**SỬ DỤNG CẤU TRÚC CÂY NHỊ PHÂN- BINARY TREES**

***Nội dung***

* Một số định nghĩa và tính chất liên quan đến cây nhị phân cần nhớ
* Mô tả một phần tử trên cây và mô tả một cây nhị phân
* Khi nào dùng cây nhị phân
* Tại sao lại là cây nhị phân
* Tóm tắt về phép duyệt cây nhị phân
* Minh hoạ cây không thứ tự
* Minh hoạ cây BST-Binary Search Tree

1. **Một số định nghĩa và tính chất liên quan đến cây nhị phân cần nhớ**

**Cây nhị phân:** Tập nút trong đó có 1 nút gốc, một nút có tối đa 2 nút con.

**Các loại nút:** Nút gốc (root), nút cha (father), nút con (child), nút trên (ancestor), nút dưới (descendant), nút lá (leaf, terminal), nút trung gian/ nút trong (internal)

**Đường đi- path**: Đường duy nhất từ nút gốc đến nút đang được xét.

**Mức** (level)của một nút, **chiều cao** (height) của cây.

**Cây có thứ tự- ordered tree**: cây chứa tập trị thoả mãn một tiêu chuẩn định trước.

**Cây** **BST**: Cây nhị phân có thứ tự với điều kiện: trị nút này sẽ lớn trị nút con trái và nhỏ hơn trị của nút con phải. Chính nội dung các tác vụ thêm/xoá/ sửa của cây giúp xác định cây này cây thông dụng/ cây có thứ tự hay không (BST là một cây có thứ tự).

**Một số quan hệ trên cây nhị phân**.

* Số cạnh = số nút – 1
* Số nút là >= số
* Số nút tại mức L n(L) <= 2L
* Số nút lá của cây có chiều cao H: leaves <= 2H
* Số nút của một cây nhị phân đầy đủ với i nút trung gian: n= 2i+1
* Só nút lá c ủa một cây nhị phân với i nút trung gian: leaves = i+1

1. **Mô tả một phần tử trên cây và mô tả một cây nhị phân**

**class BinTreeNode <T>** { **class BinTree <T>** {

T **info;** BinTreeNode<T> **root ;**

Node<T> **left;** …….

Node<T> **right;** …….

…….. …….

} }

**Nhận xét**: Info trong nút nên có các đặc điểm:

* Có khả năng so sánh để có thể mở rộng thành nút được dùng cho các tình huống cần sắp xếp ( nút trong cây BST chẳng hạn)
* Có trị nào đó nhằm phân biệt duy nhất (key) để giúp quá trình tìm kiếm với điều kiện là có tối đa **một** kết quả.

1. **Khi nào dùng cây nhị phân**

* Dữ liệu có phân cấp tự nhiên như: danh sách gia phả, tập trạng thái trong các trò chơi.
* Dữ liệu có phân cấp dạng cấu trúc: Dữ liệu được trình bầy trên cửa số trình duyệt ( cây DOM, Document Object Model)….

1. **Tại sao lại là cây nhị phân**

Dĩ nhiên chúng ta có thể dùng cây n-phân. Nếu dùng cây n-phân, tại mỗi nút chúng ta phải lưu trữ một mảng **n** tham khảo đến n nút con nhưng không phải lúc nào cũng có đủ n con 🡪 phí bộ nhớ.

Chúng ta hoàn toàn có thể dùng cây nhị phân để biểu diễn cây n phân với ý nghĩa của các tham chiếu như sau:

* **Tham chiếu left** chỉ đến nút con đầu
* **Tham chiếu right** chỉ đến nút anh em cùng cha.
* Hai tham chiếu này mang ý nghĩa khác với hai tham chiếu được đề cập trong cây nhị phân ban đầu 🡺 Các thuật toán trên cây phải được viết lại cho phù hợp.

1. **Tóm tắt về hép duyệt cây- Traversing**

**Duyệt cây:** Quá trình viếng thăm từng nút trong một cây.

**Cơ chế duyệt cây:**

Cả cây nhị phân chỉ được quản lý bằng nút gốc, mọi con đường đều đi từ nút gốc và theo nguyên tắc biết nút cha mới biết được nút con nên mọi cách duyệt đều có nguyên tắc chung là:

* Tại một thời điểm chỉ viếng thăm một nút.
* Trật tự các nút sẽ được viếng thăm phải có cách đi đến bằng cách lưu trữ nút sẽ đi đến vào stack hệ thống( cơ chế hàm đệ quy) hoặc stack/ hoặc queue tự quản lý.
* **Các cơ chế thống dụng**

**Duyệt theo mức** 🡪 Duyệt theo chiều rộng/ chiều ngang của cây ( Breadth-first traversal): Khi duyệt 1 nút thì cất các nút con vào queue.

**Duyệt theo nhánh** 🡪 Duyệt theo độ sâu, Depth-first traversal 🡪 Dùng kỹ thuật đệ quy và có 6 thứ tự duyệt (V: visit, viếng thăm nút hiện hành) VLR, VRL, LVR, RVL , LRV, RLV – Dùng stack hệ thống (hàm đệ quy) hoặc kỹ thuật lặp có sự hỗ trợ của stack tự tạo.

**Biến thể:** Duyêt có xâu kết cả một nhánh ( Threads Tree), Morris traversal

**Chính tác vụ duyệt cây là cơ bản nhất để giải hầu hết các tác vụ trên cây dù cây đó là cây có hoặc không có thứ tự**

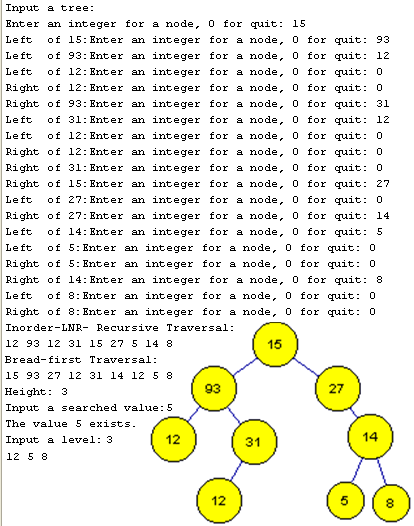
( chi tiết các phép duyệt này đã được đề cập chi tiết trong sách giáo khoa)

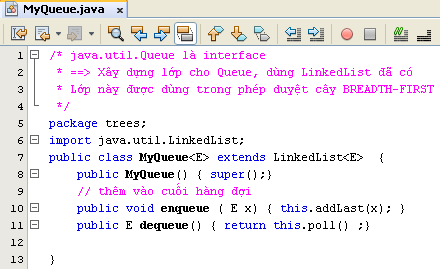
**Cần chú ý rằng các phép duyệt cây là chung cho mọi cây kể cả cây BSY. BST là một cây đặc biệt nhằm giải quyết tốt các bài toán quản lý dữ liệu dạng cây có yêu cầu về tìm kiếm sao cho hiệu quả (O(logn)) nên các tác vụ chèn/ xoá/ tìm kiếm được hiệu chỉnh phù hợp với thứ tự dữ liệu đã được ấn định trước.**

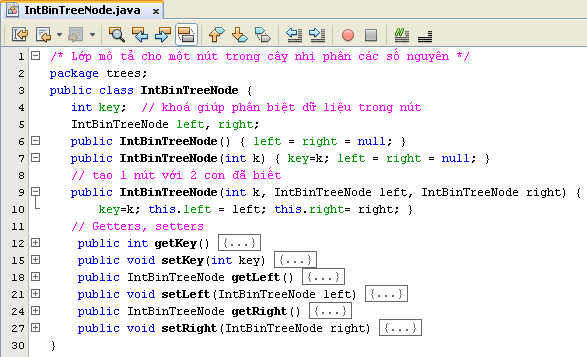
1. **Minh hoạ về cây không thứ tự**

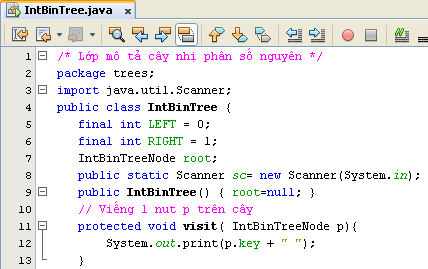
**Thí dụ**

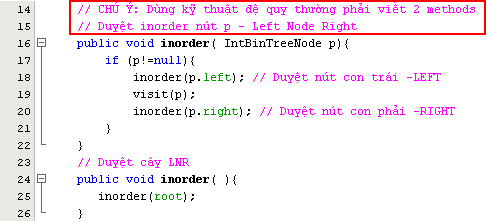
Thí dụ sau minh hoạ cách dùng pháp duyệt cây để nhập vào 1 cây số nguyên sau đó thực thi các thao tác như xuất cây (theo bề sâu và bề rộng), tìm chiều cao của cây, tìm kiếm trên cây, in ra các nút trong một mức. Kết quả của một lần chạy như hình sau:

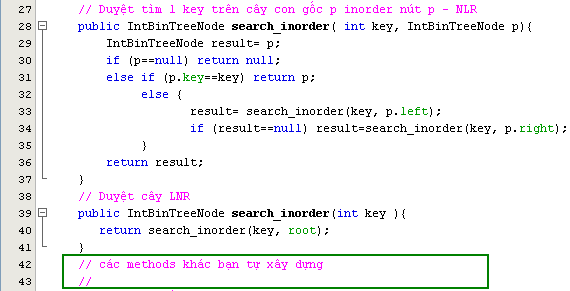












*Chú ý khi nhập dữ liệu vào cây*

Ngôn ngữ C++ hỗ trợ tham số dạng tham chiếu nên chúng ta có cơ hội làm biến đổi đối số ngoài đã truyền cho hàm. Do vậy, chúng ta có thể truyền **đối số null** từ ngoài hàm để rồi địa chỉ được cấp phát động bên trong hàm sẽ cập nhật vào đối số null bên ngoài hàm. Trong khi đó, Java chỉ hỗ trợ truyền tham số cho hàm dạng tham trị nên khi truyền tham số cho hàm là một tham khảo mang trị **null** thì trị **null này được chép vào tham số.** Việc cấp phát động bên trong hàm không truyền lại giá trị địa chỉ cho đối số ngoài 🡪 Đối số bên ngoài vẫn là **null.** Hình sau minh hoạ việc không thay đổi đối số dạng tham khảo trong Java.

|  |  |
| --- | --- |
| Xét phương thức **input**(…) trong Java  **void input( Student obj) {**  **obj= new Student (“Minh”, 7);**  **….**  **}** | Và dùng hàm này trong hành vi **m()** như sau:  **void m() {**  **Student st1= null; //1**  **input (st1); //2**  **…**  **}** |

Bộ nhớ stack khi phát biều phương thức **m()** thực thi như sau:

**st1 = null**

Stack cho **m()**

**st1 = null**

**obj=7000**

**obj=null**

Stack cho **input ()**

7000

**Minh, 7**

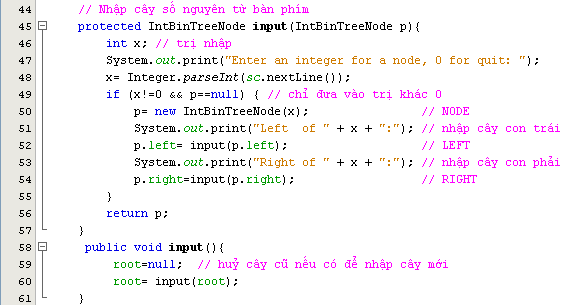
**Trước phát biểu obj= new Student(...) Sau phát biểu obj= new Student(...)**

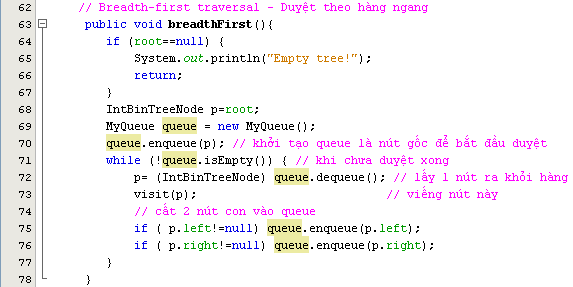
Như vậy, sau khi hành vi **input(…)** thực thi xong, tham khảo **st1** vẫn mang trị **null**.

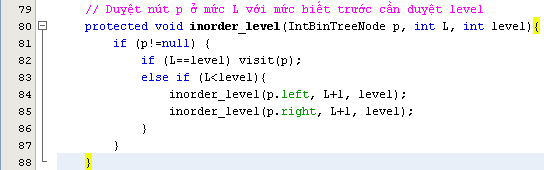
**Rút kinh nghiệm**: Hai cách để nhập

*Cách 1*: Cấp phát bộ nhớ trước khi nhập trị vào các đối tượng. Nếu việc nhập trị này gây ra dữ liệu không được chấp nhận thì huỷ đối tượng này đi sau khi nhập.

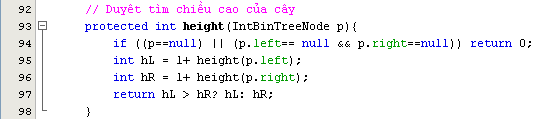
*Cách 2*: Hàm nhập trả về trị tham khảo đến đối tượng mới được cấp phát để có thể gán lại cho đối số ngoài( tham khảo hai hành vi **input(…)** dưới đây.

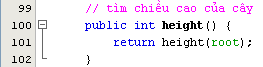


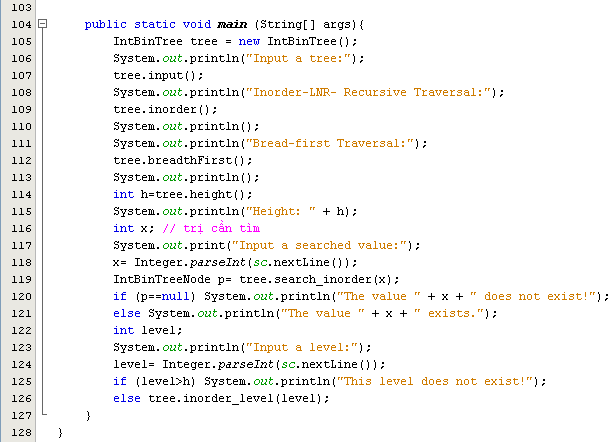








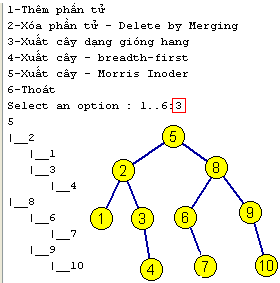
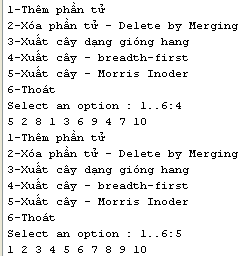


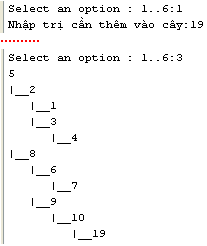
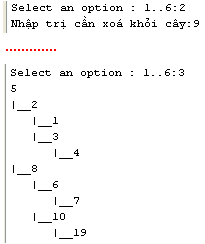


1. **Minh hoạ về cây BST các số nguyên**

Chương trình sau minh hoạ cách sử dụng một cây BST các số nguyên. Ban đầu, khởi tạo một cây BST cân bằng mang các trị 1..10. Sau đó cho phép user thêm/xoá phần tử, xem cây.

**Kết quả một lần chạy chương trình**

**Các lớp cần xây dựng**

Lớp Menu

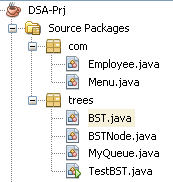
**Lớp Menu**: Đã xây dựng từ bài trước

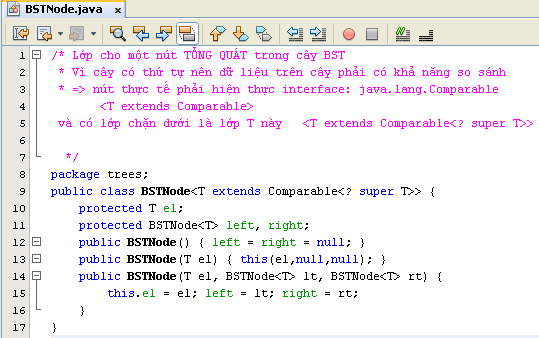
**Lớp BSTNode**: Mô tả chung cho 1 nút trên cây BST. Vì data trong nút có thứ tự nên buộc lớp con sau này phải implements interface Comparable.

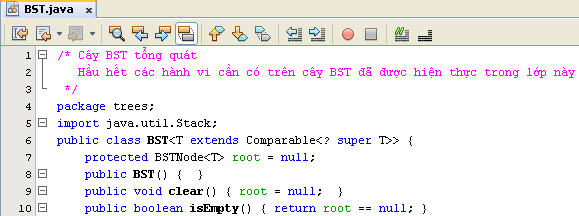
**Lớp BST**: Mô tả cho 1 cây BST có dữ liệu BSTNode

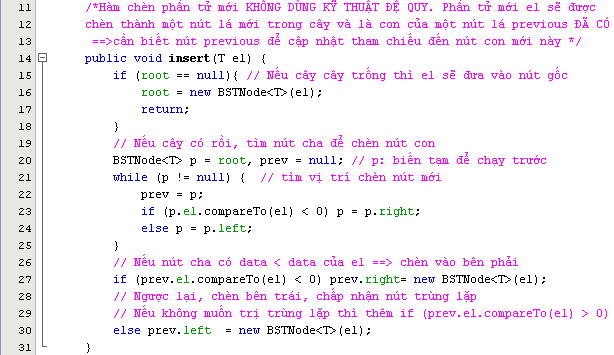
**Lớp MyQueue** mô tả 1 hàng đợi đã có ở demo trước.

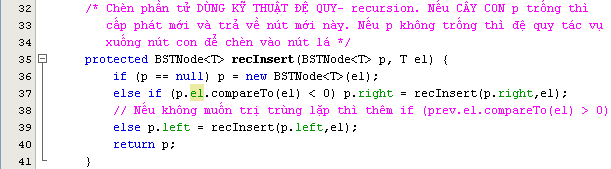
**Lớp TestBST**: Cây có dữ liệu Integer. Lớp Integer đã implements interface Comparable rồi.

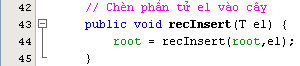


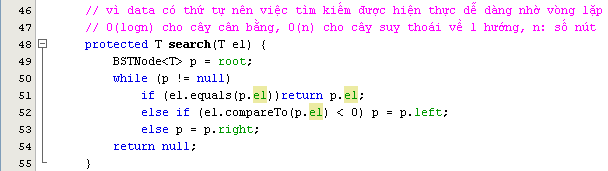




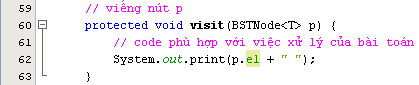


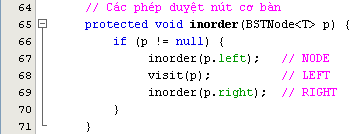


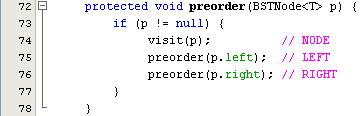


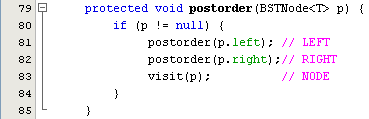


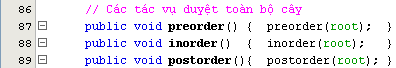


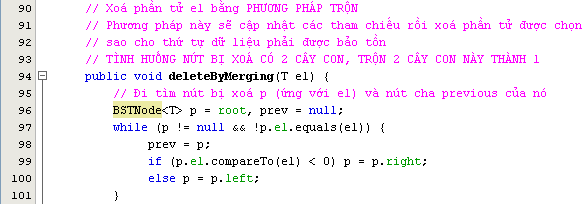


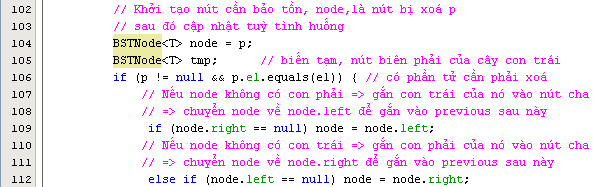


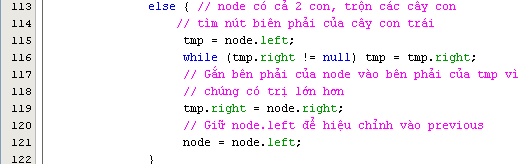


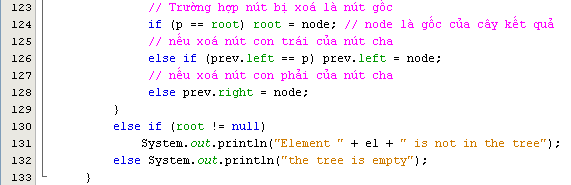


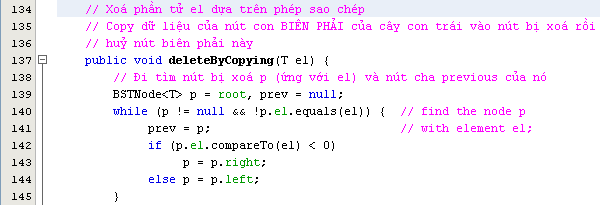


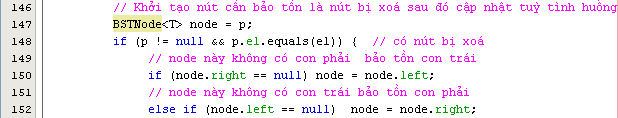








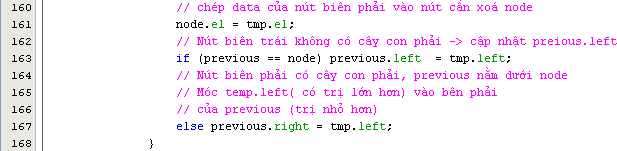


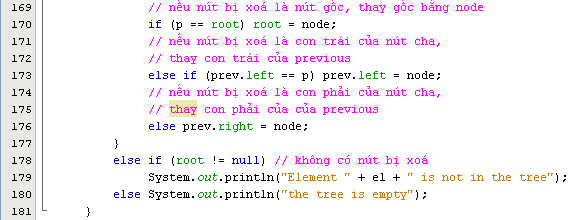


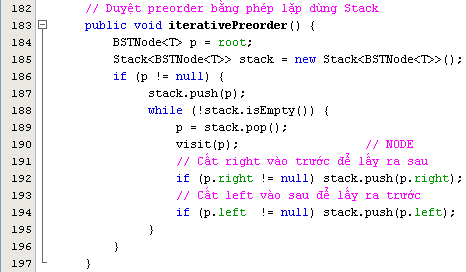


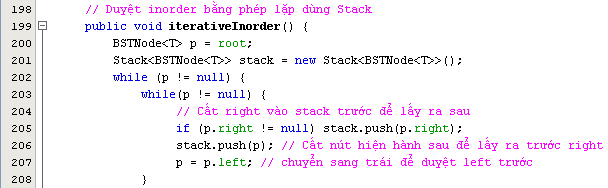


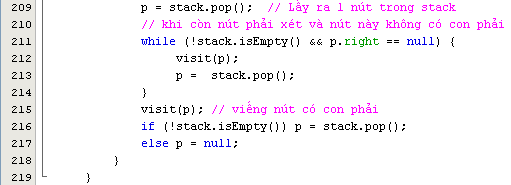


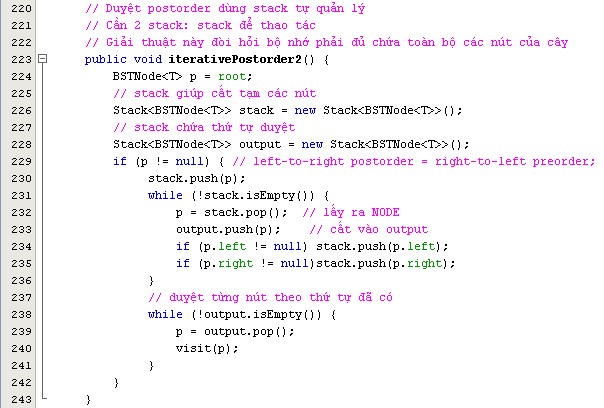


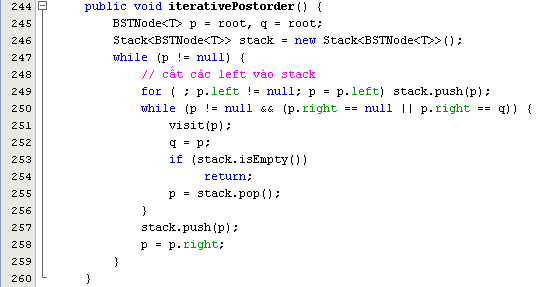


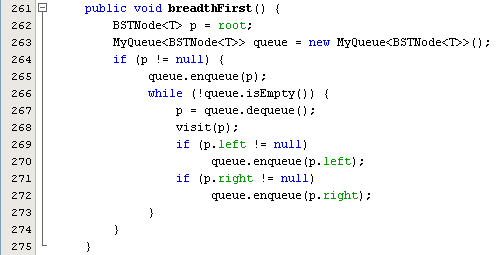


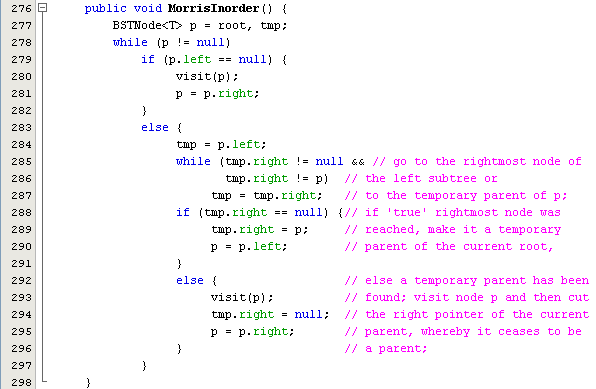


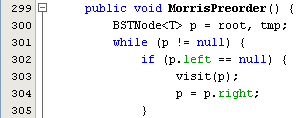


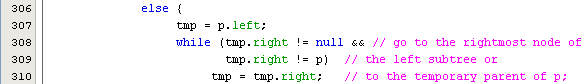


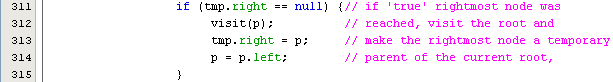


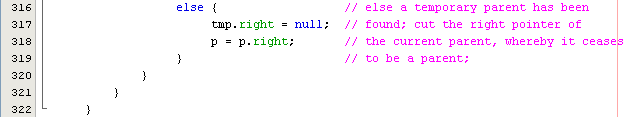


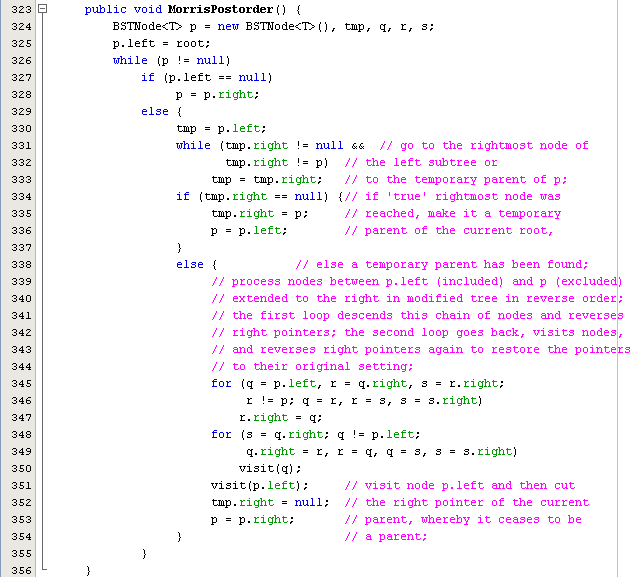


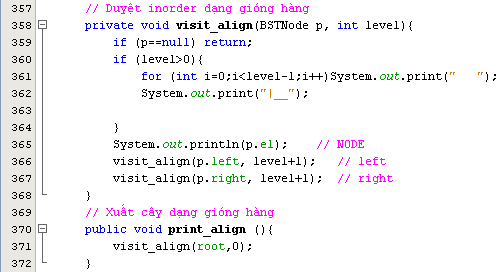


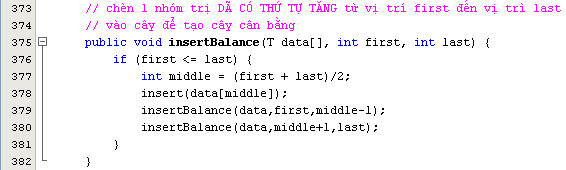


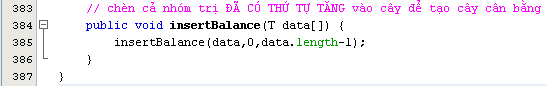


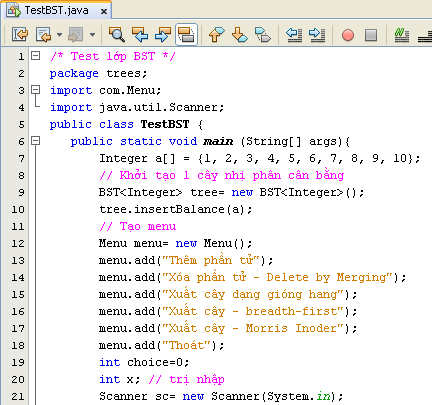


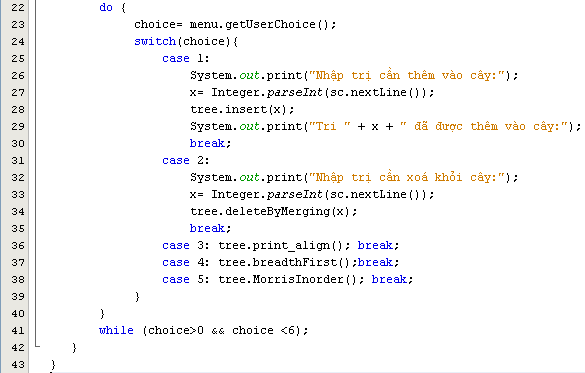












**Bài tập**

Sử dụng cây BST dựa trên mã sinh viên (code), viết chương trình quản lý danh sách học sinh <code, name, mark> có các tác vụ: thêm/xoá/sửa điểm học sinh.

***Gợi ý:*** Xây dựng lớp Student có implements interface java.lang.Comparable, override hai hành vi:

public int compareTo(Object st) {

return this.code.compareTo(((Student)st).code);

}

public String toString(){

return code + “,” + name + “,” + mark;

}

***Hành vi compareTo(…) được dùng để so sánh mã khi thêm sinh viên vào cây cũng như tìm kiếm sinh viêndựa trên mã học sinh.***

***Hành vi toString() được dùng khi xuất học sinh.***