**How to run the code**

**How to read the output**

The output has the following format:

<Class message is from>: <message>

example:

Parser:begin(T1)

The parser class printed out the following message, where message is begin(T1) i.e. it read the command begin(T1)

TM: New RWTransaction: T1

Transaction is created with transaction number 1

**Object description**

Parser:

* Parses the input text
* Controls the timestamp
* Sends each command i.e. split by semicolon or newline to the transaction maanger

The transaction manager:

* Receives the command details from the parser
* TODO: format on different inputs
* Acquires locks as needed i.e. with RWTransaction types
* The wait-die protocol kicks in when locks are not acquired, which is stems from conflicting access
* TODO: different actions per inputs
* Wait die. If locks aren’t obtained, the transaction waits if the rejected transaction is older. It is added t o queue transaction. Otherwise, the younger, rejected transaction is aborted.
* A wait for graph is represented by a queue of transaction. Each transaction has information on which transaction it is waiting on. On abort or on end of the transaction causing the wait will be taken out of the queued and set to active transaction and its operations that are in queue will be run.
* Manager validates rw transactions (ro transaction do not need validation). A transaction is valid conditions if it did not access a variable on a site that went down after that sire failed and.

Transaction

* An operation is not sent to a transaction unless the transaction manager has acquired all locks and ensured that there are sites to read from
* Operations requiringlocks will pass locks to the site. Those locks are given by the transaction manager

The DataManager

* Controls access to the sites and knows which the variables are in which sites
* All transactions have access to the data manager

Site

* Site controls access to its variables
* Any events are recorded in the site manager
* It gives locks away to the transaction manager, which will give it to the requesting transaction

Variable

* Each variable has a list of versions

Versions

* Each version

**Command details**

For each command what happends

* Abort. Abort initiates from transaction manager. The transactions are erased and the locks for those are released by the transaction. It notifies the lock managers of each site it has accessed to erase its locks. Lock Manager, TM, Site
* On recovery, the variables that aren’t replicated are set to allow reads and the variable that are replicated reset the pointers pointing to latest committed version and the latest version (that is possibly not committed). The Site and Variable objects.
* On failure the site set all its variables to not allow reads. Its site is set to down. All locks are erased

Test case:

* test if a write to a recovered site sets the variable to allowreads and properly resets the version pointers
* read from T2 should queue and read after recovery?

begin(T1)

begin(T2)

fail(2)

W(T2,x8,88); R(T2,x1)

W(T1, x5,91)

end(T2)

recover(2)

end(T1)