Project 2 (Index Library) Proposal
May Zhai
Bill Ge

a) API Documentation
Please see API_Documentation folder in the Requirements Checklist Files folder. This folder contains the exported Javadocs for all code. BTree.html is a copy of the documentation for the BTree and Datastore.html is a copy of the documentation for the datastore.

b) Design Document
Everything is in Java

Index Tree
The index tree is a generic implementation of a B+ tree

- The user specifies the order of the tree (default order is 1)
- The tree allows the user to insert a key, delete a key, search for a key, ranged queries, get a linked list of leaf nodes (in order), and print the tree (breadth-first)
- The implementation of the tree requires that the user specifies a Key object as the key, where the object can be of any class that extends the abstract Key class
- Leaf nodes of the tree contain keys that can be mapped (using the KeyValueMap class) to an offset in the datastore (which can be used to get the record itself in the datastore)
- We use serializable to store the structure of the BTree on disk

Datastore
Our datastore will consist of fixed width records stored in a binary file

- Each record is a total of 120 characters (fixed width)
    - Rank: 3 characters
    - Year: 4 characters
    - Song Title: 25 characters
    - Artist Name: 25 characters
    - Features: 5 characters
    - Size: 8 characters
    - Time:4 characters
    - Group: 5 characters
    - Debut Year: 4 characters
    - State: 17 characters
    - Country: 20 characters
- Records can be located by knowing the offset of the record

Caching/Buffering

- Whenever the user looks up a current record or adds a record, we will cache that request – cache the key and the data in a hashtable, so that when the user requests the key again, we can return the data from the hashtable instead of looking it up on disk

Concurrency

- All BTree methods are made synchronized to prevent multiple threads from accessing the tree at the same time
- In the sample application, the ConcurrecyTest class, we spawn multiple threads to test whether BTree supports multithreading. We add the same keys in each thread, with multiple threads. In the end, the Btree structure should look like as if only one thread has added the data, since the tree will not allow users to add the same key more than once

c) Timeline

Priorities are numbered where 1 = most urgent and ascending numbers represent decreasing priority

Proposal: Thursday, October 13, 2011
Goals:

- 1. implemented tree structure and be able to index keys (mzhai)
- 2. implemented the feature to create datastore from a CSV file (bge)

Key indexing or Datastore: Thursday, October 20, 2011
Goals:

- 1. caching implementation (mzhai)
- 2. buffering implementation (bge)
- 3. concurrency control (mzhai)

Record Indexing/Caching/Buffering: Wednesday, November 2, 2011
Goals:

- 1. API implementation (bge)
- 2. sample application with sample data (mzhai)
- 3. documentation about how to use test program (bge)
- 4. update document describing any changes made since the original proposal (mzhai)
- 5. updated copy of the API documentation (bge)

The plan is to completely the above goals by Monday, October 31st. This way, there will be some time left over in order to fix any last minute mistakes.
Final Deadline: Thursday, November 5, 2011