Sequence Diagrams

Within a **sequence diagram**, an object is shown as a box at the top of a dashed vertical line (see Figure 5-1).

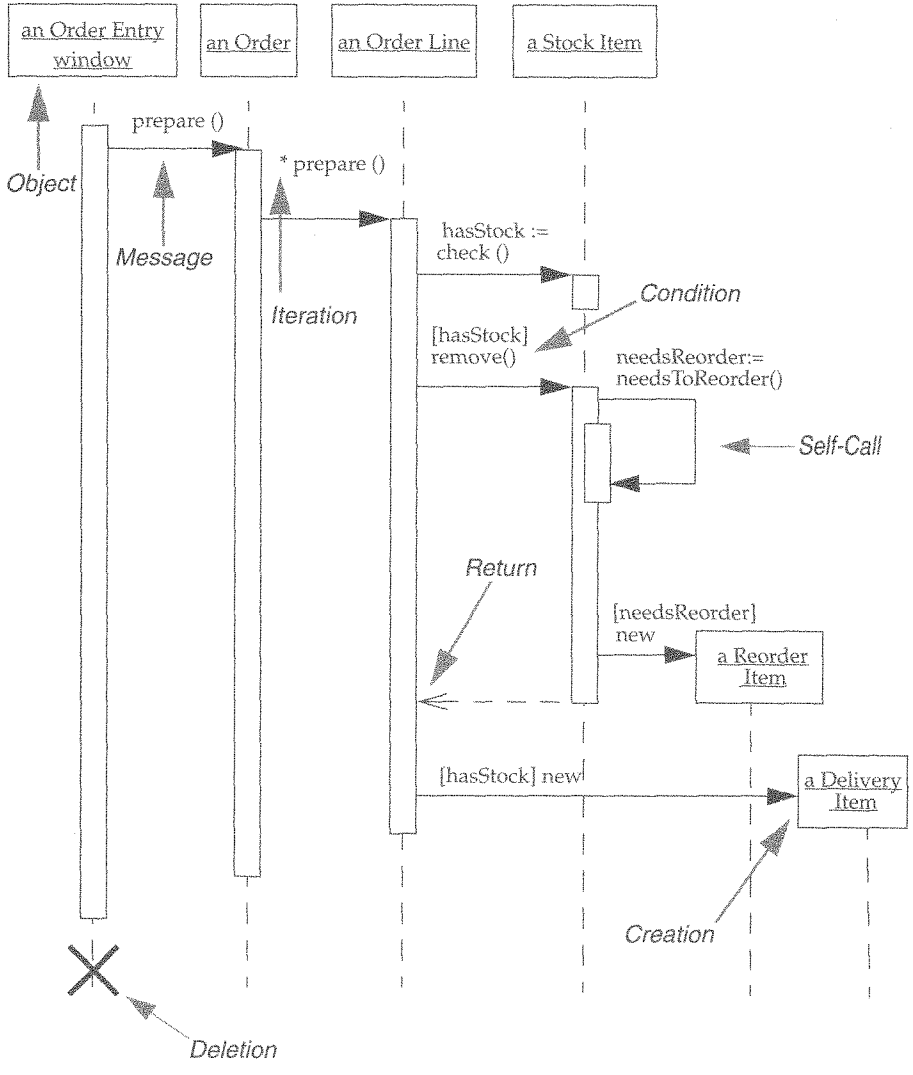

Figure 5-1: *Sequence Diagram*

This vertical line is called the object's **lifeline**. The lifeline represents the object's life during the interaction. This form was first popularized by Jacobson.

Each message is represented by an arrow between the lifelines of two objects. The order in which these messages occur is shown top to bottom on the page. Each message is labeled at minimum with the message name; you can also include the arguments and some control information. You can show a **self-call**, a message that an object sends to itself, by sending the message arrow back to the same lifeline.

To show when an object is active (for a procedural interaction, this would indicate that a procedure is on the stack), you include an activation box. You can omit activation boxes; this makes the diagrams easier to draw, but harder to understand.

Two bits of control information are valuable.

First, there is a **condition**, which indicates when a message is sent (for example, *[needsReorder]*). The message is sent only if the condition is true. Conditions are useful in simple cases like this, but for more complicated cases, I prefer to draw separate sequence diagrams for each case.

The second useful control marker is the **iteration marker**, which shows that a message is sent many times to multiple receiver objects, as would happen when you are iterating over a collection. You can show the basis of the iteration within brackets, such as *[for all order lines].

Figure 5-1 includes a **return**, which indicates a return from a message, not a new message. Returns differ from the regular messages in that the line is dashed. Some people draw a return for every message, but I find that clutters the diagram, so I draw them only when I feel they add clarity. The only reason I used a return in Figure 5-1 is to demonstrate the notation; if you remove the return, I think the diagram remains just as clear. That's a good test.

As you can see, Figure 5-1 is very simple and has immediate visual appeal. This is its great strength.

One of the hardest things to understand in an object-oriented program is the overall flow of control. A good design has lots of small methods in different classes, and at times it can be tricky to figure out the over-