# Visitor Design Pattern

Problem: It’s difficult to set the Postage prices of items of different types especially when the price depends on weight for regular prices and discounted prices.

Solution: It can be solved by moving the action to visitor that is need to be performed on many objects of various types. In the given example, the PostageVisitor and the DiscountPostageVisitor has the methods to calculate the regular and discounted prices of the items that belongs to Fragile, Parcel or Envelope category. Fragile, Parcel and Envelope has accept method with visitor as argument and calls visit method of visitor and passes itself as argument which satisfies the requirement of double dispatch. In this way we can separate the logic from an object structure on which it operates. As a result, we can add new operations to existing subject structure without modifying the structure.

# Decorator Pattern

Problem: When a combination of new functionalities need to be added independently on the top of main functionality, it’s difficult to create new class with every possible combination of new functionalities. For example; a froyo (Frozen Yogurt) that can be made with simple froyo plus different toppings on it will require a new class for every combination of toppings on simple froyo.

Solution: For the given example, SimpleFroyo is the main object and different types of topping are different functionalities to be added on the top which does not affect the behavior of the SimpleFroyo. The object can be decorated by nesting different decorators (toppings) within each other as they all conform to the same interface (FroyoDecorator) and forward their messages to the main object (SimpleFroyo). This is an effective way to extend functionality which does not affect the behavior of the original object instead of subclassing.