This Project is again based on the SimpleDB database system. SimpleDB is a multi-user transactional database server, which means it has to deal with concurrency. SimpleDB was written by Professor Dr. Edward Sciore and is widely used as a teaching tool for database systems. For this project, you will be handling deadlocks in SimpleDB.

SimpleDB uses strict 2-phase locking, which does lead to the possibility (even likelihood) of deadlocks. SimpleDB currently uses a timeout mechanism to deal with a deadlock. Your task is to implement one of the following:

1. The first option is to break the current system. Come up with a scenario where the current deadlock handling gives very poor results (or could you cause it to give a non-serializable schedule?) For example, what happens if there is a continuous stream of short transactions that write a single item, and a single long transaction that (slowly) reads the entire database - could you come up with a situation where the long transaction never, or at least takes an unreasonably long time, to complete? This task requires you to consider all the concepts that you have learned from class and use them intelligently to break the current deadlock handling mechanism. We recommend that one (or in a four person team, possibly two) of your members implement this approach, so that you can see if the approaches below (implemented by other team members) do better.

2. Change the current deadlock detection strategy so that it uses the Wait-Die deadlock detection strategy.

3. Change the current deadlock detection strategy so that it uses the Wound-Wait deadlock detection strategy.

4. Change the current deadlock detection strategy so that it uses any other deadlock prevention or detection strategy you saw in the class apart from the aforementioned ones. One approach would be a precedence-based graph-based detection strategy. This could be considerably more difficult than wait-die or wound-wait; it is okay for a four person team if two of you do option 1 (coordinate to make sure you do something different), and one does option 2, the other option 3.

**Project Description:**

The tasks for this projects are-
**Task 1 (Decide):** What is the approach that you have implemented. Why did you pick that? For option 1, describe a weakness of the default SimpleDB timeout, and how you would exploit that weakness.

**Task 2 (Design and Implement):** Implement the approach that you have picked. For option 1, this will include a means to create the tables/data used, and to execute the SQL queries (which may be code that generates a stream of queries.) Give a detailed description of the implementation technique in terms of SimpleDB in the report. For option 1, this should describe the data and queries used.

**Task 3 (Analyze):** Analyze the pros and cons of your implementation and show the drawbacks along with the advantages of the method that you have implemented. Your descriptions should be in terms of SimpleDB and your implementations only. For this task, You may want to explain the test cases that you have used to test your code. Note that these test cases are to show that your system prevents deadlock, but need not show that it is better than the original, and can be done using something like ConcurrencyTest.java rather than as SQL queries. Include scenarios in your test case where a transaction aborts and explain why it aborts.

For option 1, this task is to explain what you expect to happen with your particular data/queries and why, and how you will measure performance (wall-clock time? Counting I/Os? Something else?)

**Task 4 (Evaluate):** Evaluate your implementations against the test cases that you have written and explain how your implementation works for those test cases. For option 1, showing that the default SimpleDB mechanism does poorly means also showing that it would be possible to do better, perhaps even a “best possible” performance - what is the best performance your set of transactions could get, perhaps by delaying the start of some transactions, or changing the order, so you avoid locks conflicting and SimpleDB doesn’t need to abort transactions)

**Task 5 (Compare):** Test the implementations of all of the team members with the “System Breaking Scenario” that one of you has implemented for “Option 1”. Explain the behavior of the algorithms you have implemented, and the original timeout mechanism, with respect to the option 1 scenarios. Explain which works better and why.

Note: If none of the team members has implemented option 1, we will provide some example test cases you can use, or as a team you may come up with scenarios to demonstrate how your approaches can do better than the original timeout mechanism.

**Project Deliverables:**

1. Your Code along with test cases, and a ReadMe file using turnin (just like Project 1 and 2, except this time it is turnin -c cs448 -p project3 <submission folder>). Note that you need to connect to one of the CS lab machines, there are several ways of doing this. If you are on campus, you can use VPN to use resources as if you are on campus. 
https://www.itap.purdue.edu/connections/vpn/
2. A project report that covers all the points that are mentioned in the project description submitted in gradescope.

For each student, Task 1 to Task 4 must be done individually, although you can discuss your ideas with others. You will need to coordinate with your team members to ensure that you pick different approaches (even if you both feel the same approach will give the biggest improvement). Only task 5 will involve multiple team members. The report should be written only by yourself!!

Additional Instructions:

- Submit your team (2 to 4 members) to your PSO instructor by Tuesday, April 13.
- If you cannot find a teammate, submit that to your PSO instructor who will assign you teammate(s). Note that you may end up having someone added to your team, even if you already have two or three.
- You can pick your own teammate(s) but they need to be from the same PSO.

Code Pointers and Suggestions:

- You'll find relevant code in simpledb/tx. ConcurrencyTest.java shows ways to see what is going on with lock management, but for option 1 you should run actual SQL transactions that cause problems with SimpleDB’s timeout mechanism.

- The methods sLock, xLock, and unLock may need to take the transaction’s id or a timestamp as an argument.

- The variable “locks” might need to be changed so that a block maps to a list of information about transaction ids holding a lock on the block (instead of just an integer), since multiple transactions may hold shared locks. You could do some simple encodings like using a negative transaction id to denote an exclusive lock (which may result in fewer changes to the code), or take a cleaner approach.

- Each time through the while loop in sLock and xLock, you may need to check to see if the thread needs to be aborted. If so, then the code should throw a LockAbortException.

- You might also need to make trivial modifications to the classes Transaction and ConcurrencyMgr so that the transaction id and/or timestamp gets passed to the ConcurrencyMgr and LockTable methods.

- Please start with an unmodified SimpleDB (unless you need things from your previous projects.) You should only need changes in simpledb.tx and simpledb.tx.concurrency.

- Use “ConcurrencyTest.java” (inside simpledb.tx) along with your own test cases to test your implementation.