P09 Patient Record System

Overview

This assignment involves the implementation of a simple Patient Record System using Binary Search Trees (BST). A patient record system is usually located within a health care provider setting. It is mainly dedicated to collecting, storing, retrieving, and making available clinical information important to the delivery of patient care. Our BST will store a set of patient records where the lookup key information is the date of birth of the patient. You are going to learn how BSTs can be used to facilitate insertion and retrieval operations to and from a collection of ordered elements, in an easy and elegant way.

Learning Objectives

The goals of this assignment include:

- Implement common Binary Search Tree (BST) operations.
- Gain more Practice in recursive problem-solving.
- Gain more experience with developing unit tests.

Grading Rubric

5 points	**Pre-Assignment Quiz:** The P9 pre-assignment quiz is accessible through Canvas before having access to this specification by 11:59PM on Sunday 04/12/2020. Access to the pre-assignment quiz will be unavailable passing its deadline.
15 points	**Immediate Automated Tests:** Upon submission of your assignment to Gradescope, you will receive feedback from automated grading tests about whether specific parts of your submission conform to this write-up specification. If these tests detect problems in your code, they will attempt to give you some feedback about the kind of defect that they noticed. Note that passing all of these tests does NOT mean your program is otherwise correct. To become more confident in this, you should run additional tests of your own.
30 points	**Manual Grading and Supplemental Automated Tests:** When your final grade feedback appears on Gradescope, it will include the feedback from these additional automated grading tests, as well as feedback from human graders who review the code in your submission by hand.

Assignment Requirements and Reminders

- DO NOT submit the provided `PatientRecord.java` and `PatientRecordNode.java` source files on Gradescope and DO NOT make any change to their implementations.
- You ARE NOT allowed to add any additional import statements to the provided source files. In particular, you are not allowed to import java.util.List, or java.util.ArrayList, or java.util.Arrays classes.
- You ARE NOT allowed to add any fields either instance or static, and any public methods either static or instance to your `PatientRecordTree` class.
- You CAN define local variables that you may need to implement the methods defined in this program.
- You CAN define private methods to help implement the different public methods defined in this programming assignment, if needed.
- ALL your test methods MUST be implemented in your `PatientRecordTreeTester` class.
- In addition to the required test methods, we HIGHLY recommend (not require) that you develop your additional own unit tests (**public static methods that return a boolean**).
- Ensure that your code for every assignment is styled in conformance to CS300 Course Style Guide.

1 Getting Started

Start by creating a new Java Project in eclipse. You may call it P09 Patient Record System, for instance or any other name at your convenience. As the previous assignments, make sure that your new project uses Java 11, by setting the "Use an execution environment JRE:" drop down setting to "JavaSE-11.0.X" within the new Java Project dialog box. Then, download and add these provided PatientRecord.java, and PatientRecordNode.java source files to your project.

Read carefully through the provided source codes details. The `PatientRecord` class represents a patient's record in our Patient Record System. The patient record should be the principal repository for information concerning a patient's health care. For simplicity, we consider only two data fields in that class (the name of the patient and their date of birth). We do not consider the information related to the patient healthcare and clinical information. Notice carefully that the instance field representing the date of birth of the patient is final. This means that once initialized in the constructor, you cannot change it. We are going to use

this field as lookup key in our Patient Record System. Notice also that the `PatientRecord` class implements the java.util.Comparable interface. You can call the instance method `.compareTo())` to compare two patient records with respect to their DATE_OF_BIRTH instance fields.

The provided `PatientRecordNode` class models the binary nodes which will be used to build and implement our Binary Search Tree (BST) called `PatientRecordTree`. The specification of this class is provided in the following section.

2 PatientRecordTree and PatientRecordTreeTester Classes

We provide you in the following with the templates for the PatientRecordTree and PatientRecordTreeTester classes. Download those two files and add them to your project. Notice that we added default return statements to the incomplete methods to simply let the code compile. You are going to complete the implementation of all the methods defined in these classes and including the `TODO` tag with respect to the details provided in their javadoc method headers. **Make sure to remove the TODO tags once you complete the implementation of each method.**

Recall that you ARE NOT allowed to add any additional fields either instance or static fields to the `PatientRecordTree`. Besides, NO additional public methods must be added to this class. Read carefully all the details provided in the javadoc method headers. If a method is described to be a *recursive helper* method, you are not allowed to implement it using iteration (for or while loops). They MUST be designed and implemented using recursion.

Note that even though it is recommended to declare all the helper methods to be private, we set some of them to be public so that we can call them from our automated test methods.

You can compare two PatientRecord objects by calling the `PatientRecord.compareTo()` method with accordance to its specification. The provided implementation of that method relies on the java.util.Date.compareTo() method. PatientRecords with duplicate dates of birth are not allowed in this system. Note also that in our PatientRecordTree binary search tree, the increasing order of patient records goes from the oldest to the youngest patient. All the records of patients older than any parent must be in the left subtree rooted at its left child. Whereas, all the records of patients younger than any parent must be in the right subtree rooted at its right child.

Finally, we highlight that you are responsible for testing thoroughly your implementation of the PatientRecordTree public methods. You have to define and implement at least the FIVE unit test methods defined in the PatientRecordTreeTester class. Make sure to test every method with at least 3 different scenarios. Hints are provided in the provided javadocs method headers.

We note also that you are not required to worry if the user tries to add a patient record with a date of birth projected in the future or with a very old date of birth.

Illustrative Example

In order to provide you with a better understanding on how to use the implemented classes, we provide in the following an example of source code and its expected output.

```
PatientRecordTree bst = new PatientRecordTree();
System.out.println("Size: " + bst.size() + " Height: " + bst.height() + " Contents:");
System.out.println(bst);
bst.addPatientRecord(new PatientRecord("Norah", "11/23/1985"));
bst.addPatientRecord(new PatientRecord("George", "5/27/1943"));
System.out.println("*****************************************************");
System.out.println("Size: " + bst.size() + " Height: " + bst.height() + " Contents:");
System.out.println(bst);
bst.addPatientRecord(new PatientRecord("Adam", 8, 12, 1972));
System.out.println("*****************************************************");
System.out.println("Size: " + bst.size() + " Height: " + bst.height() + " Contents:");
System.out.println(bst);
System.out.println("Oldest patient: " + bst.getRecordOfOldestPatient());
System.out.println("Youngest patient: " + bst.getRecordOfYoungestPatient());
bst.addPatientRecord(new PatientRecord("William", 6, 4, 1998));
bst.addPatientRecord(new PatientRecord("Sarah", 1, 2, 1945));
System.out.println("*****************************************************");
System.out.println("Size: " + bst.size() + " Height: " + bst.height() + " Contents:");
System.out.println(bst);
System.out.println("Oldest patient: " + bst.getRecordOfOldestPatient());
System.out.println("Youngest patient: " + bst.getRecordOfYoungestPatient());
bst.addPatientRecord(new PatientRecord("Andrew", 4, 20, 2019));
bst.addPatientRecord(new PatientRecord("Tom", 1, 2, 1935));
bst.addPatientRecord(new PatientRecord("Sam", 9, 12, 2003));
bst.addPatientRecord(new PatientRecord("Emily", 2, 28, 2020));
System.out.println("*****************************************************");
System.out.println("Size: " + bst.size() + " Height: " + bst.height() + " Contents:");
System.out.println(bst);
System.out.println("Oldest patient: " + bst.getRecordOfOldestPatient());
System.out.println("Youngest patient: " + bst.getRecordOfYoungestPatient());
System.out.println("*****************************************************");
try {
  System.out.println("Lookup query: search for the patient who's born on 9/12/2003.");
  System.out.println("Search Results: " + bst.lookup("9/12/2003"));
  System.out.println("Lookup query: search for the patient who's born on : 1/10/2000.");
  System.out.println("Search Results: " + bst.lookup("1/10/2000"));
} catch (NoSuchElementException e) {
  System.out.println(e.getMessage());
}
```

Expected output:

```
Size:  0 Height:  0 Contents:

**************************************************
Size:  2 Height:  2 Contents:
George(5/27/1943)
Norah(11/23/1985)

**************************************************
Size:  3 Height:  3 Contents:
George(5/27/1943)
Adam(8/12/1972)
Norah(11/23/1985)

Oldest patient:  George(5/27/1943)
Youngest patient:  Norah(11/23/1985)
**************************************************
Size:  5 Height:  4 Contents:
George(5/27/1943)
Sarah(1/2/1945)
Adam(8/12/1972)
Norah(11/23/1985)
William(6/4/1998)

Oldest patient:  George(5/27/1943)
Youngest patient:  William(6/4/1998)
**************************************************
Size:  9 Height:  4 Contents:
Tom(1/2/1935)
George(5/27/1943)
Sarah(1/2/1945)
Adam(8/12/1972)
Norah(11/23/1985)
William(6/4/1998)
Sam(9/12/2003)
Andrew(4/20/2019)
Emily(2/28/2020)

Oldest patient:  Tom(1/2/1935)
Youngest patient:  Emily(2/28/2020)
**************************************************
Lookup query:  search for the patient who's born on 9/12/2003.
Search Results:  Sam(9/12/2003)
Lookup query:  search for the patient who's born on :  1/10/2000.
No search results.
```

3 Assignment Submission

Congratulations on finishing this CS300 assignment! After verifying that your work is correct, and written clearly in a style that is consistent with the CS300 Course Style Guide, you should submit your final work through gradescope.com. The only 2 files that you must submit include: `PatientRecordTree.java` and `PatientRecordTreeTester.java`. Your score for this assignment will be based on your **"active"** submission made prior to the hard deadline of Due: **9:59PM on April 15**th. The second portion of your grade for this assignment will be determined by running that same submission against additional offline automated grading tests after the submission deadline. Finally, the third portion of your grade for your submission will be determined by humans looking for organization, clarity, commenting, and adherence to the CS300 Course Style Guide.