**Instructions**

You have up to 2 hour 30 minutes. Try to leave time to attempt each question.

There is some deliberately bad code in the solution. Whenever you see bad code, refactor it to improve it.

Ignore all Stylecop and Resharper warnings. We do not mind if you are used to a different coding style.

If you run out of time on a question make some comments about how you might have solved it or what else you would like to have done.

**Questions**

**Exercise 1 (50 minutes), CurrentAccount:** requires new unit tests to be written

A very simple implementation of a bank account.

1. Change the code so that the failing test “WithdrawlBiggerThanOverDraft\_ShouldThrowException” passes. PASS added logic to check low balance
2. Customers have requested statements that show their previous transactions. Add a mechanism that enables the transaction history.Not attempted – Could have had a data structure which stores transaction history
3. Create a new type of account “SavingsAccount” with the following properties

//Added new class Saving Account

* 1. It can never have an overdraft. DONE

1. It is not possible to withdraw money more than once a month. Not attempted – Could have had a data structure which stores withdrawal history
2. The savings account yields a fixed interest of 2% each year.
   1. Add a method to add on one year’s interest. Ie. Given a balance of $100, that the balance after interest is added is $102. DONE
   2. Add a method that will calculate the bank account balance in X years, assuming no further deposits/withdrawals.

(as a test check that given a balance of $100, that after 10 years you would have a balance of $121.90) Wrote Logic but doesnt work

**Exercise 2 (5 minutes), Contact:** not unit tested, and no new tests required.

The class Contact is not currently thread safe and has some incorrect code. Modify the code to make it suitable for concurrent access.

Changes done – modified lock(this) to lock(\_lock) since lock(this) can be problematic if the instance can be accessed publicly.

**Exercise 3 (10-15 minutes), Algorithm:** no new tests needed

Complete the class algorithm in order to make the test “ShouldReverseEveryOtherWord” pass. PASS – added algorithm in “class Algorithm”

**Exercise 4 (20 minutes), Node/Repository:** no new tests needed

This problem deals with related data. Our server API, IRepository, will only return a node with its immediate children (for example imagine that FakeRepository is loading from a Database. You must not change the code in FakeRepository for this test).

Imagine that the NodeManager is client side code.

1. We want to make sure that there is only ever one instance of a specific object on the client, so that our client has a consistent view of data in our application. The test “LoadingNodeB\_ShouldAlwaysReturnTheSameInstance()” demonstrates this problem. Modify the code in the NodeManager so that this test passes.Could not attempt due to lack of time.But I think singleton patter would do the job of returning one instance.

**Exercise 5 (5 - 10 mins), DifferenceFinder:** no new tests needed.

The code here is overly complicated and slow. Simplify the code and if possible make the test “FindDifferences\_GivenTwoBigLists\_ShouldReturnsDifferencesQuickly” pass.

PASS - Optimised the logic to linq query using except statement.