Lab4 – Java Threads

This lab asks you to experiment with Java threads. This lab will not give you a comprehensive overview, but just some initial exposure to some of the concepts.

In this lab you are asked to do three things, first to analyze a faulty application, then get some familiarity with the debugger by following a netbeans tutorial on debugging multithreaded applications, and finally to use either thread synchronization or joins to solve the problem.

Part 1: A banking problem.

Pete has two accounts with XBank, a savings account and a checking account. The fees on checking account are 10% of the balance, while the interest on the savings account is 10%. Pete will be travelling for a while and won't need either of the accounts, but wants to keep a set amount on the checking account at all time for a case of emergency.

At the beginning of his trip he has a deposit of $1000 on the checking account, and a deposit of $1000 on the savings account. To balance the accounts, Pete scheduled a regular transfer of $100 from the savings to the checking account. Pete reasons that the 10% interest on the savings account will cover exactly the fee.

Upon his return, Pete finds much to his surprise that his accounts fall short of the expected $2000. He call customer service of XBank. The customer representative tells him that they can't find any problem with their banking software; they tested it a few times with the amounts Pete provided, and each time the final combined balance was $2000. The banking software completed three tasks, each by a separate thread. The first thread withdraws the fee from the checking account, the second adds the interest to the savings account, and the final thread transfers $100 from the savings to checking account.

Open the project banking, clean and build it, and run it a few times. Can you reproduce Pete's problem? Analyze and describe the problem, and how to solve it. Use the debugger if necessary. The next part will help with the use of the debugger.

Part 2: Debugging multithreaded applications

Go to http://netbeans.org/kb/docs/java/debug-multithreaded.html and take a look at the second tutorial called Deadlock. Make yourself familiar with the debugging interface, and try to understand how the "synchronized" feature works, and what the root cause of the deadlock is. The project Deadlock is included in the zip-file on Moodle.

Take also a look at http://docs.oracle.com/javase/tutorial/essential/concurrency/syncmeth.html for more information on synchronization.

Furthermore, study the use of the join method, and what it means for two threads to join: http://www.avajava.com/tutorials/lessons/how-do-i-use-threads-join-method.html. Remember that the main method of banking is a thread, too.

Part 3: Solve the banking problem.

See if you can use either the synchronized, or the join method or a combination of both to solve the banking problem, i.e. to ensure that fee, interest and transfer do not interfere. Use as little synchronization as possible. Hint: The final solution is probably much simpler than you expect.