Comp-304 : Visitor (cont.)
Lecture 30

Alexandre Denault
Original notes by Marc Provost
and Hans Vangheluwe
Computer Science
McGill University
Fall 2007
7 / 23 = 30.4%
Thursday April 12th
Friday April 13th
Saturday April 14th
Sunday April 15th
Monday April 16th
int i = 5;
float j = 4.5;
float k = i + j;
class PrettyPrinterVisitor implements Visitor {

    public visitStatementList(StatementList elem) {
        //Nothing to do
    }

    public visitStatement(Statement elem) {
        print(" ;");
    }

    public visitDeclaration(Declaration elem) {
        print(elem.getType() + " ");
    }

    public visitAssignment(Assignment elem) {
        print(" = ");
    }
}
public visitMultiply(Multiply elem) {
    print(" * ");
}

public visitFloat(Float elem) {
    print(elem.getValue());
}

public visitInt(Int elem) {
    print(elem.getValue());
}

public visitIdentifier(Identifier elem) {
    print(elem.getName());
}
When dealing with composites, who should take care of the traversal?
When dealing with composites, who should take care of the traversal?

- Composite
- Visitor
- External class (somebody else)
Using the composite to take care of the traversal is the simplest solution.

We saw something similar to this with the car example.

```java
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.

Using the Composite to control the traversal leaves us with very little flexibility.
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 considered an iterator.
External Class for Traversal

- `+traverse(elem: Element, v: Visitor)`
  - `DepthFirst`
  - `BreathFirst`
To allow different traversal orders, the traversal could be in the visitors.

This would allow each visitor to use a special traversal.
- This is the only solution for very complex traversal.

What's bad about this?
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)
    }
    ...

The Problem

- 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 represent lots of duplicate code.