Project 4:
New York City Restaurant Inspection Results
Using BST to Complete Some of the Tasks
Due date: April 23, 11:55PM EST.

You may discuss any of the assignments with your classmates and tutors (or anyone else) but all work for all assignments must be entirely your own. Any sharing or copying of assignments will be considered cheating. If you get significant help from anyone, you should acknowledge it in your submission (and your grade will be proportional to the part that you completed on your own). You are responsible for every line in your program: you need to know what it does and why. You should not use any data structures and features of Java that have not been covered in class (or the prerequisite class). If you have doubts whether or not you are allowed to use certain structures, just ask your instructor.

In this project you will continue working on the code that you developed for project 1. The objective in this project is to improve the performance of the sorting algorithms.

Objectives
The goal of this programming project is for you to master (or at least get practice on) the following tasks:
• working with code developed previously,
• implementing more advanced data structures: binary search tree,
• using the implemented binary search tree to improve the performance of the program.
Your implementation of the code for projects 1 and 2 used linear structure of an array to store all of the inspection records. In this project you should develop your own binary search tree (BST) data structure and use it to store the inspection records when performing some of the tasks. Note that the BST object should not be a complete replacement for the MyArrayList class that you wrote for project 1 (for certain tasks, using BST would not be beneficial in terms of performance).

Program Input and Output
The inputs and outputs for this project are exactly the same as in project 2.

Data Storage and Organization
In this project, you will be modifying the implementation of your findByName method in the ListOfRecords class.

Computational Task
In project 1, your implementation of the findByName method might have looked something like this:

```java
public MyArrayList findByName(String key) {
    MyArrayList results = new MyArrayList();
```
Record tmp = null;
for (int i = 0; i < list.size(); i++) {
    tmp = list.get(i);
    if (tmp.getDba().equalsIgnoreCase(key)) {
        results.add(tmp);
    }
} 
return results;

If we could organize all the restaurant inspection records so that all of the records matching the name of a given restaurant were kept "near" one another, then we could avoid having to search through all of the records when `findByName` command is executed.

Consider a binary search tree that is organized by the name of the restaurant. Since the data set contains multiple records with the same name of the restaurant a node of such tree should be capable of storing all such records. In this case, when we need to complete the `findByName` command, we simply need to locate the node that stores a particular DBA and return all of the records stored in that node.

The "data" part of such node should be defined by a class that stores a short list of records all of which have the same DBA field. For example:

```java
public class DbaList implements Comparable<DbaList>{
    String dba;
    List<Record> list;

    public DbaList( Record r ) throws IllegalArgumentException {
        if ( r == null ) throw new IllegalArgumentException ("Error: cannot create " + "DbaList with a null Record object.");
        dba = r.getDba();
        list = new LinkedList<Record>();
        list.add(0, r);
    }
    public DbaList(String dba) {
        this.dba = dba;
        list = new LinkedList<Record>();
    }
    public String getDba () {
        return dba;
    }
    public List<Record> getList () {
        return list;
    }
    public boolean add ( Record r ) {
        if (r.getDba().equalsIgnoreCase( dba )) {
            list.add(0,r);
            return true;
        }
        return false;
    }
```

In this project, you should implement a special kind of a binary search tree that is capable of storing and manipulating objects of type `DbaList`. (You may use the above class as is or you may add/change any parts of it. If you chose to use the class, you need to document it.)

**Classes and Data Structures**

All of your classes should be the same in this project as in the first two projects (although you should fix the problems that you might have discovered after submission of the previous project.)

In addition, you need to implement your own binary search tree data structure. **All of the methods listed below should be implemented using recursion.** Any additional methods can be implemented using either recursion or iterations.

Name your class `bstOfRecordsByDBA`. It should implement the following informal interface:

- `bstOfRecords ( )` - creates an empty tree;
- `DbaList get ( String dba )` - returns a reference to the `DbaList` containing all records in the tree that match given `dba`, or null if no such records exist in the tree;
- `void add ( Record r )` - adds a give record to this tree; the record should be added to a node that contains a `DbaList` object with a matching name; if no such node exists, a new node should be created and added to this tree;
- `DbaList remove ( String dba )` - removes and returns a `DbaList` object that contains all of the records matching a given `dba`; returns null if no such record exists.

You will also need to provide a class that represents nodes of a binary search tree.

**Programming Rules**

You should follow the rules outlined in the document *Code conventions* posted on the course website at [http://cs.nyu.edu/~joannakl/cs102.03_s16/notes/CodeConventions.pdf](http://cs.nyu.edu/~joannakl/cs102.03_s16/notes/CodeConventions.pdf).

You must document all your code using Javadoc. Your class documentation needs to provide a description of what it is used for and the name of its author. Your methods need to have description, specification of parameters, return values, exceptions thrown and any assumptions that they are making.

Classes’ data fields and methods should not be declared `static` unless they are to be shared by all instances of the class or provide access to such data.

**Working on This Assignment**

You should start right away! There is not much code to be written for this assignment, but you should allow sufficient time for debugging, testing, and writing of the report.

You should backup your code whenever you spend some time working on it. Save it to a flash drive, email it to yourself, upload it to your Google drive, do anything that gives you a second (or maybe third copy). Computers tend to break just a few days or even a few hours before the due dates - make sure that you have working code if that happens.
Grading

Make sure that you are following all the rules in the Programming Rules section above.

You can use the tests that were posted for the project 1 to test correctness of your program.

If your program does not compile or if it crashes (almost) every time it is run, you will get a zero on the assignment.

If the program does not adhere to the specification, the grade will be low and will depend on how easy it is to figure out what the program is doing.

50 points  implementation of the binary search tree class (this includes the node class)

30 points  implementation and changes to the rest of the code (specifically theListOfRecords class) to incorporate the use of the binary search tree.

20 points  proper documentation and program style (note: you may need to update some of your documentation from project 1 to reflect the changes)

How and What to Submit

You should submit all of your source code files (the ones with .java extensions only) in a single zip file to NYU Classes.

If you wish to use your (one and only) freebie for this project (one week extension, no questions asked), then complete the form at http://goo.gl/forms/rCKfB6ioCX before the due date for the assignment. All freebies are due seven days after the original due date and should be submitted to NYU Classes.