Homework 5: Binary Search Tree

Due: See T-Square

Important

There are general homework guidelines you must always follow. If you fail to follow any of the following guidelines you risk receiving a **0** for the entire assignment.

1. All submitted code must compile under **JDK 7**. This includes unused code, so don't submit extra files that don't compile. (Java is backwards compatabile so if it compiles under JDK 6 it *should* compile under JDK 7.)
2. Do not include any package declarations in your classes.
3. Do not change any existing class headers, constructors, or method signatures.
4. Do not add additional public methods when implementing an interface.
5. Do not use anything that would trivialize the assignment. (e.g. don't import/use `java.util.LinkedList` for a Linked List assignment. Ask if you are unsure.)
6. You must submit your source code, the `.java` files, not the compiled `.class` files.
7. After you submit your files redownload them and run them to make sure they are what you intended to submit. You are responsible if you submit the wrong files.

Binary Search Tree (recursive and non-balancing)

You will be coding a non-balancing recursive binary search tree for this homework. It is important that you carefully read the javadocs for this assignment as some important implementation details will be mentioned there. Please be aware that we are expecting a recursive BST solution. No iterative coding of the BST methods will be accepted! Make sure you utilize recursive helper methods, they will be of great use for this assignment.

In the node class we have provided a toString method that will allow you to print out your BST at any point for debugging purposes. An example of this format will be provided in the console output section. In order to utilize this toString in node you need a proper toString in the User and BST classes, so we will provide them here since we are not providing these files for you. You are to copy and paste this code. Do not alter it.

User toString:

```
public String toString() {
   return userName + "-" + name;
}
```

BST toString:

```
public String toString() {
   if (root == null) {
      return "()";
   }
   return root.toString();
}
```

Note the toString result for an empty BST will be (). The toString result for a BST with one node will have the form (username-name()()). See the example driver run for more examples.

Driver

You have been hired by a social networking startup to implement a new system for how their user data will be organized. The startup is very optimistic about their product success, and have requested that your implementation handle data from a large number of users while offering efficent operations. You have opted to use a binary search tree for this implementation and will store user objects in your binary search tree alphabetically by users usernames. Usernames are NOT case sensitive! ms123 will be equivalent to MS123.

JUnits

In addition to the toString methods, we are providing a substantial number of JUnits so that you can have a good start with testing and debugging your code. The actual testing we do will include concepts these JUnits test and more.

By seeing more JUnits, the hope is also that you will learn more about how to design your own JUnit tests.

As always, be sure your Driver is comprehensive and easily usable. As you drive your code, use the debug command often (perhaps after every option) to be sure your BST is exactly as it should be. If you are perplexed by the format of the toString for the BST, please ask.

User

Your user class should include:

1. Name of the user
2. The user's unique username (not case sensitive although you should store it as it was when added to the tree) - so your comparing must ignore case, but do not simply destroy the case inwhich the user entered the data.
3. One public constructor that takes the name and username as parameters. Class User should not have a parameterless constructor as making a User object with no information violates data integrity.
4. Implementation of the java.lang.Comparable interface so two user objects can be compared

System Interface

Your system should include an interface for handling changes so no one can destroy your backing data structure. This interface should include the following options:

1. Add - adds a user to your system. Note a duplicate username cannot be added. See example below. It must be gracefully ignored.
2. Remove - removes specified user.
3. Find - Given a specific username returns the users name.
4. List users - Displays a list of all users and their usernames sorted alphabetically by username.
5. Debug - Prints the BST in nested parenthesized form governed by the toString methods given for BST, Node, and User. Debug should just call BST's toString method and print the result.
6. Exit - quits the program.

See the example run on the next page.

This is an example of what your console should output, but it does not have to look exactly like this:

```
Please enter a command: debug
String representation of tree:
()
Please enter a command: add
Please enter username: ksimon93
Please enter name: Kevin Simon
Please enter a command: debug
String representation of tree:
(ksimon93-Kevin Simon()())
Please enter a command: add
Please enter username: Onim
Please enter name: Jay
Please enter a command: list
List of current users:
ksimon93-Kevin Simon, Onim-Jay
Please enter a command: add
Please enter username: KSIMON93
Please enter name: Not Kevin
Please enter a command: list
List of current users:
ksimon93-Kevin Simon, Onim-Jay
Please enter a command: find
Please enter a username you would like to find: Onim
Onim-Jay
Please enter a command: remove
Please enter username you would like to remove: Not in list
User does not exist!
Please enter a command: remove
Please enter username you would like to remove: ksimon93
Remove was successful!
Please enter a command: list
List of current users:
Onim-Jay
Please enter a command: add
Please enter a username: az123
Please enter a name: Jon
Please enter a command: list
List of current users:
az123-Jon, Onim-Jay
Please enter a command: debug
String representation of tree:
(Onim-Jay(az123-Jon()())())
Please enter a command: exit
Goodbye!
```

Debug in this case is used to print the toString representation of this tree. The format for this String representation per Node is (data left right). Make sure to look at Node's toString to gain a better understanding of this format. At the BST level, this toString has two possibilities, it returns () if the BST is empty, or the result of Node's toString method called with the head node when the BST is not empty. Do not modify the given toString methods. They are provided for BST, Node, and User.

Style and Formatting

It is important that your code is not only functional but is also written clearly and with good style. Efficiency also always matters. We are providing an updated style checker located in T-Square-Resources along with instructions on how to use it. (It was updated on February 5, 2014.) Be sure to run your code against this latest style checker. While it is backward compatible with the original style checker, some style rules have been made easier to pass, so it is in your interest to update to this Feb 5 style checker file.) If you feel like what you wrote is in accordance with good style but still sets off the style checker please email Hannah Lau hlau7@gatech.edu and Kush Mansingh kmansingh3@gatech.edu with the subject header of "Style XML".

Javadocs

Javadoc any helper methods you create in a style similar to the Javadocs for the methods in the interface.

Provided

The following files have been provided to you:

1. `BSTInterface.java`
2. `Node.java`
3. `BasicBSTTests.java`
4. `CheckStyle.zip` - see T-Square Resources for the latest file from February 5, 2014

Deliverables

You must submit all of the following files. Please make sure the filename matches the filenames below.

1. `BST.java`
2. `User.java`
3. `Driver.java`

Be sure you receive the confirmation email from T-Square, and then download your uploaded files to a new folder, copy over the interfaces, recompile, and run. It is your responsibility to re-test your submission and discover editing oddities, upload issues, etc. You may attach each file individually, or submit them in a zip archive.