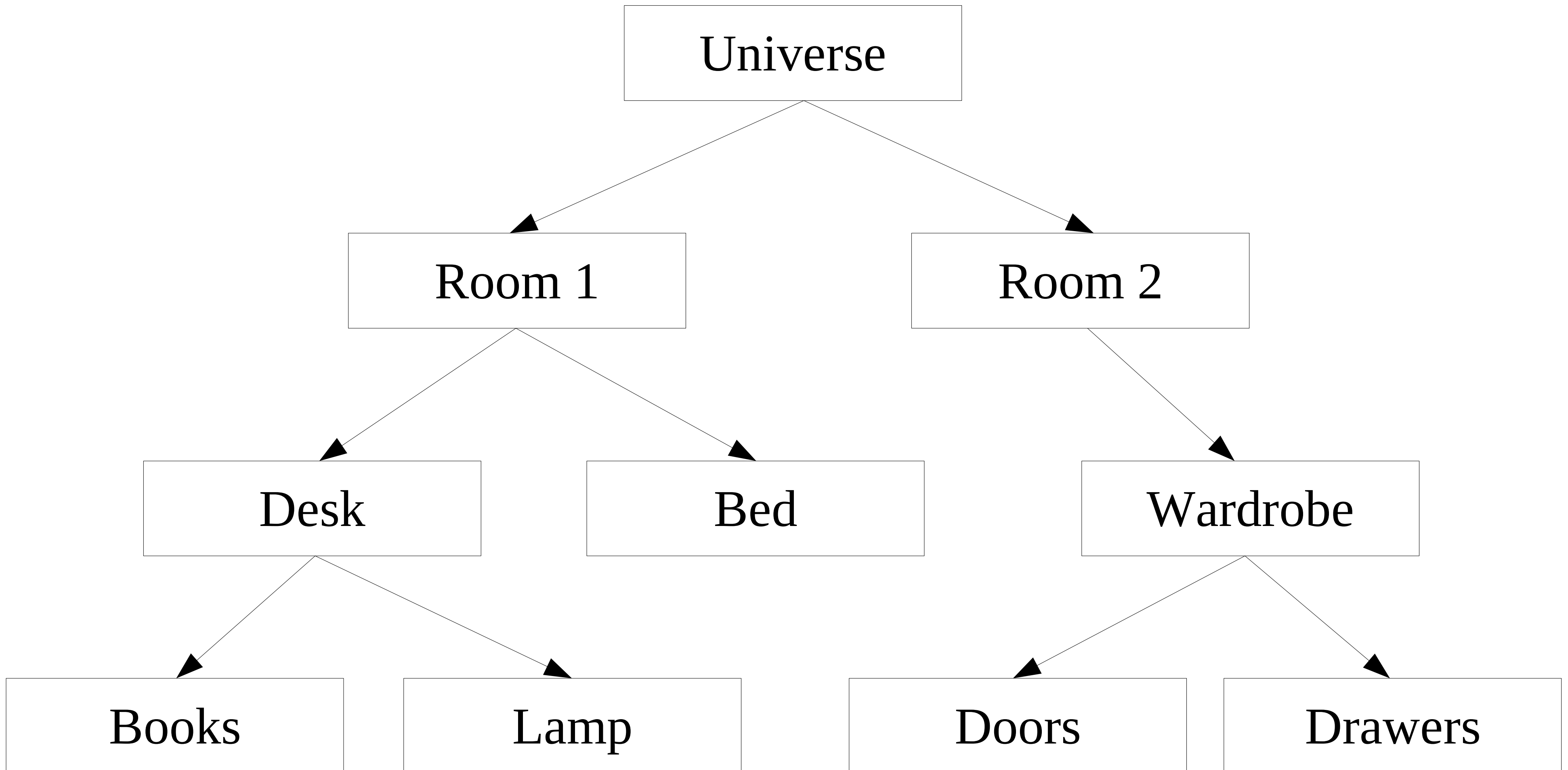
Visitor Pattern

3D Room

Scene Graphs



What if?

I want to print out the content of the universe.

To do this, I need to build a string containing a list of the items in the room.

How do I do this?

universe.toString() (recursively)

The Challenge

The class calling the universe.toString() method should not have information on how data is stored in the universe.

Thus, universe.toString() should take care of traversing the tree.

This means that each node will need to have its own toString() method.

If I want to calculate the weight of the universe, I will also need to add a getWeight() function to each node.

Is there a generic way I can traverse a structure (tree) without having to add new methods to the classes encoding the structure?

Visitor Pattern

Put an operation to be performed on the elements of an object structure in a separate class.

Separate the algorithm from the data structure.

Introduction to Compilers

A compiler is a tool that transforms a program/model from a high level representation to a lower level representation.

Java -> Bytecode

C -> Assembler

The first step of a compiler is to transform the source code into an abstract syntax tree (as specified by a language “grammar”).

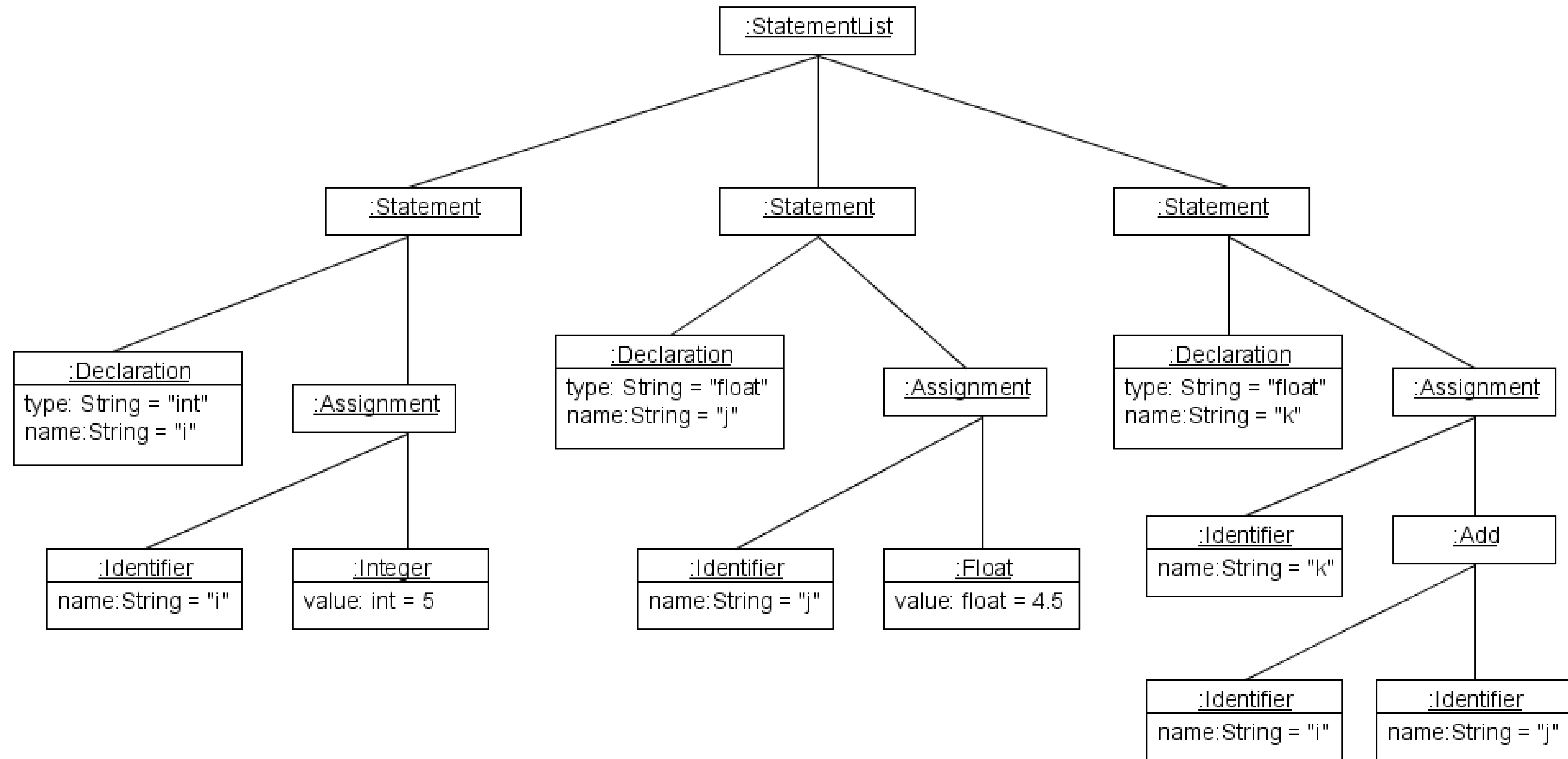
Flex + Bison in C

SableCC in Java

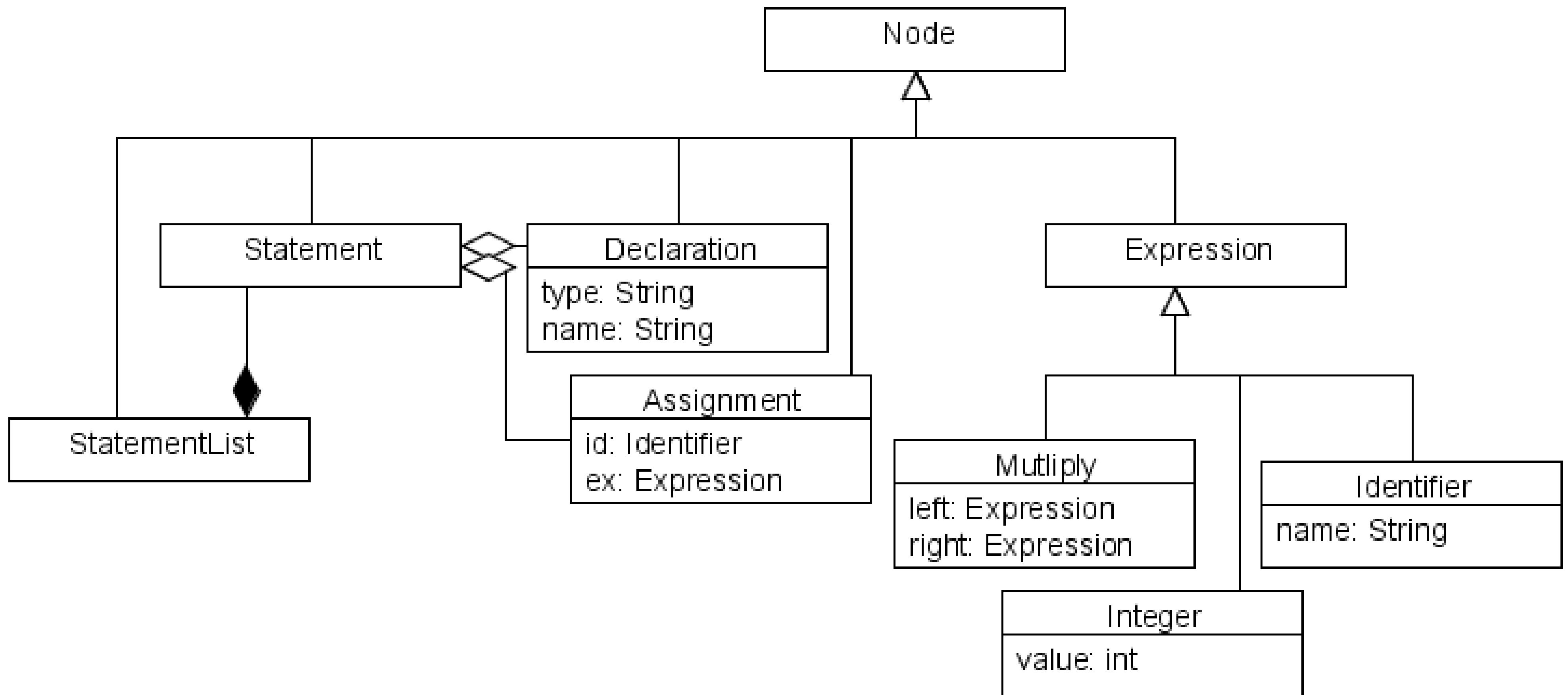
The Code (input to compiler)

```
int i = 5;
float j = 4.5;
float k = i + j;
```

The constructed AST



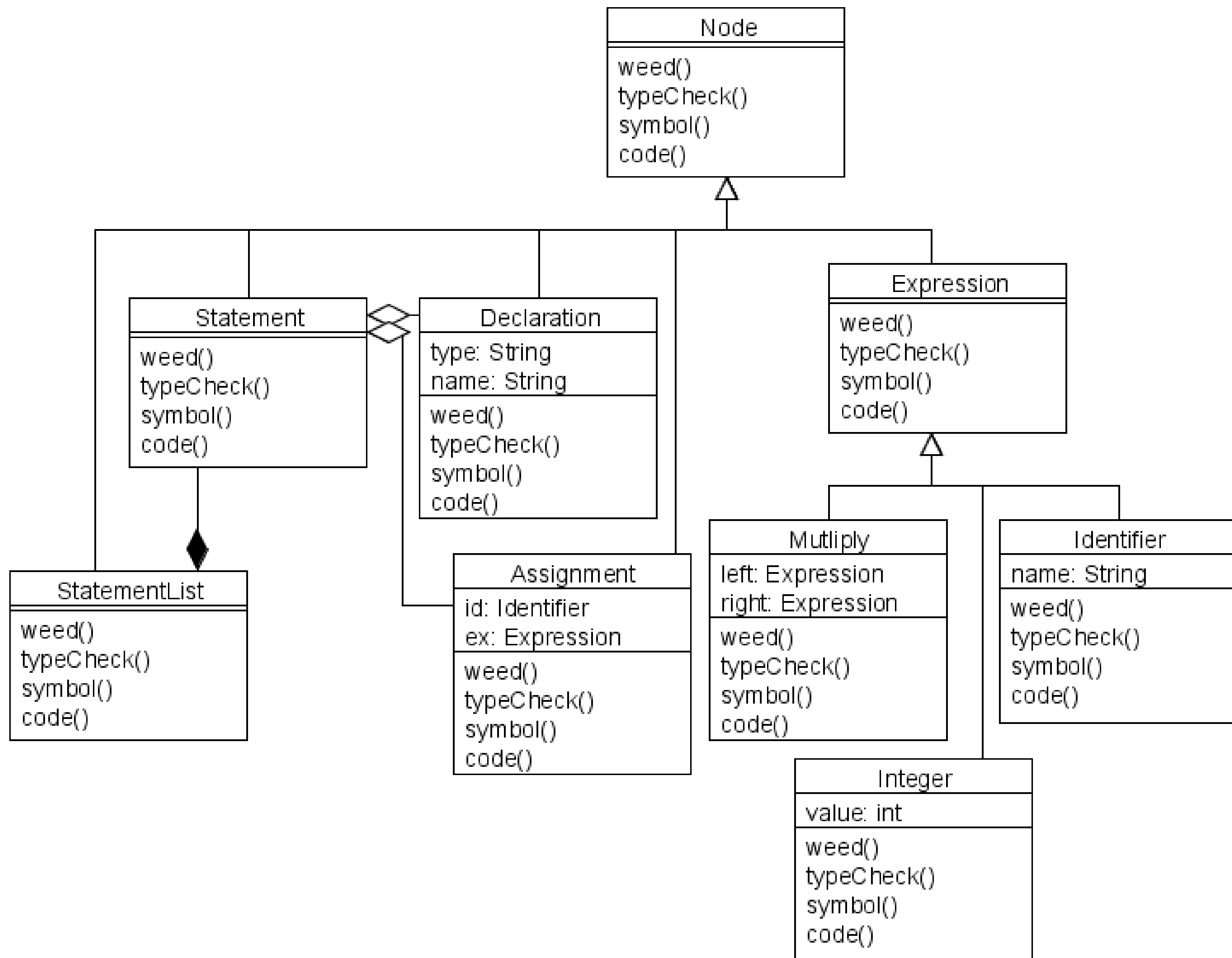
Class Diagram of the AST



Compilers Continued

- Further operations are done by traversing the tree
  - Weeding
  - Type Checking
  - Symbol Table Generation
  - Code Generation
- Do we want to add (recursive) methods to every node we need to traverse?
  - This would be the intuitive solution
  - We would need the following methods: weed(), typeCheck(), symbol(), code()

Naieve Solution



Problem

Each node class is polluted with several methods.

The implementation of an algorithm is spread over all classes.

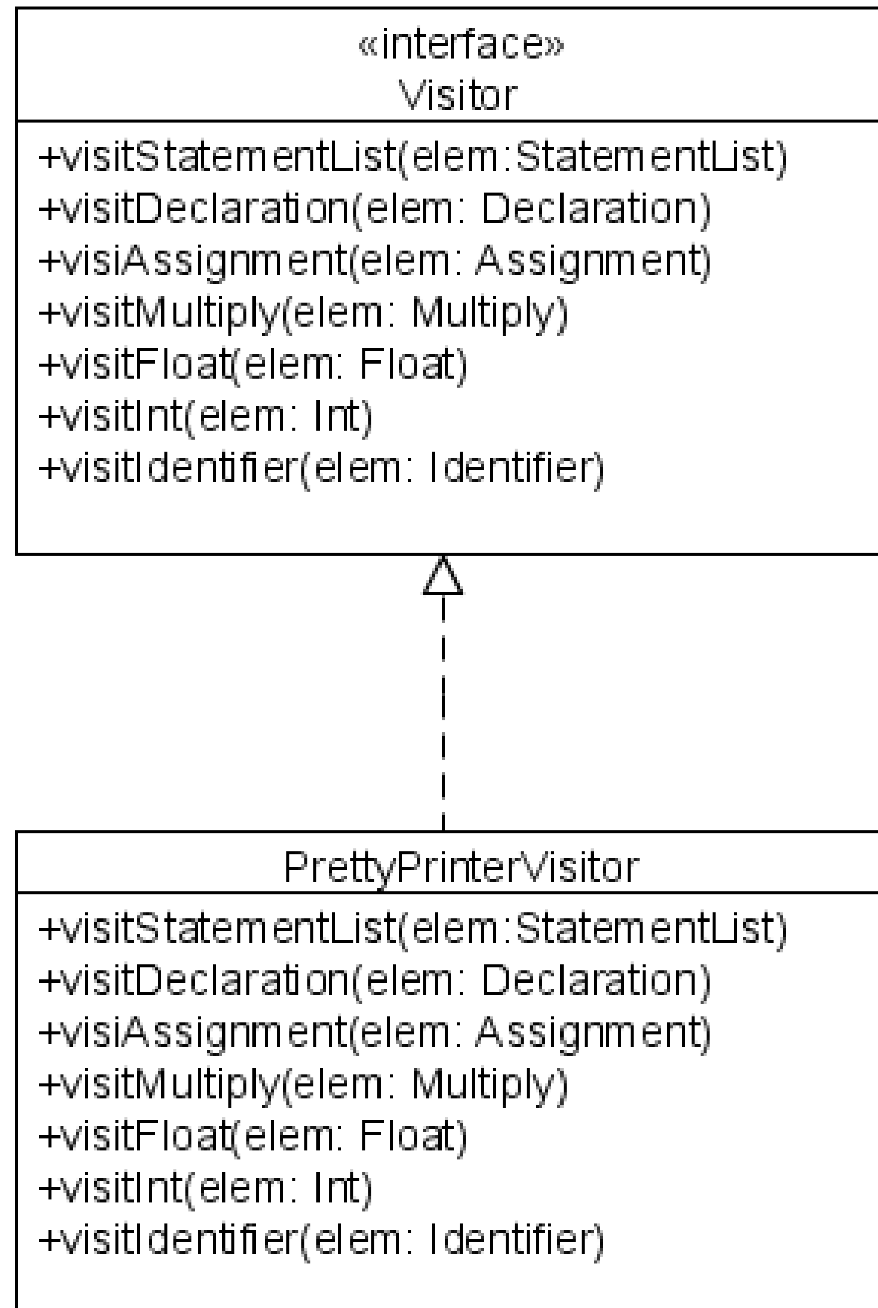
i.e. The weeding algorithm is spread across several nodes.

To keep track of the traversal, either

must use global variables

must pass arguments by reference in each method call

Visitor Pattern Solution



Advantages

The traversal algorithm is now located in a single class.

All variables needed to execute the algorithm are also in the class.

No need for global variables anymore (or variables passed by reference).

The AST class structure (tree) was not modified!

- It could even be pre-compiled!

It's easy to add new operations (new Visitor sub-class).

Disadvantages

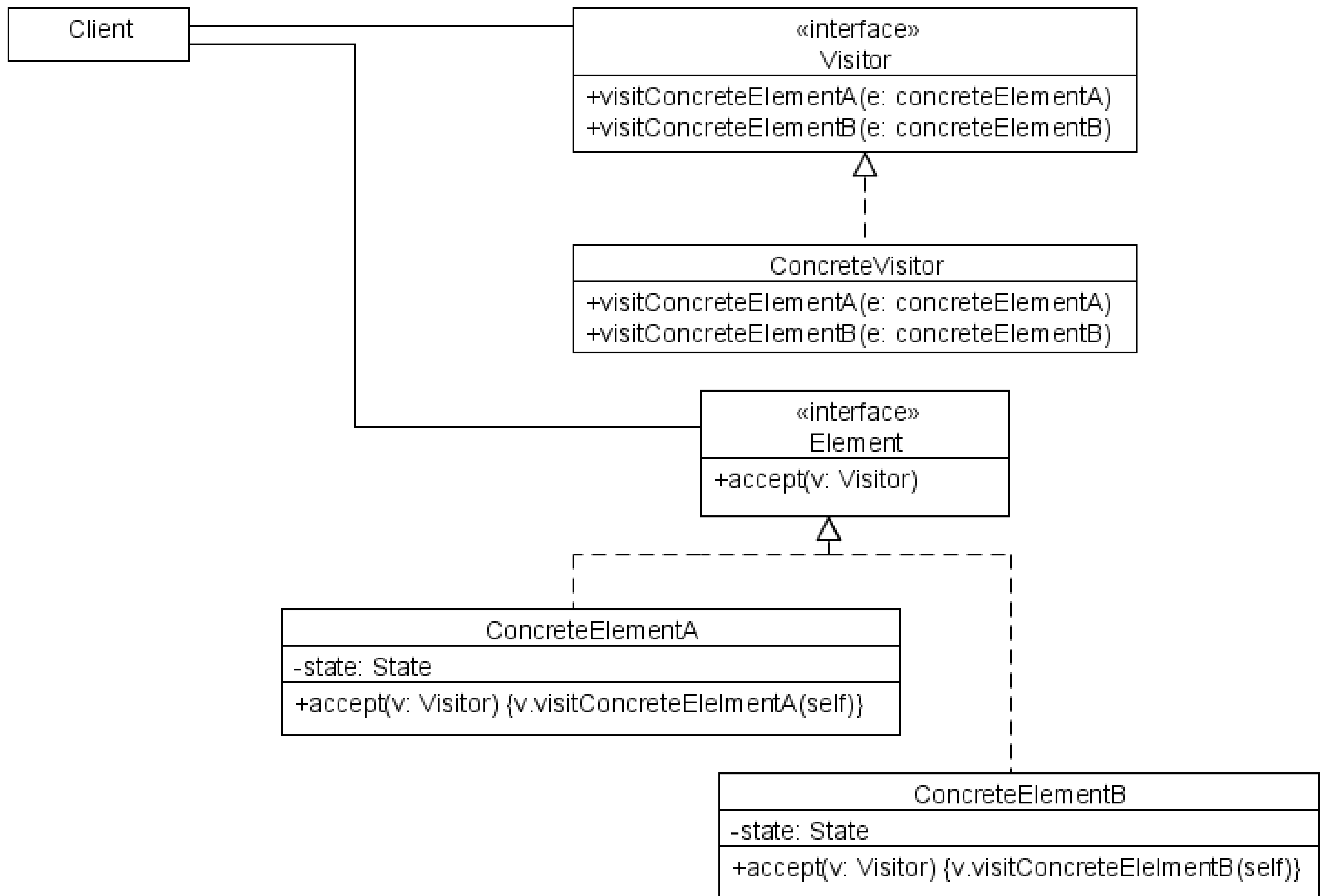
However, if a new subtype of Node is added, all the visitors must be modified.

For instance, we might want to add an 'Addition' node.

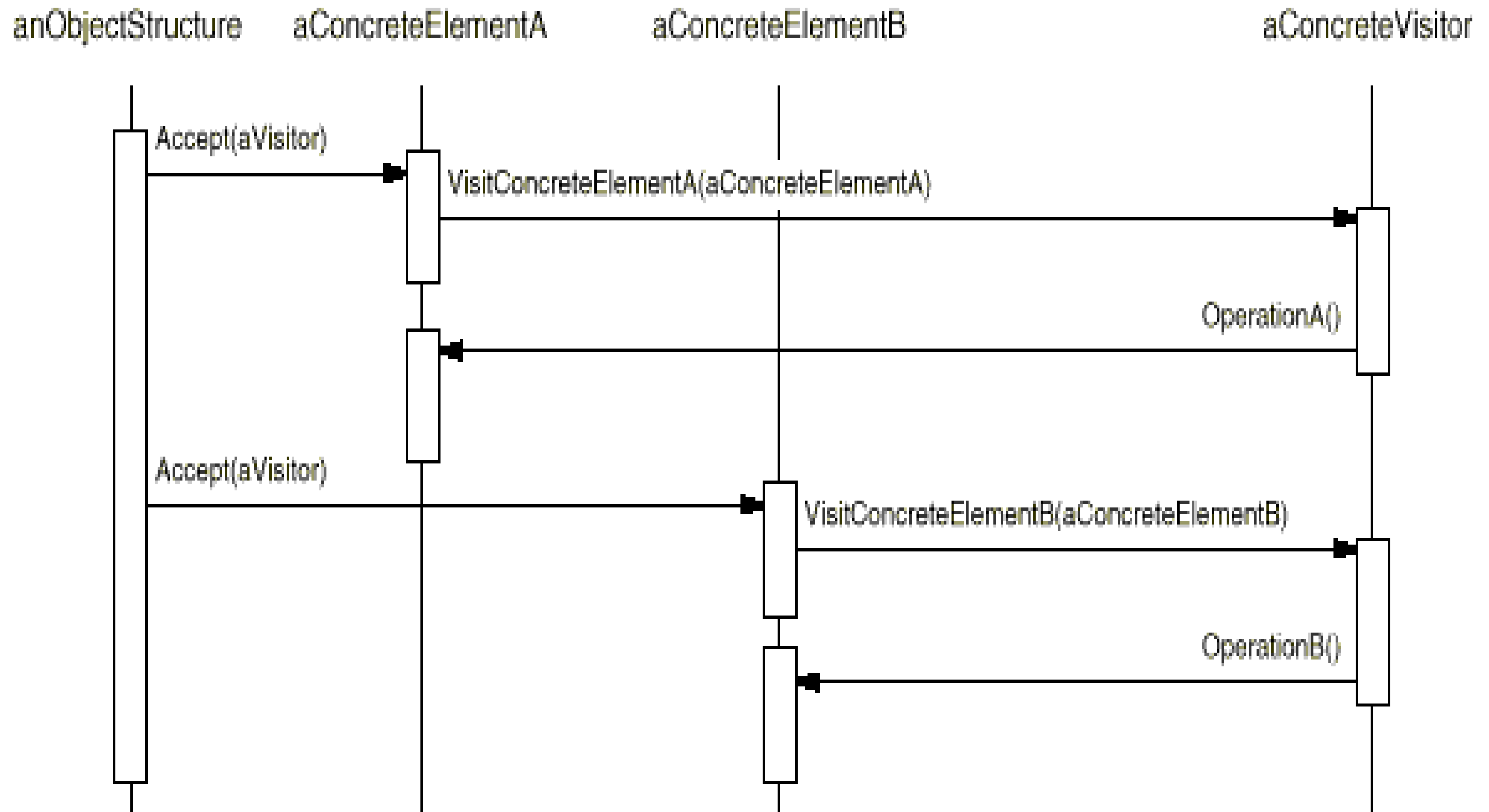
This would require a new function 'visitAddition' in each visitor.

Encapsulation could be broken if a visitor needs to access an element's internal state.

Class Diagram



Sequence Diagram “double dispatch”



Composite Elements

Data structures are often composite: a node contains references to other nodes (children, etc).

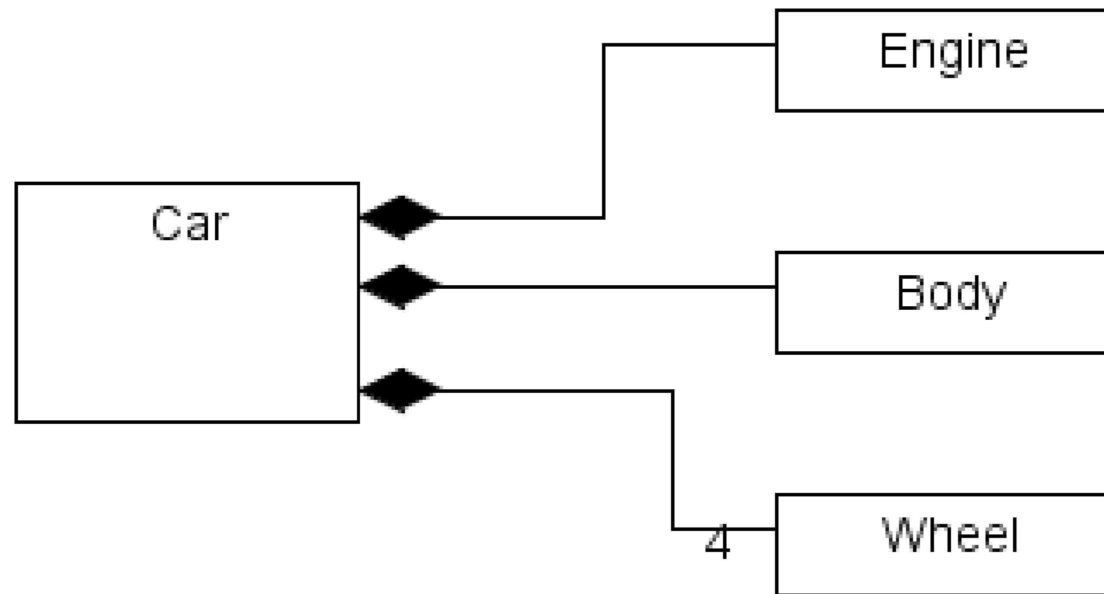
For the visitor pattern to work, the accept() calls must be propagated to the children nodes (other references).

Most often, the simplest solution is add this propagation to the accept() call of the parent.

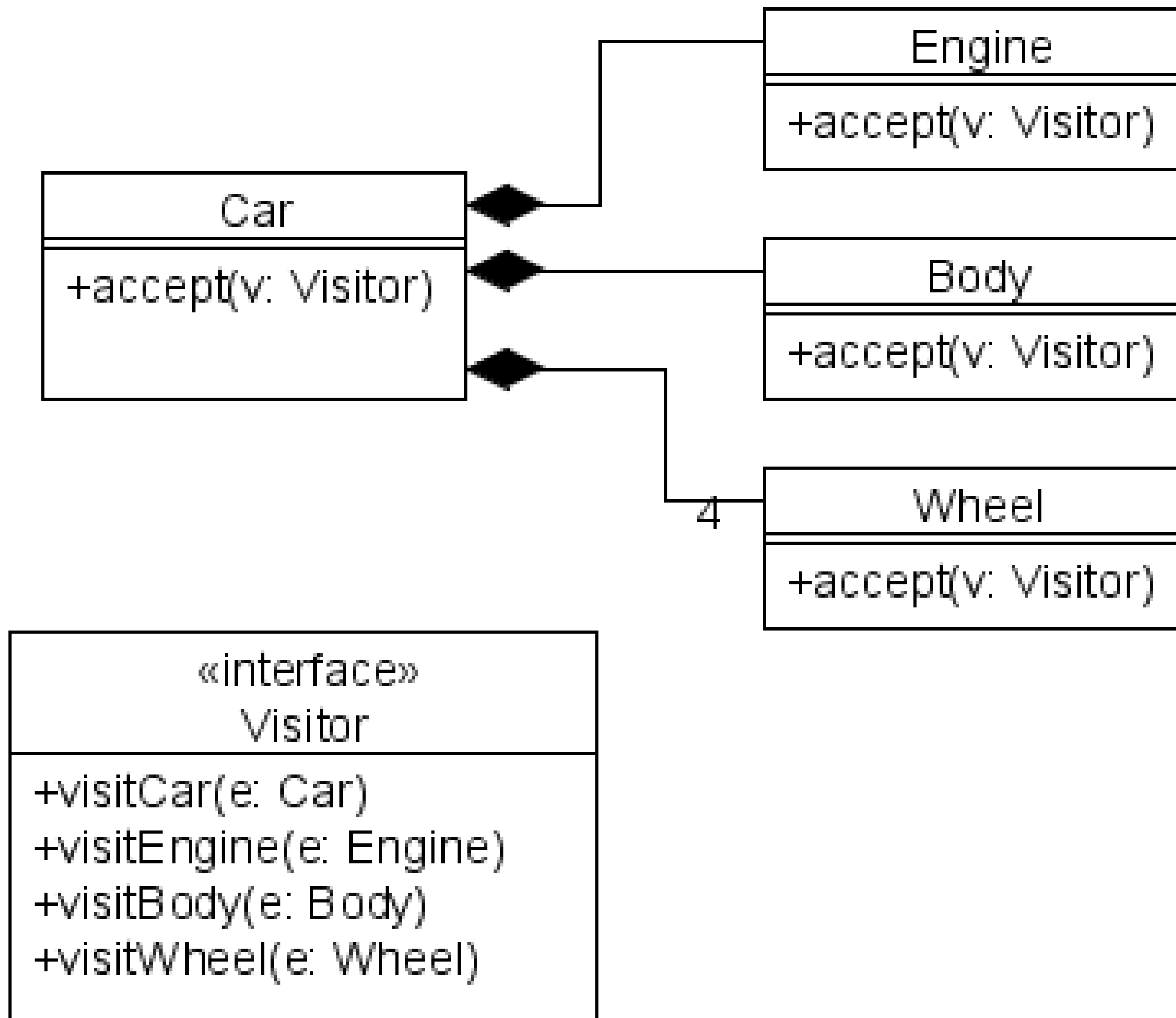
```
public void accept(Visitor visitor) {
      visitor.visit(this);

      for (Node node: nodes) {
         node.accept(visitor)
      }
}
```

Example



Add the visitor pattern



Wheel, Body, Engine

```
class Wheel {
    public void accept(Visitor visitor) {
        visitor.visitWheel(this);
    }
}

class Engine {
    public void accept(Visitor visitor) {
        visitor.visitEngine(this);
    }
}

class Body {
    public void accept(Visitor visitor) {
        visitor.visitBody(this);
    }
}
```

Car

```
class Car {

    private Engine  engine;
    private Body    body;
    private Wheel[] wheels;

    public void accept(Visitor visitor) {
        visitor.visitCar(this);
        engine.accept(visitor);
        body.accept(visitor);
        for(int i = 0; i < wheels.length; ++i) {
            wheels[i].accept(visitor);
        }
    }
}
```

Visitor

```
class PrintVisitor implements Visitor {
    private static count = 0;

    public void visitWheel(Wheel wheel) {
        count++;
        System.out.println("Visiting wheel " + count);
    }
    public void visitEngine(Engine engine) {
        System.out.println("Visiting engine");
    }
    public void visitBody(Body body) {
        System.out.println("Visiting body");
    }
    public void visitCar(Car car) {
        System.out.println("Visiting car");
    }
}
```

Caveat: multiple visits?

Composite Concerns

When dealing with composites, who should take care of the traversal?

The Composite

An External class

The Visitor

Traversal encoded in Composite

Using the composite to take care of the traversal is the simplest solution. Remember the car example.

```
public void accept(Visitor visitor) {
    visitor.visitCar(this);
    engine.accept(visitor);
    body.accept(visitor);
    for(int i = 0; i < wheels.length; ++i) {
        wheels[i].accept(visitor);
    }
}
```

Unfortunately, this only works if all the visitors need to visit the elements in the same order.

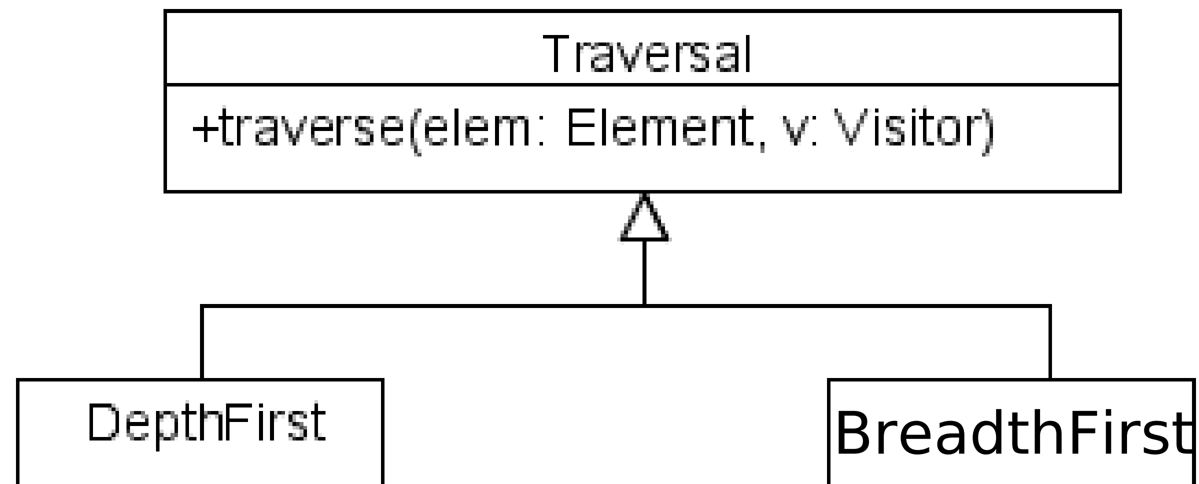
Traversal encoded in External Class

Use an external class to define the traversal.

That class would require internal knowledge of the data structure, but at least the visitor would remain generic.

This traversal object could even be an iterator.

Traversal encoded in External Class



Traversal encoded in Visitor

To allow different traversal orders (for different visitors), the traversal could be in the visitors.

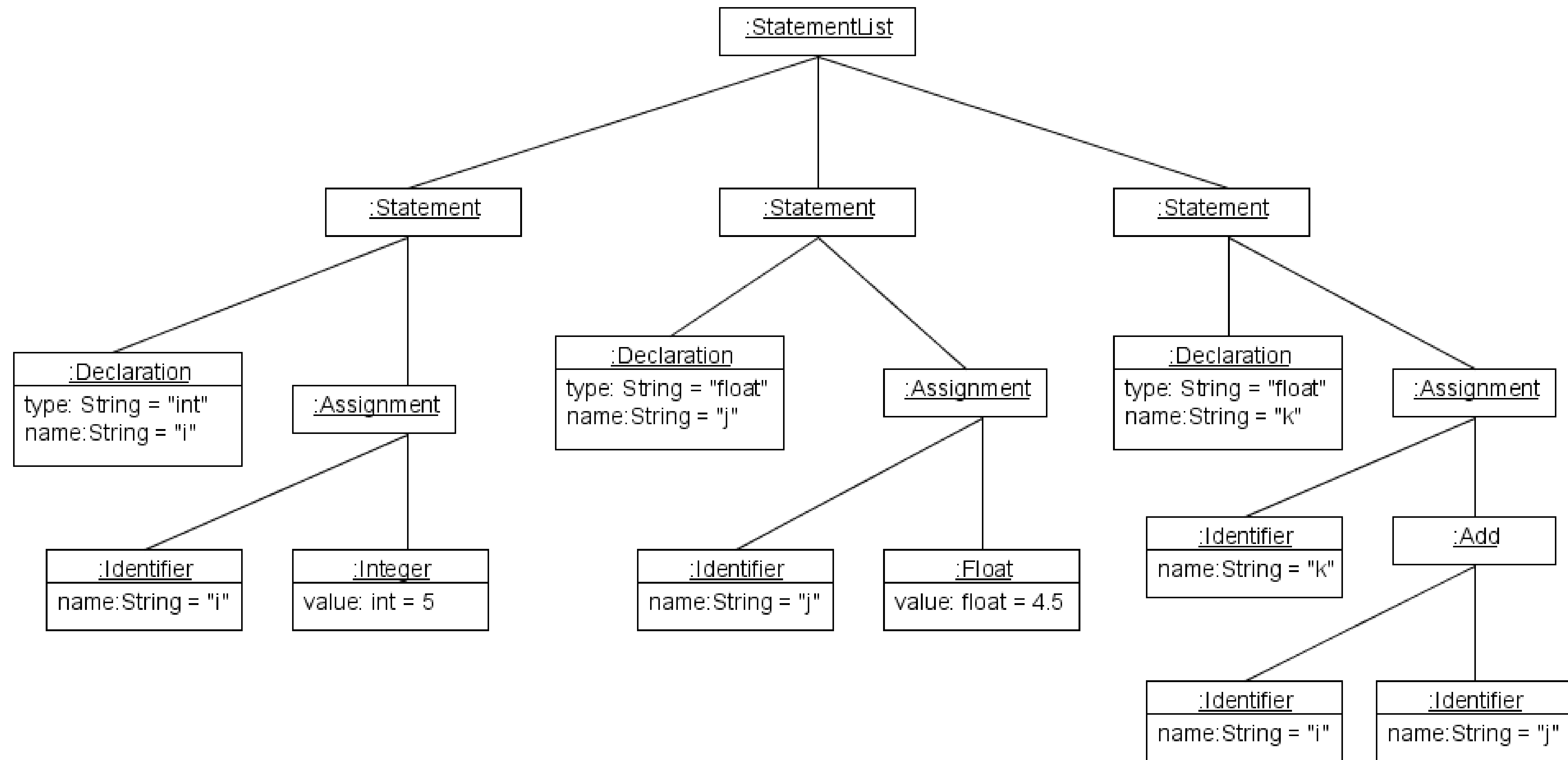
This would allow each visitor to use a specific traversal.

This is the only solution for very complex traversal.

Back to the AST example

```
int i = 5;
float j = 4.5;
float k = i + j;
```

AST example



Visitor

```
class PrettyPrinterVisitor implements Visitor {
    ...
    public visitStatement(Statement elem) {
       elem.def.accept(this)
       elem.assign.accept(this)
       print(“ ;”);
    }

    public visitAssignment(Assignment elem) {
       elem.id.accept(this)
       print(“ = “);
       elem.exp.accept(this)
    }
    ...
```

Drawbacks

The visitor needs to know and understand the data structure.

- Breaks the abstraction, creates coupling

Each visitor must include information on how to traverse the data structure.

- Can lead to lots of duplicate code.

DSheet

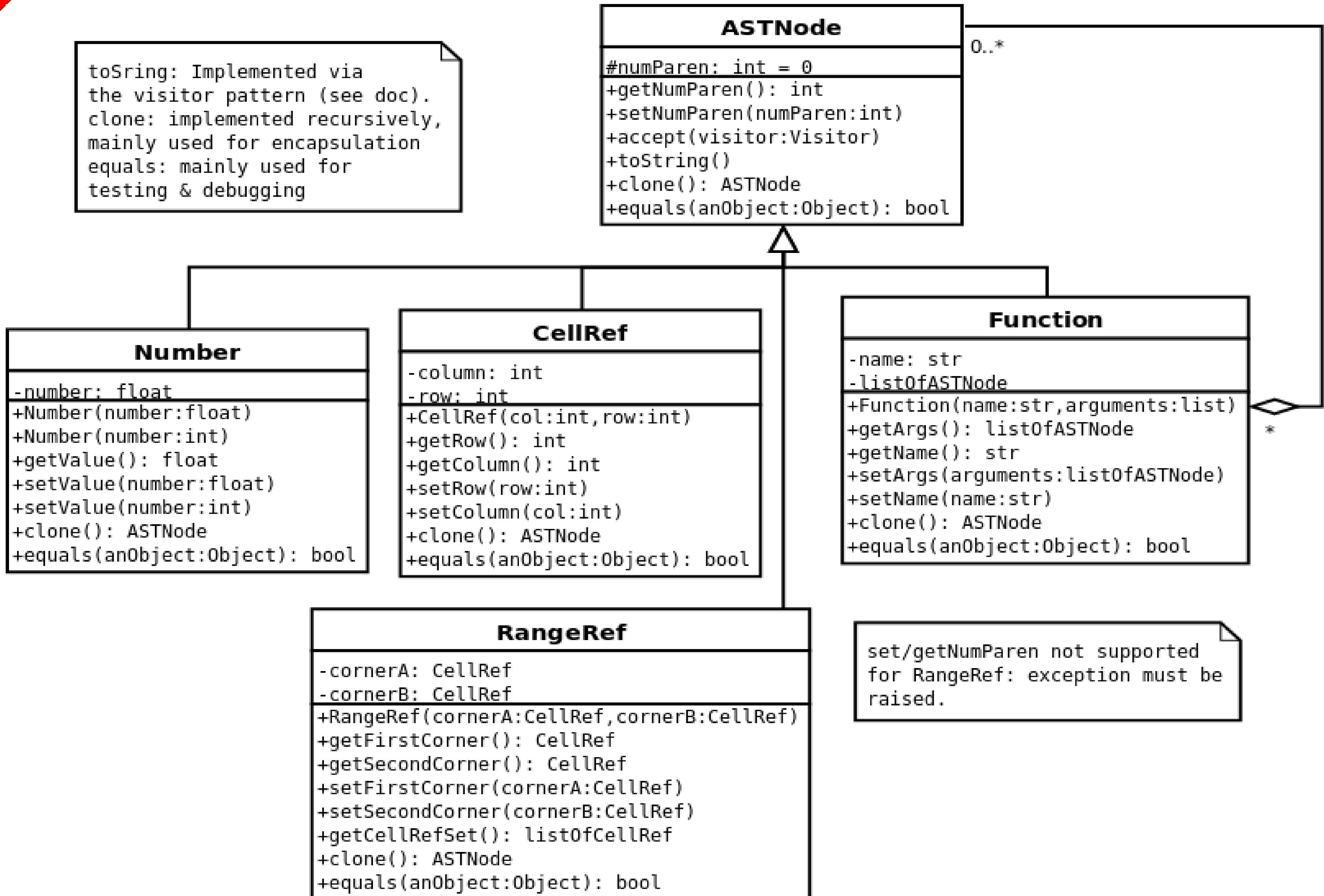
DSheet <observing subject 1> 0.90

File Edit

A9 00:36

	A	B	C	D	E	F	G	H	I	J	K
1											
2											
3											
4											
5											
6											
7											
8											
9											
10											
11											
12											
13											
14											
15											
16											
17											
18											
19											
20											
21											
22											
23											
24											
25											
26											

Dsheet: Composite Pattern ++



Dsheet: Visitor Pattern