Project #1
Computer Science 2334
Spring 2008

User Request:

"Create a Zip Code Search Engine."

Milestones:

1	Use program arguments to specify a file name.	*10 points*
2	Use simple File I/O to read a file.	*10 points*
3	Create an ADT to store information about a zip code.	*15 points*
4	Create an ADT that abstracts the use of an array of zip codes.	*15 points*
5	Implement a program that allows the user to search the database of zip codes as described below.	*20 points*
➲	Develop and use a proper design.	*15 points*
➲	Use proper documentation and formatting.	*15 points*

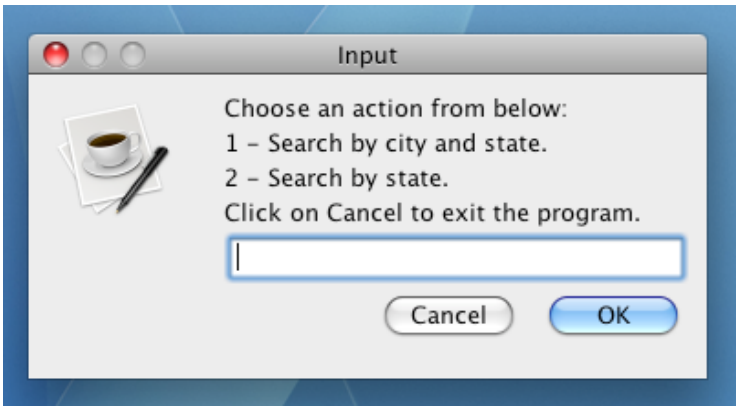
Description:

For this project, you will put together several techniques and concepts learned in CS 1323 and some new techniques to make an application that searches a large database of zip codes. This project provides the same functionality as some websites (http://www.zipinfo.com/search/zipcode.htm) that allow you to find the zip code(s) for a given city and state.

One of the best things about this project is that it uses real data (and lots of it!). There are over 40,000 zip codes in the database. To the surprise of no one, the best approach to this somewhat large problem is to decompose the problem into separate classes that can be gradually built up.

Operational Issues:

Your program will process the text file specified by the input filename (see below for information on how to read program arguments). Once the entire list of zip codes and their corresponding data have been read into your program and stored in an array, your program will display a menu to the user. You should use a *JOptionPane* to create an input dialog similar to the one shown below.



Note that if the user clicks on cancel, then the program should gracefully exit.

The basic version of the program will allow the user to perform two types of searches; you may extend the program for extra credit as discussed in the *Extra Credit* section below. The first search looks for zip codes for a city/state pair specified by the user. For example, if the user was to search for the zip codes in the city "Wahoo" and the state "Nebraska", then the results shown below are a sample of what should be displayed to the user using a *JOptionPane* message dialog.

The second type of search looks for zip codes in a specific state. The results of this type of search will also be displayed using a *JOptionPane* message dialog. Note that searches for both options should be case insensitive.

Once the user has dismissed the message dialog that shows the results of the search, the program should display the program menu. The program menu will continue to be displayed, in this fashion, followed by input and output dialogs for the chosen search, until the user clicks on Cancel, at which time the program must gracefully exit.

Zip Code Data:

The data for the zip codes is stored in a text file that you need to download from the class website. You may use Eclipse to view the contents of this file. The zip codes are listed line-by-line with the following format:

```
zip,city,state
```

Implementation Issues:

There are two Java elements in this project that may be new to some students: reading from a file, and program arguments. These Java features are summarized below.

Reading from a file:

We will discuss File I/O in depth later in the class, this project is just designed to give you a brief introduction to the technique. Reading files is accomplished in Java using a collection of classes in the java.io package. To use the classes you must import the following package:

```
import java.io.IOException;
```

The first action is to open the file. This associates a variable in the program with the name of the file sitting on the disk.

```
String fileName = "zipcode.csv";
FileReader fr = new FileReader(fileName);
```

Next the FileReader is wrapped with a BufferedReader. BufferedReaders are more efficient than FileReaders since they save groups of characters during a single operation instead of working with characters individually. Another advantage of using the BufferedReader is that there is a command to read an entire line of the file, instead of a single character at a time. This feature comes in particularly handy for this project.

```
BufferedReader br = new BufferedReader(fr);
```

The BufferedReader can now read in Strings.

```
String nextline;
nextline = br.readLine();
```

Look at the Java API listing for BufferedReader and find out what readLine() returns when it encounters the end of the file (stream). When you are finished with the BufferedReader, the file should be closed. This informs the operating system that you're finished using the file.

```
br.close();
```

Closing the BufferedReader also closes the FileReader.

Any method which performs I/O will have to throw or catch an IOException. If it is not caught, then it must be passed to the the calling method. The syntax is given below:

```
public void myMethod(int argument) throws IOException
{
        //method body here
}
```

Program Arguments:

Sometimes it is handy to be able to give a program some input when it first starts executing. Program arguments can fulfill this need. Program arguments in Eclipse are equivalent to MS-DOS or Unix command line arguments. Program arguments are handled in Java using a String array that is traditionally called args (the name is actually irrelevant.) See the slides from Lab #2 for how to supply program arguments in Eclipse.

The program below will print out the program arguments.

```
public static void main(String[] args)
{
        System.out.println(args.length +
                           " program arguments:");
        for (int i=0; i< args.length; i++)
                System.out.println("args[" + i + "] = " + args[i]);
}
```

Milestones:

A milestone is a "significant point in development." In other words, milestones serve to guide you in the development of your project. Listed below are a set of milestones for this project along with a brief description of each.

Milestone 1. Use program arguments to read in a file name.

The name of the file that stores the list of zip codes will be passed to the program using program arguments as discussed above. Type in the sample program given in the section on Program Arguments and make sure that you understand how the program arguments you provide affects the *String[] args* parameter that is passed into the main method of the program. Then, write a main method for your program that reads in the name of the data file from the program arguments.

Milestone 2. Use simple File I/O to read a file.

Before you can allow the user to search the database of zip codes, you must first be able to read a text file. Examine the section above on *Reading from a file*. A good start to the program is to be able to read in the name of

a file from the program arguments, read each line from the file, one at a time, and print each line to the console using *System.out.println()*. *Later, you will want to remove the code that prints out each line read in from the file, since the project requirements do not specify that the file is to be written out to the console as it is read.*

Milestone 3. Create an ADT to store information about a zip code.
You must create a class that holds the information related to a single zip code in the database before you can store the information that is read in from the input file. Think about what information is associated with each zip code and how to most efficiently store the information. Also, think about any methods that may help you to manage and search the data by abstracting operations to be performed on a single entry in the database that will be used by another class.

Milestone 4. Create an ADT that abstracts the use of an array of zip codes.
You are to store the object representing each entry in the database into an array of objects. However, it is not necessary for the portions of the program that will display the program's menu of searches and carry out user actions to directly operate on this array. You should create a class that encapsulates this array and allows the addition of zip codes and their related information and also supports the required search operations on the array. This class will represent the database (or collection of information associated with the program). Think about the operations that this class needs to support and how it will use the ADT created for Milestone 3. At this point, you should be able to read in the input file and create an object for each zip code, and its associated data from the file, and store the object into the array. Note that the data file used for grading may be larger than the data file provided for testing.

Milestone 5. Implement a program that allows the user to search the database of zip codes as described below.
This is where the entire program starts to take on its final form and come together. Here you will create the input and output dialogs and the menu system. Start by creating the input dialog for the second search option, which searches according to a user-specified state, and the output dialog that outputs the search results. Tie together the input dialog, the ADT from Milestone 4, and the output dialog to make this search functional and test its functionality.

Next, do the same for the first search option, which searches by city and state.

Finally, you are ready to create the menu using an input dialog and write the main loop of the program that will take a menu option as input and invoke the correct methods that were created to test the two search options. Remember that when the user clicks on cancel in the program menu, the program must gracefully exit. This can be accomplished by using *System.exit(0)*.

How to Complete this Project:

1. Revise your UML design from Lab. Be sure to clearly write your name and "Project 1" on the top of the UML diagram. Make sure to keep a copy of this when you turn it in.
2. Create the classes and methods specified in your design, but do not put code in the methods. Add the required documentation to your classes and methods as specified in the Documentation Requirements posted on the class website. This is called "stubbing" your classes and methods.
3. Run your stubbed Java files through Javadoc as described in the Lab #2 slides. This will create a set of HTML files in a directory named "docs" under your project directory.
4. Implement the design you have developed by coding each method you have defined as well as any others you have left out of your design. As you do this, make sure to modify and annotate the changes to your design on your UML and properly document all new code. A good approach to the implementation of your project is to follow the project's milestones in the order they have been supplied.
5. Test your program and fix any bugs.

6. Once you have completed the project and are ready to submit it for grading, create a new set of Javadoc files using Eclipse and inspect them to make sure your detailed design is properly documented in your source code.

Extra Credit Features:

You may extend this project with more search features for an extra 5 points of credit. Think of ways to enable a wider range of searches to be used, such as searching based on regular expressions and searching based on how a name sounds rather than how it is spelled.
To receive the full five points of extra credit, your extended search feature must be novel (unique) and it must involve effort in the design of the integration of the feature into the project and the actual coding of the feature. Also, you must indicate on your final UML design, the portions of the design that support the extra feature(s); and you must include a write-up of the feature(s) in your milestones file. The write-up must indicate what the feature is, how it works, how it is unique, and the write-up must cite any outside resources used.

You may write your program from scratch or may start from programs for which the source code is freely available on the web or through other sources (such as friends or student organizations). If you do not start from scratch, you must give a complete and accurate accounting of where all of your code came from and indicate which parts are original or changed, and which you got from which other source. Failure to give credit where credit is due is academic fraud and will be dealt with accordingly.

Due Dates and Notes:

1. Your revised design and detailed Javadoc documentation are due on ***Thursday, February 7th.*** Submit your revised UML design ***on engineering paper*** or a hardcopy using UML layout software, a hardcopy of the Javadoc documentation and a hardcopy of the stubbed source code at the ***beginning of lab***. Submit the project archive following the steps given in the Submission Instructions by ***9:00pm***.

2. The final version of the project is due on ***Thursday, February 14th.*** Submit your final UML design ***on engineering paper*** or a hardcopy using UML layout software, a hardcopy of the Javadoc documentation and a hardcopy of the source code at the ***beginning of lab***. Submit the project archive following the steps given in the Submission Instructions by ***9:00pm***.

4. You are not allowed to use the StringTokenizer class. Instead ***you must use String.split()*** and a regular expression that specifies the delimiters you wish to use to "tokenize" or split each line of the file.