**1. Introduction**This is an example of solution refactoring. The original solution contained three classes (Order, Line, Bike) used by a bicycle distributor to produce order receipts.

**2. Refactoring**As refactored, the solution has been enabled to process sales orders for any product item (not just bikes) in a manner such that it can accommodate ever-changing business requirements with regard to both discount issuance post-sale transaction order handling (e.g. receipting).  
  
Programmatically, this has been addressed via the application of certain OOP principles (abstraction, encapsulation, the introduction of certain polymorphic behavior) along with dependency injection over a layered architecture.

**2.1 Dynamic order calculation**By nature of business, the calculation of a sales order is derivative of the following variables:

a. number of line items;

b. price and quantity per each line;

c. discount possibilities at the product (line item) level;

e. discount possibilities at the aggregate (sales order) level;

f. tax rate;

Order calculation now accommodates the dynamic, varying nature of a-f above.

**2.2 Dynamic discount detection**The original domain model composition has been abstracted and extended as follows:

a. An ***ISalesOrder*** type encapsulates (*has,* contains) a collection of ***ILineItem*** types.

b. Each ***ILineItem*** type *has* a ***IProductInfo*** type and additional properties describing the sales transaction (price, quantity, discount amount, line sales total etc.);

c. Each product ***IProductInfo*** type contains properties describing the product (name, brand, model, product discount codes, etc.);

d. At the “aggregate” (i.e. non-line item level), a sales order can also contain a variety of other “data” points such as information about the customer, order dates, sub totals, tax rates, information about shipping / delivery etc.

The reality of the sales process is such that discounts are advertised / displayed at the point of sale level (pre-sales order completion). Discount issuance is a business policy driven event subject to a variety of factors which may compel sellers to introduce ever changing discount mechanisms in order to compete and “move” inventory.

The solution has been enabled to handle an unlimited amount of such factors vis-à-vis the ability to “attach” and detect discounts associated with any attribute of either the product, the line item or the sales order itself.

At order computation time, a ***IOrderManager*** type is charged with order processing. This includes (a) order calculation and (b) order and post-sale order handling which may include any sort of behavior such as receipting, printing, emailing, etc.   
  
Because discounts are material to order calculation, at order calculation time, a collection of ***IDiscountScanner*** types are injected into the ***IOrderManager*** type. ***IDscountScanner*** implementers encapsulate varying discount issuance logic based on business policy. As such, they are responsible for “scanning” each sales order at the appropriate levels (both the line item and the aggregate level) and initializing each level with the appropriate discount coefficients. This allows order calculation to occur in a manner that it can accommodate the ever-changing needs to issue discounts based on any business requirement.

**2.3 Post-sale transaction order handling:**   
The creation of a receipt is typically subject to a variety of drivers (business policy, point of sale customer requirements, etc.). At present, receipt generation and delivery has been handled in a polymorphic manner as named implementers of the ***IReceiptManager*** GetReciept() function are allowed to behave in a varying manner and injected dynamically based on the specific functionality desired by their caller. This varying behavior can include formatting (string v html), form of submittal (e-mail v. sms) etc. and further offers handling ability for any business object part of the sales transaction that may need to be handled post-sale.

**3. For further consideration:**   
Due to the test nature of the solution, the above functionality may not have been addressed optimally: In a real-world scenario, a variety of business process requirements specific to industry / individual client must be obtained. This solution is merely a demonstration of process analysis within the context of a programmatic (architectural) solution to a specific business problem with general design-time requirements.

Thank you for looking!

**GalenaHill**

/ \_A SaaS BI outfitter for the Man in the Arena\_ /

[galenahill.com](http://www.galenahill.com)