Buffer Pool

This assignment involves implementing a buffer pool as a Java generic. Because this assignment will be auto-graded using a test harness we will provide, your implementation must conform to the public interface below, and include at least all of the package, public and private members that are shown:

```
// The test harness will belong to the following package; the buffer pool
// implementation will belong to it as well.  In addition, the buffer pool
// implementation will specify package access for all data members in order
// that the test harness may have access to them.
//
package MinorP2.DS;

import java.util.LinkedList;

public class BufferPool<T> {

   class Buffer {
      long    offset;    // file offset of this data record
      T       data;      // buffered data record

      /**
       * @param offset  file offset of buffered record
       * @param data    buffered record
       */
      public Buffer(long offset, T data) { . . . }
   }

   /**
    *  @param Capacity    size limit of the pool
    *  @param dbFront     client-supplied mediator for reading records
    */
   public BufferPool(int Capacity, dbParser dbFront) { ... }

   /**
    *  @param offset      file offset of requested record
    *  @return            (reference to) requested record, or null
    */
   public T Get(long offset) { ... }

   LinkedList<Buffer> Pool;  // stores elems in MRU to LRU order
   int Capacity;
   dbParser<T> dbFront;
}
```

You may safely add features to the given interface, but if you omit or modify members of the given interface you will almost certainly face compilation errors when you submit your implementation for testing.

You may find it useful to add a number of private helper functions that are not shown above. Since those will never be called directly by the test code, the interfaces are up to you.

You must place the declaration of your `BufferPool` generic in a package named `MinorP2.DS` and specify package access for members as indicated above or compilation will fail.

Your implementation will be designed to use an accessory class, `dbParser`, which will mediate access to the file containing the actual data records. So, your `BufferPool` will not need any knowledge of how the records file is organized.

Design and implementation requirements

There are some explicit requirements, in addition to those on the *Programming Standards* page of the course website:

- You must implement the `BufferPool` generic to conform to the specification.
- The buffer pool must use the pure LRU replacement policy.
- Under no circumstances should any of the `BufferPool` member functions write output.

Testing:

We will be testing your implementation with our own test driver. We may (or may not) release information about that driver before the assignment is due. In any case, it is your responsibility to design and carry out a sensible test of your implementation before submitting it. For that purpose, you may share test code (**but absolutely no code for the buffer pool itself!!**) via the class Forum.

Evaluation:

You should document your implementation in accordance with the *Programming Standards* page on the course website. It is possible that your implementation will be evaluated for documentation and design, as well as for correctness of results. If so, your submission that achieved the highest score will be evaluated by one of the TAs, who will assess a deduction (ideally zero) against your score from the Curator.

What to turn in and how:

This assignment will be auto-graded using a test harness on the Curator system. The testing will be done under Windows (which should not matter at all) using Java version 1.6.18.

Submit a single `.java` file containing your `BufferPool` generic to the Curator System. Submit nothing else. Your solution should not write anything to standard output.

Your submitted source file will be placed in the appropriate subdirectory with the packaged test code, and will then be compiled with the test harness using the following command, executed in the root of the source directory tree:

```
javac testDriver.java
```

Instructions, and the appropriate link, for submitting to the Curator are given in the *Student Guide* at the Curator website:

http://www.cs.vt.edu/curator/.

You will be allowed to submit your solution multiple times; the highest score will be counted.

Pledge:

Each of your program submissions must be pledged to conform to the Honor Code requirements for this course. Specifically, you **must** include the pledge statement provided on the Programming Standards page in one of your submitted files.

Note on the dbParser class and other issues:

The `dbParser` class will conform to the following interface:

```
public class dbParser<T> {

   public dbParser(RandomAccessFile rf) { ... }

   public T fetch(long offset) { ... }
}
```

You can actually implement this yourself (and you'll need something like it for the GIS project anyway). For our testing of the `BufferPool`, we'll be using data files from the GIS project.

The specified `BufferPool` interface excludes some features that would frequently be included:

- the ability to modify a record *in situ* via a method call
- the ability to write a new record into the database file via a method call
- a distinction between "clean" records and "dirty" records; i.e. dealing with the possibility that the copy of a record that's in the buffer pool is newer than the one that is currently stored in the data file

You're free to explore those issues on your own.