Assignment 2

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

Assigned: Thursday, September 20

Part 1: Inheritance to Delegation. Due: Mon, Oct 1, 11:59 pm

See A2_Part1.pdf.

Part 2: Generic External Iterator Design Pattern. Due: Weds, Oct 3, 11:59 pm

(2a) The file `GenericTree.java` posted under `Resources` → `Assignments` defines a generic binary search tree class `Tree<T extends Comparable<T>>`. The file also gives the outline of a generic external iterator for this class. Your task in Part 2a is to complete the iterator definition by writing code for its constructor and the `next()` and `hasNext()` methods, as well as one more method explained below.

Implementing the tree iterator. Refer to `TreeIterator.png` for a pictorial guidance on how to implement a tree iterator. The key idea is to maintain a private *stack of trees* in the iterator class. The stack maintains the invariant that the next value to be returned by `next()` is contained in the tree node pointed to by the top of the stack. When the iterator is initially created, its constructor should stack all nodes along the *left spine* of the tree so that the node containing the smallest value in the tree as at the top of the stack. Each time `next()` is called, in addition to retrieving the next value to be returned and popping the stack, the stack should be updated so that its invariant is maintained. As the sequence diagram `A2_Part_2a.seq` clarifies, it is helpful to define a private void method `stack_left_spine()` for this purpose.

The `main` method is contained in class `GenericTree` and it makes use of the generic iterator to test whether two sets represented as BSTs are equal. Complete the definition of the generic tree iterator and run it and check whether your sequence diagram agrees with `A2_Part_2a.seq`.

There is ***no submission required for Part 2a***, but the iterator you defined in this part will serve as a basis for part 2b described below.

(2b) The file `GenericIterators.java` posted under `Resources` → `Assignments` defines generic versions of `AbsTree`, `Tree`, and `DupTree` discussed in the lectures. The file also gives the outlines of generic external iterators for these classes, called `AbsTreeIterator`, `TreeIterator` and `DupTreeIterator` respectively. Your task in Part 2b is two-fold:

(i) Complete the iterator definitions by writing code for their constructor as well as the `next()`, `hasNext()` and `stack_left_spine()` methods. The behavior of the iterator for `Tree` nodes is similar to that of Part 2a. For `DupTree` nodes, the `next()` method should return the value *v* in a `DupTree` node as many times as indicated by the count *k* associated with this node. (Of course, each invocation of `next()` returns only one value.) *Hint: In order to implement this behavior, you might think in terms of stacking not just the tree/duptree node but also a count.*

The key criterion for Part 2b (i), apart from correct behavior, is that code duplication should be avoided. In fact it is possible to define `next()`, `hasNext()` and `stack_left_spine()` entirely in `AbsTreeIterator` so that its subclasses only have to define their respective constructors.

(ii) Refer to the two test methods, `test1` and `test2`, which are called from the `main` method of the driver class, `GenericIterators`. The method `test1` represents two sets in terms of two `Tree`s and checks whether the first set is contained in the second. The method `test2` represents two bags in terms of two `DupTree`s and similarly checks for containment. See further below for a clarification of containment.

Your task is to complete the definition of the static `boolean` method `containedIn()` in class `GenericIterators` so that it works for sets as well as bags. This method invokes `TreeIterator` (for set containment) or `DupTreeIterator` (for bag containment).

The key criterion for Part 2b (ii) is that, when the containment property does not hold, the function `containedIn()` should return false without always traversing both sets/bags. This is possible because the elements of both sets/bags are `Comparable` and therefore can be enumerated in order. The method `containedIn()` should print out on the `Console` the values that are compared during its execution. The desired output is shown in file `A2_Part_2b_ii_Console_Output.png`.

Run `GenericIterators.java` to completion and check that your console output agrees with the desired output. Also, save the object diagram in a file `A2_obj2.png` using the 'Objects with Tables' option. The sequence diagram is not required.

What to Submit. Prepare a top-level directory named *A2_Part2_UBITId1_UBITId2* if the assignment is done by a team of two students; otherwise, name it as *A2_Part2_UBITId* if the assignment is done solo. (Order the *UBITId*s in alphabetic order, in the former case.) In this directory, place your source file `GenericIterators.java` and object diagram `A2_obj2.png`. Compress the directory and submit the compressed file using the `submit_cse522` command.

End of Assignment 2

Clarification of set and bag containment.

A set s1 is contained in a set s2 if every member of s1 is also a member of s2.

A bag b1 is contained in a bag b2 if every member, x, of b1 is also a member of b2 and, furthermore, the number of occurrences of x in b1 is less than or equal to the number of occurrences of x in b2.