Behaviour. So far the concept of an object will seem quite familiar to anyone used to entities in entity-relationship modelling (see Howe, 2001). However, unlike entities, objects don't just store information, they have behaviour. Historically, the reason that objects have behaviour is they were originally used in computer simulations. An aeroplane in a computer simulation stores information: it knows how high it is, how much fuel is in its tank, how fast it is flying. It can also do certain things: it can take off, ascend, fly forward, turn and land. Objects in an information system are not usually simulations in the same way (our system won't have bikes whizzing round the screen) but they do know certain things and they can do certain things. A bike object knows (stores) its type, daily hire rate and deposit. When we design the bike object we decide what it is able to do. The system will certainly need to be able to store, update and display the values of each bike object's attributes. We will also need to be able to work out the total cost of a hire for a given number of days. In object-oriented development, processes that act on related data items are bundled together with the data. In fact, an object is little more than a bundle of data items and processes that manipulate them. These processes are called *operations*. An object's behaviour is divided into operations, each of which represents something the object can do (e.g. update dailyHireRate, display deposit). An object's behaviour is triggered in response to a message sent from another object asking it to perform one of its operations. We discuss messages later in this chapter.

State. Most objects have attributes; e.g. type, dailyHireRate and deposit (see Figure 4.2). These attributes have values, e.g. type = men's, dailyHireRate = £8 and deposit = £50. Normally attribute values can change – the dailyHireRate might go up or we might change the amount we ask customers to leave as a deposit. The state of an object is determined by the values of its attributes and its links to other objects. We can tell, for example, if a bank account object is in the overdrawn state by looking at the value of the attribute balance. The reason we are interested in the state of an object is that its behaviour may vary depending on what state it is in. An obvious example of this is the type of bank account where the account holder is not allowed to overdraw. In this case, how the account responds when a request is made to withdraw money depends on how much money is in the account. If there is enough money in the account, the withdrawal is allowed. If not, the withdrawal is not allowed. State and its effect on object behaviour are discussed fully in Chapter 7.