HW #4: Multithreaded Programming

Your boss at J-Bank, an on-line electronic banking company (which will debut in October 2002) is concerned that account holders could make simultaneous deposits or withdrawals from their accounts using multiple computers, and that the integrity of the bank's records could be compromised. To ensure that thearchitecture that is based on Java servlets is safe for concurrent access, you have been asked to write an applet simulation. Your simulation should take an account with an initial balance of $1000 and model three computers trying to make transactions asynchronously. Each computer should be represented by a different Java thread that updates a shared account object. The account object must not only maintain a correct balance, but must also verify that the account balance never drops below $0; furthermore, to support auditing, a list of all transactions must also be retained in a Java Vector. If the account does not have sufficient funds to permit a withdrawal, then that transaction fails and no change is made to the balance. The three computers must each attempt 60 transactions in this fashion:

- Computer #1 only attempts to make deposits. The value of deposit i (where i is between 1 and 60) must be exactly: 50.0 * ((Math.PI * i) - Math.floor(Math.PI * i)).
- Computer #2 only attempts to make withdrawals. The value of withdrawal i must be exactly 60.0 * ((2.0 / 3.0 * Math.PI * i) - Math.floor(2.0 / 3.0 * Math.PI * i)).
- Computer #3 makes some deposits and some withdrawals; if i is odd, the transaction should be a deposit and if i is even the transaction should be a withdrawal. The magnitude of the transaction should be exactly 40.0 * ((4.0 / 5.0 * Math.PI * i) - Math.floor(4.0 / 5.0 * Math.PI * i)).
- Each computer should wait for a specified number of milliseconds between transactions, the number of milliseconds to wait should be specified using a choice menu with values {10, 50, 75, 100, 500}.

In addition to the choice menus, your applet should contain a start button that initializes and begins the simulation, a label that reflects the current account balance, and a scrollable list that contains entries indicating withdrawals that fail. It should be possible to run the simulation more than once (so you may have to reinitialize variables and widgets). After each transaction the balance label should be updated. After all of the threads complete the label should display the correct final balance and the applet should indicate that the simulation has finished.

Grades for the assignment will be based primarily on the correct use of Java's multithreaded concepts and on the clarity of your code which should use good object-oriented techniques. To earn an "A" or "A-" on this assignment you must also display a simple graphical chart indicating the value of the account after each transaction. The chart should be updated after each new transaction and should be rendered reasonably efficiently. An otherwise correct and clear solution without any chart will earn a "B+" grade.

Don't forget to include a legible and correct URL to your applet on the first page of the assignment.