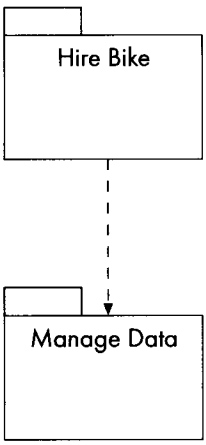

Figure 9.3 *Packages and dependencies*

the package interface can be controlled by judicious use of visibility. If a class is declared to be private, changes to it will only affect parts of the system that can see it, i.e. its own instances. Operations within a public class can be public, private or protected. In the interest of localizing the effect of changes, it is worth limiting the visibility of an operation to the parts of the system that really need to use it.

The importance of identifying dependencies is that it allows us to control the effect of introducing changes; to localize the consequences of changes to a package or a layer and so stop the effect of a change rippling through the system.

Localizing the effect of dependencies is made much easier if, as far as is possible, we can arrange our classes so that dependencies are one way only. In a client–server relationship, the dependency is one way – the client depends on the server. The client needs to know about the server's interface, but not vice versa. If two classes are mutually dependent, i.e. each class is both a client and a server, then both need to know about the other's interface. This makes a much tighter coupling between packages: a change in either class can affect the other.

In a layered architecture, packages are arranged into layers so that each layer only uses the services of the layer below it. In Figure 9.4 classes in layer 6 can only use the services of classes in layer 5, classes in layer 5 can only use the services of classes in layer 4 and so on. In this way the effects of changes are controlled. The only layers that can be directly affected by a change are the layer in which the change is made and the layer above it. If we make a change to layer 1, we know that we need to check to see if it affects itself and layer 2, but only layer 2. We only need worry about layer 3 if we change layer 2, and so on.