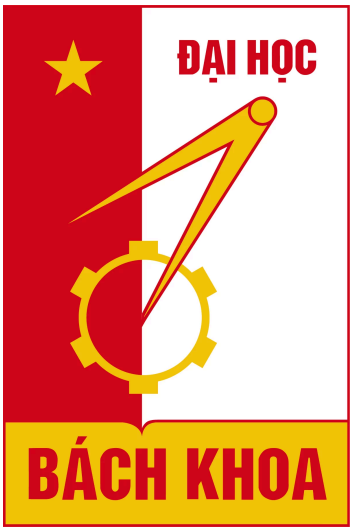
HANOI UNIVERSITY OF SCIENCE AND TECHNOLOGY
SCHOOL OF INFORMATION AND COMMUNICATION TECHNOLOGY



PROJECT REPORT

Môn: Object-Oriented Programming
Mã học phần: IT3100E
Mã lớp: 147839

TREE VISUALIZER

Nhóm 32:
Tô Thái Dương - 20205180
Bùi Hoàng Việt - 20226073

1. Introduction

In Data Structures and Algorithms (DSA), a tree is a hierarchical data structure consisting of nodes connected by edges. A tree starts with a single root node, and each node can have multiple child nodes but only one parent, creating a parent-child relationship without cycles. Trees are used to model hierarchical relationships such as file systems, organizational charts, and are crucial in various search and sorting algorithms. Each node can store data and properties like height or depth. Common types of trees include binary trees, binary search trees (BST), AVL trees, and B+ trees, each optimized for specific operations and applications.

Our goal for the project is to create a **Java Tree Visualizer Application** that will give you an overview on several types of tree and some operations we can do on them while applying basic Object-Oriented Programming concepts.

2. Workload Distribution

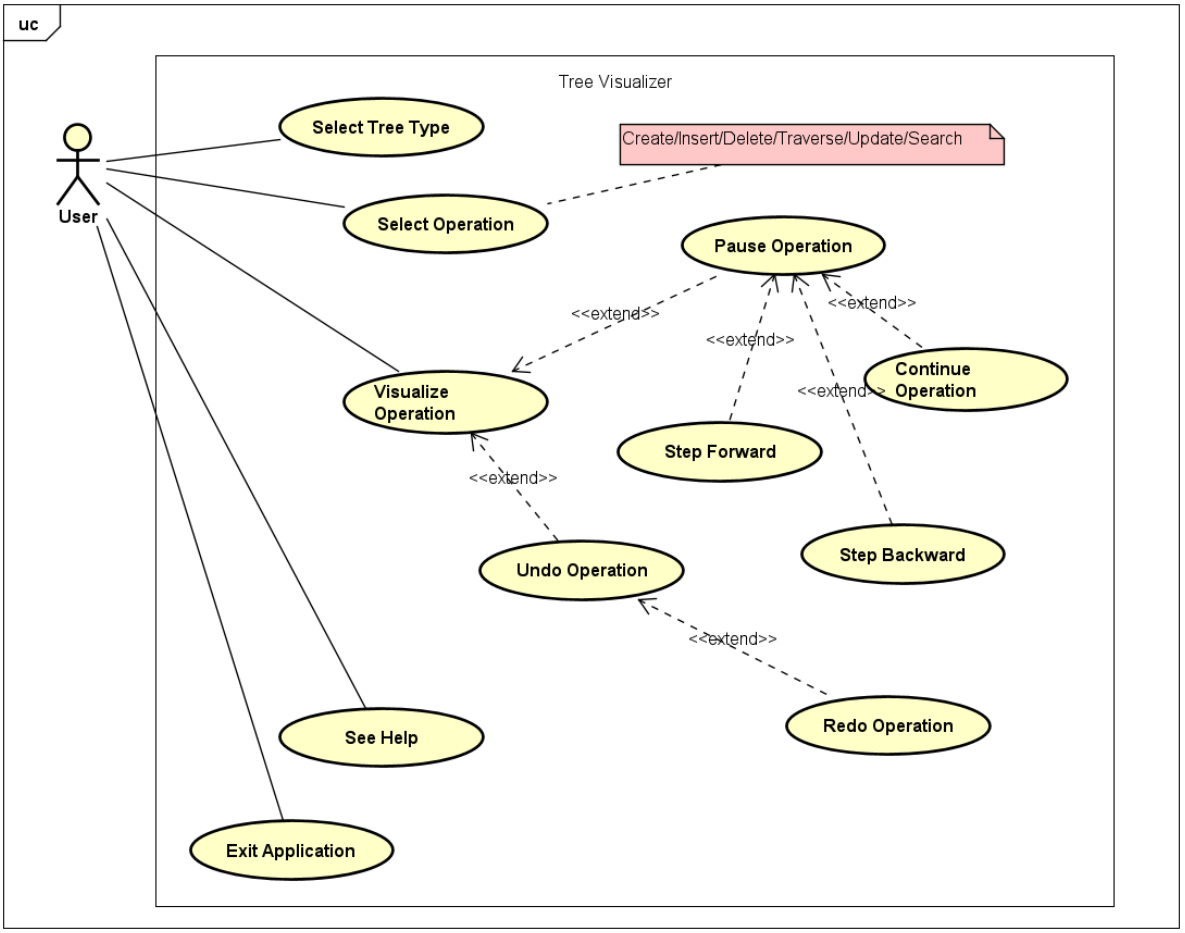
Tô Thái Dương - 20205180:

- Research on BalancedTree and BalancedBinaryTree logic.
- Implements Tree, BalancedTree, BalancedBinaryTree, TreeNodeAdapter and TreeVisualizerApp classes.
- Report and slide.

Bùi Hoàng Việt - 20226073:

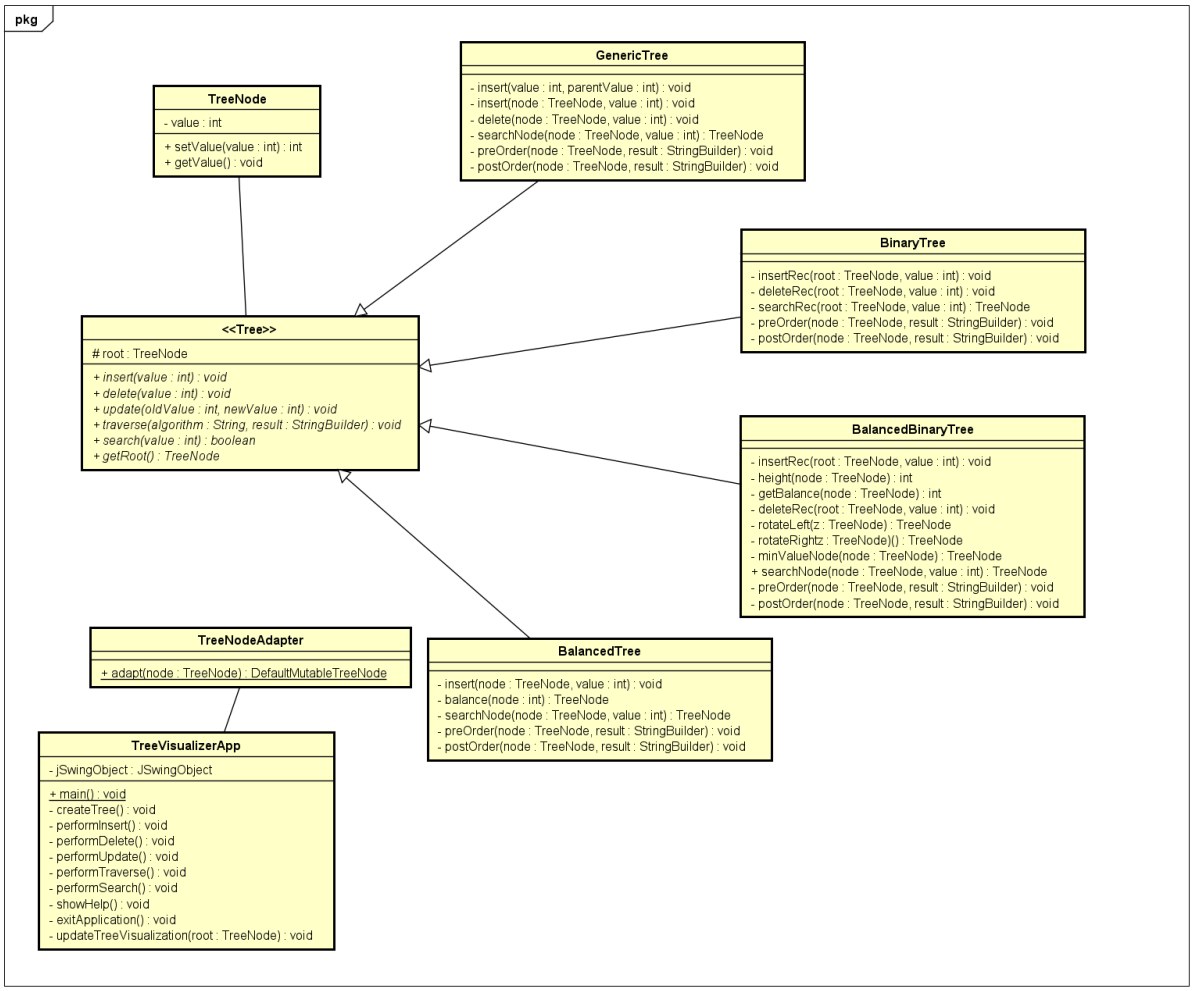
- Research on GenericTree and BinaryTree logic.
- Implements TreeNode, GenericTree and BinaryTree classes.
- Report and slide.

3. Usecase Diagram



Our main usecases include selecting tree types, operations, running them, see help and exit.

4. Class Diagram

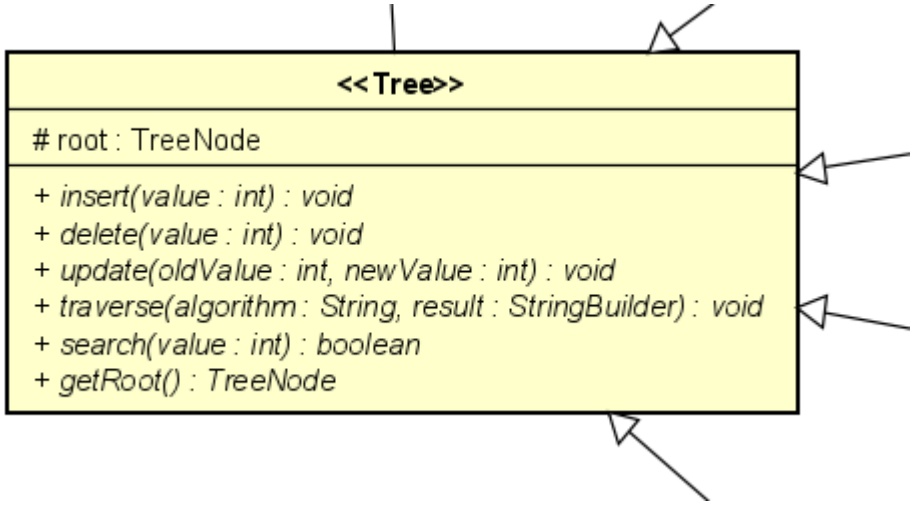


Our class diagram is rather simple. We show all the classes in our project and their association.

5. OOP Concepts Applied

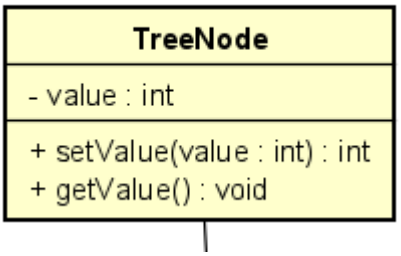
- ## Abstraction

  Every type of tree is based on the original generic tree. So we implemented an abstract class Tree which includes the root and abstract methods for every operation we will be doing on the trees.



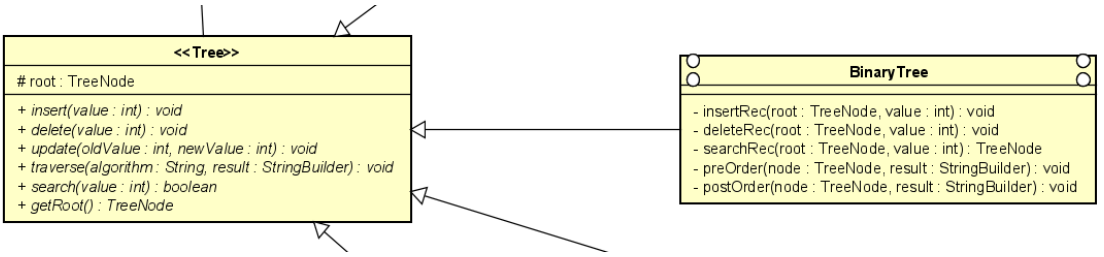
- **Encapsulation**

  The Node's value should not be accessed freely by any class, so the access modifier will be private. To access this attribute, we will have to use setter and getter methods that are implemented in the class.



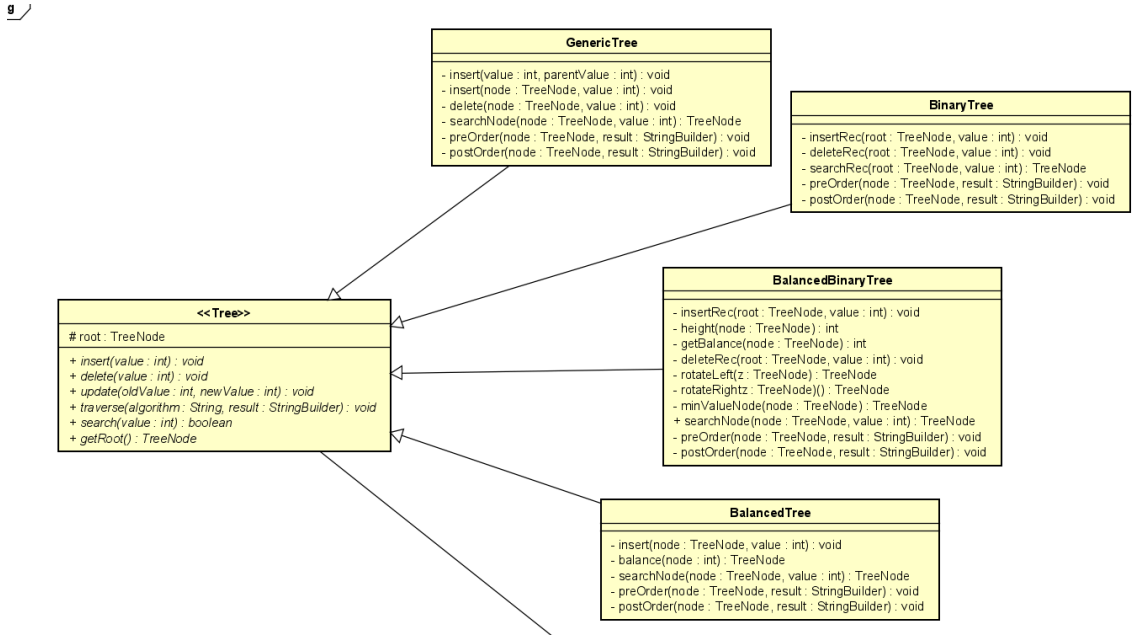
- **Inheritance**

  In implementation of the trees, we will have to inherit the attributes and methods of the abstract class Tree we implemented earlier. Each type of tree will have its own implementation of the abstract methods for the operations.



- **Polymorphism**

  There will be 4 classes for each type of tree that inherit the abstract class, thus ensuring polymorphism.



- **Composition**

  In instantiation of a Tree class, we also create the root node inside, which is instantiation of the Tree Node class. The root only exists when the Tree exists. When the Tree is deleted, the root is also gone. This is the definition of compostion.