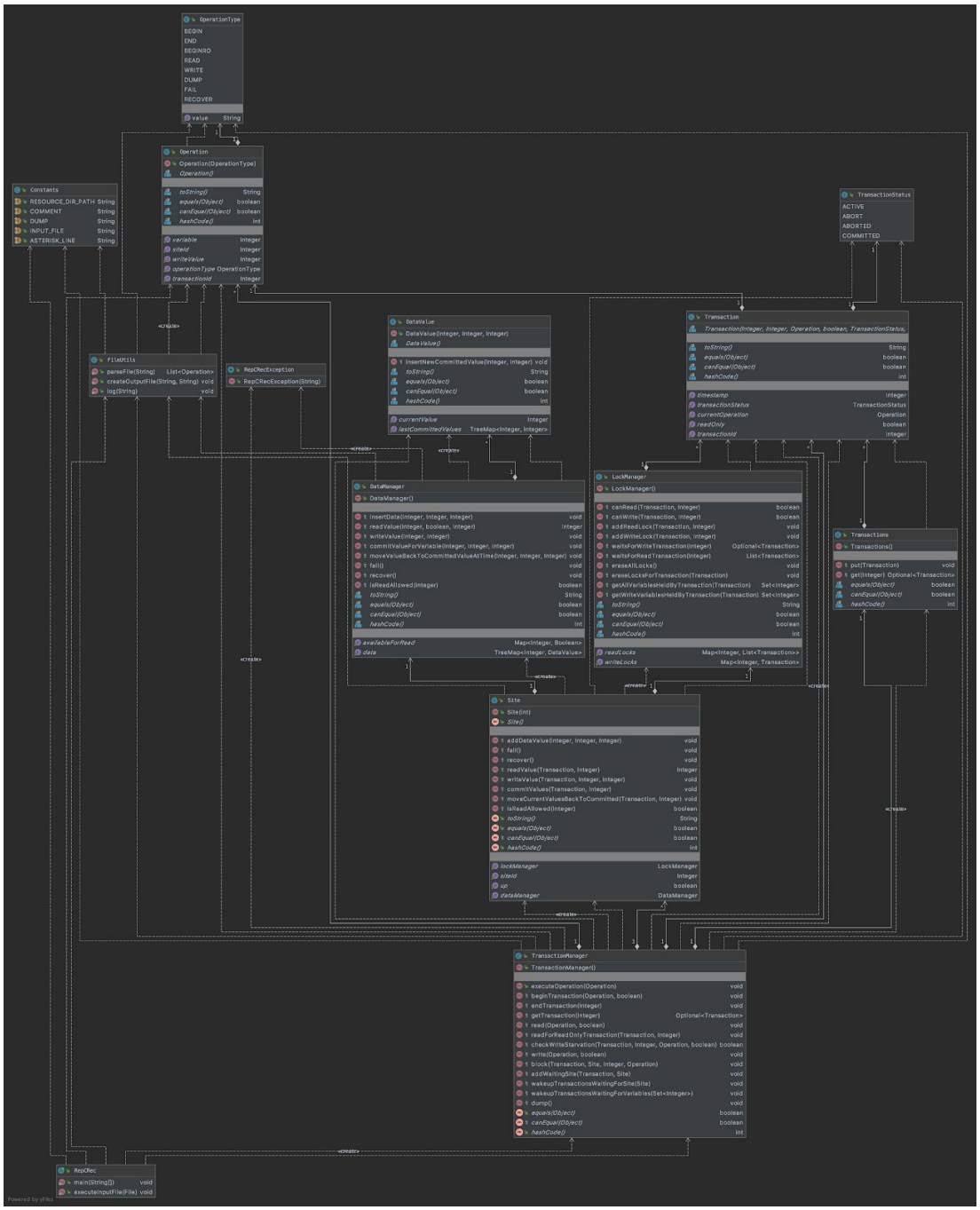
Advanced Databases
Replicated Concurrency Control and Recovery

Shiv Khattar(sk8325), Abhinav Gupta(ag7387)

We will implement a distributed database, complete with multiversion concurrency control, deadlock detection, replication, and failure recovery.



Class Diagram

1. We will implement the available copies approach to replication using strict two phase locking(using read and write locks) at each site and validation at commit time.
2. Read-only transactions will use multi-version read consistency.
3. We will detect deadlocks using cycle detection and abort the youngest transaction.
4. If a transaction is in a waiting state, it will not receive any other event.
5. There will be no starvation for writes.

Major Design Components:

- Operation: This holds the current operation that is happening. This holds information like the type of operation, transactionId, siteId, the variable and the value that needs to be written(if any).

- Transaction: This class will hold the information of each transaction. Each transaction holds it's id, timestamp at which the transaction started, what the current operation is, if the transaction is a read only transaction or not and the current status of the transaction.

- Site: This class holds the information about each site. Each site has a lockManager and a dataManager. The lock manager manages all the locks at the current site and the data manager manages the data at the current site. For each variable in the data manager, we hold the last committed value and the current value from a transaction. Read only transactions will read the value of a variable from the committed value. Lock Manager will expose APIs to add a read/write lock.

- TransactionManager: This class is the central point for all kinds of operations on every transaction. This class will manage all transactions and sites. The transaction manager also manages the waits-for graph. The TransactionManager will make sure there is no write starvation using the waitingOperations. TransactionManager will also take care of deadlock detection and removal. TransactionManager also decides whether to abort the transaction or to commit the transaction using the status field in the Transaction class. If the status is aborted, then the transaction manager will abort the transaction when the end operation is called, if it is Active then the transaction manager will commit the transaction. A transaction is moved to Aborted when a site fails where a transaction was holding a lock.