Concordia University
comp346 - Summer 2020
Operating Systems
Programming assignment 1

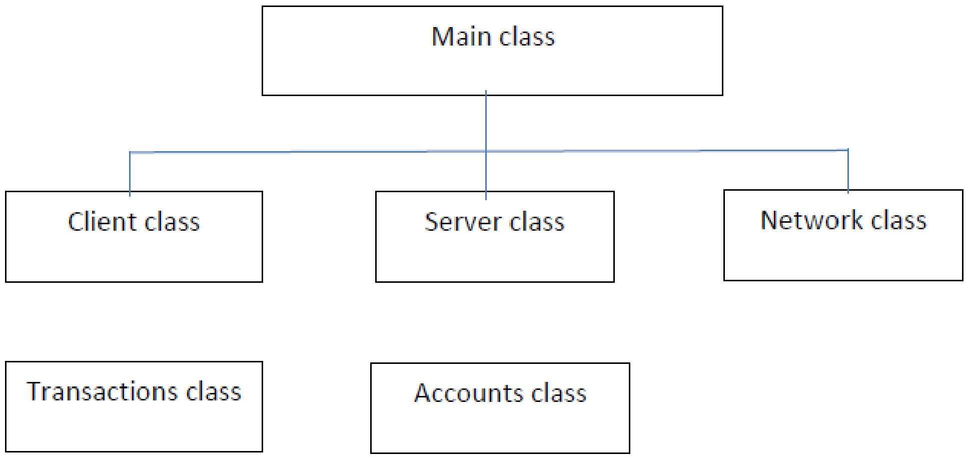
Deadline:	By 11:59pm, Friday July 10 2020
Late Submission:	No late submission
Teams:	The assignment can be done individually or in teams of 2 or 3. Submit only one assignment per team.
Purpose:	The purpose of this assignment is to apply in practice the multi- threading features of the Java programming language.

- **Problem specification.**

In a client-server system, the client application sends requests to a server application through a network connection. In such system, the user interface is implemented in the client and the database is stored in the server.
You are required to implement a client-server application to process banking transactions such as withdrawals and deposits. In modern banking systems, a costumer accesses a bank account using an access card at an ATM, at the counter or on the web.

- **Implementation.**

The following diagram illustrates the classes for the client-server banking application.



Network class
The Network class provides the infrastructure to allow the client and the server to process the transactions. The client and the server need to be connected (using connect()) to the network prior to an exchange. The Network class also implements an input buffer (inComingPacket[]) and an output buffer (outGoingPacket[]) to respectively receive transactions from the client and to return updated transactions to the client. The capacity of these buffers are 10 elements, so the network indicates whether they are full or empty.

Client class
The Client class reads all the transactions from a file (transaction.txt) and saves them in an array (transaction[]). A transaction is implemented by the Transactions.class.

Using the send() method of Network class the client transfers the transactions to the network input buffer and it yields the cpu in case the network input buffer is full.

Also, using the receive() method of Network class the client retrieves the updated transactions from the network output buffer and yields the cpu in case the buffer is empty. Each updated transaction received is displayed immediately on the console.

Server class
The Server class reads all the accounts from a file (account.txt) and saves them in an array (account[]). An account is implemented by the Accounts class.

Using the transferrIn() method of Network class the server retrieves the transactions from the network input buffer and perform the operations (withdraw, deposit, query) on the specific accounts. It yields the cpu in case the buffer is empty.

Each updated transaction is transmitted to the network output buffer using the transferOut() method of Network class and the server yields the cpu in case the buffer is full.

- **Problems.**

You need to complete the Java program that is provided by implementing 4 threads so that the client, the server and the network all run concurrently. The client has 2 threads, one for sending the transactions and another for receiving the completed transactions.

In case the input and output network buffers are full or empty each client or server thread must yield the cpu using the Java method Thread.yield(). The network thread executes an infinite loop that ends when both client and server threads have disconnected. In case the client or sever threads are still connected the network thread must continuously yield the cpu.

You must record the running times of both client threads and the server thread using the Java method System.currentTimeMillis().

You need to provide output test cases with the appropriate running times for the client and the server threads. Perform 3 different runs of the program and explain why there is a difference in the running times.

- **Sample output and test cases.**

  See attached files.

- **Evaluation.**

  You will be evaluated mostly on the implementation of the required methods, the implementation of the threads, the measurements of the running times and the voluntary sharing of the cpu by the threads.

If you miss your demo time, you will receive 0 credit for the assignment.

Evaluation Criteria

Criteria	**Marks**
Implementation of the 4 threads	40%
Implementation of the main class	15%
Implementation of the yield( ) method	15%
Implementation of the measurements of the running times	10%
Output test cases including running times	20%

Required documents.

- Source code in Java
- Output test cases
- Included in the code given, some DEBUG flags in order to help you trace the program but once your program works properly you should put the DEBUG flags in comments.

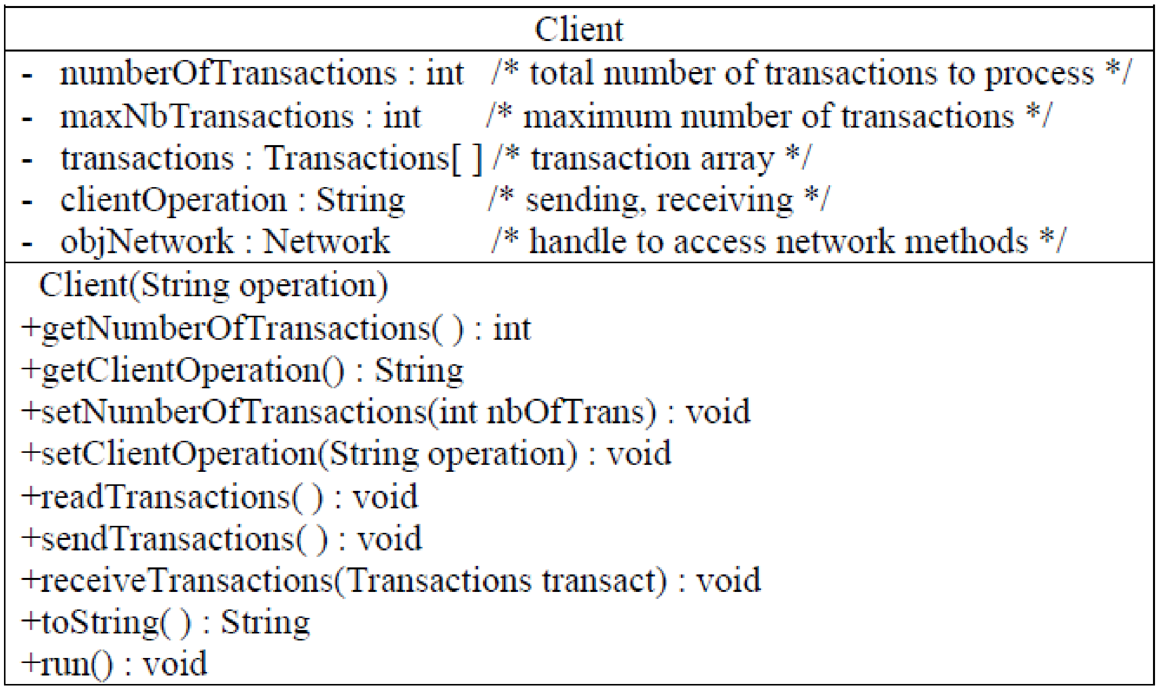
Submission.

Create one zip file, containing the necessary files (.java, .txt and test cases). If the assignment is done individually, your file should be called pa1_studentID,where pa1 is the number of the assignment and studentID is your student IDnumber. If the work is done in a team of 2 or 3 people, the zip file should be called pa1_studentID1_studentID2 or pa1_studentID1_studentID2_studentID3 where studentID1, studentID2, and studentID3 are the studentID numbers of each student.

The zip file should be uploaded to Moodle before the due date. No late submissions are accepted.

Detailed implementation of the classes

- Client class



```
                          Client
- numberOfTransactions : int   /* total number of transactions to process */
- maxNbTransactions : int      /* maximum number of transactions */
- transactions : Transactions[ ] /* transaction array */
- clientOperation : String      /* sending, receiving */
- objNetwork : Network          /* handle to access network methods */
-----------------------------------------------------------------------
 Client(String operation)
+getNumberOfTransactions( ) : int
+getClientOperation() : String
+setNumberOfTransactions(int nbOfTrans) : void
+setClientOperation(String operation) : void
+readTransactions( ) : void
+sendTransactions( ) : void
+receiveTransactions(Transactions transact) : void
+toString( ) : String
+run() : void
```

- Transactions class

Transactions
- accountNumber : String /* account number */ - operationType : String /* deposit, withdrawal, query */ - transactionAmount : double /* transaction amount */ - transactionBalance : double /* updated account balance */ - transactionError : String /* NSF, invalid amount or account, none */ - transactionStatus : String /* pending, sent, received, done */
Transactions( ) + getTransactionType( ) : String + getAccountNumber( ) : String + getTransactionAmount( ) : double + getTransactionBalance( ) : double + getTransactionError( ) : String + getTransactionStatus( ) : String + setAccountNumber(String accNumber) : void + setTransactionType(String opType) : void + setTransactionAmount(double transAmount) : void + setTransactionBalance(double transBalance) : void + setTransactionError(String transError) : void + setTransactionStatus(String transStatus) : void + toString( ) : String

- Server class

Server
-   numberOfTransactions : int /* total number of transactions received */ -   numberOfAccounts : int   /* total number of accounts saved */ -   maxNbAccounts : int   /* maximum number of accounts */ -   transaction : Transactions   /* a transaction to process */ -   objNetwork : Network   / * handle to access network methods */ -   account : Accounts[]   /* account array */
Server( ) +getNumberOfTransactions( ) : int +getNumberOfAccounts( ) : int +getMaxNbAccounts() : int +setNumberOfTransactions(int nbOfTrans) : void +setNumberOfAccounts(int nbOfAcc) : void +setMaxNbAccounts(int nbOfAcc) :void +initializeAccounts( ) : void +findAccount(String accNumber) : int +processTransactions(Transaction trans) : boolean +deposit(int i, double amount) : double +withdraw(int i, double amount) : double +query(int i) : double +toString( ) : String +run() : void

- Accounts class

Accounts	
- accountNumber : String	/* unique account number */
- accountType : String	/* chequing, saving, credit */
- firstName : String	/* first name of account holder */
- lastName : String	/* last name of account holder */
- balance : double	/* account balance */
+ getAccountNumber( ) : String	
+ getAccountType( ) : String	
+ getFirstName( ) : String	
+ getLastname( ) : String	
+ getBalance( ) : double	
+ setAccountNumber(double accNumber) : void	
+ setAccountType(String accType) : void	
+ setFirstName(String fName) : void	
+ setLastname(String lName) : void	
+ setBalance(double bal) : void	
+ toString( ) : String	

- Network class

```
                              Network
-   clientIP : string                          /* IP of client application */
-   serverIP : string                          /* IP of server application */
-   portID : int                               /* port ID of client application */
-   clientConnectionStatus : String            /* connected, disconnected */
-   serverConnectionStatus : String            /* connected, disconnected */
-   maxNbPackets : int                         /* capacity of network buffers */
-   inComingPacket : Transactions[10]          /* network input buffer */
-   outGoingPacket : Transactions[10]          /* network output buffer */
-   inBufferStatus, outBufferStatus : String   /* normal, full, empty */
-   inputIndexClient, inputIndexServer, outputIndexServer,
    outputIndexClient : int                    /* buffer index position */
-   networkStatus : String                     /* active, inactive */
--------------------------------------------------------------------------
  Network(String context)
+ getClientIP( ) : String
+ getServerIP( ) : String
+ getPortID( ) : integer
+ getClientConnectionStatus( ) : String
+ getServerConnectionStatus( ) : String
+ getInBufferStatus( ) : string
+ getOutBufferStatus( ) : string
+ getNetworkStatus( ) : String
+ getInputIndexClient( ) : int
+ getInputIndexServer( ) : int
+ getIOutputIndexClient( ) : int
+ getOutputIndexServer( ) : int
+ setClientIP(String cip) : void
+ setServerIP(String sip) : void
+ setPortID(int pid) : void
+ setClientConnectionStatus(String connectStatus) : void
+ setServerConnectionStatus(String connectStatus) : void
+ setNetworkStatus(String netStatus ) : void
+ setInBufferStatus(String inBufStatus) : void
+ setOutBufferStatus(String outBufStatus) : void
+ setInputIndexClient(int i1) : void
+ setInputIndexServer(int i2) : void
+ setOutputIndexClient(int o2) : void
+ setOutputIndexServer(int o1) : void
+ connect(String IP) : boolean
+ disconnect(String IP) : boolean
+ send(Transactions inPacket ) : boolean
+ receive(Transactions outPacket) : boolean
+ transferOut(Transactions outPacket ) : boolean
+ transferIn(Transactions inPacket) : boolean
+ toString( ) : String
+run() : void
```