Task 1 (70%). Multi-Version Timestamp Ordering Protocol

In this exercise, you will implement a main-memory key-value store with Multi-Version Timestamp Ordering (MVTO) protocol.

MVTO is similar to the MVCC protocol (described in class). The main difference is that write operations are not buffered. Thus, reading an uncommitted value is possible and that could cause cascading aborts.

MVTO Specification:
When a transaction T_i starts, MVTO assigns a timestamp $TS(T_i)$ to it.
For every database object O, MVTO keeps a sequence of versions $<O_1, O_2, \dots, O_n>$. Each version O_k contains three data fields:

- Content: the value of O_k.
- $WTS(O_k)$: the timestamp of the transaction that inserted (or wrote) O_k.
- $RTS(O_k)$: the largest timestamp among all the timestamps of the transactions that have read O_k.

Actions:

- **Reads:** Reads in MVTO always succeed and a transaction never waits as a version of the requested object is returned immediately.
  Specifically, suppose that transaction T_i issues a read(O) operation.
  1) The version O_k that is returned is the one with the largest timestamp **less than or equal** to the timestamp of the transaction T_i ($TS(T_i)$).
  2) If $TS(T_i) > RTS(O_k)$, the $RTS(O_k)$ is updated with the timestamp of the transaction T_i.

- **Writes:** Suppose that transaction T_i issues a write(O) operation.
  1) The version O_k with the largest write timestamp **less than or equal** to $TS(T_i)$ is found.
  2) If $TS(T_i) < RTS(O_k)$, the write is rejected and T_i is rolled back.
  3) If $TS(T_i) \geq RTS(O_k)$ and $TS(T_i) = WTS(O_k)$, the contents of O_k are overwritten.
  4) If $TS(T_i) \geq RTS(O_k)$ and $TS(T_i) > WTS(O_k)$, a new version O_m of O is created, having $RTS(O_m) = WTS(O_m) = TS(T_i)$.

- **Commits:** Processing the commit of transaction T_i is delayed for recoverability, until all transactions (T_j , j != i) that wrote versions read by T_i have successfully committed. If any of the transactions T_j aborts, T_i should abort as well.

Questions:

Your task is to implement the following methods:

1. `void insert(int xact, int key, int value)throws Exception`
   Creates an object with the specified key and the given value for the specified transaction. Throw an exception if the user tries to create an object using a key that exists already.
2. `void write(int xact, int key, int value)throws Exception`
   Updates the value for the object with the given key for the specified transaction. Throw an exception if the user tries to write a non-existing key. In which other cases this method should throw an exception?
3. `int read(int xact, int key) throws Exception`
   Reads the value of the object associated with the given key in the specified transaction. Throw an exception if the user tries to read a non-existing key.
4. `void commit(int xact)throws Exception`
   Commits the given transaction. If needed, it waits for other transactions to commit in a non-blocking manner.
5. `void rollback(int xact)throws Exception`
   Performs the rollback for the given transaction. This method will undo all the operations that have been executed for the given transaction, and performs cascading aborts, if required.

In addition, you have to provide an answer to the following question:

6. Assume a workload consisting mainly of long-running, write-heavy transactions. Which one of Multi-Version Timestamp Ordering (MVTO) and Optimistic Concurrency Control (OCC) supports higher (transaction completion) throughput? Explain briefly (one sentence) why.

As you can infer from the signatures of the above methods, the objects stored in the key-value store are <int,int> key-value pairs.

When an operation is refused by the protocol, throw an exception and abort the transaction, making sure that you undo all the operations performed by the aborted transaction.

Think carefully about all the other cases where an operation should be refused and an exception should be thrown (e.g. when we try to execute an operation in a transaction that is not running).

Garbage collection of versions is not required. You can store all versions.

The interface is given to you in MVTO.java. We will test automatically, so please do not modify the interface. Any changes to the provided interface might cause the automatic tests to fail, which will have an impact on your grade.

We are giving you a sample test file and a shell script to run different tests. You can download them from here. We will use these tests, as well as some additional similar ones to grade your submitted implementations.

Deliverables:

- MVTO.java, containing your implementation.
- MVTO.pdf, containing your answer to Question 6.