Andrew Shih (704032348)
Roger Lee (704018489)

Write up: CS 143-Lab1

In beginning of the lab, the both of us had only done minimal work in Java in the past, and we had to familiarize ourselves with the syntax and structure of the language. It is also our first times working with databases, so the concepts were all new to us.

Exercise 1:
For exercise 1, we used an a vector to keep track of TDItems in TupleDesc.java and implemented the rest of the class as followed. For merge, I just created a new TupleDesc object and added all items from td1, and then from td2. For our Tuple.java class, we stored everything in an array because no other data structures were imported. However, when we got to the Iterator function, we were confused because there was no immediate way to return an iterator of an array. Then when we looked at what api's were imported, we saw that there was an Arrays library that we had not seen before. Then looking at the functions in that library, we realized we could use the asList function to set the Array to a list and return the iterator of the resulting list. This was the only major point of confusion in the first exercise. The rest was mainly straightforward.

Exercise 2:
For exercise 2, we had to create a new class called Table to account for keeping track of the table names and the primary key. We then created two hashmaps, one that mapped a file's ID to the table, and one that mapped a table to the database file. The rest of the class was straightforward in the function implementations. The only tricky part was realizing that we needed to add an extra type, because in discussion, the TA initially told us to use a hashmap that mapped integers to database files directly. But then nothing would have been able to store the names of the primary key fields.

Exercise 3:
For exercise 3, the only tricky part about the bufferpool was searching through the several java files and finding the right method on obtaining a page to put into the hashmap if the page did not already exist. Thanks to the hints on piazza, we were able to figure out exactly where to go.

Exercise 4:
For exercise 4, there were three classes to implement, two of which were very straightforward. The third class, HeapPage.java, was a little more complicated since the constructor was written, and we had to figure out how to write the rest of the functions by understanding the constructor.

In getHeaderSize, we ran into an error because we forgot to typecast the integers inside the Math.ceil function. Thus the integer expression inside Math.ceil is truncated/floored before being passed to Math.ceil. Thus our Math.ceil statement was actually flooring our expression. We spent many hours trying to debug this.

Exercise 5:
For exercise 5, all that the HeapFile needed to store was a TupleDesc that describes the schema of the table, and a File backing this HeapFile on disk. The constructor simply stores those two vital pieces of information in member variables. The getFile and getTupleDesc functions just returned the requested member variable. The getId function returned the hashing of the absolute filename of the file underlying the HeapFile as suggested. The readPage function created a RandomAccessFile from the File member variable, calculates the offset for the page specified by the pid parameter, seeked to that offet, and read the page at the offset into a byte buffer. Then the RandomAccessFile was closed and then a HeapPage, constructed from the pid and byte buffer, is returned. The iterator function returned a HeapFileIterator which implements DbFileIterator by calling the Tuple iterator from HeapPage. The hasNext and next functions iterate over all the HeapPages in HeapFile and use the iterator of each HeapPage to iterate over its Tuples.

Exercise 6:
For exercise 6, SeqScan stored a TransactionId, a TableId, a TableAlias, a HeapFile Iterator, and a HeapFile. SeqScan implemented DbIterator mostly by calling HeapFile's iterator, which was quite straightforward. The reset function appropriately repeats part of the constructor to reset the TableId and TableAlias of the SeqScan. The getTupleDesc function gets the TupleDesc from the HeapFile and then prefixes all the field names with the TableAlias. The getAlias function simply returns the stored TableAlias. The getTableName function fetches the TableName by looking it up in the Database Catalog with the stored TableId.