Lab Worksheet 3: Internals of Binary Trees

The goal of this worksheet is:

- To gain better understanding of how to manipulate the structure of a binary tree.
- To understand Java API documentation.
- To write code to implement the key methods of a Proper Binary Tree.
- To gain skills that will be necessary for the programming assignment that begins soon.

In previous labs we were working as programmers who *use* data structures. Now we are working as programmers that *create* data structure implementations.

Download the `W3-Source.zip` file from Brightspace import it into IntelliJ.

This project contains a "`doc`" folder that contains the Javadoc you will need for this lab. To explore this, open the "`index.html`" file. Pay special attention to the `AbstractBinaryTree` class in the `dsa.impl` package.

There are two classes in the project:

- `ProperLinkedBinaryTree` will become a full implementation of a proper binary tree, once you have finished this lab. Your code will go in this file.
- `ProperTreeTest` contains some code to help you to test your implementation.

Part 1: Create an initial tree

A Proper Binary Tree initially contains 1 node: a root node that is external (i.e. its element is `null` and it has no children). To make this happen, you must update the `ProperLinkedBinaryTree` constructor so that it contains an external root node and has a size of 1.

Hint: The `AbstractBinaryTree` class has a useful method named `newPosition(…)`, which creates a `BTPosition` object and returns it (`BTPosition` is an implementation of `IPosition`). Remember that `ProperLinkedBinaryTree` extends `AbstractBinaryTree`.

Part 2: Expand an external node

Next, you should implement the `expandExternal(…)` method.

1. If a user tries to expand an internal node, an exception should be thrown.

   Hint: One way to throw an exception is to use:
   ```
   throw new RuntimeException( "Write a message here…!" );
   ```

2. If a user tries to expand an external node, you need to store an element in it, create two external children, and increase the size of the tree by 2.

Hint: Because expandExternal(…) accepts an IPosition type as a parameter, you will need to **cast** this to a BTPosition before you can access its `parent`, `right`, `left` and `element` attributes.

```
public void expandExternal(IPosition<T> p, T e) {
      BTPosition node = (BTPosition) p;
      // now you can access these attributes of "position"
}
```

Part 3: Removing a node

The final part is to remove a node from a Proper Binary Tree. Remember, a node can only be deleted if it has 1 or more external children. You will need to do the following:

1. If the user tries to remove a node with two internal children, throw an exception in the same way as before.

2. If the left child is external, remove the node and the left child.

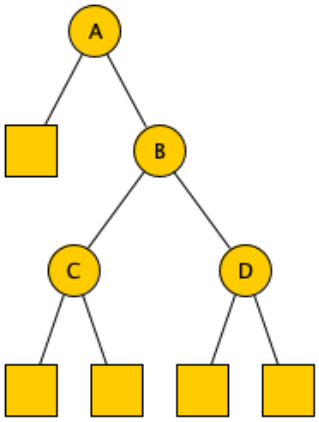
   Hint: You will need to adjust one of the children of the node's parent, and the parent of the right child. You don't need to actually *remove* an object – you need to connect the nodes that *are* still part of the tree together. Java's garbage collection will eventually delete any unused nodes.

3. If the left child was internal, but the right child is external, remove the node and the right child.

4. Remember if the node is the root of the tree, this is a special case because it has no parent. The root of the tree must be changed by this operation.

5. Run the main method in `ProperTreeTest` to see if you get the correct output. The expected output is below.

   Hint: This output shows the structure of the tree. The root is first, then its children are listed: left first, then right. For example, "`TEST 1`" below represents this tree:



======== TEST 1 ==========

A

B

C

D

B

C

D

```
========== TEST 2 ==========
A
       □
       B
              C
                     □
                     □
              D
                     □
                     □

Expected Result! No updated tree printed.
========== TEST 3 ==========
A
       □
       B
              C
                     □
                     □
              D
                     □
                     □

A
       □
       B
              □
              D
                     □
                     □
```