# CSE 214 – Homework III

Instructor: Ritwik Banerjee

This homework document consists of 3 pages. Carefully read the entire document before you start coding. This homework has two parts. The first requires you to implement the `java.lang.Comparable` interface, and then use comparable objects together with the core functionality of a binary search tree. The second part is about creating a hash table with chaining.

---

Like the last homework, a part of the code is already given to you. Your task is to fill in the blanks. The code given to you consists of three packages. The contents are as follows, with interfaces in green, and classes in blue (with abstract classes italicized):

package: **datastructures**
- `BinaryTree`
- `BinarySearchTree`
- `BinaryTreeNode`
- `InOrderTraversal`
- `PreOrderTraversal`
- `PostOrderTraversal`
- *`Traversal`*
- `Set`
- `ChainedHashSet`

package: **products**
- `Laptop`

package: **app**
- `Main`

In the code that is being provided to you, **it has been clearly marked which methods and/or classes you may not modify. Please follow this rule strictly**, since otherwise your homework submission will not be graded. Further, the tasks to be accomplished are marked with "TODO" in the code comments. You *are* allowed to write additional methods and/or instance or static fields, as and when you feel the need for them.

## 1 Total order and binary search trees

Your tasks (total worth 50 points) consist of making the `Laptop` object comparable, completing the binary search tree functionality, and completing the three types of binary tree traversals.

### 1.1 Tasks

1. **Comparing laptops** (4)

   Use the `java.lang.Comparable` interface to compare laptops in terms of their price. For this exercise, make sure that **cheaper laptops are "greater"**.

2. **Binary Search Tree Algorithms**

   Write the code for the `add`, `remove` and `find` methods in the `BinarySearchTree` class, and check for equality between two nodes in binary trees.

   (a) The `add` method must adhere to the insert algorithm studied in class. (5)

   (b) The `remove` method must adhere to the delete algorithm studied in class. (7)

   (c) The `find` method must adhere to the search algorithm studied in class, but with one change: *the return type in the code is the entire path starting from the root node all the way up to the item being searched for*. **Read the documentation of this method as provided in the code to understand the details**. (7)

   (d) The `equals` method in the `BinaryTreeNode` class is not implemented. Currently, it is a dummy method that always returns `false`. You must implement a correct equality checking method. In the lectures, we have studied how to check equality of recursive data structures. Use that technique to write this method. You will realize that if you do this, then, you can check the equality of two binary trees by simply checking whether or not their root nodes are equal. (6)

3. Lastly, you have to implement three types of binary tree traversal algorithms: *inorder*, *preorder*, and *postorder*. These are to be implemented in the three classes given to you, all of which implement the `Traversal` interface. You should make no changes to the interface. (21)

## 2 Chained hashing for a set

Your tasks in this section consist of implementing a set data structure that is based on maintaining the set's elements in a hash table using direct addressing and chaining. This part is also worth a total of 50 points. You will need to study the `Set` and `ChainedHashSet` code and complete the incomplete code given to you. There is a small test code provided to you in `Main`, so that you can see how this should should be tested for correctness. Keep in mind that this is just a sample test, and you are expected to test your code much more extensively.

1. Complete the `size()` method. (3)

2. Complete the `isEmpty()` method. (2)

3. Complete the `contains()` method using the search algorithm discussed in our lectures. (10)

4. Complete the `add(E e)` and `remove(E e)` methods using the insert and delete operations on hash tables (with chaining), as discussed in our lectures. The hash function you employ for this homework must be the division method. Complete the `toString()` method. For this method, the code is commented with Javadoc, and you must adhere to the format explained there. (30)

5. Write the method to check equality of laptops. Keep in mind that this must be the standard Java way of checking equality, so be careful about having the correct method signature and arguments! (5)

## 3 Running the code

The code should be run using the `public static void main(String... args)` in `app.Main` class. You will notice that initially, this class does not compile. But once you make the `Laptop`

class implement the `Comparable` interface, it will. Study this method carefully. The user input format is already described there. Here's an example run for part 1:

```
Enter product instances (format: <brand>,<processor-speed>,<memory>,<price>,<screen-size>):
hitachi,2.33,4,499,13
hp,1.6,2,379,15
apple,2.66,8,1250,14
apple,3.33,8,1600,14
microsoft,2.66,8,999,14
done
'-- Laptop{brand='hitachi', processorSpeed=2.33, ram=4, price=499, screenSize=13.0}
    |-- Laptop{brand='apple', processorSpeed=2.66, ram=8, price=1250, screenSize=14.0}
    |   |-- Laptop{brand='apple', processorSpeed=3.33, ram=8, price=1600, screenSize=14.0}
    |    '-- Laptop{brand='microsoft', processorSpeed=2.66, ram=8, price=999, screenSize=14.0}
    '-- Laptop{brand='hp', processorSpeed=1.6, ram=2, price=379, screenSize=15.0}
Enter traversal type:
preorder
Laptop{brand='hitachi', processorSpeed=2.33, ram=4, price=499, screenSize=13.0}
Laptop{brand='apple', processorSpeed=2.66, ram=8, price=1250, screenSize=14.0}
Laptop{brand='apple', processorSpeed=3.33, ram=8, price=1600, screenSize=14.0}
Laptop{brand='microsoft', processorSpeed=2.66, ram=8, price=999, screenSize=14.0}
Laptop{brand='hp', processorSpeed=1.6, ram=2, price=379, screenSize=15.0}
```

## Guidelines

- **Can I change the package structure?** **No.**
- **Can I change the given code?** **Not if it is marked as unchangeable, or leads to the need to change code that is marked as unchangeable.**
- **The main method doesn't use the search and delete algorithms.** The main method given to you is a part of the test code that will be used to grade your submission. Usually, test codes are not given out at all, but in this case, we are hoping that this helps you test your code! You are free to change the main method while you test your own code. However, you must revert to the original main method before submitting your code. Remember, this code is part of the testing process. The TA cannot be held responsible for failure to test the code if you have made changes here!
- **What should I submit?** A single .zip archive consisting of the entire codebase with the package structure in tact. Extract your archive and double-check its structure before submitting to be sure of this. Again, failure to maintain this will lead to the TA not being able to test your code properly.

**Some additional points to be strictly enforced:**

- **My code compiles on my laptop but ...** Then you have changed some code that you were not supposed to change, or submitted something that is not in line with the guidelines provided. **If your code does not compile, it will not be graded**.
- **I submitted the class files by mistake.** Class files are not code. It cannot be graded.
- **My zip file is empty but I can show you the code on my laptop.** You must verify that you are, indeed, submitting the proper contents. We cannot grade *anything* except for the submission made on Blackboard within the deadline.

# 4 Submission Deadline

**11:59 pm, Nov 25 (Wednesday)**