I didn’t use a Linked List so I’ll just describe what it is

**Linked List:** list that has no bit shift so each piece of data pretty much point at each other instead.This makes deleting and adding much faster than arraylist as there is no bit shift/resize. If I wanted to get a piece of data, it would be faster to get it at the front or back of the list because it reads data one by one. Linked lists are used for queues and are generally faster when they are kept small or well organized. A single piece of data(string/int/double) in a linked list has two data references rather than the data and index itself. This causes Linked List to use more memory.

**Array lists:** list that is stored dynamically. Adding and deleting can/will cause a bit shift. Good for storing data. Faster to get something towards the middle of the list than Linked list because you can call to the index directly. Every piece of data is stored as the data and an index.

ArrayList assignment

import java.util.\*;

public class addList {

/\*

Programmer: Peter G Rutherford

Assignment Chapter: Chapter 9

Purpose: My purpose is to learn Java and get a good grade.

Date modified: 03/6/2021

IDE/Toool used: NetBeans IDE 8.2

\*/

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

List<String> list = new ArrayList<String>();

boolean a = true;

while(a==true){

System.out.println("Would you like to add to the list?(yes or no): ");

String b = scanner.nextLine();

if(b.equalsIgnoreCase("yes")){

System.out.println("Enter a Name to add to the list.");

String c = scanner.nextLine();

list.add(c);

}else if(b.equalsIgnoreCase("no")){

System.out.println("\n\n\n LIST \n\n"

+ "--------------------\n" + list.toString());

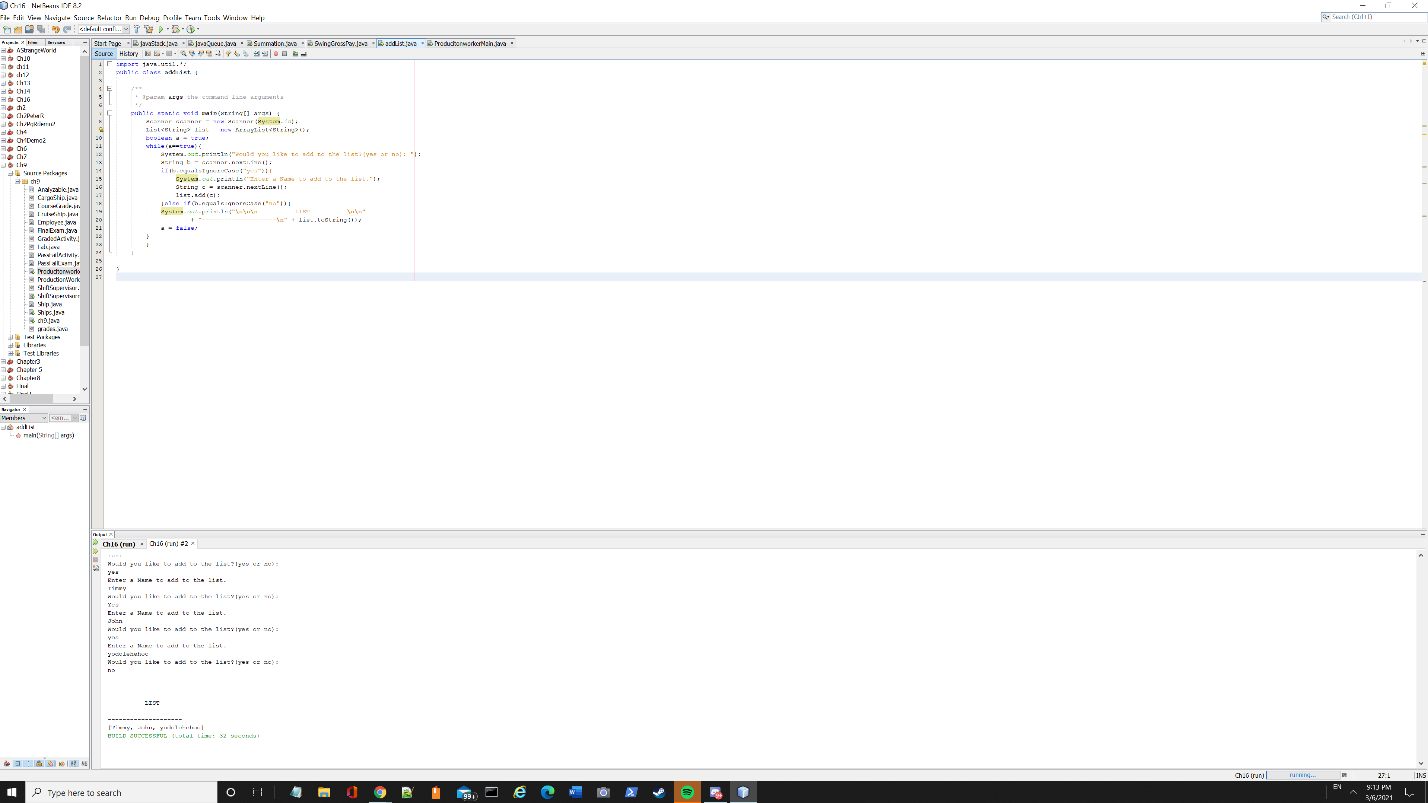
a = false;

}

}

}

}



­­­­­­

BinaryTree.java

I found this binary tree model on <http://www.newthinktank.com/2013/03/binary-tree-in-java/>

I was able to understand the concept of a binary tree much better with the video!

**Binary tree:** A way of storing data, like a linked list. It can point to two different pieces of data in a sort of hierarchy. You start with a root node and a corresponding key, if you add a parent key that is lower than the root, it goes to the left of root. If you add one higher than the root, it goes right of root. If you add one higher than the root but lower than the right parent, it will go to the left of the right parent (called a child). This applies to the other side.

public class BinaryTree {

Node root;

public void addNode(int key, String name) {

Node newNode = new Node(key, name);

if (root == null) {

root = newNode;

} else {

Node focusNode = root;

Node parent;

while (true) {

parent = focusNode;

if (key < focusNode.key) {

focusNode = focusNode.left;

if (focusNode == null) {

parent.left = newNode;

return;

}

} else {

focusNode = focusNode.right;

if (focusNode == null) {

parent.right = newNode;

return;

} } } } }

public void inOrderTraverseTree(Node focusNode) {

if (focusNode != null) {

inOrderTraverseTree(focusNode.left);

System.out.println(focusNode);

inOrderTraverseTree(focusNode.right);

}}

public void preorderTraverseTree(Node focusNode) {

if (focusNode != null) {

System.out.println(focusNode);

preorderTraverseTree(focusNode.left);

preorderTraverseTree(focusNode.right);

}

}

public void postOrderTraverseTree(Node focusNode) {

if (focusNode != null) {

postOrderTraverseTree(focusNode.left);

postOrderTraverseTree(focusNode.right);

System.out.println(focusNode);

}}

public Node findNode(int key) {

Node focusNode = root;

while (focusNode.key != key) {

if (key < focusNode.key) {

focusNode = focusNode.left;

} else {

focusNode = focusNode.right;

}

if (focusNode == null)

return null;

}return focusNode;

}

public static void main(String[] args) {

BinaryTree tree = new BinaryTree();

tree.addNode(50, "Boss");

tree.addNode(25, "Vice President");

tree.addNode(15, "Office Manager");

tree.addNode(30, "Secretary");

tree.addNode(75, "Sales Manager");

tree.addNode(85, "Salesman 1");

tree.inOrderTraverseTree(tree.root);

// theTree.preorderTraverseTree(theTree.root);

// theTree.postOrderTraverseTree(theTree.root);

//System.out.println(theTree.findNode(75));

}}

class Node {

int key;

String name;

Node left;

Node right;

Node(int key, String name) {

this.key = key;

this.name = name;

}

public String toString() {

return name + " has the key " + key;

}

}

