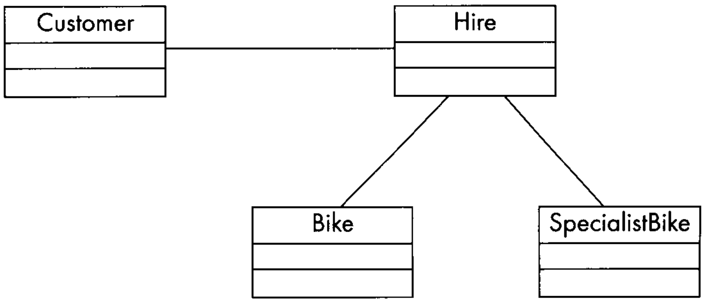

Figure 5.5 *Wheels class diagram with a Hire class included*[a]

a. *We are only guessing about relationships at this stage, we will consider them later in the chapter.*

the dates in Bike we will have to decide how many times a bike is likely to be hired and how many past transactions we need to keep track of. If we get our estimates wrong we will be either wasting storage space or finding that we have to explain to an irate user why their brand new expensive system won't allow a customer who wants to hire 20 bikes for a birthday party to hire more than six bikes.

It is much simpler and less messy to store the dates in a Hire class associated with both the Customer class and the Bike class as in Figure 5.5. Each hire object will contain the attributes relating to only one bike and one customer.

The requirements list in Figure 5.2 indicates that our system is going to have to cope with customers who hire several bikes at once. A father may wish to hire two bikes for his two sons for the week at half-term, and two adult bikes for himself and his wife for the weekend only. We don't want the system to treat this as four separate transactions – see requirement R5 in Figure 5.2. The customer will expect to pay for all four bikes at once and have a single receipt. With the classes we have so far it's hard to see where to put data about payments and an operation to work out totals. The most obvious place is in Customer. However, Wheels have told us they want to store past transactions both for auditing purposes and so that, if a customer wants to hire the same bikes he had last time, we have a record of it (requirement R2). This would mean storing multiple sets of data about payments in Customer. It would be better to have a separate Payment class and store details about financial transactions there, as in Figure 5.6.

Identify the attributes

Many attributes will appear as nouns in the text being analysed. Sometimes we have to make difficult decisions about where to put