Implementing the Visitor Pattern Using Functions and Composition

José Paumard
PHD, JAVA CHAMPION, JAVA ROCK STAR

@JosePaumard https://github.com/JosePaumard

Agenda

Among the patterns from the GoF

The Visitor pattern is both very powerful

And also very complex to implement

Let us make it dynamic

By implementing it using lambdas!

What Does the Visitor Pattern Do?

Visitors represent an operation to be performed
on the elements of an object structure.
Visitors let you define a new operation
without changing the classes
of the elements on which it operates.

Erich Gamma *et ali*

Take an existing set of classes

Maybe in a hierarchy

And add operations on those classes

Without modifying them!

All these classes needs to do

Is to expose an accept(Visitor) method

How Does the Visitor Pattern Work?

Visiting Car Parts

```
public class Car { … }
public class Body { … }
public class Wheel { … }
public class Engine { … }
```

```
public class Car {

    Engine engine = …;
    Body body = …;
    Wheel wheel1 = …;









}
```

Visiting Car Parts

```
public class Car { … }
public class Body { … }
public class Wheel { … }
public class Engine { … }
```

```
public class Car {

    Engine engine = …;
    Body body = …;
    Wheel wheel1 = …;


    void accept(Visitor visitor) {
        engine.accept(visitor);
        body.accept(visitor);
        wheel1.accept(visitor);


        visitor.visit(this);
    }
}
```

Visiting Car Parts

```
public interface Visitor {

    void visit(Car car);
    void visit(Wheel wheel);
    void visit(Engine engine);
    void visit(Body body);

}

public class Engine {

    void accept(Visitor visitor) {
        visitor.visit(this);
    }
}
```

```
public class Car {

    Engine engine = …;
    Body body = …;
    Wheel wheel1 = …;


    void accept(Visitor visitor) {
        engine.accept(visitor);
        body.accept(visitor);
        wheel1.accept(visitor);


        visitor.visit(this);
    }
}
```

Visiting Car Parts

```
public interface Visitor {

    void visit(Car car);
    void visit(Wheel wheel);
    void visit(Engine engine);
    void visit(Body body);

}
```

```
public class Car {

    Engine engine = …;
    Body body = …;
    Wheel wheel1 = …;


    void accept(Visitor visitor) {
        engine.accept(visitor);
        body.accept(visitor);
        wheel1.accept(visitor);


        visitor.visit(this);
    }
}
```

Visiting Car Parts

```
public interface Visitor {

    void visit(Car car);
    void visit(Wheel wheel);
    void visit(Engine engine);
    void visit(Body body);
    void visit(Bumper bumper);
}
```

```
public class Car {

    Engine engine = …;
    Body body = …;
    Wheel wheel1 = …;
    Bumper bumper = …;

    void accept(Visitor visitor) {
        engine.accept(visitor);
        body.accept(visitor);
        wheel1.accept(visitor);
        bumper.accept(visitor);

        visitor.visit(this);
    }
}
```

To implement a new operation

You just need to implement a visitor

But you still need those accept() methods

What if you do not have them

And need to visit this hierarchy?

With lambdas you can define operations

On classes

Without adding them to the class!

A lambda is a method

That you can pass as a parameter

Or record in a registry

Demo

Let us see implement a visitor

In a dynamic way

To visit classes

That have no accept() method

It is possible to create complex patterns

By using two operations on functions:

- composition or chaining

- partial application

Creating a registry could make a Visitor

Module Wrap Up

What did you learn?

Lambdas are methods that live outside of classes

You can pass them around

You can store them in a registry

And then add behavior to a class

From outside of this class