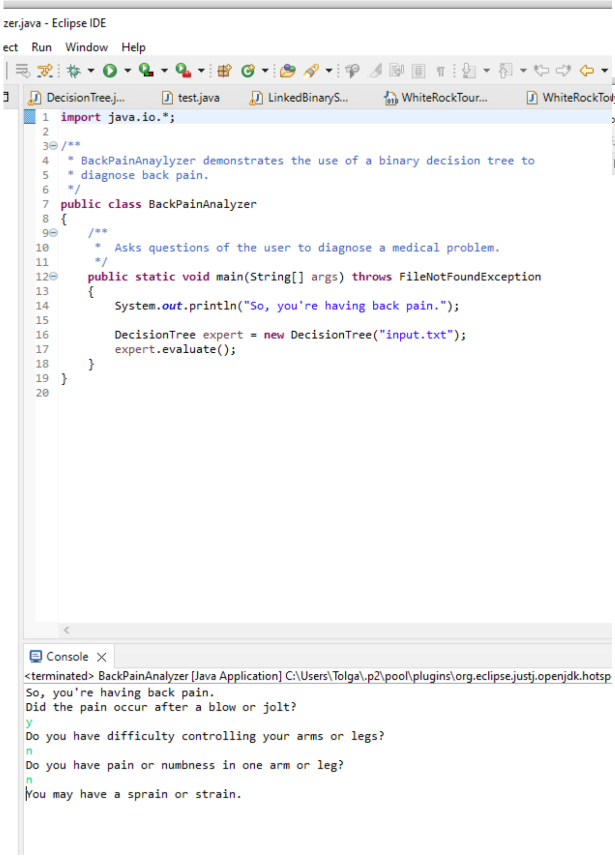
COMP 2231 ASSIGNMNET 4

Tolga Olcay

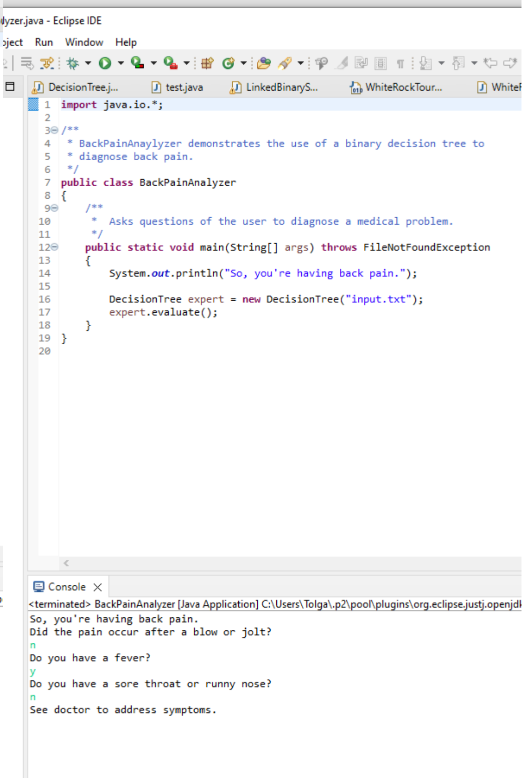
T00666715

1. Backpain Analyzer



```
import java.io.*;

/**
 * BackPainAnaylyzer demonstrates the use of a binary decision tree to
 * diagnose back pain.
 */
public class BackPainAnalyzer
{
    /**
     *  Asks questions of the user to diagnose a medical problem.
     */
    public static void main(String[] args) throws FileNotFoundException
    {
        System.out.println("So, you're having back pain.");

        DecisionTree expert = new DecisionTree("input.txt");
        expert.evaluate();
    }
}
```

```
<terminated> BackPainAnalyzer [Java Application] C:\Users\Tolga\.p2\pool\plugins\org.eclipse.justj.openjdk.hotsp
So, you're having back pain.
Did the pain occur after a blow or jolt?
y
Do you have difficulty controlling your arms or legs?
n
Do you have pain or numbness in one arm or leg?
n
You may have a sprain or strain.
```

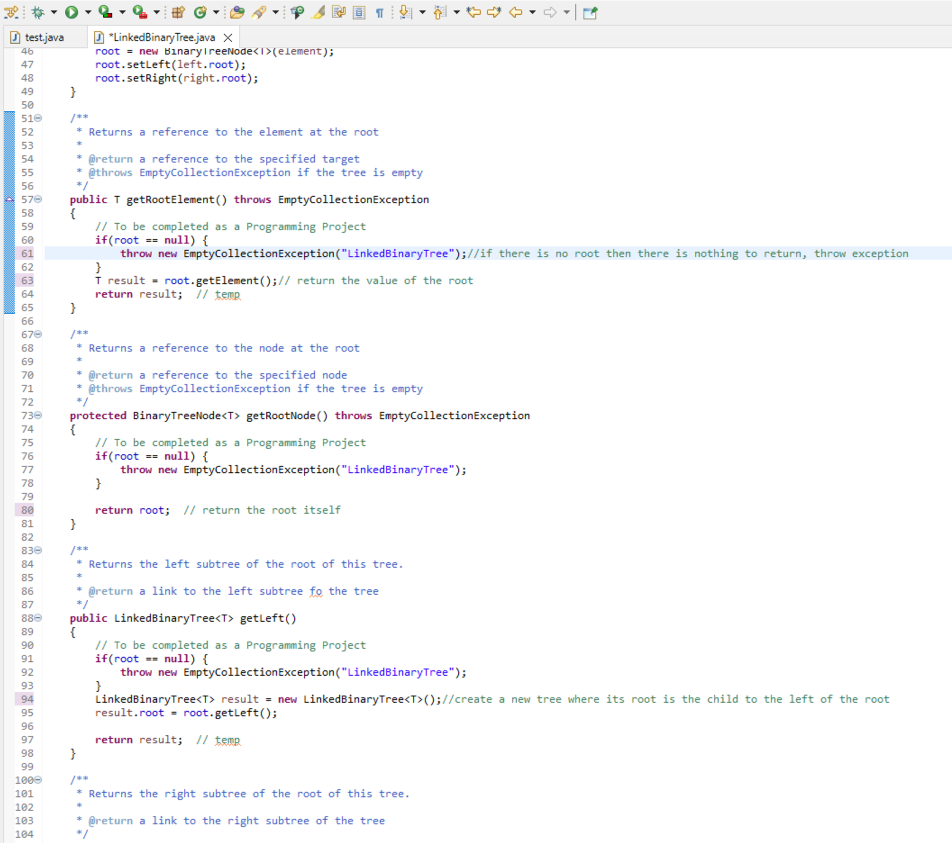


```
import java.io.*;

/**
 * BackPainAnaylyzer demonstrates the use of a binary decision tree to
 * diagnose back pain.
 */
public class BackPainAnalyzer
{
    /**
     *  Asks questions of the user to diagnose a medical problem.
     */
    public static void main(String[] args) throws FileNotFoundException
    {
        System.out.println("So, you're having back pain.");

        DecisionTree expert = new DecisionTree("input.txt");
        expert.evaluate();
    }
}
```

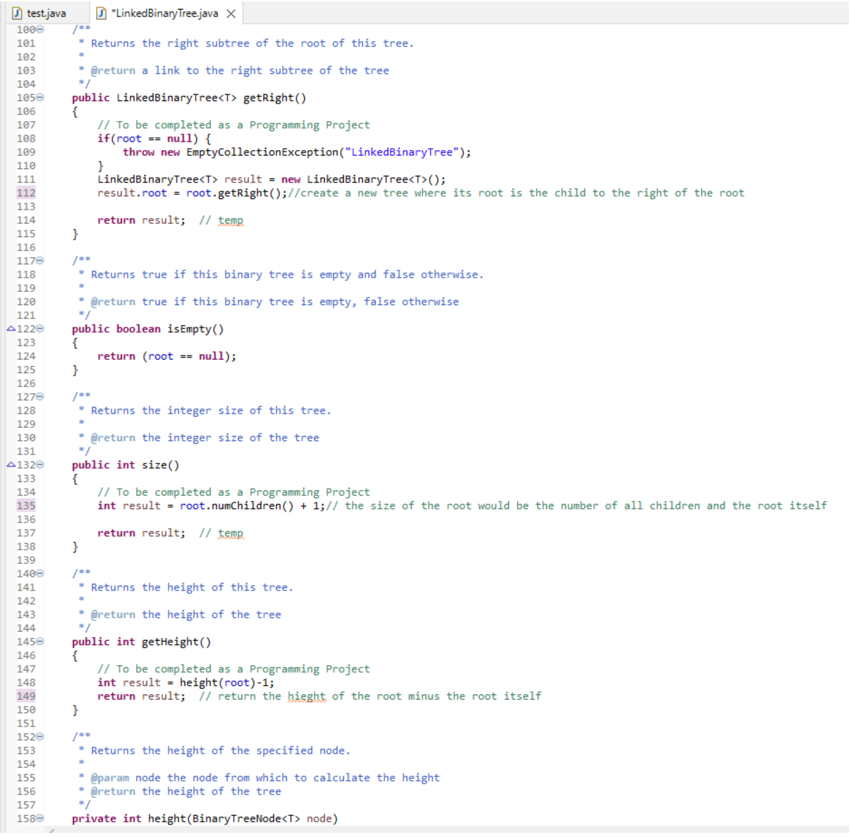
```
<terminated> BackPainAnalyzer [Java Application] C:\Users\Tolga\.p2\pool\plugins\org.eclipse.justj.openjdk
So, you're having back pain.
Did the pain occur after a blow or jolt?
n
Do you have a fever?
y
Do you have a sore throat or runny nose?
n
See doctor to address symptoms.
```

Here are two different examples of the backpain analyzer. As you can see, the analyzer navigates through the tree to provide appropriate responses to the user depending if they have answered yes or no.

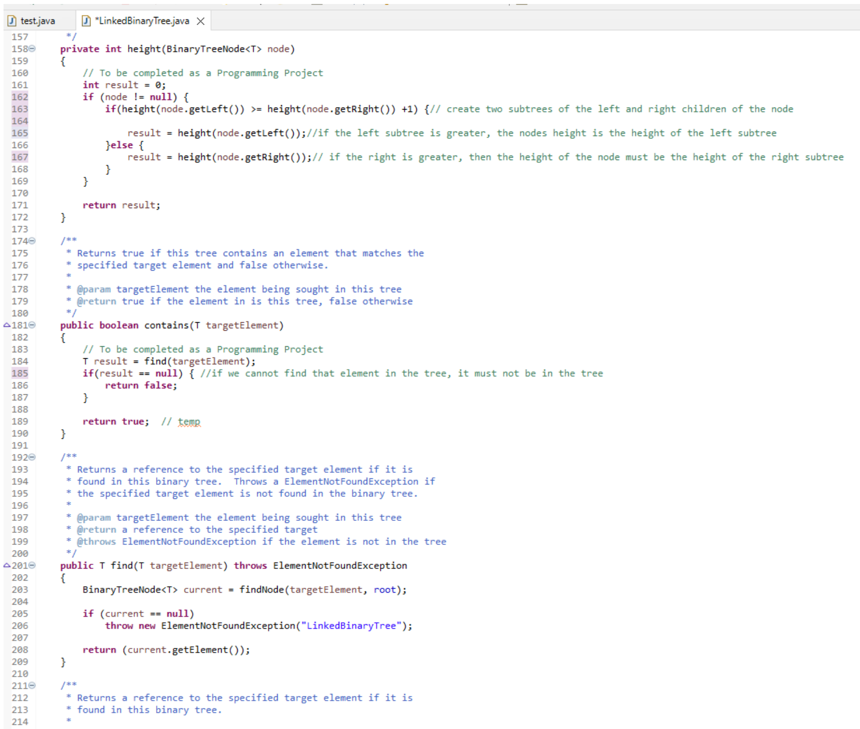
But how does this work? Let me show you:



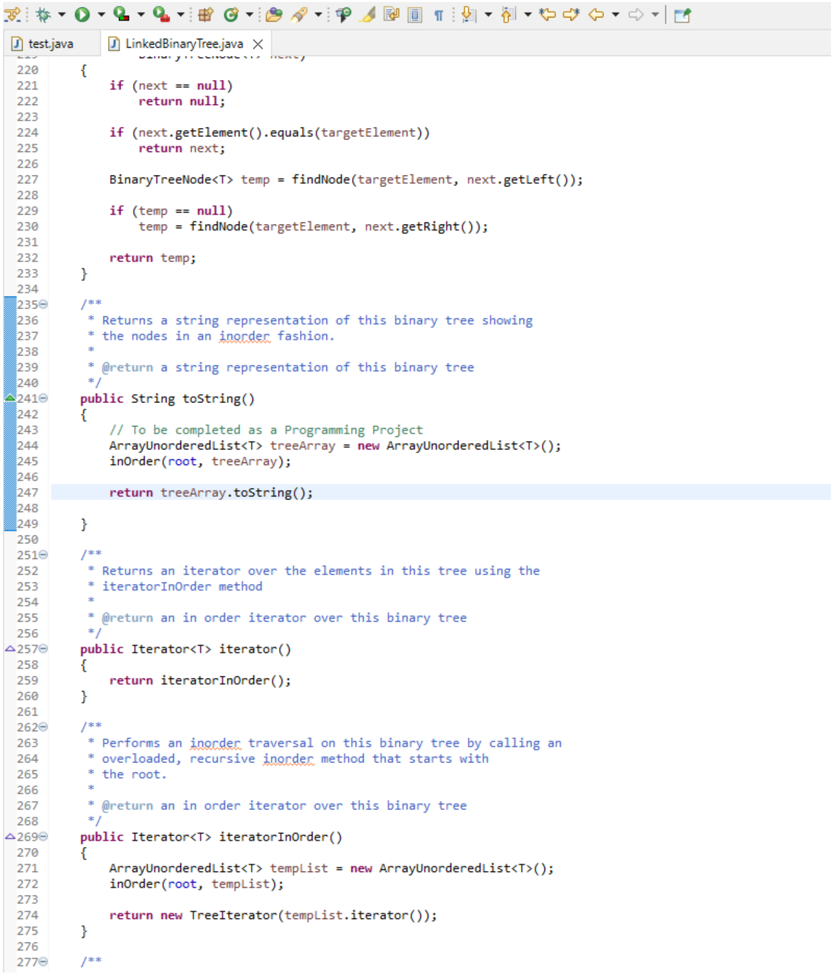
```java
        root = new BinaryTreeNode<T>(element);
        root.setLeft(left.root);
        root.setRight(right.root);
    }

    /**
     * Returns a reference to the element at the root
     *
     * @return a reference to the specified target
     * @throws EmptyCollectionException if the tree is empty
     */
    public T getRootElement() throws EmptyCollectionException
    {
        // To be completed as a Programming Project
        if(root == null) {
            throw new EmptyCollectionException("LinkedBinaryTree");//if there is no root then there is nothing to return, throw exception
        }
        T result = root.getElement();// return the value of the root
        return result;  // temp
    }

    /**
     * Returns a reference to the node at the root
     *
     * @return a reference to the specified node
     * @throws EmptyCollectionException if the tree is empty
     */
    protected BinaryTreeNode<T> getRootNode() throws EmptyCollectionException
    {
        // To be completed as a Programming Project
        if(root == null) {
            throw new EmptyCollectionException("LinkedBinaryTree");
        }

        return root;  // return the root itself
    }

    /**
     * Returns the left subtree of the root of this tree.
     *
     * @return a link to the left subtree fo the tree
     */
    public LinkedBinaryTree<T> getLeft()
    {
        // To be completed as a Programming Project
        if(root == null) {
            throw new EmptyCollectionException("LinkedBinaryTree");
        }
        LinkedBinaryTree<T> result = new LinkedBinaryTree<T>();//create a new tree where its root is the child to the left of the root
        result.root = root.getLeft();

        return result;  // temp
    }

    /**
     * Returns the right subtree of the root of this tree.
     *
     * @return a link to the right subtree of the tree
     */
```



```java
    /**
     * Returns the right subtree of the root of this tree.
     *
     * @return a link to the right subtree of the tree
     */
    public LinkedBinaryTree<T> getRight()
    {
        // To be completed as a Programming Project
        if(root == null) {
            throw new EmptyCollectionException("LinkedBinaryTree");
        }
        LinkedBinaryTree<T> result = new LinkedBinaryTree<T>();
        result.root = root.getRight();//create a new tree where its root is the child to the right of the root

        return result;  // temp
    }

    /**
     * Returns true if this binary tree is empty and false otherwise.
     *
     * @return true if this binary tree is empty, false otherwise
     */
    public boolean isEmpty()
    {
        return (root == null);
    }

    /**
     * Returns the integer size of this tree.
     *
     * @return the integer size of the tree
     */
    public int size()
    {
        // To be completed as a Programming Project
        int result = root.numChildren() + 1;// the size of the root would be the number of all children and the root itself

        return result;  // temp
    }

    /**
     * Returns the height of this tree.
     *
     * @return the height of the tree
     */
    public int getHeight()
    {
        // To be completed as a Programming Project
        int result = height(root)-1;
        return result;  // return the hieght of the root minus the root itself
    }

    /**
     * Returns the height of the specified node.
     *
     * @param node the node from which to calculate the height
     * @return the height of the tree
     */
    private int height(BinaryTreeNode<T> node)
```



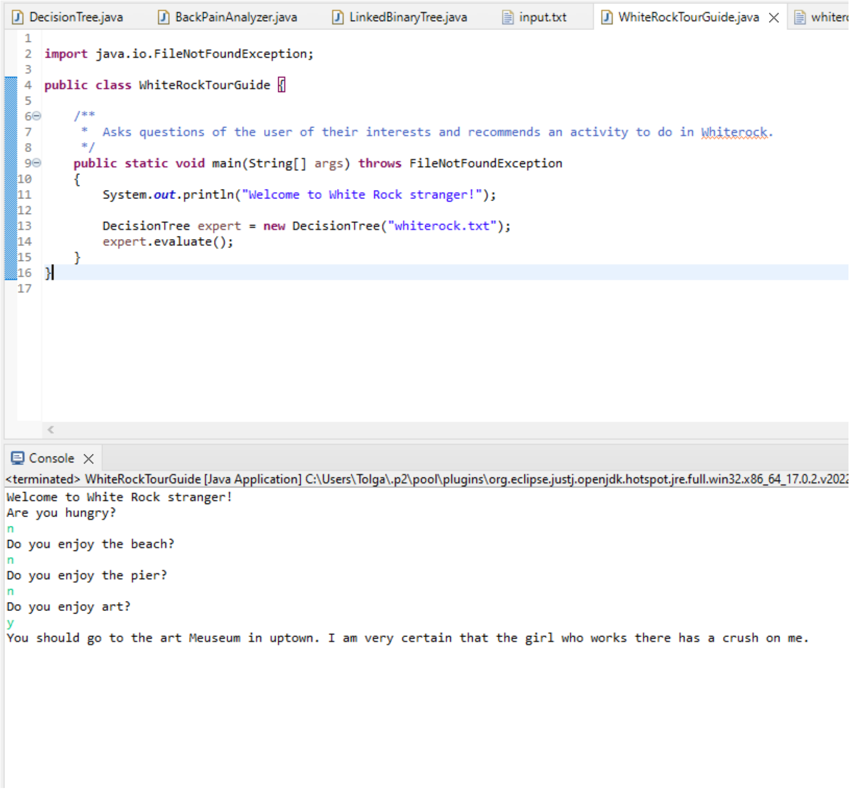
```java
     */
    private int height(BinaryTreeNode<T> node)
    {
        // To be completed as a Programming Project
        int result = 0;
        if (node != null) {
            if(height(node.getLeft()) >= height(node.getRight()) +1) {// create two subtrees of the left and right children of the node

                result = height(node.getLeft());//if the left subtree is greater, the nodes height is the height of the left subtree
            }else {
                result = height(node.getRight());// if the right is greater, then the height of the node must be the height of the right subtree
            }
        }

        return result;
    }

    /**
     * Returns true if this tree contains an element that matches the
     * specified target element and false otherwise.
     *
     * @param targetElement the element being sought in this tree
     * @return true if the element in is this tree, false otherwise
     */
    public boolean contains(T targetElement)
    {
        // To be completed as a Programming Project
        T result = find(targetElement);
        if(result == null) { //if we cannot find that element in the tree, it must not be in the tree
            return false;
        }

        return true;  // temp
    }

    /**
     * Returns a reference to the specified target element if it is
     * found in this binary tree.  Throws a ElementNotFoundException if
     * the specified target element is not found in the binary tree.
     *
     * @param targetElement the element being sought in this tree
     * @return a reference to the specified target
     * @throws ElementNotFoundException if the element is not in the tree
     */
    public T find(T targetElement) throws ElementNotFoundException
    {
        BinaryTreeNode<T> current = findNode(targetElement, root);

        if (current == null)
            throw new ElementNotFoundException("LinkedBinaryTree");

        return (current.getElement());
    }

    /**
     * Returns a reference to the specified target element if it is
     * found in this binary tree.
     *
```



```java
    {
        if (next == null)
            return null;

        if (next.getElement().equals(targetElement))
            return next;

        BinaryTreeNode<T> temp = findNode(targetElement, next.getLeft());

        if (temp == null)
            temp = findNode(targetElement, next.getRight());

        return temp;
    }

    /**
     * Returns a string representation of this binary tree showing
     * the nodes in an inorder fashion.
     *
     * @return a string representation of this binary tree
     */
    public String toString()
    {
        // To be completed as a Programming Project
        ArrayUnorderedList<T> treeArray = new ArrayUnorderedList<T>();
        inOrder(root, treeArray);

        return treeArray.toString();

    }

    /**
     * Returns an iterator over the elements in this tree using the
     * iteratorInOrder method
     *
     * @return an in order iterator over this binary tree
     */
    public Iterator<T> iterator()
    {
        return iteratorInOrder();
    }

    /**
     * Performs an inorder traversal on this binary tree by calling an
     * overloaded, recursive inorder method that starts with
     * the root.
     *
     * @return an in order iterator over this binary tree
     */
    public Iterator<T> iteratorInOrder()
    {
        ArrayUnorderedList<T> tempList = new ArrayUnorderedList<T>();
        inOrder(root, tempList);

        return new TreeIterator(tempList.iterator());
    }

    /**
```

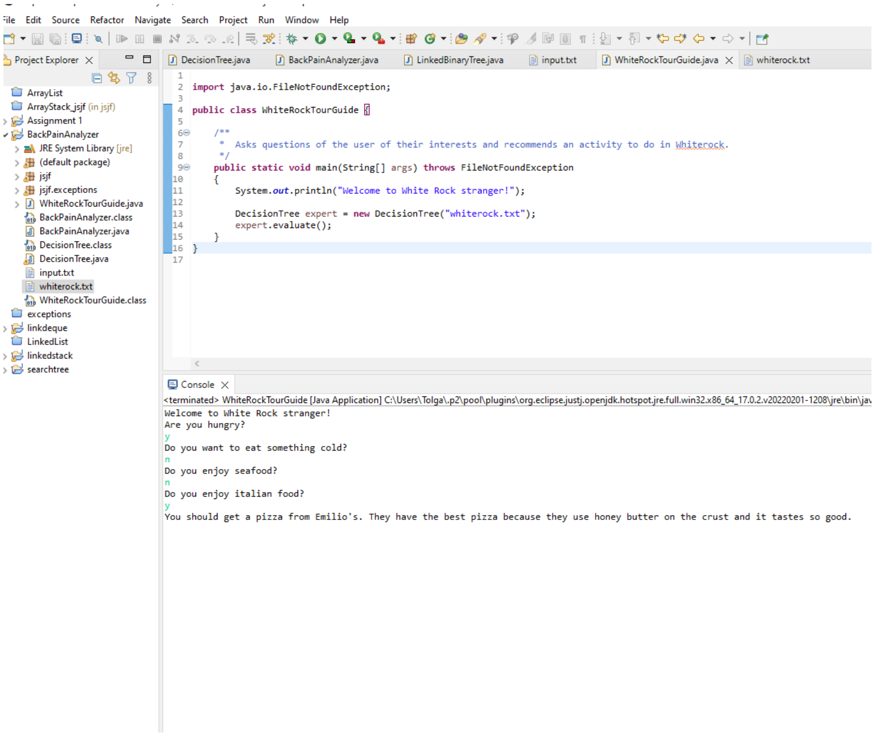
## 2. WHITEROCK DECISION TREE:

I made my own decision tree which is more complex than the back pain analyzer and has 22 children nodes..



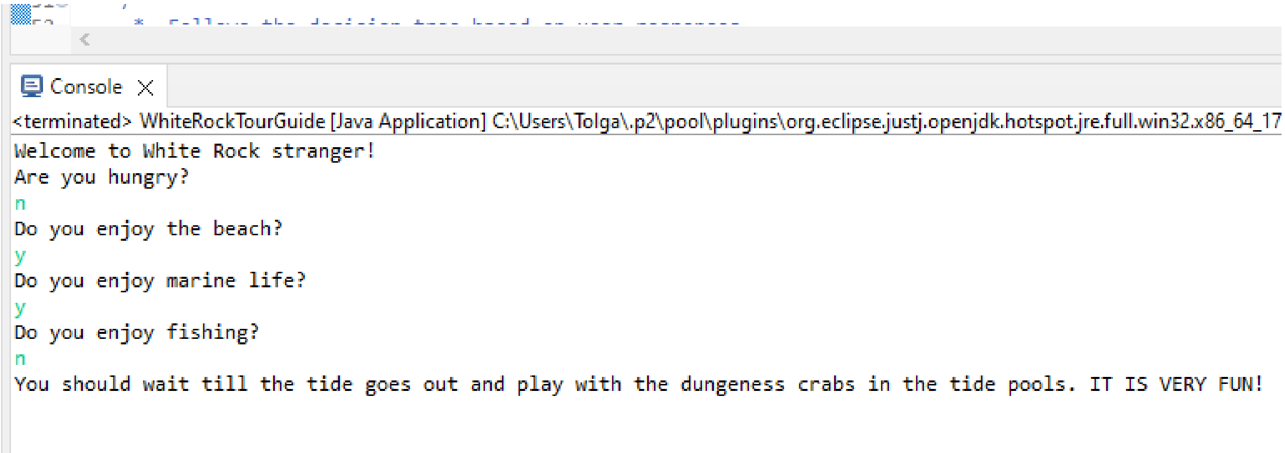
```

import java.io.FileNotFoundException;

public class WhiteRockTourGuide {

    /**
     *  Asks questions of the user of their interests and recommends an activity to do in Whiterock.
     */
    public static void main(String[] args) throws FileNotFoundException
    {
        System.out.println("Welcome to White Rock stranger!");

        DecisionTree expert = new DecisionTree("whiterock.txt");
        expert.evaluate();
    }
}

```

```
<terminated> WhiteRockTourGuide [Java Application] C:\Users\Tolga\.p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_17.0.2.v2022
Welcome to White Rock stranger!
Are you hungry?
n
Do you enjoy the beach?
n
Do you enjoy the pier?
n
Do you enjoy art?
y
You should go to the art Meuseum in uptown. I am very certain that the girl who works there has a crush on me.
```



```

import java.io.FileNotFoundException;

public class WhiteRockTourGuide {

    /**
     *  Asks questions of the user of their interests and recommends an activity to do in Whiterock.
     */
    public static void main(String[] args) throws FileNotFoundException
    {
        System.out.println("Welcome to White Rock stranger!");

        DecisionTree expert = new DecisionTree("whiterock.txt");
        expert.evaluate();
    }
}

```

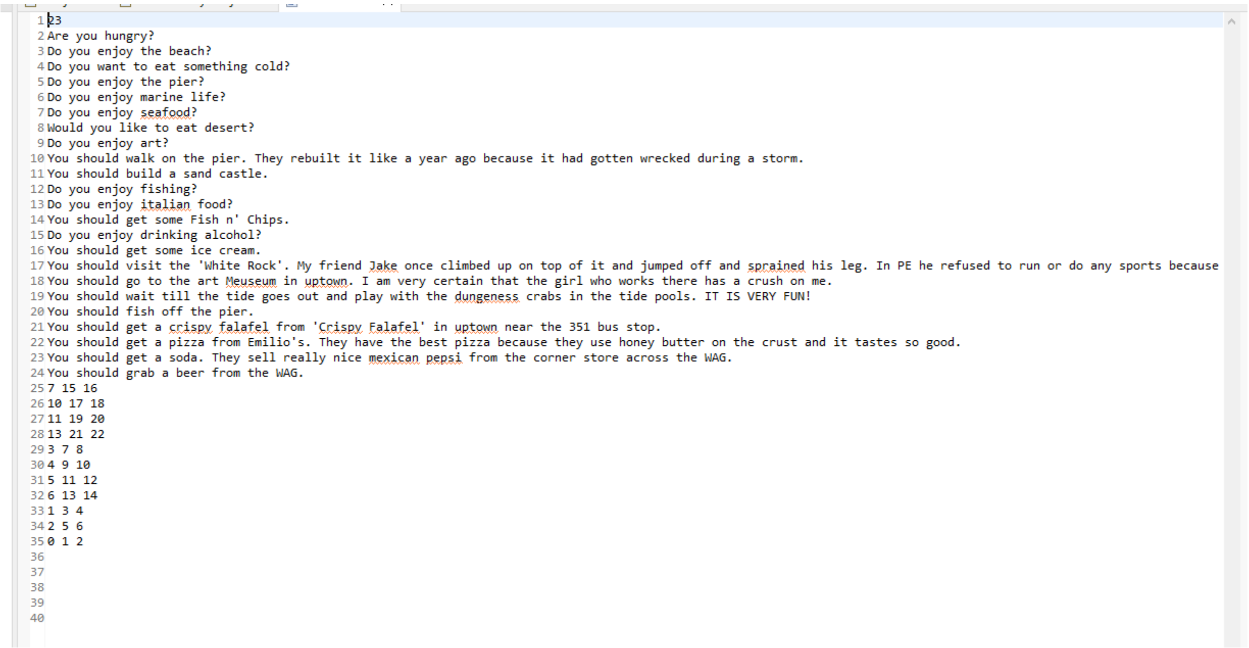
```
<terminated> WhiteRockTourGuide [Java Application] C:\Users\Tolga\.p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_17.0.2.v20220201-1208\jre\bin\jav
Welcome to White Rock stranger!
Are you hungry?
y
Do you want to eat something cold?
n
Do you enjoy seafood?
n
Do you enjoy italian food?
y
You should get a pizza from Emilio's. They have the best pizza because they use honey butter on the crust and it tastes so good.
```



```
<terminated> WhiteRockTourGuide [Java Application] C:\Users\Tolga\.p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_17
Welcome to White Rock stranger!
Are you hungry?
n
Do you enjoy the beach?
y
Do you enjoy marine life?
y
Do you enjoy fishing?
n
You should wait till the tide goes out and play with the dungeness crabs in the tide pools. IT IS VERY FUN!
```

The goal of this program is to be a tour guide for people who are new to WhiteRock and offer them suggestions on fun activities they can do based on the users preferences. The idea is to have it similar to an authentic conversation with someone from WhiteRock, as if you were speaking to them, and they were telling you what food you should try or place to visit etc.. This program works identically to the Backpain analyzer except for the fact that instead of using the tree from input .txt, it uses a tree I created called whiterock.txt

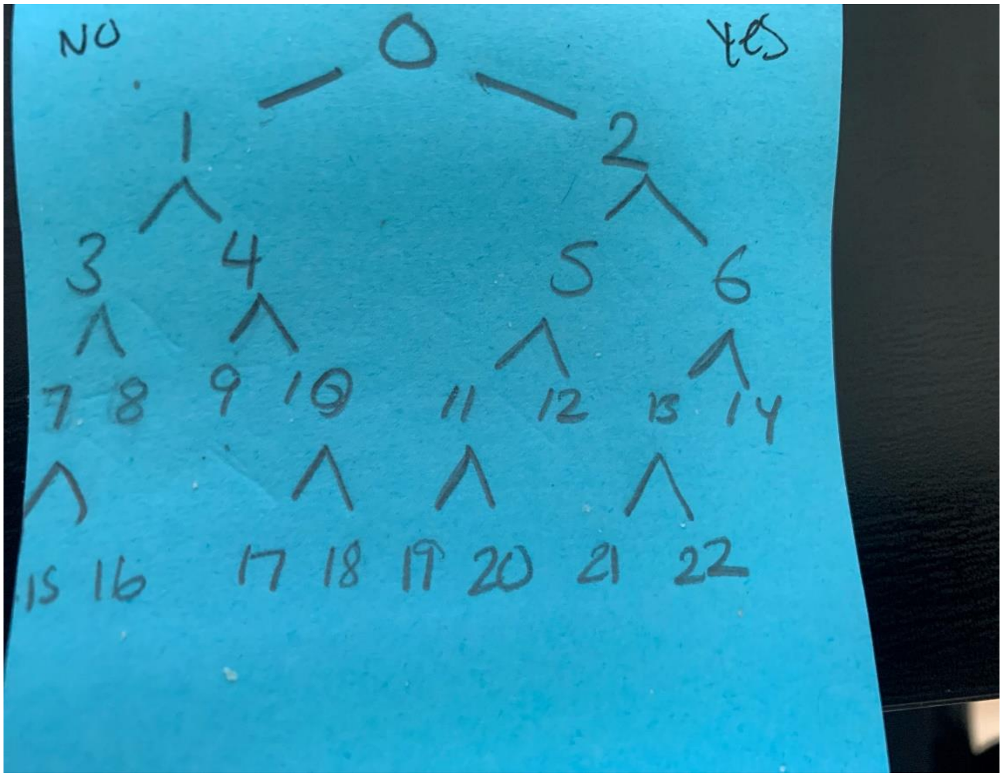
Let me show you what that looks like:



```
23
Are you hungry?
Do you enjoy the beach?
Do you want to eat something cold?
Do you enjoy the pier?
Do you enjoy marine life?
Do you enjoy seafood?
Would you like to eat desert?
Do you enjoy art?
You should walk on the pier. They rebuilt it like a year ago because it had gotten wrecked during a storm.
You should build a sand castle.
Do you enjoy fishing?
Do you enjoy italian food?
You should get some Fish n' Chips.
Do you enjoy drinking alcohol?
You should get some ice cream.
You should visit the 'White Rock'. My friend Jake once climbed up on top of it and jumped off and sprained his leg. In PE he refused to run or do any sports because
You should go to the art Meuseum in uptown. I am very certain that the girl who works there has a crush on me.
You should wait till the tide goes out and play with the dungeness crabs in the tide pools. IT IS VERY FUN!
You should fish off the pier.
You should get a crispy falafel from 'Crispy Falafel' in uptown near the 351 bus stop.
You should get a pizza from Emilio's. They have the best pizza because they use honey butter on the crust and it tastes so good.
You should get a soda. They sell really nice mexican pepsi from the corner store across the WAG.
You should grab a beer from the WAG.
7 15 16
10 17 18
11 19 20
13 21 22
3 7 8
4 9 10
5 11 12
6 13 14
1 3 4
2 5 6
0 1 2
```

The numbers on the bottom are used to organize the tree based on the children of each node

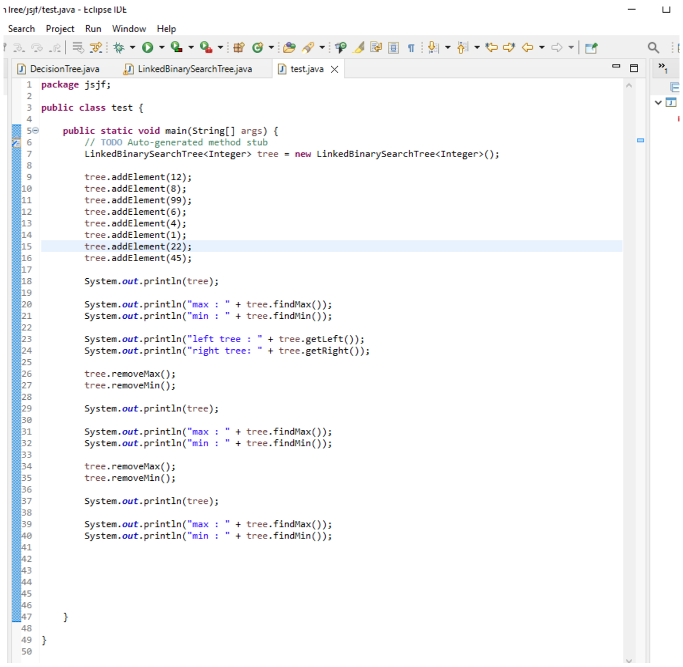
As you can see there are much more nodes and paths in this tree because it asks more questions. Heres a more simplified look of how this tree to give you a better idea of how it works:



Now you see..

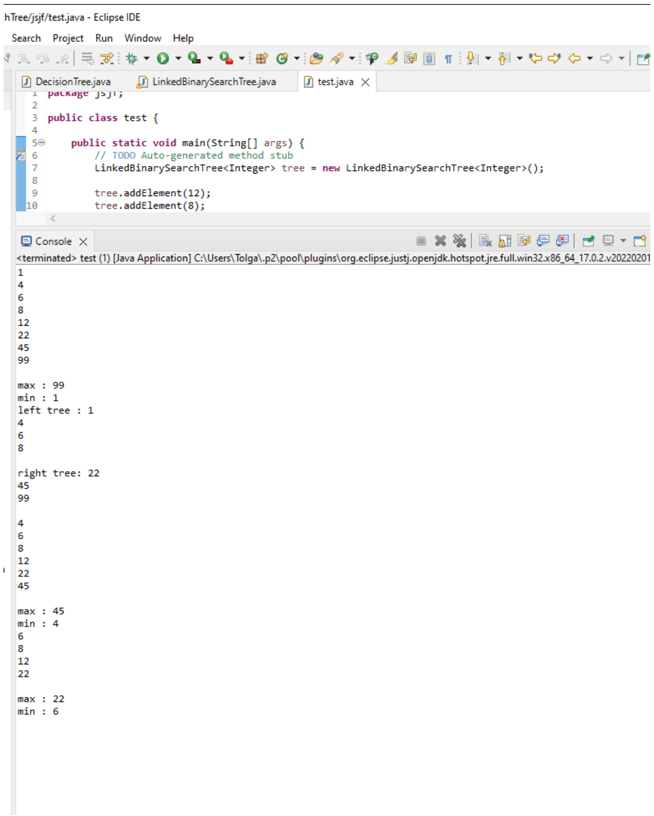
3. Linked Binary Search Tree

Here is the implementation of a binary search tree.


ı Iree/jsjf/test.java - Eclipse IDE
Search
Project
Run
Window
Help
DecisionTree.java
LinkedBinarySearchTree.java
test.java

```java
package jsjf;

public class test {

    public static void main(String[] args) {
        // TODO Auto-generated method stub
        LinkedBinarySearchTree<Integer> tree = new LinkedBinarySearchTree<Integer>();

        tree.addElement(12);
        tree.addElement(8);
        tree.addElement(99);
        tree.addElement(6);
        tree.addElement(4);
        tree.addElement(1);
        tree.addElement(22);
        tree.addElement(45);

        System.out.println(tree);

        System.out.println("max : " + tree.findMax());
        System.out.println("min : " + tree.findMin());

        System.out.println("left tree : " + tree.getLeft());
        System.out.println("right tree: " + tree.getRight());

        tree.removeMax();
        tree.removeMin();

        System.out.println(tree);

        System.out.println("max : " + tree.findMax());
        System.out.println("min : " + tree.findMin());

        tree.removeMax();
        tree.removeMin();

        System.out.println(tree);

        System.out.println("max : " + tree.findMax());
        System.out.println("min : " + tree.findMin());






    }

}

```

We will add elements to this integer tree, print the tree, find the max/min integers in the tree, create sub trees, and remove the min/max integers in the tree.


hTree/jsjf/test.java - Eclipse IDE
Search
Project
Run
Window
Help
DecisionTree.java
LinkedBinarySearchTree.java
test.java
Console

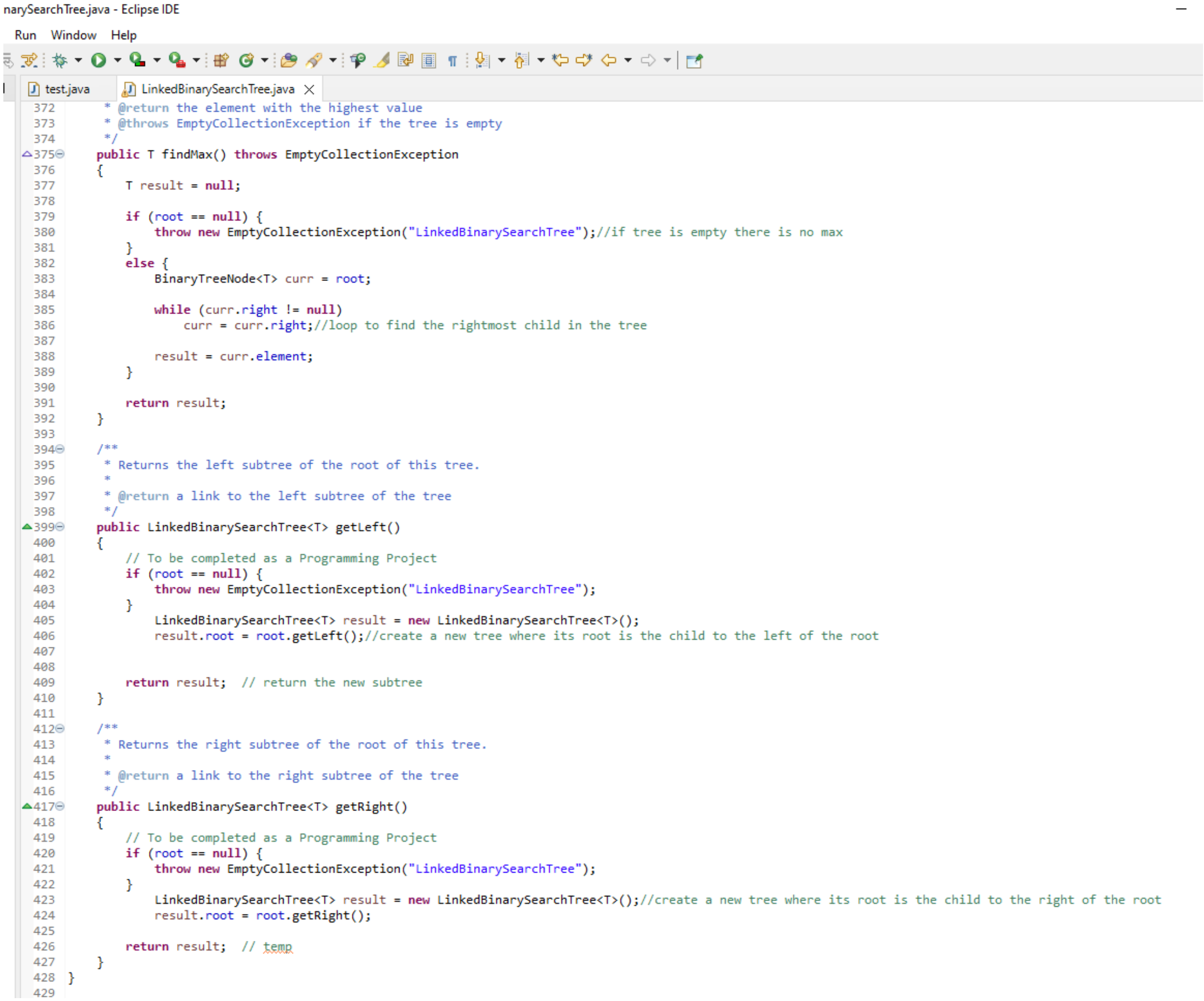
```java
package jsjf;

public class test {

    public static void main(String[] args) {
        // TODO Auto-generated method stub
        LinkedBinarySearchTree<Integer> tree = new LinkedBinarySearchTree<Integer>();

        tree.addElement(12);
        tree.addElement(8);
```

```
<terminated> test (1) [Java Application] C:\Users\Tolga\.p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_17.0.2.v20220201
1
4
6
8
12
22
45
99

max : 99
min : 1
left tree : 1
4
6
8

right tree: 22
45
99

4
6
8
12
22
45

max : 45
min : 4
6
8
12
22

max : 22
min : 6
```

Here are our results. As you can see it prints the tree in order of the lowest to highest integers, that’s because it prints using in-order. It accurately shows that the highest value is 99 and the lowest value is 1.

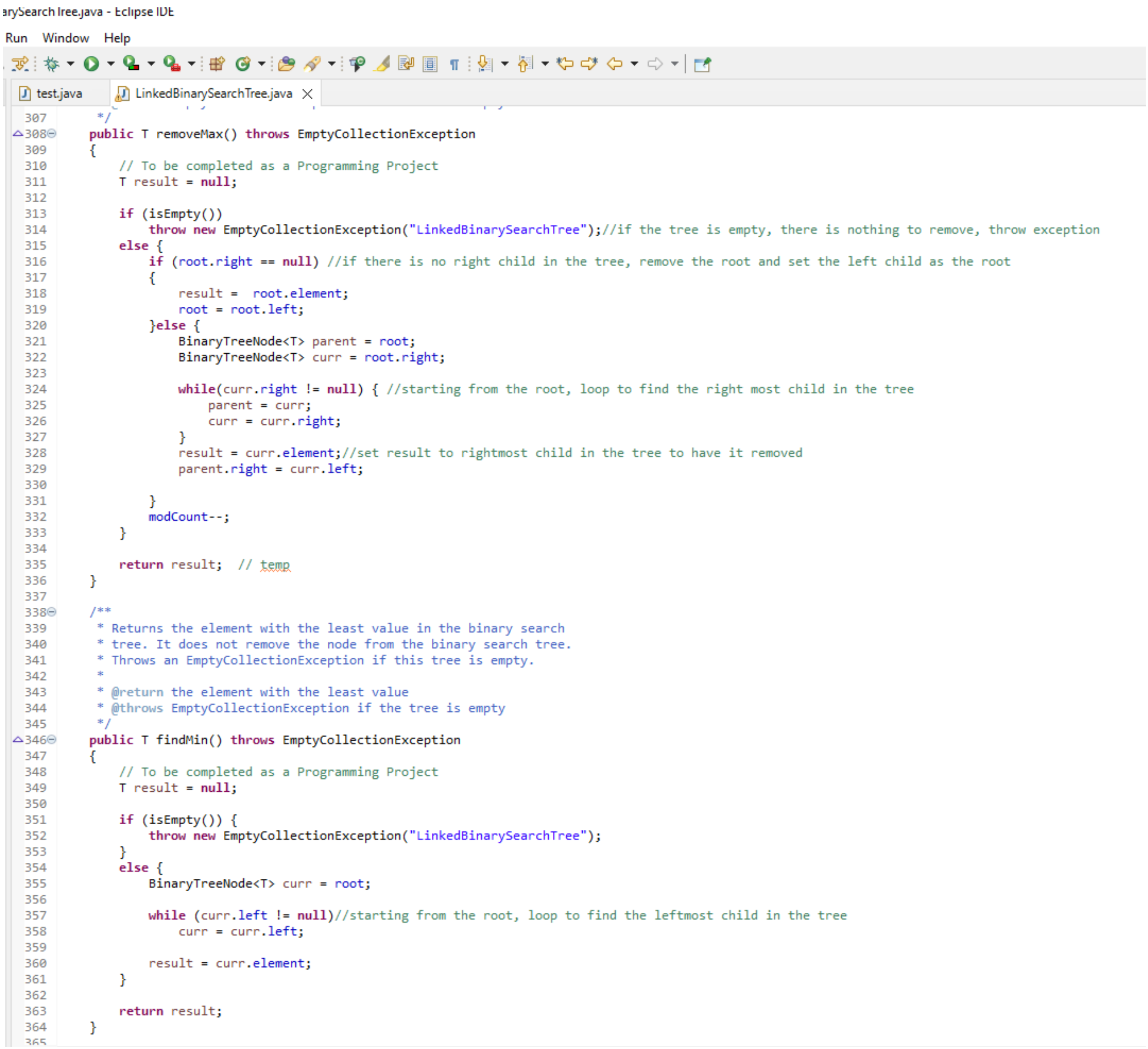
Then when we create the left  sub tree and the right subtree you can see that the left subtree contains all the values that are less than the root (12) and the right subtree contains all the values greater than 12. Notice how none of them contain 12? That’s because 12 was our root so when we create subtrees based on its children, it will not be present in those subtrees.

Lets take a look at how this all works:



narySearchTree.java - Eclipse IDE
Run Window Help
test.java
LinkedBinarySearchTree.java

```
     * @return the element with the highest value
     * @throws EmptyCollectionException if the tree is empty
     */
    public T findMax() throws EmptyCollectionException
    {
        T result = null;

        if (root == null) {
            throw new EmptyCollectionException("LinkedBinarySearchTree");//if tree is empty there is no max
        }
        else {
            BinaryTreeNode<T> curr = root;

            while (curr.right != null)
                curr = curr.right;//loop to find the rightmost child in the tree

            result = curr.element;
        }

        return result;
    }

    /**
     * Returns the left subtree of the root of this tree.
     *
     * @return a link to the left subtree of the tree
     */
    public LinkedBinarySearchTree<T> getLeft()
    {
        // To be completed as a Programming Project
        if (root == null) {
            throw new EmptyCollectionException("LinkedBinarySearchTree");
        }
            LinkedBinarySearchTree<T> result = new LinkedBinarySearchTree<T>();
            result.root = root.getLeft();//create a new tree where its root is the child to the left of the root


        return result;  // return the new subtree
    }

    /**
     * Returns the right subtree of the root of this tree.
     *
     * @return a link to the right subtree of the tree
     */
    public LinkedBinarySearchTree<T> getRight()
    {
        // To be completed as a Programming Project
        if (root == null) {
            throw new EmptyCollectionException("LinkedBinarySearchTree");
        }
            LinkedBinarySearchTree<T> result = new LinkedBinarySearchTree<T>();//create a new tree where its root is the child to the right of the root
            result.root = root.getRight();

        return result;  // temp
    }
}
```


arySearch Iree.java - Eclipse IDE
Run
Window
Help
test.java
LinkedBinarySearchTree.java

```java
     */
    public T removeMax() throws EmptyCollectionException
    {
        // To be completed as a Programming Project
        T result = null;

        if (isEmpty())
            throw new EmptyCollectionException("LinkedBinarySearchTree");//if the tree is empty, there is nothing to remove, throw exception
        else {
            if (root.right == null) //if there is no right child in the tree, remove the root and set the left child as the root
            {
                result =  root.element;
                root = root.left;
            }else {
                BinaryTreeNode<T> parent = root;
                BinaryTreeNode<T> curr = root.right;

                while(curr.right != null) { //starting from the root, loop to find the right most child in the tree
                    parent = curr;
                    curr = curr.right;
                }
                result = curr.element;//set result to rightmost child in the tree to have it removed
                parent.right = curr.left;

            }
            modCount--;
        }

        return result;  // temp
    }

    /**
     * Returns the element with the least value in the binary search
     * tree. It does not remove the node from the binary search tree.
     * Throws an EmptyCollectionException if this tree is empty.
     *
     * @return the element with the least value
     * @throws EmptyCollectionException if the tree is empty
     */
    public T findMin() throws EmptyCollectionException
    {
        // To be completed as a Programming Project
        T result = null;

        if (isEmpty()) {
            throw new EmptyCollectionException("LinkedBinarySearchTree");
        }
        else {
            BinaryTreeNode<T> curr = root;

            while (curr.left != null)//starting from the root, loop to find the leftmost child in the tree
                curr = curr.left;

            result = curr.element;
        }

        return result;
    }
```

It should be noted that the completion of **ArrayUnorderedList** was required to print the trees, since after all, an unordered array list is used to store all the values of the tree which are then converted to a string.