CS-141 Introduction to Programming
Creating a Simple Database

In this lab, you will implement a simple database, i.e. a program that consists of an organized collection of data together with a user interface that enables one to read, write, and access the data in multiple ways. One familiar example is The Internet Movie Database (IMDb). There are also many online music databases.

The lab will be divided into several parts:

- Part 1: Pick a subject. Build a database with a simple class structure. Add a menu-style user interface.
- **Part 2: Add the ability to read and write the data from files.**
- Part 3: Add the ability to store and search more complex data. You will need to add additional classes, modify classes from part 1, and modify the file I/O from part 2 in order to read this more complex data.

Part 2
File Input/Output

Goals: Practice reading and writing to ASCII text files.

Instructions:

1. Create an empty ASCII text file by going to the menu `File->New File`. Under `Categories`, select `Other`, and under `File Types`, select `Empty File`. The select the `Next` button, and fill in the `File Name` e.g. datafile.txt. Make sure you place the file in your project's `src` folder.
2. For each item in your database (e.g. Movie), you need to consider what data needs to be provided and how you want it listed in the file. You could place all the information for a given item on one line separated by some delimiter (e.g. space, comma, or semi-colon) assuming there is a delimiter which will never appear as part of the data. For example, movie names are often multiple words so it would not be possible to use a space as the delimiter. However, a semi-color might work. For example:

   Movie name1; year1; director1
   Movie name2; year2; director2
   etc

   If no good delimiter exists, then you could put each component on a separate line, e.g.

   Movie name1
   year1
   director1
   Movie name2
   Year2

Director2

Etc

Include enough data so that you can carefully test your program. Note, you can always add more data later.

3. The data being read is to be stored in the ArrayList contained in your Database class. It therefore makes sense to place the file IO code in this class. (Alternatively, you could create an entirely separate FileIO class but that is probably not necessary unless you find your Database class is getting really huge). Below is sample code for File IO. In lecture, we will go over how this code works and how it will need to be modified for your data. We assume the name of the data file is stored in the String member variable called `inFileName`. It is not a good idea to write the data out to the same file, so we also have an `outFileName` stored as a member variable. You only need to write the file out if you plan on allowing the user to add or remove items from the database (this is recommended!).
4. Add the code below to your Database class and modify it for your specific subject and data.
5. Test your code thoroughly.

Sample Code for File Input/Output

```
/**
  * Read state & population data from a file. Note: Assumes the
  * delimiter is a ":"
*/
public void readArrayFile() {
try {
   File infile = new File(inFileName);
   Scanner in = new Scanner(infile);
   while (in.hasNext()) {
         String line = in.nextLine();    // Read a line in the file
         String[] tokens = line.split(":"); // Split line using ":"

         // Remove extra whitespace from name
         String name = tokens[0].trim();

         // Convert the year (stored as String) to an integer:
         int population = Integer.parseInt(tokens[1].trim());

         // Create a State object with data
         State state = new State(name, population);
         stateDataArray.add(state);    // Add state to ArrayList
   }
   in.close();
} catch (FileNotFoundException e) {
   System.out.println("Input File " + inFileName + " is not found. ");
   System.exit(0);
}
}
```

```
/** Write database out to File */
// This works because the database's toString method prints the data as it
// appears in the data file.  If this is not the case, you need to provide
// for formatting code.
public void writeArrayFile() {
   try {
         System.out.println("Saving modified data to " + outFileName);
         PrintWriter out = new PrintWriter(outFileName);
         out.print(this);  // works because of how toString is written
         out.close();
        } catch (FileNotFoundException e) {
         System.out.println("Error writing to " + outFileName);
         System.exit(0);
        }
}
```