ETEC3702 – Concurrency
Lab 1 – Simple Communication

Due date: 28 January 2020 by the end of class.

For this lab you are going to write a concurrent program in Python that simulates a bank account. A core function of a system designed to store bank accounts would include a way of storing and retrieving the account balance.

Create a bank account class as follows:

```
class BankAccount(object):
    def __init__(self,initialBalance):
        self.balance=initialBalance
        self.transactionLog=[]
        self.transactionLog.append("initial balance:" +
                                    str(initialBalance))
    def getBalance(self):
        time.sleep(random.uniform(0,0.00001))
        temp=self.balance
        time.sleep(random.uniform(0,0.00001))
        self.transactionLog.append("getBalance:"+str(temp))
        return temp
    def setBalance(self,amount):
        time.sleep(random.uniform(0,0.00001))
        self.balance=amount
        time.sleep(random.uniform(0,0.00001))
        self.transactionLog.append("getBalance:"+str(amount))
```

Part 1: Sequential execution and testing

Add additional methods to the bank account class as follows:

```
def withdraw(self,amount):
     # This method withdraws funds by:
     #      getting the balance using self.getBalance.
     #      subtract the specified amount.
     #      restore the amount using self.setBalance.
     #      log the transaction as: "widthdraw("+str(amount)+")"
    def deposit(self,amount):
     # This method deposits funds by:
     #      getting the balance using self.getBalance.
     #      adding the specified amount.
     #      restore the amount using self.setBalance.
     #      log the transaction as: "deposit("+str(amount)+")"
```

Test this program by creating an account instance with 1000 dollars in it then sequentially calling the following methods:

deposit(500)
withdraw(50)
withdraw(10)

After this, print the transaction log and the final balance, then exit the program.

Execute the program and record the final balance.

Part 2: Analysis

Analyze the operation of the methods from above and answer the following questions assuming that all three methods were executed concurrently.

Answer / complete the following:

a. How many total orderings are there for the three methods' statements.

b. How many distinctly different final balances can result?

c. List all of the final balances identified above.

Part 3: Concurrent execution and testing

Rewrite the program so that it does the following:

- Create a thread for each of the three method calls.
- Start each of the three method threads simultaneously so that all three methods are executed concurrently.
- Wait for all three threads to complete.
- Print out the transaction log and the final balance.

Answer / complete the following:

a. Run your program and observe the output. Record the results of 30 trial runs.

b. Did the output match your analysis from part 2?