COMPX201-25A & COMPX241-25A (HAM): Assignment Two

Binary Search Tree

**Due: Friday 18th April, 11:59pm**

This assignment is intended to give you experience building, maintaining, and using a binary search tree (BST). You must write your own BST class and its required operations.

**Part One:**

Define Java classes to implement a Binary Search Tree and its required operations. Your program should follow the specifications given below:

1. **Appliance:** Define a class called Appliance in a file called Appliance.java. This class is to implement the following:

* A private member variable called “category” to hold the category string.
* A private member variable called “price” to hold the price float.
* A private member variable called “name” to hold the name string.
* A constructor that takes two strings and a float as arguments and sets the appropriate member variables.
* getCategory() – returns the category of the appliance.
* getPrice() – returns the price of the appliance.
* getName() – returns the name of the appliance.
* toString() – override the default Java toString method to give a String representation of the appliance. E.g.

Fridge | Whiteware | $1,049.00

* compareTo(Appliance other) – Compares this object with the specified object for order. Returns a negative integer, zero, or a positive integer as this object is before, equal to, or after than the specified object. Appliances should be primarily sorted by Category (lexicographically). Appliances should be secondarily sorted by price, with cheaper appliances coming first. If there are two appliances with the same category and price, then sort them by name (lexicographically). Check the example output file to see appliances listed in order.

1. **BST:** Define a class called ApplianceBST in a file called ApplianceBST.java. This class is to implement an unbalanced BST using self-referential nodes. Your solution should support the following public methods, using **recursion** where applicable:

* insert(Appliance a) – adds the appliance to the BST, maintaining ordering alphabetically using your compareTo method. Assume duplicates are not allowed.
* remove(Appliance a) – remove the specified value from the BST.
* search(Appliance a) – search the tree to find the specified value and return a Boolean true if the value is found, false otherwise.
* getHeight() – calculates the total height of the BST (including the root).
* getMinimum() – returns the minimum element in the BST (i.e. A should be returned from a BST of elements: A, B, C, D, E).
* getMaximum() – returns the maximum element in the BST (i.e. E should be returned from a BST of elements: A, B, C, D, E).
* print() – displays the tree in the Console. This method should follow in-order traversal, with each value on a separate line.

1. **The Node:** Define a class called Node for the nodes in your ApplianceBST. It can be either an external class in a separate file called Node.java, or an inner class of ApplianceBST. It should have the following:

* A public member variable to hold the value of a node as an Appliance.
* A public member variable to hold a link to the left subtree.
* A public member variable to hold a link to the right subtree.
* A constructor that takes an Appliance value as an argument and copies that value into the node’s member variable. The constructor should also initialise the left and right subtree links to null.

1. **Debugging:** Write a program class that creates one or more of your ApplianceBST objects and tests that all your methods work as per the specification. Example output from one such file will be provided to you. Your test program will not be marked but will be beneficial for your own solution development process.

**Part Two:**

In order to use this composite index effectively, functionality that lets the user search for a range of appliances needs to be added. This includes searching for all appliances in a given category and searching for appliances within a given price range. These functions shouldn’t iterate over the whole tree and filter the results, but instead only search the relevant branches of the tree. Add the following functions to ApplianceBST.java:

* printCategory(String c) – displays all Appliances in category c in the Console. The appliances should be printed from lowest price to highest price.
* printCategoryWithPriceRange(String c, float min, float max) – displays Appliances in category c within given price range i.e. min < price < max. The appliances should be printed from lowest price to highest price.
* printCategoryAbovePrice(String c, float min) – displays Appliances in category c with a lower bound on price i.e. min < price.
* printCategoryBelowPrice(String c, float max) – displays Appliances in category c with an upper bound on price i.e. price < max.

Define a new class called ApplianceLookup. This class should have a main method, and a collection of supporting methods. Your ApplianceLookup class should be able to read the text file provided on Moodle, process it into separate name, category, and price values, and store each item from the file as an Appliance in your ApplianceBST.

Your ApplianceLookup class should then allow the user to select options via the command line to do the following:

1. Search for an appliance in the BST and print if it is included or not.
2. Add a new appliance to the BST.
3. Remove an appliance from the BST.
4. Print all the appliances with a given category.
5. Print all the appliances within a given category with a min and/or max price.

The application user interface should present the available options to the user along with instructions on how to select an option and input the desired information. It should handle errors gracefully (i.e. not crash if an invalid input is entered, but provide helpful feedback).

**Assessment**:

Completing Part One can earn up to a B grade, but to be eligible for an A+ you must also implement Part Two. Your solution will be marked on the basis of how well it satisfies the specification, how well you have approached the problem (Solution Quality), how well-formatted and easy to read your code is (Code Quality), and whether each class and public method has at least some comment explaining what it does, what it’s for, and what any of its arguments are (i.e. documentation). Your code should compile and run as a console program from the command-line (i.e. no GUI or IDE).

**Submission**:

Create an empty directory (i.e. folder) using your student ID number as the directory name. Place copies of your source code (.java files) in this directory. If you wish to communicate with the marker any additional information then you may include a plain text README file, but nothing else (e.g. no compiled code (.class files) or IDE folders or files). Compress and upload this directory through the Moodle submission page for this assignment.