Project 2 - Appendix

Here are some additional suggestions for testing the project, and submission instructions.

1. **Submission**
    a. Make changes to server and create zip with file name *simpledb_<unityid>.zip*(tar or gz).
    b. Create Readme file with following contents :
        i. Unity ID.
        ii. List of Files Changed.
        iii. Note on how to test for a scenario in the description (if unit tests /test suite already written) .

2. **Testing for Part 1 - Buffer replacement policy.**

    Init a simpleDB Client
    ```
    SimpleDB.init("simpleDB");
    ```

    Creating a Block -
    ```
    Block blk1 = new Block("filename", 1);
    ```

    Creating a basicBufferMgr
    ```
    BufferMgr basicBufferMgr = new SimpleDB().bufferMgr() ;
    ```

    Pin a Block
    ```
    basicBufferMgr.pin(blk1);
    ```

    Catching Buffer Exception
    ```
    try {
    basicBufferMgr.pin(blk1);
    }
    catch (BufferAbortException e) {}
    ```

Here is a sample testing scenario -

1. Create a list of files-blocks.
2. Check the number of available buffers initially. All should be available as none of them have been pinned yet.
3. Keep pinning buffers one by one and check the number of available buffers.
4. When all buffers have been pinned, if pin request is made again, throw an exception
6. Unpin a few buffers and see if you are still getting an exception or not.
7. Try to pin a new buffer again, and check your replacement policy while seeing which currently unpinned buffer is replaced.

For eg: Consider there are 8 buffers.
pin(1), pin(2), pin(3), pin(4), pin(5), pin(6), pin(7), pin(8), unpin(3), unpin(2).
Buffer replaced next: 2

3. Recovery Manager Details

Method Declarations (changes) .

Change the following existing methods to the declaration given below.

```
 public int setInt(Buffer buff, int offset , int oldval , int
newval) {
      /**
      * @param buff the buffer containing the page
      * @param offset the offset of the value in the page
      * @param oldval the old value to be written
      *@param newval the new value to be written
      */

      }

      public int setString(Buffer buff, int offset, String  oldval ,
String newval) {
      /**
      * @param buff the buffer containing the page
      * @param offset the offset of the value in the page
      * @param oldval the old value to be written
      *@param newval the new value to be written
      */
      }
```

Method Declarations (Additions) .

Add the following method declarations in LogIterator.java.

```
 public BasicLogRecord  nextForward(){
/**
    * Moves to the next log record in forward order.
    * If the current log record is the latest in its block,
    * then the method moves to the next block,
    * and returns the log record from there.
    * @return the next log record
    */

}

   private void moveToNextForwardBlock() {
    /**
     * Moves to the next log block in forward  order,
     * and positions it at the first record in that block.
     */

    }
```

4. **Testing for Part 2 - LogRecordIterator and Recovery.**

Init a simpleDB Client.

```
SimpleDB.init("simpleDB");
```

Creating a Block.

```
Block blk1 = new Block("filename", 1);
```

Create a RecoveryManager with id = 123.

```
RecoveryMgr rm = new RecoveryMgr(123);
```

Commit a transaction .

```
rm.commit();
```

Recover a transaction.

```
rm.recover();
```

Sample setInt

```
int lsn = rm.setInt(buff, 4, 1234);
```

```
buff.setInt(4, 1234, txid, lsn);
```

Flushing all transactions

```
bm.flushAll(txid);
```

Using Log Record Iterator to print records .

```
LogRecordIterator it = new LogRecordIterator();
System.out.println(it.next());
```

Here is a sample testing scenario for LogRecordIterator -

1. Create a block and pin it to a buffer.
2. Create a recovery manager for a transaction (txid=123)
3. Use setInt and setString to set logs for the transaction with txid=123.
4. Use LogRecordIterator it.next() to see the logs are read in forward manner and prints old and new value.
5. Use multiple blocks and repeat the above steps.

For eg: Consider there is a transaction with txid=123.
Create two blocks blk1 and blk2. Use setInt() and setString() on both blocks.
Use LogRecordIterator it.next() to see the logs are read in forward manner from one block to other block.

Here is a sample testing scenario for Recovery -

1. Create multiple RecoveryMgr for transactions with txid=1, txid=2 and txid=3 respectively.
2. Create a block, pin it to a buffer and write multiple logs(different types of LogRecord eg. START, COMMIT, SETINT, SETSTRING) for each transaction.
3. Unpin and pin that buffer again to check changes are reflected in the buffer.
4. Commit one of the transactions.
5. Try to recover all transactions.
6. Verify that the recovery is done as per the description.

For eg.
Consider two transaction txid=1 and txid=2
Add following log record for txid=1

1. setInt(oldval=5, newval=10) [UPDATE log record]
2. Commit transaction for txid=1. [COMMIT log record]

Add following log record for txid=2

3. setInt(oldval=5, newval=10) [UPDATE log record]
4. setString(oldval="Hello",newval= "World") [UPDATE log record]

1. Start the recovery phase for txid=1
    a. Undo Stage (Already implemented in simpledb- reading backwards).
        i. For *commit* log record
            1. Add that transaction to commit list.
        ii. For update record setInt(oldval=5, newval=10)
            1. Transaction already in commit list .
    b. Redo Stage (reading forward).
        i. for update record setInt(oldval=5, newval=10)
            1. Current record is an *update* ,restore old value and txid=1 is in committed list , restore new value at specified location.
            2. For *commit* log record, the record is not update record.

2. Start the recovery phase for txid=2
    a. Undo Stage (Already implemented in simpledb- reading backwards).
        i. For update record setString(oldval="Hello",newval"World") .
            1. restore old value at specified location.
        ii. For update record setInt(oldval=5, newval=10)
            1. restore old value at specified location.
    b. Redo Stage (reading forward).
        i. For update record setInt(oldval=5, newval=10)
            1. Current record is an *update* and txid=2 is not in committed list.
        ii. For update record setString(oldval="Hello",newval"World") .
            1. Current record is an *update* and txid=2 is not in committed list.

Note: This is a suggestion on testing strategy. There can be other scenarios as well.