**Object-Oriented Language and Theory**

**Mini-Project Team 5 – ICT Report**

**Members of Team 5 – ICT:**

Tran Quang Hung – 20226045

Nguyen Huu Hoang – 20225972

Nguyen Dinh Son - 20225997

Pham Phuong Huy - 20225977

**Table of contents**

[1. Mini-project description](#Chap1)

* 1. [Mini-project requirement details](#Part1Chap1)
  2. [Use case diagram and explanation](#Part2Chap1)

2. [Assignment of member](#Chap2)

2.1. [Detail for classes/methods](#Part1Chap2)

2.2. [Reference usage](#Part2Chap2)

[3. Design](#Chap3)

[3.1. A general class diagram](#Part1Chap3)

[3.2. Detailed class diagrams](#Part2Chap3)

[3.3. Explanation of the design](#Part3Chap3)

1. **Mini-project description**
   1. **Mini-project requirement details**

* Overview:
  + Design a program to display and explain some basic operations four types of tree:
    - Generic tree
    - Binary tree
    - Balanced tree
    - Balanced binary tree
  + Operations: create, insert, delete, update, traverse, search.
* Specifications:
  + GUI
  + Design:
    - Consider only undirected-weight trees, with integer node values and no duplicated node values allowed.
    - For the balanced tree and balanced binary tree, the maximum difference in distance from root of the leaf nodes must be chosen by the user.
    - On the main menu: title of the application, navigation bar for user to choose between the four types of tree, help menu and quit
      * User must select a type of data structure before getting into the visualization.
      * The help menu shows basic usage and aim of the project.
      * Quit button exits the application, ask for confirmation.
  + In the visualization:
    - User can choose to visualize one of six operations, by selecting an option on the operations menu, and then provide the necessary parameter.

|  |  |  |
| --- | --- | --- |
| Operations | Parameters | Description |
| Create | None | Create a new empty tree |
| Insert | Value of parent node, value of new node | Add the new node with specified value as a child of the specified parent node |
| Delete | Value of the node | Delete the node from the tree |
| Update | Current value of the node, new value of the node | Change the node with current value to new value |
| Traverse | Algorithm (DFS or BFS) | Traverse all node in the tree (highlight current node in each step of traversal) |
| Search | Search value | Search for the node value in the tree |

* When an operation starts to execute, on the code panel, the pseudo code (or actual code) should be displayed, and the currently executing line is highlighted to help the user keep track of the process. On the bottom bar, the user can see the progress bar of the executing operation and choose to pause, continue, or go backward or forward a step in the execution.
* The user can also undo or redo operations from the bottom bar.
* Always have a Back button for user to return to main menu at any time.
  1. **Use case diagram and explanation**
* Use case diagram:

A diagram of a diagram

Description automatically generated

* Explanation: How the users interact to the software with use cases
  + In the main menu, user can: select type of tree, view the help menu, exit program.
  + After selecting the type of tree in the main menu, user can choose 1 operation among 6 operations to visualize at a time. Any time user can select “Back” button and select other types of tree. This action does not delete the details of inputted tree (save all tree states). User can see the help menu and exit with confirmation in the main menu.
  + Those are considered as use cases because:
    - They contain sequence of actions.
    - They are performed by the program.
    - They all give observable result to actor.

1. **Assignment of member**
   1. **Detail for classes/methods**

* Package oop.ict.project.tree:
  + Package oop.ict.project.tree.generic: the whole team
  + Package oop.ict.project.tree.binary: Nguyen Dinh Son
  + Package oop.ict.project.tree.balanced: Nguyen Huu Hoang
  + Package oop.ict.project.tree.balancedbinary: Pham Phuong Huy
* Package oop.ict.project.shape: the whole team
* Package oop.ict.project.gui:
  + Class ScreenController: Tran Quang Hung
  + Class ScreenMenuController: Tran Quang Hung
  + Class GraphicTree: Tran Quang Hung
  + Class MainMenuScreen: Tran Quang Hung
  1. **Reference usage**
* Generic tree traversal: *Traversals in a Generic Tree | Solution*

<https://www.youtube.com/watch?v=YnufWAWOfI8&t=518s>

* GUI implement:
  + Circle and Line for visualizing tree: *github EricCanull/fxbinarytree* <https://github.com/EricCanull/fxbinarytree/tree/master/src/main/java/shape>
  + GraphicTree: *git EricCanull/fxbinarytree* <https://github.com/EricCanull/fxbinarytree/tree/master/src/main/java/controller>
* Other references for built-in packages, classes, methods: *Java Documentation* <https://docs.oracle.com/en/java/>

1. **Design**

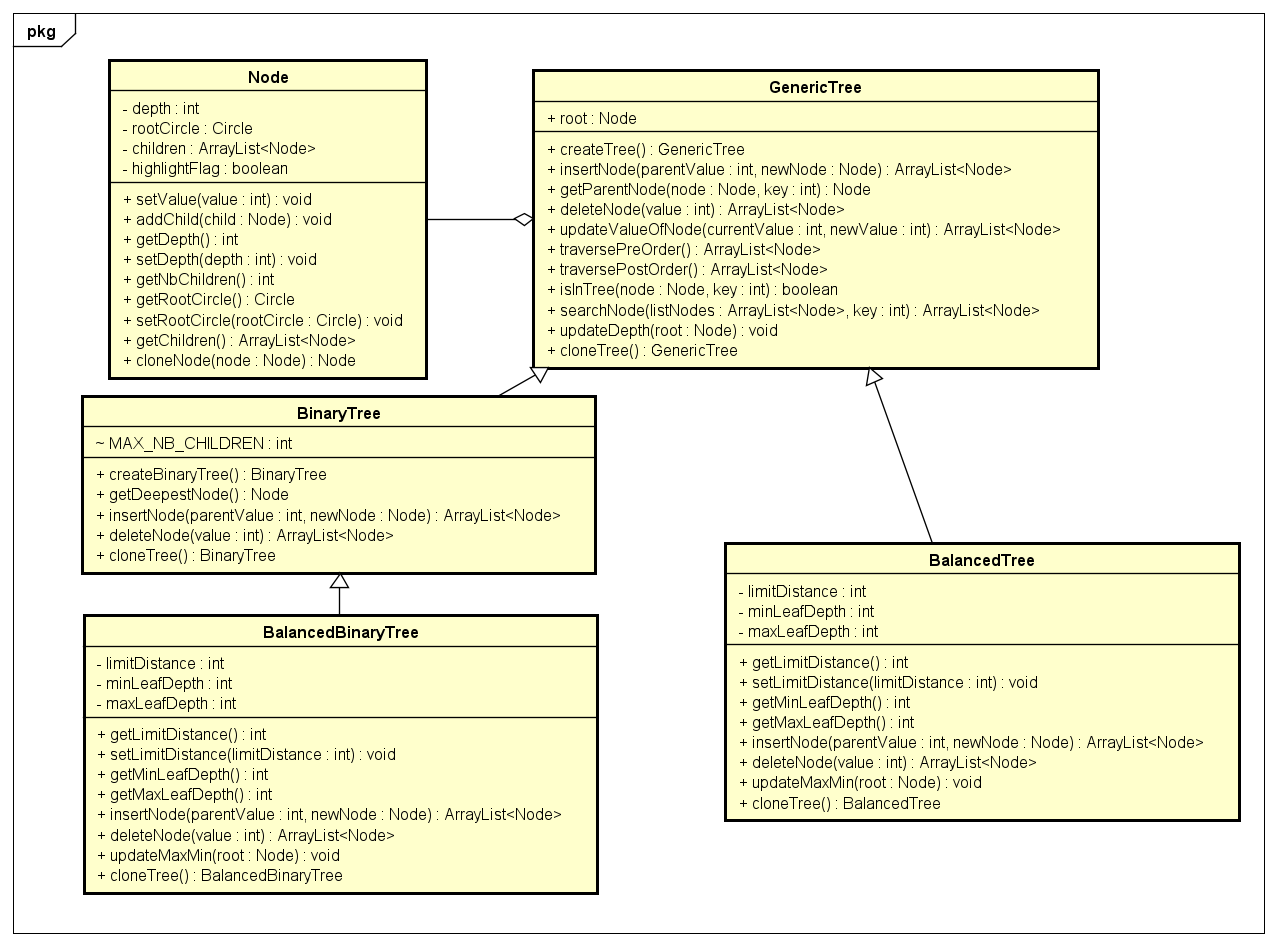
**3.1.** **A general class diagram**

A diagram of a diagram

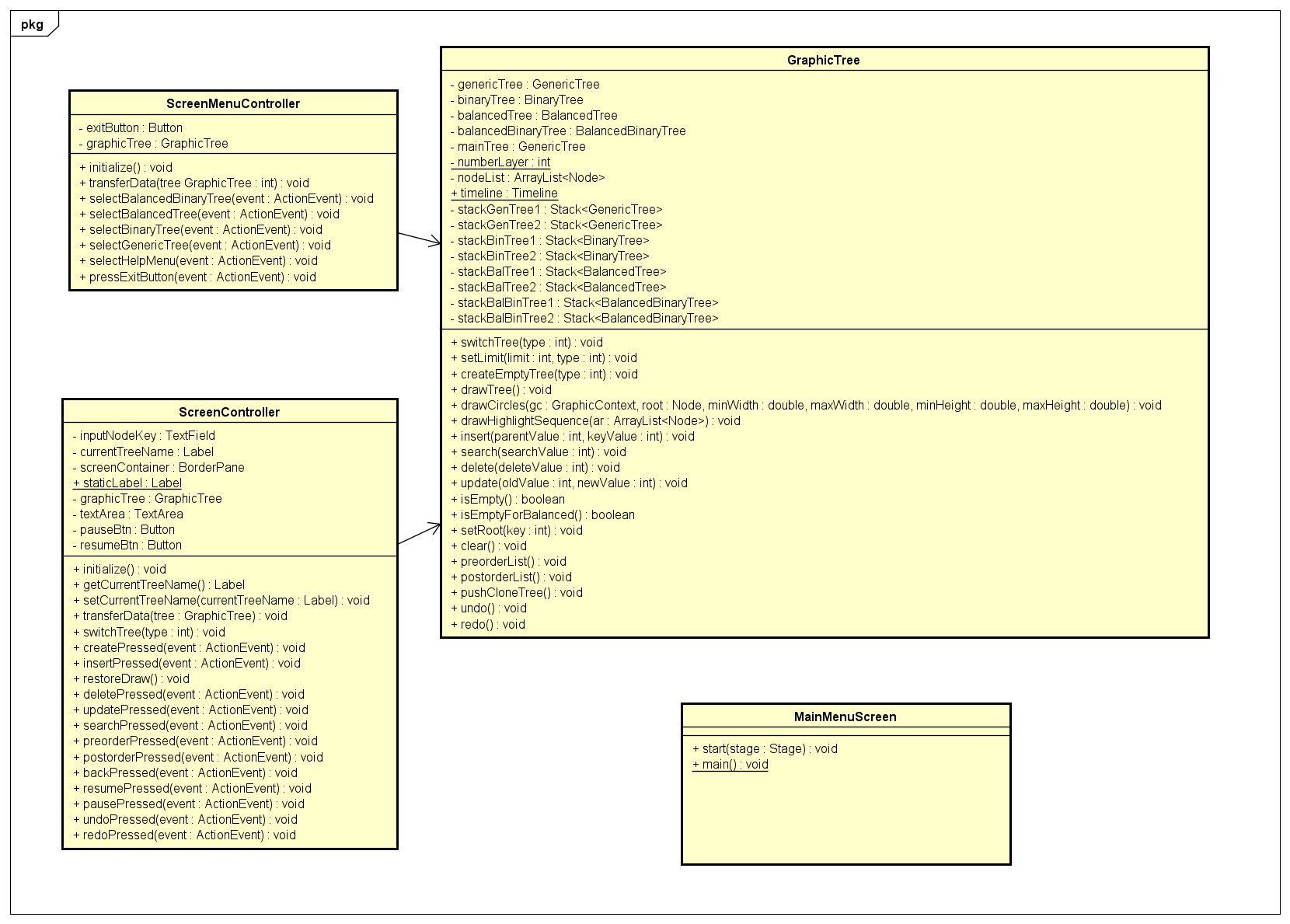
Description automatically generated

**3.2.** **Detailed class diagrams**

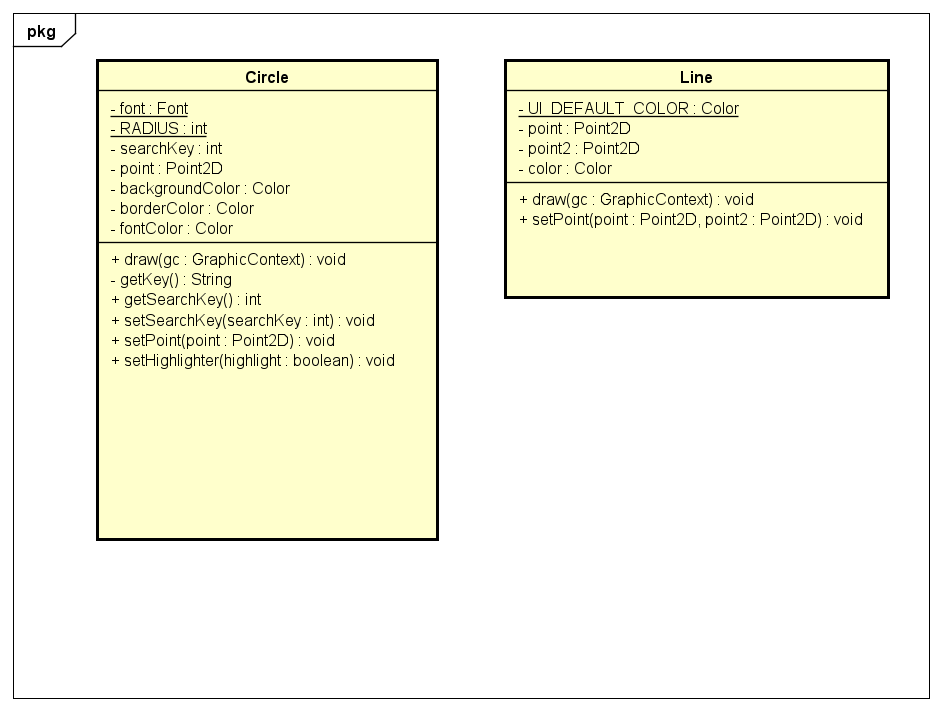
* Package oop.ict.project.tree contains 4 classes: GenericTree, BinaryTree, BalancedTree, BalancedBinaryTree.



* Package oop.ict.project.gui:



* Package oop.ict.project.shape:



**3.3.** **Explanation of the design**

* Describe the relationships between classes
  + Between tree classes:
    - GenericTree is the superclass of other 3 classes as all other types of tree have the same attributes and operations. Override technique is used to reuse some methods from Generic Tree when we applied for Binary, Balanced and Balanced Binary Tree.
    - BalancedBinaryTree: inherited from BalanceTree and BinaryTree as it is a kind of special binary tree.
    - Class Node: one node can have multiple children so the attribute contains an ArrayList of Node. Node is also a part of Tree so Aggregation is used in this case.
  + Between package oop.ict.project.gui:
    - ScreenMenuController controls the MainMenuScreen, ScreenController controls GraphicTree.
    - Those two controller classes use GraphicTree class, so Association is used in this case.
    - Designe a class named Graphic Tree to determine which elements need to visualize a tree, and also some methods to help us illustrate operations on each tree data structures. Polymorphism is applied in Graphic Tree by upcasting the other four trees to the main tree. Then almost all the operations were used in the main tree.
  + Between package oop.ict.project.shape:
    - Circle and Line are used for GUI and do not have relationship.
* The implementations of some important methods
  + Implement GenericTree’s methods which will be inherited from other tree classes: Using an attribute called root of type Node and basic knowledge about algorithms.
  + Child classes of GenericTree override methods: insertNode, deleteNode because they have other constrains to satisfy their properties.
  + Implementation of GUI:
    - ProcessArea, PseudoArea, History Box and ResultArea is used to make user understand what happening. Each time a method is executed, they print what this app is doing.
  + Implementation of GraphicTree’s methods:
    - drawTree(…): draw lines and circles which are assigned a part of the total screen. Number of layers (depth): maximum 8 layers.
    - drawHighlightSequence(): highlight each node needed to go through in each operation, using Timeline and KeyFrame objects to highlight one node at a time.
  + Implementation of Undo and Redo:
    - Using stack which contains the tree of previous states of the working tree.
    - Using deep copy: whenever we change the working tree, the copy of previous state of that tree will not be affected. Implement method cloneTree() in each tree class.
    - Undo using 1 stack and Redo using another stack. There are 8 stacks, each 2 stacks of one kind of tree.
  + Implementation of Pause and Resume
    - Using timeline.pause và timeline.play with setting visible of those button (only in the middle of the operations).