dzone.com

Visitor Pattern Tutorial with Java Examples

James Sugrue

6–8 minutes

Today we're going to take a look at the Visitor pattern. Of all of the patterns that I've used so far, Visitor is by far the most powerful and convenient.

Vistors in the Real World

A real world analogy always helps with the understanding of a design pattern. One example I have seen for the Visitor pattern in action is a taxi example, where the customer calls orders a taxi, which arrives at his door. Once the person sits in, the visiting taxi is in control of the transport for that person.

Shopping in the supermarket is another common example, where the shopping cart is your set of elements. When you get to the checkout, the cashier acts as a visitor, taking the disparate set of elements (your shopping), some with prices and others that need to be weighed, in order to provide you with a total.

It's a difficult pattern to explain in the real world, but things should become clearer as we go through the pattern definition, and take a look at how to use it in code.

Design Patterns Refcard
For a great overview of the most popular design patterns, DZone's Design Patterns Refcard is the best place to start.

The Visitor Pattern

The Visitor is known as a **behavioural** pattern,as it's used to manage algorithms, relationships and responsibilities between objects. Thedefinition of Visitor provided in the original Gang of Four book on DesignPatterns states:

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

What the Visitor pattern actually does is create an external class that uses data in the other classes. If you need to perform operations across a dispate set of objects, Visitor might be the pattern for you. The GoF book says that the Visitor pattern can

provide additional functionality to a class without changing it. Let's see how that can work, first by taking a look at the classic diagram definition of the Visitor pattern:

Image title

The core of this pattern is the **Visitor** interface. This interface defines a visit operation for each type of ConcreteElement in the object structure. Meanwhile, the **ConcreteVisitor** implements the operations defined in the Visitor interface. The concrete visitor will store local state, typically as it traverses the set of elements. The element interface simply defines an **accept** method to allow the visitor to run some action over that element - the **ConcreteElement** will implement this accept method.

Where Would I Use This Pattern?

The pattern should be used when you have distinct and unrelated operations to perform across a structure of objects. This avoids adding in code throughout your object structure that is better kept seperate, so it encourages cleaner code. You may want to run operations against a set of objects with different interfaces. Visitors are also valuable if you have to perform a number of unrelated operations across the classes.

In summary, if you want to decouple some logical code from the elements that you're using as input, visitor is probably the best pattern for the job.

So How Does It Work In Java?

The following example shows a simple implementation of the pattern in Java. The example we'll use here is a postage system. Our set of elements will be the items in our shopping cart. Postage will be determined using the type and the weight of each item, and of course depending on where the item is being shipped to.

Let's create a seperate visitor for each postal region. This way, we can seperate the logic of calculating the total postage cost from the items themselves. This means that our individual elements don't need to know anything about the postal cost policy, and therefore, are nicely decoupled from that logic.

First, let's create our general visitable interface:

Now, we'll create a concrete implementation of our interface, a Book.

As you can see it's just a simple POJO, with the extra **accept** method added to allow the visitor access to the element. We could add in other types here to handle other items such as CDs, DVDs or games.

Now we'll move on to the Visitor interface. For each different type of concrete element here, we'll need to add a visit method. As we'll

just deal with Book for now, this is as simple as:

The implementation of the Vistor can then deal with the specifics of what to do when we visit a book.

As you can see it's a simple formula, but the point is that all the calculation for book postage is done in one central place.

To drive this visitor, we'll need a way of iterating through our shopping cart, as follows:

Note that if we had other types of item here, once the visitor implements a method to visit that item, we could easily calculate the total postage.

So, while the Visitor may seem a bit strange at first, you can see how much it helps to clean up your code. That's the whole point of this pattern - to allow you seperate out certain logic from the elements themselves, keeping your data classes simple.

Watch Out for the Downsides

The arguments and return types for the visiting methods needs to be known in advance, so the Visitor pattern is not good for situtations where these visited classes are subject to change. Every time a new type of Element is added, every Visitor derived class must be amended.

Also, it can be difficult to refactor the Visitor pattern into code that wasn't already designed with the pattern in mind. And, when you do add your Visitor code, it can look obscure. The Visitor is powerful, but you should make sure to use it only when necessary.

Next Up

Later on this week, we're going to visit the Proxy pattern.

Enjoy the Whole "Design Patterns Uncovered" Series:

Creational Patterns

- Learn The Abstract Factory Pattern
- Learn The Builder Pattern
- Learn The Factory Method Pattern
- Learn The Prototype Pattern
- Learn The Singleton Pattern

Structural Patterns

- Learn The Adapter Pattern
- Learn The Bridge Pattern
- Learn The Composite Pattern

- Learn The Decorator Pattern
- Learn The Facade Pattern
- Learn The Flyweight Pattern
- Learn The Proxy Pattern

Behavioral Patterns

- Learn The Chain of Responsibility Pattern
- Learn The Command Pattern
- Learn The Interpreter Pattern
- Learn The Iterator Pattern
- Learn The Mediator Pattern
- Learn The Memento Pattern
- Learn The Observer Pattern
- Learn The State Pattern
- Learn The Strategy Pattern
- Learn The Template Method Pattern
- Learn The Visitor Pattern

Visitor pattern Java (programming language) Element Interface (computing) Object (computer science)

Opinions expressed by DZone contributors are their own.