SPRING 2004: **COP 3530** DATA STRUCTURES
[PROGRAMMING ASSIGNMENT 4; DUE APRIL 8 IN CLASS.]

Problem Description

Your program should repeatedly prompt for the names of a set of movie stars. For each such set, it should list out all movies in which they have co-starred. Each set of stars is terminated by a blank line. For example, if the input list is as follows:

```
Guinness, Alec
Sellers, Peter
Connor, Kenneth (I)
<Blank Line>
```

Then the output should be as shown below:

```
1. Guinness, Alec
2. Sellers, Peter
3. Connor, Kenneth (I)
have co-starred in the following movies:
1. Ladykillers, The (1955)
2. To See Such Fun (1977)
```

What to Submit

Submit the output obtained when your program is run on the input file called `Query.txt` available from your course web page. Note that the file contains several queries. Each query consists of a set of stars. Each set of stars is separated by a blank line. The file is terminated with two blank lines.

As usual, also submit the source code for your program and the output of `Javadoc`. Your floppy diskette should contain all the requisite `.java, .class, .html, .dat, .out` files that are relevant for the grader to check the program. Do not include the large database of actors and actresses on the zip. Make sure that the hard copy you submit is the same as the copy on the floppy, and is compilable from the floppy. Also include a statement swearing that the submitted work is your own.

Input

There are two data files; both have identical formats. These files are: actors file and actresses file. These files are both compressed in .gz format, and were obtained from the Internet Movie Database. Combined, they are 65.8 Mbytes (compressed!) and were last updated Mar 26, 2004. These files are available in the computer lab in the folder \\`manatee\giri`. YOU MAY NOT COPY THEM INTO ANY OF THE LOCAL FIU MACHINES BECAUSE YOU WILL EXCEED YOUR DISK QUOTA. These datafiles contain approximately 800,000 actors/actresses in a total of less than 250,000 movies, with about 3 million roles. These files also list TV roles, but you must not include TV roles in your analysis.

Before you run on the large data sets, use the small (uncompressed) file sample.list to debug the basic algorithms. In this data file, there are six actors, named `Ana, A; Berman, Bill; Casper,`

`Cup; Dill, Diana; Engel, Elaine` and `Fogel, Frank`, who have been in movies such as X, Y, and Z.

Here are some useful observations about the data files. You can read the input file line by line by wrapping a FileInputStream inside a BufferedInputStream inside a GZIPInputStream inside an InputStreamReader inside a BufferedReader. You may not uncompress the file outside of your program.

1. There are over 200 lines of preamble that can be skipped. This varies from file to file. However, you can figure it out by skipping all lines until the first occurrence of a line that begins with "Name", and then skipping one more.
2. There are many postamble lines, too, starting with a line that has at least nine dashes (i.e., `---------`).
3. A name is listed once; all roles are together; the name starts in the first column.
4. A movie title follows the optional name and a few tab stops ('\t'). **Warning:** There are some messed up entries that have spaces in addition to tab stops.
5. The year should be part of the movie title.
6. Movies made in 2003 or later should be skipped.
7. A TV movie, indicated by (TV) following the year, is to be skipped.
8. Archive material, indicated by (archive footage), is to be skipped.
9. A TV series, indicated by a leading " in the title is to be skipped.
10. A video-only movie, indicated by (V) following the year is also to be skipped.
11. Blank lines separate actors/actresses, and should be skipped.

Strategy

In order to compute your answers, you will need to store the data that you read. The main data structures are (1) a `Map` in which each key is the name of an Actor and each value is the object for that Actor that contains the corresponding list of movies that the actor has starred in, and (2) a second `Map`, in which key is a movie and each value is the list of Actors in the movie (i.e., the cast). A movie is represented simply as a `String` that includes the year in which it was made, but an Actor object includes the name of the actor, and the list of movies in which the actor has starred. The resulting class should look like this:

```
public class ActorsDB {
  private static final class Actor
  {
     String name;

     // list of movies for this actor/actress
     ArrayList aList = new ArrayList();
```

```
    int    numSM = 0;  // to store temporary data, if needed

    public void Actor (String s)
       { name = s; }

    public String toString( )
       { return name; }

    public int hashCode( )
      { return name.hashCode( ); }

    public boolean equals( Object other )
      { return (other instanceof Actor) &&
               ( (Actor) other ).name.equals( name ); }
  }

  // Open fileName; update the maps
  public void loadFile( String fileName ) throws IOException
    { ... }

  // find movies common to stars in list actorList
  public void findCommonMovies( ArrayList actorList )
    { ... }

  // maps an actor name to an Actor object
  private Map actorNameToActor = new HashMap( );

  // maps a movie name to an Actor object
  private Map moviesToActor = new HashMap( );
}
```

Potential Problems

The description above is pretty much what you have to do, except that you must take extra steps to avoid running out of memory. This can be achieved by increasing the maximum size of the Java Virtual Machine from the default of 64M to 192M. You may not increase it any higher than that. If you are running the java interpreter from the command line, the magic option is -Xmx192m. If you are using an IDE, you will have to consult their documentation.

Even with the increased size of 192M you could run out of memory, because if you find two movies that are the same, but are on different input lines, the default setup will create two separate (yet equal) String objects and place them in the value lists of two different actors. Since there are 3 million roles, but only 200,000 movies, this means that you will have ten times as many `String` objects as you really need. What you need to do is to make sure that each movie title is represented by a single `String` object, and that the maps simply store references to that single `String` object. The `String` class has a method call `intern`. If you invoke it, the return value on equal `String`s will always reference the same internal `String` object.

When you maintain the list of movies for each actor, use an `ArrayList`. It takes little effort to

ensure that the capacity in the `ArrayList` is not more than is needed, and you should do so, to avoid wasting space (since there are 700,000 such array lists).

When you construct the HashMaps, you can issue a parameter (the load factor). The higher the load factor, the less space you use (at the expense of a small increase in time). You should play around with that too; the default is 0.75.

You must be very careful to avoid doing any more work in the inner loops than you need to, including creating excessive objects.

Extensions for the bored Here are some suggestions:

- Given as input an integer k and a list of stars (call it list A), find the names of all stars that have co-starred with every star in list A in at least k movies.
- Given a list of movies (call it list M), find a movie star that has acted in all of the movies in list M.
- Find the pair of actors that have co-starred in the most number of movies.
- Given a list of stars (call it list A), name the star in list A who has costarred with the largest number of Academy Award winners.