VISITOR PATTERN

Visitor pattern is defined (by the GoF) to be :
“ The visitor pattern is a way of separating an operation from an object structure on which it operates. … “

And thus the consequence would be the ability to add methods to a class without having to modify its structure every time.

I – MOTIVATION :

For an example in the real world, suppose you have a mansion. Suppose you’re that rich. Now after drowning in guilt about all of the sins you have committed to be where you are right now, you decide to do something good for humanity. But you’re an egotistic narcissist that is incapable of doing something solely for the sake of others, you think it’s only fair to gain some popularity for the work. And then it occurs to you “My house is so bigggg , there are certain parts of the house that I have never even used, why not turn it into a charity house ? ….. Brilliant !! “. Right now it seems the benefits would outweigh the price and intuitively a probable plan.

So you want to turn parts of your house into a charity section, where people can enter and get what they need . In a technical term, this is equivalent of trying to modify a predefined structure when a new feature is required. New feature here is providing each type of helpless people with supplement. There are problems that you are bound to face sooner or later.

You have prepared a room for theHomeless which offers food, essentials and basic supplies. But suppose you want to expand your “ business “, you realize that theCrippled need helps too ! So you modify one more room just for them, containing all handy equipments, treadmills and stuffs. Then, it has come into your mind that theAddicts need helps too ! So again you make change to another room. But being a smartass who has made a lot of money, it is clearly problematic the direction we are heading. Each time a new type of people appears, you need to modify your house a bit. Such expensive and inefficient procedure is not going to work, and you’ll be exhausted long before any profit to you or the society is well established.

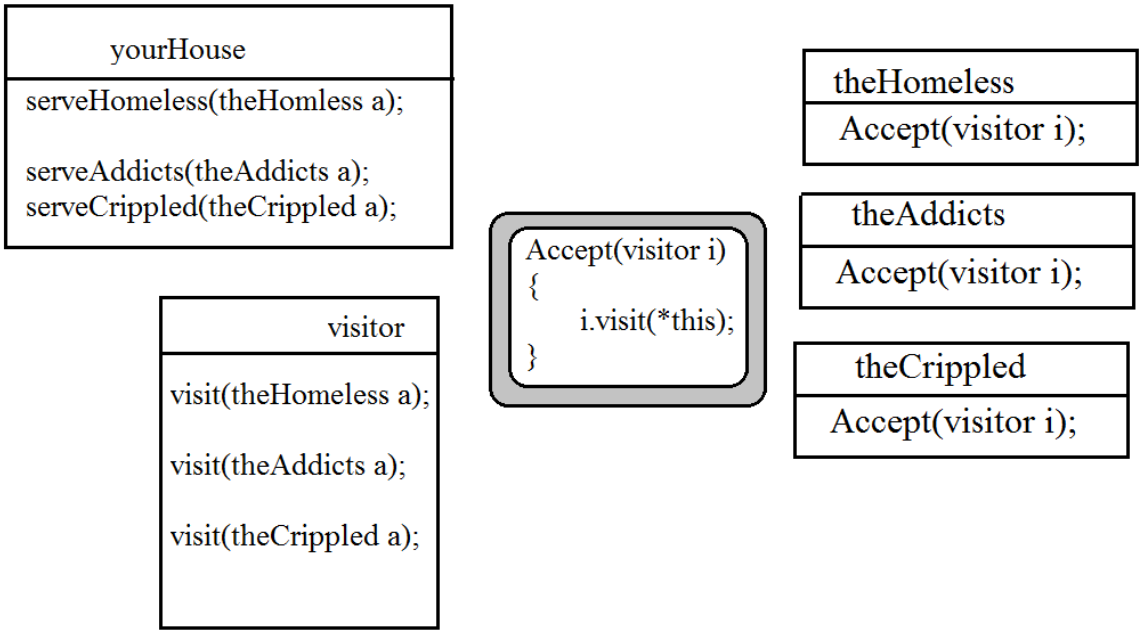
This is the problem the visitor pattern was designed to solve.

II – VISITOR PATTERN :

It was at this moment that you know you messed up. " There has got to be a more convenient way to help people " – you think to yourself. Fortunately there are people in the past that encountered the similar problem, well except perhaps they are just trying to ease their works and not help people. Point is, they have scienced the shit out of it and came up with the visitor pattern.

The cleverer approach should be joining force with an actual charity house. A charity house is an organization that already has all its fund and manpower for the sake of helping people. It's a pure structure that exists solely for one purpose. But that's boring. Of course if you will think of such an obvious solution within a few attempts. The clever part is when you hand over all the work to the charity house but still keeps the fame, which is what you want in the first place.

So you want to reserve the reputation, while avoiding all the work. One mechanism for that is to setting up a service in which poor people come into YOUR HOUSE and receive a TICKET that REDIRECTS them to the actual charity house. The cooperation is exclusive so that poor people cannot get access to the charity house without the tickets.



A ticket is a visitor. Every poor people must accept the ticket passed into the function in order to get access to the charity house. The accept method of poor people calls a method visit that is internalized to the Visitor class. Your house is the class in which presents the service of providing tickets. You have a nicely defined class and you do not want to modify its structure, so you define another class that is specified for the operations you want to perform, which is the charity house, or namely the visitor class. Inside the visitor class you have methods that return different type of poor people along with their explicitly defined supplement.

III – SHORTCOMINGS

One obvious problem is that the argument and the return type of the visiting methods have to be known in advance. Therefore one cannot using template for generic data types. And if you have polymorphism, at every level of a virtual function you have to implement another accept method. Although every accept method only has the task of invoking the visit method exclusively to the visitor class.

A lot of code is required. The visitor class with one abstract mode per class, and accept method per class. It is a tedious code to write. Each time we add a new class, a new method must also be added too. As we the metaphor above discussed, we have solved the problem of compromising stability for changes by compromising resource for changes instead. But the troublesome work which is when a new class comes into existence, methods applicable to it must also be implemented elsewhere and in scattered places.

Then there's the ironic situation where the designer was not thoughtful and decided to implement the system without a visitor pattern when one is needed. All structures that need visiting are to be altered accordingly. Remember that in the example above, both your mansion, the charity house, the tickets and the mechanism of exclusiveness as well as all principles of science has been pre-established, so that visitor pattern only is discovered, not directly created and imposed into the environment. Pattern already implies what occurs in nature and to be perceived and brought into mind by human. But in programming, we are playing God and thus it is our job to have in mind the appropriate design before the actual implementation, otherwise things would get ugly and the world would end in chaos, equivalent to you quitting the project.