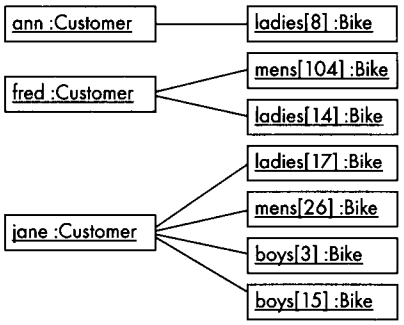

Figure 4.7 *Partial object diagram showing Customer objects and the Bike objects they hired*

understand and because the maintainer does not have to read the entire program to check for dependencies, just the modules affected.

- Unless the interface itself is changed, the internal code of a module can be changed without affecting any client modules. In this sense the internal code of a module (as well as any data) is encapsulated, it is hidden from the rest of the program.

What is a class?

So far our discussion has been entirely about objects and object diagrams. On an object diagram each object is represented by a rectangle, as in Figure 4.7.

Figure 4.7 represents three customers and the bikes they hired; this might model the hire transactions that took place in one hour at the Wheels bike shop. In this diagram we are using simple strings (e.g. ann) for the Customer object names, and array references (e.g. ladies[8]) for the names of the Bike objects.

We can see that, using an object diagram like this, we would very quickly run out of space if we tried to model all the hire transactions that took place in the course of a week; to model all the transactions that took place in a year would be out of the question. The problem with object diagrams is that they take up a lot of space. They are useful for exploring complicated or confused areas of the system, but they are not an efficient way to model the complete set of objects in a system. Wheels have 600 bikes; if they hired them to 5000 customers over the course of a year, we would not attempt to model all of these hires on one object diagram. It is more efficient and convenient to think in terms of classes of objects.