2 Lab Assignment 2

You should recognize, by looking at the sample code given so far, that making the assumption that a search on a record will be done using a `String` data item is inherently out of sync with the concept of encapsulating data and operations into abstract data types. The Right and Proper Way to do this would be to implement a `Key` class and to handle all the searches by comparing `Key` entries. This approach also allows us to change our input data to be more realistic. If you think about it, the input line that requires an entire record to be read so that the names can be compared is very artificial; if we had an entire record, we wouldn't need to look up the record. In a real situation, we will probably have only a key, such as last name or Social Security Number, and we will want to use this to do a lookup in the data to find the entire record.

Start with the source code for the `FlatFile` example in the zip file for this assignment. Your assignment is to change this code to implement a simple `Key` class that will allow you to compare an input key against a key value computed from the `Record`, but to hide from the `Record` class any information about what value is being used as the key. This way, if you wish to change the instance variable used as the key, you need only change the code in the `Key` class but do not need to make any changes to the `Record` class.

The general change you are to make to the code you download is this: You are still permitted to consider the search key to be of `String` type everywhere that a search key is to be used. This is convenient because `String` data is the most flexible kind of data you can have, since pretty much anything qualifies as a `String` (in contrast to numeric data, which must be of a certain format, cannot contain non-numeric characters except as used in scientific notation, etc.). However, the search key produced for a given data item of `Record` type is to be a string created from the `Record` item by a method in a separate `Key` class. That way, all the information about how to create a search key from an instance of a `Record` is managed by the code in the `Key` class and that information is hidden from the `Record` class.

2.1 STEP ONE

First look at the driver code of `Driver.java` and at the input data of `zin`. The previous input has been changed; it now is of the form

```
number N of records to read and add to the structure
the N records themselves, one per line
number M of search keys to be used
the M search keys themselves, one per line
```

The revised `Driver.java` code handles the reading and use of this input file. You don't have to change this code, but until you make the further changes to the rest of the code from the download, this driver program will have compile errors.

2.2 STEP TWO

You will now have to change the `FlatFile.java` code somewhat. First, a new version (or a second implementation with a different signature) of `readFile` will have to take both the count and the input file as arguments instead of just the input file.

The major change to `FlatFile.java`, however, is that you will have to include a method `findRecordWithKey` that for the most part replaces the earlier search method. This method should be declared with an opening line

```
public int findRecordWithKey(String theKey)
```

This code will still run the loop through the data array, but the comparison that tests whether the key matches should now be

```
if(this.getRecord(i).getKey().equals(theKey))
```

instead of the line that invokes the old `compareName` method.

2.3 STEP THREE

This line of code mentioned above invokes a method `getKey` in the `Record` class; that method can be in its entirety

```
public String  getKey()
{
  return Key.getKey(this);
}
```

that returns a `String` by calling the `static` method `getKey` in the `Key` class you write. You need to implement the `getKey` method in your `Record` class.

```
10
Herbert  2A41 789.0123 390
Lander   2A47 789.7890 146
Winthrop 3A71 789.4667 611
Marion   2A13 789.1536 750
Adams    3A56 789.8702 522
Smith    2A46 789.5275 101
Axolotl  3A22 789.2749 102
Zokni    3A39 789.9111 350
Igen     2A21 789.6050 240
Nem      2A89 789.3383 790
4
Smith
Buell
Lander
Axolotl
```

Figure 1: Input data for Lab 2

2.4 STEP FOUR

You now need to implement the `Key` class. This class needs to contain only the `static` method `getKey` that returns the `name` instance variable of the instance of `Record` passed to it as an argument.

Sample input and output for the program appears as the `zin` and `zout` files on the website. That data is presented in Figures 1 and 2 respectively.

```
Driver:  create the flat file and write it
Driver:  file created is
File is empty.
Driver:  empty file was created
Driver:  read 10 records
Driver:  10 records have been read
Driver:  write the file as read
Driver:  file read was
subscript 0 Herbert    2A41  789.0123   390
subscript 1 Lander     2A47  789.7890   146
subscript 2 Winthrop   3A71  789.4667   611
subscript 3 Marion     2A13  789.1536   750
subscript 4 Adams      3A56  789.8702   522
subscript 5 Smith      2A46  789.5275   101
subscript 6 Axolotl    3A22  789.2749   102
subscript 7 Zokni      3A39  789.9111   350
subscript 8 Igen       2A21  789.6050   240
subscript 9 Nem        2A89  789.3383   790
Driver:  done with the write
Driver:  search for key 'Smith'
FlatFile:  found 'Smith' after 6 probes
Driver:  name 'Smith' appears as
Smith      2A46  789.5275   101
Driver:  done with search 0
Driver:  search for key 'Buell'
FlatFile:  did not find 'Buell' after 10 probes
Driver:  name 'Buell' not in the file
Driver:  done with search 1
Driver:  search for key 'Lander'
FlatFile:  found 'Lander' after 2 probes
Driver:  name 'Lander' appears as
Lander     2A47  789.7890   146
Driver:  done with search 2
Driver:  search for key 'Axolotl'
FlatFile:  found 'Axolotl' after 7 probes
Driver:  name 'Axolotl' appears as
Axolotl    3A22  789.2749   102
Driver:  done with search 3
```

Figure 2: Output data for Lab 2