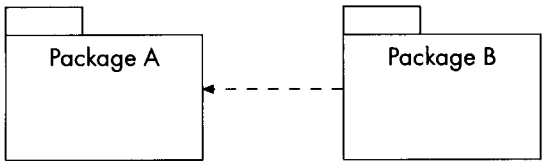

Figure 9.2 *Packages and dependencies*

When we group classes into packages we need some basis for doing so. We want to design packages that are cohesive and independent. One strategy is to group classes into functional subsystems, like the Wheels subsystems described above. Another strategy is to group classes by type, e.g. a group of the interface classes, a group of control classes, a group of domain classes, and a group of database classes. One advantage of this strategy is that the groups of classes can then be allocated to different programming teams who have skills in particular areas, e.g. interface programming, database programming, etc. Another advantage of grouping classes by type is reuse. For example, a package of interface classes may be suitable for reuse in a different application.

Dependencies. Package diagrams allow us also to specify dependencies between packages. A dependency exists between packages if a change in one can affect the other. If a change in package A can affect package B, then package B depends on package A. A dependency is modelled by a dashed arrow going from the dependent package to the one it depends on, see Figure 9.2.

A dependency exists between two packages, A and B, if a dependency exists between any class in package A and any class in package B. A dependency exists between two classes if, for example, they have a client–server relationship. In a client–server relationship, the client (i.e. the dependant) will be affected by a change to the server's interface. For example, let us suppose that the server changes an operation's signature from findBike(bike#) to findBicycle(bike#), if the client then sends a message that matched the old interface, e.g. bike[36].findBike(), it won't work.

In the example of the Wheels subsystems described above, several classes in the 'Hire Bike' package send messages to the Bike class, i.e. they use the services of the Bike class. The Bike class is defined in the 'Manage Data' package, therefore the 'Hire Bike' package depends on 'Manage Data', see Figure 9.3.

Localizing changes. In a well-designed system, only changes to a class's interface will affect its dependants. A package interface consists of the public operations of its public classes. The size of