# Sample Solution for Homework 5

## Problem 1 AMP, p. 190: Exercise 93: SimpleReadWriteLock (5 Points)

Please see the Java source code for a sample solution to this problem.

## Problem 2 AMP, p. 190: Exercise 94: ReentrantReadWriteLock (4 Points)

In principle, one could implement this functionality by upgrading the status of the lock-holding thread from a reader to a writer. To do so, the read-write lock implementation would need to do the following. When the write lock is acquired, then the implementation would need to ensure that there is at most one thread that is holding a read lock and if there is such a thread, it is in fact the same thread that is acquiring the write lock. That is, the implementation would need to keep track of the identities of all the read lock holders. This would considerably increase the complexity of the lock implementation, specifically for a reentrant lock where a single thread can hold a read lock multiple times.

## Problem 3 AMP, p. 191: Exercise 95: Savings Account (8 Points)

Parts 1 and 2: Please see the Java source code for a sample solution to this problem.

Part 3: Some of the transfer calls may not return. The problem with the provided implementation is that the transfer method acquires the lock on the calling thread's account and then calls withdraw on the reserve while still holding the lock on the thread's account. If the call to withdraw on the reserve account blocks because its balance is too low, then only the lock on the reserve will be released but not the lock on the account of the calling thread. While this lock is held, any attempts to deposit on the account will be blocked as well. Hence, if two threads mutually transfer from each others accounts, and both accounts have low balance, they will block any deposits on the two accounts and deadlock the transfers.

## Problem 4 AMP, p. 192: Exercise 97: Rooms (8 Points)

Please see the Java source code for a sample solution to this problem.