Computer Science
Engineering School

Software
Engineering

Lab 07
L-Value Decoration
(Visitor Design Pattern)

Francisco Ortín Soler

University of Oviedo

Objective

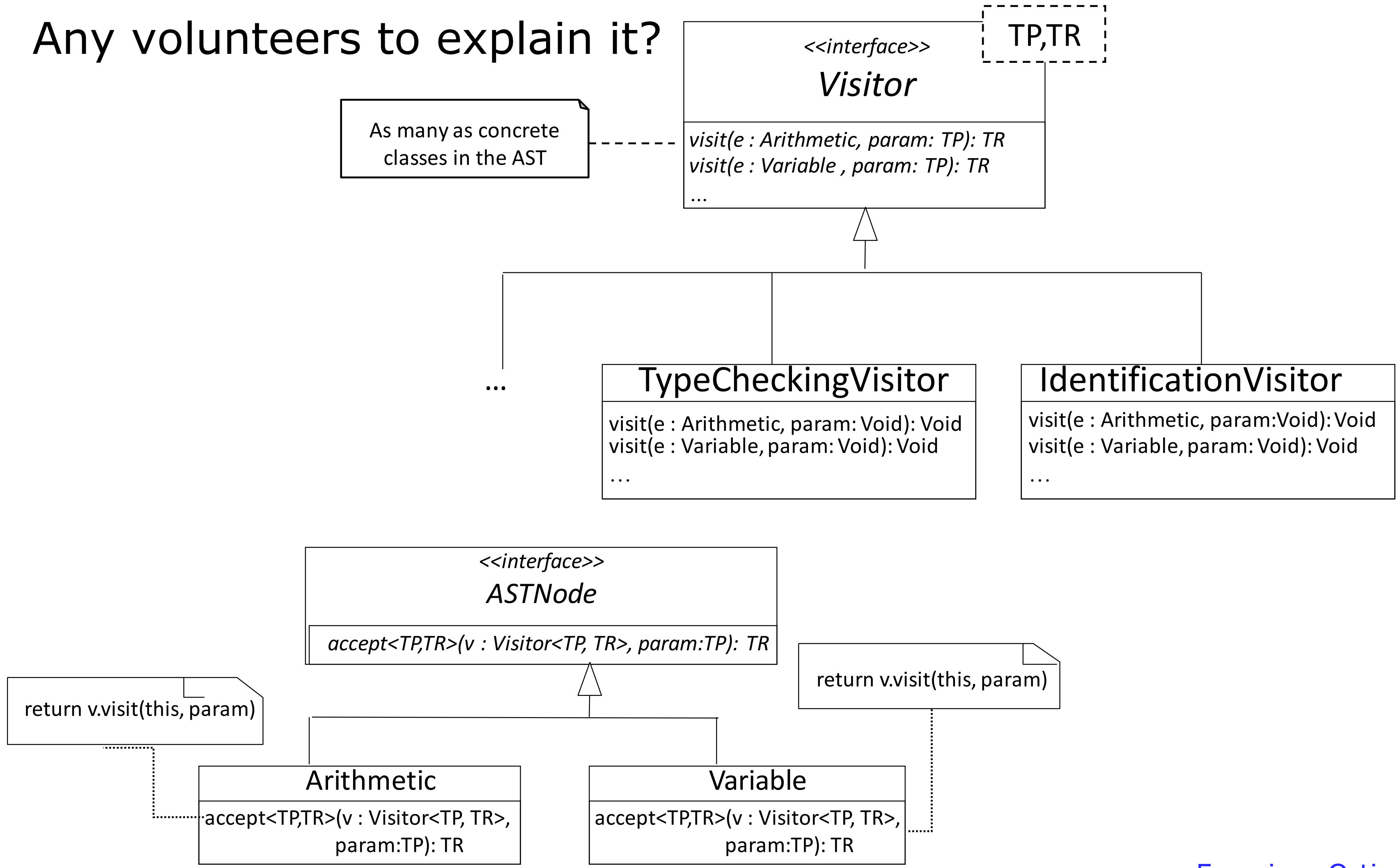
- Decorate all the expressions in the AST with a new l-value attribute
  - Implement it with the Visitor design pattern

Problem

- We will traverse the AST for
  - Identifying the variable definitions
  - Type checking
  - Computing offsets of variables
  - Generating code for statements and definitions
  - Generating values of expressions
  - Generating addresses of expressions
- We do not want to modify the AST any time we want to add a new traversal

Visitor design pattern

- Remember it from previous lectures
- Any volunteers to explain it?


TP,TR
<<interface>>
Visitor
visit(e : Arithmetic, param: TP): TR
visit(e : Variable , param: TP): TR
…
As many as concrete classes in the AST
…
TypeCheckingVisitor
visit(e : Arithmetic, param: Void): Void
visit(e : Variable, param: Void): Void
…
IdentificationVisitor
visit(e : Arithmetic, param:Void): Void
visit(e : Variable, param: Void): Void
…
<<interface>>
ASTNode
accept<TP,TR>(v : Visitor<TP, TR>, param:TP): TR
return v.visit(this, param)
return v.visit(this, param)
Arithmetic
accept<TP,TR>(v : Visitor<TP, TR>, param:TP): TR
Variable
accept<TP,TR>(v : Visitor<TP, TR>, param:TP): TR

Implementation

- Let’s define a **`TypeCheckingVisitor`** class in a **`semantic`** package to annotate the l-value attribute of expressions
  - `getLvalue()` and `setLvalue(boolean lvalue)` must be added to `Expression`
  - `TypeCheckingVisitor` will be extended with type-checking functionality in lab 09
- Question: How do we implement the `visit` methods for?
  - `Program`
  - `Variable`
  - `Assignment`

Autonomous work

1. Implement `TypeCheckingVisitor` using the Visitor design pattern

2. Test it with
   a) `input.txt`: Check with Introspector that **all** the expressions have the correct l-value
   b) `input1-wrong.txt` and `input2-wrong.txt`: all the expected semantic errors are detected and shown