Assignment 4: Tests, Contracts, Aspects

(may be done by a team of at most two students)

Assigned: Monday, November 9

Due: Tuesday, November 24, 11:59 pm **(Parts 1, 2, 3)**

Part 1: JUnit Test for DupTree

Refer to file `JUnit.zip` containing three files: `BST.java`, `BST_DupTree_Test.java`, and `A4_Part1_Console_Output.txt`. The file `BST.java` contain three classes: `Tree`, `DupTree`, and `TreeIterator`.

Your task in this part of the assignment is to develop JUnit tests for the class `DupTree`. The file `BST_DupTree_Test.java` gives the overall outline of the code to be developed by you.

How to develop `BST_DupTree_Test.java`:

- Create an Eclipse project called `BST` and import the file `BST.java` into this project. Right-click on the project and choose `New` → `Other` → `Java` → `JUnit` → `JUnit Test Case`. Click Next and enter the name `BST_DupTree_Test`. A skeletal class with this name will be created. Copy the contents of the class outline given as part of the assignment, and complete the outline of the following methods as indicated below.

- `setup()`: Build a duptree by inserting *25 random numbers* in the range `0..24`. Also record these numbers in a Java `ArrayList` object. Sort the array list after all numbers are added.

- `check_invariant()`: Use *`assertTrue`* to check the binary search tree property. You need to define the boolean `ordered()` function, as illustrated in Lecture 18 slide #6.

- `test_insert()`: Create two iterators, one for `DupTree` and the other for `ArrayList`, and check using *`assertTrue`* that every number returned by one iterator is also returned by the other. This ensures that `insert` has inserted all the numbers correctly and without any spurious extras.

- `test_delete()`: Insert 10 random values into the duptree, each in the range `0..24`. For each value `v`:

  (a) Obtain the count associated with `v` using `get_count()` – this function is to be written by you in class `BST_DupTree_Test`.

  (b) Delete `v` from the duptree and check that the count has decreased by one if `v`'s original count was more than one; otherwise, check that `v` is no longer in the duptree.

- In the above four methods, generate `Console` output using `System.out.println` in order to log how the tests progressed.

Once developed, run the project as a *JUnit Test* and check that you get an output similar to what is illustrated in the file `A4_Part1_Console_Output.txt`. Use this name for your output file as well.

Note: Since random numbers are inserted into the duptree, you are likely to have different lists of numbers in your Console output.

What to Submit. Prepare a top-level directory named *A4_Part1_UBITId1_UBITId2* if the assignment is done by a team of two students; otherwise, name it as *A4_Part1_UBITId* if the assignment is done solo. (Order the *UBITId*s in alphabetic order, in the former case.)

In this directory, place the files: `BST.java`, `BST_DupTree_Test.java` and your `A4_Part1_Console_Output.txt`. Compress the top-level directory and submit the compressed file using `submit_cse522` (grads) or `submit_cse410` (undergrads). Only one submission per team is required.

Part 2: Contracts for Trees and Iterator

Refer to file `AbsTree.java` which defines binary search trees using `AbsTree`, `Tree`, and `DupTree` classes. Your task in this part of the assignment is to write contracts for the `insert` and `delete` methods of class `AbsTree` and also for the constructor, the `next()` and `stack_tree_nodes()` methods of class `AbsTree_Iterator`.

Note that the post-conditions for `insert(n)` and `delete(n)` need to ensure that the counts are appropriately updated for the object containing the value `n`. Hence, it is helpful to introduce in classes `Tree` and `DupTree` methods `insert` and `delete` which delegate the actual task of insertion and deletion to their respective superclass methods, but add a small amount of code to support the enforcement of post-conditions.

What to Do. The missing Java codes and contracts are indicated via comments, and these are the places where you need to make changes. *Do not modify other parts of the file.* More specifically, in `AbsTree` and `DupTree` you need to:

a. Define the `boolean` methods `member(int n)` and `ordered()` in class `AbsTree.`
b. Define `Contract.Requires()` for `AbsTree.delete(int n)` so that it applies to trees and duptrees.
c. Define `Contract.Ensures()` for `DupTree.insert(int n)` and `DupTree.delete(int n)` suitably.

(You are not required to make any additions to class `Tree`.)

In `AbsTree_Iterator`, you need to:

a. Define `Contract.Requires()` and `Contract.Ensures()` for the constructor. The former should state that the input tree is ordered and the latter should state that the stack-top has the minimum value in the tree.

b. Define `Contract.Requires()` and `Contract.Ensures()` for `next()`. The former should state that there are more values in the tree/duptree, and the latter should state that the next value will be the ascending order.

c. Define Contract.Ensures for stack_tree_nodes() by stating that the next smallest value in the tree/duptree is at the top of the stack.

Run AbsTree.java augmented with your contracts and ensure that the program works correctly. (Run using *Run Configurations*, not *Debug Configurations*.)

***What to Submit*.** Prepare a top-level directory named *A4_Part2_UBITId1_UBITId2* if the assignment is done by a team of two students; otherwise, name it as *A4_Part2_UBITId* if the assignment is done solo. (Order the *UBITId*s in alphabetic order, in the former case.)

In this directory, place the revised AbsTree.java. Compress the directory and submit the compressed file using submit_cse410 (undergrads) or submit_cse522 (grads). Only one submission per team is required.

Part 3: Aspect-Oriented Programming (to be assigned)

End of Assignment 4