To set a breakpoint on the desired lines, click in the gutter on the left side of the desired line of code. Each time you click the line, the line will cycle through one of three breakpoint settings: none, hard breakpoint, and future breakpoint. A hard breakpoint is denoted by a red \( \text{B} \), and a future breakpoint is denoted by an orange \( \text{F} \). Lines of code that exist outside the body of a function will cycle only through no breakpoints and future breakpoints; code within a function body will cycle through no breakpoints, hard breakpoints, and future breakpoints.

With breakpoints now set, you can try initiating an action that will cause a breakpoint to be encountered. The breakpoints.html page has a single button. Clicking the button calls the \( \text{doFoo} \) method, which does nothing more than create a couple of variables. When the \( \text{doFoo} \) button is clicked in the browser window, the \( \text{doFoo} \) method is called, and the breakpoint on line 15 is encountered. When this breakpoint is encountered, the execution of the JavaScript suspends until further notice. The Venkman window opens in front of the browser window, and the breakpoint is now highlighted, as shown in Figure 7-14.

Now that execution has been suspended, you’re free to perform tasks such as inspecting variable values and viewing the call stack to determine the exact order of execution.

So far, the hard breakpoint has probably worked exactly as you expected it to work. Now, how about the future breakpoint? How can you suspend execution at the future breakpoint since there is no action or event that you control that causes its execution?

Recall that JavaScript code that exists outside the body of a function is executed as soon as the page is loaded. So, to encounter the breakpoint at line 11, you need to switch to the browser window and click the Reload button. Doing so will reload the page and suspend...