# BÁO CÁO MINI PROJECT OOP

# TOPIC: VISUALIZATION OF OPERATIONS ON TREE DATA STRUCTURES

Mã lớp: 143577

Nhóm: 21

## I. Assignment of members

1) Lê Quang Minh 20215088 (Leader): Algorithm and UI
   - Xây dựng cấu trúc project.
   - Xây dựng các lớp cây (Tree, GenericTree, BinarySearchTree,…)
   - Xây dựng giao diện màn hình làm việc với các cây.
   - Xây dựng controller cho BST và CompleteBalanceBST.
   - Review công việc mọi người làm.
2) Nguyễn Phúc Mạnh 20215087:
   - Không làm đủ phần việc được giao
3) Đoàn Quang Minh 20210606:
   - Làm slide
   - Không tham gia viết sourcecode.
4) Hoàng Nhật Minh 20210607:
   - Không làm phần việc được giao.

## II. Open source clarification

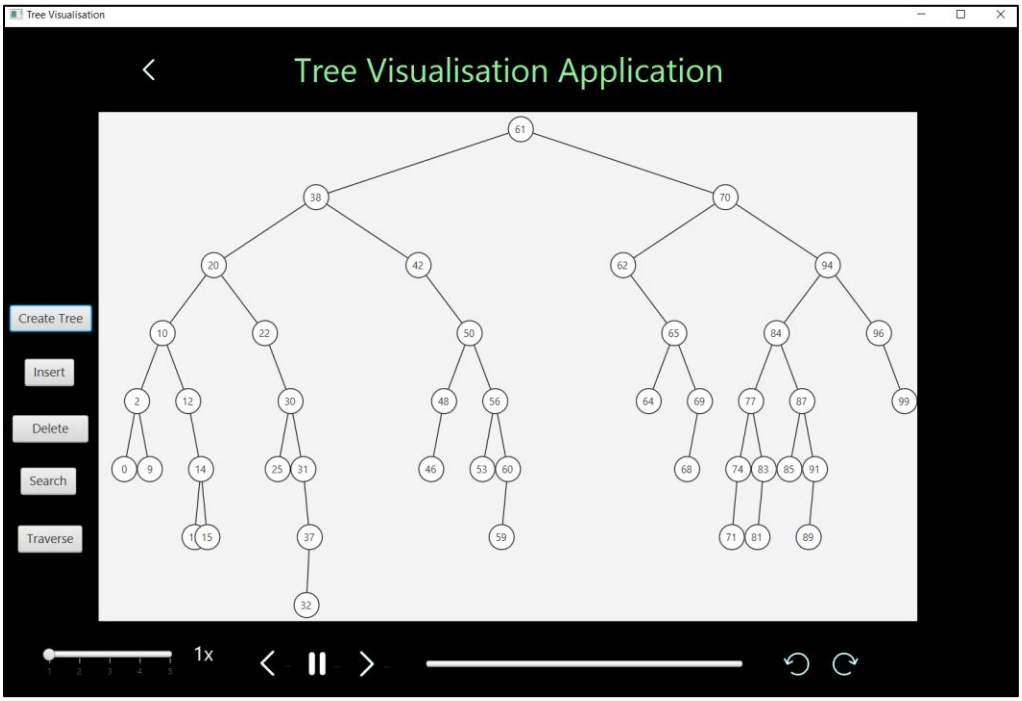
- Phương thức in cây ra console (mục đích chính để test các phương thức):
  - Binary Search Tree Printer: https://www.baeldung.com/java-print-binary-tree-diagram
  - Generic Tree Printer: Github Copilot Chat
- Phương thức biển đổi cây nhị phân tìm kiếm thành cây nhị phân tìm kiếm cân bằng hoàn chỉnh: https://stackoverflow.com/questions/52724898/a-complete-binary-search-tree-with-level-order-insert-in-java/52749727?fbclid=IwAR0CFzrigTpPCBsZP1fmWzLqr58oJercCy00ov7ldgTYaWBS-ovJd3GMmZ0#52749727

By Mr K.Nicholas: https://stackoverflow.com/users/3795036/k-nicholas

- Phương thức vẽ cây lên màn hình: Github Copilot Chat

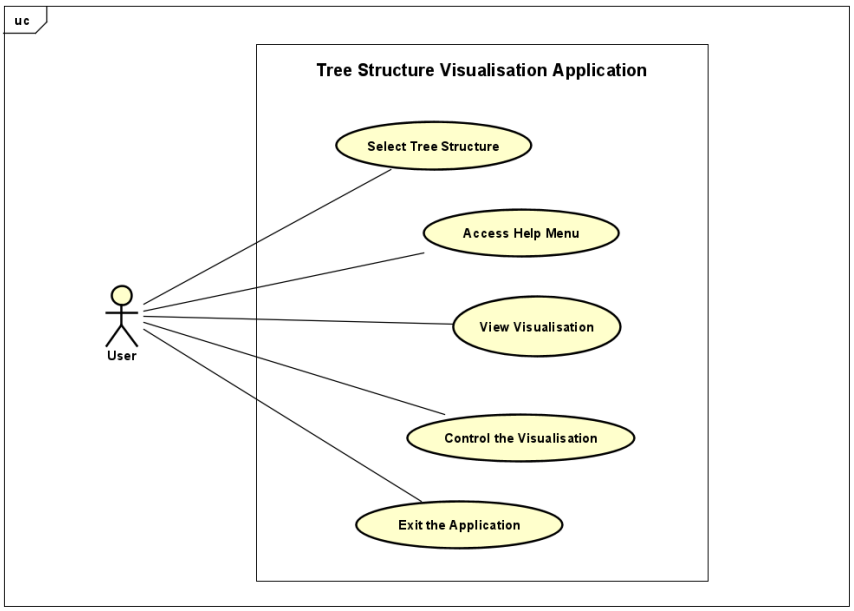
## III. Project description

Đây là 1 phần mềm sử dụng JavaFX để mô phỏng lại các thao tác có thể thực hiện được trên cấu trúc dữ liệu cây, bao gồm tạo cây, thêm, xóa, tìm nốt trong cây, duyệt cây theo DFS hoặc BFS.




*Giao diện thao tác với cây*

Phần mềm trên được xây dựng dựa trên sơ đồ use case sau đây:

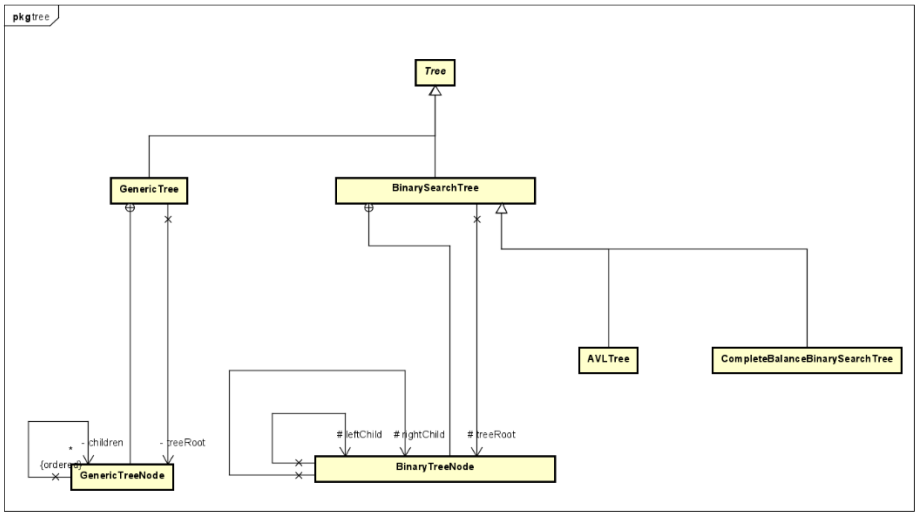



*Use case diagram*

Khi mở chương trình, người dùng có thể làm các việc sau:

1. **Select Tree Structure:** Lựa chọn 1 loại cây để thực hiện các thao tác lên đó.
2. **Access Help Menu:** Mở Help Menu để đọc hướng dẫn, giới thiệu về phần mềm.
3. **View Visualisation:** Thực hiện và theo dõi các thao tác như thêm, sửa, xóa, tìm kiếm, duyệt trên cây.
4. **Control the Visualisation:** Trong khi thao tác có thể điều chỉnh tốc độ, các bước thực hiện.
5. **Exit the Application:** Thoát ra khỏi phần mềm.

# IV. Design and Implementation details




*General class diagram*

## A. Các class trong project:

Có 7 class tương ứng với 1 class trừu tượng, 4 loại cây mà người dùng có thể thao tác lên và 2 loại nút:

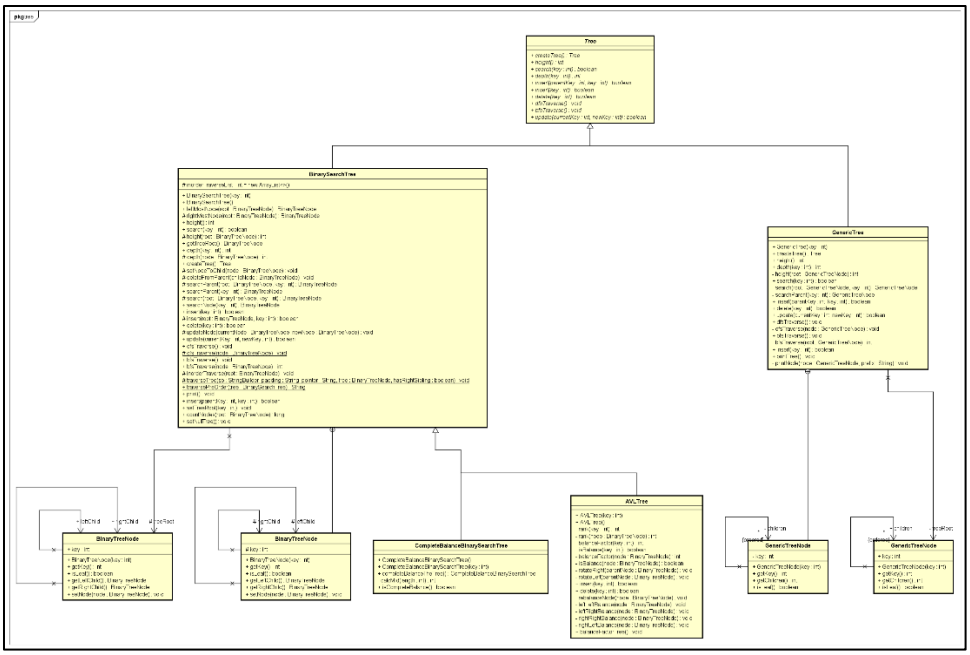
1. **Tree (Cây):** Là lớp trừu tượng và cha của cây GenericTree và BinarySearchTree.
2. **GenericTree (Cây thông thường):** Là cây mà mỗi nút có không giới hạn số nút con.
3. **BinarySearchTree (Cây nhị phân tìm kiếm):** Là cây thỏa mãn các điều kiện sau :
    - Mỗi nút có tối đa 2 con: con trái và con phải
    - Giá trị nút con trái nhỏ hơn nút cha và giá trị nút cha nhỏ hơn con phải
    - Cây con trái và cây con phải của mỗi nút cũng là 1 cây nhị phân tìm kiếm

    Cây nhị phân tìm kiếm là cha của cây AVL và cây CBBST.
4. **AVLTree (Cây Adelson-Velskii Landis):** Là cây nhị phân tìm kiếm tự cân bằng khi thêm – xóa nút.

Cây nhị phân cân bằng là cây có chiều cao cây con trái sai khác với cây con phải không quá 1, do đó độ cao cây không quá $\log_2(N)$ với N là số nút trong cây. Cây AVL có các phương pháp xoay để tự cân bằng lại bản thân.

5. **CompleteBalanceBinarySearchTree (Cây nhị phân tìm kiếm cân bằng hoàn chỉnh):** Là cây nhị phân tìm kiếm có các tính chất sau:
    - Tất cả các nút trong 1 hàng của nó được lấp đầy (có thể trừ hàng thấp nhất).
    - Hàng thấp nhất các nút được xếp từ trái sang.
    - Có các tính chất của cây nhị phân tìm kiếm.

    Nếu hàng cuối cũng được lấp đầy thì người ta gọi đó là Perfect Balance Binary Search Tree (Cây nhị phân tìm kiếm đầy đủ)



*Class Diagram (xem chi tiết trong file astah)*

Ta chia **Tree** (abstract class) ra làm 2 loại cây chính: **GenericTree** và **BinarySearchTree** do đặc tính của chúng khác nhau và tạo 2 lớp nút cho từng loại. Một Tree sẽ chứa một TreeNode làm gốc. Từ **BinarySearchTree** ta chia tiếp làm 2 loại cây: **AVLTree** và **CompleteBalanceBinarySearchTree** (**CBBST**). Project này không sử dụng Interface vì các phương thức của cây có thể kế thừa trực tiếp từ lớp tổ tiên Tree.

B. Một số các phương thức quan trọng:

Các phương thức, thao tác trên cây đa số không có gì phức tạp, các thao tác phức tạp thường sẽ nằm ở lớp **AVLTree** và **CBBST**.

- **BinarySearchTree**:
  - **leftMostNode(BinaryTreeNode root):** phương thức nhận đầu vào một nút, trả về nút trái nhất của nút đó.
  - **rightMostNode(BinaryTreeNode root):** phương thức nhận đầu vào một nút, trả về nút phải nhất của nút đó.
  - **setNodeToChild(BinaryTreeNode node):** phương thức nhận đầu vào một nút, kiểm tra xem nếu nút đó thiếu 1 trong 2 con thì thay nút đó bằng con còn lại.
  - **print()**: Vẽ cây lên màn hình console, sử dụng để test các phương thức.

  Tương tự với **printTree** của **GenericTree**.

- **AVLTree**:
  - **rank():** trả về chiều cao của nút + 1, dùng để đánh giá độ cân bằng của cây.
  - **balanceFactor():** trả về hiệu chiều cao giữa con trái và con phải, nếu là -1, 0, hoặc 1 thì cây cân bằng.
  - **isBalance():** nếu balanceFactor thuộc -1, 0 hoặc 1 thì cây cân bằng.
  - **rebalanceNode():** kiểm tra xem nút hiện tại đang bị lệch theo trường hợp nào: trái-trái, trái-phải, phải-trái hay phải-phải, từ đó sử dụng các hàm **rotateLeft()** và **rotateRight()** để xoay lại cây, giúp cây về trạng thái cân bằng.
- **CBBST**:
  - **completeBalanceTheTree():** thực hiện việc xếp lại cây bằng cách tính toán số nút, chiều cao, chọn nút qua hàm **calcMid** để chọn đc thứ tự nút add vào cây cho hợp lý để cho cây thành cân bằng hoàn chỉnh.
  - **isCompleteBalance():** kiểm tra xem cây hiện tại có phải cây cân bằng hoàn chỉnh không bằng cách duyệt bfs, nếu gặp 1 nút không phải nút đầy đủ (có cả con trái và con phải) thì các nút sau phải là nút lá, nếu có 1 nút không phải lá thì cây không cân bằng hoàn chỉnh.

C. Các kĩ thuật hướng đối tượng trong thiết kế:

- **Inheritance**:
  - GenericTree, BinarySearchTree kế thừa Tree.
  - AVLTree, CBBST kế thừa BinarySearchTree.
- **Aggregation**:
  - GenericTreeNode thuộc GenericTree.
  - BinaryTreeNode thuộc BinarySearchTree, AVLTree và CBBST.
- **Composition**: Không có.
- **Polymorphism**: Không có.
- **Abstraction**: Lớp trừu tượng Tree chứa các phương thức trừu tượng mà mỗi cây cần có.
- **Encapsulation**: Mỗi lớp cây đều có các thuộc tính private và sử dụng các phương thức để giao tiếp.

## V. Video Demo

Link Video