```
          commands = null;
        }
        //Now we have commands for the robot.
        int size = cmds.length;
        for (int i=0; i<size; i++)
          processCommand(cmds[i]);        //Move the robot.
      }
    }
```

The only line of code changed is the line at //1—the `if (commands == null)` is changed to a `while` loop. According to this code, each thread that is awakened retests the condition it is waiting on before proceeding. Each thread checks to see whether the `commands` variable is still non-`null` before accessing it. This is to be sure that a thread that was awakened earlier did not change the value of this variable.

Notice that the `InterruptedException` thrown by the `wait` method is ignored. It is typically a bad idea to ignore an exception. (For more on this topic, see PRAXIS 17.) However, this case can be an exception to that rule. You can choose to ignore the exception when the thread is interrupted because with a spin lock, the code will retest the condition. If the condition is not satisfied, the thread will reenter the wait state by calling `wait`. If an `InterruptedException` signals an error in your design, you should not ignore this exception.

Spin locks are a simple and cheap way to ensure proper behavior and execution of code that waits on condition variables. However, if you forget to use them, then your code might work some of the time, but when the timing of the threads is just so, your code will fail. Code that fails occasionally represents some of the more difficult bugs to track down. The utilization of spin locks removes this potential bug.

PRAXIS 55: Use `wait` and `notifyAll` instead of polling loops

Before the advent of more advanced communication mechanisms, programmers relied on other techniques to communicate between different parts of a system. One commonly used technique was the *polling loop*.

A *polling loop* consists of code in one thread that sits in a loop and continually tests a particular condition. The condition the polling thread waits on is ultimately changed by another thread. When the condition is satisfied, the code performs some task. If the condition is not satisfied, the code might sleep for a brief amount