P09 Camp Badger

Overview

The staff of UW Madison's Recreation & Wellbeing Division have decided to start up a new overnight summer camp for kids age 8-14 named "Camp Badger". Hearing of your awesome programming skills, they have decided to come to you for assistance. You and your team are tasked with creating an enrollment application that uses a binary search tree to manage campers as well as provide some other basic functionalities such as assigning cabins and general statistics. You are allowed (and encouraged) to have one other person work on this project with you. As usual, you must indicate your partner and vice versa on the pre-assignment quiz.

Learning Objectives

The goals of this assignment include implementing and using a binary search tree as well as re-enforcing previous course topics.

Grading Rubric

5 points	**Pre-Assignment Quiz:** The P9 pre-assignment quiz is accessible through Canvas before having access to this specification by 9:59PM on Sunday 11/17/2019. Access to the pre-assignment quiz will be unavailable after its deadline.
20 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 of this, you should run additional tests of your own.
15 points	**Additional Automated Tests:** When your manual grading feedback appears on Gradescope, you will also see the feedback from these additional automated grading tests. These tests are similar to the Immediate Automated Tests, but may test different parts of your submission in different ways.
10 points	**Manual Grading Feedback:** After the deadline for an assignment has passed, the course staff will begin manually grading your submission. We will focus on looking at your algorithms, use of programming constructs, and the style and readability of your code. This grading usually takes about a week from the hard deadline, after which you will find feedback on Gradescope.

1 Getting Started

Start by creating a new Java Project in eclipse called P09 Camp Badger, for instance. You have to ensure that your new project uses Java 8, by setting the "Use an execution environment JRE:" drop down setting to "JavaSE-1.8" within the new Java Project dialog box.

2 Camper Class

In your project you will have a java class of the name `Camper` that uses the Comparable interface to represent a camper. This interface will allow us to define how the program will compare one Camper to another. Start with the code provided Camper.java. You will need to implement the compareTo() method required by the interface before moving onto the next step.

3 Camper Binary Search Tree: Set-Up

Now we will begin creating our binary search tree for these campers. First and foremost you will need a Java class that represents the nodes of your tree. Add this CampTreeNode.java to your project. Read it over and understand how it works as it is important to writing your binary search tree. You are not allowed to add any other instance fields to this class.

Once you have added that class, create a new Java class this time by the name of `CamperBST`. This class will be an implementation of a binary search tree that uses campers. It should begin with two instance fields: a public `CampTreeNode` called root and a private `int` called size. You are not allowed to add any other instance fields to this class unless specified otherwise by the project specifications. Implement a constructor for the `CamperBST` class that will create an empty binary search tree along with the two methods described below.

```java
public int size();//returns the current size of the CamperBST
public boolean isEmpty(); //returns true if the tree is empty, false otherwise
```

Create another new Java class by the name of `CampManager`. This class will be responsible for having an instance of a `CamperBST` and calling the various operations on it along with some other basic tasks in regards to managing Camp Badger. For now just give it the following fields and implement both the constructor and printStatistics() method as described in this Javadoc.

```java
private CamperBST campers;
private final static String [] CABIN_NAMES = new String[]{
"Otter Overpass", "Wolverine Woodland", "Badger Bunkhouse"};
```

Finally create another new Java class this time by the name of `CampEnrollmentApp`. The purpose of this class is to use an instance of the `CampManager` to execute certain commands as read from a text file. This class will have a main method that will sometimes throw an IOException. For the time being, in the main, create an instance of the `CampManager` class and add the following line code that will return each line of the text file as a string. Write code to parse each line of the file and perform the designated action given the command. In order to use the code provided for the `CampManager` class you will need to import `java.nio.file.Files`, `java.nio.file.Paths`, and `java.util.List`. The List class is a parent class to ArrayList and functions similarly. For now just get the "Statistics" command working. (See the end of this doc for the list of commands and sample output.) Later you will implement the functionality to parse the other commands and execute them.

```
List <String> fileLines = Files.readAllLines(Paths.get("sim.txt"));
```

4 Enrolling Campers: Binary Search Tree Insertion

In your `CamperBST` class implement the recursive approach to inserting nodes into a binary search tree. You may work under the assumption that two campers with exactly the same first and last name will not be added to the tree. Here's some boilerplate code to get you started!

```
//starts tree insertion by calling insertHelp() on the root and
//assigning root to be the subtree returned by that method
public void insert(Camper newCamper)
{root = insertHelp(root, newCamper);}

/** Recursive helper method to insert.
* @param current, The "root" of the subtree we are inserting into,
*  ie the node we are currently at.
* @param newCamper, the camper to be inserted into the tree
* @return the root of the modified subtree we inserted into
*/
private CampTreeNode insertHelp(CampTreeNode current, Camper newCamper)
{ /*YOUR RECURSIVE IMPLEMENTATION OF INSERT HERE*/ }

//Prints the contents of this tree in alphabetical order
//based on the string "lastName, firstName"
public void print() { printHelp(root); }
private void printHelp(CampTreeNode current){
   if(current==null) {return;}
   printHelp(current.getLeftNode());
   System.out.println(current.getData());
   printHelp(current.getRightNode());}
```

Once you have done this return to the `CampManager` class and implement the enrollCamper() method as defined in the Javadoc for the class. Campers of the ages 8-9 should be in "Otter Overpass", ages 10-12 in "Wolverine Woodland", and ages 13-14 in "Badger Bunkhouse" After that return to the `CampEnrollmentApp` and implement the "Enroll" command. You can use this as a tool to test insertion (and other operations) by editing the commands in "sim.txt".

5 Unenrolling Campers: Binary Search Tree Deletion

In your `CamperBST` class implement the recursive approach to deleting nodes from a binary search tree. (**Hint:** In the case where a node has two children, it may be helpful to write a recursive helper method to find it's successor.) Here's some boilerplate code to get you started!

```
/** Deletes a Camper into the binary search tree if it exists.
* @param key, the camper to be deleted from the tree
* @throws NoSuchElementException if it is thrown by deleteHelp
*/
public void delete (Camper key)
   throws NoSuchElementException
{root = deleteHelp(root, key);}


/** Recursive helper method to delete.
* @param current, The "root" of the subtree we are deleting from,
*  ie the node we are currently at.
* @param key, the camper to be deleted from the tree
* @return the root of the modified subtree we deleted from
* @throws NoSuchElementException if the camper is not in the tree
*/
private CampTreeNode deleteHelp(CampTreeNode current, Camper key)
{ /*YOUR RECURSIVE IMPLEMENTATION OF DELETE HERE*/ }
```

Once you have done this return to the `CampManager` class and implement the unenrollCamper() method as described in the Javadoc for the class. After that, return to the `CampEnrollmentApp` and edit it to support the "Unenroll" command.

6 Printing Campers: Binary Search Tree Traversals

In your `CamperBST` class, implement the recursive approach to traversing nodes in a binary search tree given one of the three traversal orders: PREORDER, POSTORDER, INORDER.

You should import and use the Java LinkedList class. Here's some boilerplate code to get you started!

```java
//LinkedList to maintain current traversal
private LinkedList<Camper> traversedLList;

//returns an iterator of camper in the correct order as designated
public Iterator<Camper> traverse(String order)
{
   //first time traversing need to initialize LinkedList
   if(traversedLList==null){
      traversedLList = new LinkedList<Camper>();
   }
   else{
      //clear the list to start over for a new traversal
      traversedLList.clear();
   }
   traverseHelp(root, order);
   return traversedLList.listIterator();
}
/** Recursive helper method to traverse. Will take the current CampTreeNode's data
*and add it to traversedLList based on the given order. Then continue to recurse on
*the correct subtree.
* @param current, the root of the current subtree we are traversing
* @param order, the type of traversal to perform
*/
private void traverseHelp (CampTreeNode current, String order)
{/*YOUR IMPLEMENTATION OF TRAVERSE HERE*/}
```

Once you have done this return to the `CampManager` class and implement the traverse() method as described in the Javadoc for the class. After that, return yet again to the `CampEnrollmentApp` and edit it to support the "Traverse" command.

7 Assignment Submission

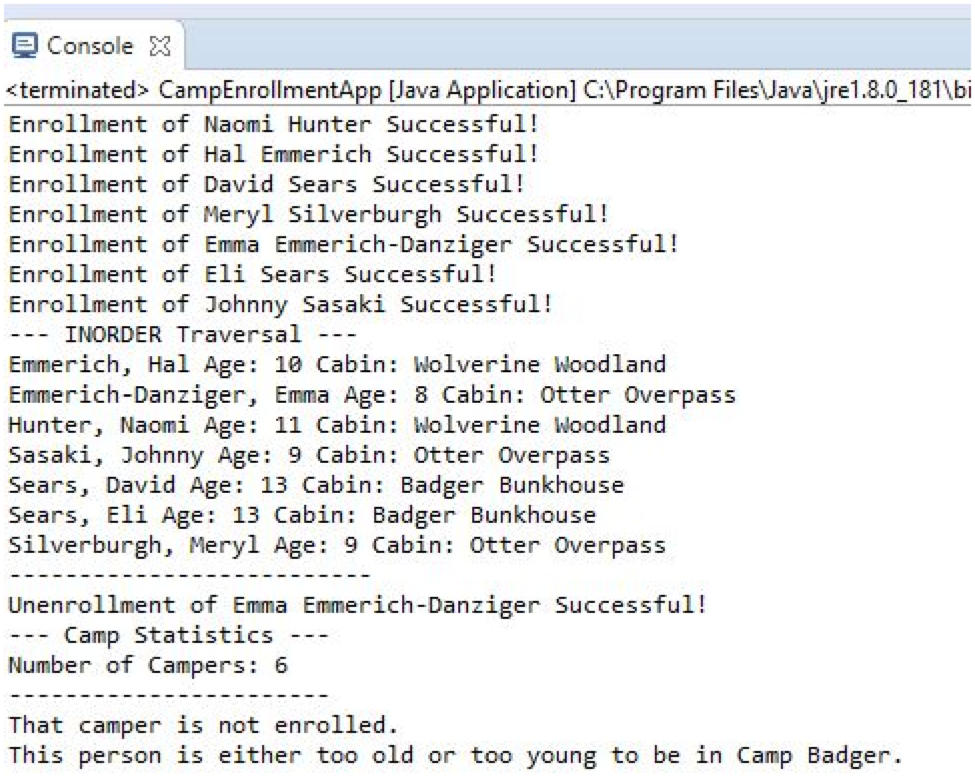
Once you have completed the project (which includes testing your code thoroughly) submit only the `Camper.java`, `CampTreeNode.java`, `CampEnrollmentApp.java`, `CamperManager.java`, and `CamperBST.java` source files to Gradescope on or before the deadline. Remember to give your files and classes headers, methods proper Javadoc headers and comments and any other specifics that are given by the CS300 Course Style Guide. Your score for this assignment will be based on your **"active"** submission made prior to the hard deadline of Due: **9:59PM on November 20th**. 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.

Application Commands & Sample Output

For the sake of simplicity you can work under the assumption that everything in a command is separated by only one space. You may also assume that each command has all of the required arguments entered correctly. Note that the order of a traversal will be in all caps. (Ex. PREORDER) Here are the commands that the `CampEnrollmentApp` class will support:

- Enroll → E <lastName> <firstName> <age>
- Unenroll → R <lastName> <firstName>
- Traverse → T <order>
- Statistics → S

This text file yields the output below when running the program and is a good baseline to see if your program is running correctly. The print statements should be handled in the `CampEnrollmentApp` class for "Enroll", "Uneroll", and "Traverse" commands while the `CampManager`class handles print statements for the "Statistics" command.


Console
<terminated> CampEnrollmentApp [Java Application] C:\Program Files\Java\jre1.8.0_181\bi

```
Enrollment of Naomi Hunter Successful!
Enrollment of Hal Emmerich Successful!
Enrollment of David Sears Successful!
Enrollment of Meryl Silverburgh Successful!
Enrollment of Emma Emmerich-Danziger Successful!
Enrollment of Eli Sears Successful!
Enrollment of Johnny Sasaki Successful!
--- INORDER Traversal ---
Emmerich, Hal Age: 10 Cabin: Wolverine Woodland
Emmerich-Danziger, Emma Age: 8 Cabin: Otter Overpass
Hunter, Naomi Age: 11 Cabin: Wolverine Woodland
Sasaki, Johnny Age: 9 Cabin: Otter Overpass
Sears, David Age: 13 Cabin: Badger Bunkhouse
Sears, Eli Age: 13 Cabin: Badger Bunkhouse
Silverburgh, Meryl Age: 9 Cabin: Otter Overpass
--------------------------
Unenrollment of Emma Emmerich-Danziger Successful!
--- Camp Statistics ---
Number of Campers: 6
-----------------------
That camper is not enrolled.
This person is either too old or too young to be in Camp Badger.
```

Fun (and Helpful) Information

- String compareTo()
- The Rules of Recursion (Courtesy of *Data Structures and Algorithms in C++* by Mark Allen Weiss)
  1. You must always have some base cases, which can be solved without recursion.
  2. For cases that are to be solved recursively, the recursive call must always be a case that makes progress toward a base case.
  3. Assume that all recursive calls work.
  4. Never duplicate work by solving the same instance of a problem in separate recursive calls.
- You are allowed to add your own methods to any of the classes.
- Thinking about how recursive search works on a binary search tree is a good starting point to approaching deletion.
- Start early! This project is not trivial.