By the time we come to do the interaction diagrams, we have identified what the system will do and documented it in the use cases and use case scenarios. In the class diagram, we have identified the classes of objects that we think we will probably need, plus some of the attributes, and we have had a guess at what the relationships will be. A message from one object to another means that there should be association between the classes to which they belong. This is a useful check that we have got the associations right. If we find that there is no association on the class diagram between objects that need to message each other in an interaction diagram, then we need to amend the class diagram.

For a message from one object to another to work, the target object must understand the message. It can only do this if we have already defined that operation on its class. Interaction diagrams, therefore, act as a check that we have got the operations right.

Interaction diagrams can be shown at different levels of detail depending on when they are used during development. At their most detailed they can serve as comprehensive specifications of the use cases.

We would not expect an interaction diagram to be drawn for every possible scenario of every use case – a representative selection is enough. Generally these interaction diagrams model what normally happens in the successful execution of a use case (often referred to as the 'happy day' scenario) and the main alternative routes through, including ones where the use case goal is not achieved.

## Technical points

*Shading.* On sequence diagrams we can shade activation boxes to give a more precise indication of when an object is *actively processing*, see Figure 6.19. This is different from when it is *active*. When :Customer sends a message to :Payment, it is still active, but it has passed control to :Payment. At this stage :Payment is doing the processing and :Customer, though still active, is just waiting for a response. :Customer cannot do any processing while it is waiting.

Here is a rather more complicated example that takes place when :Payment receives the calcTotalPayment() message.

- :Payment is active as soon as it receives the calcTotalPayment() message
- The calcTotalPayment() operation starts processing
- calcTotalPayment() stops processing when it sends the issueReceipt() message
- issueReceipt() is also an operation on :Payment, so a separate activation box is opened for the length of time that issueReceipt() is executing