Assignment 1 report

T-106.5600 Concurrent Programming
Hannu Huhtanen 2014

The reactor pattern is implemented with a Dispatcher that may have arbitrary number of EventHandlers registered to it. Each EventHandler has a handle that produces Events which are passed to the handler which consumes them. Dispatcher makes sure that the consuming of Events happens sequentially while Event production happens asynchronously.This is implemented so that the Dispatcher spawns a separate thread for each registered handler to wait for Handle to produce Events and then add those Events to a queue. At the same time the Dispatcher's main thread retrieves Events one by one from the queue and passes them to the corresponding Handler. Handlers are not designed to work concurrently so the Dispatcher doesn't take another Event before the first one's Handler has finished. The queue used is a blocking queue with finite length. So in case there are a lot more events than what Handlers can handle the production of Events is halted temporarily. In a more common situation where there are no Events waiting the Dispatcher waits until Events come available.

The only object that is shared by multiple threads is the queue implementation BlockingEventQueue and thus it has to be thread safe to guarantee correct function of the system. All other parts of the system are used only in a single thread context. To provide thread safe access to underlying Collection it is always accessed in a synchronized block. Blocking aspect of the queue is implemented so that threads that try to retrieve items from an empty queue will be suspended until there are elements in the queue and same with threads that try to put items into a full queue. After each insertion and removal all sleeping threads are notified so they can check if they are free to continue their operation. This should make the queue totally thread safe. It may be isn't the most efficient implementation of a blocking queue but at least there shouldn't be room for error in this implementation.

All the code has gone through the course provided tests and passed all of them multiple times. In addition of automated tests some amount of manual testing has been done and no errors or bugs have surfaced as a result of either of these. The hangman server implementation is also designed to recover most error inputs or at least provide some what helpful error messages. The project code should work according to specification in all circumstances.

There weren't many problems while completing this project. Most of the problems resulted directly from the limitations set in task specifications. Writing effective and beautiful

concurrent code is hard without the help of java.util.concurrent library. I also wanted to use interrupts to inform Dispatcher's WorkerThreads to terminate but noticed that because the Handle implementations in tests swallowed the interrupts I had to use a different approach. Also based on the specification and the example code I was unsure whose responsibility it is to deregister EventHandler from the Dispatcher when handle produces a null message. I chose to play it safe and in my implementation it is handled by both the Dispatcher and the hangman server. Also especially when using a tool like telnet it is annoying that the server doesn't say anything after establishing a TCP connection. I personally prefer that TCP based protocols start with a server sent message which in this case could have easily been the instructions for registering and playing the game.