ASSIGNMENT 2: ORGANIZATIONAL HIERARCHY

Goal: The goal of this assignment is to get some practice with trees and search trees. Search Trees are one of the most important data structures we will study in this class – they are regularly used in large database systems for storing indexes on the records.

Problem Statement: We want to maintain the list of employees in a company. We will be concerned with two quantities associated with each employee in the company -- ID of the employee (you can assume no two employees in the company have the same ID), and the level of the employee. The level denotes where the person stands in the hierarchy. Level 1 denotes the highest post in the company (say the owner), level 2 comes below level 1 and so on. There is only 1 person at level 1, but there can be several employees at level i > 1. Each level i employee works under a level i-1 employee, which is his/her immediate boss. Given an employee A, we can form a sequence of employees A',A'', A''', ... where A works under A', A' works under A'', and so on. We say that each employee in A',A'', A''',... is a boss of A. We would like to implement a suitable tree-based data-structure so that we can implement the following operations :

```
public interface OrgHierarchyInterface {

        public boolean isEmpty();  /* Returns true if the organization is empty. */
        public int size();  /* Returns the number of employees in the organization */
        public int level(int id) throws IllegalIDException, EmptyTreeException;  /* Returns the level
of the employee with ID=id */

        public void hireOwner(int id) throws NotEmptyException, EmptyTreeException;  /* Adds the first
employee of the organization, which we call the owner. There is only one owner in an org who cannot be
deleted. */
        public void hireEmployee(int id, int bossid) throws IllegalIDException, EmptyTreeException;  /*
Adds a new employee whose ID is id. This employee will work under an existing employee whose ID is
bossid (note that this automatically decides the level of id, it is one more than that of bossid). */
        public void fireEmployee(int id) throws IllegalIDException, EmptyTreeException;  /* Deletes an
employee who does not manage any other employees. Note that this can not be the owner. If it is the
owner, throw the IllegalIDException. */
        public void fireEmployee(int id, int manageid) throws IllegalIDException, EmptyTreeException;
/* Deletes an employee (id) who might manage other employees. Manageid is another employee who works at
the same level as id. All employees working under id will now work under manageid. Note that this can
not be the owner. If it is the owner, throw the IllegalIDException. */

        public int boss(int id) throws IllegalIDException, EmptyTreeException;  /* Returns the id of
the immediate boss, the employee. Output -1 if id is the owner's ID  */
        public int lowestCommonBoss(int id1, int id2) throws IllegalIDException, EmptyTreeException;
/* Returns the ID of the employee A who is a boss of both id1 and id2, and among all such persons has
the largest level. In other words, we want to find the common boss who is lowest in the hierarchy in
the company. If one of the input ids is the owner, output -1 */

        public String toString(int id) throws IllegalIDException, EmptyTreeException; /* This method
returns a string that contains the IDs of all the employees of the organisation rooted at id. It should
contain the employees level-wise. So first it should have id, then ids of all the employees directly
under id, and then all employees directly these employees and so on. Make sure that in the string
levels are comma separated and employees inside a level are space separated. Among employees at the
same level, the output should be sorted in increasing order of the ids.*/
}
```

Write a program which implements such a data-structure. Credit will be given to choice of proper data-structures and efficiency. Whenever possible, **implement each method in no more than O(log(n))** where n is the number of employees in the organization. If some methods cannot be implemented in O(log(n)), have a justification ready for why it is not possible.

Input Output:

The input output format is as follows: You need to add your code to OrgHierarchy.java file. DO NOT edit other files. You can make as many classes in the OrgHierarchy.java file. You are also given a tester.java file which creates a tree and calls the functions in **OrgHierarchyInterface** class. After you are done with your implementation, you can use this file for testing your code on different trees.

Here is an example of how the following tree is built :-

```
/*
  Tree-
            1
          /   |   \
        3     2    12
       / \    |    |
      8   4   7    10
     / \
    16  5
*/

[In] : hireOwner(1)
...    [1]
[In] : hireEmployee(3,1)
...    [1]
…     /
… [3]
[In] : hireEmployee(2,1)
…         [1]
…     /       \
… [3]          [2]
[In] : hireEmployee(12,1)
…            [1]
…     /       |      \
… [3]       [2]      [12]
[In] : hireEmployee(8,3)
…                  [1]
…         /         |         \
…      [3]         [2]         [12]
…     /
…   [8]


[In] : hireEmployee(4,3)
…                  [1]
…         /         |         \
…      [3]         [2]         [12]
…     /   \
```

```
…  [8]    [4]
[In] : hireEmployee(7,2)
…                [1]
…        /        |        \
…     [3]        [2]        [12]
…    /   \        |
…  [8]    [4]    [7]
[In] : hireEmployee(10,12)
…                [1]
…        /        |        \
…     [3]        [2]        [12]
…    /   \        |           |
…  [8]    [4]    [7]         [10]
[In] : hireEmployee(16,8)
…                      [1]
…              /        |        \
…           [3]        [2]        [12]
…          /   \        |           |
…       [8]    [4]     [7]         [10]
…      /
…    [16]
```

```
[In] : hireEmployee(5,8)
…                      [1]
…              /        |        \
…           [3]        [2]        [12]
…          /   \        |           |
…       [8]    [4]     [7]         [10]
…      /  \
```

… **[16] [5]**

[In] : **size()**

[Out] : 10

[In] : **toString**(1)

[Out] : 1,2 3 12,4 7 8 10,5 16

… (Note that there is no space at the start or end of the string, the comma has no space before or after, integers are level wise sorted in ascending order and separated by a single space. You have to ensure this is exactly what your function returns.)

[In] : **level**(5)

[Out] : 4

[In] : **fireEmployee**(7)

… **[1]**

… **/ | **

… **[3] [2] [12]**

… **/ \ |**

… **[8] [4] [10]**

… **/ **

… **[16] [5]**

[In] : **fireEmployee**(8, 4)

… **[1]**

… **/ | **

… **[3] [2] [12]**

… **| |**

… **[4] [10]**

… **/ **

… **[16] [5]**

[In] : **boss**(5)

[Out] : 4

```
[In] : toString(1)

[Out] : 1,2 3 12,4 10,5 16

[In] : toString(3)

[Out] : 3,4,5 16

[In] : lowestCommonBoss(3, 10)

[Out] : 1
```

What is being provided?

Your assignment folder contains 6 java files:

1. EmptyTreeException.java : Defines a custom exception (Do not modify this file)
2. IllegalIDException.java: Defines a custom exception (Do not modify this file)
3. NotEmptyException.java: Defines a custom exception (Do not modify this file)
4. OrgHierarchyInterface.java: Defines the organization hierarchy interface (Do not modify this file)
5. OrgHierarchy.java : You have to implement all the functions of OrgHierarchyInterface in this file and you can add your own classes and methods also.
6. Tester.java: File with the main function that tests all the methods implemented in the OrgHierarchy.java file.

How to run?

The starter folder contains a Makefile with the necessary commands to run the program, if you are unable to run the makefile just copy-paste the commands in the all: section and run sequentially in the shell.

What to submit?

1. Submit your code in a .zip file named in the format **<EntryNo>.zip.** Make sure that when we unzip your file a folder named <your_entry_number> should yield, in addition to the Starter folder, a "writeup.txt" file should be produced.
2. In summary if your entry number is 2016CS50393, then you need to submit a zip file 2016CS50393.zip. This zip file on extraction gives a directory structure as follows:

    2016CS50393

    -------Starter (Contains classes folder and all java files)

    -------writeup.txt

You will be penalized for any submissions that do not conform to this requirement.

1. The writeup.txt should have a line that lists names of all students you discussed/collaborated with (see guidelines on collaboration vs. cheating on the course home page). If you never discussed the assignment with anyone say None.

After this line, you are welcome to write something about your code, though this is not necessary.

Evaluation Criteria

The assignment is worth 6 points. Your code will be autograded at the demo time against a series of tests. Four points will be provided for correct output subject to a verification of efficient implementation, and two points will be provided for your explanations of your code and your answering demo questions appropriately.

What is allowed? What is not?

1. This is an individual assignment.
2. Your code must completely be your own. You are not to take guidance from any general purpose code or problem specific code meant to solve these or related problems. Remember, it is easy to detect this kind of plagiarism.
3. You are not allowed to use built-in (or anyone else's) implementations of trees, nodes or other basic data structures. A key aspect of the course is to have you learn how to implement these data structures. You are, however, required to use Java's built in java.Util.Vector class.
4. You should develop your algorithm using your own efforts. You should not Google search for direct solutions to this assignment. However, you are welcome to Google search for generic Java-related syntax.
5. You must not discuss this assignment with anyone outside the class. **Make sure you mention the names in your write-up in case you discuss with anyone from within the class.** Please read academic integrity guidelines on the course home page and follow them carefully.
6. Your submitted code will be automatically evaluated against another set of benchmark problems. You get a significant penalty if your output is not automatically parsable and does not follow input-guidelines.
7. We will run plagiarism detection software. Anyone found guilty will be awarded a suitable penalty as per IIT rules.