For this assignment my approach was to build a tree of Nodes. Each Node has a field of id, minWealth, percent, squared, and a list of its children, which were all necessary pieces of information to successfully write this program as will be illustrated later.

The idea behind my intendToTransferWealth method was to build a tree of Nodes as this method is called. For example, intendToTransferWealth was called from 1 to 2, I set 2 to be a child of 1 by using a BFS algorithm to find the *from* Node and set the *to* Node as its child. See the pseudo-code below for a demonstration.

```
Queue q;
q.add(root)

While q is not empty:
    While the size of q > 0:
        Node p = q.peek
        Dequeue a Node
        If the id of p == from:
            Add a new Node with id of to as a child of p
```

The idea is mostly the same for setRequiredWealth. I still use a BFS to find the Node whos id is equal to the id param. The only difference is when I find the right Node I just set that Node’s minWealth equal to the wealth param.

By the time solveIt is called the tree of Nodes should be set up. I use a Post Order algorithm to end up at the bottom left of the tree. I check to see if the Node I am at has any children if it does not I keep progressing. Once I get to a Node with children, first I check if his wealth is squared. If it is then I just set his minWealth to the square root of his current minWealth. Then I find the largest amount of money between the children `largestChildValue = child.getMinWealth() / (child.getPercent() * 0.01);` and just keep bubbling up until the root now has his minWealth set to the minimum amount of money needed to inject into the system to satisfy all the requirements.