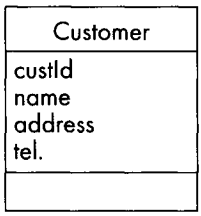

Figure 5.10 *Customer class*

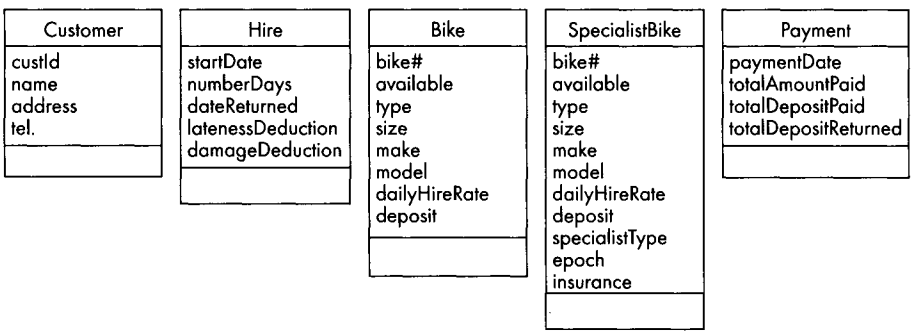

Figure 5.11 *Wheels class diagram with attributes but without operations or relationships*

Payment class. As far as the Payment class is concerned, it would seem sensible to record at least a payment date and the actual payments made: i.e. the total amount of deposit paid and the total amount paid for the hires. As Wheels deduct money from the deposit if a bike is damaged or late, we could also store the amount of deposit returned to the customer. This would give us the classes shown in Figure 5.11. It is not strictly necessary to store any of these totals, as they can all be calculated, but it seems sensible to keep some record of what was actually paid to facilitate any queries in the future.

Identify relationships between the classes

During analysis we have not yet got an exact notion of how objects will need to communicate with each other; the relationships that we include at this stage model real-life relationships that we think might be useful. We will not have an exact idea of the navigable paths we need to build in until after we have done the interaction diagrams (see Chapter 6).

Associations and multiplicity. As we have decided that we need a Hire class and that an object of this class holds data about the hiring of a bike, we will need an association between the Hire class and the Bike class. The multiplicity of the association needs a bit of