CS 587
Creators: Haomin He, Zicheng Ren

HashBucketPage.java
countEntries() counts the total entries in the pages in the list. It pins each page onto buffer pool and calls getEntryCount() from SortedPage to gather the entry counts. After iterating through the pages it returns the total entry counts.

insertEntry() returns true if the page is modified, false otherwise. If the next page is invalid, add a new page at the end of the list. If the page is valid, insert entry in this page.

deleteEntry() returns true if deleting an entry made the page dirty, false otherwise. It calls deleteEntry() from SortedPage to delete entry from bucket. If the entry is not in the list throws illegalstate. It looks for the next page in the list and checks if its entry count is 0. If it is 0 delete the empty page and point to the next page.

HashIndex.java
HashIndex() checks if there exists a file in the library if the filename is not null. It creates a new index file if the name does not exist and if the filename is not null add it to library.

finalize() deletes the index file if it’s temporary by calling deleteFile().

deleteFile() deletes the index file from the database and frees all its pages. It traverses through hash dirpage and counts entries for each directory. If filename is valid, delete from library.

insertEntry() inserts a new key, rid data entry into the index file. First, we declare and initialize variables and gets the hash value for the search key, given the depth. Second, checking entry length, display IllegalArgumentException if the entry is too large. MAX_ENTRY_SIZE = (PAGE_SIZE - HEADER_SIZE - SLOT_SIZE), it means the biggest size an entry could be. We pin the page the hash value locates at. And insert a data entry into a bucket, apply insertEntry to the primary page of the bucket, return true if inserting made this page dirty, false otherwise. Unpin the page after insertion.

deleteEntry() deletes the specified data entry from the index file. First, we declare and initialize variables and check the boundary to make sure the hash value is greater and equals to HashDirPage.MAX_ENTRIES. Pin the HashDirPage and try to delete the entry, throw IllegalArgumentException exception if failed.

printSummary() prints a high-level view of the directory, namely which buckets are allocated and how many entries are stored in each one.

HashScan.java
An HashScan object retrieves all records with a given RID of the record. It is created only through the function openScan() in the HashIndex class. In each HashScan object, there are HashDirPage, HashBucketPage, PageId objects. Also it implements pin() and unpin() functions from Minibase BufferManager, because HashScan.java should have at most one page pinned at any given time. While unpinning, the data is always UNPIN_CLEAN because we don't modify the value.

finalize() is called by the garbage collector when there are no more references to the object; closes the scan if it's still open. First, we need to check the being scanned hash bucket page's pid is valid or not, if it is valid, closes the index scan, releasing any pinned pages.

close() closes the index scan, releasing any pinned pages. First, we need to check the being scanned hash bucket page's pid is valid or not, if it is valid, releasing any pinned pages and make the pid invalid. And unpin the page with UNPIN_CLEAN.

getNext() gets the next entry's RID in the index scan. First, we need to check the being scanned hash bucket page's pid is valid or not, if it is valid, do index scan and see if there is a next bucket. Then return the next record id, or throw IllegalStateException if the scan has no more entries.

References:

Hashing Assignment-DBIplem-Winter2018.pdf
Ramakrishnan, Gehrke - Database Management Systems
https://github.com/yu-yang/project/blob/master/Java
https://github.com/munafahad/
https://github.com/AhmedGad/Database/blob/master/decompiled