Exercise 6-1

Consider the model in Figure A-27 which describes purchases of a product by customers of a small mail order company The company changes the way it does business to allow customers to walk in off the street and pay cash. No customer needs to be associated with a cash purchase.

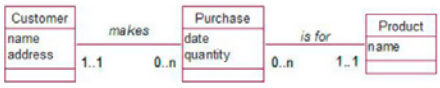

Figure A-27. *Customers purchase products.*

Discuss how effective the following changes to the data would be.

- Change the optionality at the customer end of the relationship to 0 so that not all purchases need a customer.

  This solution solves the immediate problem of a cash sale not requiring a customer. With care, it can be made to work. However, there is the problem of not having a record of the customer when a mail order purchase needs to be invoiced or delivered.

- Leave the optionality as 1 but include a dummy customer object with the name "Cash Customer."

We now have to nominate a customer for the purchase. This is probably a bit safer than the preceding option. However, one of the objectives for the project was to provide statistics on customer habits. Mr. Cash Customer is likely to have a large number of purchases and this could skew the results for statistics such as average value or average number of purchases. It is possible, if you remember, to remove purchases made by "Cash Customer" before doing statistics, but it is a fiddle. It depends on how important these statistics are and how often they are required.

- Create subclasses of `Customer`: `Cash_Customer` and `Account_Customer`.

  This really doesn't achieve anything more than the preceding option and is much more difficult to implement. We are not collecting information about our cash customers, so having a class for them is pointless.

- Create subclasses of `Purchase`: `Cash_Purchase` and `Account_Purchase`.

  We will have many cash purchases (with no customer) and many account purchases (with a mandatory customer), so a class for each is a reasonable idea. It makes deriving statistics for the different customer types simple. It doesn't really have much advantage over the dummy cash customer for the present set of requirements, but if we want to start recording the method of payment (cash, EFTPOS, credit card, etc.) then they will be easy to add. Figure A-28 shows how this can be modeled.

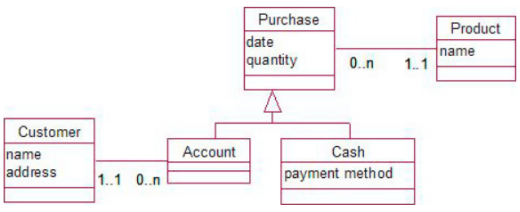

Figure A-28. *A purchase can be for a customer with an account or can be an anonymous cash transaction.*

Exercise 6-2

A farmer keeps information about the application of fertilizer to his crops (e.g., amount, date, etc.). His farm is made up of large sections which are divided into fields. Usually an application of fertilizer is applied to an entire section, but occasionally it is to an individual field. How would you model this?

We can start by having a class called `Application` which can record the date and the amount of fertilizer and which is associated with some sort of area—a section or a field. Sections and fields are both "areas" that have something in common (name, size, the fact that fertilizer can be applied, etc.), and so this is a candidate for generalization. Have a look at the model in Figure A-29.

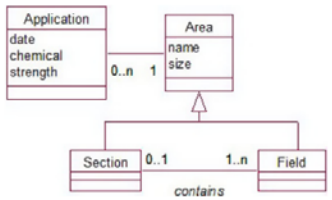

Figure A-29. *Fertilizer applications are made to areas which are either sections or fields.*

In addition to an area being either a section or a field we also have that fields may be one of many in a section. Now we can record that an application has been to a section and know which fields have received the treatment. We can also make an application to a single field if we wish.

Figure A-29 is a good solution; however using the composite pattern, as in Figure A-30, gives a more general solution. We can record an application for an entire farm, which is made up of sections which are made up of fields. We can also record the fertilizer application at any of these levels. This is analogous to the checks on buildings problem discussed in Chapter 6.

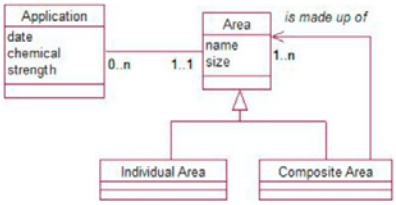

Figure A-30. *The composite pattern allows applications to all levels of area.*

Exercise 6-3

A volunteer library has staff, members, and books. It is necessary to know which books are on loan to whom and how to contact the borrower, and to record fees charged for overdue books. Reference books cannot be borrowed. Members are fined $5 per day for overdue books but staff do not receive fines. How might you model this situation? Some initial classes are shown in Figure A-31.

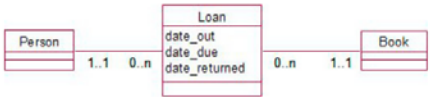

Figure A-31. *People can borrow books.*

We know that we have two types of books (ordinary and reference) and two types of people (staff and members). The important question now is: do we need inheritance? For people, the only difference in the problem stated is in the value of a fine for overdue books. There is no behavioral difference. We could create a `Type` class (with objects for staff and member) which is associated with each person. This class could record the value of the fine for that type ($5.00 for members, $0.00 for staff). Similarly we could categorize each book as being of a type and build into the use cases a rule that reference books cannot be associated with a loan. This different behavior, however, probably warrants considering an inheritance solution. Figure A-32 depicts a possible solution with an association to represent the different types of person and inheritance for the different types of book. Note that we have introduced an abstract `Book` class so that we can make changes to the ordinary and reference book classes independently.

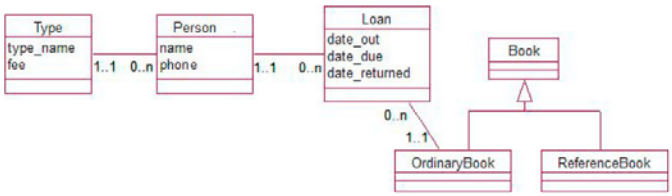

Figure A-32. *One of many possible ways to represent different types of books and people*