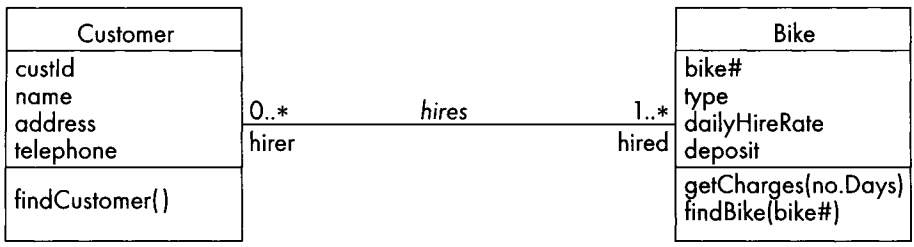

Figure 5.4 Initial class diagram for the Wheels system

Noun analysis gives us a good start, but it is important to remember that it is only a starting point and is very unlikely to produce a complete or satisfactory set of classes. Some objects simply may not appear in the requirements description in the guise of a noun. Iterations and further modelling will uncover other classes or might get rid of some of the classes. Just remember that at this stage the list of candidate classes is not cast in stone; common sense and an understanding of the application domain should never be abandoned.

It is interesting to compare this with our initial attempt at a class diagram in Figure 5.1 reproduced in Figure 5.4 for ease of comparison.

In Figure 5.4 *hires* is shown as a relationship, not a class. Object-oriented system developers are often faced with a decision about whether something is a class or an association; indeed, we have listed it above as one of the criteria for rejecting candidate objects. The reason we have not rejected hire as an object is that there is data associated with it. The requirements list in Figure 5.2 mentions *details of a hire transaction including the start date, estimated duration.* If there is data associated with an object, i.e. details about the object that we need to store for the system to work, then that is a good enough reason to make it a class at this stage.

There is another compelling reason for keeping Hire as a class. Wheels' customers pay for their hires in advance but sometimes keep the bikes longer than they paid for. To calculate how much they owe when they return the bike we need to store both the start date of a hire transaction and its estimated duration. If we store this information in the Customer object we will have to cater for storing multiple sets of dates because a customer may hire several bikes for different durations. If we store the data in the Bike object we might also find we are storing multiple dates as we want to keep a record of past hire transactions. Storing multiple sets of dates is difficult because we have to make messy decisions about how many attributes we need to design into a class. If we decide to store the dates in the Customer class we need to make a decision about the maximum number of bikes a customer is likely to hire. If we store