CSE 341, Fall 2004, Assignment 6 (version 1)
Due: Tuesday 30 November, 9:00AM

Overview: You will define 4 Smalltalk classes (all subclasses of `Object`) and add methods to them. Put your classes in a category `lastname_hw6` where `lastname` is your last name. Do not change any code in other categories.

1. Define a class `Leaf` with an instance variable `str`, which for now we assume will hold a `String`. Define a setter method `string:`, but do *not* provide a getter method.

2. Define a class `BinaryNode` with instance variables `left` and `right`, which for now we assume will hold either a `Leaf` or a `BinaryNode`. Define a setter method `left:right:`, but *not* getter methods.

3. Add `concatAll` methods to `Leaf` and `BinaryNode`. The return value should be a `String` which is all the strings in the tree concatenated together left-to-right. Do not worry about efficiency. Smalltalk already has a method for concatenating two strings.

4. Add a *class method* to `BinaryNode` called `firstAlphabetical:and:`. This method takes two strings and returns the one that comes first alphabetically. The case of letters does not matter; so `BinaryNode firstAlphabetical: 'Hi' and: 'hI'` can return either argument and be correct. Do *not* use existing `String` methods for comparing strings. More specifically send only the messages `size` and `at:` to the arguments. You can send other messages to characters in the arguments. (Use a "while loop".) Sample solution 10 lines.

5. Add `firstAlphabetical` methods to `Leaf` and `BinaryNode`. The return value should be the `String` that is the first string alphabetically in the tree. Use your solution to the previous problem.

6. Define a class `NaryNode` with an instance variable `arr`, which for now we assume will hold an array of strings. Define a setter method `array:`, but do *not* provide a getter method. Define methods `concatAll` and `firstAlphabetical` as in previous problems. For a size-zero array, `concatAll` should return `''` and `firstAlphabetical` should send the error message `'firstAlphabetical on empty NaryNode'`.

7. Add `iterate:onEmpty:` methods to `Leaf` and `BinaryNode`. (Note `NaryNode` is a little trickier, so it is EC1.) The first argument is a block taking one argument and the second argument is a block taking zero arguments. The return value is an iterator implemented as follows:

   - It is an array with two elements.
   - The first element is the result of sending `value: str` to the first argument of `iterate:onEmpty:` where `str` is a `String` in the tree.
   - The second element is a block taking no arguments. If there are no more strings in the tree, this block is the second argument to `iterate:onEmpty:`. Else it is a block that evaluates to an iterator for the remaining strings in the tree.

   Also, the iterator should work "left to right" so the first element in the iterator is the first block applied to the leftmost string in the tree, and so on. Hint: We did something similar in class in Scheme, but in class the interior nodes of the tree also had elements on them.

8. Add a *class method* to `BinaryNode` called `concatAll:after:`. The result of `BinaryNode concatAll: t after: s` where `t` is a tree and `s` is a string is a string that is `s` followed by all the strings in `t` concatenated together. However, the only message you may send to `t` is `iterate:onEmpty:`. Hint: Make the second argument to `iterate:onEmpty:` be `[{}]`. That way you can test if the iterator has more strings by seeing if the size of the array is 0 or 2. Sample solution: 6 lines.

9. Define a class `FunnyNumber` with an instance field `num` (for holding a number), a getter method (`num`), and a setter method (`num:`). Define one other method (quite possibly an infix method) that takes another `FunnyNumber` and returns a new `FunnyNumber` holding the *sum* of self's `num` and the argument's `num`. This other method should have a name such that you can build a tree t holding `FunnyNumber` objects (instead of strings) and `(BinaryNode concatAll: t after: 0)` returns the sum of the tree's elements.

Examples: The file `hw6.text` on the website has a couple examples that should help you understand the methods you need to write.

Extra Credit:

EC 1 Add an appropriate `iterate:onEmpty:` method to `NaryNode`.
Hint: Use a helper method `iterate:from:onEmpty:`.

EC 2 Add a *class method* `concatAll2:after:` to `BinaryNode` that is just like `concatAll:after:` except executing `BinaryNode concatAll2: t after: s` creates only *one* `String` object and no other collections, arrays, trees, lists, etc.
Hint: Iterate through `t` twice.

Turn-in Instructions

- "File out" the category you created and name the resulting file `lastname_hw6.st`.
- Email your solution to `brianhk@cs.washington.edu`.
- The subject of your email should be *exactly* `[cse341-hw6]`.
- Your `.st` file should be an *attachment*.