CSci 364
Spring, 2020
Programming Assignment #2 (Bank account)
Due: Feb. 18, 2020
75 points

The Problem
Use Java JDK 8 or newer to write a non-deadlocking application to simulate a bank account with separate deposit and withdrawal threads.

The Assignment
Your program shall read command line arguments that indicate the number of deposit threads, the number of withdrawal threads, the initial account balance (an integer), and the number of milliseconds the main thread must sleep after starting the other threads and before interrupting the other threads.

```
$ java -cp build Main 2 4 500 3000
```

This command line would run the simulation with two deposit threads, four withdrawal threads, and an initial account balance of 500. The main thread would start the deposit and withdrawal threads before sleeping for 3 seconds and then interrupt the other threads.

You must check for valid command line arguments. There must be at least one deposit thread and one withdrawal thread. Also, the initial balance must be greater than 0.

The threads must be java.lang.Runnable objects. The threads must be synchronized to ensure exclusive access to the account by one thread at a time. Also, no thread shall cause the account balance to go below zero or above 2x the initial account balance.

Keep in mind that the critical section for a deposit or withdrawal thread is when it checks and modifies the account balance.

A deposit thread repeatedly picks a random amount to deposit, enters its critical section, and checks the balance. If the balance is ok, the thread deposits the amount.

A withdrawal thread repeatedly picks a random amount to withdraw, enters its critical section, and checks the balance. If the balance is ok, the thread withdraws the amount.

A thread must wait if the account balance does not meet the condition for that thread. Before leaving its critical section, a thread must notify all other threads that it is giving up exclusive access.

The amount of a withdrawal or deposit must be a random number between 10 and 100 (see below) determined at the time of the withdrawal or deposit.

```
int amount = ThreadLocalRandom.current().nextInt(1, 100);
```

If a thread is interrupted, it should catch the InterruptedException and use “break” to exit its loop. After being interrupted each thread shall print its name, the total amount it deposited or withdrew, the number of times it deposited or withdrew, and the number of times it had to wait on its account condition.

A Main or Driver class (its main method) should instantiate and initialize all data objects. After starting each thread (Thread.start()), the main thread should sleep for a number of milliseconds

as specified on the command line. When the main thread wakes up, it should interrupt each thread and wait for the thread to complete. After all threads have been joined, main should print the final account balance.

Write account, deposit runnable, withdraw runnable and main classes in separate Java source code files. All variables shall be private. If you need to access a variable from another class, write an accessor method (getter or setter).

Testing

1) Make sure that your update the balance correctly.
2) Run your simulation several times and ensure that the summary data reported by the threads changes.

Submit to Blackboard

Submit your src/ director (with Java source code files) and your Ant build script using the directory structure below. Your Ant script should have targets to clean and compile the source code. These files should be submitted as an archive.
lastname-firstname.hw2.[zip | .tar | .tar.gz].

```
hw2/
        build.xml
        src/[Java source files]
```