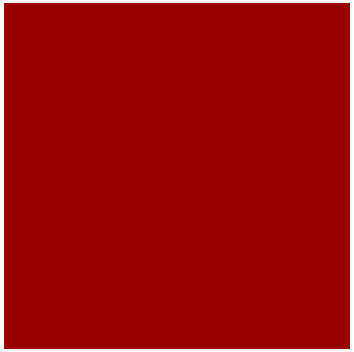
COM S 342

Recitation 09/16/2019 – 09/18/2019



Topic

- Visitor Pattern

- Q&A

Visitor Pattern

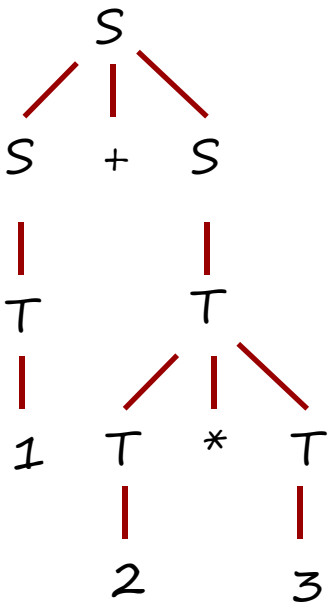
- The interpreter runs a three-step loop (Read Eval Print Loop) REPL
- The first step read consumes a program and produces an AST representing the program
- Eval consumes an AST and produces its value.

The problem is:

How to consume the AST?

Visitor Pattern

- Here is a paser tree. The expression of it is 1+2*3
- The value of this expression would depend on its subexpressions "1" and "2 * 3"
- It is easy to understand that we need to traverse this tree to evaluate it.



Visitor Patter

- Two strategies to consume the AST and require traversing it
  - Extend the implementation of each class to implement the functionality, or
  - Extend the implementation of each class to implement a generic traversal functionality.
- The second one is easier to maintain

Visitor Pattern

- In AST class, every class implemented a method accept that takes an object of type Visitor as a parameter and invokes method visit on that object

```
asbtract class CompoundArithExp extends Exp {
   List<Exp> _rep;
   public CompoundArithExp(List<Exp> args){
      _rep = new ArrayList<Exp>();
      _rep.addAll(args);
   }
   public List<Exp> all(){
      return _rep;
   }
}
class AddExp extends CompoundArithExp {
   public AddExp(List<Exp> args){
      super(args);
   }
   public Object accept (Visitor visitor ) {
      return visitor.visit(this);
   }
}
```

Visitor Pattern

- The type Visitor is defined as follow.
- The interface provides a method visit for each concrete AST node
- Concrete AST traversal functionalities can be implemented by extending the Visitor interface

```
interface Visitor <T> {
   T visit (NumExp e);
   T visit(AddExp e);
   T visit (SubExp e);
   T visit (MultExp e);
   T visit (DivExp e);
   T visit(Program p);
}
```

Visitor Pattern

- How to evaluate the expression (* 1 2 3)?

Step 0: Read & construct the AST object as follows

```
Exp exp1 = new NumExp(1);
Exp exp2 = new NumExp(2);
Exp exp3 = new NumExp(3);
List<Exp> expList = new ArrayList<Exp>();
expList.add(exp1);
expList.add(exp2);
expList.add(exp3);
MultExp multExp = new MultExp(expList);
Program prog= new Program(multExp);
```

Visitor Pattern

- How to evaluate the expression (* 1 2 3)?

Step 1: To calculte the expression, we will call the method **valueOf** in class **Evaluator**, then this method invokes method **accept** on the obeject **multExp**

```
Evaluator eval = new Evaluator();
Value val = eval.valueOf(p);

Value valueOf(Program p) {
   return (Value) p.accept(this);
}
```

Visitor Pattern

- How to evaluate the expression (* 1 2 3)?

Step 2: The method **accept** on the obeject **multExp** will causs method **visit(MultExp e)** in class Evaluator to run.

This method iterates over the component expressions **exp1**, **exp2** and **exp3**, and invokes method **accept** on each object.

```
public Value visit(MultExp e) {
    List<Exp> operands = e.all();
    double result = 1;
    for(Exp exp: operands) {
        NumVal intermediate = (NumVal) exp.accept(this);
        result *= intermediate.v();
    }
    return new NumVal(result);
}
```

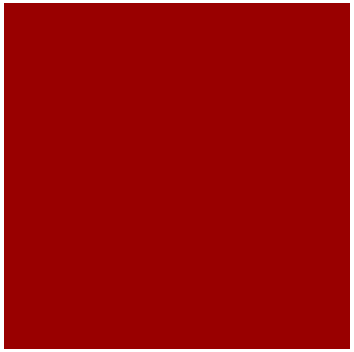
Visitor Pattern

- How to evaluate the expression (* 1 2 3)?

Step 3: The method **accept** on the obeject **NumExp** will causs method **visit(NumExp e)** in class Evaluator to run, returning result strings "1","2" and "3".

```
public Value visit(NumExp e) {
    return new NumVal(e.v());
}
```

Visitor Pattern



- How to evaluate the expression (* 1 2 3)?

Step 4: Get and return the result the method **visit(MultExp)**

```
public Value visit(MultExp e) {
    List<Exp> operands = e.all();
    double result = 1;
    for(Exp exp: operands) {
        NumVal intermediate = (NumVal) exp.accept(this);
        result *= intermediate.v();
    }
    return new NumVal(result);
}
```

Visitor Pattern

- How to evaluate the expression (* 1 2 3)?

Step 5: Return the final result 6

```
Evaluator eval = new Evaluator();
Value val = eval.valueOf(p);

Value valueOf(Program p) {
   return (Value) p.accept(this);
}
```

Q&A