Pattern Description

**Name**:

# Motivation

(Alex)

This is supporting the first 2 principles for the SOLID design. Single responsibility principle and open/closed principle.

The single responsibility principle which states that each class in a given system should have one single responsibility and a change that could occur in the system must affect the implementation of that class.

The open/ closed principle states that classes, functions from a system should be submitted for extensions but are not for modification.

This principle tells us that code can be written in such a way that whatever functionality will be added to it will not affect the integrity of that code.

In this case changes made in one class will not affect the other existing classes that are dependent on that class.

Visitor design pattern supports both these claims since changes to certain classes happen on a separate interface and modifications that occur only affect the needed class and not the other classes that are dementing to it.

Source 1: <https://www.slideshare.net/makabee/solid-49254559>

Source 2: <https://stackify.com/solid-design-open-closed-principle/>

# Solution

(Alex)

This design pattern allows for the adding of methods to different classes without modifying them significantly.

Different methods can be made depending on the class they are implemented for.

This pattern also allows for defining external classes which can in term extend other classes without the need of editing them extensively.

This method implies for the use of 2 interfaces which can diminish the duplication of multiple methods from different classes which are in an inheritance state to other classes.

Instead of just using all the methods from defend classes in one interface, a second interface can be made which only uses methods needed for a certain class and not all of them.

In case changes are needed and happen in one class, it will only affect that class and will have no consequences what so ever with regards to the other classes.

Let’s assume that we can just implement methods in each single class like tax calculation as an example.

Well what if the classes become bigger, more complex? Instead of just doing that and having our work load increased significantly we can simply implement these methods in another interface which can support these methods even if updates happen in any of the classes.

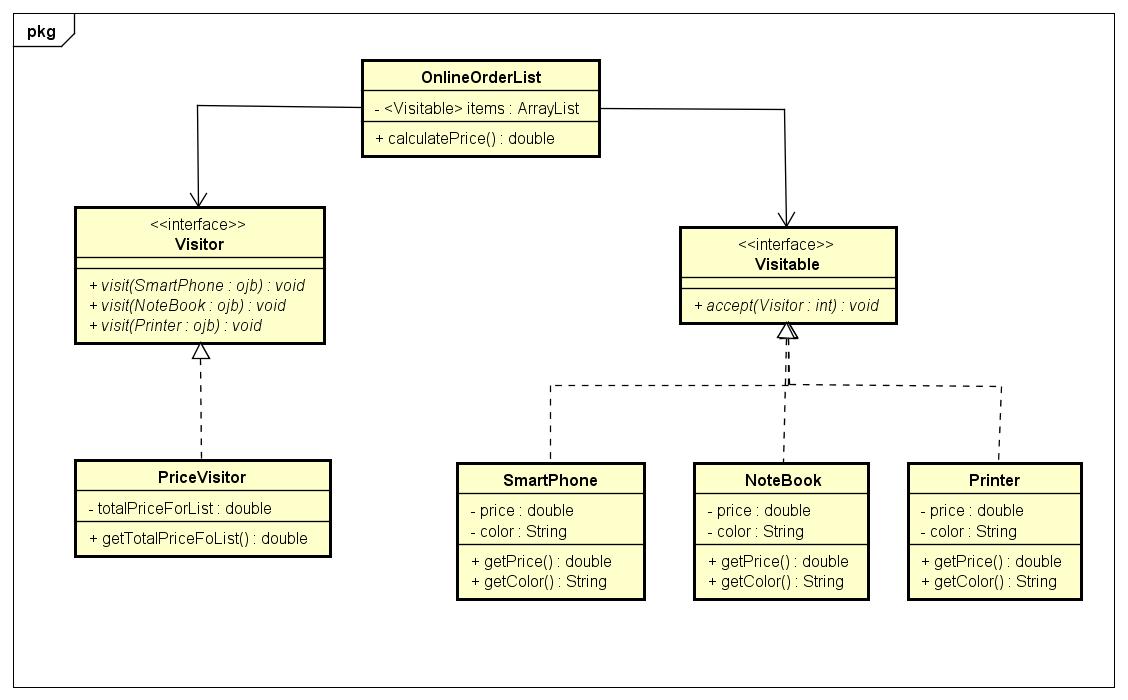
Source 1: <https://www.tutorialspoint.com/design_pattern/visitor_pattern.htm>

Source 2: <https://www.geeksforgeeks.org/visitor-design-pattern/>

## Structure

(Alex)

**Figure 1 Visitor class diagram**



## Participants

(Alex)

**SmarthPhone** class contains 2 attributes of price type double and colour of type String, a **getPrice** and a **getColour** methods and is an inheritance of **Visitable** interface.

**NoteBook** class is the same as the previous class and it also inherits the **Visitable** interface.

**Printer** class is also the same as the other 2 classes written above and shares an inheritance relation with the **Visitable** interface.

**Visitable** class or in this case interface is the link between the client and the 3 item classes. The method that it contains is **accept(Visitor:int) void** which allows the client to retrieve information about the 3 other classes.

**OnlineOrderList** class which contains the **Visitable** object as an array list inside of it so that the item classes which are inheritances will be stored.

It also contains a method called **calculatePrice()** which will return a price based on what item or items the client has chosen.

**Visitor** class is the second interface in this class diagram and contains the item classes as objects inside it.

It is a link between the client class and the rest of the item classes and serves as communication class.

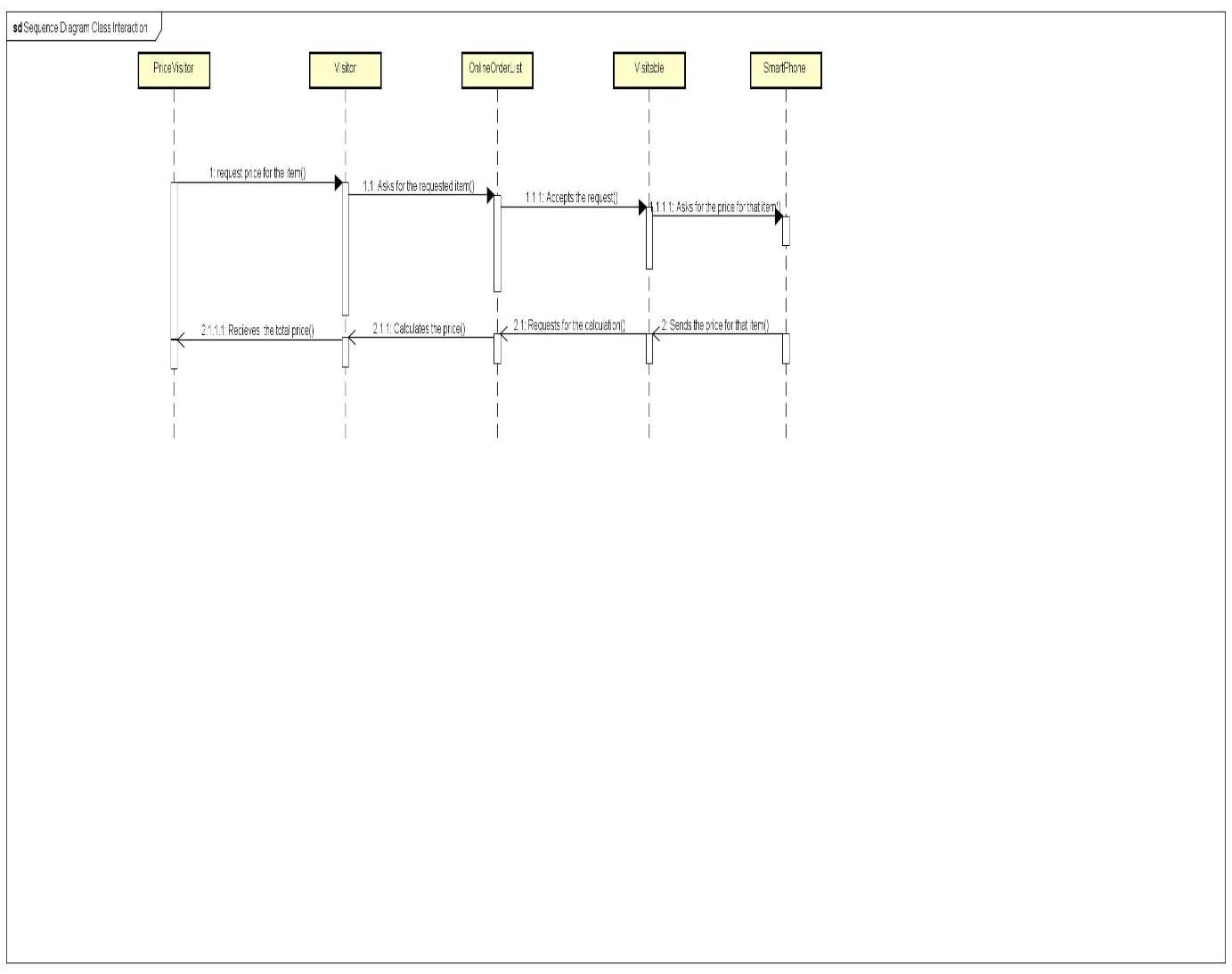
**PriceVisitor** is our client class. It contains an object called **totalPriceForList** of type double and a method of

**getTotalPriceForList** which will return the price for the desired items using the **Visitor** class to do so.

## Collaborations

(Alex)

**Figure 2 Sequence diagram of class interaction**



# Consequences

Visitor pattern has the advantage that it can take away some of the responsibility away from the product classes(SmartPhone, Printer and NoteBook) and sends the responsibility to the PriceVisitor class which can modify the values for each object. In addition the Visitor patter makes the code much cleaner.

Negative aspect is that If the user chooses to add a new product ( similar to Printer, Smart phone, NoteBook) he needs to add methods to the Visitor interface otherwise it won’t recognize the object.

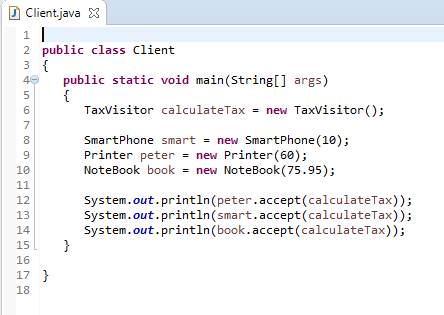
# Implementation

How is the pattern implemented?

# Sources

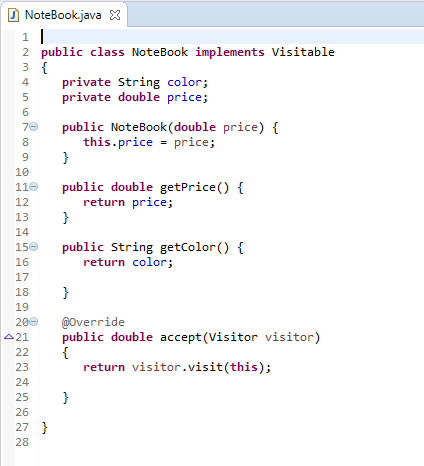
The sources you used to find the information (at least two excluding Wikipedia and Stack Overflow)

This is the Client:

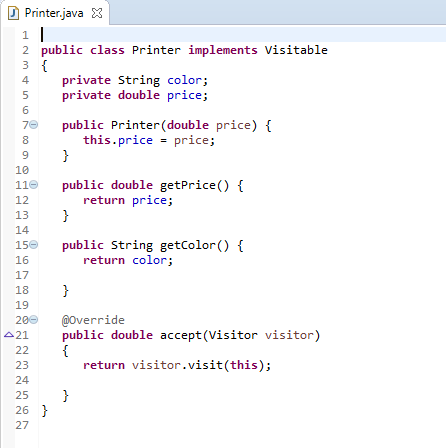


Client

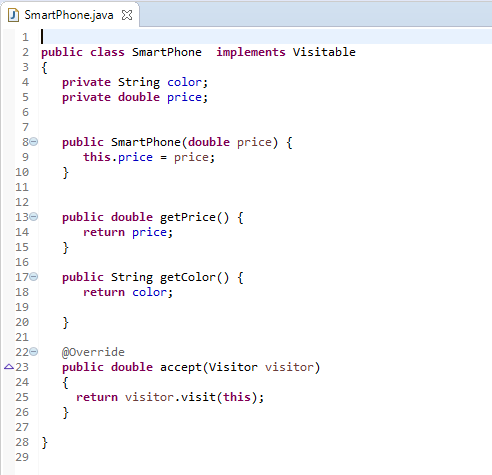
These are the elements:



NoteBook



Printer



Smartphone

