CS181, Programming Languages

Assignment 4

(due: 11-13-2000)

For both problems submit only the source files (.java) using the WWW-Turnin. Remember to include your name in the files!

Problem one (Compression)

Make a command line Java application that takes a file name as a command line argument, reads the file line by line and converts it into binary format. The input file has the following format:

```
Name   Id  x1     x2     y1     y2     z1     z2
string int double double double double double double

Example line:
  object_one 1 0.1 0.2 1.4 1.8 0.8 1.2
```

Newlines, tabs and spaces should be truncated in the output file. (Hint: Use the *DataOutputStream* class for producing the output file.)

Problem two (Caching)

Built an LRU cache mechanism for storing objects in memory. An LRU cache replaces the entry that was least recently used, whenever the cache is full. A general solution would be to associate every entry with a unique key using an associative array and keep a queue with the keys sorted in descending order according to when they were last accessed (the least recently accessed key is first in the queue.) When a request is made for fetching an object, first the cache is checked to see if it contains that object. If that is true, the queue is updated and the particular key associated with the object is moved to the tail. If the object was not already in the cache, it has to be loaded (for example from a file on the disk) and added in the cache. If the cache is not full, the object is stored in memory and the key associated with it is added at the tail of the queue. If the cache was full, some entry has to be removed. The LRU policy states that the entry associated with the key that lies in the head of the queue, has to be removed. So the object gets removed from the cache and its key from the queue. Now, there is a free slot in the cache for the new entry.

Create an **abstract** class named *LRUCache* that should use a *HashMap* as the associative array and a *LinkedList* as the queue. The *HashMap* should have a fixed number of entries. Use a counter to hold the number of entries cached. The size of the cache should be configurable through a command line argument. The *LRUCache* class should implement two methods.

- **public Object get(Object key)**: This function request the object associated with *key* from the cache.
- **public abstract Object read(Object key)**: This function is abstract and no implementation should be given. A class that subclasses *LRUCache* should provide the specifics of how an object is read. In such a way you may have various subclasses of *LRUCache* that might read from files, over a TCP socket, or anything you might think of!

To test your generic LRU cache implementation create a class named *MyCache* and implement the **read** method to parse the **binary** file of problem one. The key for indexing the objects should be the object id, thus ids should be unique. Create a class named *Box* to represent the objects (notice that each line of the file represents a 3D Box.) *MyCache* should override method **get** with the following definition: **public Box get(Integer id)**. So, the return type should be a *Box*, and the key should be an *Integer*.

Hints:

```
In class MyCache:
  public Object read(Object key) {
    // key is an Integer, so cast it.
    Integer id = (Integer) key;

    // read object with that id from the file.
    ...
  }

  public Box get(Integer id) {
    Box b = (Box) super.get((Object) id);
    return b;
  }
```

In your main method of *MyCache* randomly generate some object ids and try to get them from the cache. Use information messages to see when you have a cache hit or miss. You may also request the user to give the object ids.