Assignment 4 --- Lists & Recursion

This assignment has two parts; the first part covers lists where the goal is to construct and visualize a new Singly Linked List implementation using Bridges SLelements instead of Linked Node objects. The second part covers recursion in which you will be asked to design four recursive functions.

Part I

Overview

The primary goals of this program are as follows:

(1) **Understand** how the methods in the SLList class work.

(2) **Create** a Tester class to demonstrate the Singly Linked List data structure. Assign visual properties to the elements of the List.

The **Tester** should create an instance of the Singly Linked List class. The list will store a sequence of integers. The list should be visualized a number of times as elements are added to it, and each set of new elements should feature different visual properties. Visualizing the data structure and adding visual properties to list elements can be accomplished easily with the Bridges API. See the Tasks section for specific details and instructions.

Tasks

Follow the instructions on this website to add the Bridges library to BlueJ:
http://bridgesuncc.github.io/bridges_setup_java_bluej.html

Create a project named Training in BlueJ and re-create this example:
http://bridgesuncc.github.io/Hello_World_Tutorials/SLL.html

Documentation for Bridges classes can be found at the following link:
http://bridgesuncc.github.io/doc/java-api/current/html/index.html

Singly Linked List Assignment–

--- Create a subclass ListOperastions for the SLList class with the following functions:

- findRightPlaceAndInsert(element): Boolean - this method finds the right place for the element based on chronological order then add the object to the list at that place. The method returns true when the object is added to the list successfully.

- findAndHighlight (element): true - this method returns true if the element is in the list. The method changes the *size* and *color* of the node holding that element.

- updateLabel method that sets the *label* equal to the new element's value

- create methods to change the current element's color, size, shape, and opacity. These are all supported with Bridges methods accessible through SLelement's *getVisualizer* method. Since the driver won't have direct access to the list's *current* SLelement, you need to call these respective Bridges methods from the ListOperations class using arguments passed from the driver.

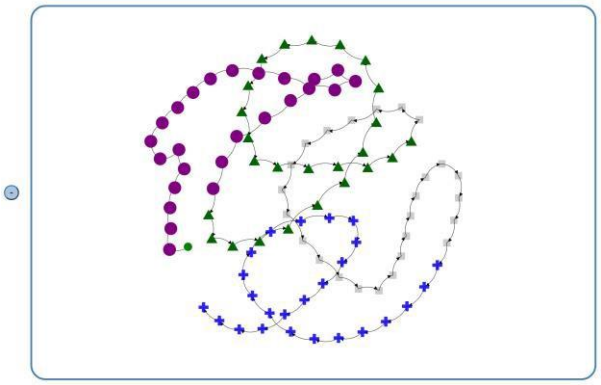
---- note: remember to import bridges.base.SLelement library to the ListOperations subclass so you can use the SLelement's getVisualizer method(s).

Tester –

--- Create a Tester class with a main method
--- Import *bridges.connect.Bridges* to allow you to create a Bridges object
--- Initialize a Bridges object with the assignment number, your username, and your API key. (See the Bridges template on Moodle for details)
--- Create an instance of your SLList class and parameterize it to hold Integers
--- Add 100 new elements to your SLList. The elements should simply contain the index of whichever loop construct you use, and should be inserted in increasing order (from 0 to 99)
--- Every successive set of 25 elements should have different visual properties from the other 75. You are free to use whichever attributes you like:
  - color can be set to any css or RGB color
  - opacity can be set between 0 and 1 (note: to change opacity, color has to be set first).
  - size can be set between 1 and 50
  - shape can be set from among the following options: [square, diamond, triangle-•-down, cross, triangle-•-up, circle]
--- After every successive set of 25 elements are added to the list, call Bridges' setDataStructure method passing in the head of your list and then call Bridges' visualize method
--- Allow the user to add a new element to the list
--- Allow the user to search and element in the list, highlight that element wen found.

Deliverables –

The final Bridges assignment your program should generate will be something like this:



Scoring Rubric

Tester: 90 points

--- Up to 10 points for appropriate documentation and comments
--- Up to 10 points for correctly adding 100 elements to the list in increasing order
--- Up to 10 points for setting the labels for all the added elements
--- Up to 20 points for setting new distinct visual properties for each new set of 25 elements
--- Up to 20 points for visualizing the data structure after each set of 25 elements are added (you will generate a total of 4 visualizations with 25, 50, 75, and 100 elements, respectively)
--- Up to 10 points for allowing the user to add an element to the list
--- Up to 10 points for allowing the user to look for an element in the list and highlighting that element if found.

Singly Linked List: 60 points

--- Up to 10 points for appropriate documentation and comments

--- Up to 10 points for filling the survey:

https://unccpsych.az1.qualtrics.com/jfe/form/SV_dcK4tmEmIQ68hDv

--- Up to 40 points for correctly designing the subclass and adding methods to set color, opacity, shape, and size for the current element

Total points available: 150

Part II

Overview

The primary goal of this part of the assignment is to design recursive functions.

Tasks

--- Create a new project
--- Design a recursive method for each of the method declarations below
--- Create a main method and test your designed methods

public static int multiply (int a, int b)
This method takes to numbers a and b and recursively multiply them together. Assume that a and b are positive integers. The only arithmetic operation allowed in this problem is addition of two integers.

public static int findMin (int array[])
This method takes an array of integers and return the smallest element of the array.

public static int reverse (int number)
This method returns the positive integer obtained when the digits of the parameter is reversed. For example, when called with the parameter 12345, this method would return 54321. (Do this with proper integer arithmetic instead of just simply converting to String, reversing that and then using parseInt to extract the result).
Hint: The number of digits of a number can be obtained by taking the integer part of the logarithm in base 10 of the number + 1 (i.e, num digits = log10(n) + 1).

public static countPaths (int n, int m)
You are standing at the point (n,m) of the grid of positive integers and want to go to origin (0,0) by taking steps either to left or down: that is, from each point (n,m) you are allowed to move either to the point (n-1,m) or the point (n,m-1). Implement the method that based on recursion counts the number of different paths from the point (n,m) to the origin.

Scoring Rubric

Tester:

--- Up to 20 points for appropriate documentation, and **tracing** (can be hand written, an uploaded photo for the tracing is acceptable).
--- Up to 20 points for correctly designing a method recursively.

Total points available: 100