


Home

PUBLIC

 Stack Overflow

Tags

Users

Jobs

TEAMS

 Create Team

# Visitor Design pattern with inner tree

Ask Question

I am having some trouble understanding the Visitor design pattern in respect to a tree. From my understanding, the tree itself would need an accept method that would take in a parameter of the visitor interface. As you traverse through the tree, the visitor.visit() method would need to access each node and data within the tree.

Assume the tree needs to be a private inner class.

The part I am getting hung up on is the parent class of the tree needs to have a method that will need to visit all nodes of the tree and has the visitor interface as a parameter as well.

I am still pretty new to Java and I would greatly appreciate any direction.

[java](#) [visitor-pattern](#)

asked Nov 14 '14 at 19:31



[cranek](#)

53 2 5

Look at composite design patern, maybe helps you – [Unlink](#) Nov 14 '14 at 19:36

## 1 Answer

Although the Visitor pattern seems often to be presented in terms of walking a tree of nodes of varying runtime type, it is important to understand that the fundamental characteristic of the Visitor pattern is *double dispatch*. As such, the Visitor pattern is equally applicable to any situation where you want to customize behavior based on the class of an object, but don't know statically exactly what that class will be.

Example:

```

interface Shape {
    public void accept(ShapeVisitor v);
}

interface ShapeVisitor {
    public void visit(Circle c);
    public void visit(Square s);
    public void visit(Shape h);
}

class Circle implements Shape {
    double radius;
    public void accept(ShapeVisitor v) {
        v.visit(this); // The correct overload is chosen at compile time
    }
}

class Square implements Shape {
    double edge;
    public void accept(ShapeVisitor v) {
        v.visit(this); // The correct overload is chosen at compile time
    }
}

class AreaVisitor implements ShapeVisitor {
    double area;

    public void visit(Circle c) {
        area = Math.PI * c.radius * c.radius;
    }

    public void visit(Square s) {
        area = s.edge * s.edge;
    }

    public void visit(Shape h) {
        // A Shape implementation other than Square or Circle
        throw new RuntimeException(
            "I don't know how to compute the area of a " + h.getClass());
    }
}

```

The `AreaVisitor` can compute the area of any shape it understands, even though the shape classes themselves provide no method for it, even when the specific type of shape is not statically known. (Look Ma, no tree!)

Where you do happen to have a tree, the Visitor's various `visit()` methods can choose which child nodes to visit, **or** the visited nodes can make that decision in their `accept()` methods. Which is more appropriate depends on the nature of the Visitor interface and of the types being visited. In either case, if the visitor is intended to traverse the tree in some way, then it is usual to start the process by asking the root node to `accept()` the Visitor object.

If there happens to be an object representing the whole tree (other than the root node), then much the same design choice I just described must be made at that level, too. You would start the visitation by passing your visitor to some method of the container object. That method certainly has the alternative of passing the visitor on to the `accept()` method of the tree's root node. If the container happens to be of a class that the Visitor is designed for, though, then it might also have the alternative of passing itself to the appropriate one of the Visitor's `visit()` methods.

Do note, however, that the Visitor pattern depends on the types that are to be distinguished by the visitor (`Circle`, `Square`, and `Shape` in the example) to be accessible in the scope of the Visitor interface and all its concrete implementations. You can use it to visit objects of other types, but they must be subtypes of one of the types for which the Visitor was designed.

answered Nov 14 '14 at 20:21



[John Bollinger](#)

71.5k 6 33 70

---