Description of Lab 3:

Lab 3 handles Transactions within SimpleDB. One aspect of them is locking, which prevents broken states within the database where multiple transactions are dependent on each other. For SimpleDB, this is handled in the form of Strict Two Phase Locking. To handle the locking across the database, we created a `LockManager`. This lock manager keeps track of which transactions are holding locks on each page, whether the locks are exclusive or shared, and which pages each Transaction has a lock on. This is the base functionality that the `BufferPool` utilizes to check whether a transaction should get a page when they call `getPage()`.

With locking comes the possibility of deadlocks—where there are cyclical dependencies between more than one transaction. The `LockManager` also takes into consideration the deadlocks and can check the dependencies of all Transactions to make sure that they are completely acyclical. This is double checked before a transaction can acquire a lock, so that we prevent the deadlock from occurring before it happens.

Part of this transaction system is aborting and committing. Without this, the transactions would be pointless as they would not ensure the ACID properties. Aborting is useful particularly for handling deadlocks, as it allows transactions to be stopped before they can create the dependency cycles. Prior to a commit, the transaction's data is just in-memory so that it can be aborted. However, once the transaction is committed, there is no opportunity to abort them since the data has been written to disk.

Design decisions:

- `Locks` only occur in `BufferPool#getPage()`, since it is the only spot where the database hands out references to the in-memory pages.
- Deadlock detection uses DFS to find cycles between transactions who are locking pages. This is a standard algorithm for cycle detection and should catch more complex cases than simple two transaction deadlocks.
- When there will not be a deadlock, but a lock cannot be acquired, `wait()`.
- Calling `notifyAll()` when a transaction releases any locks to potentially allow waiting threads to proceed.
- Store `PageId -> LockSet` in `LockManager`, since going backward is not as common. Still supported with `getTransactionPages(TransactionId tid)`.
- Use `LockSet` as an ADT for the lock on a specific page. Improves readability and allows for operations on the set of `holders` directly instead of adding additional static methods to `LockManager` to interact with them.
- Allow `LockSets` to be shared or exclusive to handle different behavior depending on who is requesting a lock.

Example of a unit test that could be added to improve the set:

One unit test that could be added to improve the testing suite would be to have additional deadlocking tests. Currently, the deadlocks can pass while the system still deadlocks on the `TransactionTests`. Adding additional tests for this section of the unit tests would allow students to better understand the situations where their code deadlocks instead of trying to debug 5-10 concurrent `Threads`.

Changes you made to the API: None.

Describe any missing or incomplete elements of your code:

`TransactionTest` does not seem to pass consistently. Sometimes `NoSuchElementException()`s are thrown and other times it times out.

Additional Feedback: None.