Yu Hu
05/02/2017

Final Project: Visitor pattern

The paper studies a behavior design pattern - visitor pattern. It allows addition of new operations to extant object structures without modifying the structure. In an application using Visitor pattern, there's a client that contains visitable elements. When the client calls an operation, the visitable element accepts a visitor and perform certain operation. By using visitor pattern, the algorithm is separated from the object structure.

Visitor pattern uses double dispatch. Double dispatch is a mechanism that dispatches a function call to different concrete functions depending on the runtime type of the two objects involved in the call. In other words, double dispatch calls concrete function based on the type of concrete Visitor and the type of the concrete Visitable element. The UML diagram of visitor pattern is shown in Fig.1.

The motivation of using a visitor pattern is that double dispatch structure allows addition of new operations without "polluting" the original object structure. Many distinct and unrelated operations need to be performed on objects in a heterogeneous aggregate structure. Visitor pattern avoids casting the pointer to the correct type before performing the desired operation. It figures out the type of object and the type of visitor at run time (Design). Assume there's an app that generate report for every employees in the company. There are two type of Employees: HourlyEmployee and LongTermEmployee. If single dispatch structure is used, both concrete classes will extends an abstract Employee class that contains generateReport() method. There are three situations that needs to be considered. First, the app needs to cast the Employee object to the correct type before calling generateReport() method. Second, every time a new operation is added, says createHTMLReport(), the programmer needs to add createHTMLReport() to the abstract Employee class and implements the newly-added methods in every sub-classes. Lastly, when the format of the report changes, the programmer needs to go into every classes and modify each generateReport() method.

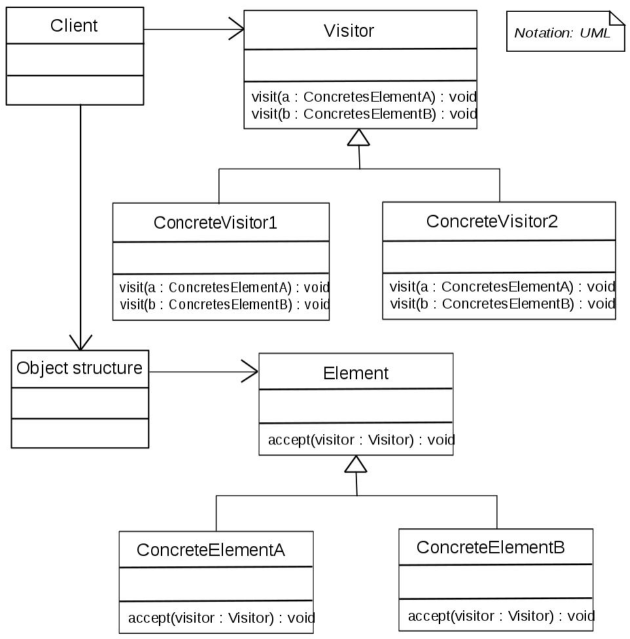

Fig. 1. UML diagram of Visitor pattern.

In the Employee example, single dispatch design violates the single responsibility principle. According to Robert C. Martin, “A class should have only one reason to change” (Martin). The single responsibility principle states that every class should have responsibility over a single part of a functionality required for a software and the responsibility should be entirely encapsulated by the class. The concrete Employee classes should not be responsible for the format of the report. Therefore, changing the report format inside the concrete Employee class violates the single responsibility principle.

By contrast, the double dispatch visitor pattern allows more flexible addition of new operations and modification to previous operation. This paper uses a Shopping cart app to demonstrate the Visitor pattern. The UML diagram of the Shopping cart app is shown in Fig. 2. The shopping cart stores all the shopping items the user selected. It calculates the total prices after the discounts of all the items are taken into consideration. Also, it calculates the shipping price. For every concrete shopping element class, it should be responsible for storing basic information such as name, price, production date. However, it should not store information such as discount or shipping cost which is subjected to change. Discount, for example, varies due to changes such as seasons, market demand, and trends. Retail business encourages consumption by having more discount for buying greater amount. Therefore, the shopping item class should not be responsible for calculating discount or postage.

Visitor pattern separates the operations that calculate discount and postage from the shopping items. Two concrete visitor classes ShippingVisitor and DiscountVisitor are used to implement the specific calculation algorithm. The client is a shopping cart that contains shopping items which are visitable elements. The concrete visitable elements are Book, Clothing, Snack and Fruit. The advantage of visitor pattern in the shopping cart app is that it is adaptive to changes. In real life, the marketing strategy changes rapidly. The discount and shipping strategies for shopping items change frequently. Visitor classes allows more versatile adaptation of different algorithms.

In the DiscountVisitor class, discount for each Concrete element is defined separately. Book has 50 percent off discount for special offer. Fruit and Snack are time sensitive, so they have 80 percent of discount for those that are going to expire soon. Demand for clothes vary due to seasonal change. Hence, Clothing has special offer depending on the seasons: in the summer, winter clothes are 50 percent off, vice versa. Other clothing items are always 20 percent of, a common strategy noticed in the clothing industry.

In the ShippingVisitor class, shipping cost varies depends on how much the consumer spends. Like Amazon, user that buys more than 35 dollars items enjoys free shipping. Otherwise, shipping for books and fruits depends on the weight of the item. Shipping for snack is free to encourage consumption. Shipping for Clothings is 5 dollar per clothing item.

In the client class Cart, a HashSet of Visitable elements are stored. When calculateTotalPrice() and getShippingCost() is called, the program decides during runtime the type of concrete visitable element and of concrete visitor element. Visitor pattern is useful to recover lost type information. Though items in the shopping cart are of different types, the program does not need to cast the object to the correct type.

There is limitation with visitor pattern. The return type of the visiting methods need to be known in advance. Even though visiting pattern can add new operations by adding new visitor classes, new visiting methods of different return type are excluded. In the case of the shopping cart cart app, all the visiting methods need to return type double. Also, visiting method doesn't allow parameter.

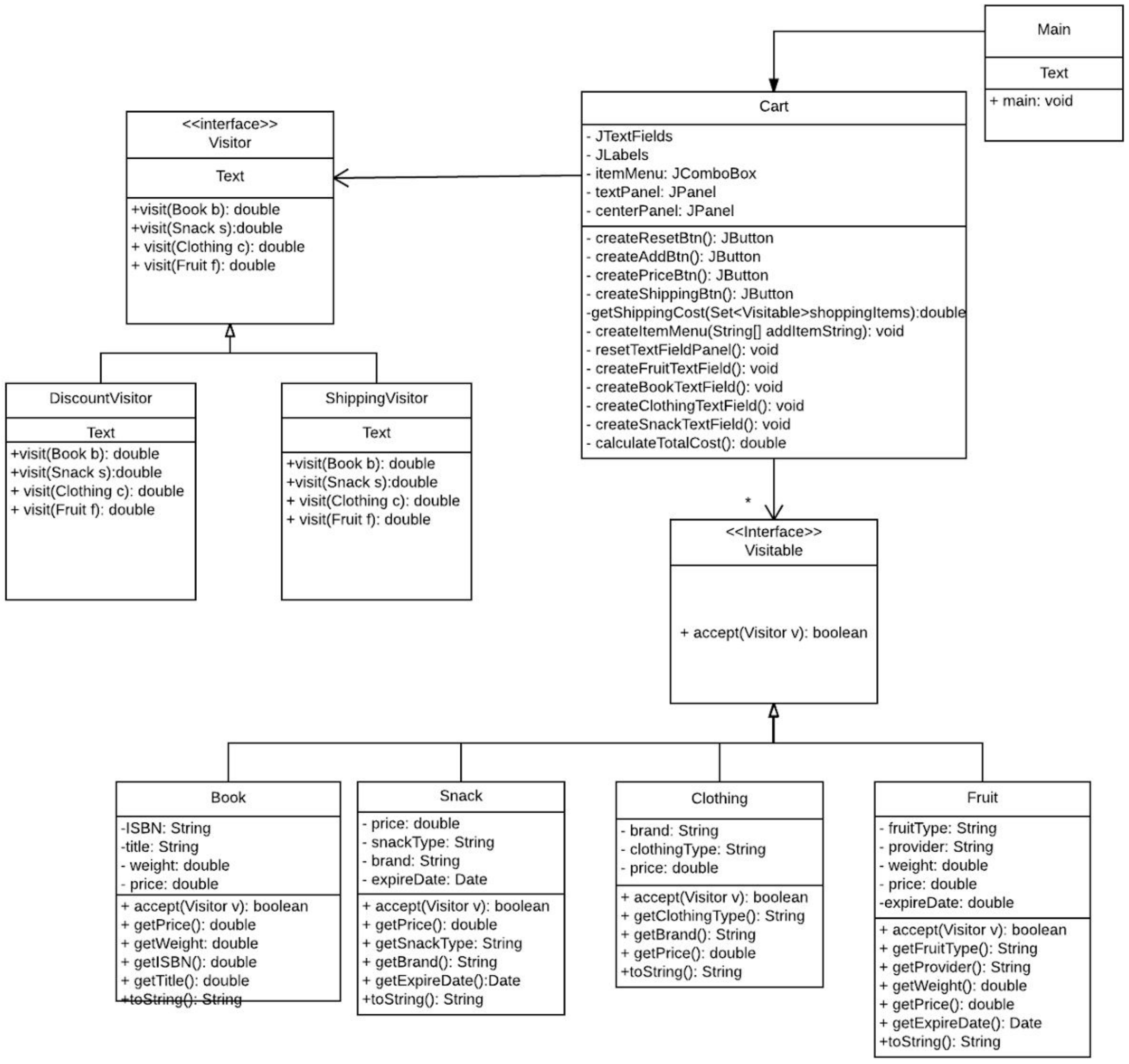

Figure 2. UML diagram of shopping cart app.

Work Cited

Martin, Robert C. (2003). *Agile Software Development, Principles, Patterns, and Practices*. Prentice Hall. p. 95. ISBN 978-0135974445.

Design Pattern Simply Explained. N.p.: n.p., n.d. *Sourcemaking*. Web. 1 May 2017.