To illustrate polymorphism in action, we will create an object called jeeves of the ButlerRobot class with two legs as his meansOfMobility and English as his language. If we send him the message jeeves.perform() he will respond by answering the door, see Table 4.5. The responses of mart, a :MartianRobot, and pat, a :DogRobot, are also shown in Table 4.5; we have set you an exercise to work out the responses of objects of the rest of the classes, see Exercise 4.11.

To illustrate again how polymorphism works in the code, let us create an array of seven Robot objects, named automaton[7]. We can populate this with objects of all of the instantiable classes in the Robot hierarchy as in Figure 4.31.

We can iterate through the array sending the perform() message to each of the objects in turn (see Figure 4.32); each will respond according to the method implementation of their class.

Reuse

Software developers, unlike their colleagues in most forms of hardware development and indeed most other industries, have never really gone in for component reuse. The most frequently documented reasons for this seem to be as follows.

- Programmers don't seem to like using code written by other people – the 'Not Invented Here' syndrome.
- Problems are caused if components that might be suitable for reuse are written in a different programming language, or for a different hardware platform from that used by the new system.
- There are many libraries of software components both commercial and in-house. However, library components are often either too specific or too general. In both cases they don't quite do what is required. Software libraries are not always well documented and it can be hard to find out exactly what a module does.

The object-oriented approach offers some help.

- Libraries of classes now exist and are widely used.
- The inheritance mechanism allows programmers to tailor library classes to meet the requirements of a new system.
- Often a group of related classes is more useful than a single class as a component for reuse. Classes related by composition form a coherent software unit with a clearly defined public interface.
- Well designed classes and compositions of classes are cohesive and easy to understand; they have a clear and easily identifiable