Report Mini-project Object-Oriented Programming

**Topic: Visualization of operations on tree data structure**

**Team 13**

1. **Mini-project description.**
2. **Types of tree and operation.**

 Tree is a useful data structure with lots of applications in computer science. In this project, we will design a program to display and explain some basic operations for trees.

In the visualization, the program only considers undirected-weight trees, with integer node values and no duplicated node values allowed.

There are four types of trees we consider:

* Generic tree (no special properties): A tree is a nonlinear hierarchical data structure

that consists of nodes connected by edges and contains no cycles.

* Binary tree: A binary tree is a tree where each node has at most two children.
* Balanced tree: A balanced tree is a tree where each leaf node is not more than a difference of value k in depth away from the root than any other leaf.

(User is asked to input an integer k which specifies the maximum depth-differences between any two leaf nodes)

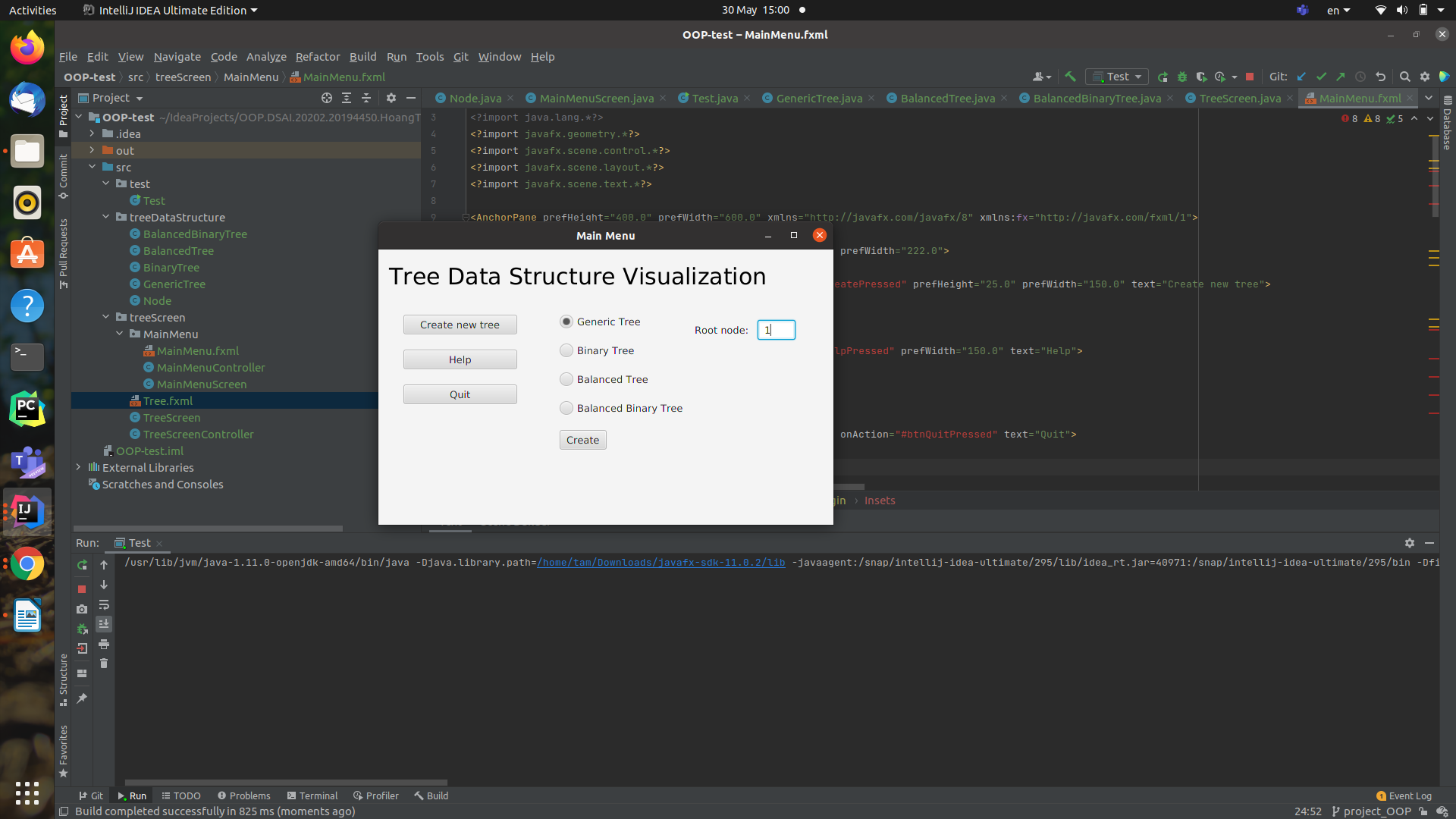
* Balanced binary tree: A balanced binary tree has the properties of both a binary tree and a balanced tree.

There are 6 basic operations implemented:

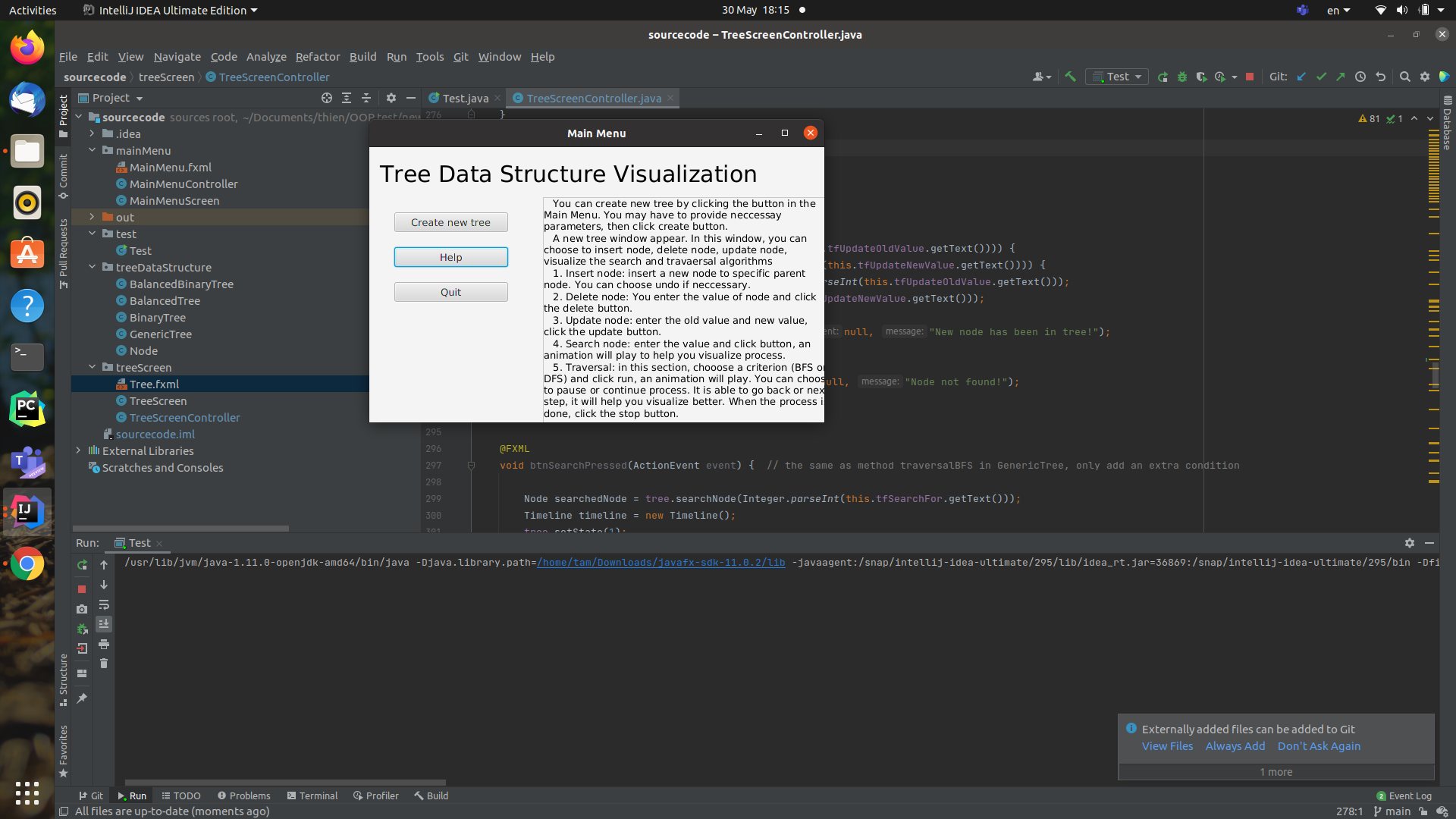
* Create: this operation takes “node” as input and output a tree with the node as root node.
* Insert: this operation takes “value of the parent node” and “value of a new node” as input and adds the new node with specified value as a child of the specified parent node to the tree.
* Delete: this operation takes “value of a node” as input and deletes the node with that value from the tree.
* Update: this operation takes “current value of the node” and “new value of the node” as input and changes the node with current value to the new value.
* Traverse: this operation takes “algorithm” (DFS or BFS) as input and shows the traversal of all nodes in the tree corresponding to the algorithm (highlight current node in each step of traversal).
* Search: : this operation takes “search value” as input and output the node of the tree with the search value.

1. **Main menu**

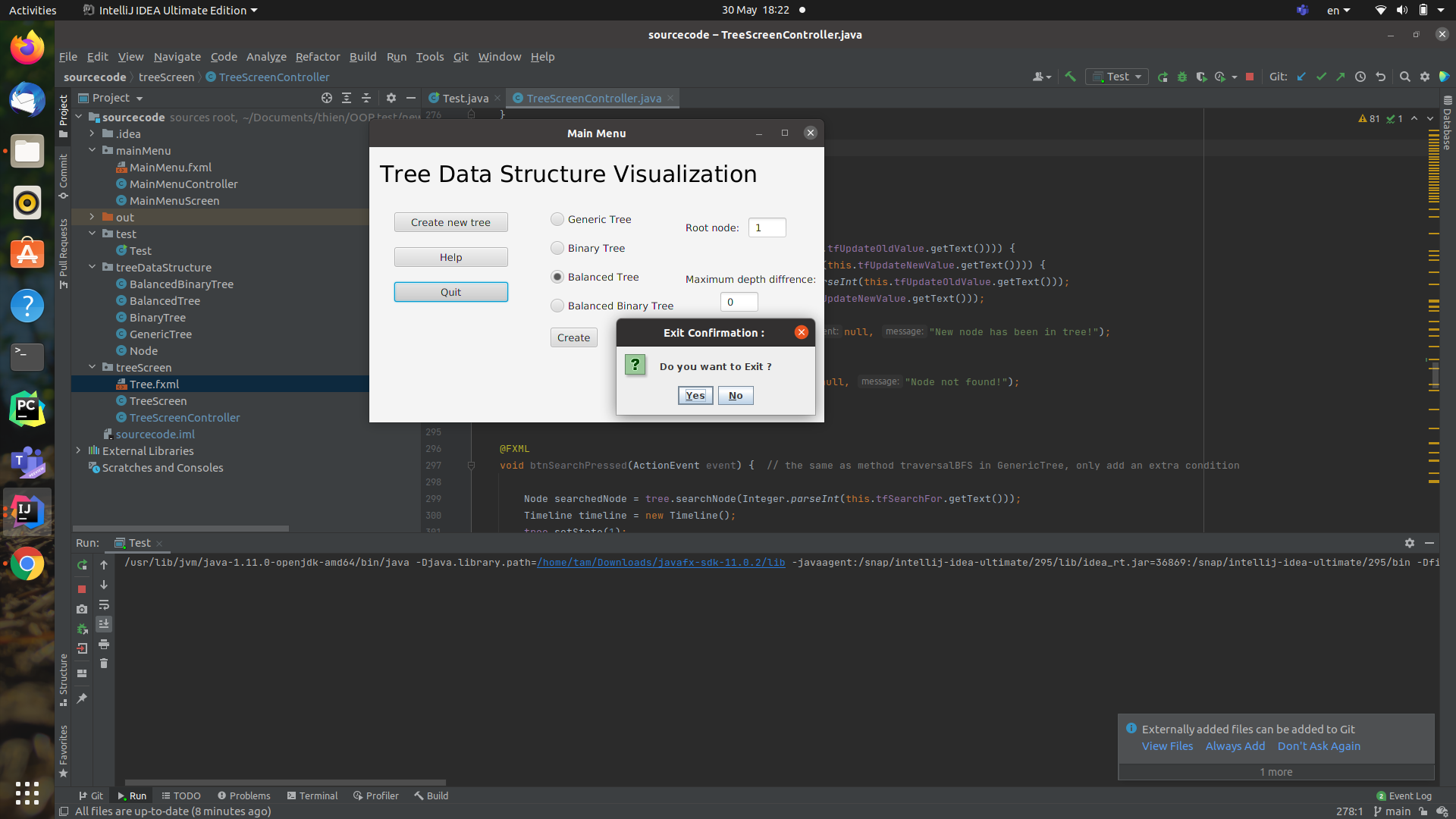
When running the visualization, we start at the main menu screen which consists of 3 buttons: Create new tree, help, quit.



* “Create new tree”: this button shows a text field to identify root node value and 4 options to choose the type of tree user wants to visualize: Generic tree, Binary tree, Balanced tree and Balanced binary tree.
  + When we choose Balanced tree or Balanced binary tree, another text field will appear and ask for the maximum depth-different value k.
  + After user has chosen the type of tree and given the sufficient information, pressing the “Create” button will take the user to the visualized screen.



* “Help”: this button will show an instruction on how to use the visualization and the aim of the project.



* “Quit”: this button will show a confirm message before exit the visualized program.

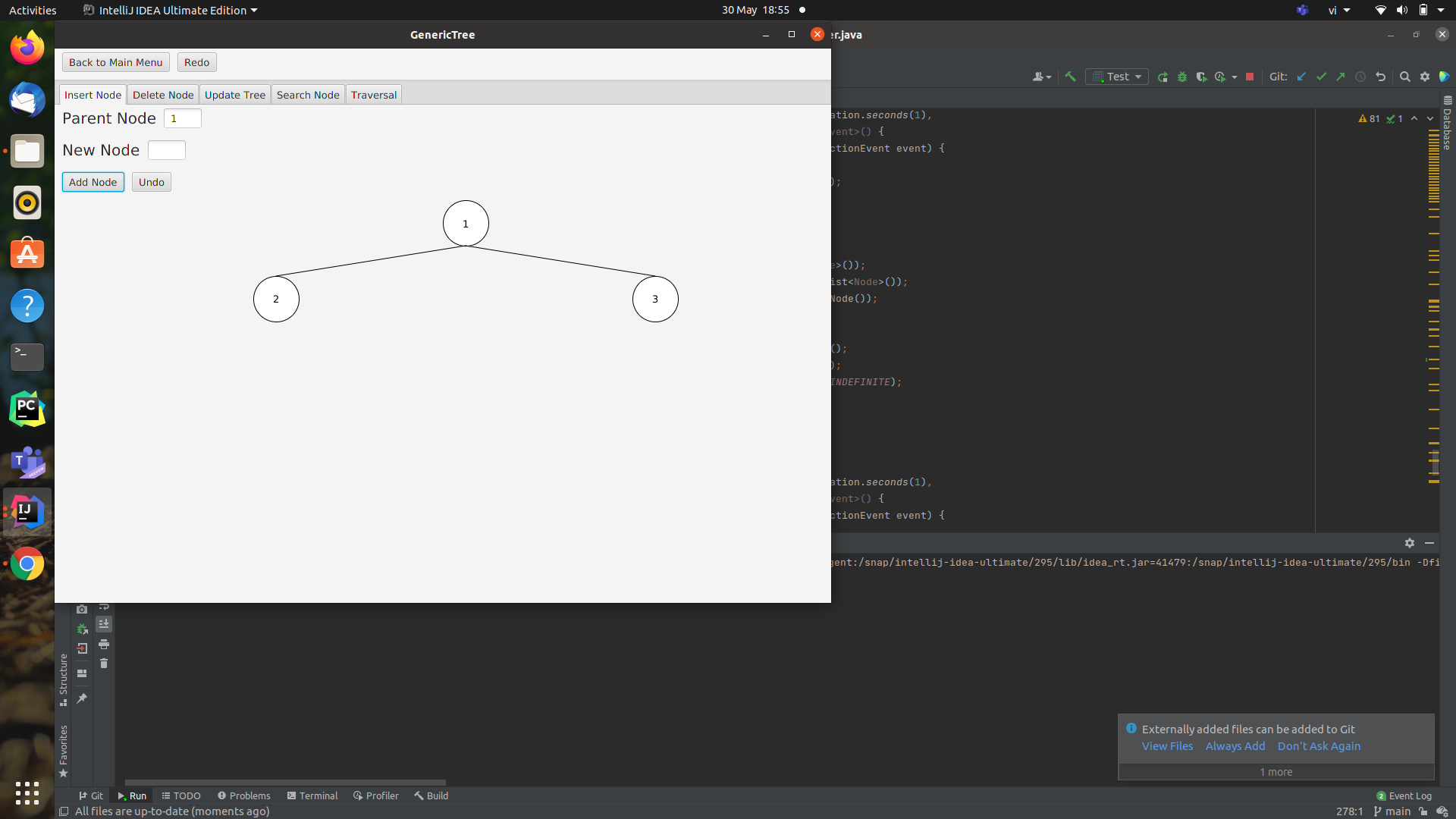
1. **Visualized screen.**

With the type of tree already specified in the main menu screen, this screen visualizes the steps of the basic operations applied to the tree. There is always a “back to main menu” button for the user to return to the main menu.

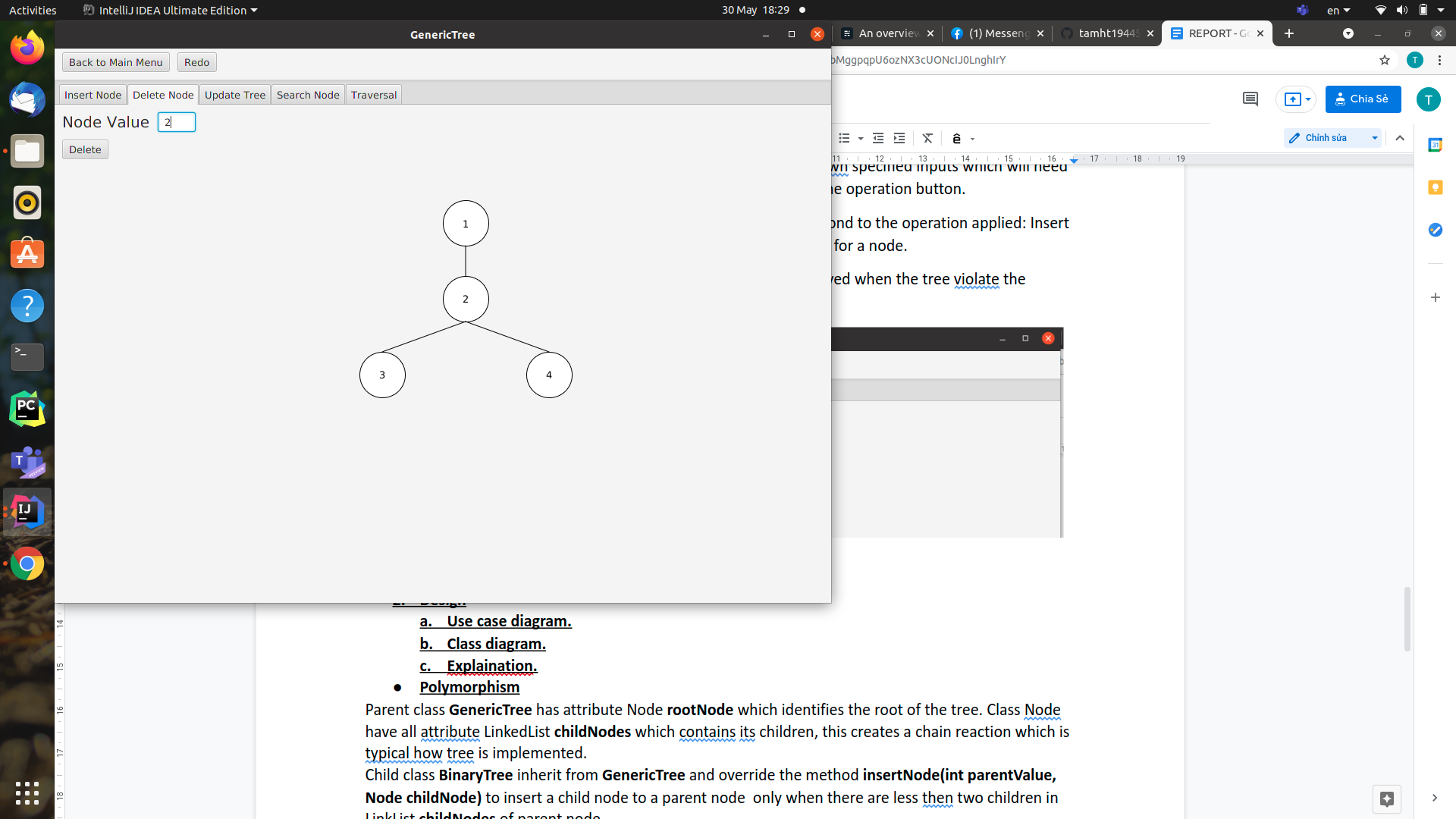
In this screen, users can choose, from the options bar, one of the 5 basic operations: “Insert”, “Delete”, “Update”, “Traverse” and “Search”. Each operation has its own specified inputs which will need to be filled in by the user in the text fields appearing when the user clicks the operation button. There will always be a notification for the user when the value entered is not suitable (insert value node already existed,etc...).

The tree consists of nodes under the options bar that will change corresponding to the operation applied: Insert a node, Delete a node , Update a node, Traverse the tree and Search for a node. There will always be an “Undo” button to turn back 1 step and a “Redo” button to turn back to only node 1.

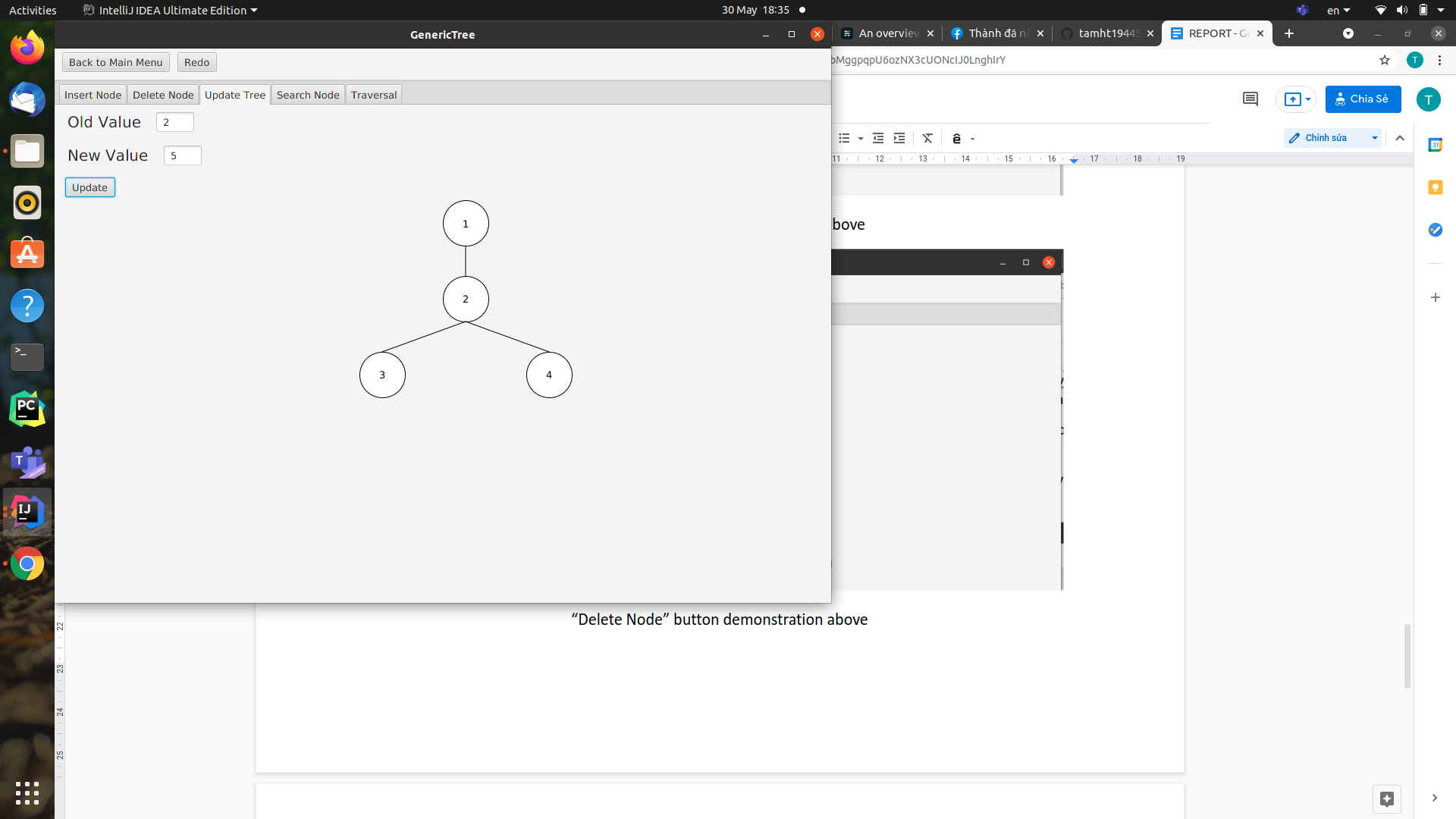
For Balanced Tree and Balanced Binary Tree, a warning will be displayed when the tree violates the balanced property.



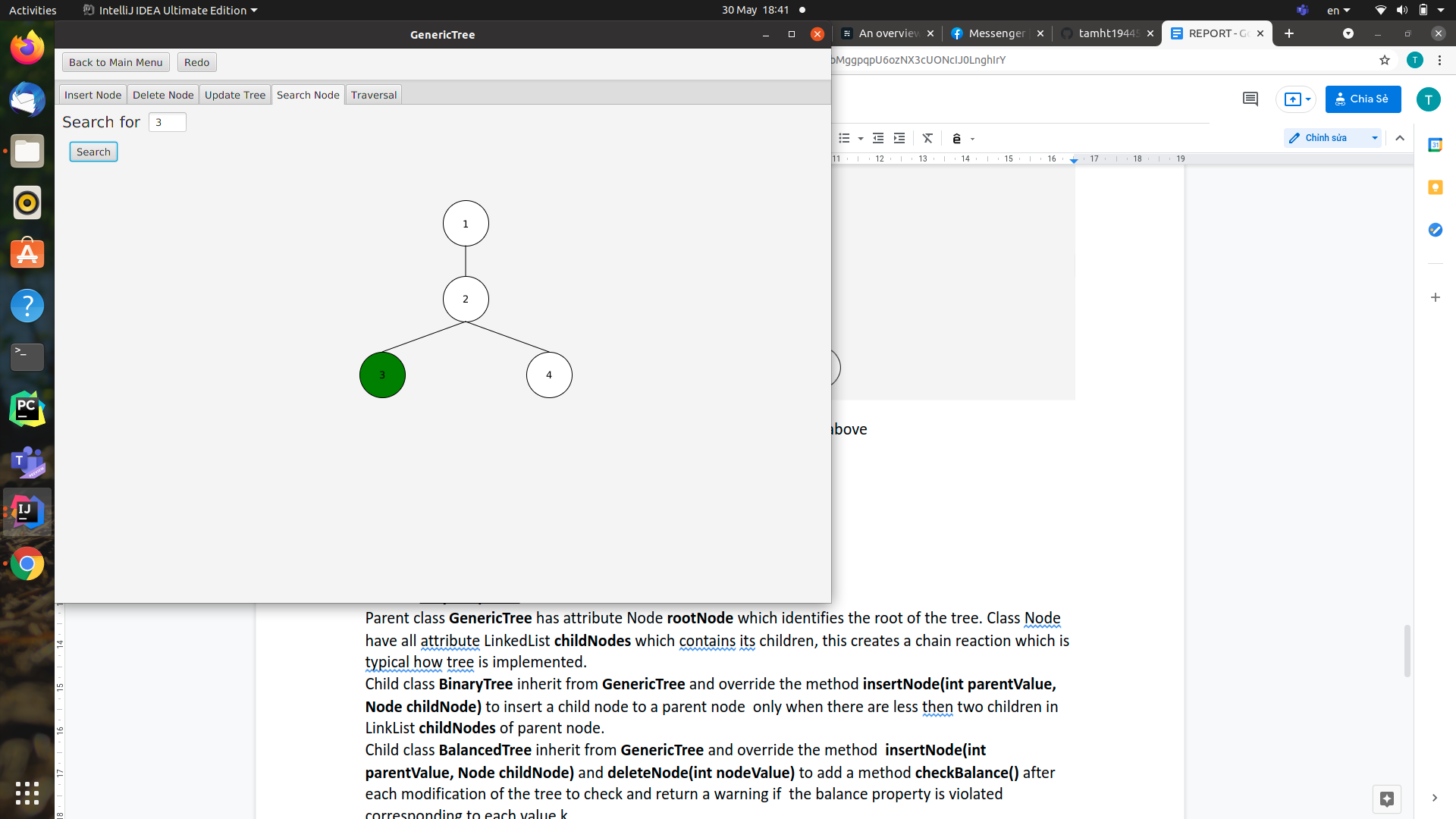
“Insert Node” button demonstration above



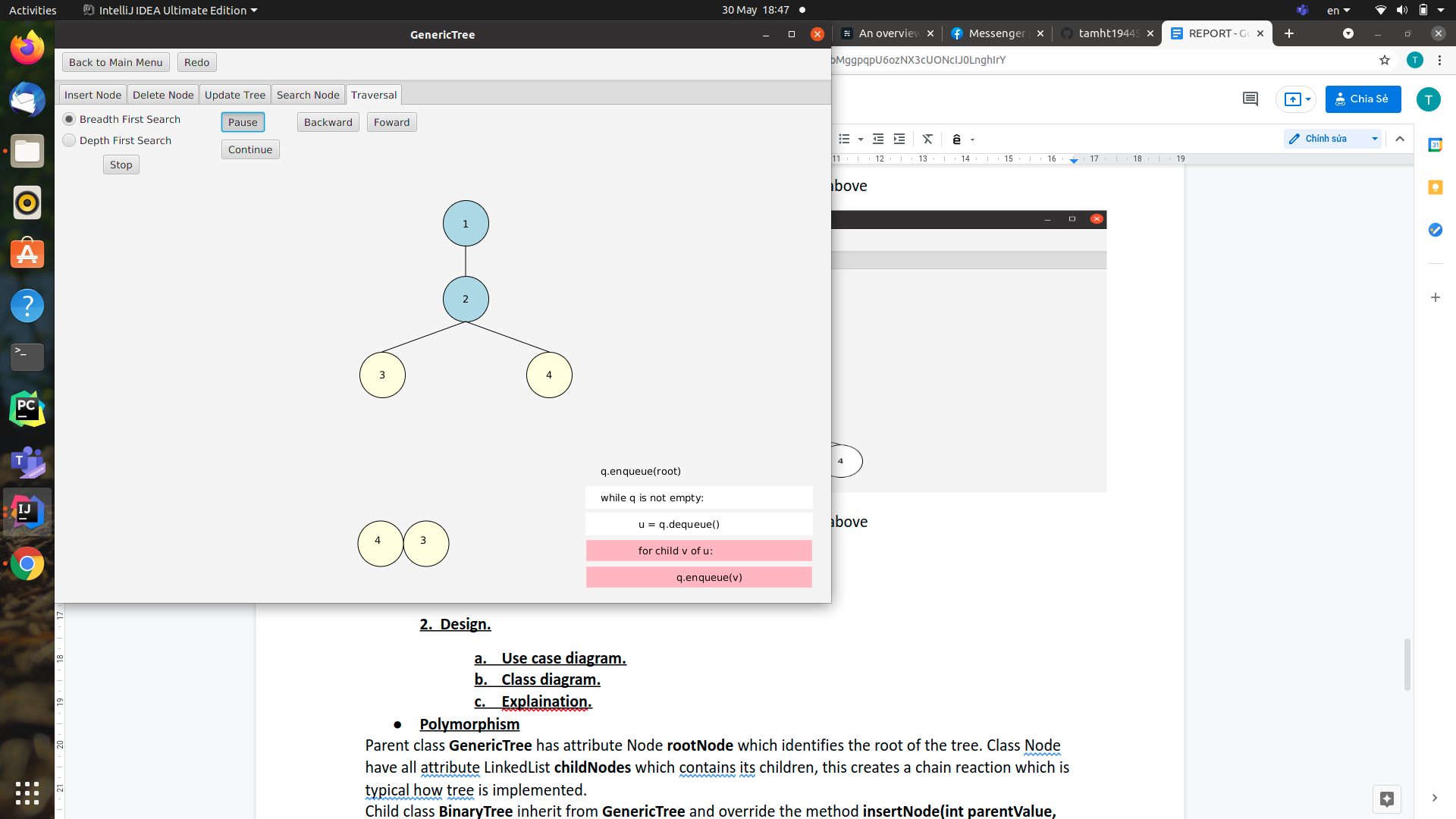
“Delete Node” button demonstration above



“Update Tree” button demonstration above



“Search Node” button demonstration above

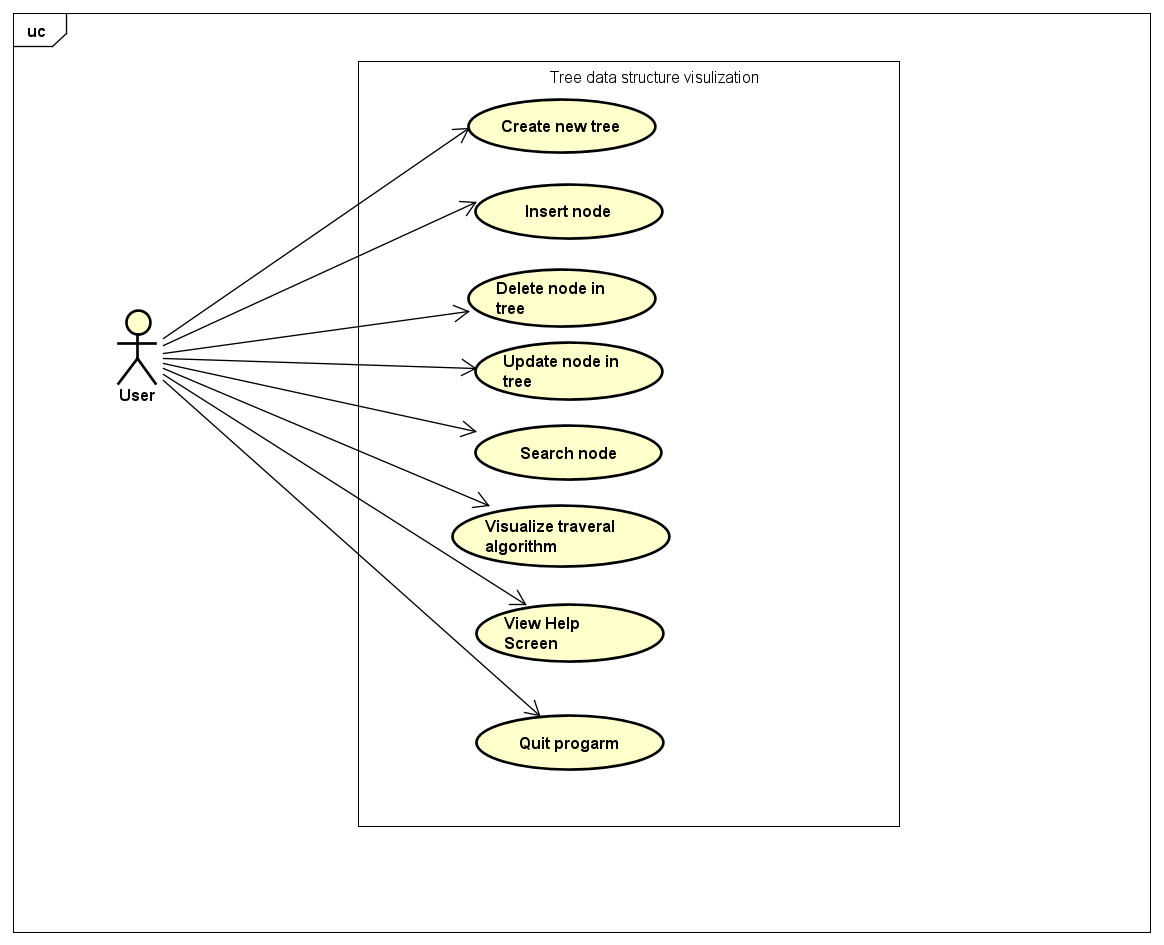


“Traversal” button demonstration above

"Traversal" button has additional features: “Stop”, “Pause”, “Continue”, “Backward” and “Forward” to manipulate the steps in the process of traversal through the tree. Moreover, the pseudo code highlights each step of the process.

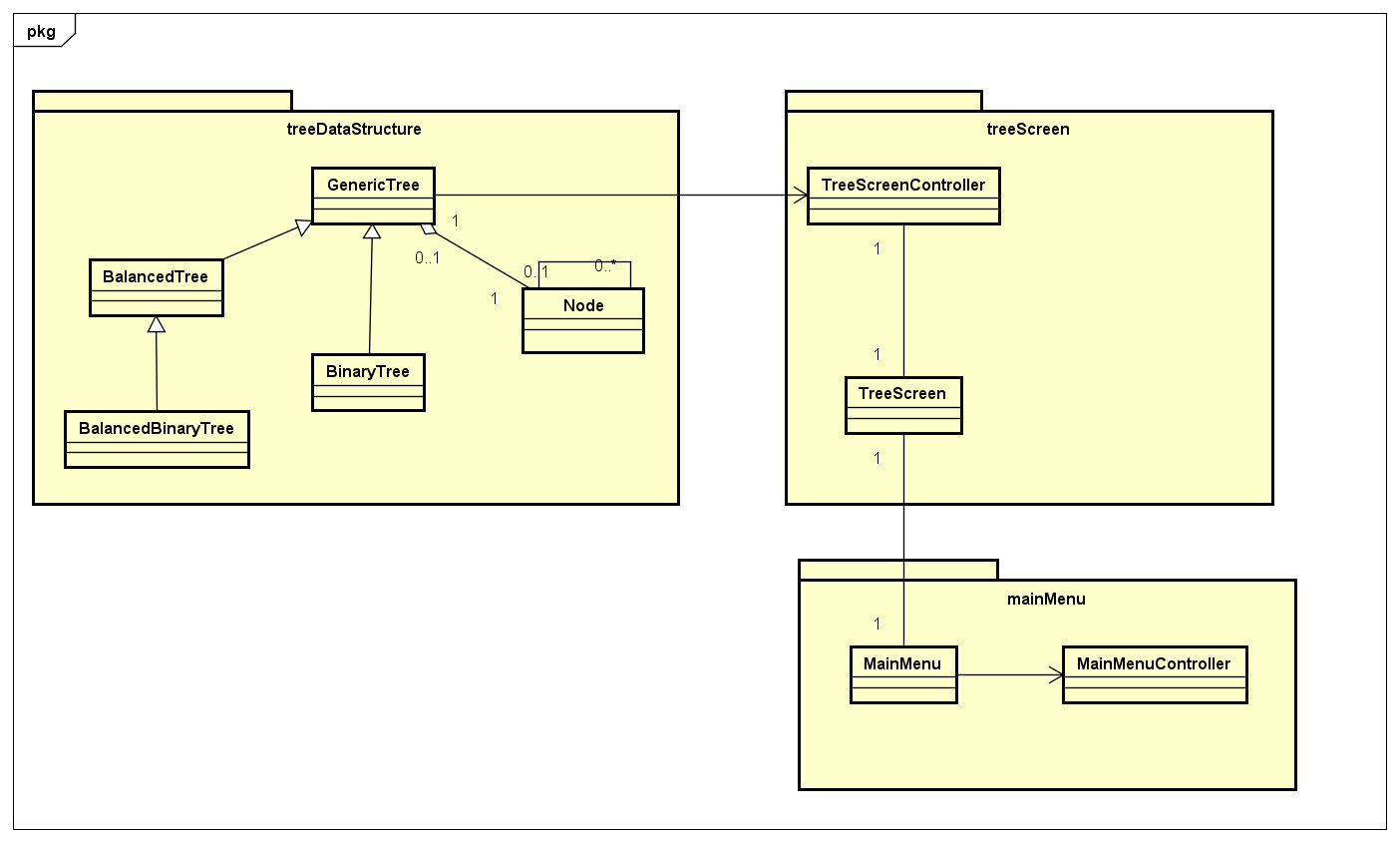
**2. Design.**

1. **Use case diagram.**

****

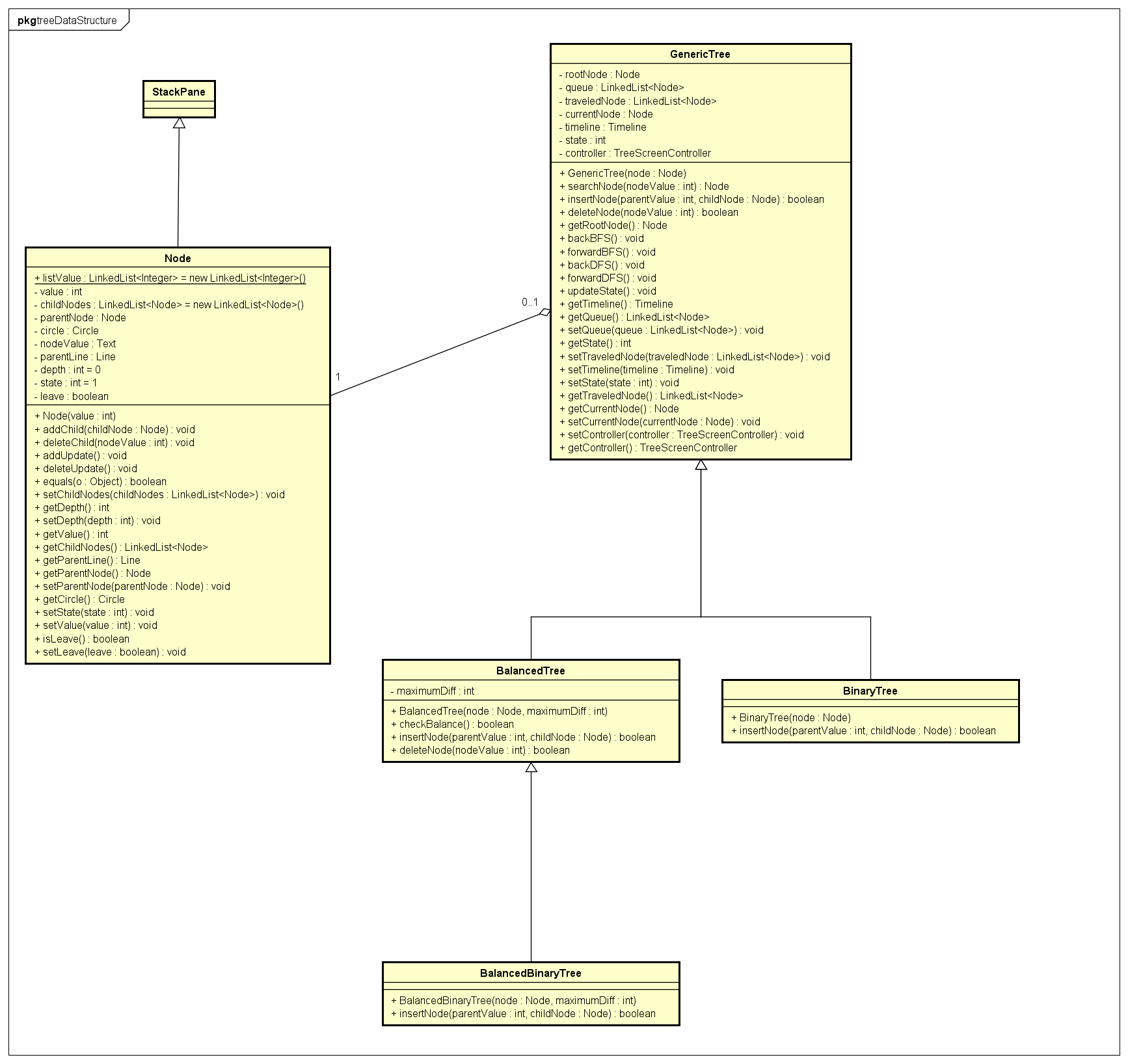
Our program has eight use cases, in which the program performs a sequence of actions and gives observable results to the user.

1. **Class diagram.**

****

The general class diagram explains the overview of our program. In this pic, we can see all of the class and how they associate with each other.

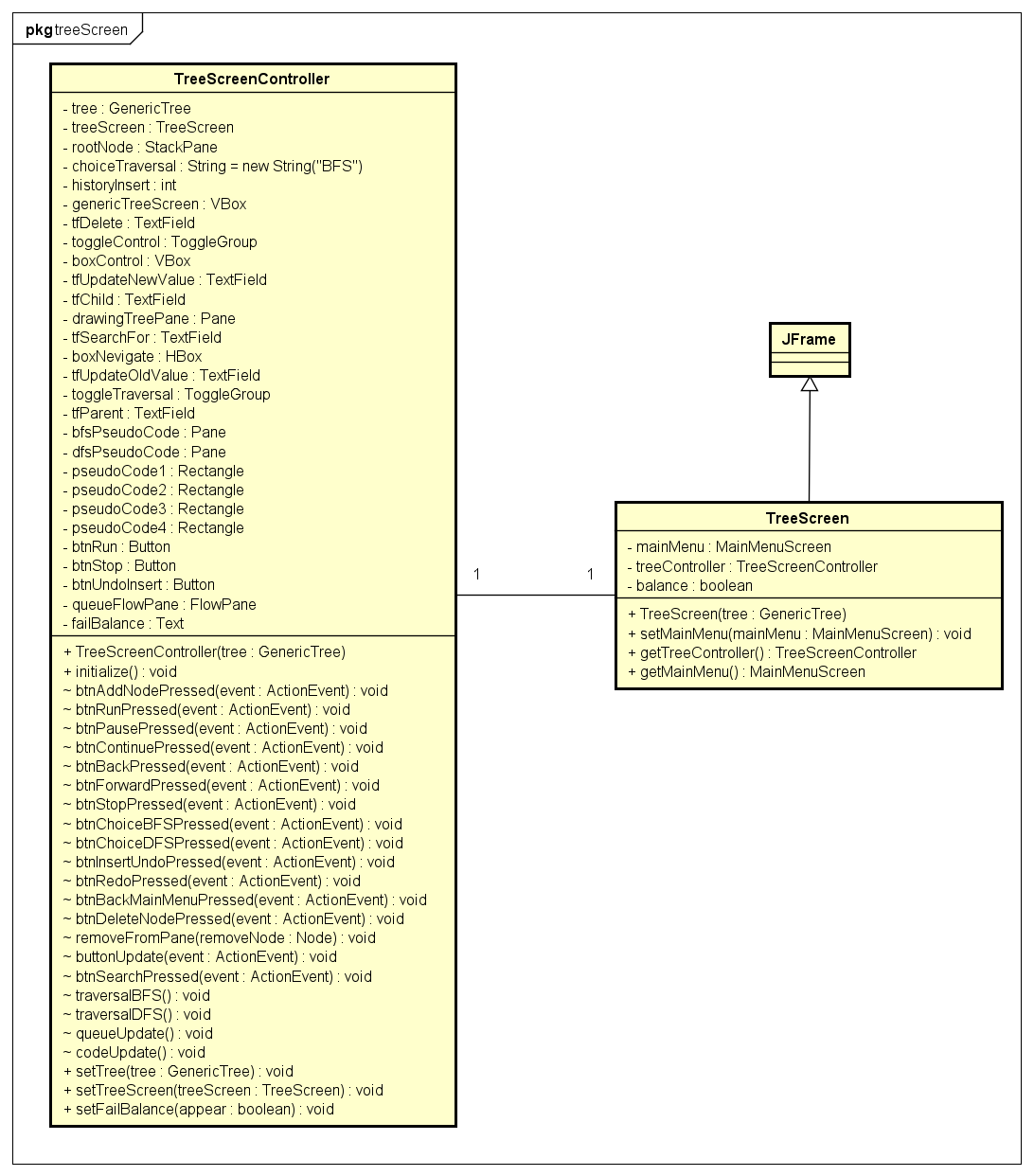
We have three packages, treeDataStructure contains different tree classes and node classes. Package mainMenu contains the main window classes and treeScreen contains the window for displaying tree and tree operation.



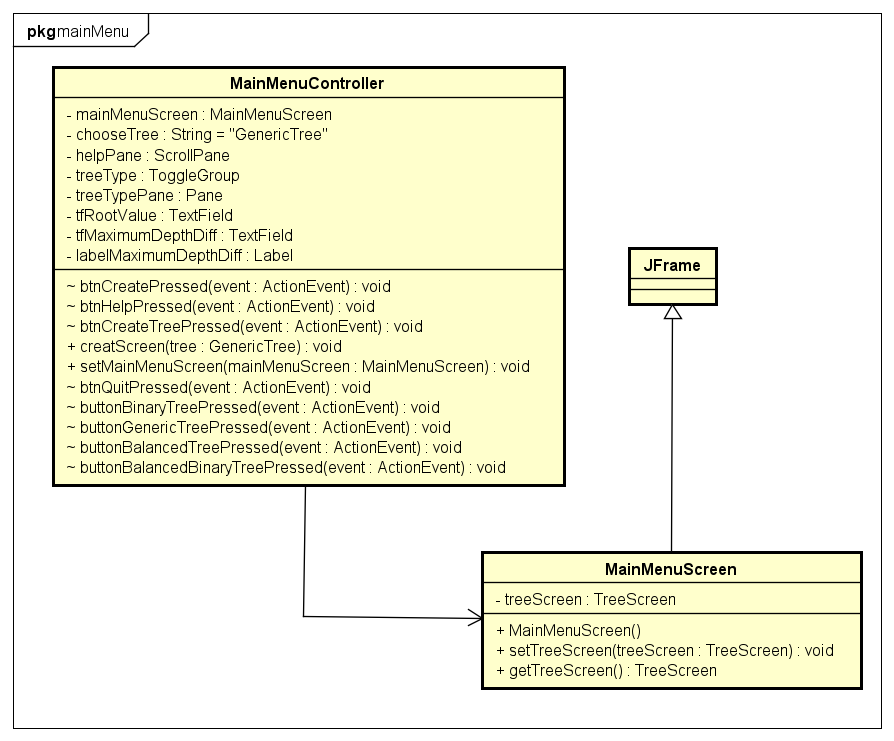
The detail class diagram of treeDataStructure package contain a lot of information and may by hard to read. We will describe some basic and important feature. The node class extends StackPane, with circle, line and value text as attributes, so that we can specify javafx components for our problem. Node also contains a list of child nodes, and parent nodes. The attribute state is the state of the node when traversal operation is executing (traveled, discovered, or undiscovered).

A generic tree contains a root node. We use queue, traveledNode, currentNode, timeline and state for executing traversal operations.

In child classes of GenericTree, we only need to override some different methods. For detail, the Binary tree needs a slightly added feature in insertNode method to keep the number of child nodes not greater than two. Balanced tree needs a new method to check if the tree is balanced or not, and check it every time a user inserts or deletes a node.



In the package treeScreen, the class TreeScreenController has many attributes and methods to interact with the user throw graphical interface. The TreeScreen is a swing window, and we integrate javafx code into this application.



Similar to the treeScreen package, in the mainMenu package, we integrate javafx code in MainMenuController into the Swing application MainMenuScreen. We apply association to change from the main menu and tree screen and inversely.

**3. Explanation**

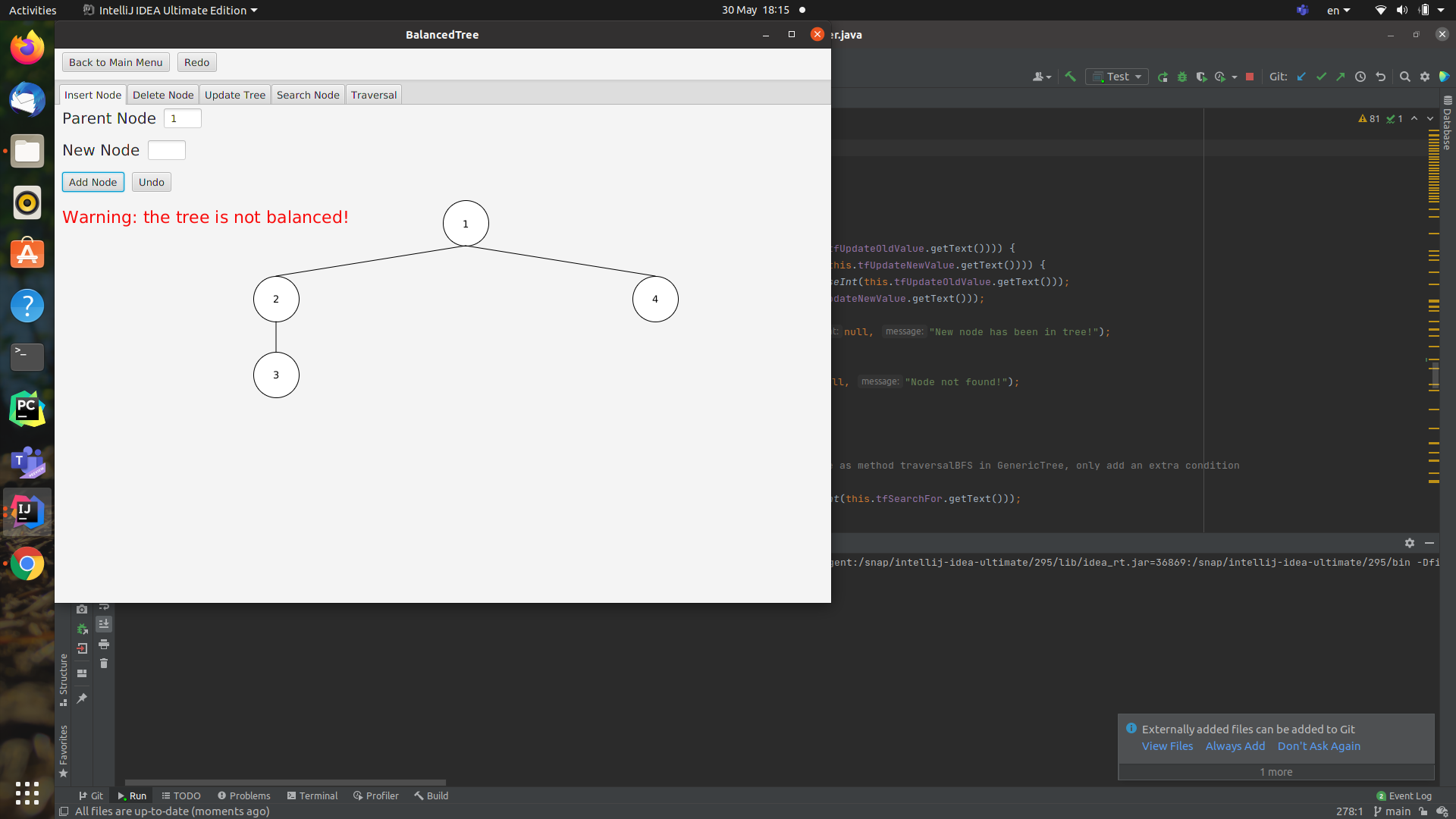
**OOP technique:**

* Inheritance: BinaryTree, BalancedTree inherit from GenericTree; BinaryBalancedTree inherits from BalancedTree.
* Polymorphism: insertNode method in GenericTree, BinaryTree, BalancedTree, BinaryBalancedTree; deleteNode method in GenericTree, BalancedTree.
* Aggregation: GenericTree contains Node (rootNode).
* Association: Each node has its child nodes and its parent node; association in GenericTree, TreeScreenController, TreeScreen, MainMenuScreen, MainMenuScreenController.

Parent class **GenericTree** has attribute Node **rootNode** which identifies the root of the tree. Class Node has all attributes LinkedList **childNodes** which contains its children, this creates a chain reaction which is typical of how tree is implemented.

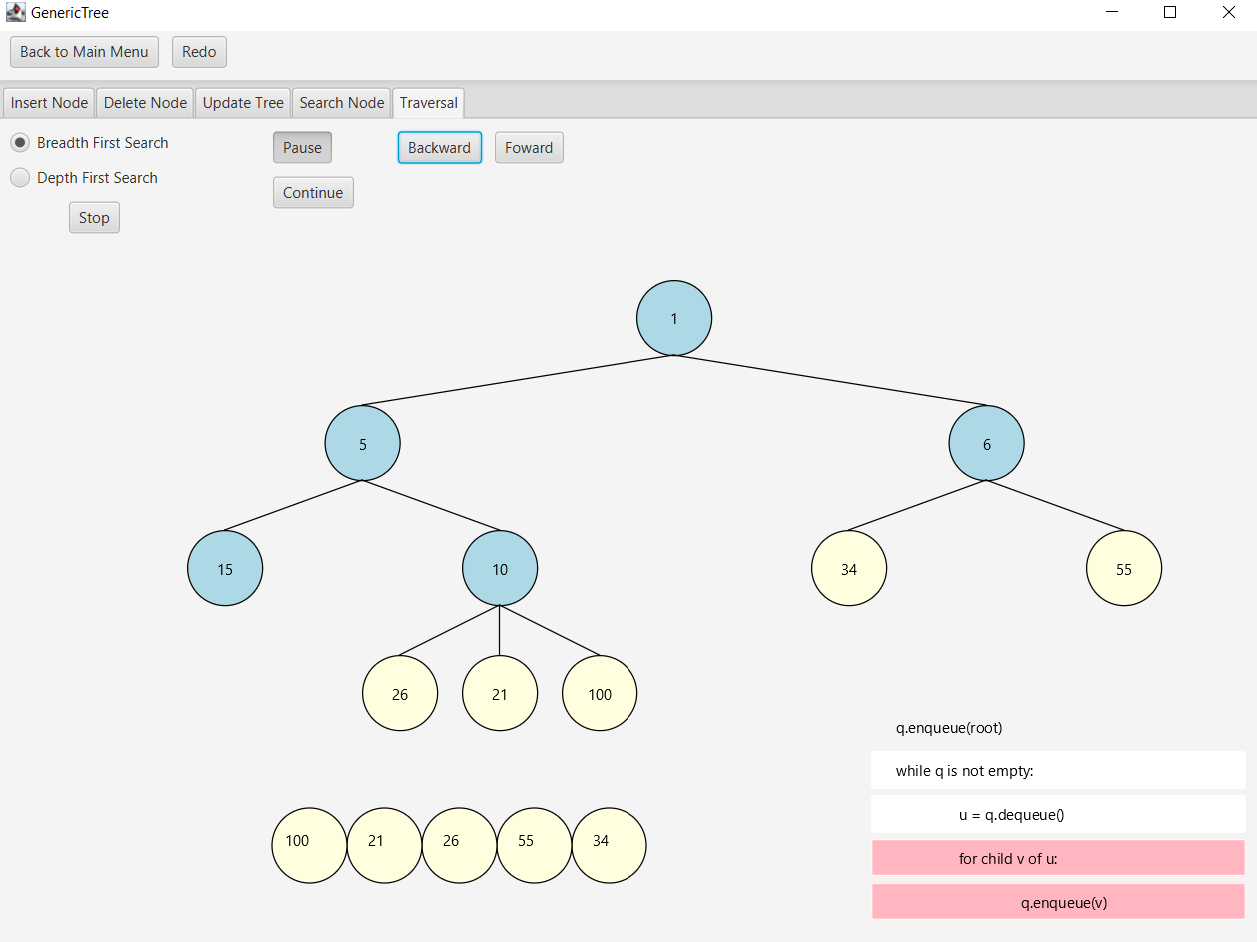
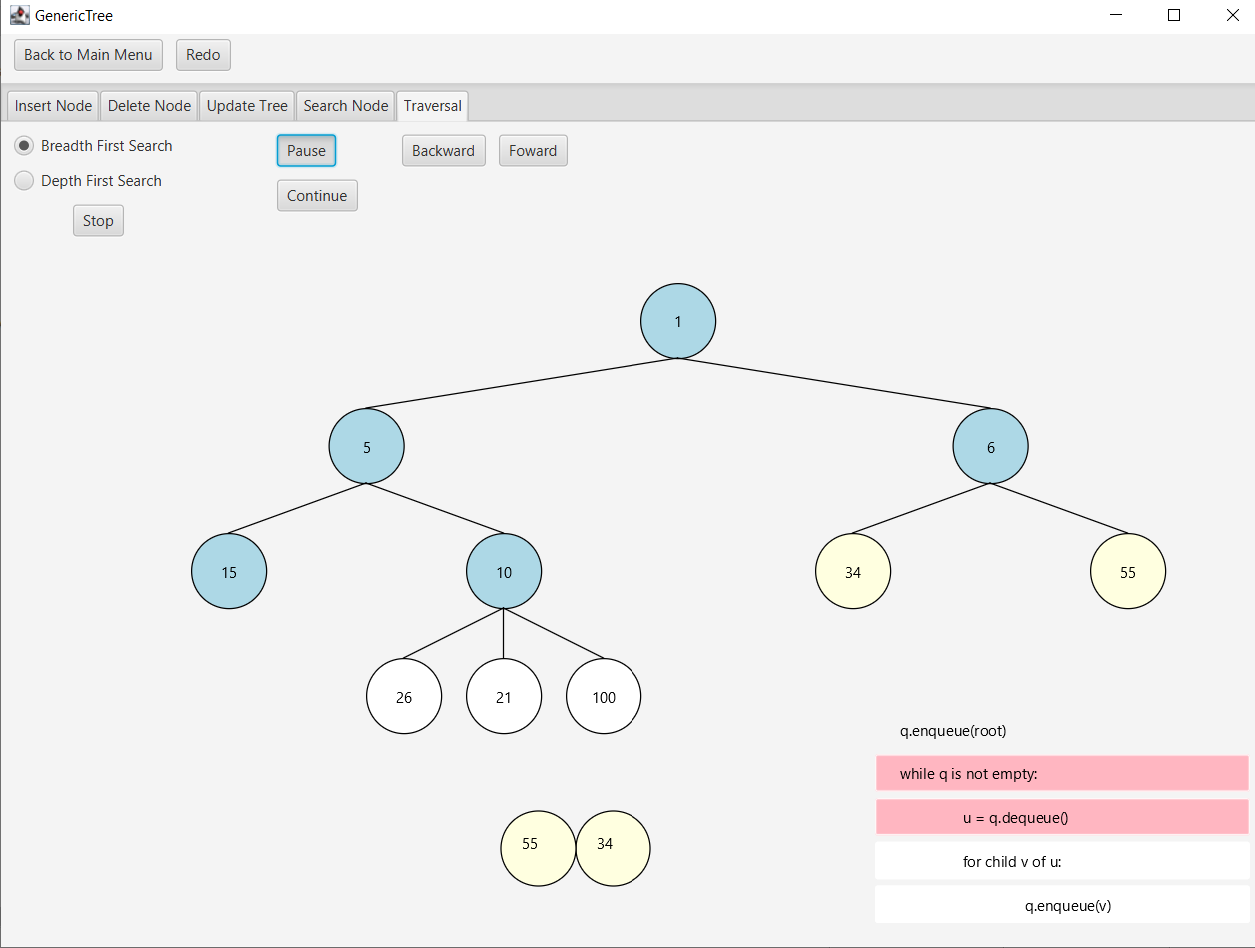
Child class **BinaryTree** inherit from **GenericTree** and override the method **insertNode(int parentValue, Node childNode)** to insert a child node to a parent node only when there are less than two children in LinkList **childNodes** of parent node.

Child class **BalancedTree** inherit from **GenericTree** and override the method **insertNode(int parentValue, Node childNode)** and **deleteNode(int nodeValue)** to add a method **checkBalance()** after each modification of the tree to check and return a warning if the balance property is violated corresponding to each value k.



This is an example of violating the balance property when k = 0

Another important feature is the traversal operation. To implement it with a back, forward button, and highlight pseudo code, we divide the hold process into small pieces, similar to the pseudo code (backBFS, forwardBFS method). Each step takes a new node from queue (or stack in DFS), represented by state 1, or enqueue child nodes of current node, represented by state 2.



*Figure. Two state in tree traversal*

**4.Member contribution.**

Hoang Thien Tam – 20194450: implement the delete button, update button, search button, tree inheritance and tree buttons in Main Menu.

Tran Quoc Khanh – 20190085: Design class diagram, implement Generic Tree and Node ,BFS and DFS operation, insert operation, main menu, report.

Hoang Van Khanh – 20194440: implement Binary, balanced and balanced binary tree, slide, report, testing.