TOPIC: VISUALIZATION OF OPERATIONS ON TREE DATA STRUCTURES

- TEAM 9
- Course: Object-oriented Programming 131678 – 2022.2

Dr. Nguyen Thi Thu Trang

16/07/2023

Member name	Tasks
Nguyễn Nam Hải	diagram (35%), GUI (100%), animated operations (100%), data structures (50%), node (40%), exception handler (70%), report (20%)
Nguyễn Song Hào	diagram (35%), data structures (50%), traverse (100%), backward (100%), forward operations (100%), node (40%), report (40%)
Hà Hoàng Hiệp	diagram (20%), node (20%), exception handler (30%), help button (100%), report (10%)
Trần Thị Hiền	diagram (10%), testing, decoration, slide, report (30%), balanced binary tree (50%)

1. INTRODUCTION

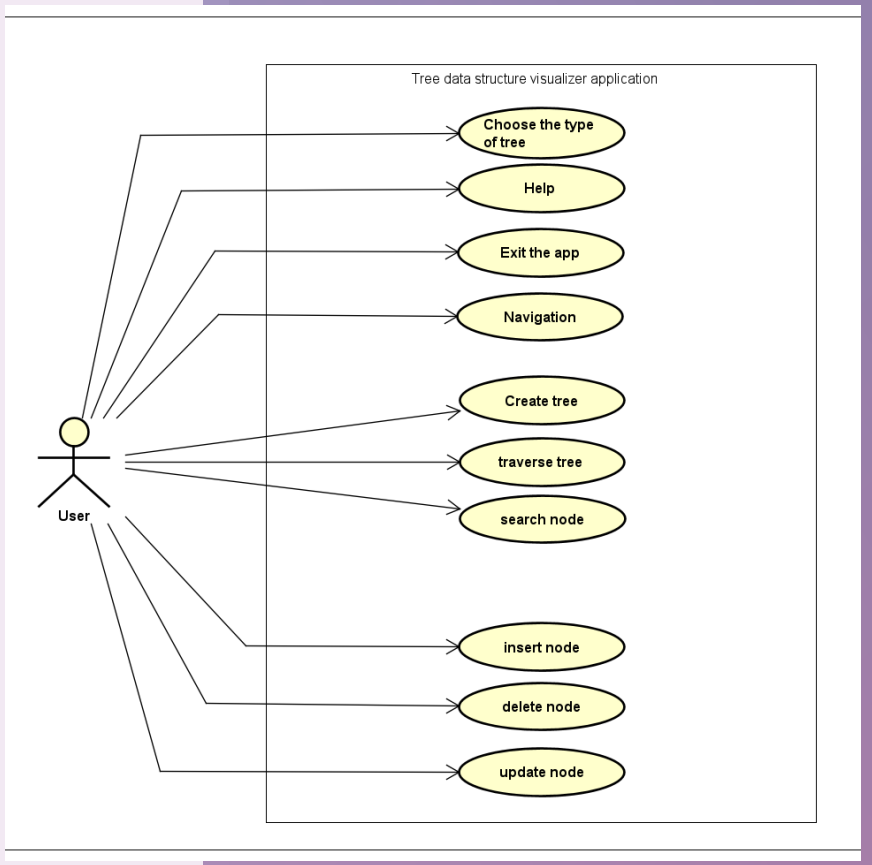
The project "Visualization of Operations on Tree Data Structures" aims to design a program that allows users to visualize and understand basic operations on different types of trees. The project's objectives include the development of a user-friendly interface, clearly presenting and explaining basic tree operations, and ensuring interactivity and usability.

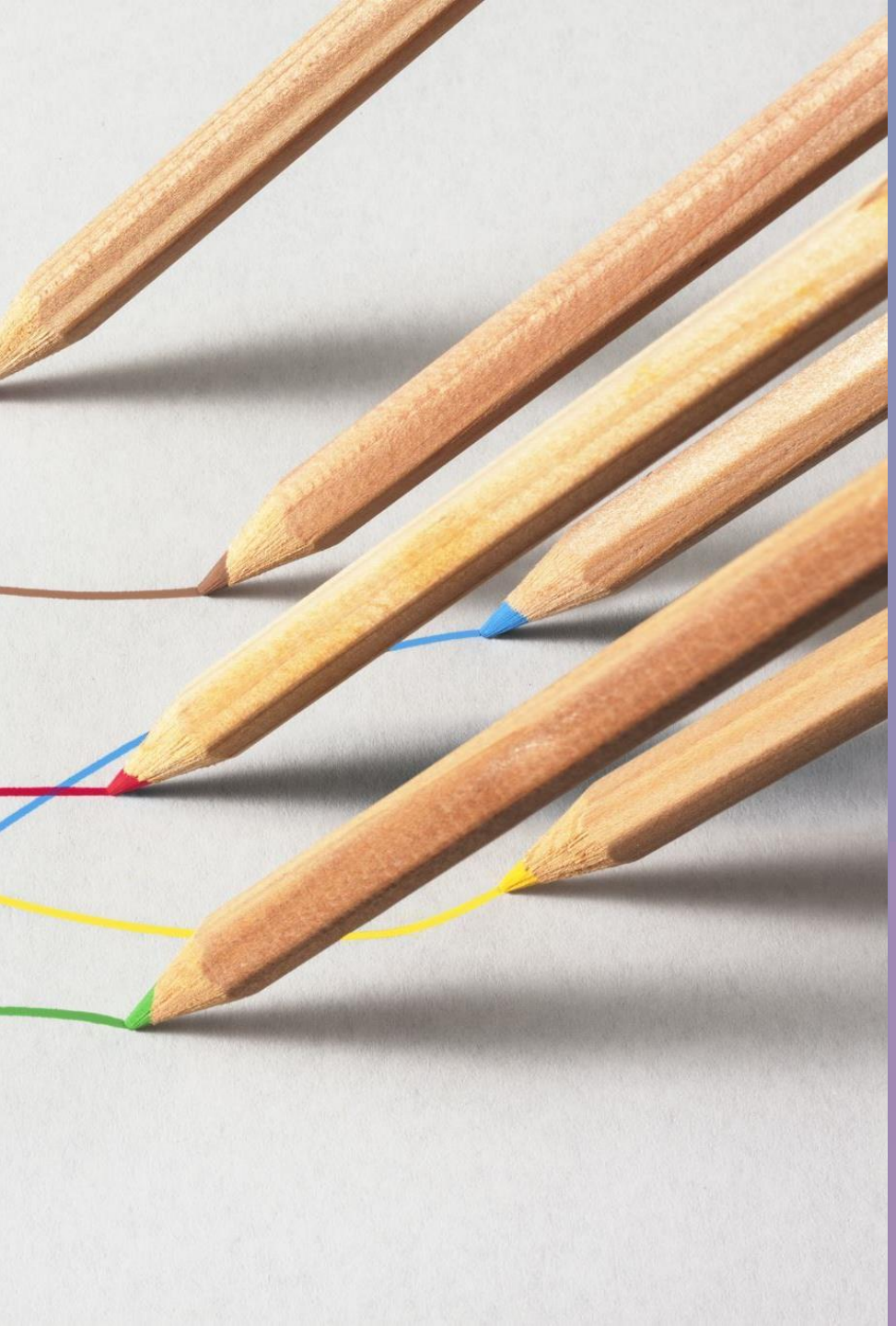
The project requires the development of a program to visualize and explain basic operations on four types of trees: a generic tree, a binary tree, a balanced tree, and a balanced binary tree

The application offers functions such as creating a new tree, inserting, deleting, updating nodes, searching for nodes, and traversing the tree. It provides an intuitive interface for users to interact with tree structures.

2. USE CASE DIAGRAM

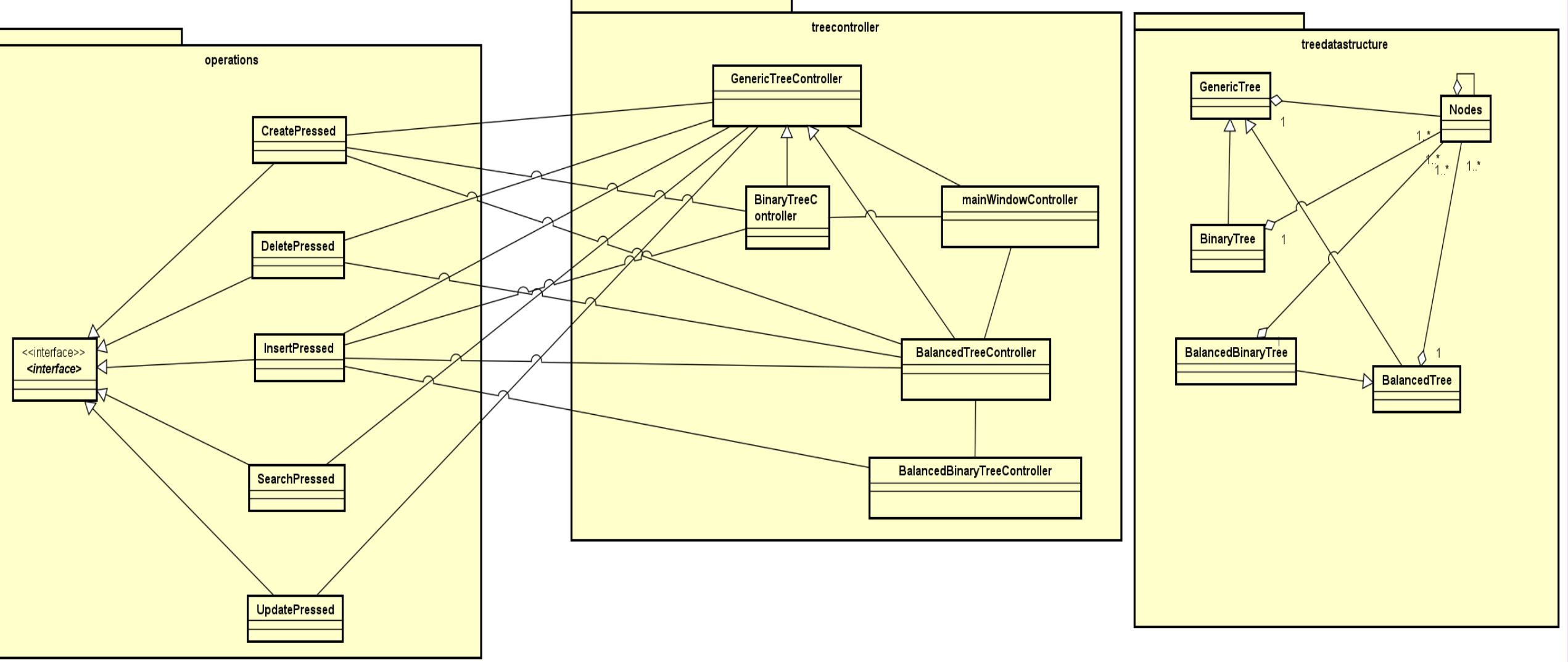
- The software provides a user-friendly interface for choosing tree types and visualizing them
- Users can perform essential tree operations such as creating, traversing, searching, inserting, deleting nodes, and updating values
- A "Help" button offers insights into the tree type being visualized, and an "Exit" button allows for easy program termination





3. PROJECT DESIGN

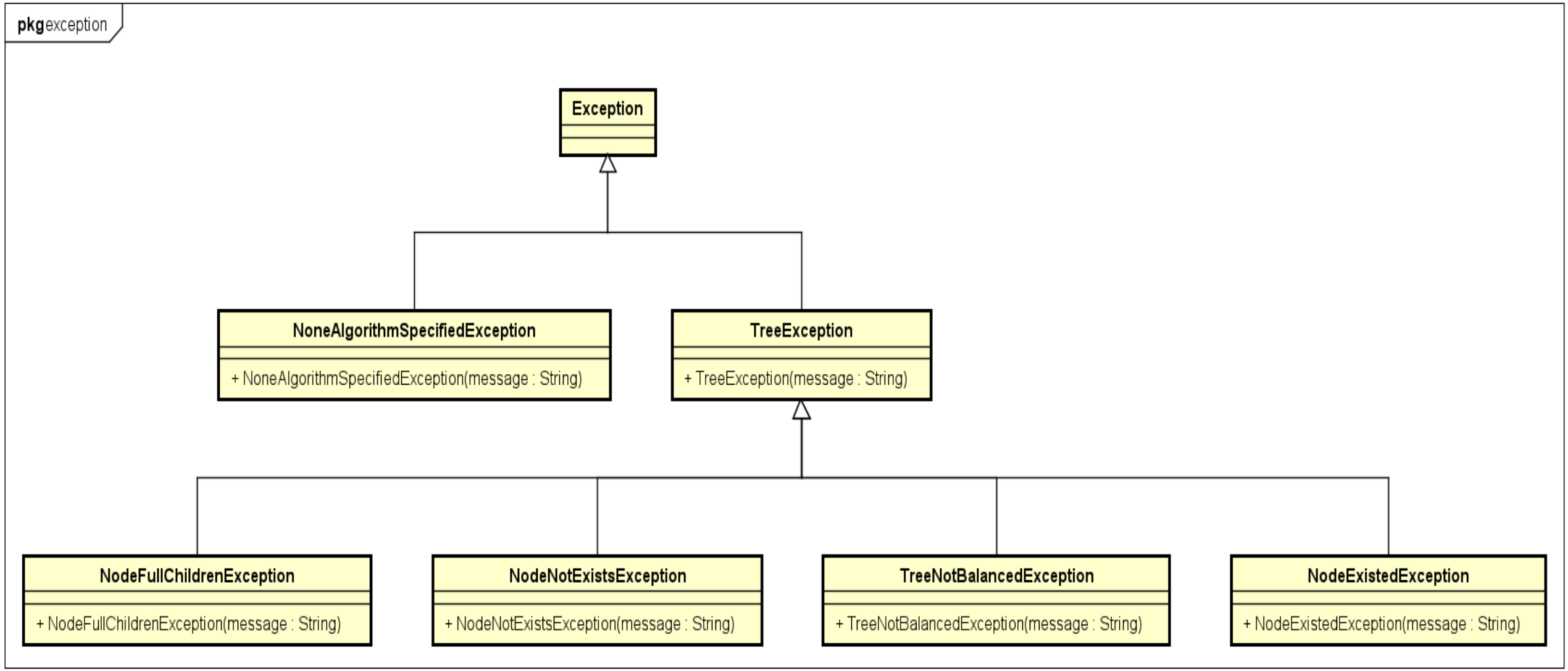
3.1. GENERAL CLASS DIAGRAM



3. The tree controller package contains controller files that manage the user interface for specific tree data structures. It handles events from the GUI interface and calls appropriate methods in the tree data structure package to perform operations on the tree. This package also defines user interface components and provides methods to display and update the tree on the user interface.

3.2. DETAILED CLASS DIAGRAM

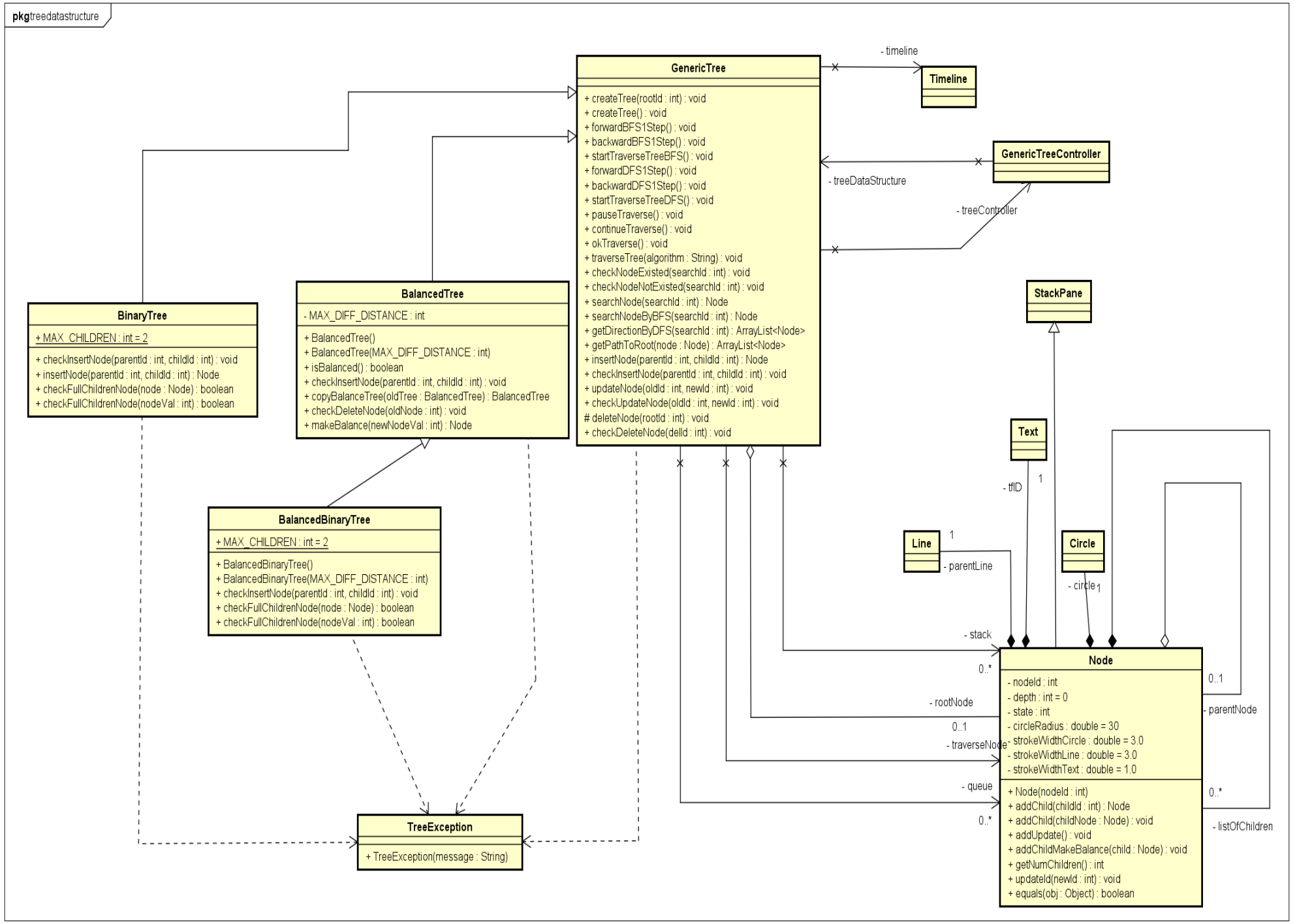
3.2.1 EXCEPTION CLAS DIAGRAM



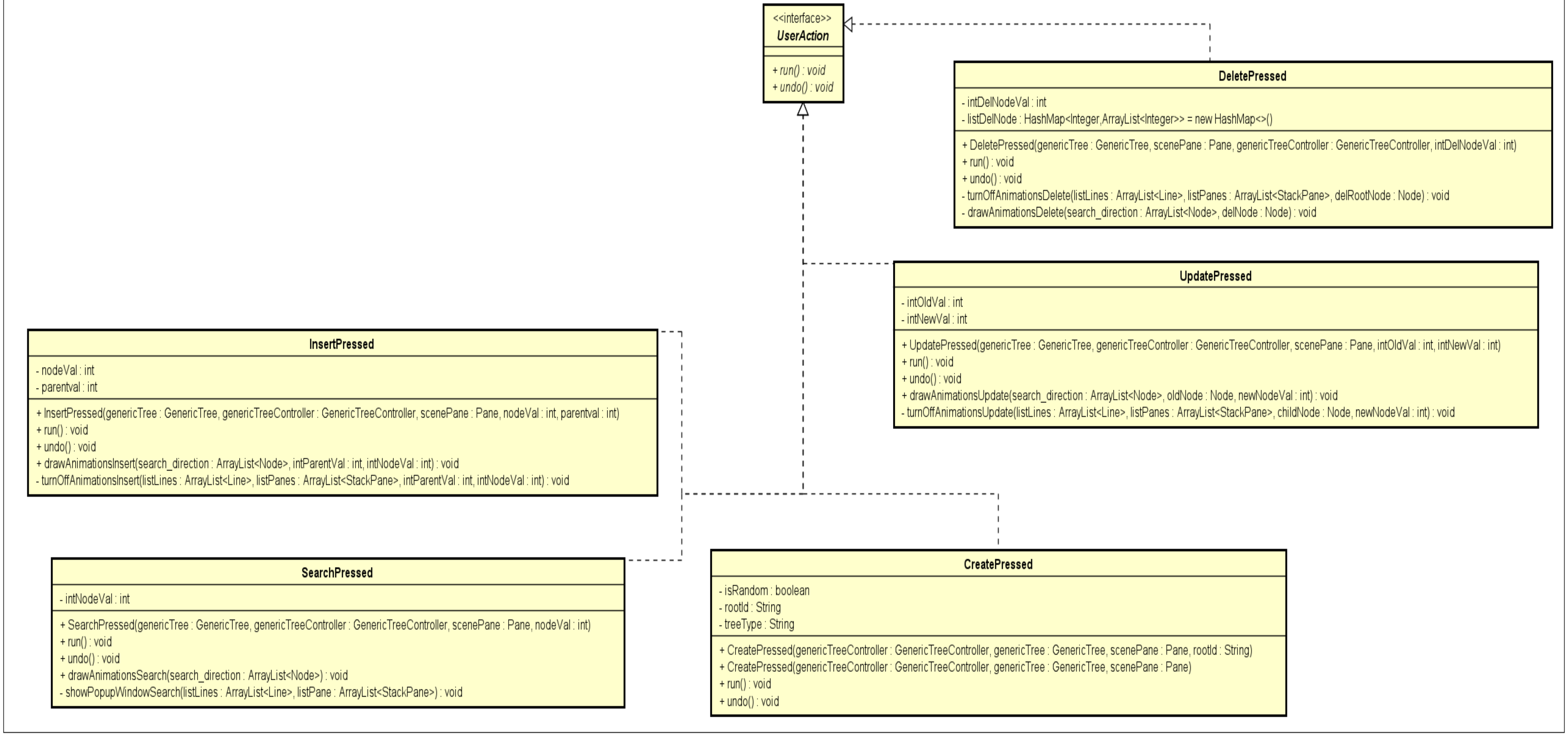
TreeException is a parent exception class for four specific tree-related exceptions: TreeNotBalancedException, NodeFullChildrenException, NodeNotExistsException, and NodeExistedException. These exceptions handle issues related to unbalanced tree structure, exceeding maximum children limit, accessing non-existent nodes, and inserting duplicate nodes, respectively.

3.2.2. TREE DATA STRUCTURE CLASS DIAGRAM

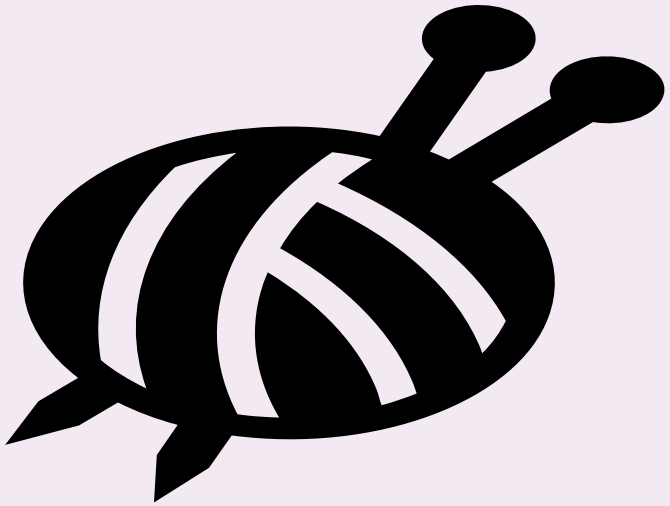
- The data structure package consists of four classes: Generic tree, Binary tree, Balanced tree, and Balanced Binary tree
- The Generic tree class handles various tree operations and exceptions, allowing nodes to have an arbitrary number of children.
- The Binary tree class, inheriting from the Generic tree class, enforces the binary tree property, raising NodeFullChildrenException if a node already has the maximum number of children.
- The Balanced tree class, a subclass of the Generic tree, maintains a balanced tree structure with limited depth differences among leaf nodes. It includes algorithms to rebalance the tree after invalid operations.
- The Balanced Binary tree class extends the Balanced tree class and ensures the binary property is maintained during insertions.



3.2.3. OPERATION CLASS DIAGRAM


<<interface>>
UserAction
+ run() : void
+ undo() : void
DeletePressed
- intDelNodeVal : int
- listDelNode : HashMap<Integer,ArrayList<Integer>> = new HashMap<>()
+ DeletePressed(genericTree : GenericTree, scenePane : Pane, genericTreeController : GenericTreeController, intDelNodeVal : int)
+ run() : void
+ undo() : void
- turnOffAnimationsDelete(listLines : ArrayList<Line>, listPanes : ArrayList<StackPane>, delRootNode : Node) : void
- drawAnimationsDelete(search_direction : ArrayList<Node>, delNode : Node) : void
UpdatePressed
- intOldVal : int
- intNewVal : int
+ UpdatePressed(genericTree : GenericTree, genericTreeController : GenericTreeController, scenePane : Pane, intOldVal : int, intNewVal : int)
+ run() : void
+ undo() : void
+ drawAnimationsUpdate(search_direction : ArrayList<Node>, oldNode : Node, newNodeVal : int) : void
- turnOffAnimationsUpdate(listLines : ArrayList<Line>, listPanes : ArrayList<StackPane>, childNode : Node, newNodeVal : int) : void
InsertPressed
- nodeVal : int
- parentval : int
+ InsertPressed(genericTree : GenericTree, genericTreeController : GenericTreeController, scenePane : Pane, nodeVal : int, parentval : int)
+ run() : void
+ undo() : void
+ drawAnimationsInsert(search_direction : ArrayList<Node>, intParentVal : int, intNodeVal : int) : void
- turnOffAnimationsInsert(listLines : ArrayList<Line>, listPanes : ArrayList<StackPane>, intParentVal : int, intNodeVal : int) : void
SearchPressed
- intNodeVal : int
+ SearchPressed(genericTree : GenericTree, genericTreeController : GenericTreeController, scenePane : Pane, nodeVal : int)
+ run() : void
+ undo() : void
+ drawAnimationsSearch(search_direction : ArrayList<Node>) : void
- showPopupWindowSearch(listLines : ArrayList<Line>, listPane : ArrayList<StackPane>) : void
CreatePressed
- isRandom : boolean
- rootId : String
- treeType : String
+ CreatePressed(genericTreeController : GenericTreeController, genericTree : GenericTree, scenePane : Pane, rootId : String)
+ CreatePressed(genericTreeController : GenericTreeController, genericTree : GenericTree, scenePane : Pane)
+ run() : void
+ undo() : void

3.2.4. CONTROLLER CLASS DIAGRAM



The controller package contains controller classes and operations performed on trees. There are a total of 6 controller classes: GenericTreeController, BalancedTreeController, BinaryTreeController, BalancedBinaryTreeController, MainWindowController, and HelpController.

MainWindowController controls 4 buttons for creating trees and 1 button to exit the application. HelpController controls the "Help" pop-up window.

The Create, Traverse, Update, and Search operations have the same execution method for all 4 types of trees. Traverse operation has functionalities like stop, continue, backward, and forward.

BalancedTreeController overrides methods for insert and delete operations to maintain tree balance.

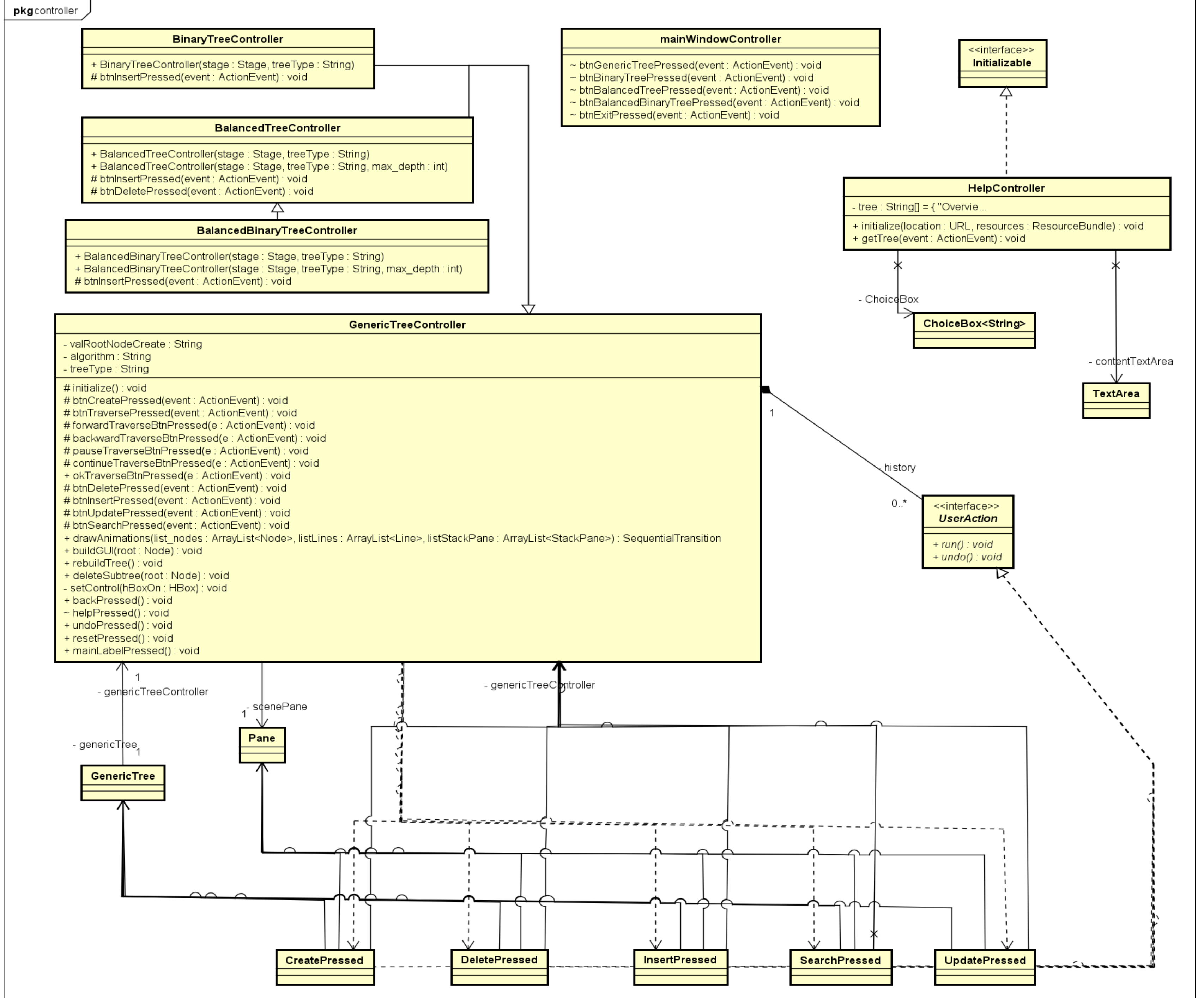
BinaryTreeController only overrides the method for insert operation since binary trees have a maximum of two children.

BalancedTreeController and BinaryTreeController inherits from GenericTreeController, and perform 4 common basic operations.

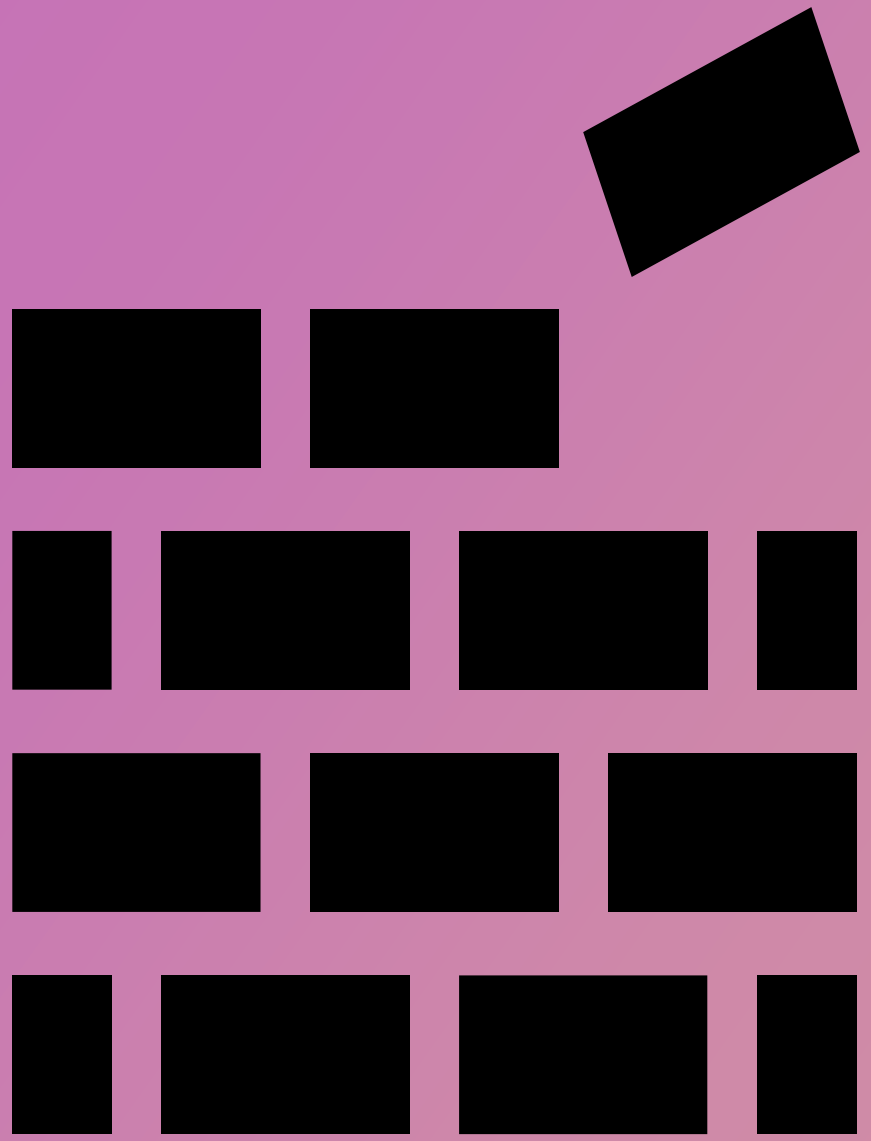
GenericTreeController is the main controller class with main attributes such as treeDataStructure, scenePane, and history. It controls 6 basic operations and 4 other operations on the tree.

BinaryBalancedTreeController inherits from BalancedTreeController and overrides the insert method.

BalancedBinaryTreeController inherits from BalancedTreeController and has similar functionalities.



3.3. Explanation of OOP techniques in our design



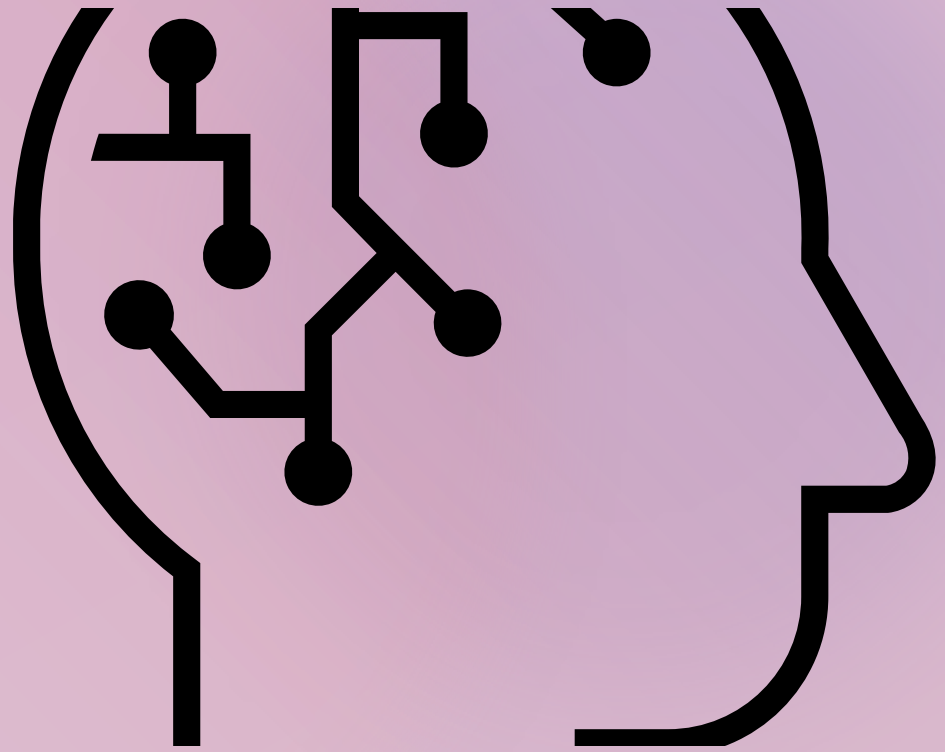
1. Inheritance

2. Encapsulation

3. Polymorphism

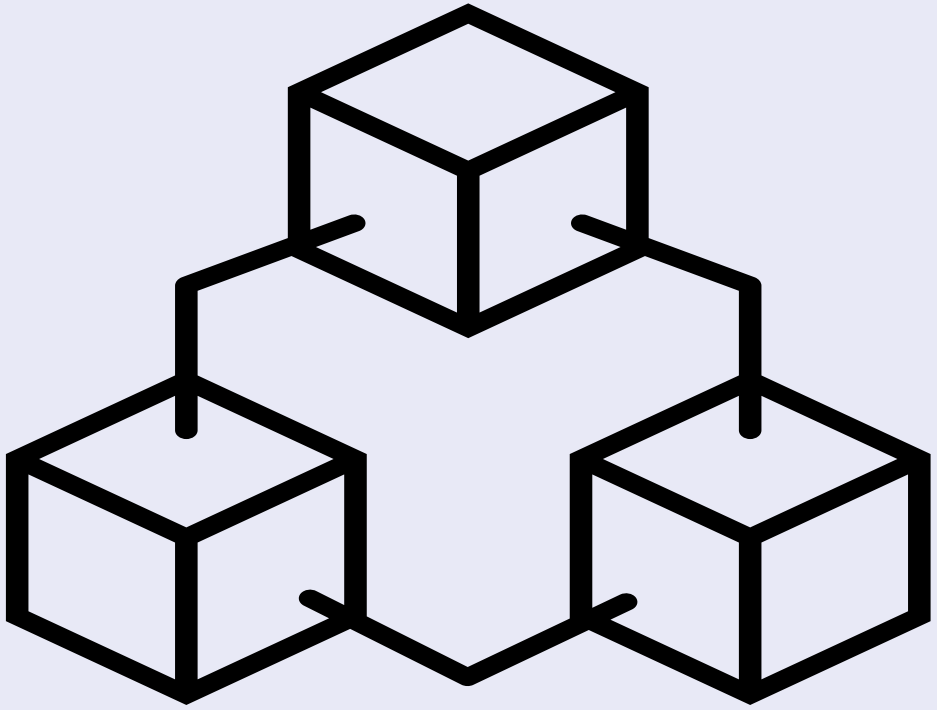
4.Association/Aggregation/Composition

5. Dependency



Inheritance

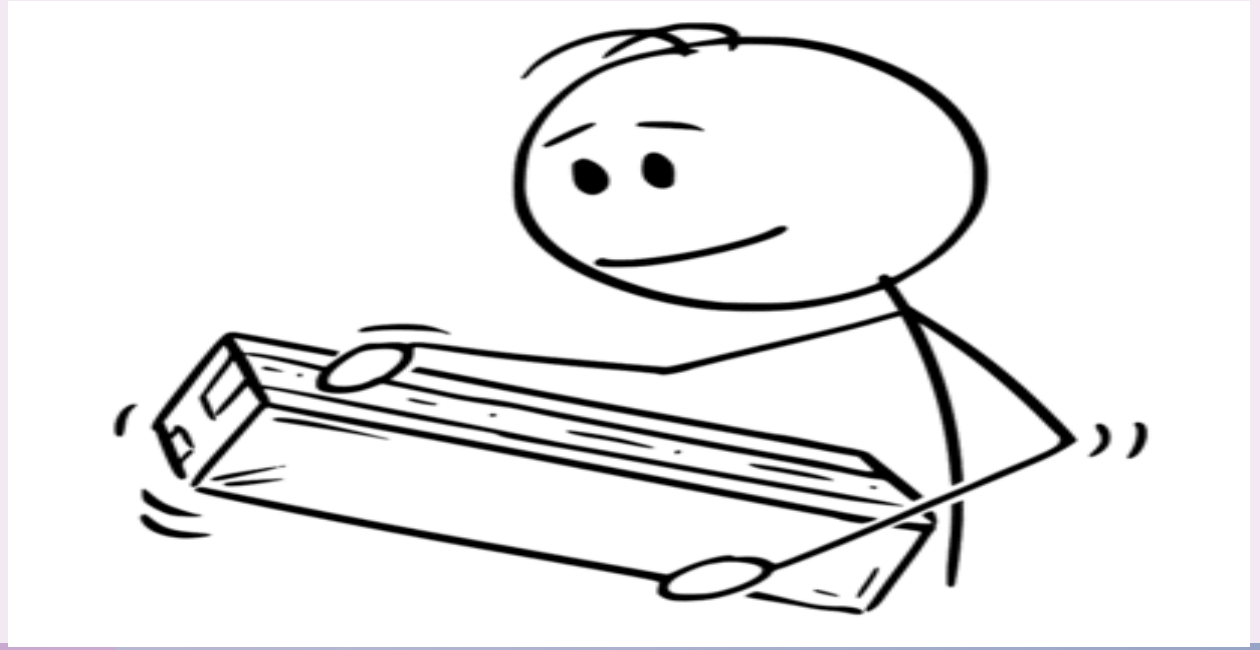
Inheritance is one of the essential techniques widely used in our project. We applied inheritance when implementing data structure classes and controller classes.



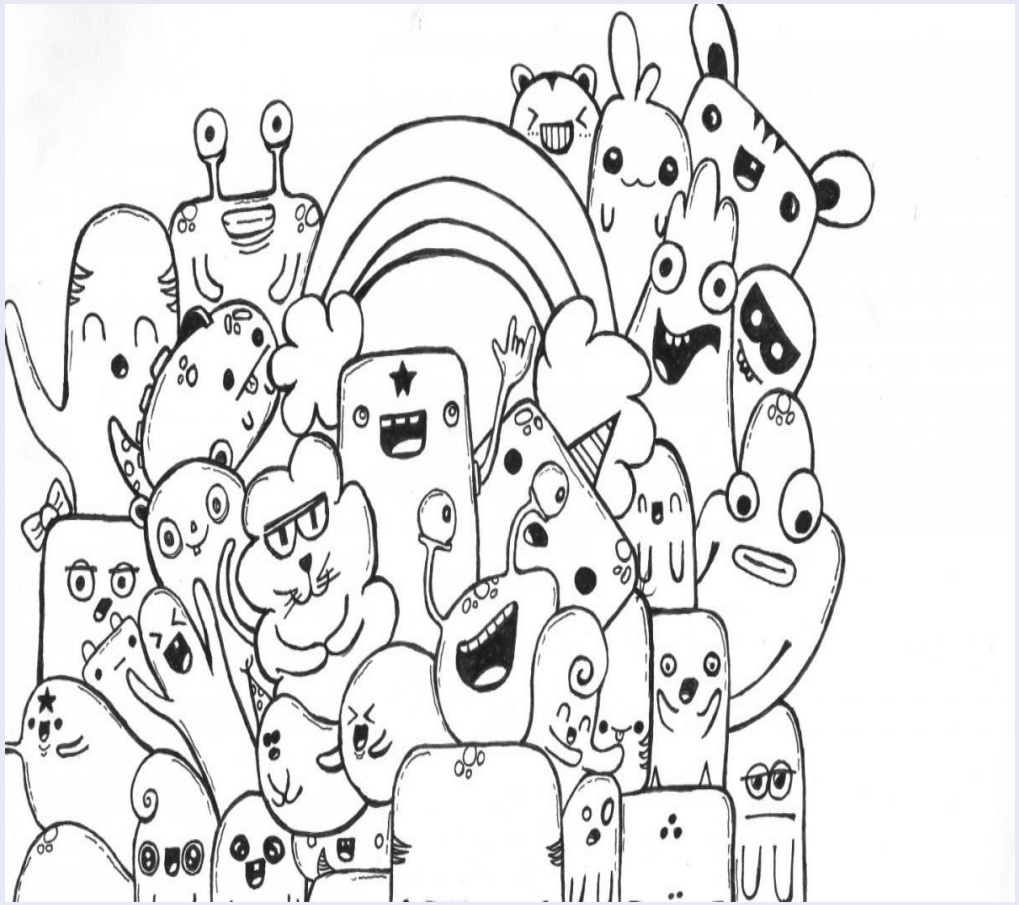
- Implementing data structure class and controller class.
- BinaryTree and BalancedTree inherit from the GenericTree, which is the baseline tree, and the BalanceBinaryTree inherits from the BalancedTree.
- The controllers also inherit from the GenericTreeController like the tree data structure class.

Encapsulation

- We've used the Encapsulation technique in almost all classes to hide the internal state and implementation details from external entities.
- We set 'private' for some attributes such as NodeId, depth, …
- If other classes want to access the attributes, they can use getter and setter methods.



Polymorphism



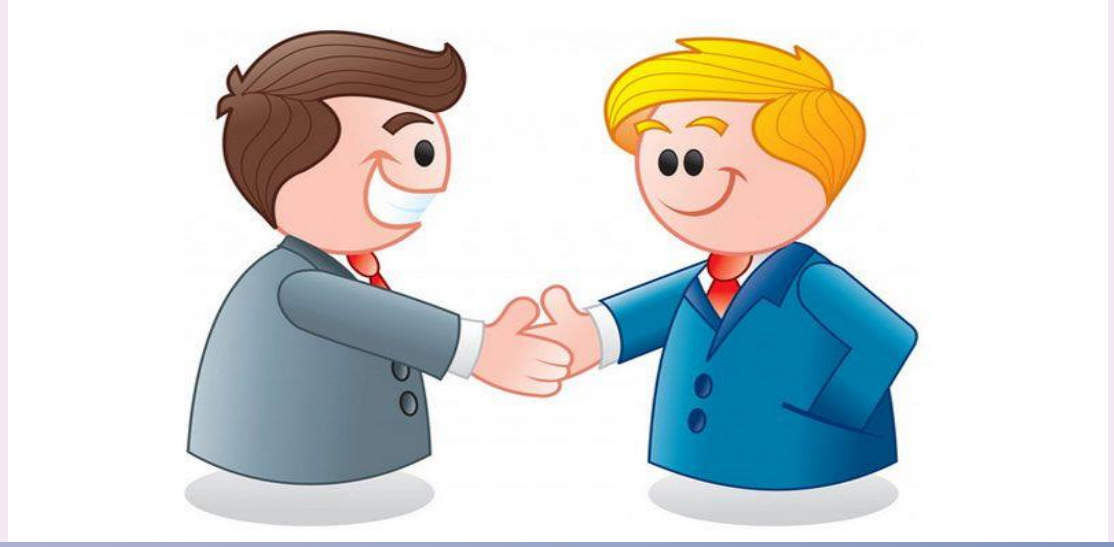
- .Implementing each operation.
- .In each operation implementation, there are cache of data structure objects and controller objects.
- .Implement each operation that is fit with the baseline tree, which is GenericTree.
- .Then we will downcast other trees to its base line tree when doing any operation.

Association

The association relationship is presented when a GenericTreeController controls and manipulates a GenericTree.

Aggregation

- The aggregation relationship is used when the Node has its parent which is also a Node and when the Node is deleted, its parent still exists.
- In addition, this relationship is presented where the GenericTree has a root Node.



COMPOSITION

- The composition is used when the Node contains three objects Circle, TextField, Line which are used to display the node on screen.
- Besides, the composition relationship is used when the Node has many children (which are stored in an attribute arraylist listOfChildren) and its children are gone when the Node does not exist.
- In addition, the GenericTreeController has an array named history which contains many UserAction-implemented objects and when the GenericTreeController is destroyed, the history is gone too.

DEPENDENCY

- The controller classes depend on the tree data structure classes to control the trees based on user input and events.
- The operation classes (e.g, CreatePressed, InsertPressed, DeletePressed) depend on the GenericTree class to perform the corresponding tree operations.
- In the package-level, the main application class depends on the 'controller' package. It relies on the functionality provided by the controller classes to manage the application's behavior.
- The operation classes interact with the 'UserAction' interface, contributing to a well-structured and maintainable codebase.
- In the operations of four types of tree, we also use the Exceptions, so the Trees are dependent on the Exceptions.

DEMO

THANKS