

CS 61B Lab 10
April 3-4, 2014

Goal: to give you experience with general trees and encapsulation. The trees we use in this lab use the `SibTree` data structure described in Lecture 24 (which you should review now), and are encapsulated in the same manner as the Homework 5 lists (whose design was the subject of Lecture 19). `SibTrees` are designed to ensure that the `SibTree` and `SibTreeNode` invariants (which are written out in their respective files) cannot be violated.

Please make sure you have a partner for this lab.

Copy the Lab 10 directory by doing the following, starting from your home directory.

```
cp -r ~/cs61b/lab/lab10 .
```

All the code is in the `tree` package. You can compile it from your `lab10` directory with `javac -g tree/*.java`. Extensive test code is provided and can be run with `java tree.SibTree`.

Familiarize yourself with the fields and methods of the `SibTree` and `SibTreeNode` classes. `SibTree` has two fields, one inherited from the `Tree` abstract class.

```
int size;           // The number of SibTreeNodes in the SibTree.
SibTreeNode root;  // The node that serves as the root of the tree.
```

`SibTreeNode` has six fields, two inherited from the `TreeNode` abstract class.

```
Object item;       // Item stored at this SibTreeNode.
boolean valid;     // True if and only if this is a valid node.
SibTree myTree;    // The SibTree that contains this node.
SibTreeNode parent; // This node's parent.
SibTreeNode firstChild; // This node's first (leftmost) child.
SibTreeNode nextSibling; // This node's next sibling to the right.
```

As with the Homework 5 lists, the `Tree` class defines certain nodes to be invalid. In contrast to the Homework 5 lists, valid and invalid nodes are distinguished solely through the state of the `valid` field. When a `TreeNode` is removed from a tree, it becomes invalid. Methods like `parent()`, `child()`, and `nextSibling()` return an invalid node (never null!) if no such node exists. You may create an invalid node by calling the zero-parameter `SibTreeNode()` constructor. You may test whether a node `n` is valid by calling `n.isValidNode()`.

Every valid `SibTreeNode` is in some tree, specified by the `myTree` field.

Your task is to implement the `parent()`, `insertChild()`, and `removeLeaf()` methods of the `SibTreeNode` class. After you write each one, you may use the test code to check your progress.

Part I: Accessing a Node's Parent (1 point)

Fill in the body of the `parent()` method in `SibTreeNode.java`. `parent()` returns the `SibTreeNode` that is the parent of "this" `SibTreeNode`. If "this" node is the root, return an invalid node.

Throw an `InvalidNodeException` if "this" node is not valid.

Part II: Inserting New Children (3 points)

Fill in the body of `insertChild()`. `insertChild()` takes two parameters: an item and an integer `c`. Create a new child that is the `c`th child (from the left) of "this" node, and references the item indicated. Existing children numbered `c` or higher are shifted one place to the right to accommodate. If `c < 1`, act as if `c` is 1. If "this" node has fewer than `c` children, the new node is the last sibling.

Don't forget that `SibTrees` have a `size` field that needs to be updated.

Throw an `InvalidNodeException` if "this" node is not valid.

BONUS Part III: Removing a Leaf (1 bonus point)

Fill in the body of `removeLeaf()`, which removes "this" node from the tree if it is a leaf, and does nothing if it is not a leaf. Upon completion, "this" node should be invalid.

As always, throw an `InvalidNodeException` if "this" node is not valid.

Check-off

Show your TA or Lab Assistant the code you have written, and run the test program. You'll receive points for each part that runs without printing any error messages. You can receive up to 5 points out of 4.

(Note to TAs: be stringent about the bonus point, and don't give extra time.)