'affichage
80      public String toString () {
81    return this.from.id()+"->"+this.to.id();
82      }
83      /** methode d'affichage grammaire dot **/
```

```
84      public String toDot () {
85    String arc = this.toString();
86    // gestion des attributs dot
87    String str = "";
88    if (this.label().length()!=0)
89          str = "label="+this.label()+",";
90    String couleur="color="+this.color();
91    return arc+" ["+str+couleur+"]\n";
92      }
93
94      /** methode d'affichage grammaire dot **/
95      public String toDot2 () {
96    String arc = this.toString();
97    // gestion des attributs dot
98    String str = "";
99    if (this.label().length()!=0)
100         str = "label="+this.label()+",";
101   String couleur="color="+this.color();
102   return arc+" ["+str+couleur+"]\n";
103     }
104 }// fin d'Arc
```

2.3 La classe Node

```
1  package metrique;
2  /*****************************************************/
3  import java.util.*;
4  /** Classe pour manipuler un noeud d'un graphe **/
5  public class Node implements Comparable {
6      /** identifiant du Node **/
7      private int id;
8      /** listes de successeurs et de predecesseurs **/
9      private TreeSet succ;
10     private TreeSet pred;
11     /** attributs dot **/
12     private String label;
13     private String color;
14     private String shape;
15     /** permet de compter le nb d'appels a compareTo **/
16     public static final int NBCOMPARETO = 0;
17
18     /** constructeur **/
19     Node (int i) {
20   this.id = i;
21   this.succ = new TreeSet();
22   this.pred = new TreeSet();
23   this.label=Integer.toString(i);
24   this.color="black";
25   this.shape="ellipse";
26     }
27     /** constructeur **/
28     Node (int i, String label) {
29   this.id = i;
30   this.succ = new TreeSet();
```

```
31      this.pred = new TreeSet();
32      this.label=label;
33      this.color="black";
34      this.shape="ellipse";
35        }
36        /** accs    l'identifiant **/
37        public int id (){
38      return this.id;
39        }
40        /** accs    l'ensemble des successeurs **/
41        public SortedSet succ (){
42          return this.succ;
43        }
44        /** accs    l'ensemble des predecesseurs **/
45        public SortedSet pred (){
46          return this.pred;
47        }
48        /** mthodes d'accs    un attribut **/
49        public String color (){
50      return this.color;
51        }
52        /** mthodes d'accs    un attribut **/
53        public void setColor (String c){
54      this.color = c; // possibilit de tester la validit de c
55        }
56        /** mthodes d'accs    un attribut **/
57        public String shape (){
58      return this.shape;
59        }
60        /** mthodes d'accs    un attribut **/
61        public void setShape (String s){
62      this.shape = s; // possibilit de tester la validit de c
63        }
64        /** mthodes d'accs    un attribut **/
65        public String label (){
66      return this.label;
67        }
68        /** mthodes d'accs    un attribut **/
69        public void setLabel (String l){
70      this.label = l;
71        }
72        /** ajout d'un successeur **/
73        public boolean addSucc (Arc a ) {
74      return this.succ.add(a);
75        }
76        /** test de l'existence d'un successeur **/
77        public boolean containsSucc (Arc a) {
78      return this.succ.contains(a);
79        }
80        /** suppression d'un successeur **/
81        public boolean removeSucc (Arc a) {
82      return this.succ.remove(a);
83        }
84        /** ajout d'un predecesseur **/
```

```
85      public boolean addPred (Arc a) {
86    return this.pred.add(a);
87      }
88      /** test de l'existence d'un predecesseur **/
89      public boolean containsPred (Arc a) {
90    return this.pred.contains(a);
91      }
92      /** suppression d'un predecesseur **/
93      public boolean removePred (Arc a) {
94    return this.pred.remove(a);
95      }
96
97      /** methode de comparaison necessaire pour implementer Comparable
98    permet de rechercher un Node dans un TreeSet **/
99      @Override
100     public int compareTo (Object o) {
101     Node n = (Node) o ; // C'est Node qui est entre
102   if (this.id > n.id())
103       return 1;
104   if (this.id < n.id())
105       return -1;
106   return 0;
107     }
108
109     /** methode equal **/
110     @Override
111     public boolean equals(Object o) {
112       return super.equals(o);
113     }
114
115     /** methode hashCode **/
116     @Override
117     public int hashCode() {
118       return super.hashCode();
119     }
120
121
122     /** methode d'affichage **/
123     public String toString () {
124   return this.id+" ";
125     }
126     /** methode d'affichage grammaire dot **/
127     public String toDot () {
128   return this.id+" [label="+this.label+",color="+this.color+",shape="+
        this.shape+"]\n";
129     }
130
131
132 }// fin de Node
```

2.4 La classe Graphe

```
1  /*
2   * To change this license header, choose License Headers in Project
```

```
    Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
package metrique;

/**
 *
 * @author Mohammed
 */
/*****************************************************/

import java.util.*;
import java.util.logging.Level;
import java.util.logging.Logger;
import java.io.*;

/**
 * Classe Graphe permettant de manipuler des graphes. Reprsentation par
    listes
 * de successeurs
 **/
public class Graphe {
  /** ensemble de Node (ou NodeSet) **/
  private TreeSet treeSet;
  public static final String BLACK = "black";

  /** constructeur *a*/
  Graphe() {
    this.treeSet = new TreeSet();
  }

  /** accs    l'ensemble de Node **/
  public SortedSet getS() {
    return this.treeSet;
  }

  /** ajout d'un Node dans le graphe **/
  public boolean addNode(Node node) {
    return this.treeSet.add(node);
  }

  /** test de l'existence d'un Node dans le graphe **/
  public boolean containsNode(int i) {
    return this.treeSet.contains(new Node(i));
  }

  /** suppression d'un Node dans le graphe **/
  public boolean removeNode(Node node) {
    if (!this.treeSet.remove(node))
      return false;
    for (Iterator I = node.succ().iterator(); I.hasNext();) {
      Arc arc = (Arc) I.next();
      Node dexiemeNode = arc.to();
```

```
55        dexiemeNode.removePred(arc);
56      }
57      for (Iterator I = node.pred().iterator(); I.hasNext();) {
58        Arc arc = (Arc) I.next();
59        Node dexiemeNode = arc.from();
60        dexiemeNode.removeSucc(arc);
61      }
62      return true;
63    }
64
65    /** accs    un Node du graphe **/
66    public Node getNode(int i) {
67      if (this.containsNode(i))
68        for (Iterator I = this.treeSet.iterator(); I.hasNext();) {
69          Node node = (Node) I.next();
70          if (node.id() == i)
71            return node;
72        }
73      return null;
74    }
75
76    /** test de l'existence d'un Arc dans le graphe **/
77    public boolean containsArc(int i, int j) {
78      if (this.containsNode(i) && this.containsNode(j)) {
79        Node from = this.getNode(i);
80        Node to = this.getNode(j);
81        return from.containsSucc(new Arc(from, to));
82      }
83      return false;
84    }
85
86    /** accs    un Arc du graphe **/
87    public Arc getArc(int i, int j) {
88      if (this.containsArc(i, j)) {
89        Node from = this.getNode(i);
90        for (Iterator I = from.succ().iterator(); I.hasNext();) {
91          Arc arc = (Arc) I.next();
92          if (arc.from().id() == i && arc.to().id() == j)
93            return arc;
94        }
95      }
96      return null;
97    }
98
99    /** ajout d'un Arc dans le graphe **/
100   public boolean addArc(Arc arc) {
101     int i = arc.from().id();
102     int j = arc.to().id();
103     if (this.containsNode(i) && this.containsNode(j) && !this.containsArc
          (i, j)) {
104       arc.from().addSucc(arc);
105       arc.to().addPred(arc);
106       return true;
107     }
```

```
108       return false;
109     }
110
111     /** suppression d'un Arc dans le graphe **/
112     public boolean removeArc(Arc arc) {
113       int i = arc.from().id();
114       int j = arc.to().id();
115       if (this.containsNode(i) && this.containsNode(j) && this.containsArc(
              i, j)) {
116         arc.from().removeSucc(arc);
117         arc.to().removePred(arc);
118         return true;
119       }
120       return false;
121     }
122
123     public int nbNodes() {
124       return this.treeSet.size();
125     }
126
127     public int nbArcs() {
128       int nb = 0;
129       for (Iterator I = treeSet.iterator(); I.hasNext();) {
130
131         Node node = (Node) I.next();
132         nb += node.succ().size();
133       }
134       return nb;
135     }
136
137     /* colorier tous les noeuds et arcs d'une meme couleur */
138     public void setColor(String color) {
139       for (Iterator I = treeSet.iterator(); I.hasNext();) {
140         Node node = (Node) I.next();
141         node.setColor(color);
142         for (Iterator J = node.succ().iterator(); J.hasNext();) {
143           Arc arc = (Arc) J.next();
144           arc.setColor(color);
145         }
146       }
147     }
148
149     /** methode d'affichage **/
150     public String toString() {
151       String graph = "";
152       String nodes = "Nodes = {";
153       String arcs = "Arcs = {";
154       graph += "G --> " + this.nbNodes() + "Nodes \n";
155       graph += "      " + this.nbArcs() + "Arcs \n ";
156       for (Iterator I = treeSet.iterator(); I.hasNext();) {
157         Node node = (Node) I.next();
158         nodes += " " + node.toString();
159         SortedSet succ = node.succ();
160         // Parcours de l'ensemble des successeurs
```

```
161       for (Iterator J = succ.iterator(); J.hasNext();) {
162         Arc arc = (Arc) J.next();
163         arcs += " " + arc.toString();
164       }
165
166     }
167     return graph + nodes + "}\n" + arcs + "}\n";
168
169   }
170
171   /** methode d'affichage grammaire dot **/
172   public static void toDot(Graphe graph, String filename) throws
          IOException {
173         try{
174                         FileOutputStream fich = new FileOutputStream(
                            filename);
175                         DataOutputStream out = new DataOutputStream(fich);
176                        try{
177
178
179
180     out.writeBytes("digraph G {\n");
181     String nodes= "";
182     String arcs = "";
183     // parcours de l'ensemble de Node
184     for (Iterator I = graph.treeSet.iterator(); I.hasNext();) {
185       Node node = (Node) I.next();
186       nodes = node.toDot();
187       SortedSet succ = node.succ();
188       // parcours de l'ensemble des successeurs de N
189       for (Iterator J = succ.iterator(); J.hasNext();) {
190         Arc originalArc = (Arc) J.next();
191
192         String arc = originalArc.toString();
193         String str = "";
194         String label = "";
195         if (originalArc.label().length() != 0)
196           label = "label=" + originalArc.label() + ",";
197         String color = "color=" + originalArc.color();
198         str += arc + " [" + label + color + "]\n";
199                                 arcs = new StringBuilder(arcs).append(" ").
                                    append(str).toString();
200       }
201     }
202                         }finally{
203                             fich.close();
204                         }
205
206             }catch(Exception e){
207                Logger l=Logger.getLogger("Graphe");
208                  l.log(Level.INFO,e.toString());
209
210                                 }
211                   }
```

```
212
213
214   public static Graphe divGraph1(int nb, boolean visu) {
215     Graphe graphe = new Graphe();
216     for (int i = 2; i <= nb; i++) {
217       Node node = new Node(i);
218       if (i % 2 == 0 && visu) {
219         node.setColor("red");
220       }
221       graphe.addNode(node);
222     }
223     // ajout des arcs
224     for (int i = 2; i <= nb; i++)
225
226       for (int j = 2; j <= nb; j++)
227
228         if (i % j == 0) {
229           Node premierNode = graphe.getNode(i);
230           Node dexiemeNode = graphe.getNode(j);
231           Arc arc = new Arc(dexiemeNode, premierNode);
232           if (visu) {
233             int div = i / j;
234             String label =Integer.toString(div);
235             arc.setLabel(label);
236           }
237           graphe.addArc(arc);
238         }
239
240     return graphe;
241   }
242
243   // ---------
244   /* parcours en largeur */
245   public void parcoursLargeur(boolean visu) {
246
247     // couleur noire (inexplore) pour tous les noeuds
248     for (Iterator I = treeSet.iterator(); I.hasNext();) {
249       Node node = (Node) I.next();
250       node.setColor(BLACK);
251     }
252     // parcours  partir d'une source inexplore
253     for (Iterator I = this.treeSet.iterator(); I.hasNext();) {
254       Node node = (Node) I.next();
255       if (node.color().equals(BLACK))
256         parcoursLargeur(node,visu);
257     }
258     // remet les sommets en noir
259     for (Iterator I = treeSet.iterator(); I.hasNext();) {
260       Node node = (Node) I.next();
261       node.setColor(BLACK);
262     }
263   }
264
265   /* parcours en largeur  partir d'une source */
```

```
266   private void parcoursLargeur(Node nodeLargeur, boolean visu) {
267     nodeLargeur.setColor("blue");
268     ArrayList tableauNode = new ArrayList();
269     tableauNode.add(nodeLargeur);
270     while (!tableauNode.isEmpty()) {
271       Node node = (Node) tableauNode.get(0);
272       for (Iterator I = node.succ().iterator(); I.hasNext();) {
273         Arc arc = (Arc) I.next();
274         Node dexiemeNode = arc.to();
275         if (dexiemeNode.color().equals(BLACK) && !tableauNode.contains(
              dexiemeNode) ) {
276           tableauNode.add(dexiemeNode);
277           if (visu)
278             arc.setColor("blue");
279         }
280       }
281       tableauNode.remove(node);
282       node.setColor("blue");
283       }
284
285     }
286
287
288   /* parcours en profondeur */
289   public void parcoursProfondeur(boolean visu) {
290     // couleur noire (inexplore) pour tous les noeuds
291     for (Iterator I = treeSet.iterator(); I.hasNext();) {
292       Node node = (Node) I.next();
293       node.setColor(BLACK);
294     }
295     // parcours   partir d'une source inexplore
296     for (Iterator I = this.treeSet.iterator(); I.hasNext();) {
297       Node node = (Node) I.next();
298       if (node.color().equals(BLACK))
299         parcoursProfondeur(node, visu);
300     }
301   }
302
303   /* parcours en profondeur rcursif   partir d'une source */
304   private void parcoursProfondeur(Node initialNode, boolean visu) {
305     initialNode.setColor("red");
306     for (Iterator I = initialNode.succ().iterator(); I.hasNext();) {
307       Arc arc = (Arc) I.next();
308       Node node =arc.to();
309       if (!node.color().equals("red")) {
310         if (visu)
311           arc.setColor("green");
312         parcoursProfondeur(node, visu);
313       }
314     }
315   }
316
317   /*************************************************************/
318   /** mthode statique de gnration d'un graphe alatoire **/
```

```
319   public static Graphe randomGraphe(int nb) {
320     Graphe graphe = new Graphe();
321     Random r =new Random();
322     // ensemble S de Node
323     int i = nb;
324     while (i > 0) {
325       int id =r.nextInt(10*nb);
326       if (graphe.addNode(new Node(id)))
327         i--;
328     }
329     // ensemble A d'Arcs
330     for (Iterator I = graphe.getS().iterator(); I.hasNext();) {
331       Node premierNode = (Node) I.next();
332       for (Iterator J = graphe.getS().iterator(); J.hasNext();) {
333         Node dexiemeNode = (Node) J.next();
334         int choice =r.nextInt(10);
335         /*
336          * choice compris entre 0 et 10. si choice < 5: on n'ajoute pas
337          * l'arc (N1,N2) si choice > 5: on ajoute l'arc
338          */
339         if (choice > 5 && premierNode != dexiemeNode)
340           graphe.addArc(new Arc(premierNode, dexiemeNode));
341       }
342     }
343     return graphe;
344   }
345
346
347
348   /** mthode statique de gnration d'un DAG alatoire **/
349   public static Graphe randomDAG(int nb) {
350     Graphe graphe = new Graphe();
351     Random r =new Random();
352     // ensemble S de Node
353     int i = nb;
354     while (i > 0) {
355       int id = r.nextInt(10 * nb);
356       if (graphe.addNode(new Node(id, "n" + id)))
357         i--;
358     }
359     // ensemble A d'Arcs
360     for (Iterator I = graphe.getS().iterator(); I.hasNext();) {
361       Node premierNode = (Node) I.next();
362       for (Iterator J = graphe.getS().iterator(); J.hasNext();) {
363         Node dexiemeNode = (Node) J.next();
364         int choice = r.nextInt(10);
365         if (dexiemeNode.id()>premierNode.id() && choice > 5) {
366             Arc arc = new Arc(premierNode, dexiemeNode);
367             int il = arc.from().id();
368             int jl = arc.to().id();
369             if (graphe.containsNode(il) && graphe.containsNode(jl) && !
                    graphe.containsArc(il, jl)) {
370               arc.from().addSucc(arc);
371               arc.to().addPred(arc);
```

```
372
373
374           }
375         }
376       }
377     }
378     return graphe;
379   }
380
381
382   /************************/
383   /** calcul des sources **/
384   public SortedSet sources(boolean visu) {
385     TreeSet sources = new TreeSet();
386     for (Iterator I = treeSet.iterator(); I.hasNext();) {
387       Node node = (Node) I.next();
388       if (node.pred().isEmpty()) {
389
390         sources.add(node);
391         if (visu)
392           node.setShape("box");
393       }
394     }
395     return sources;
396   }
397
398   /** calcul d'un tri topologique **/
399   public List triTopologique(boolean visu) {
400     SortedSet sources = this.sources(visu);
401     ArrayList choisis = new ArrayList();
402     Random r= new Random();
403     while (!sources.isEmpty()) {
404       // choix d'un Node dans sources
405       int nb = r.nextInt(sources.size());
406       if (nb == 0)
407         nb++;
408       Node nX = null;
409       for (Iterator I = sources.iterator(); I.hasNext();) {
410         if (nb == 1)
411           nX = (Node) I.next();
412         else
413           I.next();
414         nb--;
415       }
416       sources.remove(nX);
417       choisis.add(nX);
418       // maj des sources par parcours des succ de x
419       if(nX != null) {
420         sourcesBis(nX, choisis, sources, visu);
421       }
422
423     }
424     return choisis;
425   }
```

```
426   public void sourcesBis(Node nX ,List choisis, SortedSet sources ,
          boolean visu) {
427     for (Iterator I = nX.succ().iterator(); I.hasNext();) {
428       Arc aX = (Arc) I.next();
429       Node nV = aX.to();
430       boolean ajouty = true;
431       for (Iterator J = nV.pred().iterator(); J.hasNext();) {
432         Arc vA = (Arc) J.next();
433         Node zN = vA.from();
434
435         if (!choisis.contains(zN))
436           ajouty = false;
437
438       }
439       if (ajouty) {
440         sources.add(nV);
441         if (visu)
442           aX.setColor("red");
443       }
444     }
445
446   }
447
448   // test de l'existence d'un cycle
449   public boolean cycles(boolean visu) {
450     List tri = this.triTopologique(false);
451     if (tri.size() != this.nbNodes()) {
452       if (visu) {
453         for (Iterator I = this.getS().iterator(); I.hasNext();) {
454           Node node = (Node) I.next();
455           if (!tri.contains(node))
456             node.setColor("blue");
457         }
458       }
459       return true;
460     } else
461       return false;
462   }
463
464 }// fin de Graph
```

2.5 La classe TPGraphe

```
1  /*
2   * To change this license header, choose License Headers in Project
        Properties.
3   * To change this template file, choose Tools | Templates
4   * and open the template in the editor.
5   */
6  package metrique;
7  /**
8   *
9   * @author Mohammed
10  */
```

```
11 import java.io.*;
12 import java.util.*;
13 import java.util.logging.Level;
14 import java.util.logging.Logger;
15
16 public class TPGraphe {
17
18   public static void main (String[] arg) throws IOException {
19     // graphe des diviseurs
20     Graphe graph1 = Graphe.randomDAG(10);
21                 Logger l=Logger.getLogger("TPGraphe");
22
23                 if(l.isLoggable(Level.INFO)){
24                 l.info(graph1.toString());
25                 }
26     Graphe.toDot(graph1,"prof.dot");
27     graph1.parcoursProfondeur(true);
28     Graphe.toDot(graph1,"profDiv.dot");
29     graph1.setColor("black");
30     List list = graph1.triTopologique(true);
31                 if(l.isLoggable(Level.INFO)){
32                 l.info(list.toString());
33                 }
34     Graphe.toDot(graph1,"profDiv.dot");
35
36
37
38   }
39 }
```