## Introduction

In this chapter we start by looking at the problems associated with traditional approaches to developing software and at how object-orientation addresses these problems. We introduce the object as the fundamental building block of object-oriented software, i.e. as the basis of the software architecture. We briefly summarize why the object-oriented community base their software decomposition on the object rather than on traditional top-down functional decomposition. We explain the concept of a class and the relationship between objects and classes. Finally, we examine the different relationships that may exist between objects and explain what they mean in the context of the developing system.

## Why a new development method was needed

### *Problems with the structured approach to software development*

This section looks at the way in which systems were developed before object-orientation, and at some of the problems that resulted from using the traditional approach.

*Functional decomposition.* For many years, software systems were developed using a structured approach based on functional decomposition. This meant that developers decomposed and then constructed the system according to the main areas of activity – in other words, the subsystems that were identified corresponded directly to tasks that the system had to carry out. For example, in a bike hire system, such as the one used in this book, the system would probably be based on subsystems or processes dealing with issuing a bike, returning a bike, maintaining records for bikes and for customers etc. Each of these processes would perform a separate function, and the data about bikes, transactions and customers would be passed freely between them. In functional decomposition, there was a clear separation between data and process, and data items could frequently be accessed by any part of the program. Figure 4.1 shows data about bikes being transferred between a data store that holds records about bikes and the processes that deal with issuing and returning bikes to customers. This is a potential source of problems, because the bike details are accessible to other parts of the system with no protection from processes that may access and modify them in error.

*Maintenance.* Structured software designed using functional decomposition brought with it big maintenance problems. Changes to the code frequently introduced new bugs; a change in one place