|  |  |
| --- | --- |
| Đại học khoa học tự nhiên | |
| BÁO CÁO ĐỒ ÁN | |
| BINARY SEARCH TREE FRAMEWORK | |
| **MÔN HỌC: MẪU THIẾT KẾ HƯỚNG ĐỐI TƯỢNG VÀ ỨNG DỤNG** | |
| 1212148 NGUYỄN ĐẠT NHẬT HUY  1212230 NGUYỄN HẢI MIÊN  1212443 PHẠM MINH TRÍ  1212090 VÕ VĂN THÀNH ĐÔ  1212496 LÊ VĂN TÙNG |
| **6/8/2016** | |

|  |
| --- |
|  |

Contents

[A. SƠ ĐỒ LỚP 2](#_Toc453164151)

[B. CÁC MẪU THIẾT KẾ ĐƯỢC ÁP DỤNG 4](#_Toc453164152)

[1. Composite 4](#_Toc453164153)

[2. Singleton 5](#_Toc453164154)

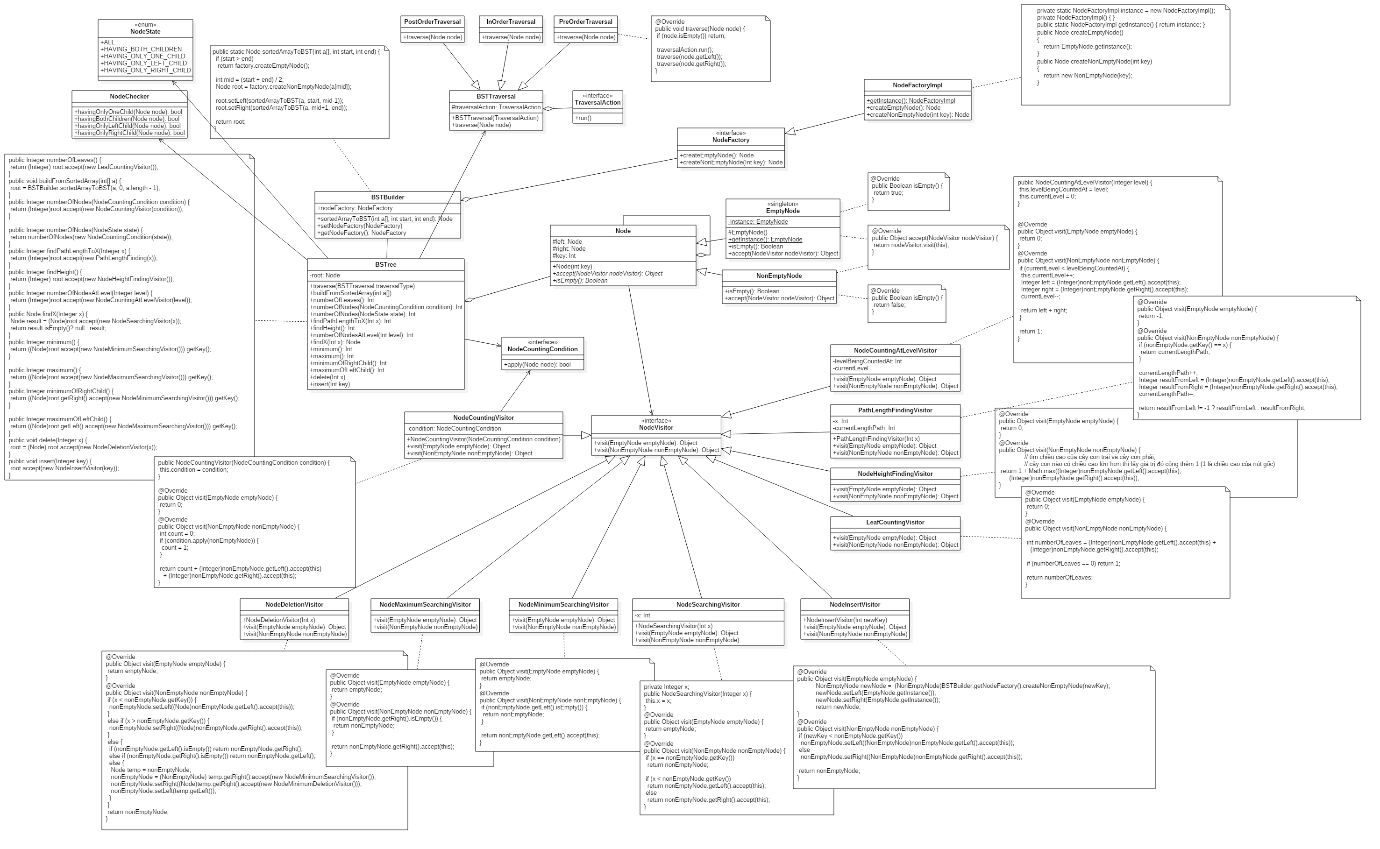
[3. Strategy 6](#_Toc453164155)

[4. Visitor 6](#_Toc453164156)

[5. State 8](#_Toc453164157)

[6. Abstract Factory 9](#_Toc453164158)

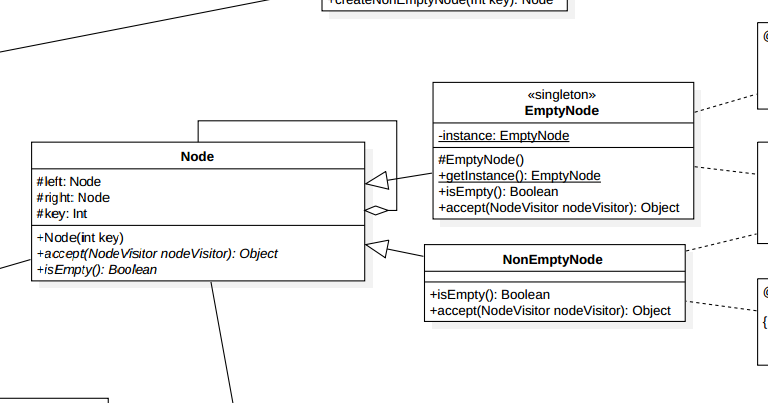
[C. BẢNG DANH SÁCH TÍNH NĂNG 12](#_Toc453164159)

1. SƠ ĐỒ LỚP

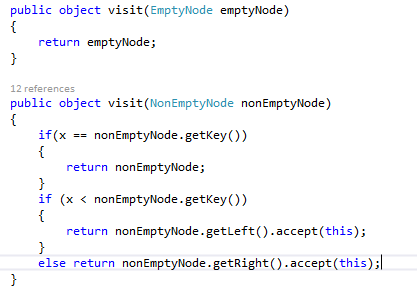
* **Lớp BSTree**: là đối tượng cây nhị phân tìm kiếm có 1 thuộc tính là 1 đối tượng Node là root(nút gốc), và các hàm xử lí dựa vào root.
* **Lớp Node**: là đối tượng nút trong cây nhị phân, là lớp abstract định nghĩa một số phương thức cho lớp con kế thừa.
* **Lớp EmptyNode**: kế thừa lớp Node, là nút rỗng. Khi 1 nút có một con thì con còn lại là nút rỗng.
* **Lớp NonEmptyNode**: kế thứa lớp Node, chứa 2 đối tượng Node khác.
* **Lớp NodeFactory**: interface sử dụng cho mẫu AbstractFactory, dùng để khởi tạo Node (EmptyNode và NonEmptyNode)
* **Lớp NodeFactoryImpl**: lớp thể hiện của interface NodeFactory, thể hiện một factory cụ thể sản xuất ra một loại Node cụ thể.
* **Lớp BSTTraversal**: lớp interface định nghĩa phương thức duyệt cây cho các lớp con kế thừa.
* **Lớp TraversalAction:** interface định nghĩa method run(), method này sẽ chạy khi visit từng node trong lúc duyệt cây.
* **Lớp PostOrderTraversal, InOrderTraversal, PreOrderTraversal**: các lớp con của lớp BSTTraversal viết lại phương thức duyệt cây đại diện cho 3 cách duyệt khác nhau.
* **Lớp BSTBuilder**: lớp khởi tạo cây nhị phân tìm kiếm thông qua mảng được truyền vào.
* **Lớp NodeVisitor**: interface định nghĩa lại phương thức duyệt qua từng nút(visit).
* **Lớp NodeDeletionVisitor**: viết lại phương thức duyệt(visit) kế thừa của lớp NodeVisitor, để duyệt qua từng nút tìm nút cần xóa và xóa.
* **Lớp NodeMinimumSearchingVisitor**: viết lại phương thức duyệt(visit) kế thừa của lớp NodeVisitor và trả về Node có giá trị nhỏ nhất.
* **Lớp NodeMaximumSearchingVisitor**: viết lại phương thức duyệt(visit) kế thừa của lớp NodeVisitor và trả về Node có giá trị lớn nhất.
* **Lớp NodeSearchingVisitor**: viết lại phương thức duyệt(visit) kế thừa của lớp NodeVisitor, trả về Node có giá trị bằng với giá trị thuộc tính x của lớp ta sẽ truyền vào.
* **Lớp NodeInsertVisitor**: viết lại phương thức duyệt(visit) kế thừa của lớp NodeVisitor và trả về Node mới thêm vào.
* **Lớp NodeCoutingAtLevel**: viết lại phương thức duyệt(visit) kế thừa của lớp NodeVisitor để đếm số Node tại mỗi mức của cây nhị phân.
* **Lớp PathLenghtFindingVisitor**: viết lại phương thức duyệt(visit) kế thừa của lớp NodeVisitor và trả về độ dài đường đi từ Node root đến Node có giá trị x.
* **Lớp NodeHightFindingVisitor**: viết lại phương thức duyệt(visit) kế thừa của lớp NodeVisitor và trả về độ cao của cây.
* **Lớp LeafCountingVisitor**: viết lại phương thức duyệt(visit) kế thừa của lớp NodeVisitor và trả về số lượng Node lá.
* **Lớp NodeCoutingVisitor**: viết lại phương thức duyệt(visit) kế thừa của lớp NodeVisitor và trả về số lượng Node theo yêu cầu(đếm số Node có 2 con, 1 con….)
* **Lớp NodeCountingCondition**: lớp kiểm tra điều kiện được truyền vào(node 1 con, node 2 con, node có con trái….) trả về giá trị true hoặc false.
* **Lớp NodeState**: lớp enum định nghĩa các trạng thái của node (có 1 con, 2 con, có con trái, con phải…).
* **Lớp NodeChecker**: lớp sử dụng các phương thức static public để kiểm tra các trạng thái của Node.

# CÁC MẪU THIẾT KẾ ĐƯỢC ÁP DỤNG

## Composite

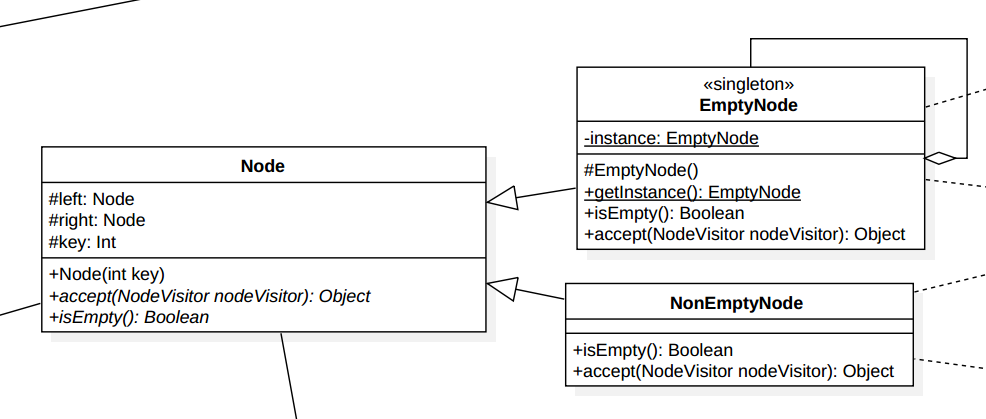


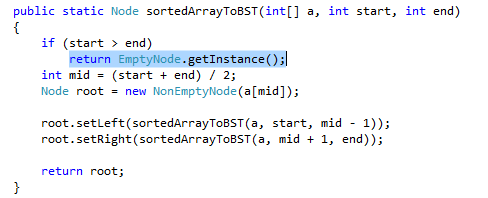
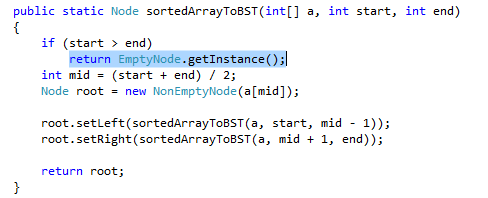
**NodeSearchingVisitor.cs**



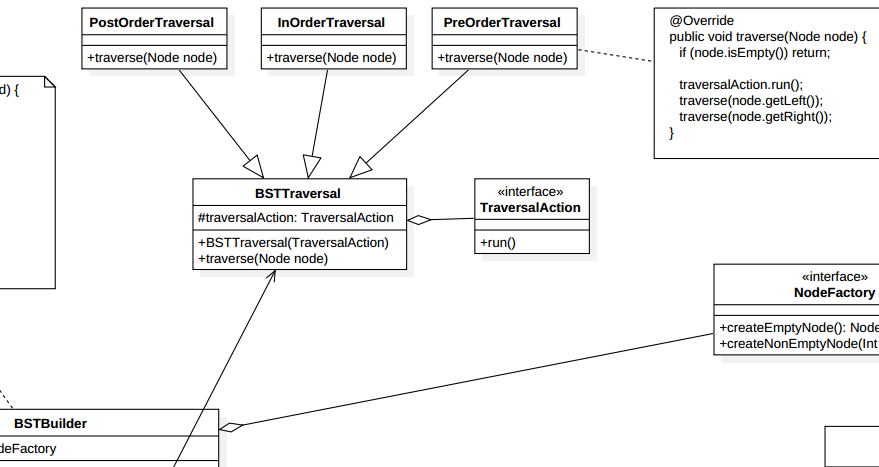
* Ở đây lớp component của mẫu composite là class Node. Ở đây lớp composite là lớp NonEmptyNode kế thừa lớp Node và có thể chứa 2 đối tượng Node khác, lớp EmptyNode thể hiện node rỗng, còn đối với nút lá của cây nhị phân thì vẫn là NonEmptyNode và chứa 2 đối tượng EmptyNode.

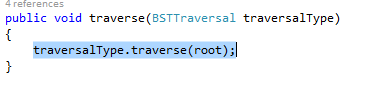
## Singleton

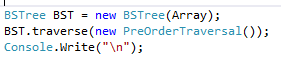


* Sử dụng khi khởi tạo cây nhị phân từ mảng đã sắp xếp
* Sử dụng khi thêm nốt mới, lúc này nốt mới là nốt lá nên sẽ không có cây con
* Vì các đối tượng của lơp EmptyNode giống nhau về mọi mặt nên để tiết kiệm bộ nhớ nên ở đây chỉ cần 1 thể hiện duy nhất => áp dụng mẫu Singleton cho lớp EmptyNode

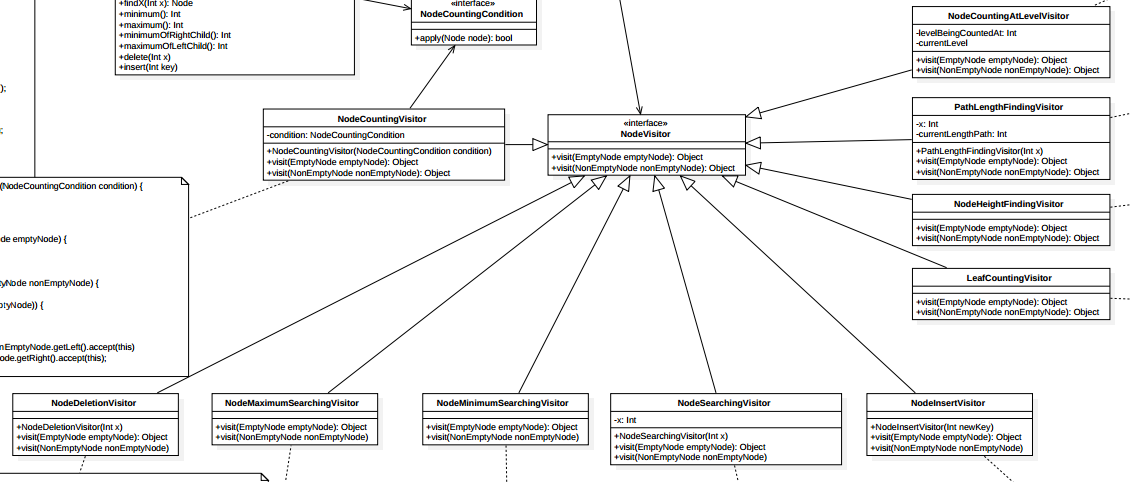
## Strategy



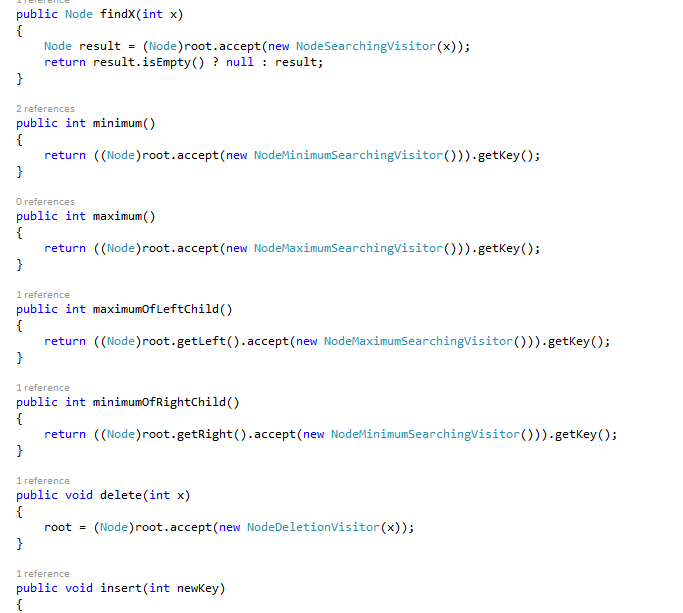
* Trong class BSTree sẽ sử dụng hàm traverse để truyền 1 đối tượng “cách duyệt” vào.



* Code sử dụng mẫu
* Có 3 cách thức duyệt cây là PreOrder, InOrder và PostOrder, khi sử dụng cây, sử dụng mẫu Strategy sẽ giúp chuyển đổi qua lại giữa các phương thức duyệt cây một cách dễ dàng.

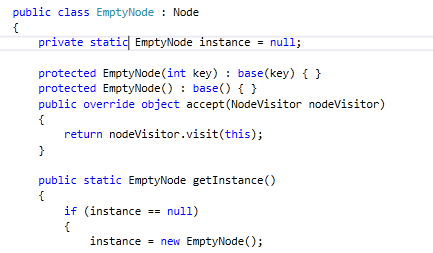
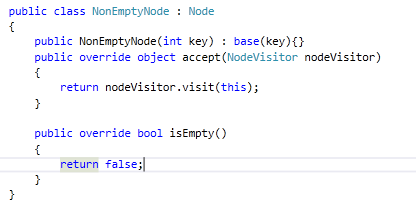
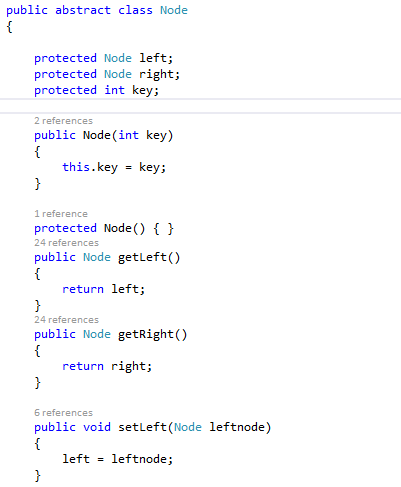
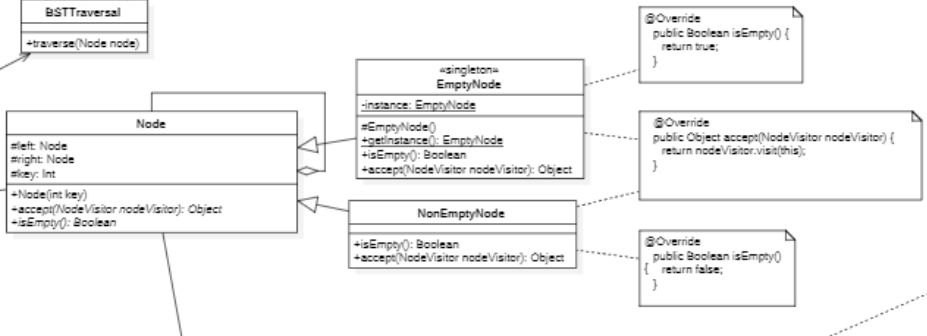
1. Visitor 

* **BSTree.cs**



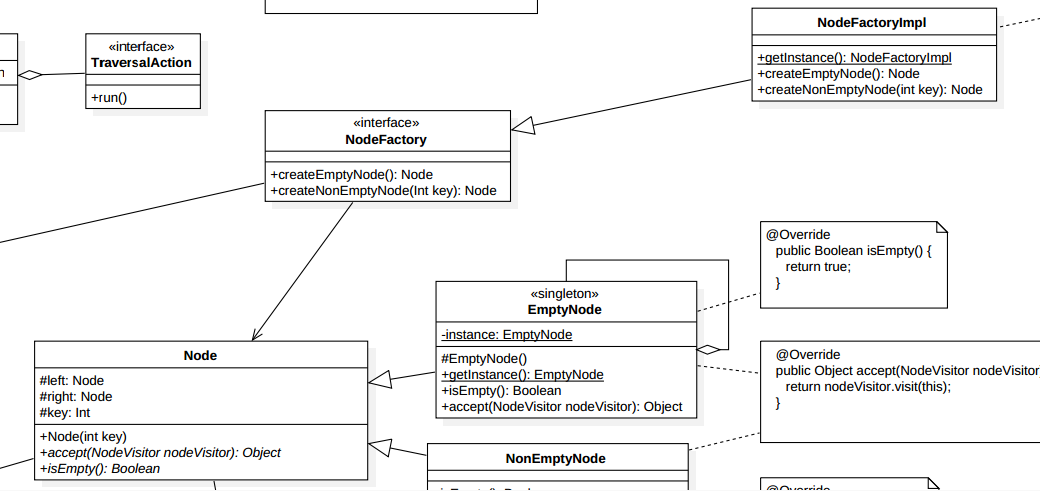
* Vì framework có sử dụng mẫu State cho Node (EmptyNode và NonEmptyNode) nên mẫu Visitor được áp dụng để lược bỏ các đoạn code kiểm tra xem Node đang xử lí có phải Node rỗng hay không. Ngoài ra, việc sử dụng mẫu này giúp tách rời thuật toán ra khỏi cấu trúc của framework, khiến cho việc mở rộng và sửa đổi trở nên dễ dàng hơn rất nhiều. Cụ thể là trong ứng dụng này, cứ mỗi chức năng tương ứng thì sẽ có một visitor tương ứng được tạo ra.

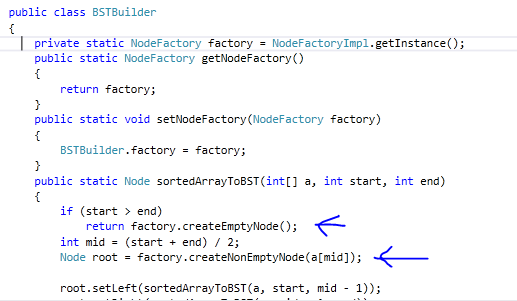
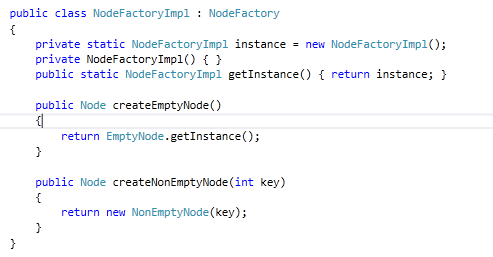
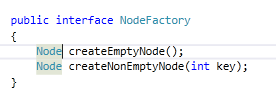
## State

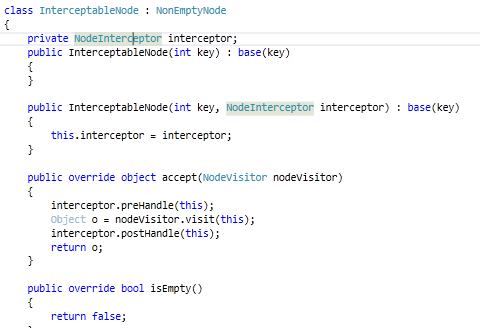
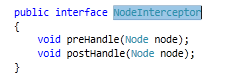
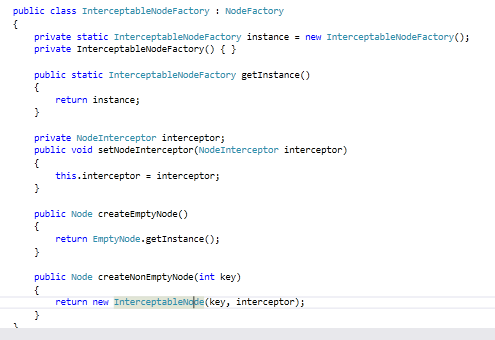


* Vì Node sẽ có 2 trạng thái: rỗng hoặc không rỗng, thay vì node rỗng sẽ được quy ước là NULL thì ở đây, nhóm em áp dụng mẫu State để chia Node thành 2 trạng thái là EmptyNode và NonEmptyNode, điều này giúp giảm thiểu các lỗi khi xử lí với đối tượng NULL và dễ dàng chuyển đổi qua lại giữa EmptyNode sang NonEmptyNode (khi xóa node) hay NonEmptyNode sang EmptyNode (khi thêm node).

## Abstract Factory





* Áp dụng mẫu AbstractFactory cho việc quản lí khởi tạo các node vì trong tương lai, Node có thể sẽ được mở rộng ra thành nhiều loại. Cụ thể hơn, trong ứng dụng của nhóm, tụi em muốn can thiệp vào hàm Node#accept(NodeVisitor) để in ra màn hình giá trị của Node trước khi chạy thuật toán. Giải pháp là tụi em sẽ tạo ra một loại Node khác kế thừa lại Node cũ và override lại hàm accept(), gọi là **InterceptableNode**, loại Node này sẽ chứa một reference của interface **NodeInterceptor** có 2 method là preHandle() và postHandle() (xem code bên dưới)
* Để có thể đưa loại **InterceptableNode** vào sử dụng, tụi em tạo một loại factory khác là **InterceptableNodeFactory**, class này có 2 nhiệm vụ: một là khởi tạo Node để đưa vào sử dụng trong framework, hai là “bơm” implementation của **NodeInterceptor** (do người dùng định nghĩa)vào trong từng **InterceptableNode** 
* **InterceptableNodeFactory#setNodeInterceptor(NodeInterceptor)** cho phép client “bơm” implementation của **NodeInterceptor** vào
* **BSTree#setNodeFactory(NodeFactory)** cho phép người dùng “bơm” thể hiện của **NodeFactory** vào, cụ thể ở đây là **InterceptableNodeFactory**
* Kết quả, ta có thể dễ dàng intercept từng node mà không phải thay đổi source code hiện có.

# BẢNG DANH SÁCH TÍNH NĂNG

|  |  |  |
| --- | --- | --- |
| STT | CHỨC NĂNG | MỨC ĐỘ HOÀN THÀNH |
| 1 | Duyệt cây theo thứ tự Left-Node-Right. | 100% |
| 2 | Duyệt cây theo thứ tự Node- Left-Right. | 100% |
| 3 | Duyệt cây theo thứ tự Left-Right-Node. | 100% |
| 4 | Đếm số node lá của cây | 100% |
| 5 | Đếm số node có một cây con | 100% |
| 6 | Đếm số node có 2 cây con | 100% |
| 7 | Đếm số node có cây con trái | 100% |
| 8 | Đếm số node có cây con phải | 100% |
| 9 | Xác đinh chiều cao của cây | 100% |
| 10 | Xác định số node trên cây | 100% |
| 11 | Xác định số node trên từng mức của cây | 100% |
| 12 | Xác định độ dài đường đi từ node gốc đến node có giá trị là x trên cây | 100% |
| 13 | Tìm node có giá trị x trên cây | 100% |
| 14 | Tìm node có giá trị nhỏ nhất trên cây | 100% |
| 15 | Tìm node có giá trị lớn nhất trên cây | 100% |
| 16 | Tìm nút có giá trị nhỏ nhất của cây con bên phải | 100% |
| 17 | Tìm giá trị lớn nhất của cây con phía bên trái | 100% |
| 18 | Xóa một nút khỏi cây | 100% |
| 19 | Xây dựng ứng dụng minh họa trên môi trường web | 100% |
| 20 | Vẽ được giao diện cây trên màn hình | 100% |
| 21 | Tạo ra được hiệu ứng với các thao tác trên cây | 90% |