CS 321 Data Structures (Fall 2021)
Programming Project #1 (50 points)

Introduction:
In this project, we will design an implementation of a **cache** using a linked list data structure available in the Java class LinkedList. The main class should provide multiple overloaded constructors to allow a 1-level or 2-level cache to be created. We will also write a test program to test our cache implementation. The data to be stored in cache should be generic objects but an instance of a cache only stores a given type of objects.

Description:
Concept: From the dictionary: *"a collection of items of the same type stored in a hidden or inaccessible place."* In computing, a cache is a storage mechanism implemented in hardware or software. A cache is used to improve access time in the future as well as to improve throughput. For example, CPUs have one or two levels of cache built in to improve memory access. A web server may cache the most requested pages in memory so that it can return them faster to users. Caching is a widely used technique in computing. We can use several different types of data structures to implement a cache abstract data type (for example: a linked list and others that we will see later this semester). Caching is often implicit or hidden from the user.

Operation:
Read: If a data item has a copy in cache, application can read this data item from cache directly. The usage of cache is as follows. Whenever an application requires a data item, it searches the cache first. If it is a cache hit, then the cache returns the data item to the application and the data item will be move to the first position in the cache (we call it the Most Recently Used MRU scheme). On the other hand, if it is a cache miss, then the application needs to read the data item from disk and then the data item from disk will be added to the first position of the cache. Note that if the cache is full, the last entry (oldest one) in the cache will be removed before a new entry can be added.

Write: Similarly, whenever an application writes a data item to disk, the system will perform the same write operation to the cache copy of the data item (if any) and then move it to the first position in cache. Note that the `write` operation is equivalent to a `remove` operation followed by an `add` operation.

One-level Cache:
A single-level cache and it works as described above.

Two-level Cache:
A 2nd-level cache sits behind the 1st-level cache. Usually, the 2nd-level cache is much bigger than the 1st-level cache. Assume the 2nd-level cache contains all data in the 1st level cache, which is called (`multilevel inclusion property`). Two-level cache works as described in the following pseudo-code:

```
Search cache1;
Increment the # of cache1 references;
if cache1 hits
   then Increment the # of cache1 hits;
        Move the hit data item to the top of both cache.
   else Search cache2;
        Increment the # of cache2 references;
        if cache2 hits
           then Increment the # of cache2 hits;
                Move the hit data item to the top of cache2;
                Add the data item to the top of cache1;
           else Add the data item to the top of both cache;
```

Hit Ratio:
Some terms used to define hit ratio are:
HR_1: 1st-level cache hit ratio
HR_2: 2nd-level cache hit ratio
HR: (global) cache hit ratio

NH_1: number of 1st-level cache hits
NH_2: number of 2nd-level cache hits
NH: total number of cache hits ($= NH_1 + NH_2$, in case of two-level cache simulation)

NR_1: number of references to 1st-level cache
NR_2: number of references to 2nd-level cache (= number of 1st-level cache misses)
NR: total references to cache ($= NR_1$, in case of two-level cache simulation)

- One-level cache: $HR = \frac{NH}{NR}$
- Two-level cache: $HR_1 = \frac{NH_1}{NR_1}$ $\quad HR_2 = \frac{NH_2}{NR_2}$ $\quad HR = \frac{NH}{NR} = \frac{NH_1+NH_2}{NR_1}$

What you need to do:

1. Clone the git repo for all the class examples and project starter code with the following command:

   ```
   git clone https://github.com/BoiseState/CS321-resources
   ```

   The test files and sample results for this project are in the subfolder `projects/p1`.

2. Create the project in your favorite IDE.

3. Design the `Cache` class along with related private (or public) supporting classes. The `Cache` should at least have the following public methods – `getObject`, `addObject`, `removeObject`, `clearCache`, along with two constructors (one for 1-level cache and the second for 2-level

cache) as well as an appropriate `toString` method that prints out all the stats. You may use an interface, private classes, private methods to help you with the design.

4. Design and write a test program `CacheTest.java`. It should read the input file and parse it into words. Use the delimiters `"\t "` (a tab and a space)to match results exactly with the posted results. It's usage should be

   `java CacheTest 1 <cache size> <input textfile name>` or
   `java CacheTest 2 <1st-level cache size> <2nd-level cache size> <input textfile name>`

   The cache size(s) and the text file should be input as command line arguments. Your program should create a 1-level cache (option 1) or a 2-level cache (option 2) with the specified size(s) and read in the input text file word by word. For each word, search the cache(s) to see whether there is a cache hit and update the cache accordingly. We will use the file `Encyclopedia.txt` and a small text file `small.txt` (in the same directory as the Encyclopedia.txt does) to test your program.

5. Your program should output the cache hit ratio(s) after reading all words from the input text file. You can find the sample outputs:

   ```
   result-small.txt
   result1k2k-Alice-in-Wonderland.txt
   result1k2k-Encyclopedia.txt
   result1k5k-Encyclopedia.txt
   ```

   in the git repo for the class under `projects/p1`.

6. Measure the elapsed time (in Milli-seconds) to run your program

   ```
    java CacheTest 2 1000 2000 Encyclopedia.txt
   ```

   and report the time required in a the`README.md` file. We can call the method `System.currentTimeMillis(` to measure the time or use the native timing library as shown in the class examples.

7. A template for the `README.md` file is provided in the class repository in the file `README_TEMPLATE.md` found in the `projects` folder in the class git repository.

Submission:

Copy your project to `onyx` by copying all of your files to an empty directory (with no subdirectories). It should only have the following files (remove all data files, `.class` files etc):

`Cache.java, CacheTest.java, README.md, (additional Java files if you used them in your solution`

Then submit as shown below FROM WITHIN your project directory:

```
submit amit CS321 p1
```