COMP 30070 Practical Exam 2017

Attempt Parts 1-4 in sequence. **Submit only one Ruby program finally**. Aim to get each part working correctly before moving on to the next. The bulk of the marks will be awarded for how much you get working, so be sure to submit a working solution, commenting out sections of code if necessary. Marks are also awarded for clean design and coding, formatting and (light) commenting. You may write your program in one file called `main.rb` if you wish, or use several `.rb` files.

Test cases: are not necessary to achieve an A+ grade. Bonus marks will be awarded if test cases are provided.

Git: As explained in the Rules, you must develop your solution under Git source code control, and commit every 10 to 15 minutes.

Quality: The main script may become messy as you proceed to the later questions. Don't worry overly about this, focus on the quality of the code in your classes.

After downloading the input files and expected output at:
http://csserver.ucd.ie/~meloc/oop.zip
**** turn off your wireless network and bluetooth before the exam commences ****

1. An Account comprises an id (an integer) and a balance (a floating point), and provides the methods deposit(amount), withdraw(amount) and to_s. A Bank comprises a number of accounts, and provides methods add_account(id), deposit(id, amt), withdraw(id, amt) and to_s. deposit(id, amt) deposits the given amount into the account with the given id; a similar definition applies to withdraw. In the Bank class, the mapping from id to Account is stored in a hash table.

   Implement the Bank and Account classes as described above and write a main script that instantiates Bank, creates three accounts in this Bank object, makes a deposit and withdrawal to each account, then prints the Bank object using to_s.

   (25 marks)

2. A Transaction comprises an id (the id of the account to which it should be applied) and a non-zero amount. If the amount is positive, it is to be deposited to the specified account, otherwise it is interpreted as a withdrawal. Implement this class.

   The file *accounts.txt* contains a list of account numbers, while *transactions.txt* contains a list of transactions. Extend your main script to instantiate another Bank object, read *accounts.txt*, create the relevant accounts in the Bank object, and then process the transactions from *transactions.txt*. Finally print the Bank object.

   (25 marks)

3. Add the `Enumerable` mixin to the Bank class, so that the `each` iterator yields the accounts in the bank one by one, and accounts are compared based on their

balance. Hence extend your main script to print (1) the largest balance across all accounts, (2) the average account balance (3) the total amount held in the bank.

(20 marks)

4. Create a Teller class. A Teller has a name, and is associated with a Bank. It can process a transaction (delegating detailed processing to Bank), takes a fixed number of minutes process a single transaction and keeps track of how many transactions it has processed. Extend the Bank class to hold a number of tellers.

   Your main script should now create three teller objects named Jack, Emma and Paul, who process a transaction in 1, 2 and 3 minutes respectively. Load the bank with accounts from *accounts.txt*. Read the transactions from *transactions.txt* into an array, which is subsequently passed to the process_transactions_randomly/ smartly methods described below.

   (a) Add a process_transactions_randomly method to Bank that takes an array of transactions, and processes them in sequence, allocating each transaction to a randomly chosen teller for processing.

   (b) Add a process_transactions_smartly method to Bank that takes an array of transactions, processes them in sequence and allocates them in a more intelligent way to the tellers. The goal is to minimise the time it will take the tellers to process the transactions, i.e. to minimise the maximum time any teller spends processing transactions.

   Perform the setup described above, then invoke process_transactions_randomly, then print the state of each teller. Perform the setup described above again, invoke process_transactions_smartly, then print the state of each teller.

   (30 marks)

Submission

1. The folder/directory you submit should contain:
   1.1. All the `.rb` files you wrote, including one called `main.rb` that runs the main script for the program.
   1.2. Your test cases (if any), including `all_tests.rb` that runs all your tests.
   1.3. A file called `statement.txt` containing your name, student ID and a short statement of what you achieved, e.g., "Parts 1-3 completed correctly, Part 4 attempted but unfinished."

2. If you finish early, call an invigilator to supervise your submission. Otherwise, at the end of the exam (1) do a final commit to your Git repository, (2) switch your wireless on, (3) push your local repository to your exam GitHub repository and (4) check your GitHub repository to make sure all your files are there.

3. **Immediately after pushing to GitHub**, zip your working folder/directory and email it to mel.ocinneide@ucd.ie.