Report on the Mini Project

Topic: Visualization of operations on tree data structures

Team 9

Nguyễn Nam Hải – 20214894
Nguyễn Song Hào – 20214895
Hà Hoàng Hiệp – 20214921
Trần Thị Hiền – 20214896

Course: Object-oriented Programming 131678 – Semester 2022.2
Dr. Nguyen Thi Thu Trang
16/07/2023

ACKNOWLEDGEMENT

To complete this project, we have received invaluable help from everyone involved.
First and foremost, we would like to express our heartfelt gratitude to Dr. Nguyen Thi Thu Trang, our main lecturer for the Object-Oriented Programming course. She not only granted us the opportunity to embark on this project but also displayed unwavering support and meticulous guidance, leading us through its completion with utmost dedication and care.. We have gained a wealth of knowledge, particularly in OOP techniques and the Java language. These acquired knowledge and skills not only hold value in the current project but also promise to greatly benefit us in our future endeavors.

Secondly, we would like to extend our sincere appreciation to our classmates who dedicated their time and effort to assist us. Their knowledge sharing and support have helped us tackle challenging questions and enhance our report. This created a positive learning environment and motivated us to overcome difficulties.

Thanks to the contributions and support of everyone involved, the project has successfully overcome challenges and achieved favorable outcomes. We take great pride in what we have accomplished and believe that this project will continue to deliver value and provide an excellent user experience.

Once again, we would like to express our heartfelt thanks and gratitude to all those who have contributed to the success of this project.

1. INTRODUCTION

1.1 Provide an overview of the project

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. Trees are a fundamental data structure in computer science, and this project focuses on providing a user-friendly interface to display and explain various tree operations.

1.2 State the objectives of the project

- Develop an intuitive and visually appealing user interface: The project aims to create a user interface that is aesthetically pleasing and easy to navigate. It should allow users to interact with different types of trees and perform operations in a seamless and intuitive manner.
- Display and explain basic tree operations: The project aims to provide clear visual representations and detailed explanations for operations such as creating a tree, inserting nodes, deleting nodes, updating nodes, and traversing the tree. This helps users understand the process and purpose of each operation.
- Ensure interactivity and usability: The project aims to provide an interactive experience for users, allowing them to pause, resume, or step through the execution of operations. It should also offer the ability to undo or redo ope
rations and provide a "Back" button for users to return to the main menu at any time.

By achieving these objectives, the project aims to enhance users' understanding of tree data structures and their operations through interactive visualization and informative explanations.

1.3 List the team members and their assigned tasks

Below are the individual contributions of each member in the project. In principle, the main contributor of each section is mentioned; however, this project is significantly done on a team basis i.e. ideas are shared and debated on as a whole, rather than from individual decisions.

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%)

2. REQUIREMENTS ANALYSIS

2.1 Describe the detailed requirements of the project

In this project, the following specific requirements are identified for developing a program to visualize and explain basic operations on four types of trees:

- Requirements for tree data:
    - Generic tree (no special properties): A nonlinear hierarchical data structure consisting of nodes connected by edges and containing no cycles.
    - Binary tree: A tree where each node has at most two children.
    - Balanced tree: A tree where each leaf node is "not more than a certain distance" away from the root compared to any other leaf.
    - Balanced binary tree: A balanced binary tree that possesses the properties of both a binary tree and a balanced tree.
- User Interface requirements:
    - The user interface (GUI) needs to be designed to meet the following requirements:
        - Support undirected-weight trees with integer node values, and disallow duplicate node values.
        - For balanced trees and balanced binary trees, the maximum difference in distance from the root to the leaf nodes should be chosen by the user.
- Design requirements:
    - On the main menu, include the application title, a navigation bar for users to choose between the four types of trees, a help menu, and a quit button.
    - Users must select a type of data structure before entering the visualization mode.
    - The help menu should provide basic usage instructions and the project's objectives.
    - The exit button should exit the application, with a confirmation prompt for the user.

2.2 Identify the main tasks and functionalities of the application

The application encompasses the following core tasks and functionalities:

Create Tree:

- Users can create a new empty tree in the application. This task allows the initialization of a tree structure.

Insert Node:

- Users can insert a new node with a specified value as a child of a designated parent node. This task facilitates the expansion of the tree structure by adding new nodes.

Delete Node:

- Users can delete a specific node from the tree based on its value. This task enables the removal of nodes, thereby modifying the tree structure.

Update Node:

- Users can modify the value of an existing node to a new value. This task provides flexibility in updating node values within the tree.

Search:

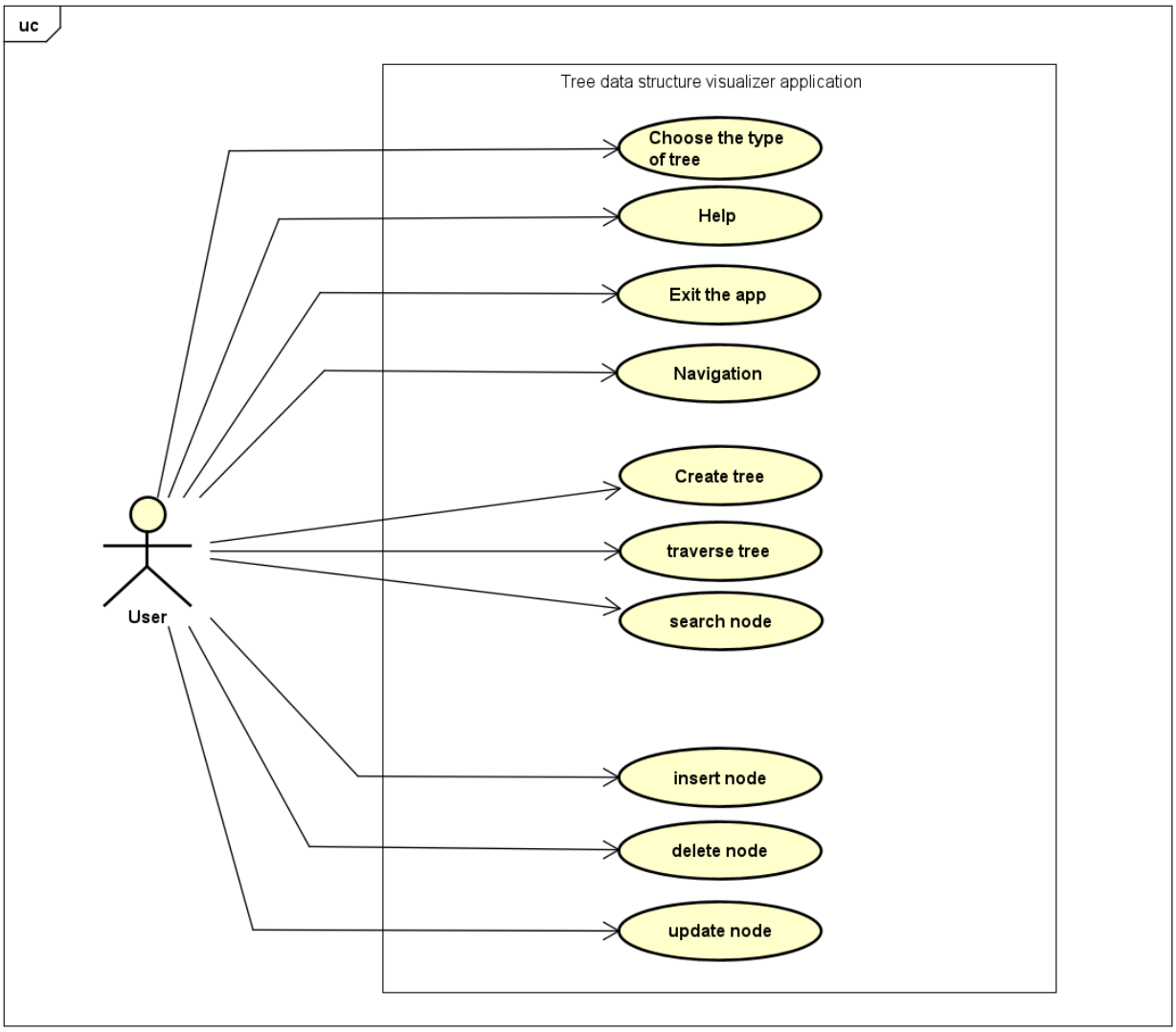
- Users can search for a node with a specific value within the tree. This task allows users to locate nodes based on their values.

Traverse:

- Users have the option to traverse the entire tree using either Depth-First Search (DFS) or Breadth-First Search (BFS) algorithms. During traversal, the current node is highlighted at each step, aiding users in understanding the tree's structure and organization.

These tasks and functionalities collectively provide users with the ability to interact with and manipulate tree structures effectively within the application.

2.3 Include a use case diagram to illustrate user interactions with the software

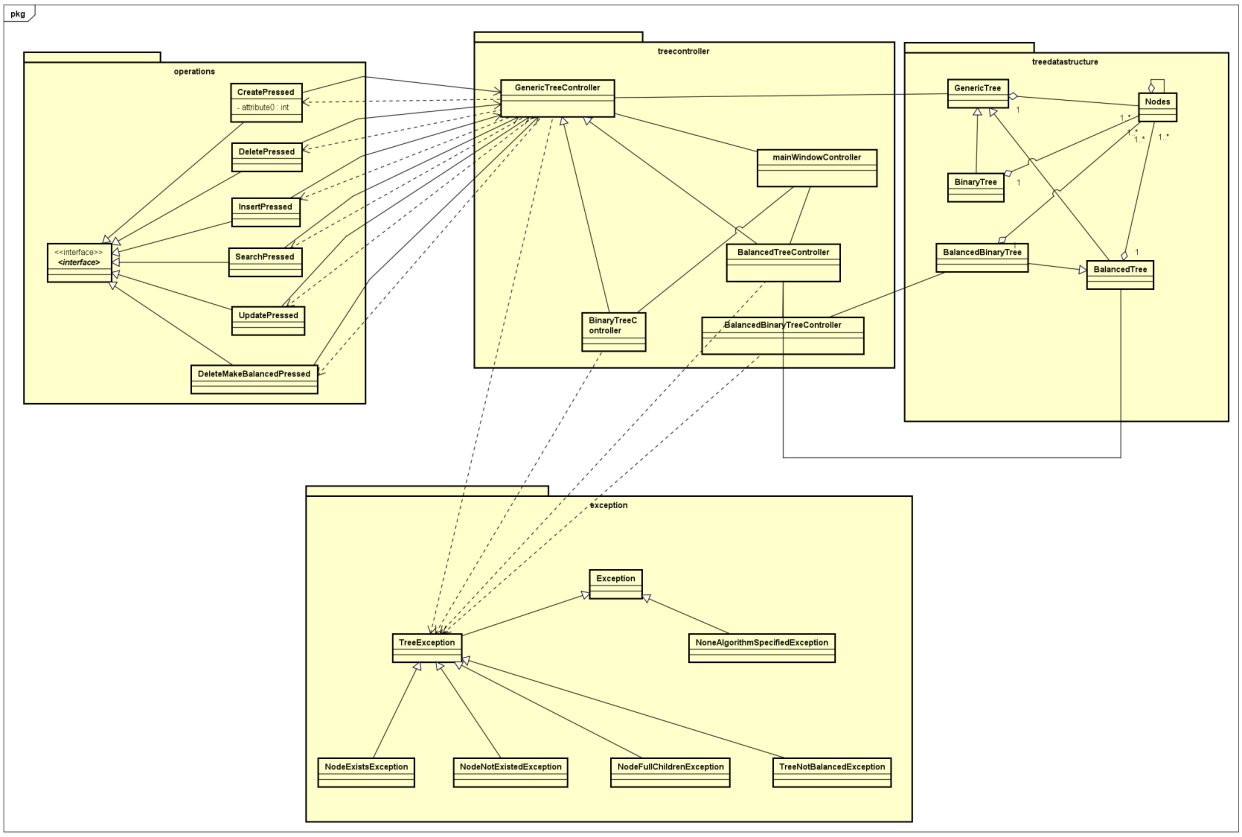


The software offers users an intuitive interface to choose the desired tree type for visualization. Within each tree visualization scene, users can seamlessly perform six fundamental tree operations, including creating a tree, traversing its structure, searching for specific nodes, inserting new nodes, deleting existing nodes, and updating node values. Furthermore, a helpful feature is the inclusion of a dedicated "Help" button, which provides users with valuable insights into the specific tree type being visualized. Lastly, the presence of an "Exit" button ensures a seamless termination of the program.

3. PROJECT DESIGN

3.1. General class diagram: Illustrate the classes and their relationships

During the software development process, we have built four main packages. These packages include the operations package, the tree data structure package, the tree controller package, and the exception package.



1. Operations package: This package contains code related to the operations and functionalities performed on the tree. It provides a graphical user interface (GUI) for visually interacting with and performing operations on the tree.

2. Tree data structure package: This package includes the tree data structures and classes that implement basic operations on the tree. It ensures data management and performs necessary operations such as adding, deleting, searching, and updating nodes in the tree.

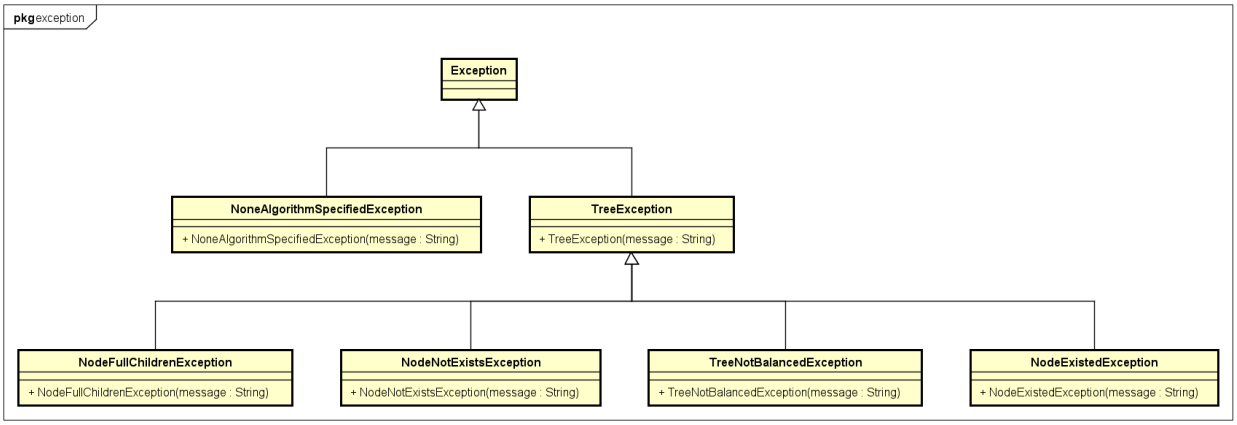
3. Tree controller package: This package contains the control files used to manage the user interface for specific tree data structures. It handles events from the user interface and calls appropriate methods in the tree data structure package to perform operations on the tree. This package also includes code related to the user interface of the tree. It defines UI components such as buttons, text boxes, labels, and others to create a user-friendly interface for interacting with the tree. This package also provides methods and classes to manage the display and update of the tree on the user interface.

4. Exception package: This package consists of six exceptions, throwed when some constraint of operations is violated.

These packages interact with each other to create a complete software for displaying, manipulating, and managing trees in a user interface environment.

3.2. Detailed class diagram: Illustrate the class and explain the relationships between classes and the implementation of important methods

3.2.1. Exception class diagram

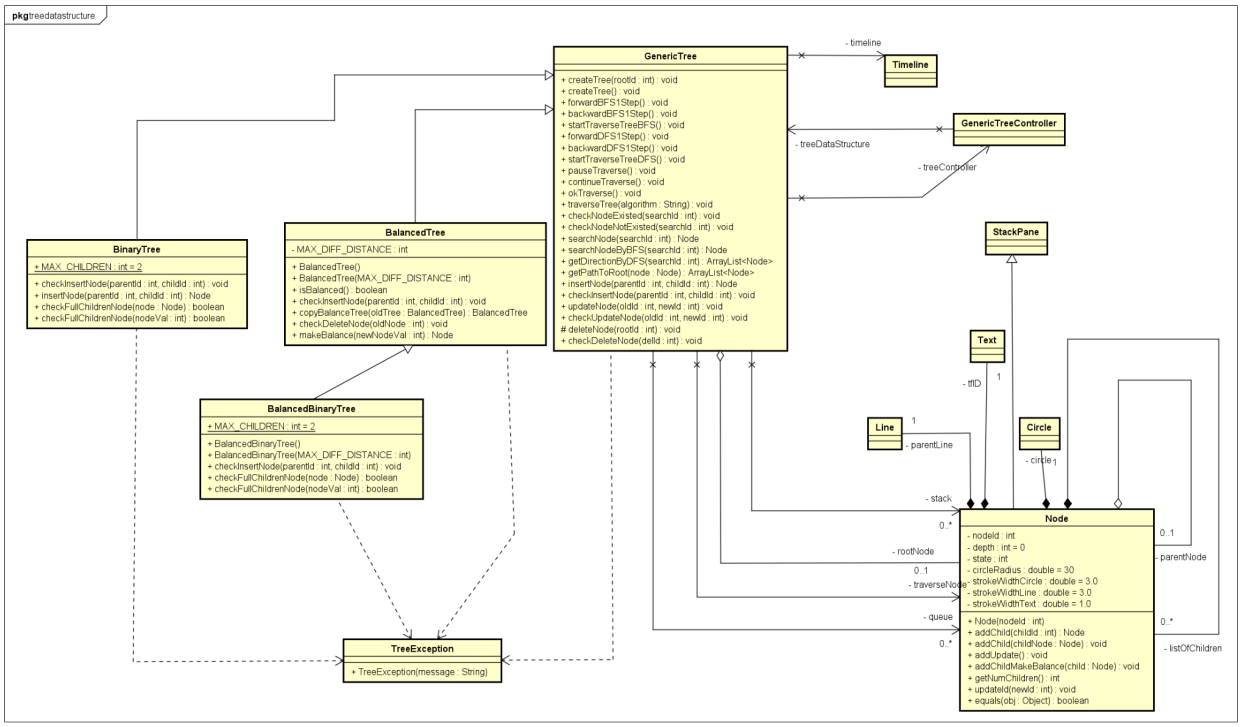


TreeException is a broad exception class that serves as a parent to four specific tree-related exceptions, each addressing different issues that may arise during tree operations.

- TreeNotBalancedException: This exception is raised when a tree's structure becomes unbalanced due to changes in the tree's nodes or the insertion/deletion of elements. TreeNotBalancedException helps to identify and handle such situations to ensure the tree maintains its balanced properties.
- NodeFullChildrenException: This exception is thrown when attempting to insert a new node into a binary tree, but the target node already has the maximum number of children allowed by the binary tree property. When this limit is violated during insertion, NodeFullChildrenException is raised, preventing the tree from deviating from its binary structure.
- NodeNotExistsException: This exception is raised when trying to access or perform an operation on a node that does not exist within the tree. It helps to handle scenarios where an operation or search is attempted on a node that hasn't been inserted or has been previously removed from the tree. NodeNotExistsException assists in identifying and managing such cases to prevent errors and unexpected behavior.
- NodeExistedException: This exception occurs when an attempt is made to insert a node into the tree, but the node already exists within the tree. It helps to address situations where duplicate nodes are not allowed in the tree, and a NodeExistedException is raised to avoid violating the tree's uniqueness property.

3.2.2. Tree data structure class diagram

The data structure package consists of four classes representing four types of trees, used for handling operations and exceptions under the hood. The data structure package concludes Generic tree class, Binary tree class, Balanced tree class and Balanced Binary tree class.



We will present detailed four classes in below:

- Generic tree class: Generic tree is the tree in which a node can have an arbitrary number of children's nodes. In the class, we handle six operations of tree as well as the corresponding exceptions for each operation under the hood. For each operation, we also develop a checking algorithm, to check if the operation is valid or not.

- Binary tree class: Binary tree class inherits from Generic tree class, with a few modifications in insert operation. This is due to the fact that each node in the binary tree has only its maximum of two children nodes. Therefore, when operating node insertion to a particular node, we have to check if the node is full of children or not. If the node cannot be added more, NodeFullChildrenException is raised.

- Balanced tree class: Balanced tree class is a subclass of the Generic tree class, designed to represent and manage a balanced tree data structure. Unlike a regular generic tree, a balanced tree maintains a specific property that ensures the depth difference among each leaf node is limited, leading to a more efficient tree structure. We have developed two algorithms for insert operation and delete operation to make the tree re-balanced again. That is, given an invalid operation that makes the tree unbalanced, the user would

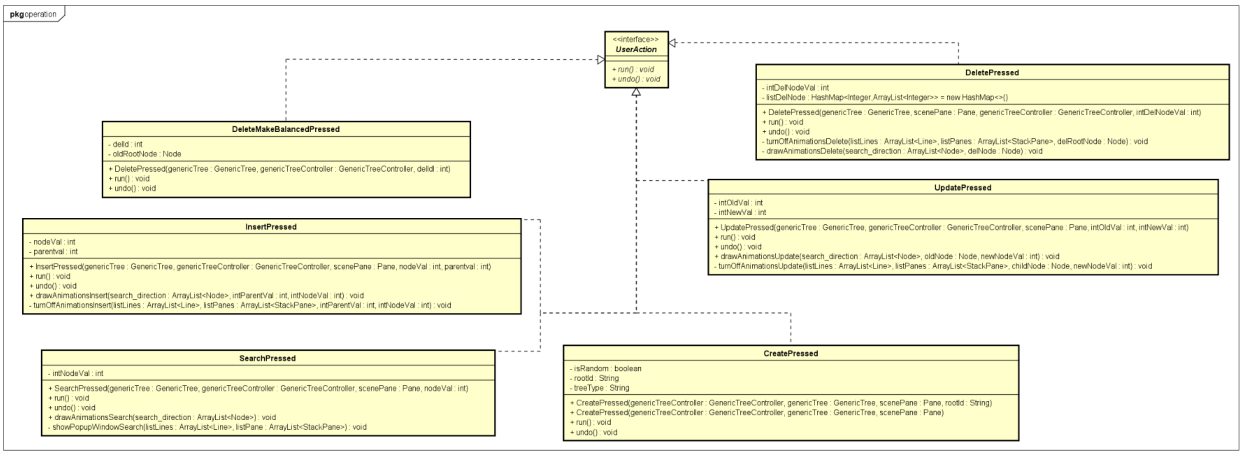
have the decision of undoing the invalid operation or not. If the user still wants to do the operation, we will run our algorithms, which depend on the current operation, to re-balance the tree. However, the trade-off if we still want to do the operation while maintaining the balance structure is that there are some nodes having their parent node changed.

- Balanced Binary tree class: Balanced Binary tree class inherits from the Balanced tree class and introduces additional methods and logic to maintain the binary property during insertions.

3.2.3. Operation class diagram

The package operation is used to support executing, displaying to GUI for the 5 operations: Create, Insert, Delete, Update, Search and the undo operations of these. This package contains 1 interface named UserAction and 5 classes which implement the interface UserAction. The 5 classes are responsible for 5 operations mentioned above, they are named after the operations, CreatePressed, InsertPressed, DeletePressed, UpdatePressed, SearchPressed. Besides, we have one more class named DeleteMakeBalancedPressed, which serves for the delete and make balance operation of BalancedTree.

The interface UserAction has two methods which are run() and undo(). The run() is executed when the button of operation on the screen is pressed by the user. The undo() is executed when the button “undo” on the screen is pressed.



Now, we will describe more details about these classes. These classes have the same following attributes:

+ genericController (GenericController) - the controller of the screen.
+ genericTree (GenericTree) - the tree that the user is working with.
+ scenePane (Pane) - the pane to display the tree.

Each class has some more different attributes for the purpose of executing the corresponding operation.

+ The CreatePressed class has two more attributes isRandom (boolean) and rootID (String) which are the user’s option (random or manual) to create the tree and the root’s ID given by the user, respectively.

+ The InsertPressed class has three more attributes. The first two ones are parenVal (int), nodeVal (int) which are the parent's ID and child's ID. The third attribute is parent (Node) which is used when the operation undo performs.
+ The DeletePressed class has three more attributes. The first one is intDelNodeVal (int), it is the ID of the node needing to be deleted, entered by the user. The two remain attributes are parentDelNode (Node) which is the node of the deleted node and listDelNode (HashMap <<Integer, ArrayList<Integer>>) which containing the ID of the node (as key) and the array of the IDs of its children (as value). These two attributes are used when the undo operation of deleting performs.
+ The UpdatePressed class has two more attributes. These are inOldVal (int) and intNewVal (int) which are old ID and new ID of the node that need to be updated, respectively. These attributes are also used when the undo operation performs.
+ The SearchPressed class has one more attribute. That is intNodeVal (int) which is the ID of the node that the user wants to find.
+ The DeleteMakeBalancedPressed class has one more attribute oldRootNode (Node) which is the rootNode of genericTree , it serves for the undo operation of Delete and Make balanced.

We did mention the functionality of the run() and the undo() methods which are implemented from the interface UserAction. Beside these 2 methods, the four classes InsertPressed, DeletePressed, UpdatePressed, SearchPressed also having two other methods which helps the method run():

+ drawAnimation(): to draw GUI animation of the corresponding operation, what it basically does is highlight the nodes and the lines from the root to the certain node in each operation. We implement this by using Transition of JAVAFX.
+ turnOffAnimation(): to clear the GUI animation and call the corresponding operation for the genericTree and make changes for the scenePane.

Class CreatePressed

The run() method is to create a tree. If the user chooses the manual option (isRandom = false), then the method calls the createTree method of the tree, then sets the controller for that tree and adds the root node to the scenePane. Otherwise, a random tree (random number of nodes and random IDs of nodes) is created without violating any constraints of the type of tree. There is no animation for this operation, so the two methods drawAnimation() and turnOffAnimation() are not used.

The method undo() simply just clear the tree by setting its root to be null and remove the tree from the scencePane.

Class InsertPressed

The run() method is to insert a node to the tree. Firstly, it finds the parent node (by method search ()) for the undo functionality. Then, it highlights all the nodes from root to the parent node (by using drawAnimation() method). Then, it will insert the node to the genericTree, display the new node to GUI and remove the animation from GUI.

The undo() method just removes the child node from the tree and scenePane (GUI).

Class DeletePressed

The run() method is to delete a node from the tree. Firstly, it finds the parent node of the deleted node and saves the information of the subtree that has the root as the deleted node,

for the undo functionality. Then, it highlights all the nodes from the root to the deleted node like the previous class does. After that, it deletes the node in the genericTree, and adjusts the animation of the tree on screen.
The undo() method inserts the node and its subtree into its parent node which is saved at the beginning of the method run().

Class SearchPressed

The run () method is to search a node. Firstly, it finds the node by using the search method of genericTree. Then, it highlights all the nodes from root to the search node like the previous class does. After that, it will pop up a window to announce whether the node exists and is found, and remove the animation on GUI.
The undo() method simply prints out a line “Search operation undo”.

Class UpdatePressed

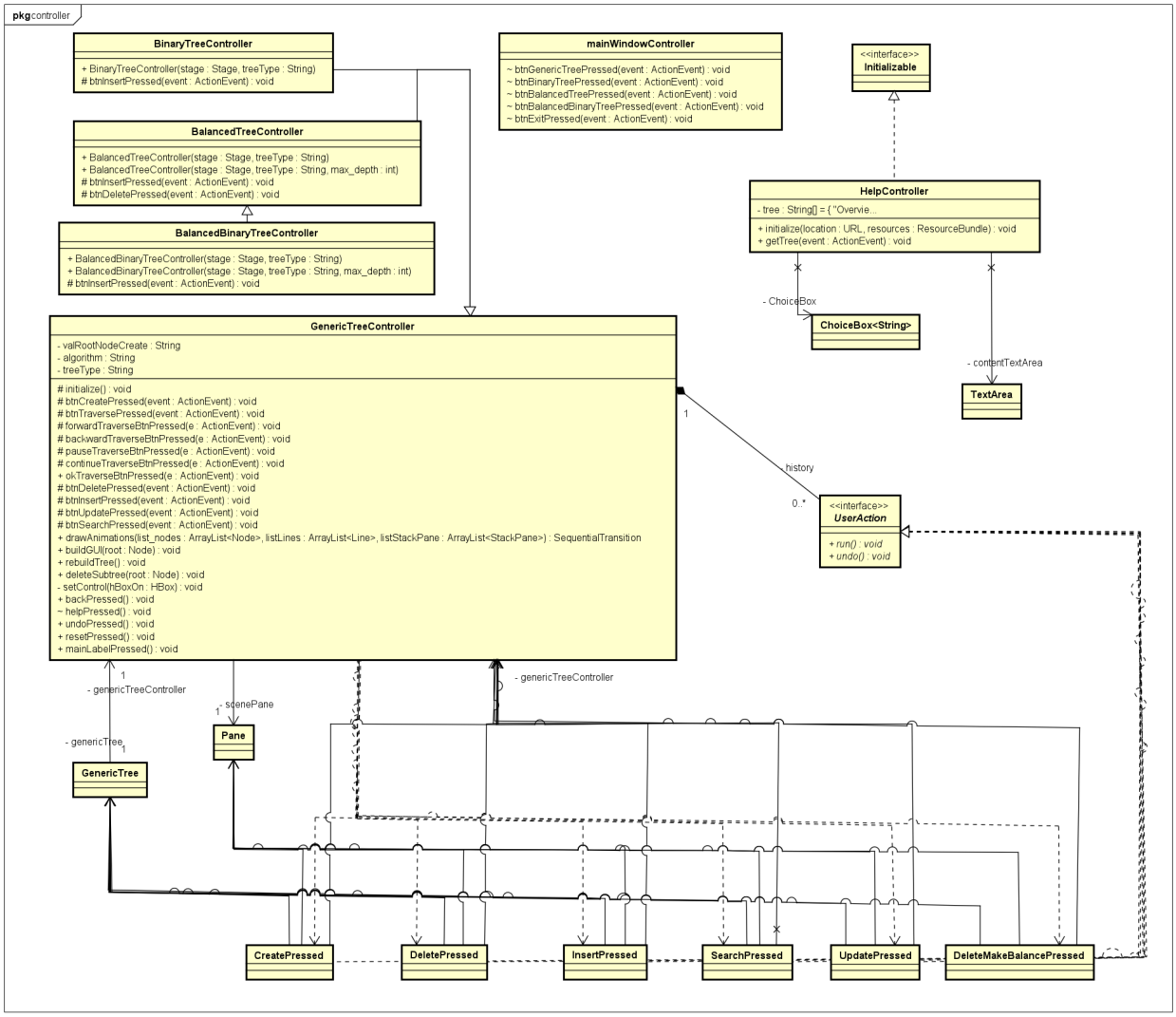
The run () method is to change the ID of a node. Firstly, it finds the node by using the search method of genericTree. Then, it highlights all the nodes from the root to the updating node like the previous class does. After that, the ID of the node is changed by using the method update of genericTree and the ID on the screen is changed.
The undo() method just simply searches for the node with the new ID and then recovers the ID to the old one.

Class DeleteMakeBalancedPressed

This class is used when the user deletes a node in the BalancedTree and that makes the BalanceTree unbalanced. In this case, it is a substitution for the DeletePressed class.
The run() method is to delete the node on the tree and then make the tree balance by using the method of rotating a tree, which is defined in the class BalancedTree.
The undo() method just simply sets the old rootNode for the BalancedTree and deletes the new BalancedTree which is rotated by the method run(). Then, it displays the old tree on screen.

3.2.4. Controller class diagram

The package controller contains not only the package operation we mentioned above but also, of course, the controller classes. There are 6 controller classes.Firstly, we have the GenericTreeController which is the biggest controller class. Two controllers BalancedTreeController and BinaryTreeController inherit from the GenericTreeController. Then , we have the BalancedBinaryTreeController inheriting from the BalancedTreeController. These are the controllers for each type of tree’s screen. Besides, we have the MainWindowController which controls the main window of the application. The final controller is HelpController which controls the pop-up “Help” window of the app.



The MainWindowController controls 4 buttons to create 4 types of tree and 1 button to Exit.
The HelpController controls the help pop-up window.
The GenericTreeController has some main attributes:

+ treeDataStructure (GenericTree): the data structure tree that the user is interacting with.
+ scenePane (Pane): the Pane lies in the center of the BorderPane of the scene. It contains the display of the tree.
+ history (Array<UserAction>): to store the history of operations that the user calls.

The GenericTreeController controls 6 basic tree operations including Create, Traverse, Insert, Delete, Update, Search and 4 other operations including Back, Undo, Reset, Help.
For each of the 5 basic operations (except for the Traverse one), the handler function that is defined in the controller is performed as following:

1. Get the required ID of the nodes, given/entered by the user.
2. Check the condition (if needed) to perform the operation.
   If the condition is satisfied, then perform the next step.
   Else, stop and display an announcement/warning to the user.

3. Create an object that implements the interface UserAction in package operation, the object to create is corresponding to the operation. (Example: operation Create → object CreatePressed).
4. Call the function run() of the object, which will draw animations, clear them and do stuff for the tree and screen, as we mentioned above.
5. Add the object to the attribute history of the controller, for the purpose of executing undo operation if the user requires. (This step is no need for the operation Search because the data structure tree remains the same after calling search method).
6. Clear the textfield that the user enters the ID of nodes.

The Traverse operation doesn't need any class to perform the operation because there's no undo operation for this Traverse operation. Besides, this operation also needs stop, continue, backward, forward functionalities which we implement by using the TimeLine class of javafx and we can't find out the way to perform this by the class that implements the interface UserAction.

Four operations Create, Traverse, Update, Search have the same executing way for all 4 types of tree. So, the BalancedTreeController and BinaryTreeController can inherit these 4 operations from the GenericTreeController. The BalancedTreeController needs to override the method for insert and delete operation for the reason that inserting or deleting a node can make the tree unbalanced, whereas the BinaryTreeController just needs to override the method for inserting because a BinaryTree node has the maximum number of children at two children. The BinaryBalancedTreeController inherits from the BalancedController but it overrides(just a bit) the method for inserting because of the node's maximum number of children.

3.3. Explanation of OOP techniques in our design

Inheritance

Perhaps inheritance is the technique that is widely used in our project. We have used inheritance when implementing data structure class and controller class. Specifically, in data structure class, the BinaryTree and BalancedTree inherit from the GenericTree, which is the baseline tree, and the BalanceBinaryTree inherits from the BalancedTree. In general, due to the fact that there are some trees that are identical to the other trees with only few modifications when handling exceptions, the idea of inheritance has to be made in order to avoid redundant code. 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. For example, in class Node, 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. In conclusion, using Encapsulation helps us create a logical and coherent structure for building trees.

Polymorphism

Firstly, we have used the Polymorphism technique when implementing an "undo" operation. We want to keep track of all of the user's actions, regardless of any operations. Hence, the idea of having an Interface for all operations has to be made, since we can upcast any operation into the interface, leading to an efficient operation cache.

Secondly, we have used the technique when implementing each operation. In each operation implementation, there are cache of data structure objects and controller objects. That being said, we would need to implement each operation that is fit with the baseline tree, which is GenericTree, then we will upcast other trees to its baseline tree when doing any operation. The idea of polymorphism leads to efficient code maintainance.

Association/Aggregation/Composition

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

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.

The composition is used when the Node contains three objects Circle, TextField, Line which are used to display the node on screen. When the Node is gone, its three objects also disappear. 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. So there is a composition relationship between the class GenericTreeController and the interface UserAction.

Dependency

Dependency is a fundamental concept in OOP and there are several dependencies between classes and packages in this project. Firstly, we have the controller classes (GenericTreeController, BalancedTreeController, and BalancedBinaryTreeController) depending on the tree data structure classes (GenericTree, BalancedTree, BalancedBinaryTree) in order 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. Secondly, 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. Moreover, 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.

4. USER INTERFACE ACCOMPLISHED

4.1 Describe the user interface of the application

The user interface of the application consists of two main views: the main interface and the selected tree type interface.

The main interface, has a user-friendly layout called BorderPane, which includes the following components:

- Center: A centered VBox that contains buttons for selecting the type of tree structure, such as Generic Tree, Binary Tree, Balanced Tree, and Balanced Binary Tree. Additionally, there is an exit button for closing the application.
- Top: A VBox that houses a title label with the text "Tree Data Structure Visualizer."

Once the user selects a tree type, the interface dynamically switches to the corresponding view. This view also follows the BorderPane layout and encompasses the following components:

- Top: A VBox containing navigation and control buttons, including options like "Back," "Reset," "Undo," and "Help."
- Center: A Pane named scenePane, where the tree visualization is displayed, providing users with a graphical representation of the data structure.
- Bottom: A StackPane named stackPaneInput, which serves as an interactive input area for performing various tree operations, including search, creation, traversal, deletion, insertion, and updating.
- Right: A VBox named vBoxPseudo, consisting of a StackPane named stackPanePseudo, where the pseudo code related to the selected algorithm (BFS or DFS) is displayed. Additionally, there is another StackPane named stackPaneController that allows users to control the traversal process.
- Left: A VBox containing a TitledPane named Operations, which presents buttons for each operation, such as Create, Insert, Delete, Update, Traverse, and Search.

4.2 Explain the functions and features of the user interface

The main interface of the application displays a list of selection buttons for users to choose the type of tree, including Generic Tree, Binary Tree, Balanced Tree, and Balanced Binary Tree. When users click on any of these buttons, the interface will switch to the corresponding view based on the selected tree type. Additionally, there is an "Exit" button for users to exit the application.

On the interface corresponding to each tree type, users will find the following components:

- "Back" button: Allows users to return to the main interface and select a different tree type.
- "Reset" button: Resets the tree's state to its initial state.
- "Undo" button: Reverses the previous operation on the tree, restoring the state before the operation.
- "Help" button: Provides understanding about the type of tree the software is visualizing

Other components on the interface, specific to each tree type, include:
- Input fields: The interface provides input fields for users to provide detailed information about the tree and the desired operations. These include:

+ "Root node’s value" input field: Enables users to input the value for the root node when creating the tree.
+ "Parent node’s value to insert" input field: Allows users to input the value of the parent node they want to insert into the tree.
+ "Child node’s value to insert" input field: Allows users to input the value of the child node they want to insert into the tree.
+ "Node’s value to delete" input field: Enables users to input the value of the node they want to delete from the tree.
+ "Node’s value to update" input field: Allows users to provide a new value for the node they want to update in the tree.
+ "Node’s value to search" input field: Enables users to input the value they want to search for within the tree.

- Operation buttons: The interface provides operation buttons for users to perform specific operations on the tree. When users click on each button, the corresponding result will be displayed. These include:
+ "Create tree" button: When clicked, the tree will be created based on the inputted root node value.
+ "Insert node" button: Allows users to insert a new node into the tree based on the inputted node value.
+ "Delete node" button: Users can delete a node from the tree based on the inputted node value.
+ "Update node" button: Enables users to update the value of a node in the tree based on the inputted node value.
+ "Traverse tree" button: When users click this button, the tree will be traversed, and the corresponding result will be displayed.
+ "Search" button: When clicked after inputting the value to search, the search result will be displayed.

When users click the "Traverse tree" button on the interface, the tree will be traversed using the previously selected method (DFS or BFS).

If the user has chosen Depth-First Search (DFS) as the traversal method, the traversal process will start from the root node and continue in a left-to-right, top-to-bottom order. As the tree is traversed, the nodes will be marked or displayed in the order of DFS traversal. The result of the tree traversal will be visually displayed on the interface, allowing users to observe the traversal order and the tree's structure.

If the user has chosen Breadth-First Search (BFS) as the traversal method, the traversal process will start from the root node and continue based on the levels of the nodes, traversing nodes at the same level before moving to the upper or lower levels. As the tree is traversed, the nodes will be marked or displayed in the order of BFS traversal. The result of the BFS tree traversal will also be displayed on the interface, enabling users to observe the traversal order and the tree's structure.

Furthermore, the software will display the pseudocode representing the traverse algorithm. The user can pause or continue the program during execution.

After the user performs operations such as tree creation, searching, and tree traversal on the interface, the results will be displayed to provide users with a clear visualization of the tree's structure and outcomes. The interface represents this information using graphical charts, providing users with an overall and intuitive understanding of the tree and the results of their tree operations.

4.3 Specify any libraries, frameworks, or tools used for UI development

In this project, we utilize the following libraries, frameworks, and tools for UI development:

1. JavaFX: JavaFX is a UI development platform for Java applications. It provides a set of classes and components to build beautiful and interactive user interfaces. We employ JavaFX to create the main user interface for the application.

2. FXML: FXML is an XML-based markup language for building JavaFX user interfaces. We employ FXML code to define the structure and components of the user interface. FXML allows us to describe the user interface in a visual and readable manner, separating the UI design from Java code.

3. Scene Builder: Scene Builder is an intuitive tool for designing JavaFX user interfaces through a drag-and-drop interface. We utilize Scene Builder to easily and quickly create and modify the user interface. Scene Builder enables us to visually add, remove, and arrange UI components, combined with FXML code to construct the precise user interface as required.

4. Astah: Astah is a software analysis and design tool that supports drawing UML diagrams and building models. We employ Astah to design the user interface model and the data structure of the application.

5. Java Development Kit (JDK): JDK is the official Java software development kit provided by Oracle. We use JDK to compile and run the Java application.

6. Git: Git is a distributed version control system used to track and manage source code during development. We utilize Git to manage versioning and merge changes in the source code.

With the support of JavaFX, FXML, Scene Builder, Astah, and other tools like JDK and Git, we can develop flexible user interfaces and easily manage the project's development process.

5. RESULTS AND EVALUATION

5.1 Present the outcomes of the project

The project has yielded the following outcomes:

- Successful deployment of the user interface based on four types of trees: Generic Tree, Binary Tree, Balanced Tree, and Balanced Binary Tree.
- Implementation of core functionalities such as tree creation, node insertion, node deletion, node updating, and tree traversal.
- Intuitive and interactive visualization of tree operations.
- Compatibility and stable performance in the deployed environment.

5.2 Evaluate the fulfillment of the project requirements

The project has effectively met the defined project requirements. Specifically, the following requirements have been fulfilled:

- The user interface displays different types of trees and allows users to select and switch between them.
- Key functionalities such as tree creation, node insertion, node deletion, node updating, tree traversal, and node search have been successfully implemented.
- The user interface provides components for user input and facilitates the execution of operations on the trees.
- The results of tree operations are presented in a visual and comprehensible manner.

5.3 Assess the performance and reliability of the application

The performance and reliability of the application have been evaluated through rigorous testing and experimentation. The assessment results are as follows:

- The application operates smoothly and reliably, even when dealing with large and complex trees.
- The execution time of basic operations, such as tree creation, node insertion, node deletion, tree traversal, and node search, is considered fast and performs well, even with a significant number of nodes.
- The user interface is responsive and remains stable during user interactions.
- The application demonstrates a high level of stability, with no critical errors or significant issues encountered during usage.

In summary, the project has achieved its intended goals, delivering a visually appealing and functional application for visualizing and performing operations on tree data structures. The successful implementation of required features and functionalities, including node search,

demonstrates the fulfillment of project requirements. The application exhibits satisfactory performance and reliability, offering a seamless user experience. Continuous monitoring, optimization, and thorough testing will further enhance the application's performance and reliability in the long run.

6. USER GUIDE

6.1 Provide instructions for using the application for end-users

Step 1: Installation

- Download the application setup (zip file) from the provided source on GitHub.
- Unzip the downloaded file and initiate the installation process.

Step 2: Launching

- Open the 'MainWindowApplication.java' file in the 'test' package and run it.
- The main menu will be displayed.

Step 3: Main Menu

- In the main menu, you will see the application title and a navigation bar to choose between the 4 types of trees: generic tree, binary tree, balanced tree, and balanced binary tree.
- Click on the desired tree type to proceed.

Step 4: Tree Visualization

- After selecting a tree type, the application will display the visualization of the chosen tree structure. It will provide a clear representation of the nodes and their relationships.

Step 5: Performing Operations

- On the tree visualization screen, you can perform various operations on the selected tree. These operations include:
    - Create, Insert, Delete, Update, Traverse, and Search.
- To perform an operation, navigate to the respective operation from the available options.
- Provide the necessary parameters for the operation.
    - For example, in the INSERT operation, you need to enter the value of the parent node and the value of the new node.
- Click the button to initiate the operation.

Step 6: Operation Execution

- During the execution, the code panel will display pseudo code for the specific operation (e.g., traverse operation).
- The executing line will be highlighted, allowing you to follow the progress of the operation.

Step 7: Controlling the Execution

- You can pause, continue, go backward, or forward during the execution of the traverse operation.

Step 8: Undo, Reset, Back, and Help

- The application supports the undo functionality, allowing you to undo operations.
- The Reset button allows you to reset the entire process.
- At any point, you can press the Back button to return to the main menu.
- Press the Help button for usage instructions and information about tree types.

6.2 Include installation and configuration guidelines

6.2.1 System Requirements:

- Operating System: Windows, MacOS.
- Java Runtime Environment (JRE) or later.
- JavaFX library**.**

6.2.2 Configuration:

- Ensure that you have JavaFX installed on your system and properly configured with your Java Development Kit (JDK).

  *Note: If you encounter any issues, make sure you have the latest JDK and JavaFX version installed and try running the application again.

6.2.3 Uninstallation:

- To uninstall the application, simply delete the folder that you downloaded from the provided source on GitHub.

7. CONCLUSION

7.1 Summarize the findings and achievements of the project

The project has achieved significant findings and accomplishments. It has successfully developed a user-friendly and functional application that enables visualization and execution of operations on tree data structures. The project's outcomes include the successful implementation of essential features, intuitive visualization of tree operations, and compatibility with the deployed environment. These achievements highlight the project team's ability to provide an effective solution for working with different types of trees. The application's user-friendly interface and robust functionality contribute to its success in facilitating tree-related operations and enhancing user experience.

7.2 Evaluate the overall development process

Requirements Analysis: The project team conducted a comprehensive analysis of requirements, ensuring a clear understanding of desired features and user expectations.
Design and Implementation: The team successfully translated the requirements into an intuitive user interface and implemented the necessary functionalities using appropriate technologies and tools.
Testing and Quality Assurance: Rigorous testing was conducted to ensure the stability, reliability, and performance of the application. Bugs and issues were promptly addressed, resulting in a robust and reliable product.
Collaboration and Project Management: Effective collaboration and project management methodologies were employed, facilitating smooth coordination among team members and timely completion of tasks.

7.3 Suggest future development and improvements

To enhance the project and address areas for improvement, the following suggestions are proposed:

Functionality Expansion: Consider adding additional tree types or advanced operations to cater to diverse user needs and applications.

User Interface Refinement: Continuously improve the user interface by incorporating user feedback, enhancing visual appeal, and optimizing usability.

Performance Optimization: Conduct testing and performance optimization to ensure the application can efficiently handle large and complex tree structures.
Error Handling and Validation: Enhance error handling and input validation mechanisms to provide intelligent and user-friendly error messages.
Cross-Platform Support: Explore compatibility with different operating systems, expanding the application's reach to a wider user base.
By integrating these suggestions, future development efforts can maximize the application's capabilities, user satisfaction, and overall success.

8. REFERENCES

[1] Baeldung:Link: https://www.baeldung.com/
[2] Data Structures and Algorithms in Java by Robert Lafore:
Link:https://everythingcomputerscience.com/books/schoolboek-data_structures_and_algorithms_in_java.pdf
[3] Gluon Scene Builder User Guide:
Link: https://docs.gluonhq.com/scenebuilder/#_introduction
[4] Introduction to Algorithms by Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein:
Link:https://dl.ebooksworld.ir/books/Introduction.to.Algorithms.4th.Leiserson.Stein.Rivest.Cormen.MIT.Press.9780262046305.EBooksWorld.ir.pdf
[5] Java Code Geeks: https://www.javacodegeeks.com/
[6] JavaFX Documentation Project:
Link: https://fxdocs.github.io/docs/html5/
[7] Oracle Scene Builder Documentation:
Link: https://docs.oracle.com/javafx/scenebuilder/overview.htm
[8] Stack Overflow: Link: https://stackoverflow.com/
[9] Visualgo - BST Visualization: Link: https://visualgo.net/en/bst