Trees - Summary

In this assignment, you will implement a bunch of different trees in the classroom and outside of the classroom. In the classroom you will create all the base classes: `Node`, `BSTNode`, `Tree`, `BSTTree`. You will be provided with a link to a video that provides you with the code necessary to start this project. Yout must first watch this video.

During lecture, you will have time to make a lot of progress on `BSTExtraNode` and `BSTExtraTree`, too. You should take a look at the TicTacToeNode implementation to get some idea on how to complete these classes.

You will only slightly modify TicTacToeNode and TicTacToeTree to get those to work. They should work once you plug in your `Board` class.

> **Important:** You may add any extra helper method you want so long as the provided Tests pass. You may not change the tests.

Base Classes

The `Node` and `Tree` classes work together.
The `BSTNode` is a **Binary Search Tree** Node. It has an integer value. It should have at most two children. Most of your functionality will be developed in the node classes.

BSTNode

You must implement the following in the classroom with Mr. Stride. If you miss class, look at the *partial soln* files in Canvas. However, students will have a difficult time understanding and completing this assignment if they do not implement the base classes themselves.

```java
public class BSTNode extends Node implements Comparable<BSTNode> {
   // See BSTNode.java for more details.
   public void setLeft(BSTNode node) {}
   public void setRight(BSTNode node) {}
   public BSTNode getLeft() {}
   public BSTNode getRight() {}
   public int getInfo() {}
   public String toString() {}
   public String getNodeDisplay() { }
   private void printPostOrder() {}
   private void printPreOrder() {}
   private void printInOrder() {}
}
```

BSTTree

When you correctly implement the BSTTree and BSTNode code, the application will display a Binary Search Tree with 10-20 random nodes with random data. The data value will be displayed in the node. (This is accomplished by overriding the `getNodeDisplay` method.) When you hover over a node, you'll see the some information displayed in the popup window. (This is accomplished by overriding the `toString` method.)

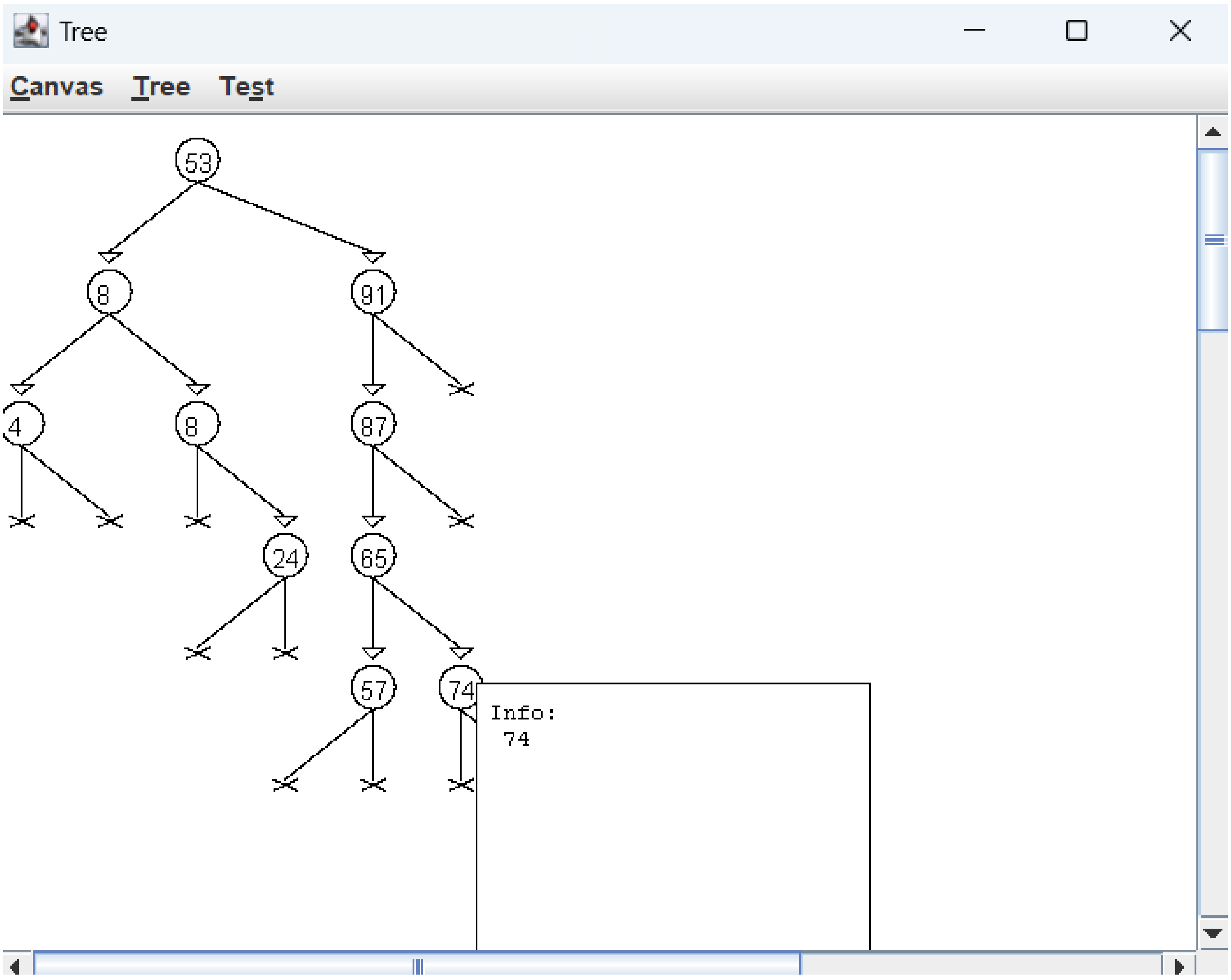
Furthermore, when the Tree is correctly implemented, there will be output in the console window when the user clicks on a node.
Sample Console Output

```
// User clicks on the root node (53) as seen below
Values in order:
4  8  8  24  53  57  65  74  87  91

// User SHIFT+clicks on the root node
Values in Post-order
4  24  8  8  57  74  65  87  91  53

// User CTRL+clicks on the root node
Values in Pre-Order
53  8  4  8  24  91  87  65  57  74
```

Here is an example of what your BST Tree may look like. Notice that in this tree you may have duplicate values. All child nodes to the left are less than the parent. All child nodes to the right are greater than or equal to the parent.



You must implement the following:

```
public class BSTTree extends Tree {
   public boolean add() { }
   public boolean add(int value) { }
   private boolean add(BSTNode toAdd) { }
   public static Tree createSomeTree() { }
}
```

BSTExtraNode

The BSTExtraNode is a BSTNode and holds extra information.

You should strive to get all of these done in class. Ask for help. Collaborate!!

```
public class BSTExtraNode extends BSTNode {
   // See BSTExtraNode.java for more details
   public String toString() { }
   public void processInfo() { }
   public boolean contains(BSTExtraNode other) { }
   private int countNodes() { }
   private int countHeight() { }
   private int sumTotal() { }
}
```

BSTExtraTree

Implement the following. See the file `BSTExtraTree.java` for details.

```
public class BSTExtraTree extends BSTTree {
   public boolean contains(int value) { }
   public boolean add(int value) { }
   public static BSTExtraTree createSomeTree() { }
}
```

When you correctly implement a BSTExtraTree, you'll have most of the same behavior that a BST Tree provided:

- Nodes will display their integer value
- Clicking on a node will display their traversal values

The BST Extra Tree created will have 15-25 random nodes. In this Tree, no duplicates are allowed. Each node will contain extra information beyond their integer value. Each node will have:

- The value (as stored in the parent class)
- The count of descendant nodes including itself

- The node's height. Leaf nodes have a height of 0
- The total value of all its descendants including itself

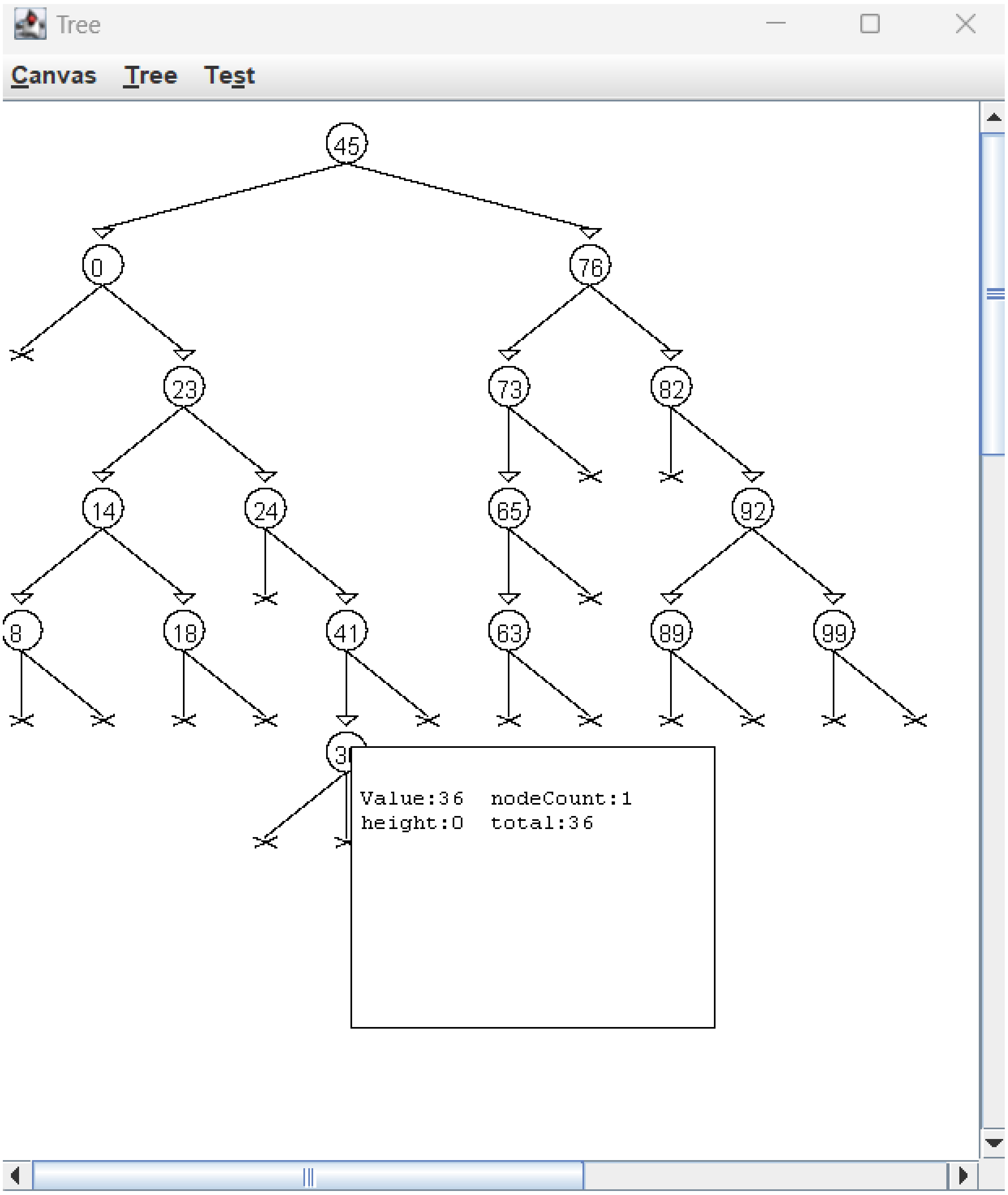
> Note that in this example, the popup window is on a leaf node. This means that the totalCount == 1 (just itself). And the total == 36 (just its own value). Its height == 0 because it is a leaf. Hovering over internal nodes will have different values.

Node Calculations

Each calculation will be done recursively. Simply consider how a single node can accomplish its own calculation in isolation. Here are more specifics:

- A node's height is: 1 + max(leftChild's height, rightChild's height)
- A node's total (sum) is: this node's value + rightChild's sum + leftChild's sum
- A node's count is: 1 + leftChild's count + rightChild's count

Each of these calculations should be done once and then cached in the node's instance field. The `toString` method will simply format and display the instance fields.



```
Value:36  nodeCount:1
height:0  total:36
```

createSomeTree

Each Tree class has a `static` method `createSomeTree` that you **must implement.** Each method will create a Tree with the correct size and properties.

TicTacToe Tree

The TicTacToe Tree will leverage the `Board` class that you developed in an earlier homework assignment. If you have implemented the Board class correctly, the full TicTacToe tree will be created for you. Each node in the TicTacToeTree will have the following information displayed in the popup window:

- The current board
- The count of games X will win from this node

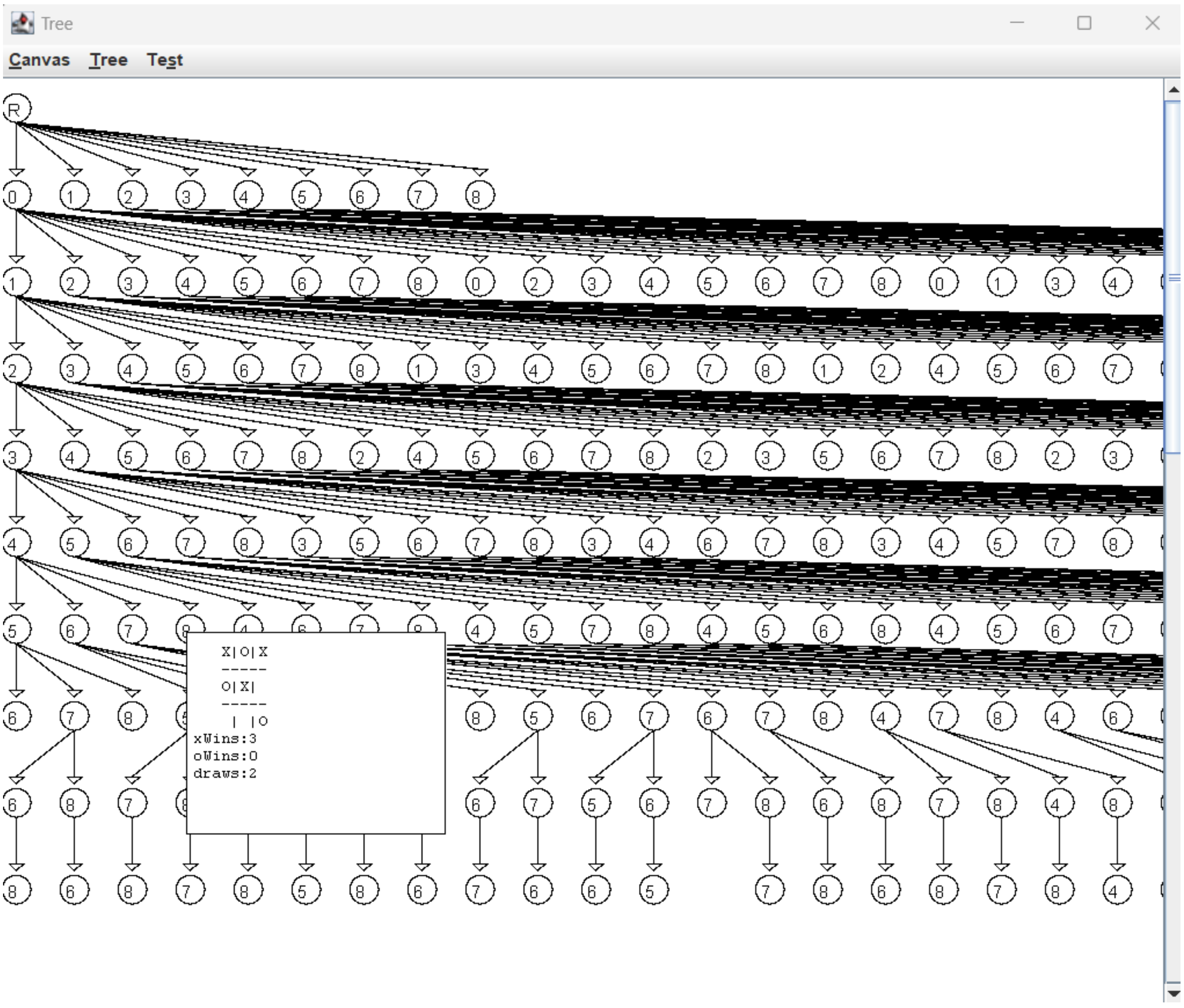
- The count of games O will win from this node
- The count of games that end up in a Draw from this node
- The Move made from the parent node to get to this node

There are a few reasons why this TicTacToe code is provided for you.

- To use as a reference to implement BSTExtraTree.
- Many students have found that implementing the TicTacToe Tree to be too difficult to do on their own. Consider deleting the code and reimplementing yourself. Are you *that* good?!
- This is an illustration of how properly implemented classes (The Board class) can be reused in a multitude of interesting ways!

The TicTacToe tree will be huge! The display is truncated. If you want to drill into a specific node, double-click a node. To escape out back to the full tree, right-click.

Each node should display the Move made as a number. The popup window should display the TicTacToe board as well as the game statistics. See the example below. In the example you'll see that the root node displays with an "R". The other nodes display a number 0-8 which represents the move location made from the parent. You'll see that the popup windows shows a board after six moves were made. The last move was at location 8 (the bottom-right).



```
  X|O|X
  -----
  O|X|
  -----
   | |O
xWins:3
oWins:0
draws:2
```

Files to Submit

You must submit the following NINE (9) files on https://css.MrStride.com

- `Node.java`
- `Tree.java`
- `BSTTree.java`
- `BSTNode.java`
- `BSTExtraNode.java`
- `BSTExtraTree.java`
- `Board.java`
- `TicTacToeNode.java`
- `TicTacToeTree.java`

The following files are provided for you. Do NOT submit them. Do NOT change them.

- `DrawInfo.java`
- `Main.java`
- `Move.java`
- `TestStudent.java`
- `TreePanel.java`

Grading

This project is graded on mostly functionality. However, note the following:

- Comment *quality and quantity* will *NOT* be graded on this assignment.
- Conventions & Style means that **spacing on comments is still graded.**
- All `TODO` comments need to removed because the student implemented the requirements.
- All mouse over behavior on the nodes must work.
- All trees should be created and the information in the nodes should be fully visible in the pop-up window. This means that there should be a new line in the toString implementation.
- The TicTacToe board should fully draw in the pop-up window.
- Be sure to run the Tests provided in the Menu.
- The GUI needs to compile and run. Student should run `Main.java` and not rely solely on `TestStudent.java`.

Rubric

```
Points     Description
------     ----------------------------
    20     Pass functional requirements
     5     Conventions & style
------
    25     TOTAL
```