Problem 1 Threading

A Monitor is a block of data with some procedures, that can only be accessed by one process at any time. The procedures can range from simple Setters/Getters to complex calculations. Usually, Fields are not directly accessible, but are accessed via the mentioned Setters/Getters. If two Processes try to access any monitored (in Java synchronized) method of the Monitor, the second process is halted until the first completes. This enables one to definitely say "one process executed before another", also known as *happened-before-relationship*

Problem 2 Transparency in Java RMI

Access Yes, after some setup. Once remote objects are received and assigned, one can use them as any local object.

Location Yes and no. No, because one still has to know where to fetch the remote objects from. Once the remote objects are fetched however, they can be accessed without any further location-knowledge about them.

Concurrency Yes, because each procedure call will result in a new thread on the server. Of course the server providing the remote object may hold an internal state which can have problems with race conditions, but RMI itself is concurrent transparent.

Replication

Failure No, RMI will tell the user and/or program that it encountered an error and will leave it to them to handle the problem.

Mobility Yes, if the servers network address is not hardcoded but can be changed (for example with the help of a config-file).

Performance No. The connection to the remote server always needs more time than a local call. However, if the machines performance is by magnitudes worse than the servers performance, this extra networking effort can still be cheaper than a local procedure call. For example: Render a complex 3D-Scene either on a small netbook or on a serverfarm. While the call of the procedure is fast on the netbook itself, the procedure is only fast on the serverfarm.

Scaling Yes, if the underlying algorithm is scalable. As with all types of scalability this is of course a limited yes. Network connection limitations may slow the process flow so much that a proper execution is almost impossible. For small scales there should be however usually no problems.

Problem 3 RMI - single-threaded vs. multi-threaded

Single-Thread The thread has to compute the arguments (total: 5ms), marshalls them (5.5ms), send them (5.7ms), transmit them (10.7ms), the server receives them (10.9ms), demarshalls them (11.4ms), processes them (21.4ms), marshalls the result (21.9ms), sends them (22.1ms), transmit them (27.1ms), the client receives the result (27.3ms) and demarshalls it (27.8ms). The whole thing happens again, for a total of **55.6 ms**.