CSE 341, Spring 2004, Assignment 6
Due: Friday 21 May, 9:00AM

Last updated: 14 May

In this assignment, you will define several Smalltalk classes and methods for "ropes" which we have used as an example in earlier parts of the course.

Preliminaries: We have provided some code for you, in `hw6.st`. Here is what you should do:

1. Save a fresh image.
2. Open a new project in the image.
3. Create a category called hw6.
4. "FileIn" `hw6.st` by:
   (a) Choosing `Open...` and then `File List` from the "World" menu.
   (b) Clicking on `hw6.st` to get a menu (right-click on Windows) and choosing `fileIn`.

You should now have 4 classes in the `hw6` category:

- `Rope` is a subclass of `Object`. The other 3 classes are subclasses of `Rope`.
- `MtRope` has no instance variables.
- `EltRope` has two instance variables, `x` (for holding an element of the rope) and `r` (for holding a smaller rope). `x:r:` sets these fields.
- `TwoRope` has two instance variables, `r1` (for holding a smaller rope) and `r2` (for holding a smaller rope). `r1:r2:` sets these fields.

Problems:

1. Add `sum` methods to `MtRope`, `EltRope`, and `TwoRope`. (So you write one method for each class, for a total of three methods.) Assuming all the elements in a rope are numbers, `sum` should compute the sum of all the elements. If a rope has no elements, the sum is 0. (Note: Your methods should actually work whenever the elements accept an infix `+`, as we will investigate in problem 3.) (Hint: I see no reason for method bodies to be longer than 1 line.)
2. Add `numAs` methods to `MtRope`, `EltRope`, and `TwoRope`. Your methods should compute how many of the elements in the rope are the character `a`. (You write the character as `$a` and test if an object is equal to a character with the infix `=` message.) (Note: The `Character` class is in the `Collections-Text` category if you want to see what messages it accepts, but you do not need to for this problem.)
3. Define a class called `MyCharacter` that is a subclass of `Object` and has one instance variable (intuitively for holding a character, though its up to clients to use the class correctly).
   - Define a method `c:` that sets an instance's variable to the value the method is passed.
   - Define an infix method `+` that takes a number and increments that number if the instance variable holds `$a` and returns the number unchanged otherwise. (Note: To define an infix method `+` the syntax is just `+ aNumber` *`method-body`* where `aNumber` is the name for the argument.)

   Note: Now if you have a rope with elements that are instances of `MyCharacter`, then sending the `sum` message to the rope should return the number of elements holding the `$a` character!
4. Define a class called `OneElt` that is a subclass of `Rope` with one instance variable, an element. (So there is no additional rope, only one element.) Add a method `x:` to set the element, and methods `sum` and `numAs`, such that your previous solutions still work for ropes that contain instances of `OneElt`.

5. Add an `isSumPos` method to the `Rope` class. This method should return true if the sum of the elements (presumably all numbers) in the rope is positive, and false otherwise. Your solution should be one (short) line.

6. Add a *class method* to the `Rope` class called `fromCollection:`. When given an instance of a `Collection` (this class is in the `Collections-Abstract` category), the method should produce a new rope containing exactly the elements in the collection. (Hint: Every `Collection` instance accepts the `do:` message, which takes a code block expecting one argument and evaluates the code block once for each of the collection's elements, passing the element to the block.) Note: Arrays are collections, so you should be able to send `fromCollection:` the argument `{2.  4.  7}` and get a rope with 3 elements.

7. **Small Extra Credit:** In problem 3, it would have been more convenient and better OO-style to have `MyCharacter` subclass `Character` rather than use an instance variable holding a `Character` instance. Write a short, precise English paragraph explaining:

   - Why the implementation of the `Character` class makes this approach fail.
   - How a Java class would prohibit subclassing.
   - How the approach in the `Character` class does more than prohibit (useful) subclassing, but does not actually prohibit (unuseful) subclassing.

8. **Extra Credit:** For this problem, a "visitor" is an object with two methods:

   - The `doIt:` method takes a rope. For different "visitors" this method will do different things, but part of what it will do is call the rope's `visit:` method (see below), passing `self`.
   - The `onElt` method returns a block accepting one argument. The argument should be an element of a rope.

   Do the following:

   - Add a `visit:` method to each of the 4 subclasses of `Rope`. Each method expects to be passed a "visitor". The purpose of `visit:` is to evaluate the code block returned by the visitor's `onElt` method for each element of the rope. Of course, this will require "visiting" smaller ropes in some cases.
   - Write a `SumVisitor` class that is a "visitor". Calling the `doIt:` method on a rope `r` should produce the same result as sending the `sum` message to `r`. However, the methods in `SumVisitor` must not send any message to `r` except `visit:`.
   - Write a `NumEltsVisitor` class that is a "visitor". Calling the `doIt:` method on a rope `r` should produce the number of elements in `r`. Again, you must not send any message to `r` except `visit:`.

   Note: The "visitor pattern", also known as "double-dispatching" is an OO idiom for adding new operations to a subclass hierarchy without adding methods to the subclasses.

Turn-in Instructions

- "FileOut" your classes to `.st` file. Using your operating system (not Squeak), name the file `lastname_hw6.st`, where `lastname` is replace with your last name.
- Email your solution to `martine@cs.washington.edu`.
- The subject of your email should be *exactly* `[cse341-hw6]`.
- Your `.st` file should be an *attachment*.
- For the "small" extra credit, include your paragraph in plain text in the body of your email (not as an attachment).