CSE 332: Data Structures and Parallelism

Exercises (Concurrency)

Directions: *Submit your solutions on **Gradescope**. You must submit a pdf file.*

EX15. Concurrency! (20 points)

(a) [2 Points] Consider this pseudocode for a bank account supporting concurrent access. Assume that `Lock` is a valid locking class, although it is not in Java. If `Lock` is NOT re-entrant, then the code below is broken. Give an example where the `withdraw` method would "block forever". **Refer to line numbers in the code in your answer.**

```
1  class BankAccount {
2    private int balance = 0;
3    private Lock lk = new Lock();
4    int getBalance() {
5      lk.acquire();
6      int ans = balance;
7      lk.release();
8      return ans;
9    }
10   void setBalance(int x) {
11     lk.acquire();
12     balance = x;
13     lk.release();
14   }
15   void withdraw(int amount) {
16     lk.acquire();
17     int b = getBalance();
18     if(amount > b) {
19       lk.release();
20       throw new WithdrawTooLargeException();
21     }
22     setBalance(b – amount);
23     lk.release();
24   }
25 }
```

(b) [4 Points]

Below is a new version of the `withdraw` method your friend proposes. Explain why `newWithdraw` doesn't "block forever" (unlike the original code) even if locks are NOT re-entrant. **You must refer to line numbers in the code in your answer.**

```
1    void newWithdraw(int amount) {
2      lk.acquire();
3      lk.release();
4      int b = getBalance();
5      lk.acquire();
6      if(amount > b) {
7        lk.release();
8        throw new WithdrawTooLargeException();
9      }
10     lk.release();
11     setBalance(b – amount);
12     lk.acquire();
13     lk.release();
14   }
```

(c) [7 Points] Show that `newWithdraw` is incorrect by giving an interleaving of two threads, both calling `newWithdraw`, in which a withdrawal is forgotten/lost. **You must refer to line numbers in the code in your answer.**

(d) [7 Points] (Not related to the bank account example above) Suppose we have a B Tree supporting operations insert and lookup **(deletion is NOT a supported operation on this particular B Tree)**. Consider two options to synchronize threads accessing the tree:

(a) one lock for the entire tree that both operations acquire/release.

(b) one lock per node in the tree. Each operation acquires the locks on all the nodes along the path from the root to the leaf it needs and then at the end releases all these locks once the operation is completed.

Explain which approach is preferrable and why.