**CS 2110 In-Class Quiz #2**

1. Suppose that we are given a binary tree that contains a dictionary of N words. Each node contains a word and has a left and a right child. For the first 4 problems assume the tree is sorted into alphabetic order (that is, the tree is a *binary search tree*).

|  |  |  |
| --- | --- | --- |
| T | F | By convention, tree nodes should support the “comparable<T>” interface. For tree node X, this interface defines an operation X.compareTo(Y) which returns a number less than 0 if tree node X is smaller than Y, equal to zero if X is equal to Y and greater than 0 if X is greater than Y.  |
| T | F | Suppose that you happen to create a tree in which the root node is Zyzzyva (a snouted beetle sometimes found in association with tropical palms). This is the last word in the English dictionary, hence all other words will be found in the left subtree of the root, and the right subtree of the root will be empty.  |
| T | F | In the special case just described (where the root happens to contain Zyzzyva), the tree will look more like a list, with depth N-1, and all the words laid out along the left child pointers of each tree node. All the right child pointers in the tree will be null. |
| T | F | Since a list is a special case of a tree, List<T> can also be used as an interface to a tree whose nodes are of type T. |
| T | F | For the tree shown to the right, an inorder traversal would print B A C |  |
| T | F | For the tree shown to the right, a preorder traversal would print B A D |  |
| T | F | If you have a list of sorted data and want to turn it into a *balanced* BST a way to do that is to write a recursive method that inserts the middle item in the list (if the list is of even length, just pick the item ``on the left’’ of the midpoint), then recursively calls itself on the left half of the list, and then on the right half of the list (excluding the item you already inserted, obviously). The base case would be a list of length 1, and for this your method would just insert the single item on the list, then return. |
| T | F | A second way to create a BST from a list of sorted data is to pick a random item, insert it into the tree and remove it from the list, stopping when there are no more items to insert. |
| T | F | The two methods just described would create the identical BST.  |
| T | F | Java does not provide any way for a dynamic method of a class to call a static method, or access a static field, or for a static method to call a dynamic method, or access a dynamic field. |

2. Write a delete method for the double-linked list class shown below. The method should find the node with the given key (if any) and delete it, fixing pointers and the head pointer as required.

public class List {

public ListNode head; // If non-null, points to first element

 public class ListNode {

Object key; // Key (“name”) for this node

Object value; // Data associated with this list element

List next, last; // Pointers to next and last element

public ListNode(Object k, Object v) { this.key = k; this.value = v; }

}

public void add(Object k, Object va) {

ListNode ln = new ListNode(k, v);

If(head == null) {

 // List was empty

head = ln.next = ln.last = ln;

return;

}

// Make the new ListNode the last on the list

ln.next = head;

head.last.next = ln;

ln.last = head.last;

head.last = ln;

}

public Object find(Object k) {

If(head == null) return null;

ListNode ln = head;

do {

if(ln.key.equals(k)) return ln.value;

ln = ln.next;

}

while(ln != head);

return null;

}

public void delete(Object k) {

// First find the node

If(head == null) return;

ListNode ln = head;

do {

if(ln.key.equals(k)) {

// Unlink it from the list

ln.last.next = ln.next;

ln.next.last = ln.last;

// Empty list? Set head to null.

if(head.next == head)

head = null;

return;

}

ln = ln.next;

}

while(ln != head);

}

}