Visitor vs Composition

Ask Question

As I see it visitor design pattern is very similar to the way composition work. In composition I would hold an interface member in the class and pass a concrete implementation of the interface in the constructor, and then either delegate a method to it or use it inside the class.

In visitor design pattern I also have a concrete implementation of the interface, and I send it to the visit method which then delegates the visit method to it.

To show this similarity in code, a visitor would be:

```
VisitorInterface v = new ConcreteVisitor();
MyClass c = new MyClass();
c.visit(v);
VisitorInterface dv = new DifferentVisitor();
c.visit(dv);
```

And composition would be:

```
SomeInterface i = new ConcreteImplementation();
MyClass c = new MyClass(i);
c.visit();  // called visit just to show the symmetry to visitor pattern
SomeInterface di = new DifferentImplementation();
c.changeReference(di);
c.visit();
```

I would like to hear your thoughts as to in which cases you would prefer one over the other and why

java design-patterns design

asked Jan 9 '17 at 7:36

Gilad Baruchian

416 7 21

3 Answers

Composition is the means of expressing a *has-a* relationship between objects, in other words, to model attributes of an object. A cow *has* horns. Injection is not essential to this. The Visitor pattern is a way to perform an external action on a type. They serve different purposes and operate on different portions of the object model. To answer your question, I prefer composition when the logic of the situation calls for a type to *have an* attribute, and the Visitor pattern to organize code to perform an action on instances of a type without altering the structure of the target type, as documented for that pattern.

answered Jan 9 '17 at 7:50

Lew Bloch
2,860 1 8 8

OK so if I understand you correctly, you would use composition when you consider the object to be a part of the class behavior, and a visitor when you want to do something to the object. For example if we have Car and Engine and we want car.getEngineRPM() then composition is the way, but if we want Car.getMarketValue() then you would use composition (calculate for different markets) – Gilad Baruchian Jan 9 '17 at 8:09

> "Visitor design pattern is very similar to the way composition work."

It is actually not. Composition is a basic principle for object creation like Abstraction, Encapsulation, Polymorphism etc. It is simply a ***has-a relationship***. Adapter, Composite and Decorator patterns are perfect examples for utilizing Composition principle.

Visitor pattern is rather a high level solution deduced from basic programming principles. The basic logic behind Visitor is method-overloading based on different sub-class types.

Ex: You have a base-class or interface named ***Bird*** and ***sub classes Crow, Duck*** and ***Penguin***. And you want a method of some client class to act different w.r.t. the type of the sub-class. i.e. here assume, I want a ***Hunter*** class which ***differentiates the behavior of Hunter.hunt() method w.r.t. whether it is Crow, Duck or Penguin***.

So my ***Hunter*** class looks like this.

```
public class Hunter{

    public void hunt(Crow crow){
        //crow hunting logic.
    }

    public void hunt(Duck duck){
        //duck hunting logic.
    }

    public void hunt(Penguin penguin){
        //penguin hunting logic.
    }

}
```

And if I do like this,

```
Bird bird = new Duck();
Hunter hunter = new Hunter();

hunter.hunt(bird);
```

Now this `hunter.hunt();` will automatically navigate into `hunt(Duck duck)` method and get executed.

I think you understood that there's no much relationship between composition principle and visitor pattern.

And as a final note, ***Visitor is not a good pattern in general***. The reason is, it makes you keep overloading more and more methods when different new sub-classes are getting added. In our example if you want to add new classes like Pigeon, Eagle etc., you will have to add methods `hunt(Pigeon pigeon)` and `hunt(Eagle eagle)` . This would be a terrible maintenance issue in large scale especially. So it would be better, if you use it only if there's no other option or no bouncing back.

answered Jan 9 '17 at 9:55

Supun Wijerathne

4,502 1 19 38

If you think about it a bit more you can see that both visitor and using composition grant you the same ability - flexible behavior of a function signiture during runtime. I think Lew Bloch's answer described well the difference. – Gilad Baruchian Jan 9 '17 at 10:17

but interesting points about Adapter, Composite and Decorator in relation to composition, and about visitor being a bad

but interesting points about Adapter, Composite and Decorator in relation to composition, and about visitor being a bad idea, I also thought it is a bit problematic, what you are describing is the violation of the open/close principle – Gilad Baruchian Jan 9 '17 at 10:25

@GiladBaruchian there's no run time flexibility at all in visitor pattern, only in compile time. Only polymorphism gives you runtime flexibility. And in general composition alone would not make our lives easy. But composition + polymorphism does. (Ex: Decorator pattern). – Supun Wijerathne Jan 9 '17 at 10:25

@GiladBaruchian yes, it's a violation of OCP. :)) – Supun Wijerathne Jan 9 '17 at 10:26

you can implement RobotHunter and give different set of methods, and use Hunter one time and RobotHunter the other time, it is the same as having a composition and changing the object at runtime (look at the code I put in the post, I do just that) – Gilad Baruchian Jan 9 '17 at 10:28

maybe you should think where you can use mentioned patterns in order to understand the difference:

Composite:
Compose objects into tree structures to represent part-whole hierarchies.Group of objects is to be treated in the same way as a single instance of an object.

Drawing example(swing library from java):a drawing may be composed(composites) of graphic primitives, such as lines, circles, rectangles, text, and so on. But in order to draw we manipulate composites exactly the same way we manipulate primitive objects.

Visitor:
Allows for one or more operations to be applied to a set of objects at runtime, decoupling the operations from object structure.

Basic idea is to have same mechanism how to walk through object structure(object structure can be tree and that looks similar to Composite) and then on each step via objects in structure we can have one or more operations,
in compilers AST is object struture and visitor can be used to walk via AST and generate binary code, or XML output or ...

hope that this helps

edited Jan 9 '17 at 13:07

answered Jan 9 '17 at 12:23

dstar55
610 2 9