Objective

- Understand how to use a tree data structure.

Setup

1. Download the `Lab22StarterCode.zip` file from `msBrekke.com`
2. Unzip starter code in `H:/CSIII/Lab22` folder

Overview

You will write code that stores hierarchical data in a tree data structure and uses that data to generate information about the relationships between the stored data.

Assignment

You will store the names of the British Royal Family in a tree data structure and write methods that will be able to *crawl* the data.

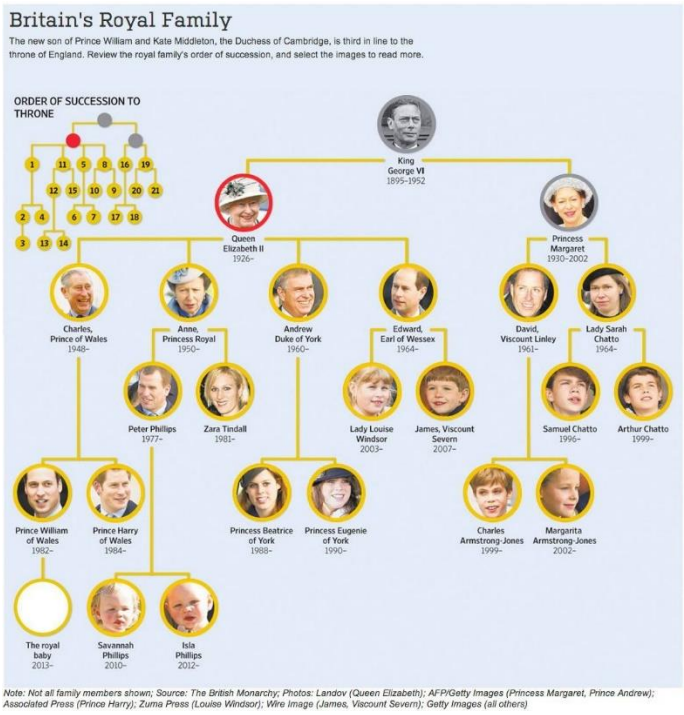
You will write the following methods:

TNode<String> find(TNode<String> node, String name)
Return a `TNode` that contains the specified data

void getPath(TNode<String> node)
Return a String that shows the path from the root to the specified node like this:

```
root -> child1 -> child2 -> node
```

The data of each child is separated by an arrow (minus, greater than), starting with the root node and ending with the child node.



Find Method

The find method takes two parameters: a `TNode<String>` and a `String`. It will return a `Node<String>` that has data that matches the `name` parameter, or `null` if no such node is found.

If `name` matches the parameter `node` data then that node is returned. Otherwise, all the children nodes are checked.

This method should be written recursively.

Populating the Tree

You will be creating a tree out of `TNode<String>` objects. The data stored in a `TNode` will be the name of someone from the British Royal Family.

The first thing you need to do is read the data from `data.txt` into a tree data structure. Use `Scanner` and `File` from `java.util` to read each line from the `data.txt` file (look back to your old file loading labs!).

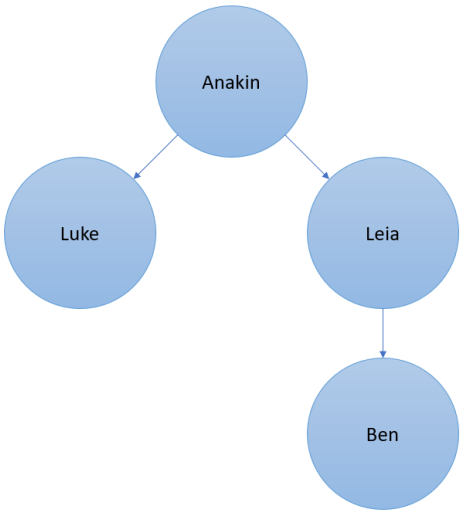
The first line from the `data.txt` file contains the data for the root node of the tree. Each other line contains data in the format:

```
A > B
```

Where `A` is the parent node of `B`. You should use the `split` method to separate the `A` and `B` Strings and use the `find` method you just wrote to *find* the `TNode` that contains `A`, then add a TNode containing `B` as a child.

For example, the following `data.txt` would generate this tree.

```
Anakin
Anakin > Luke
Anakin > Leia
Leia > Ben
```



It is guaranteed that any `A` will be added to the tree before it is used as a `B`.

Getting a Path

The `getPath` method takes a `TNode` as a parameter and returns a `String` that displays the data of every node between the root and the parameter. The returned string concatenates `" -> "` between each `TNode`'s data.

For example, a call to `getPath( BEN )` would return the `String`:

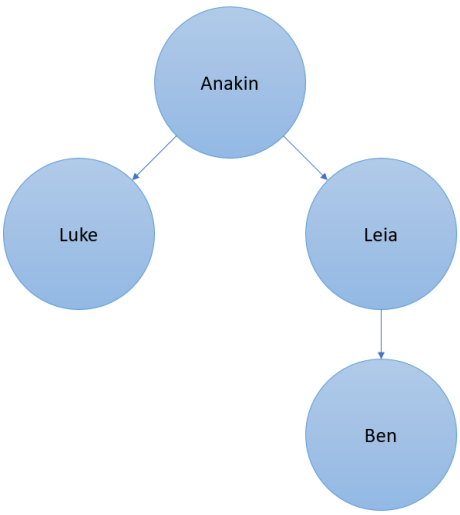
```
"Anakin -> Leia -> Ben"
```

Calling `getPath( LEIA )` would return the `String`:

```
"Anakin -> Leia"
```

And calling `getPath( ANAKIN )` would return:

```
"Anakin"
```



Testing

Use the JUnit tests to make sure your methods are working correctly.

Grading

Your program should meet all the following specifications:

- Your code is in your `H:/CSIII/Lab22` folder
- Your code compiles and runs without error
- Your `find` method should work correctly, and recursively.
- Your `getPath` method should work correctly, and recursively.