Tutorial 4
Visitor Pattern

Wei Wang

Nov. 25th, 2011

Goals

- Scenario of using Visitor Pattern
- Why naïve approaches are bad?
- Elements of Visitor Pattern
- Real world application

Scenario

- Sarah wants to find out who does NOT like Lady Gaga at UWaterloo

- For male (female) students: borrowed >1 (>3)book about Lady Gaga from school library
- For profs: purchased >2 Lady Gaga CDs
- For staff: …

Essence of this scenario

- **Iterate a variety of elements under one hierarchy**
  - **Student/prof/staff**
- **Customization of the iterating algorithm**
  - **Gender/library history/purchasing history**
- **Data aggregation**
  - **Total number of student fans? What about CS only?**
- **New algorithm may arise from future demands**
  - **Looking for fans of Justin Bieber?**
- **“*open for extension, but closed for modification*”**

Naïve solution 1

```
isLadyGagaFans(IPerson){

If(Iperson instanceof
Student){
}

Else if (Iperson instanceof
Professor){

  }
}
```

Instanceof and type cast!

Naïve Solution 2

```
class Student{



  isLadyGagaFans(){

     checklibraryRecords();

     checkGender();

     }
}
```

Algorithm defined in the student class

Problems of Naïve solutions

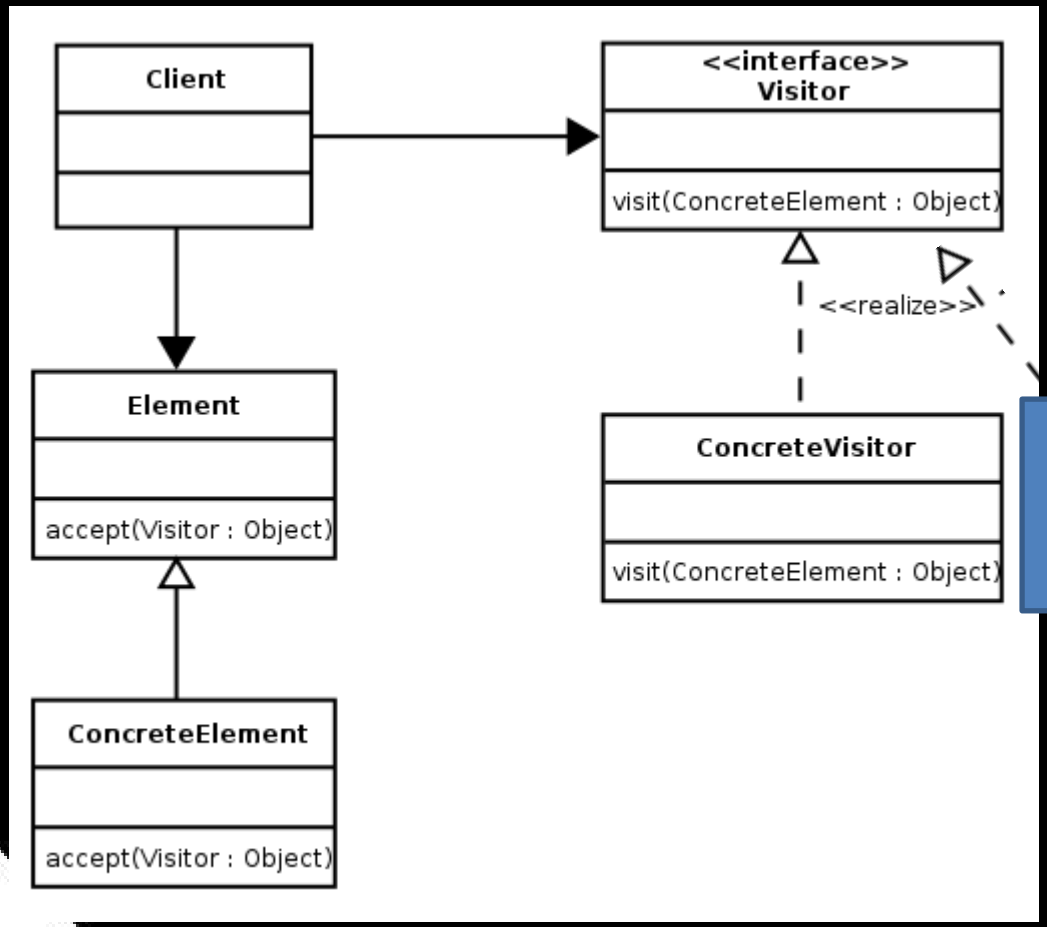
- "instanceof" solution:
  - Instanceof or type casting is error prone
  - hard coding!

- "pseudo OO" solution
  - Touch original code
  - Similar solution is scattered in many place!

Visitor Pattern Solution

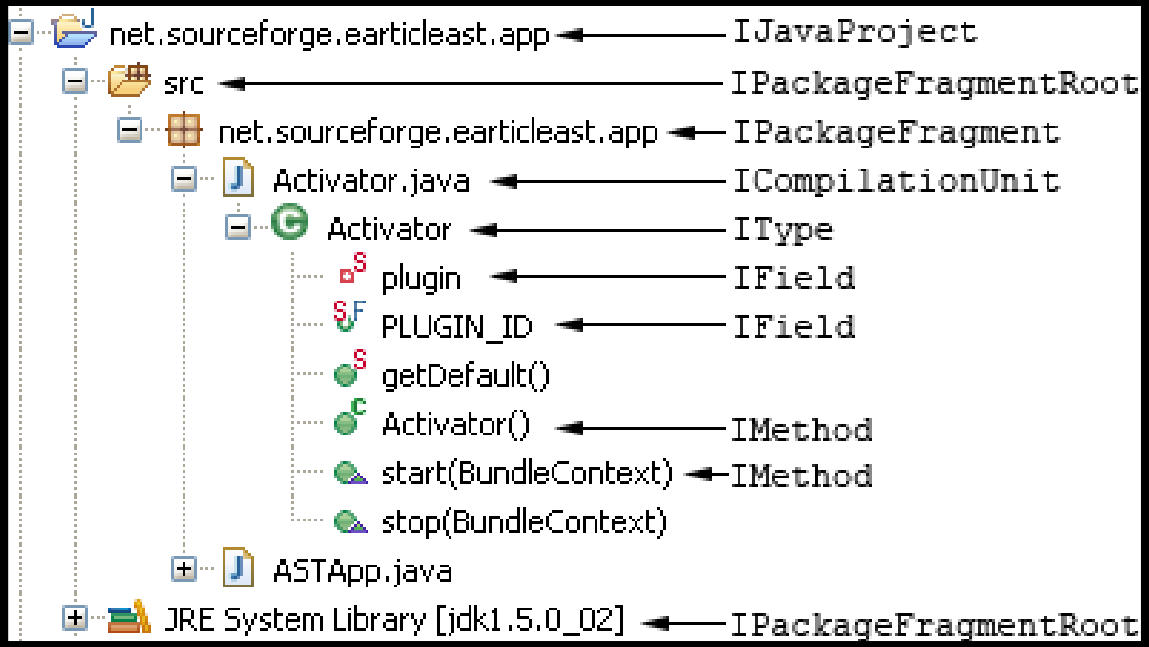
```
Class Student implement IPerson{
   void accept(Visitor visitor){
      visitor.visit(this);
   }
}

Class LadyGagaFansChecker(){
    visit(Student student){
            student.checkLibraryHistory();
            student.checkGender();
            print;
}
}

…… studentA.accept(new LadyGagaFansChecker());
```

Class Diagram of Visitor Pattern

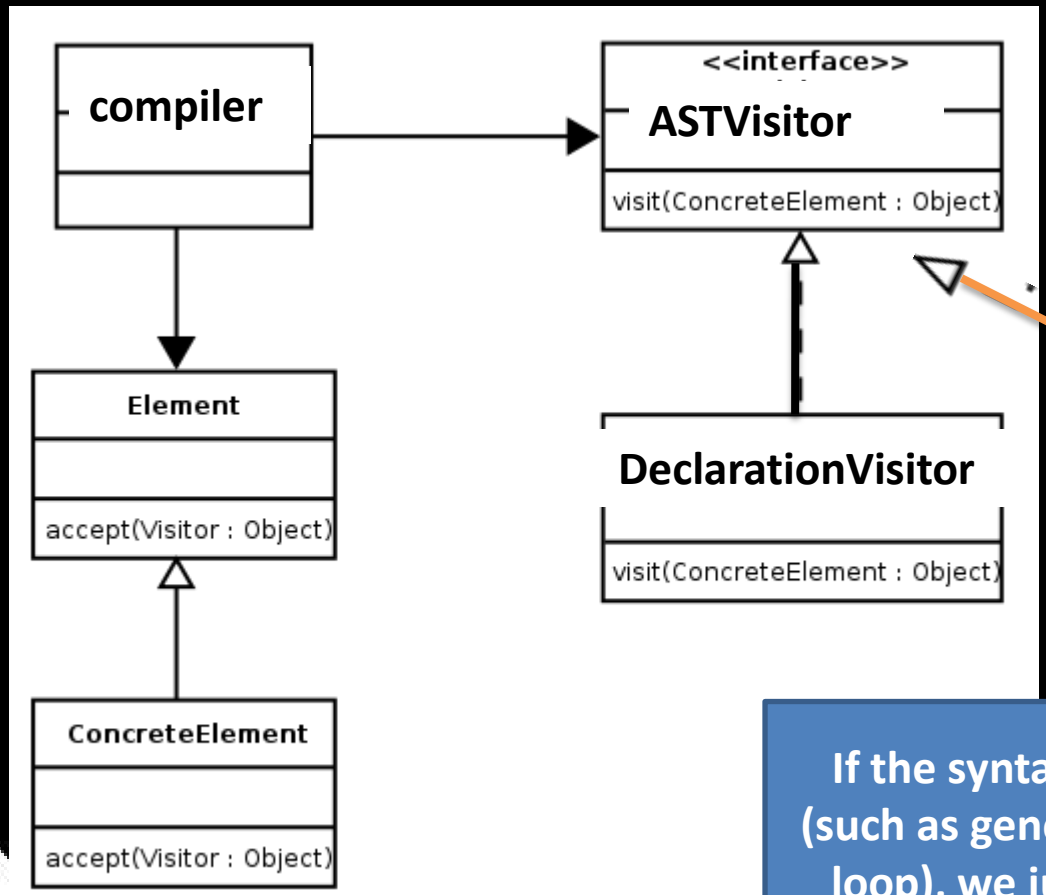


Real world application

- **Processing syntactical elements in compiler design**
  - **Eclipse JDT**



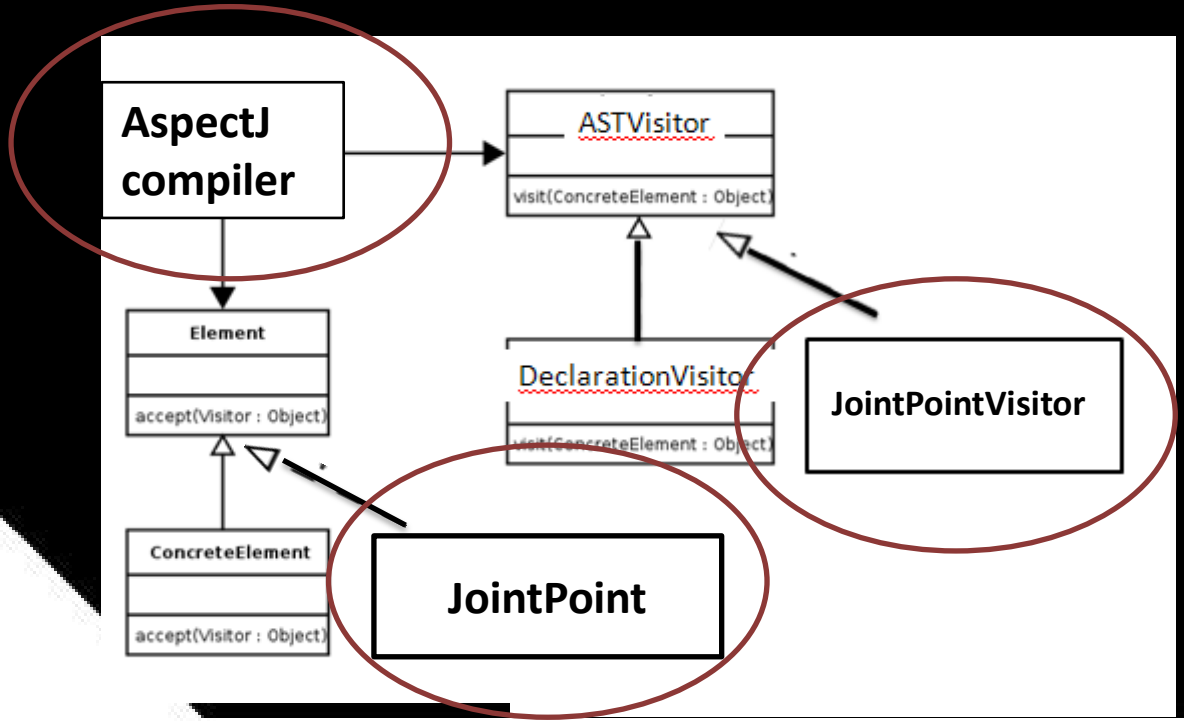
Visitor Pattern in JDT



If the syntax of Java changes (such as generics and enhanced loop), we just need add new visitors and invoke them later

What if we cannot change JDT?

- The Compiler of AspectJ reuses most of JDT elements (because we still need to compile the JAVA part of the code)



```
public class ThisJoinPointVisitor extends ASTVisitor {
    boolean needsDynamic = false;
    boolean needsStatic = false;
    boolean needsStaticEnclosing = false;
    boolean hasEffectivelyStaticRef = false;
    boolean hasConstantReference = false;
    boolean constantReferenceValue = false; // only has valid value when hasConstantReference is true

    LocalVariableBinding thisJoinPointDec;
    LocalVariableBinding thisJoinPointStaticPartDec;
    LocalVariableBinding thisEnclosingJoinPointStaticPartDec;

    LocalVariableBinding thisJoinPointDecLocal;
    LocalVariableBinding thisJoinPointStaticPartDecLocal;
    LocalVariableBinding thisEnclosingJoinPointStaticPartDecLocal;

    boolean replaceEffectivelyStaticRefs = false;

    AbstractMethodDeclaration method;

    ThisJoinPointVisitor(AbstractMethodDeclaration method) {
        this.method = method;
        int index = method.arguments.length - 3;

        thisJoinPointStaticPartDecLocal = method.scope.locals[index];
        thisJoinPointStaticPartDec = method.arguments[index++].binding;
        thisJoinPointDecLocal = method.scope.locals[index];
        thisJoinPointDec = method.arguments[index++].binding;
        thisEnclosingJoinPointStaticPartDecLocal = method.scope.locals[index];
        thisEnclosingJoinPointStaticPartDec = method.arguments[index++].binding;
    }

    public void computeJoinPointParams() {
        // walk my body to see what is needed
        method.traverse(this, (ClassScope) null);

        // ??? add support for option to disable this optimization
        // System.err.println("walked: " + method);
        // System.err.println("check:  "+ hasEffectivelyStaticRef + ", " + needsDynamic);
```

Take-away of this tutorial

- Separate the algorithm with from an object structure on which it operates
- Easy to add new operations on existing objects
  - "*open for extension, but closed for modification*"
- Almost all compilers implementation use visitor pattern to iterate syntactical elements
  - JDT & AspectJ compiler