Computing and Algorithms II: Programming Assignment 1

CS-102 **Fall 2008**

In this programming assignment, you will demonstrate your knowledge of material covered in CS-101 (and re-acquaint yourself with the Java environment) by creating a simple database system for airport abbreviation codes.

File Input

Your program will accept a single argument on the command line, which is a non-empty string giving the name of a file in the current directory.

This file will contain descriptions of airport abbreviation codes. Each line of the file will contain one record of data. Each record will be represented by two strings, separated by slashes, in the following order:

1. A three-character string representing the abbreviation for a given airport (*e.g.*, "DTW", "FNT").

2. A string representing the name of the corresponding airport (*e.g.*, "Detroit, Wayne County, MI", "Flint, MI")

Your program will begin by reading in this information and storing it internally in an appropriate format. (See *Internal Requirements*.)

Interactive Input

Your program will then interact with the user running the program, offering the user a menu of commands. The following commands should be offered at this time:

- Search the database for an abbreviation. If selected, the program should ask the user for the three-character abbreviation. Using that abbreviation, the program should display the record (if present) in the database whose abbreviation *exactly matches* the user input. If no matching record can be found, an appropriate message should be displayed.

- Search the database for a city description. If selected, the program should ask the user for the city description. Using that description, the program should display *all* records whose description *begins with* the user input. (Thus, if the user types "Flint", a record with description "Flint, MI" would successfully match.) As before, if no matching record can be found, an appropriate message should be displayed.

- Print the database. If selected, the program should print the entire contents of the database in an appropriate format.

- Exit the program. If selected, the program should terminate.

Internal Requirements

As always, your program should use good style, as defined by the CS-102 Style Requirements handout.

Your program should validate input whenever possible. For example, you should check that the user provides a single argument on the command line, that database entries read from the input file meet the required format, and that the user provides valid options during the interactive phase of the program.

Your program should catch (and handle appropriately) all exceptions generated by any system routines you use, as well as any exceptions you generate yourself. (That is, your main method should not throw any exceptions.)

Your program should be designed with modifiability in mind. You will be revising and extending this program several times throughout the semester; consequently, it is to your benefit to design your program in as modular a fashion as possible. In particular:

- You should define a class `Airport` which contains members corresponding to a given airport, and methods which deal with courses (*e.g.* print, getAbbreviation).
- You should define a class `Database` which contains methods which allow one to manipulate a collection of `Airport` objects (*e.g.* search, print). The `Database` class should never access the internal members of the `Airport` class directly.

  For this assignment, your `Database` object should be designed as an unordered array of `Airport` objects. You should define the size of this array using a `final` constant and use that constant appropriately.
- You should define a class `Prog1` with a `main` method which calls the `Database` class to perform the required operations. The `Prog1` class should never access the members of the other classes directly.

You will be replacing these classes several times through the term as we learn about different data structures and algorithms. Thus, it is to your advantage to make your design as modular as possible. A little bit of extra work now will make your work simpler later.

A Sample Session

Here is a *sample* data file which could be read by this program:

```
DTW/Detroit (Wayne County), MI, USA
FNT/Flint, MI, USA
DTT/Detroit, MI, USA (City Code)
DET/Detroit (City Airport), MI, USA
YIP/Detroit (Willow Run), MI, USA
IAD/Washington (Dulles), DC, USA
WAS/Washington, DC, USA (City Code)
DCA/Washington (Reagan), DC, USA
BWI/Baltimore, MD, USA
```

And here is a *sample* interactive session:

```
nova -: java Prog1 datafile

Welcome to the CS-102 Airport Database Program.

Current available commands:
1 --> Search for an abbreviation
2 --> Search for an airport
3 --> Print all records
9 --> Exit

Your choice? 3

DTW: Detroit (Wayne County), MI, USA
FNT: Flint, MI, USA
DTT: Detroit, MI, USA (City Code)
DET: Detroit (City Airport), MI, USA
YIP: Detroit (Willow Run), MI, USA
IAD: Washington (Dulles), DC, USA
WAS: Washington, DC, USA (City Code)
DCA: Washington (Reagan), DC, USA
BWI: Baltimore, MD, USA

Current available commands:
1 --> Search for an abbreviation
2 --> Search for an airport
3 --> Print all records
9 --> Exit
```

```
Your choice? 1

Enter abbreviation: DTW

DTW: Detroit (Wayne County), MI, USA

Current available commands:
1 --> Search for an abbreviation
2 --> Search for an airport
3 --> Print all records
9 --> Exit

Your choice? 2

Enter airport name: Washington

IAD: Washington (Dulles), DC, USA
WAS: Washington, DC, USA (City Code)
DCA: Washington (Reagan), DC, USA

Current available commands:
1 --> Search for an abbreviation
2 --> Search for an airport
3 --> Print all records
9 --> Exit

Your choice? 9

nova -:
```

Notice that this is a *sample* run; your run may operate differently (and appear differently) as long as it fulfills the requirements outlined above.

You should create several input files in order to test your program thoroughly; do not assume that simply testing the program on the input file shown above is sufficient to test your program. (What happens if the input file is empty? What if it contains too many entries?)

Submitting Your Program

Before 11:59:59 p.m., Monday, 20 October 2008 (3rd Monday), you must send a MIME-encoded email message to `jhuggins@kettering.edu` containing all source code files for your program, including a class named `Prog1` containing a `main` method.

In addition, you must deliver to the instructor a printout of your program files at the start of class on Tuesday, 21 October 2008 (3rd Monday).

Notes

1. *Start Early!* The intent of this program is to make sure you are familiar with Java and our Java environment. However, there are some new features in the program (*e.g.* file input, exception handling), and some new style requirements, that may catch you if you are not careful.

2. A reminder: please read and use the class style guidelines distributed separately. 25% of your program grade is allocated for style.

3. Input from the console and from files can be conveniently handled using the `Scanner` class. See the accompanying crib for details.

4. Recall that the `main` method for a Java program begins as follows:

   ```
   public static void main(String[] args) ...
   ```

   `args` is an array of strings; the elements of this array are the command-line arguments given to this program. `args.length` gives the length of the array, *i.e.*, the number of command-line arguments supplied to the program.

5. `String` objects should be not be compared directly with the "`==`" operator. Various methods in the `String` class will assist you in performing your searches; see the Java API for details.