CNT 4714 – Programming Assignment 2 – Summer 2014

Title: "Program Assignment 2: Multi-threaded Programming in Java Using Locks"
Points: 100 points
Due Date: Friday June 6th by 11:59 pm (WebCourses time)

Objectives: To practice programming cooperating, synchronized multiple threads of execution.

Description: In this programming assignment you will simulate the deposits and withdrawals made to a fictitious bank account (I'll let you use my real bank account if you promise to make only deposits! ☺). In this case the deposits and withdrawals will be made by synchronized threads. Synchronization is required for two reasons – (1) mutual exclusion (updates cannot be lost) and (2) because a withdrawal cannot occur if the amount of the withdrawal request is greater than the current balance in the account. This means that access to the account (the shared object) must be synchronized. This application requires cooperation and communication amongst the various threads (cooperating synchronized threads). (In other words, this problem is similar to the producer/consumer problem where there is more than one producer and more than one consumer process active simultaneously.) If a withdrawal thread attempts to withdraw an amount greater than the current balance – then it must block and wait until a deposit has occurred before it can try again. As we covered in the lecture notes, this will require that the deposit threads signal all waiting withdrawal threads whenever a deposit is completed.

1. To keep things relatively simple as well as to see immediate results from a series of transactions (deposits and withdrawals) assume that deposits are made in amounts ranging from $1 to $200 (even dollars only) and withdrawals are made in amounts ranging from $1 to $50 (again, even dollars only).

2. You should have three deposit threads and four withdrawal threads executing simultaneously.

3. Once a deposit thread has executed, put it to sleep for a millisecond or so (depends a little bit on the speed of your system as to how long you will want to sleep the depositor threads - basically we want to ensure a lot more withdrawals than deposits) to allow other threads to execute. This is the only situation in which a deposit thread will block.

4. Once a withdrawal thread has executed, have it yield to another thread (don't put it to sleep though). This will prevent a single withdrawal thread from gaining the CPU and then executing a long sequence of withdrawal operations. Withdrawal threads block if they attempt to withdraw more than the current balance in the account.

5. Assume all threads have the same priority.

6. The output from your program must look reasonably similar to the sample output shown below.

7. Do not put the threads into a counted loop for your simulation. In other words, the `run()` method should be an infinite loop.

8. Do not use the Java synchronized statement. I want you to handle the locking and signaling yourself. No monitors!

References:
Notes: Lecture Notes for Threads Parts 1 and 2.

Restrictions:
Your source files shall begin with comments containing the following information:

```
/* Name:
   Course: CNT 4714 Summer 2014
   Assignment title: Program 2 – Synchronized, Cooperating Threads Under Locking
   Due Date: June , 2014
*/
```

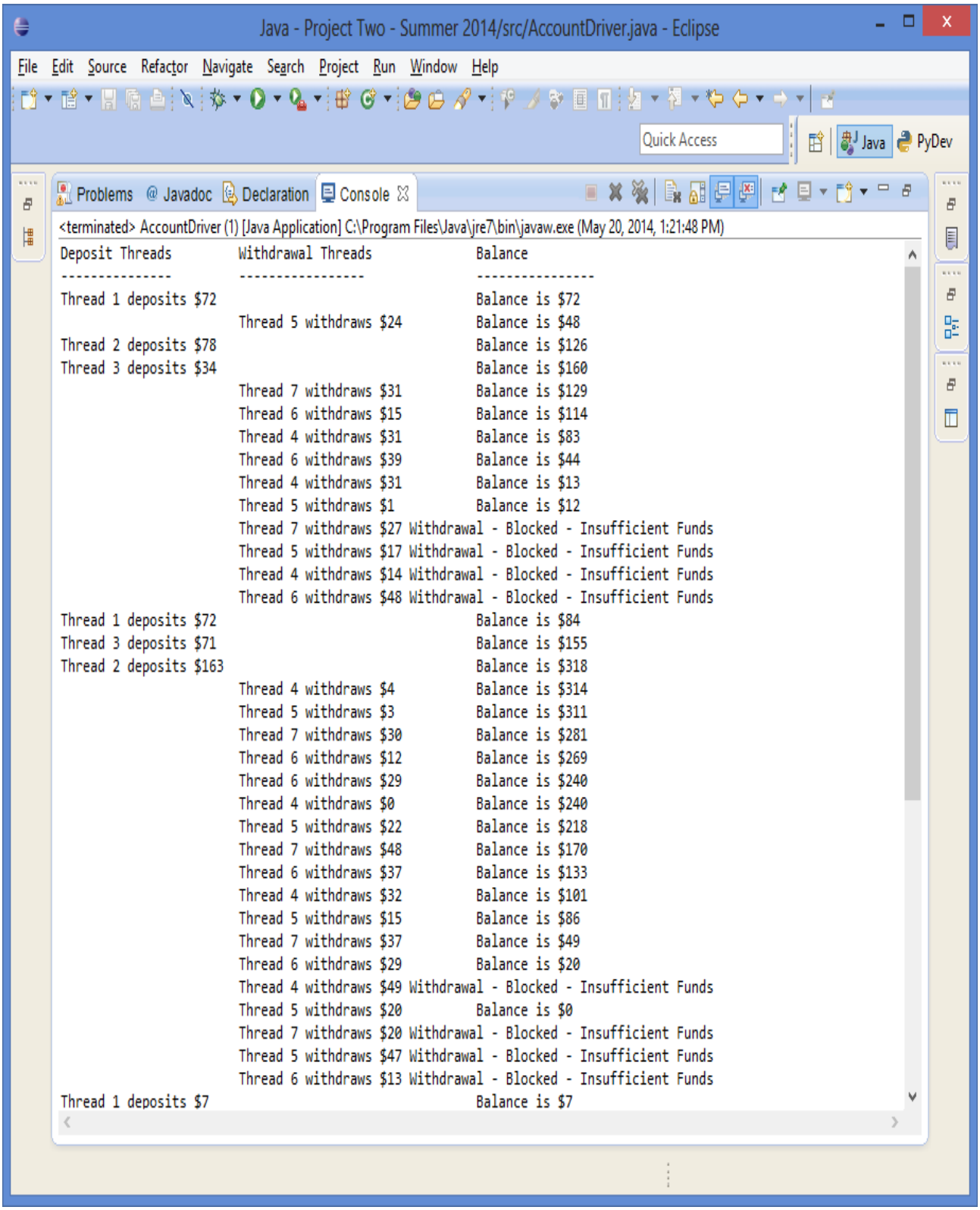
Input Specification: Internal to the program.

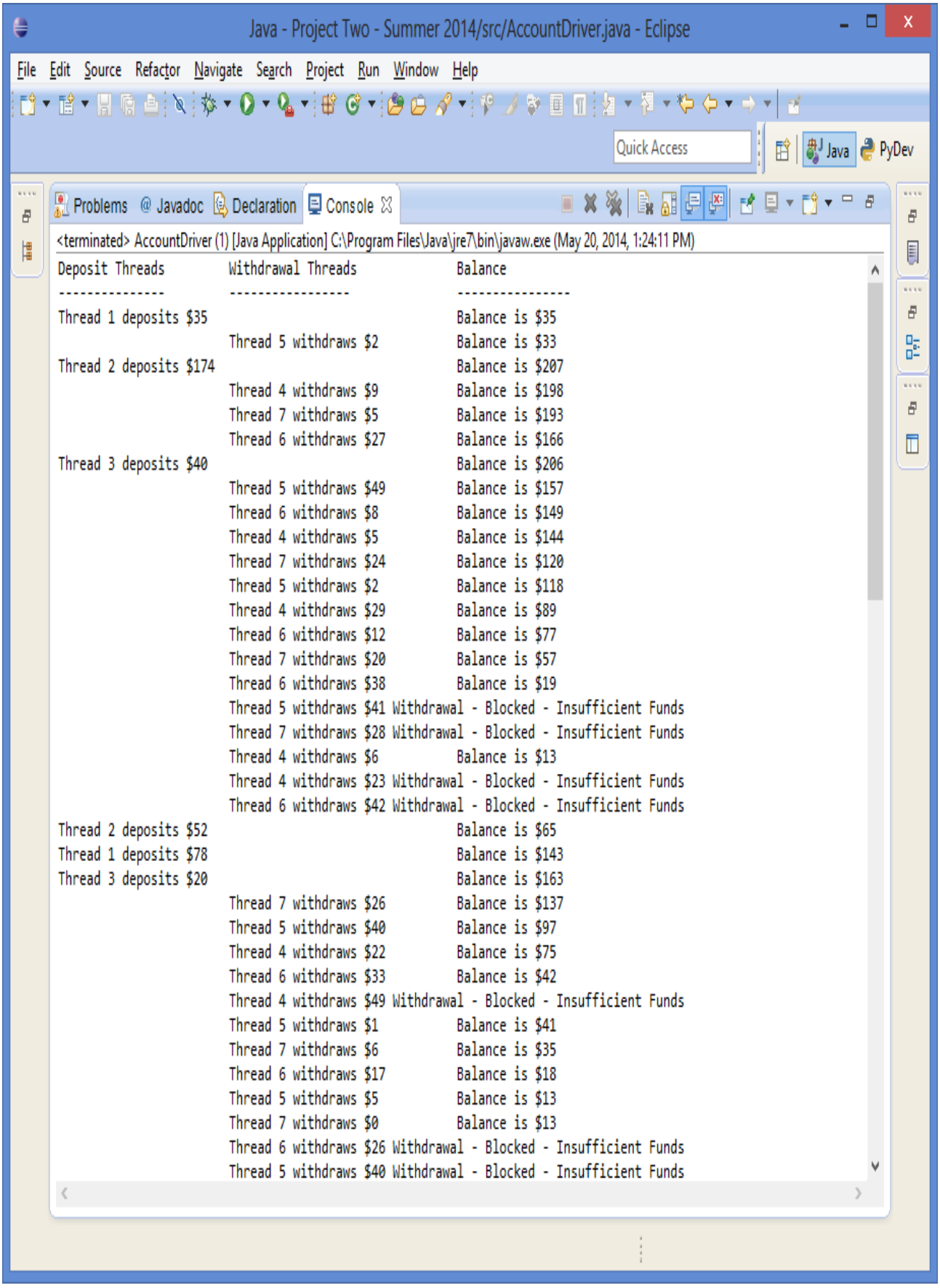
Output Specification: Console based. Your output should appear reasonably similar to the output shown below. You need to be able to demonstrate what the various threads are doing at any moment in terms of the deposits and withdrawals as well as indicating if a thread is blocked.

Deliverables:
(1) Zip up all of your .java files and submit them via WebCourses no later than 11:59pm Friday June 6th.

Additional Information:

Shown below are a couple of example screen shots of the output from this program to help illustrate how your application is to operate and display the results.




Java - Project Two - Summer 2014/src/AccountDriver.java - Eclipse
File Edit Source Refactor Navigate Search Project Run Window Help
Quick Access
Java
PyDev
Problems
Javadoc
Declaration
Console
<terminated> AccountDriver (1) [Java Application] C:\Program Files\Java\jre7\bin\javaw.exe (May 20, 2014, 1:24:11 PM)
Deposit Threads Withdrawal Threads Balance
Thread 1 deposits $35 Balance is $35
Thread 5 withdraws $2 Balance is $33
Thread 2 deposits $174 Balance is $207
Thread 4 withdraws $9 Balance is $198
Thread 7 withdraws $5 Balance is $193
Thread 6 withdraws $27 Balance is $166
Thread 3 deposits $40 Balance is $206
Thread 5 withdraws $49 Balance is $157
Thread 6 withdraws $8 Balance is $149
Thread 4 withdraws $5 Balance is $144
Thread 7 withdraws $24 Balance is $120
Thread 5 withdraws $2 Balance is $118
Thread 4 withdraws $29 Balance is $89
Thread 6 withdraws $12 Balance is $77
Thread 7 withdraws $20 Balance is $57
Thread 6 withdraws $38 Balance is $19
Thread 5 withdraws $41 Withdrawal - Blocked - Insufficient Funds
Thread 7 withdraws $28 Withdrawal - Blocked - Insufficient Funds
Thread 4 withdraws $6 Balance is $13
Thread 4 withdraws $23 Withdrawal - Blocked - Insufficient Funds
Thread 6 withdraws $42 Withdrawal - Blocked - Insufficient Funds
Thread 2 deposits $52 Balance is $65
Thread 1 deposits $78 Balance is $143
Thread 3 deposits $20 Balance is $163
Thread 7 withdraws $26 Balance is $137
Thread 5 withdraws $40 Balance is $97
Thread 4 withdraws $22 Balance is $75
Thread 6 withdraws $33 Balance is $42
Thread 4 withdraws $49 Withdrawal - Blocked - Insufficient Funds
Thread 5 withdraws $1 Balance is $41
Thread 7 withdraws $6 Balance is $35
Thread 6 withdraws $17 Balance is $18
Thread 5 withdraws $5 Balance is $13
Thread 7 withdraws $0 Balance is $13
Thread 6 withdraws $26 Withdrawal - Blocked - Insufficient Funds
Thread 5 withdraws $40 Withdrawal - Blocked - Insufficient Funds