Project: Multi-node trees for pyramid scheme

Submit Assignment

Due Tuesday by 11:59pm **Points** 100 **Submitting** a text entry box or a file upload

Objectives

At the completion of this project, students will demonstrate the ability to work with multi-node trees (also called multiway trees), where each node can have references to more than two child nodes, and as an implementation, each node stores a list of its children nodes in an ArrayList. This is a more complex data structure. Students will also demonstrate the ability to code a complex recursive method for traversing up a tree.

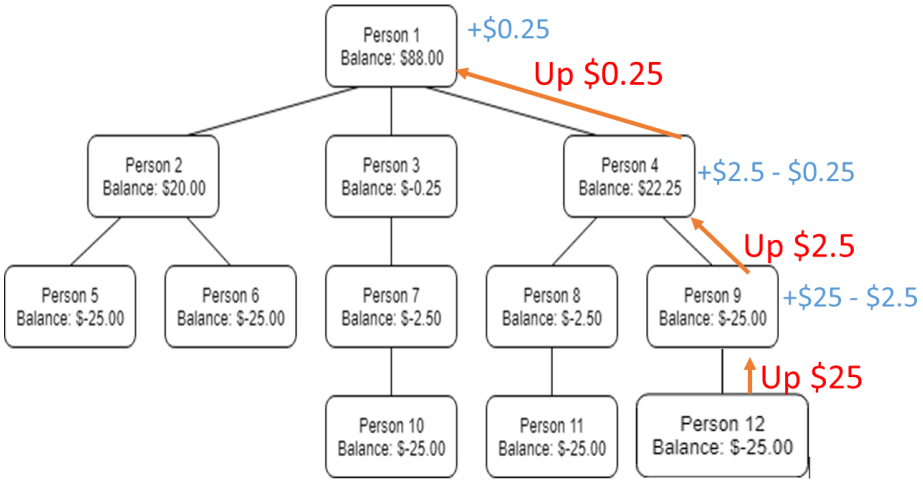
Summary

In this project, you'll be implementing methods for a LinkedTree, a data structure for generic elements held in a tree format (note that this is NOT a BINARY tree – it is a MULTINODE tree, so each node can have any number of children nodes). You'll also be implementing a method in PyramidScheme.java, which is an extension of a LinkedTree specifically for Person elements, and includes logic that is representative of a real pyramid scheme. In a pyramid scheme, each person tries to recruit people. Those people pay to join, and then try to recruit their own people. The money from those who join trickles up the tree to the person who started the pyramid scheme. The earlier you get in on it, the more money you can make. But this is illegal, and the higher up you are in the scheme the more legal trouble you will get in. So, in the PyramidScheme, each new recruit causes people higher up in the tree to earn more money. And then there is a point in the simulation where the whole pyramid scheme collapses. The project can be used to help prove the inproperness of the pyramid economy model.

Your job in the PyramidScheme code is to implement a method that returns the chain of people who should get a portion of the money when a new recruit is added.

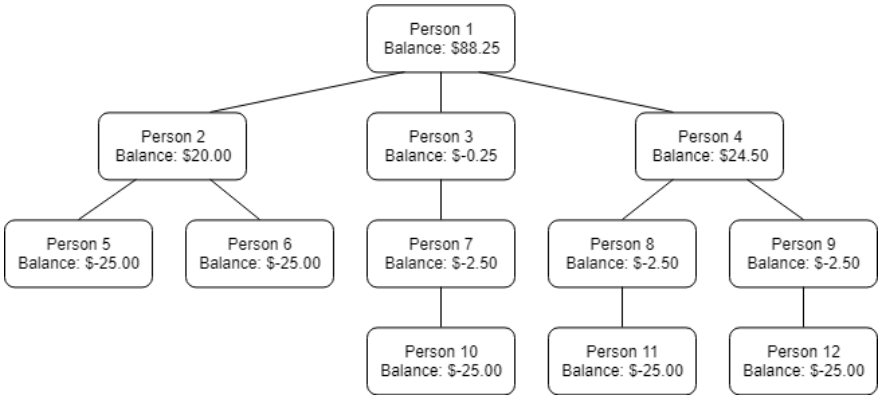
The following is a diagram of the PyramidScheme made using the default parameters ($25.00 recruit price, 10% paid up):

Let us see how the recruitment fee $25 of Person 12 flow back to his recruiter Person 9, up to the Person 1 in the recruiting line.



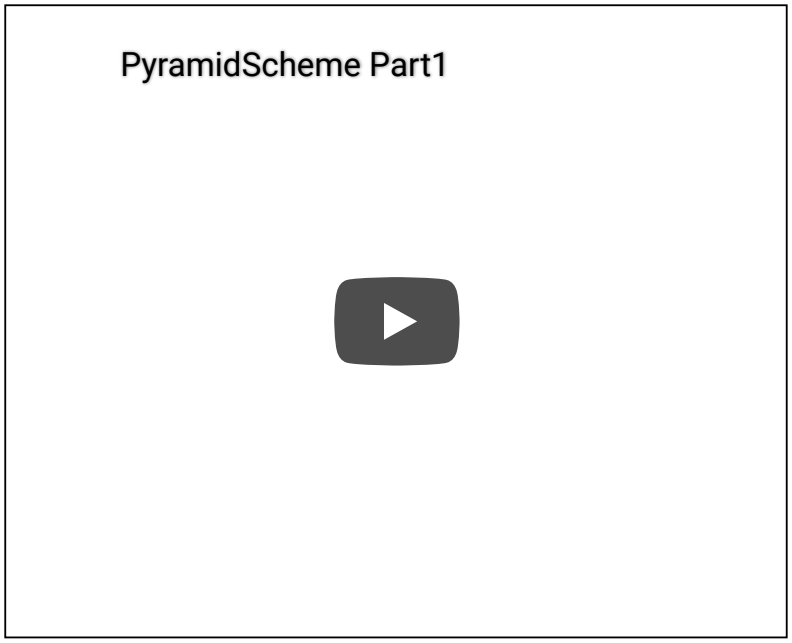
Person 12 lost $25, the recruitment fee, and he gives the $25 actually to Person 9; Person 9 keeps 90% of the $25 and gives his recruiter Person 4 the 10% of $25; Person 4 receives $2.5, keeps its 90%, contributes the remaining 10% to his recruiter Person 1; thus, Person 1 receives 0.25. The updated balance of a person is computed as his previous balance plus the amount of money that he receives and minus the amount of money that he gives out.

Now, this is what the tree looks like after Person 9 recruits Person 12:



Once you have all the code implemented you will be able to run the simulation and play around with the prices and see what happens with the pyramid scheme. In the simulation, we always end by collapsing the pyramid scheme and then the people at the top end up in a lot of trouble.

Note: If in need, please watch the video:
https://youtu.be/1C92q4GFCn0 (https://youtu.be/1C92q4GFCn0)

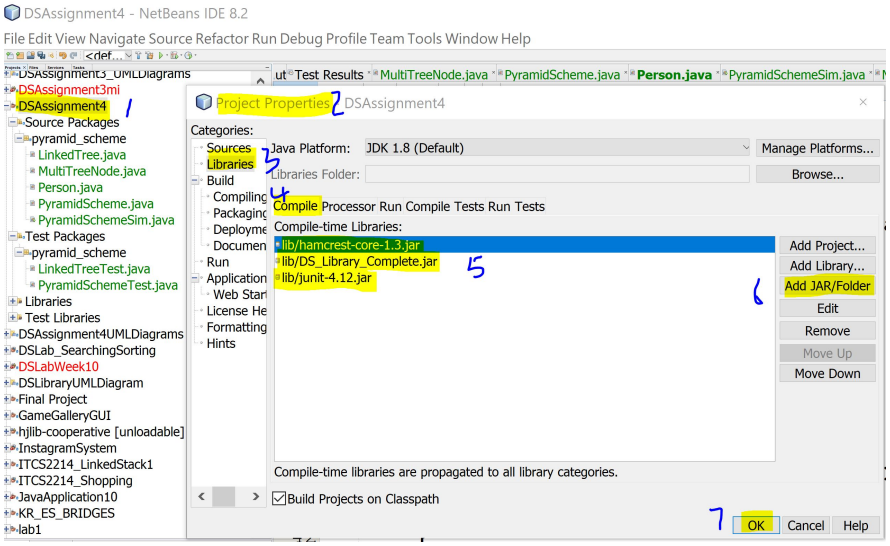


Minimize Video

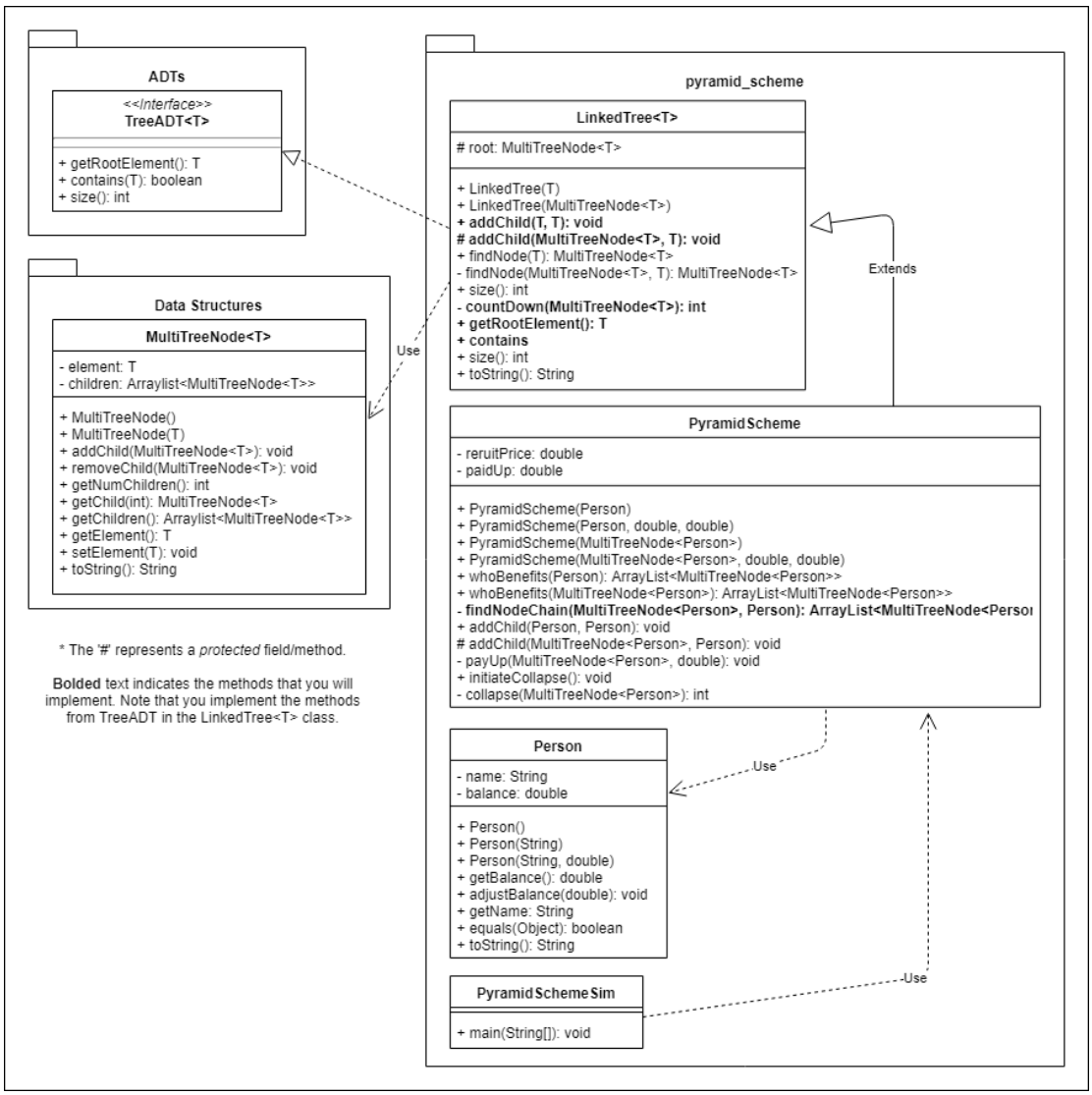
Tasks:

Download & Setup:

1. Download DS_Assignment4.ZIP from Canvas. Import the downloaded .zip file into NetBeans to make a new project, or unzip the downloaded .zip file and create a new project with the unzipped folder.
2. In the project navigation pane of the NetBeans, right click project name, DSAssignment4 and click the "Properties" button. In the pop-up dialog box, check whether these three libraries have been added in the project or not. If they are not in the project, please click "Add JAR/Folder" button to add these three .jar files in the project's lib folder. After libraries are set, click the "OK" button.



3. Inspect the code, and look at the UML diagram below to understand how everything works together.



Part 1: Finish the LinkedTree implementation

Part 1.A

The LinkedTree class uses MultiTreeNode objects to maintain its structure. Every MultiTreeNode object carries a specific data element, as usual. However, instead of having references to its left and right child node, these MultiTreeNode objects keep an ArrayList of references to all its child MultiTreeNodes, which can hold as many children as you add to it.

In this way, you can create a hierarchy of nodes. At the top of this hierarchy is the root node, which is kept as the only field in the LinkedTree class.

The LinkedTree has methods for

- adding children to specific nodes already in the tree
- finding those specific nodes
- other utility methods such as *contains* and *size*.

You must implement the following methods in LinkedTree.java:

- **getRootElement()**: This method returns the element held by the root.

- **addChild()**: This method has two overloading methods (corresponding to same function name but with different parameter lists); you will implement both.

The first takes a generic T parent and T child, invokes the existing method findNode() to locate the parent MultiTreeNode which holds the given element parent, and then, invokes another addChild method to add the child element into the located parent node.

This second overloading method takes a MultiTreeNode parent and the T child element. It's job is to create a MultiTreeNode which holds the child element, then call the node's own addChild() method to append this new child node into its children list.

If the given parent element cannot be found, an ElementNotFoundException should be thrown.

Here we do not care about whether the child element is duplicated within other sibling child nodes of the same parent node. Nothing will be done if the given child element is null.

- **Contains()**: This method would invoke the findNode() method to determine if the tree contains a T target element.
- **countDown()**: This method is a private helper method for the size method; it will use recursion to count the number of nodes in the tree.

Part 1.B

Before you write your code, please plan for your coding by

- making a new (google) report document

- describe these classes in the package briefly in the report

- describe the above-mentioned methods briefly in the report

- writing down the pseudo code (or instruction steps) of the above-mentioned methods in the report

You don't need to make the report perfect. Using your English is fine. The most important is to use the report to organize your thoughts and make your logical thinking clear.

Hint: The methods you implement in LinkedTree are all really short – none should be more than 5 lines of code, some of them can be a single line of code. If you are writing complicated methods, you are not doing this right.

Part 2: Run the tests to ensure that your LinkedTree methods are working. Do this before moving on to Part 3.

Part 3: Finish PyramidScheme Implementation

The PyramidScheme class extends LinkedTree<Person>, meaning that it has access to all the functionality of a normal LinkedTree, except it only holds Person elements. This is because the PyramidScheme has its own methods, and overrides some of its superclass' methods, that contain logic specifically meant for the Person class. The Person class is a basic representation of a person as an object, with fields for a person's name and their balance (money).

The method you need to implement in PyramidScheme.java:

findNodeChain: This method will use recursion to find a target node, and then return an ArrayList of every node above the targaet node, up to the node that was initially passed in (in most cases, the root). Essentially, you can think of this as returning an ArrayList of the target's parent, then grandparent, then great grandparent, et cetera. The whoBenefits method calls this method (with root as the parameter so that the entire tree is searched) in order to get a list of every node containing a person who benefits from the money made by the target. This will be a complex recursive method; however, its logic is similar to that of findNode in the LinkedTree class, so you should examine that to help determine the logic for this. This is the hardest part of the project.

Hint: Finding the node chain is backwards recursive: you need to make your way down through the tree, find the node, and then on the way back up, add each 'parent' node to the list. So the computation here (assigning the node to the ArrayList) happens as the recursive calls are returning and popping off the runtime stack.

You could apply recursion or use an external linear or non-linear data structure(s) that we have learned this semester to facilitate the back-tracking process without recursion.

If in need, please feel free to add facilitative private method(s).

Note: If in need, please watch the video:

https://youtu.be/zu2e3LqWUkg (https://youtu.be/zu2e3LqWUkg)

(https://youtu.be/zu2e3LqWUkg)

Requirements

You must implement the following:

In LinkedTree.java:

- getRootElement
- addChild (T, T)
- addChild(MTN<T>, T)
- contains(T)
- countdown(..) (this is called by size())

In PyramidScheme.java:

- findNodeChain(Person) (use LinkedTree's findNode as an example for this process; also look at whoBenefits to understand how these results are used)

Submission:

Part 1.A (code): Test your code. Submit your project to WebCAT through NetBeans "Submit Project" button or zip your src folder and submit the zipped src folder to the WebCAT website.

Part 1.B (report): submit your report document to Canvas.

Part 2 (honor code): submit your honor code to Canvas.

Honor Code:

Sign and submit the following declaration on Canvas.

Without this declaration, you will not receive a grade for the project.

I promise that I completed this project on my own and did not look at, borrow from or copy project code from a classmate or anyone who took the course before. All the code in this project (except what was given by the instructors) was written by me.

Describe any help you received on this project (from a professor or TA or the tutoring center):

Signature: **Date:**

Internal Assessment (not for grading)

Criteria	Ratings					Pts
Prob Solv-2Planning threshold: 3.0 pts	**5.0 pts** **Exemplary- Student applied logic to the problem to generate the best strategies and clear, concise plans for a comprehensive solution.**	**4.0 pts** **Accomplished- Student applied logic to the problem to generate reasonable strategies and clear, concise plans for a comprehensive solution.**	**3.0 pts** **Acceptable- Student applied logic, developed strategies, and created a plan, but the solution may address the defined plan, but does not address all underlying issues.**	**2.0 pts** **Needs Improvement- Student developed a plan that is partially successful in addressing all aspects of the defined problem.**	**1.0 pts** **Beginner- Student attempted to develop a plan based on logic, but the plan does not solve all aspects of the defined problem.**	--
Prob Solv-3Implementation threshold: 3.0 pts	**5.0 pts** **Exemplary- Student executes their plan flawlessly to exceed project requirements and their personal plan.**	**4.0 pts** **Accomplished- Student executes their plan nearly flawlessly and meets project requirements in a complete manner.**	**3.0 pts** **Acceptable- Student executes the essential aspects of their plan to meet the project requirements.**	**2.0 pts** **Needs Improvement- Student's attempt to execute their plan is admirable but needs improvements to meet the project requirements.**	**1.0 pts** **Beginner- Student does not execute their plan or executes a plan that does not satisfy the project requirements.**	--
Prob Solv-4Evaluation threshold: 3.0 pts	**5.0 pts** **Exemplary- Student tests solution using typical and non-typical scenarios to ensure final solution is as robust as possible.**	**4.0 pts** **Accomplished- Student tests solution using typical and some non-typical scenarios so the final solution is more robust than required.**	**3.0 pts** **Acceptable- Student tests solution using typical scenarios to ensure project requirements are met.**	**2.0 pts** **Needs Improvement- Student attempts to test their solution, but the test is not robust or the results are not taken into consideration for making improvements.**	**1.0 pts** **Beginner- Student does not test solution or negative test results are dismissed/ignored/hidden.**	--
						Total Points: