This size of card is used rather than a sheet of A4 paper to encourage system developers to restrict the size of each class in terms of the number of its responsibilities. Good object-oriented design depends on having small cohesive classes, each of which has limited, but well-defined functionality. The CRC card also encourages developers to specify responsibilities at a high level rather than write lots of low-level operations. The general rule is that no class should have more than three or four responsibilities.

One of the most effective ways of using CRC cards is in group role-play. Each member takes the role of an object of one of the classes in the system, and the group then enacts the events that take place during a typical scenario. As each responsibility arising from the scenario is identified, it is allocated to the most suitable object. If a responsibility cannot be fulfilled by the existing objects, a new object (and therefore class) will have to be identified. The aim is to minimize the number and complexity of the messages that need to be passed between the objects, and ultimately to produce classes that have a clear purpose and are internally coherent. This method of using CRC cards is popular with both developers and clients as it tends to promote ideas and facilitate discussions. CRC cards do not involve any special notation and so are easily accessible to clients and users of the system.

Figure 6.2 (repeated from Chapter 3) shows a typical scenario from the 'Issue bike' use case. In this figure, where we identify a responsibility of the system, we have added the object (in bold, e.g. **:Bike**) that will carry it out. This illustrates how the overall functionality of the 'Issue bike' use case will be divided up between the classes in the system.

Although it does not feature in this scenario, which models only the user's view of the system, we also know that we need a :Hire object to record details about the hire transaction (see requirement R4 in Figure 5.2: record the details of a hire transaction including the start date, estimated duration, customer and bike). Collectively, the objects that interact to execute a use case are known as a collaboration; the collaboration for the 'Issue bike' use case is shown in Figure 6.3.

Software developers find that the simplicity of CRC cards, their lack of detail, make them an ideal tool for exploring alternative ways of dividing responsibilities between classes. It's easy to scrap one design and start again without agonizing about the amount of work that is being discarded. Another way of exploring alternatives is to use sequence diagrams, which are discussed later in this chapter. In practice, however, sequence diagrams show too much detail and are too slow to draw to be useful for this purpose. They are much more useful for documenting the detail of design decisions once these have been reached using CRCs.