**The application should print out "Mares do eat oats." Is it guaranteed to always do this? If not, why not?**

Although the application did print “Mares do eat oats” in a few trials, it is not guaranteed to always do so, especially when the application is repeatedly running.

It is not guaranteed, firstly because the actual time interval between 2 assignment to variable ‘message’ is not solely determined by the parameter n in Thread.sleep(n). Since the two assignment may cost different amount of time to run, this time difference will accumulate as the number of cycles increases, and eventually the difference will be greater than the previously allocated Thread.sleep(n) time, and the race condition will not be resolved.

There are also other reasons why using sleep is problematic: sleep tends to oversleep. This is because internally it rearrange its priority and yields to other long running processes (thread). Also, we should only use Thread.sleep for its designed use: simulating lengthy operations while testing/debugging on a thread.

**Would it help to change the parameters of the two invocations of Sleep?**

It may help temporarily to increase n, but this only makes the print out to be right for more number of cycles, not guaranteed to be right forever. After more cycles, the same issue may still arise.

**How would you guarantee that all changes to message will be visible in the main thread?**

Instead of using sleep, wait and notify could be used (please refer to the java file ‘BadThreads.java’). This is different from waiting for a fixed amount of time by invoking sleep. Instead, it makes the thread to wait until being notified.

Another way to do this is to literally enforce a sequence of execution within the code itself. For example, we could get a reference to the CorrectorThread instance in the main thread, and make sure to invoke join on that instance before referring to message. In this way, the changes to message will be visible in the main thread.