Lab 3: NEUStore

Due date: Wednesday, October 6, end of lab period.

Lab Assignment

This is a team assignment. Each team completes and deliveres a single submission.

The purpose of this lab is to get acquainted with NEUStore, the Java package for buffer management and paginated disk access. This is a team lab, to be performed by the project teams.

NEUStore

NEUStore is a Java package providing descriptions of a number of abstract classes for creation and maintenance of paginated index structures on disk. In addition, NEUStore provides sample working implementations of the most important classes. In particular, NEUStore contains the following.

- **Buffer Manager.** NEUSTore offers an abstract class `DBBuffer` representing the DBMS buffer. Buffer management policies (the asynchronous part of the buffer manager) are represented by the abstract methods of this class, which need to be instantiated for each type of buffer manager. In addition `DBBuffer` offers the read/write/pin/unpin API (the synchronous part of the buffer manager) which allows application programs to communicate with the buffer to read, write and pin/unpin pages.

  NEUStore also offers class `LRUBuffer` which extends `DBBuffer` and implements the traditional Least Recently Used buffer management policy. This lab assignment, as well as the course project will assume that you use `LRUBuffer` as your buffer manager.

- **Disk Pages.** NEUSTore offers an abstract class `DBPage` representing a single disk/buffer page. It also offers two classes, `NaiveHeapFilePage` and `HeapFilePage` which extend `DBPage` and implement a simple heap file page (storing only integers) and a generic heap file page respectively.

  This lab assignment requires each team to study the source code of the implementations of `NaiveHeapFilePage` and/or `HeapFilePage` and modify it as needed to meet the needs of the lab.

- **Index Files.** Collections of disk pages from *index files.* NEUStore provides an abstract class `DBIndex` representing an index file on disk. Index files can contain disk pages of different types. NEUStore includes two classes, `NaiveHeapFile` and `HeapFile` which extend `DBIndex` for the naive and generic heap file implementations.

- **Tests and examples.** NEUStore comes with two examples mentioned above. The first example is an implementation of a naive heap file, which only stores single integers on disk pages. The second example is an implementation of a generic heap file, which can store any *(Key, Value)* records (where both the key and the value may be compound). In addition, `NEUSTore` provides a sample test program showing how the generic heap file can be initialized, populated and queried.

Assignment

Using NEUStore as the back end, create a program that does the following.

R1. **Dataset description.** Your program shall provide heap file storage for the data found in the RealEstate.csv file from the `HOUSES` dataset, available at

> https://wiki.csc.calpoly.edu/datasets/wiki/Houses

The file contains information about the prices of a number houses that were up for sale in San Luis Obispo county and in the North Santa Barbara county. The format of the file is

```
MLS,Location,Price,Bedrooms,Bathrooms,Size,Price/SQ.Ft,Status
```

where the attributes are as follows:

Attribute name	Type	Explanation
`MLS`:	`INT`	(Primary key) unique ID of the listing
`Location`:	`String`	Town of the listing
`Price`:	`FLOAT`	Price of the listing in dollars
`Bedrooms`	`INT`	Number of bedrooms
`Bathrooms`	`INT`	Number of bathrooms
`Size`	`INT`	Area of the house in square feet
`Price/SQ`	`FLOAT`	Price of the house per square foot
`Status`	`String`	Type of sale (Regular, Short Sale, Forclosure)

A sample record looks as follows:

```
154531,Cambria,549000.00,2,2,924,594.16,Short Sale
```

(Listing # 154531 is a two-bedroom, two-bathroom, 924 sq. ft. house in Cambria, for sale for $549,000 as a short sale, with the price of a square foot beeing $594.16)[1]

The dataset contains 781 listings. The first line of the `RealEstate.csv` file lists the names of the columns.

R2. **Format assumptions.** Your program needs only to provide storage for the data found in `RealEstate.csv`. You can (and shall) determine the lengths of the string fields in this file based on its contents. There is no reason to set the lengths of string fields to be significantly longer than the longest string in the dataset.

R3. **Disk Storage.** You shall design appropriate format for storing records from the `RealEstate.csv` file. You shall design appropriate disk page format (including page headers). You shall extend existing NEUStore implementation[2] to create appropriate classes for each disk page format you intend to use in the program, as well as for the index file in which all data will be stored on disk.

R4. **Internal API.** Your program shall provide implementations[3] of the following methods for the index file:

- insert : insert a given record into the index structure;
- delete : delete a record from the index structure;
- update : replace a specified record with a new one;

(you shall make decisions on the arguments for these methods).

R5. **Program flow.** Your program shall work as follows:

- When the program is started, it reads and parses the contents of the`RealEstate.csv` file. You can assume that the file is in the working directory of the program.
- The program then builds the index structure(s) for storing the contents of the file, and inserts, one-by-one all records from the file into the index structure.
- The program scans the index structure and outputs all records stored in it, in the order in which the records are accessed.
- The program scans the index structure and outputs records for all San Luis Obispo listings.

[1] Yes, it **is/was** a real listing.

[2] You may, if you want, write all your code from scratch, but using NEUStore-provided classes will probably be easier.

[3] You are allowed, where suitable to rely on NEUStore implementations

- The program deletes all Foreclosure listings. It outputs the contents of each deleted record.
- The program scans the index structure and outputs all records in it.
- The program reinserts one-by-one all deleted records. [4]
- The program scans the index structure and outputs all records in it.
- The program (retrieves and) updates the first record stored in the index and replaces the location of the listing with `San Francisco`, makes the house to be 5000 square feet, and updates the price per square foot accordingly.
- The program scans the index structure and outputs the first ten records in it.

Notes and Comments

The data structure you need to implement is more complex than the `NaiveHeapFilePage` class, but it is simpler (because it is fixed) than the `HeapFilePage` class. In fact, it can be done as an instance of using `HeapFilePage`, but I recommend modifying one of the two classes to create a data structure for exactly the record structure you need.

In addition to the insert/delete/update API, the program essentially requires you to implement a scan operation: starting from the first index page, read and output every single record (or read and output every single record up to a given number of records). On top of it, you need a simple *selection* operation, a scan of the index file, retrieving and outputing only the records of San Luis Obispo houses.

The delete procedure described in **R5.** can be performed as follows: your program employs a scan of the index structure, with a check of the `Status` value (this is a simple modification of the "select San Luis Obispo houses" scan), then the API's delete method is issued for each discovered record.

Submission Instructions

Your work essentially modifies the NEUStore package. You shall submit the entire package: both the new files you create, and the original NEUStore distribution files. For simplicity, submit your implementation in an archived form: either `.zip` or `.tar.gz` files are accepted. Name your file:

```
lab03.zip
```

or

[4] Feel free to "cheat" here. You can extract all "Foreclosure" records into a separate file, or create a separate global data structure, to make identification of records easier.

`lab03.tar.gz`

In addition, submit a plain text README file containing the following information:

- Group name.
- Group members, emails.
- Compilation, running instructions.
- The name of the data file you are creating. (I will run your program, and then will manually check the data file).
- A brief summary of the group's approach to the project. List any major design decisions (which classes were extended, how much code was borrowed, how much is new). Also, evaluate how hard/easy it was for your group to understand how to use NEUStore. List any outstanding questions about the functionality of NEUStore you group has.

Use `handin` to submit:

```
handin dekhtyar lab03-468 lab03.zip  README
```