CSC4251_4252 : Cours pratique — Patron de conception Visiteur

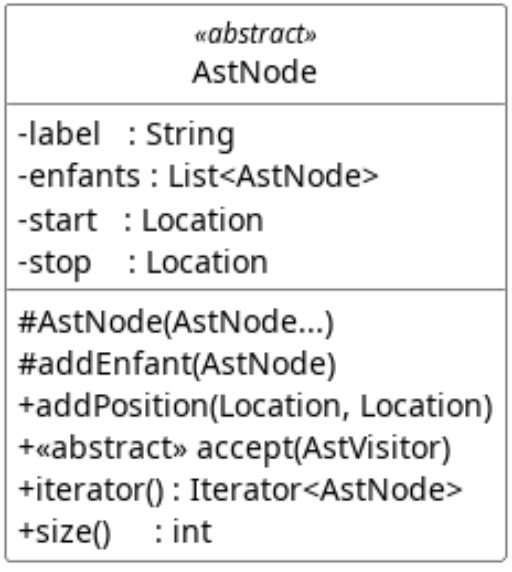
Denis Conan et J. Paul Gibson

Décembre 2024

Sommaire

1 Rappel JAVA

1.1 Classe abstraite



- Classe abstraite `AstNode` qui ne peut pas être instanciée
  - Raison 1 : on ne veut pas d'instance
  - Raison 2 : des méthodes ne sont pas définies
- Spécialisation nécessaire jusqu'à obtenir des classes instanciables
  - P.ex. la classe `ExpOpBin`

$\Longrightarrow$ La classe abstraite impose un comportement commun à toutes les classes enfants

- Possibilité de créer des tableaux ou des collections de références de type de la classe abstraite

1.1.1 Un peu de code de la classe `AstNode`

```
1  public abstract class AstNode implements Iterable<AstNode> {
2          private final String label;
3          private final java.util.List<AstNode> fils;
4          private Location start;
5          private Location stop;
6          protected AstNode(final AstNode... fils) {
7                  this.label = getClass().getSimpleName();
8                  this.fils = new java.util.ArrayList<>();
9                  for (AstNode f : fils)
10                         this.fils.add(f);
11         }
12         protected void addFils(final AstNode f) {
13                 this.fils.add(f);
14         }
15         public java.util.Iterator<AstNode> iterator() {
16                 return fils.iterator();
17         }
```

- `Iterable<AstNode>` : l'implémentation de cette interface permet à un objet `AstNode` d'être la cible d'une instruction `for-each`
  - *n* de type « `AstNode` », écrire « `for(AstNode n: n){...}` »
- `AstNode...` : écriture d'une liste d'arguments du même type sous la forme dite « varargs » ; le résultat est une `List<AstNode>`

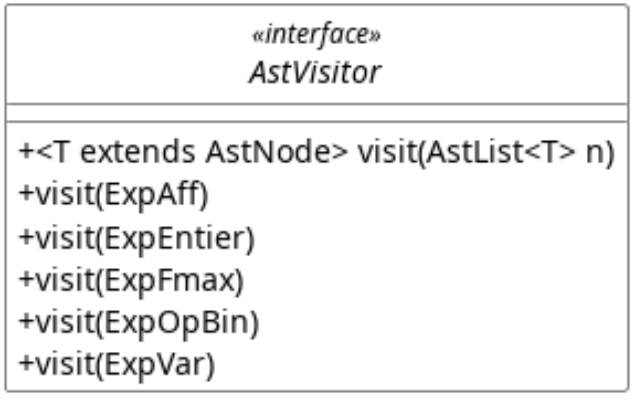
1.2 Classe paramétrée

- Exemple de concrétisation de la classe abstraite `AstNode` avec `AstList`
- `AstList` = nœud de l'AST avec fils homogènes
  - Cette contrainte n'est pas vérifiée quand on considère `AstNode` comme une collection
    - Par défaut, les enfants d'un `AstNode` sont de types différents

```
1 public class AstList<R extends AstNode> extends AstNode {
2         public AstList() {
3                 /* liste vide */ }
4         public void add(final R node) {
5                 this.addFils(node);
6         }
```

- Dans `AstList<R extends AstNode>`, « `R extends AstNode` » est un paramètre de type
  - `R` est le nom du paramètre de type : cf. méthode add
  - `AstList` est une liste d'objets de type `AstNode`
    - « `extends` » contraint le type du paramètre aux classes enfants de `AstNode`

1.3 Interface



- Les membres d'une interface sont principalement des méthodes abstraites
- Une interface est déclarée avec le mot réservé `interface`
- Une interface peut hériter (`extends`) de plusieurs interfaces
- Une classe peut implémenter (`implements`) une ou plusieurs interfaces
  - Rappel : une classe n'hérite (`extends`) que d'une seule classe

1.3.1 Un peu de code de l'interface `AstVisitor`

```
1 public interface AstVisitor {
2         <T extends AstNode> void visit(AstList<T> n);
3         void visit(ExpAff n);
4         void visit(ExpEntier n);
5         void visit(ExpFor n);
6         void visit(ExpOpBin n);
```

- Pour les visites, utilisation de la surcharge
  - Pour tous les types de nœud t de l'AST, c.-à-d. pour toutes les classes enfants de `AstNode`, une méthode « `visit(t)` »

1.4 Réalisation et surcharge de méthode

```
1  public class AstVisitorDefault implements AstVisitor {
2          public void defaultVisit(final AstNode node) {
3                  for (AstNode f : node)
4                          f.accept(this);
5          }
6          public <T extends AstNode> void visit(final AstList<T> n) {
7                  defaultVisit(n);
8          }
9          public void visit(final ExpAff n) {
10                 defaultVisit(n);
11         }
12         public void visit(final ExpFor n) {
13                 defaultVisit(n);
14         }
15         public void visit(final ExpEntier n) {
16                 defaultVisit(n);
17         }
18         public void visit(final ExpOpBin n) {
19                 defaultVisit(n);
20         }
```

- Visiteur complet qui propose une visite par défaut, qui parcourt les enfants

1.5 Redéfinition et liaison tardive

```
1  public class Gratos extends AstVisitorDefault {
2          private int nbOp;
3          public Gratos(AstNode n) {
4                  nbOp = 0;
5                  n.accept(this);
6                  System.out.println("Nombre␣d'op.␣binaire␣=␣" + nbOp);
7          }
8          @Override
9          public void visit(ExpOpBin n) {
10                 nbOp++;
11                 defaultVisit(n);
12         }
13 }
```

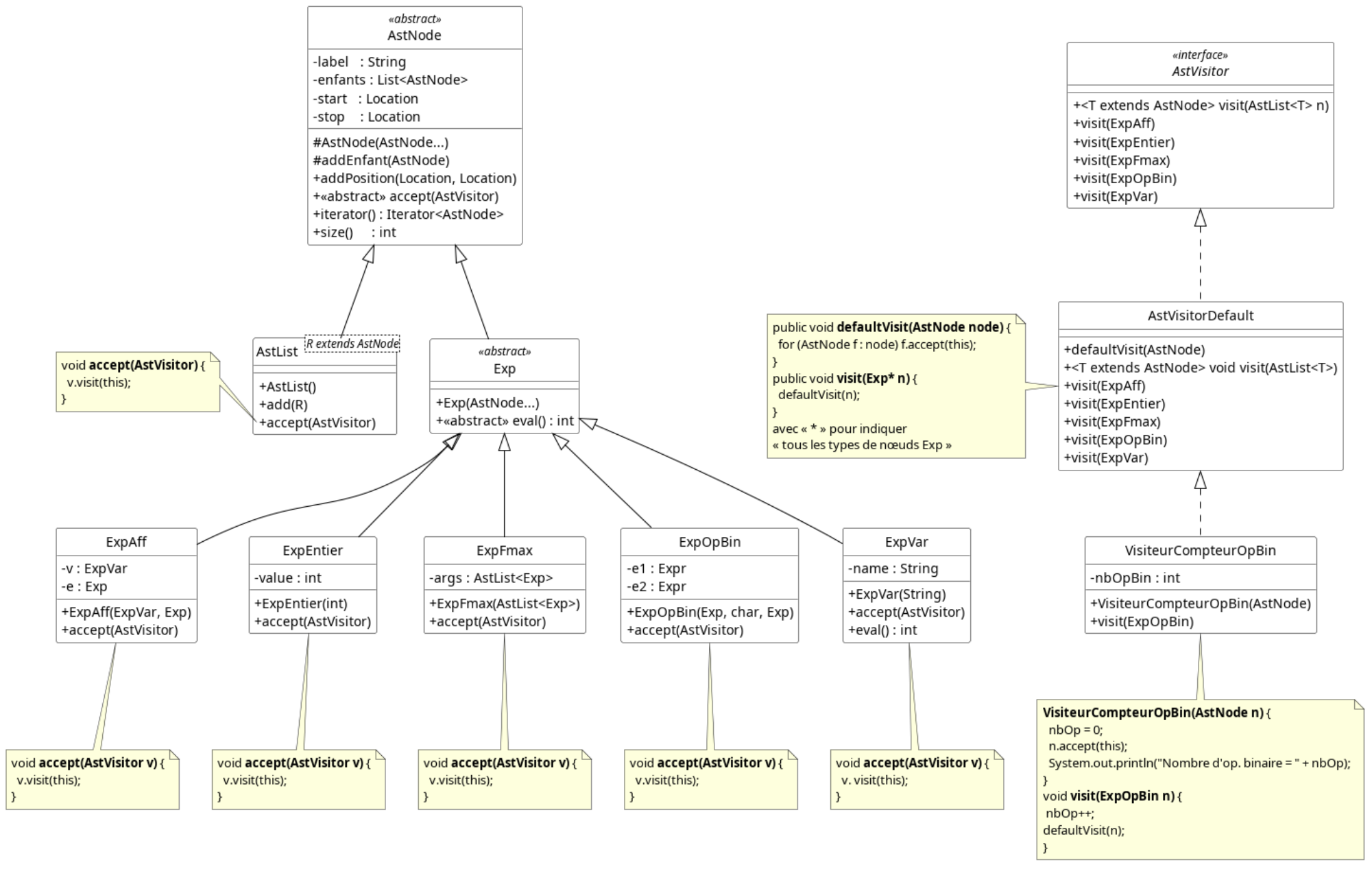
- Spécialisation de la visite par défaut (`AstVisitorDefault`) dans `VisiteurCompteurOpBin`
- Ce visiteur, pour les nœuds de type `ExpOpBin`, redéfinit la méthode `visit` afin de ne pas faire la visite par défaut de `AstVisitorDefault`

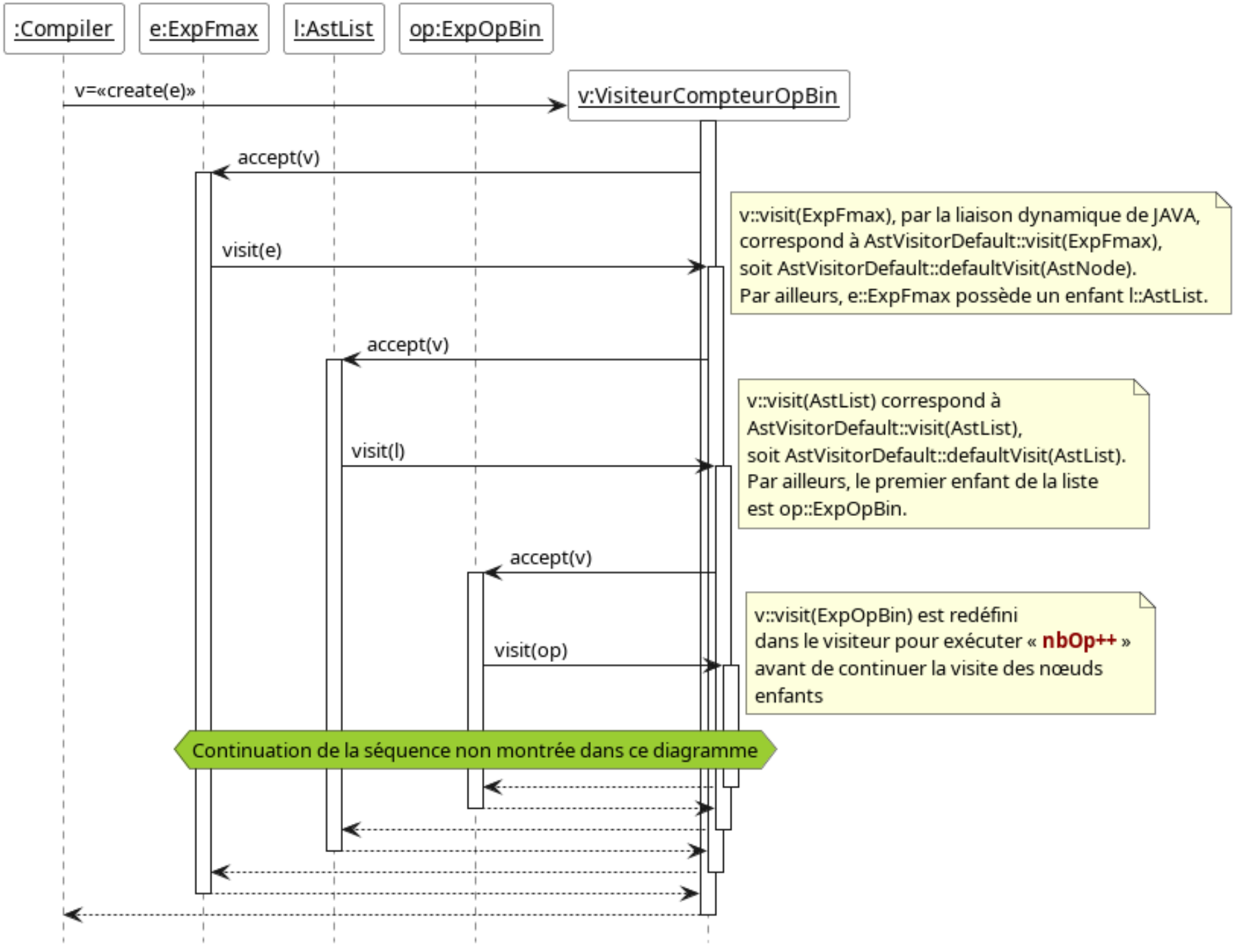
2 Patron de conception Visiteur

En utilisant l'exemple du visiteur de l'exercice CUP 9b

et avec l'entrée « `max((2*21+1)/4, a=max(42, 2), b=c=4*a/42)\n` »

. Patron de conception = une recette (non triviale) très connue qui possède un nom, donc une description à respecter


«abstract»
AstNode
-label : String
-enfants : List<AstNode>
-start : Location
-stop : Location
#AstNode(AstNode...)
#addEnfant(AstNode)
+addPosition(Location, Location)
+«abstract» accept(AstVisitor)
+iterator() : Iterator<AstNode>
+size() : int
«interface»
AstVisitor
+<T extends AstNode> visit(AstList<T> n)
+visit(ExpAff)
+visit(ExpEntier)
+visit(ExpFmax)
+visit(ExpOpBin)
+visit(ExpVar)
AstList
R extends AstNode
+AstList()
+add(R)
+accept(AstVisitor)
void accept(AstVisitor) {
v.visit(this);
}
«abstract»
Exp
+Exp(AstNode...)
+«abstract» eval() : int
public void defaultVisit(AstNode node) {
for (AstNode f : node) f.accept(this);
}
public void visit(Exp* n) {
defaultVisit(n);
}
avec « * » pour indiquer
« tous les types de nœuds Exp »
AstVisitorDefault
+defaultVisit(AstNode)
+<T extends AstNode> void visit(AstList<T>)
+visit(ExpAff)
+visit(ExpEntier)
+visit(ExpFmax)
+visit(ExpOpBin)
+visit(ExpVar)
ExpAff
-v : ExpVar
-e : Exp
+ExpAff(ExpVar, Exp)
+accept(AstVisitor)
ExpEntier
-value : int
+ExpEntier(int)
+accept(AstVisitor)
ExpFmax
-args : AstList<Exp>
+ExpFmax(AstList<Exp>)
+accept(AstVisitor)
ExpOpBin
-e1 : Expr
-e2 : Expr
+ExpOpBin(Exp, char, Exp)
+accept(AstVisitor)
ExpVar
-name : String
+ExpVar(String)
+accept(AstVisitor)
+eval() : int
VisiteurCompteurOpBin
-nbOpBin : int
+VisiteurCompteurOpBin(AstNode)
+visit(ExpOpBin)
void accept(AstVisitor v) {
v.visit(this);
}
void accept(AstVisitor v) {
v.visit(this);
}
void accept(AstVisitor v) {
v.visit(this);
}
void accept(AstVisitor v) {
v.visit(this);
}
void accept(AstVisitor v) {
v. visit(this);
}
VisiteurCompteurOpBin(AstNode n) {
nbOp = 0;
n.accept(this);
System.out.println("Nombre d'op. binaire = " + nbOp);
}
void visit(ExpOpBin n) {
nbOp++;
defaultVisit(n);
}


:Compiler
e:ExpFmax
l:AstList
op:ExpOpBin
v=«create(e)»
v:VisiteurCompteurOpBin
accept(v)
visit(e)
v::visit(ExpFmax), par la liaison dynamique de JAVA,
correspond à AstVisitorDefault::visit(ExpFmax),
soit AstVisitorDefault::defaultVisit(AstNode).
Par ailleurs, e::ExpFmax possède un enfant l::AstList.
accept(v)
visit(l)
v::visit(AstList) correspond à
AstVisitorDefault::visit(AstList),
soit AstVisitorDefault::defaultVisit(AstList).
Par ailleurs, le premier enfant de la liste
est op::ExpOpBin.
accept(v)
visit(op)
v::visit(ExpOpBin) est redéfini
dans le visiteur pour exécuter « nbOp++ »
avant de continuer la visite des nœuds
enfants
Continuation de la séquence non montrée dans ce diagramme

2.3 Visiteur `VisiteurCompteurOpBin`

```
ExpFmax[1/1(1)-1/41(41)]
\-AstList[1/5(5)-1/40(40)]
  |-ExpOpBin[1/5(5)-1/15(15)](/)
  | |-ExpOpBin[1/5(5)-1/13(13)](+)
  | | |-ExpOpBin[1/6(6)-1/10(10)](*)
  | | | |-ExpEntier[1/6(6)-1/7(7)](2)
  | | | \-ExpEntier[1/8(8)-1/10(10)](21)
  | | \-ExpEntier[1/11(11)-1/12(12)](1)
  | \-ExpEntier[1/14(14)-1/15(15)](4)
  |-ExpAff[1/16(16)-1/28(28)]
  | |-ExpVar(a)
  | \-ExpFmax[1/18(18)-1/28(28)]
  |   \-AstList[1/22(22)-1/27(27)]
  |     |-ExpEntier[1/22(22)-1/24(24)](42)
  |     \-ExpEntier[1/26(26)-1/27(27)](2)
  \-ExpAff[1/30(30)-1/40(40)]
    |-ExpVar(b)
    \-ExpAff[1/32(32)-1/40(40)]
      |-ExpVar(c)
      \-ExpOpBin[1/34(34)-1/40(40)](/)
        |-ExpOpBin[1/34(34)-1/37(37)](*)
        | |-ExpEntier[1/34(34)-1/35(35)](4)
        | \-ExpVar[1/36(36)-1/37(37)](a)
        \-ExpEntier[1/38(38)-1/40(40)](42)
```

- Exécution du visiteur avec la chaîne de caractères

  «

  ```
  max((2*21+1)/4,
      a=max(42, 2),
      b=c=4*a/42)\n
  ```

  »

- On repère :
  - Nœud racine `ExpFmax`
  - Nœud enfant `AstList`
  - Petits enfants :
    - Un `ExpOpBin`
    - Deux `ExpAff`
  - etc.