Assignment 4: Trees

CS102: Data Structures, Fall 2013
Eric Koskinen and Daniel Schwartz-Narbonne
New York University

Due: 11:55PM (23:55:00), Thursday November 7th

Context

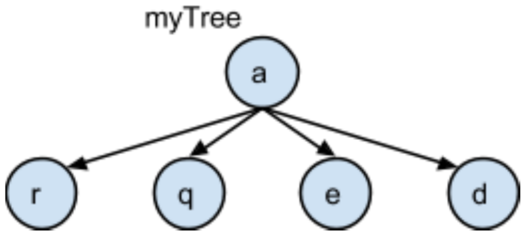
As the mobile app has been gaining popularity, users have been demanding features to allow them to combine movie information with the movies on their disk drive. Consequently, the app will now need to be able to build a representation of their disk drive. It immediately occurs to us that a Tree is the perfect data structure for this task.

Scope

As in the previous assignment, we have done most of the coding for you. Your job is to implement a couple of ***data-structure*** methods, as well as some ***application-level*** methods. In particular, you will implement a tree `findChild()` method, as well as an application method which constructs a tree to represent files organized in directories and subdirectories.

Task 1: Implement the `findChild` Tree Method

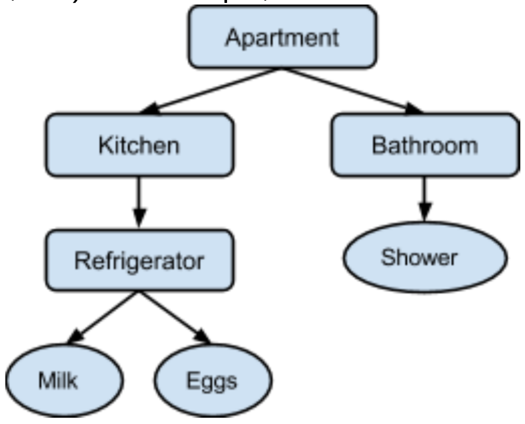
Your first task is to implement a member method of the `Tree<T>` class. In particular you will implement `findChild(T lab)` which searches through the *immediate* children of the tree, and if it finds a child subtree S such that S's label is `equal()` to `lab`, it returns S. It returns `null` if no such subtree is found. For example, let's say we have a tree called `myTree<String>` whose label is "a" and has child subtrees as follows:



If a user executed the following code:
```
        String labE = new String(“e”);
        return myTree.findChild(labE);
```
then the subtree consisting of the "e" node would be returned.

Task 2: Implement the `displayXML` Tree Method

In class we have examined pre-order and post-order traversals for displaying a tree. There are also cases, such as in this assignment, where you may want to perform a traversal that has *both* pre- and post-order behavior. Let us return to the apartment example, where we have represented the inventory in a tree, organized by location (room, closet, etc.). For example,



This tree can be displayed in XML format. XML documents are typically represented as follows:

```
        <?xml version=”1.0” encoding=”ISO-885901”?>
        <location name=”Apartment”>
                <location name=”Kitchen”>
                        <location name=”Refrigerator”>
                                <item name=”Milk”></item>
                                <item name=”Eggs”></item>
                        </location>
                </location>
                <location name=”Bathroom”>
                        <item name=”Shower”></item>
                </location>
        </location>
```

Notice that there is a `<location…>` node for every location and an `<item…>` node for individual items. Items are nested within Locations and Locations may themselves be nested in other Locations. There is one root Location (in this case labeled “Apartment”).

Now contemplate how you would implement a `displayXML()` method for a tree that would generate this output. Notice that, for a given node, some things need to be displayed ***before*** you explore the node’s children, and other things need to be displayed ***after*** you explore the node’s children.

To simplify matters, we have already implemented two methods for the labels:

`label.preString();`	*Generates a string to be displayed before child traversal.*
`label.postString();`	*Generates a string to be displayed after child traversal.*

You will need to use these methods inside the `displayXML()` method of `Tree<T>`.

Task 3: Implement the `insertPath()` Application Method

The core challenge of the application is to construct a Tree representation of the file system. Much of the work has been already done for you. You simply need to implement the following method:

```
   public void insertPath(Tree<FileNode> t, QueueList<String> filePathQueue)
```

This method constructs a subtree (or uses an existing subtree) to store the new file, represented by `filePathQueue` into `Tree<FileNode> t`. The input filePathQueue is a Queue consisting of the path to the file. For example, if we are inserting the file “`myDir/mySubDir/myFile`”, this function would be called with the following `filePathQueue`:

```
IN -> [ “myFile” , “mySubDir” , “myDir” ] -> OUT
```

This is convenient because a simple `filePathQueue.dequeue()` operation gives you the next subdirectory or, in case `filePathQueue` is of size 1, the name of the file.

You will need to use recursion to implement this method. A few important things to keep in mind:

1. `insertPath()` will only be invoked with real files that are in your current working directory on your computer (or in subdirectories thereof).
2. The final element of the `filePathQueue` will always be a file (not a directory).

Task 4: Testing

As always, testing is essential to this assignment. There are several things to test in this assignment:

1. The new Tree method `findChild()`.
    a. Tests should create Trees and exercise this method appropriately.
2. The new Tree method `displayXML()`.
    b. Tests should create Trees and generate XML. For this assignment you will not need to automatically test that the XML is correct. However, you need to **manually** ensure that it is correct. That is, you need to create Trees and ensure that the XML generated is correct. When we grade your implementation, we will run specific tests and ensure that the XML output is correct.
3. The new FileSystem method `insertPath()`.
    c. You will need to write user test that create FileSystem objects as follows:
    `FileSystem fs = new FileSystem(“path/…”)`
    d. You will need to provide a test directory structure consisting of directories, subdirectories, and simple text files, and ensure that the correct Tree is built.
    e. Your tests can examine the tree that has been constructed through the FileSystem method fs `getTree()`.
    f. Our automatic tests will run your code on ***our*** directory structure, and your grade will depend on whether the XML output exactly matches what is expected.

As with all assignments, you will use the suite of testing tools:

- `TestHarness.java` -- A TestHarness class which provides the infrastructure for the testing suite. You do not need to modify this class.
- `Test1.java` -- An example of the kind of tests you should be creating.
- `Test.java` -- The main test function which registers individual tests and then invokes all of them. You will need to **extend the code** in `main` in this class and register any tests that you create.

As with the previous assignment, we will be developing some incorrect implementations, and running your tests cases on them. One of the jobs of your test suite is to try to catch all of our broken implementations.

Documentation

In this assignment and henceforth, we will be using Javadoc:
    http://en.wikipedia.org/wiki/Javadoc
For your convenience we have already documented the majority of the code in Javadoc form. You must be consistent with this form.

Compilation and Execution

Your assignment will be automatically compiled as follows:

```
    javac *.java
```

Note that the Java compiler will automatically compile other classes that are inherited. Finally, your code will be executed as:

```
    java Test
```

Note that your implementation will be built and executed against your test cases, as well as our test cases. It is to your benefit to devise as many test cases as possible!

Submission

Please submit all .java and .txt files, as well as your report (whose filename should include your netid and name), to NYUClasses.

In particular, you should have modified the following files (and no others)

- `Tree.java`
- `FileSystem.java`
- `Test.java`
- `TestX.java` -- Your test files
- `mysample.zip` -- A single zip file containing a directory called “`mysample`” which consists of various subdirectories and files, used in testing FileSystem. ***Note:*** if you do not follow this naming convention precisely, you will lose points.

Grading

Your assignment will be evaluated based on the following:

- Sufficiency of the tests you design (40%)
- Your implementation (50%)
- Clarity and organization of the code, and sufficiency of comments. (10%)

We look forward to your submission!
-- CS102 Course Instructors

For **bonus points (mandatory for honours students)**, you may

- Give a one-line Ocaml type definition for a generic Tree.
- Implement an XML output printer for your tree when your tree's labels are strings.