Paper 4M21: OBJECT ORIENTED SOFTWARE DESIGN

OBJECT ORIENTED SOFTWARE DESIGN

Crib for Examples Paper 2

1. There are six classes shown in the diagram. They are:

   (a) Web Purchasing Application

   (b) Order - which provides four operations:

      i. add_item which takes an argument of type Item

      ii. remove_item which also takes an argument of type Item

      iii. print_delivery_order

      iv. charge_credit_card

   (c) Customer - which has three attributes (name, address and credit_card_no) and provides the operation parse_web_form.

   (d) Shopping basket which provides three operations:

      i. add_item which takes an argument of type Item

      ii. remove_item which also takes an argument of type Item

      iii. checkout

   (e) Item which has the attribute num_ordered and provides two operations, description and price.

   (f) Book which provides three operations, num_available, title and price.

   There are seven relationships shown in the UML diagram. These show that

   (a) The Web Purchasing Application contains 0 or more Customers.

   (b) The Web Purchasing Application also contains 0 or more Books.

   (c) The Web Purchasing Application also contains 0 or more Orders

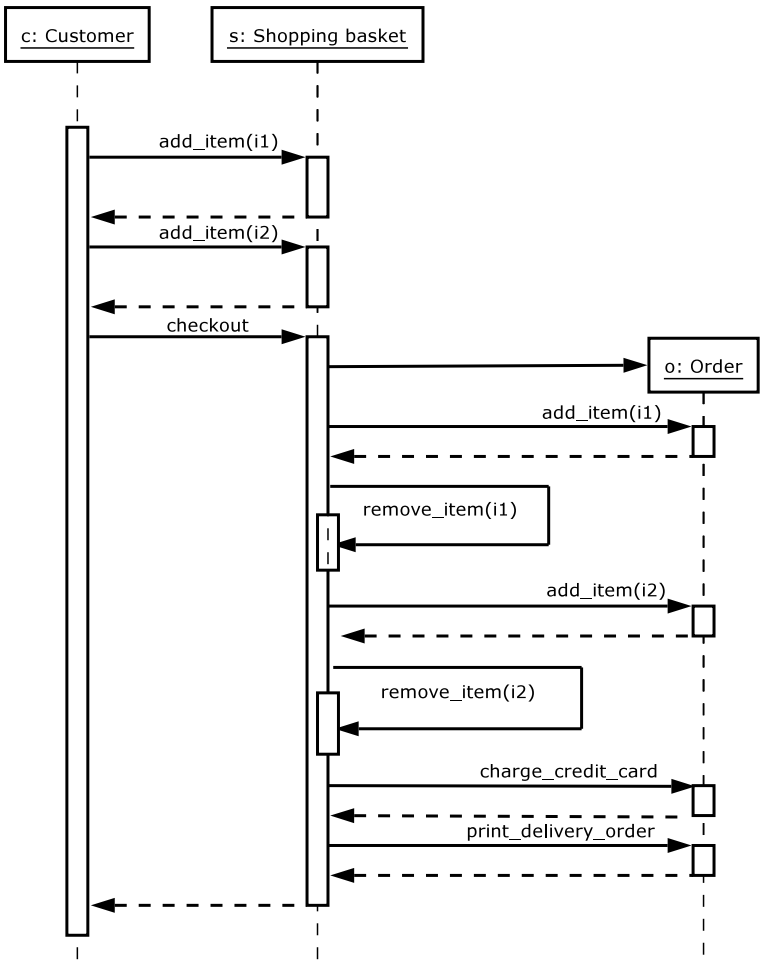
   (d) the arrow between Order and Customer shows navigability between objects of these classes (i.e. each Order has a reference (or pointer) to exactly one Customer and each Customer has a container of references (or pointers) to 0 or more Orders).

   (e) Each Customer has exactly one Shopping basket, and each Shopping basket has reference to exactly one Customer.

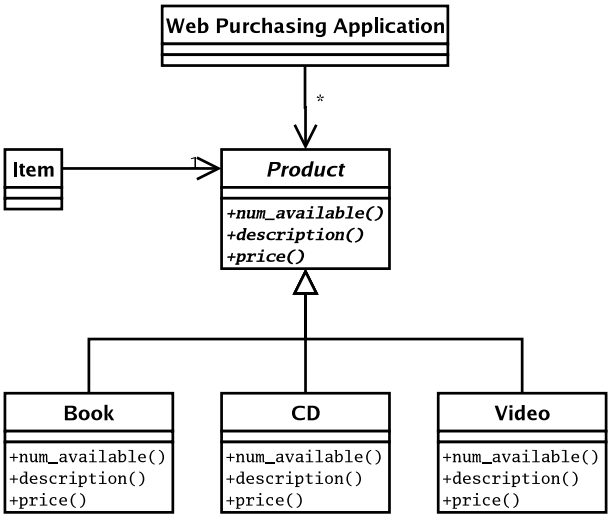
   (f) Each Shopping basket has 0 or more items (the items can be added to or removed from the shopping basket).

   (g) Each Item refers to exactly one Book.

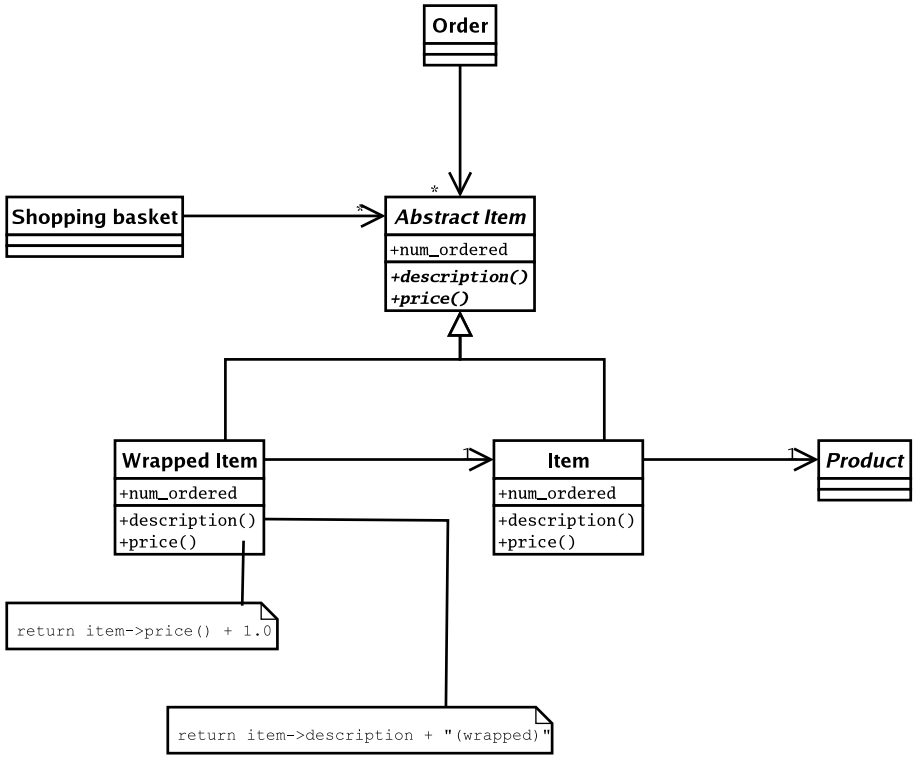
2.



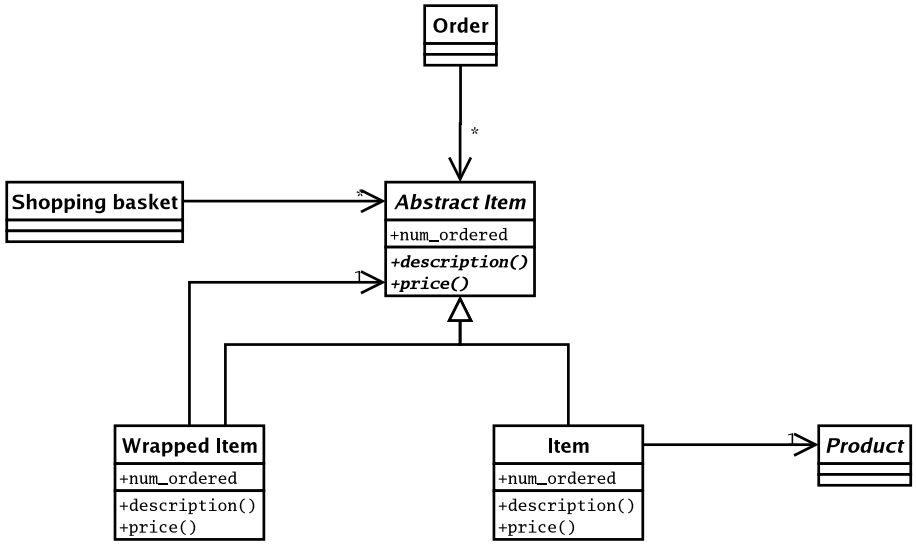
3. This requires creating a class hierarchy to replace Book which contains all the various types of product. In the diagram shown below, the title() function has been replaced by the more generic description().



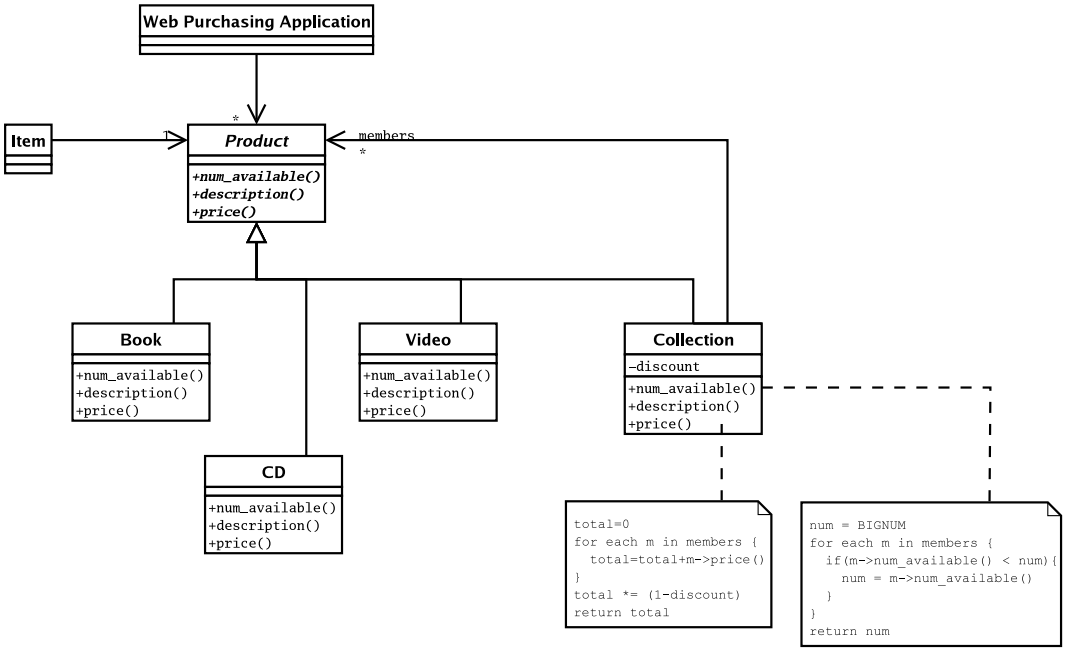
4. The decorator pattern should be applied to Item which is illustrated in the diagram shown below. (One instance where decorating Product might be better is if we only wished to offer wrapped varieties of very few particular products).



Please note that the above is not a traditional form of a decorator pattern! The pattern could also be implemented using the more traditional form of decorator as shown below - but this would permit doubly wrapping a book which would only really be useful for games of pass the parcel...

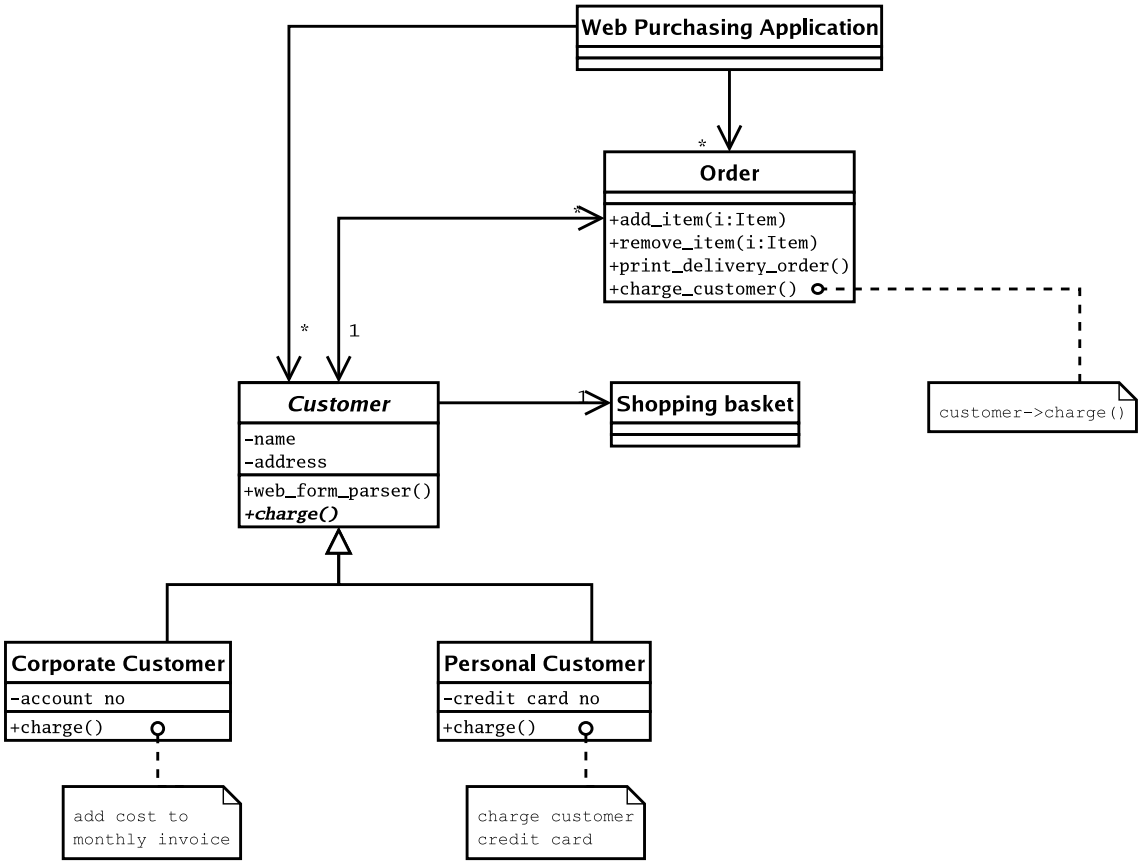


5. In this case the composite pattern should be applied to Product Line (which is the abstract class that replaced Book when we added CDs etc) as illustrated in the diagram below.



6. Two realisations are needed to solve this problem. The first is that Customer has to become a class hierarchy to support different behaviours for Personal Customers and Corporate Customers. Secondly, the charge credit card function is no longer appropriate.

   The function within Order can be changed to charge customer - but this must operate by calling an abstract function within Customer (which is now an abstract base class). This function is then implemented differently within Corporate Customer and Personal Customer either by charging the credit card as before in the case of Personal Customer or by adding the cost to the monthly invoice for the corporate customer.


Web Purchasing Application
*
Order
+add_item(i:Item)
+remove_item(i:Item)
+print_delivery_order()
+charge_customer()
*
*
1
Customer
-name
-address
+web_form_parser()
+charge()
1
Shopping basket
customer->charge()
Corporate Customer
-account no
+charge()
Personal Customer
-credit card no
+charge()
add cost to
monthly invoice
charge customer
credit card