CSE 562: Project #1

Team project. Team size: 3 students. The same teams will be selected for Project # 2. You have to use Java for implementing the project. The project deadline is March 30th, midnight. Late submissions won't be accepted unless in proven extraordinary situations. Information on how to submit will be posted not later than 2 weeks before the deadline. Also during first two weeks after publication the project description may (and probably will) be subject to some non-major alterations.

You will implement a framework for storing files with one or more indexes that use the B+-tree data structure. They will be stored in main memory, with their persistent versions being kept on disk.

1. Records consist only of variable-length character fields. The allowed characters are "a" to "z", "A" to "Z" and digits (but you don't need to check whether records conform to this). Maximum number of characters per field is set by user when creating a file.

2. Index keys (i.e. columns on which the indexes can be created) will have maximum length of 24 characters. If creating an index over a column which has maximum length longer than 24 characters is attempted, IllegalArgumentException should be thrown.

3. key values may be repeated.

4. the indexes are dense.

5. the file is organized as a heap.

6. the B+-tree nodes have between 4 and 8 index records.

7. no block structure is assumed.

8. there is no requirement to use a specific format for the data stored on disk.

You can assume the following:
file size not bigger than 1MB
record size not bigger than 1KB
whole data always will fit in the memory of the JVM.

You need to implement 3 public classes described below. All of them should be in a package "database". You may implement additional classes, but if you decide to do so, they should be nested classes or classes which are put in some new package(s) in the "database" package.

1. A public class DataManager

The class needs to implement following methods:

`public static DataFile createFile(String fileName, Map<String, Integer> descriptor):`
creates and returns a new DataFile, descriptor is a map of fields(columns names) and their maximum lengths, fileName is used to identify the file if it was to be stored on disk and then retrieved from there, fileName needs to be unique. If already used fileName is supplied throw IllegalArgumentException.

`public static DataFile restoreFile(String fileName):` restore file contents from disk, returning that file, if file of that name is already in memory or file of that name hasn't been stored on disk throw IllegalArgumentException.

`public static String print(Map<String, String> record):` print record contents in human-readable format, a record is represented as a Map that maps names of columns to data values. Return as String what was printed.

`public static void exit():` exit the system closing all files and saving all the information to disk (all references to files, indexes etc. should be removed from memory).

2. A public class DataFile

This class represents a File in which records will be stored (conceptually you can think of it as if it was a database table). How you will store the records in that file is totally up to you. Yet the records need to be stored with some order so that it would be possible to traverse all the records in the file visiting each one exactly once (to do so Iterator is added to the DataFile class).The class should contain:

`public Index createIndex(String indexName, String column):` creates and returns a B+tree index on the file over the specified column, indexName is used to identify the index if it was to be stored on disk and then retrieved from there, indexName is unique within the file. If indexName of already existing in memory for that file index is used, throw IllegalArgumentException.

`public Map<String, String> getRecord(int recordId):` recordId is internal record identifier, returns a map representing the record (or null if such a mapping doesn't exist).

`public int insertRecord(Map<String, String> record):` insert record into the file. Also update indexes over the file. Returns recordId – internal record identifier (which needs to be unique within the file). If record contains columns that don't belong to the file or value in any column is longer than maximum allowed length then throw IllegalArgumentException.

`public void dumpFile():` write file contents to disk, if that file was already saved before then overwrite the previous save.

`public String viewFile():` print the content of the file in a human-readable format. Return as String what was printed.

`public void dropFile():` delete the file, all indexes over it and all records inside: both in memory and on disk.

`public Index restoreIndex(String indexName):` restore index contents from disk. If index of this name is already over the file or index of this name is not stored on the disk throw IllegalArgumentException.

`public void dropIndex(String indexName):` delete index, both in memory and on disk. If index of this name on the file doesn't exist do nothing.

`public Iterator<Integer> iterator():` Return a new FileIterator (when Iterator is created it points to the first element of the collection).

`private class FileIterator implements Iterator<Integer>:` Iterator for fileIndex implementing the interface methods (http://download.oracle.com/javase/1.5.0/docs/api/java/util/Iterator.html), remove must be supported(remember that when you remove a record all indexes must be updated)!   You can learn how Iterators are supposed to be implemented in Java on the example of SimpleListIterator from this code: http://www.docjar.com/html/api/java/util/AbstractList.java.html. This Iterator needs to allow traversal over all records in the file. The next() method of Iterator returns recordIds.

3. A public class Index

This class represents a B+ tree index over records in a file using a chosen column. The class should contain:

`public void dumpIndex():` write index contents to disk. If that index was already saved before then overwrite the previous save.

`public void viewIndex():` print the content of the index in a human-readable format.

`public Iterator<Integer> iterator(String key):` which will return new IndexIterator, the Iterator should only traverse over records whose value in the indexed column is equal to key.

`private class IndexIterator implements Iterator<Integer>:` Similar situation as with FileIterator, also needs to implement the Iterator interface methods: remove must be supported(remember that when you remove a record all indexes must be updated)!   The order of the traversal with use of this Iterator must be consistent with the index and the key.

Other details:

There may be many iterators “opened” for the same file or index at the same time. In situation when between iterating steps index or file content is changed by some other iterator or method throw ConcurrentModificationException when next next() or remove() would be called.

Record Identifiers should be nonnegative integers.

You will need to add additional methods to above mentioned classes in order for the methods that are needed be able to work (e.g. you will need constructors for DataFile and Index). Also you may choose to add some other methods, in case you would find them helpful.