

# BỘ GIÁO DỤC VÀ ĐÀO TẠO TRƯỜNG ĐẠI HỌC ĐÔNG Á

##### -----🙞🙜🕮🙞🙜-----

FINAL REPORT

CẤU TRÚC DỮ LIỆU VÀ GIẢI THUẬT

ĐỀ BÀI:

# BÀI TẬP LỚN MÔN CẤU TRÚC DỮ LIỆU

**GVHD: Lý Quỳnh Trân**

**SVTH: Lê Minh Chiến**

**Lớp: IT22G (năm 2)**

Đà Nẵng, 22/11/2023 1

**Chương 1: Mảng động**

**Câu 0:** Viết chương trình dung mảng động để lưu danh sách các sinh viên có sử dụng Menu gồm các thao tác sau:

* Thêm sinh viên
* Hiển thị danh sách sinh viên
* Xóa sinh viên
* Update sinh viên

**CLASS STUDENT**

public class Student {

    private int id;

    private String name;

    private double mark;

    public Student(int id, String name, double mark) {

*this*.id = id;

*this*.name = name;

*this*.mark = mark;

    }

    public int getId() {

*return* id;

    }

    public void setId(int id) {

*this*.id = id;

    }

    public String getName() {

*return* name;

    }

    public void setName(String name) {

*this*.name = name;

    }

    public double getMark() {

*return* mark;

    }

    public void setMark(double mark) {

*this*.mark = mark;

    }

    @Override

    public String toString() {

*return* "Student [id=" + id + ", name=" + name + ", mark=" + mark + "]";

    }

}

**CLASS STUDENTLIST**

import java.util.ArrayList;

import java.util.List;

import java.util.Scanner;

public class StudentList {

    private List<Student> students = *new* ArrayList<>();

    public static Scanner scanner = *new* Scanner(System.in);

    public StudentList() {

    }

    Student student1 = *new* Student(1, "A", 9.3);

    Student student2 = *new* Student(2, "B", 8.0);

    Student student3 = *new* Student(3, "C", 6.7);

    Student student4 = *new* Student(4, "D", 7.2);

    public void create(){

        students.add(student1);

        students.add(student2);

        students.add(student3);

        students.add(student4);

        System.out.println("Create success!");

    }

    public void display(){

        System.out.println("List of students:");

*for* (Student student *:* students) {

            System.out.println(student);

        }

    }

    public void update(){

        System.out.println("Enter id of student you want to update:");

        int id = scanner.nextInt();

*for* (Student student *:* students) {

*if* (student.getId() == id) {

                System.out.println("Enter new name:");

                String name = scanner.next();

                System.out.println("Enter new mark:");

                double mark = scanner.nextDouble();

                student.setName(name);

                student.setMark(mark);

                System.out.println("Update success!");

*return*;

            }

        }

        System.out.println("Student not found!");

    }

    public void remove(){

        System.out.println("Enter id of student you want to remove:");

        int id = scanner.nextInt();

*for* (Student student *:* students) {

*if* (student.getId() == id) {

                students.remove(student);

                System.out.println("Remove success!");

*return*;

            }

        }

        System.out.println("Student not found!");

    }

}

**MENU**

public class Menu {

    public static int getUserChoice() {

        int choice;

        System.out.println("Press 1: Create");

        System.out.println("Press 2: Display");

        System.out.println("Press 3: Remove");

        System.out.println("Press 4: Update");

        System.out.println("Press 5: Exit");

        System.out.println("Enter choise");

        choice = StudentList.scanner.nextInt();

*return* choice;

    }

}

**MAIN**

public class Main {

    public static void main(String[] args) {

        StudentList students = *new* StudentList();

        int choice;

*while* (true) {

            choice = Menu.getUserChoice();

*switch* (choice) {

*case* 1*:*

                    students.create();

*break*;

*case* 2*:*

                    students.display();

*break*;

*case* 3*:*

                    students.display();

                    students.remove();

*break*;

*case* 4*:*

                    students.update();

*break*;

*case* 5*:*

                    System.out.println("STOP!!!!");

*return*;

*default:*

*break*;

            }

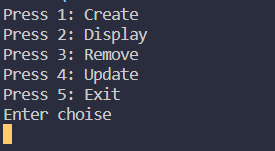
        }

    }

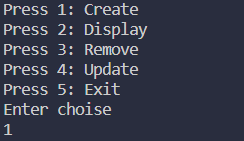
}

**Kết quả:**

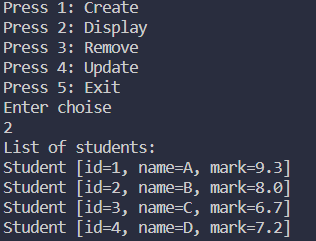
**Giao diện đầu**

****

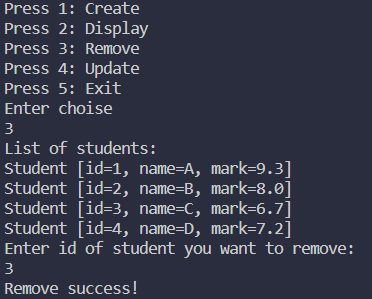
**Create (tự động thêm 4 sinh viên đã khai báo sẵn)**

****

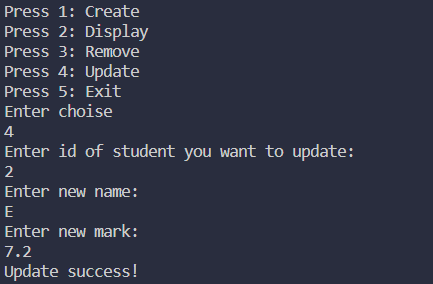
**Display (in ra danh sách sinh viên – từ create 4 sinh viên trên)**

****

**Remove**

****

**Update**

****

**Chương 2: Danh sách liên kết LinkedList**

**Câu 1:** Cho danh sách liên kết đơn SinglyLinkedList, mỗi phần tử chứa dữ liệu là số nguyên. Hãy khai báo cấu trúc dữ liệu SinglyLinkedList bao gồm các hàm sau đây:

* void addFirst(int x) – thêm một node với data x vào đầu danh sách liên kết.
* Node removeFromFront()- xóa một node đầu danh sách và trả về giá trị node xóa.
* int count()- đếm các phần tử có trong danh sách
* void printList()- duyệt và in ra data của tất cả các node trong danh sách.

**CLASS SINGLYNODE**

public class SinglyNode {

    private int singly;

    private SinglyNode next;

    public SinglyNode(int singly) {

*this*.singly = singly;

    }

    public SinglyNode() {

    }

    public int getSingly() {

*return* singly;

    }

    public void setSingly(int singly) {

*this*.singly = singly;

    }

    public SinglyNode getNext() {

*return* next;

    }

    public void setNext(SinglyNode next) {

*this*.next = next;

    }

    @Override

    public String toString() {

*return* "SinglyNode [singly=" + singly + "]";

    }

}

**CLASS SINGLYLINKEDLIST**

public class SinglyLinkedList {

    private SinglyNode head;

    private int size = 0;

    public void addFirst(int s) {

        SinglyNode newSingly = *new* SinglyNode(s);

        newSingly.setNext(head);

        head = newSingly;

        size++;

    }

    public boolean isEmpty() {

*return* head == null;

    }

    public SinglyNode removeFromFront() {

*if* (isEmpty()) {

*return* null;

        }

        SinglyNode removeNode = head;

        head = head.getNext();

        size--;

        removeNode.setNext(null);

*return* removeNode;

    }

    public void printList() {

        SinglyNode current = head;

        System.out.println("HEAD =>");

*while* (current != null) {

            System.out.println("" + current);

            System.out.println("->");

            current = current.getNext();

        }

        System.out.println("NULL");

    }

    public int count() {

*return* *this*.size;

    }

}

**MAIN**

public class Main {

    public static void main(String[] args) {

        SinglyLinkedList a1 = *new* SinglyLinkedList();

        a1.addFirst(1);

        a1.addFirst(2);

        a1.addFirst(3);

        a1.addFirst(4);

        a1.addFirst(5);

        a1.printList();

        a1.removeFromFront();

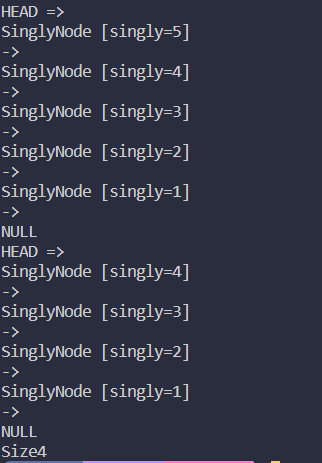
        a1.printList();

        System.out.println("Size" + a1.count());

    }

}

**Kết quả:**

****

**Câu 2:** Cho danh sách liên kết đôi DoublyLinkedList, mỗi phần tử chứa thông tin về bệnh nhân Patient bao gồm các trường mã số bệnh nhân id (số nguyên), tên bệnh nhân name (chuỗi), lịch sử bệnh medicalHistory (chuỗi), và viện phí hospitalFee (số thực). Hãy khai báo cấu trúc dữ liệu DoublyLinkedList bao gồm các hàm sau đây:

* void insertAtHead(Patient p) – thêm một bệnh nhân vào đầu danh sách liên kết đôi.
* void insertAtTail((Patient p) – thêm một bệnh nhâp p vào cuối danh sách liên kết đôi.
* PatientNode removeFromHead()-xóa một node đầu danh sách và trả về giá trị node xóa.
* PatientNode removeFromEnd()-xóa một node từ đuôi danh sách và trả về giá trị node xóa.
* int count()- đếm các phần tử Patient có trong danh sách
* void sortByName()- sắp xếp danh sách các phần tử theo tên name
* vid printList()- duyệt và in ra data của tất cả các nde bệnh nhân trng danh sách

**CLASS PATIENT**

public class Patient {

    private int id;

    private String firstName;

    private String lastName;

    private String medicalHistory;

    private float hospitalFee;

    public Patient(int \_id, String \_firstName, String \_lastName, String \_medicalHistory, float \_hospitalFee) {

*this*.id = \_id;

*this*.firstName = \_firstName;

*this*.lastName = \_lastName;

*this*.medicalHistory = \_medicalHistory;

*this*.hospitalFee = (float) \_hospitalFee;

    }

    public int getId() {

*return* id;

    }

    public void setId(int id) {

*this*.id = id;

    }

    public String getFirstName() {

*return* firstName;

    }

    public void setFirstName(String firstName) {

*this*.firstName = firstName;

    }

    public String getLastName() {

*return* lastName;

    }

    public void setLastName(String lastName) {

*this*.lastName = lastName;

    }

    public String getMedicalHistory() {

*return* medicalHistory;

    }

    public void setMedicalHistory(String medicalHistory) {

*this*.medicalHistory = medicalHistory;

    }

    public float getHospitalFee() {

*return* hospitalFee;

    }

    public void setHospitalFee(float hospitalFee) {

*this*.hospitalFee = hospitalFee;

    }

    @Override

    public String toString() {

*return* "Patient [id=" + id + ", firstName=" + firstName + ", lastName=" + lastName + ", medicalHistory="

                + medicalHistory + ", hospitalFee=" + hospitalFee

                + "]";

    }

}

**CLASS PATIENTNODE**

public class PatientNode {

    private Patient patient;

    private PatientNode next;

    private PatientNode previous;

    public PatientNode(Patient patient) {

*this*.patient = patient;

    }

    public Patient getPatient() {

*return* patient;

    }

    public void setPatient(Patient patient) {

*this*.patient = patient;

    }

    public PatientNode getNext() {

*return* next;

    }

    public void setNext(PatientNode next) {

*this*.next = next;

    }

    public PatientNode getPrevious() {

*return* previous;

    }

    public void setPrevious(PatientNode provious) {

*this*.previous = provious;

    }

    @Override

    public String toString() {

*return* "PatientNode [patient=" + patient + ", next=" + next

                + ", provious=" + previous + "]";

    }

}

**CLASS PATIENTDOUBLELINKEDLIST**

public class PatientDoublyLinkedList {

    private PatientNode head;

    private PatientNode tail;

    private int size = 0;

    public void insertAtHead(Patient patient) {

        PatientNode newNode = *new* PatientNode(patient);

*if* (head == null) {

            tail = newNode;

        } *else* {

            newNode.setNext(head);

            head.setPrevious(newNode);

        }

        head = newNode;

        size++;

    }

    public void insertAtTail(Patient patient) {

        PatientNode newNode = *new* PatientNode(patient);

*if* (tail == null) {

            head = newNode;

        } *else* {

            newNode.setPrevious(tail);

            tail.setNext(newNode);

        }

        tail = newNode;

        size++;

    }

    public boolean isEmpty() {

*return* head == null;

    }

    public PatientNode removeFromEnd() {

*if* (isEmpty()) {

*return* null;

        }

        PatientNode removedNode = tail;

*if* (tail.getPrevious() == null) {

            head = null;

        } *else* {

            tail.getPrevious().setNext(null);

        }

        tail = tail.getPrevious();

        size--;

        removedNode.setNext(null);

*return* removedNode;

    }

    public void printList() {

        PatientNode current = head;

        System.out.println("HEAD =>");

*while* (current != null) {

            System.out.println("" +

                    current.getPatient().toString());

            System.out.println(" <=> ");

            current = current.getNext();

        }

        System.out.println("NULL");

    }

    public void sortByName() {

        PatientNode current = null, index = null;

        Patient temp;

*if* (head == null) {

*return*;

        } *else* {

*for* (current = head; current.getNext() != null; current = current.getNext()) {

*for* (index = current.getNext(); index != null; index = index.getNext()) {

                    String a = current.getPatient().getFirstName();

                    char[] charArray1 = a.toCharArray();

                    String b = index.getPatient().getFirstName();

                    char[] charArray2 = b.toCharArray();

*if* (Character.compare(charArray1[0],

                            charArray2[0]) > 0) {

                        temp = current.getPatient();

                        current.setPatient(index.getPatient());

                        index.setPatient(temp);

                    }

                }

            }

        }

    }

    public int count() {

*return* *this*.size;

    }

}

**MAIN**

public class Main {

    public static void main(String[] args) {

        Patient a1 = *new* Patient(0, "Long", "Nguyen", "No", 100);

        Patient a2 = *new* Patient(1, "Tinh", "Nguyen", "No", 155);

        Patient a3 = *new* Patient(2, "An", "Ngo", "Yen", 105);

        Patient a4 = *new* Patient(3, "Hoang", "Tran", "No", 185);

        Patient a5 = *new* Patient(4, "Ngoc", "Huynh", "No", 185);

        PatientDoublyLinkedList c = *new* PatientDoublyLinkedList();

        c.insertAtTail(a1);

        c.insertAtTail(a2);

        c.printList();

        System.out.println(c.count());

        PatientDoublyLinkedList c1 = *new* PatientDoublyLinkedList();

        c1.insertAtHead(a1);

        c1.insertAtHead(a2);

        c1.insertAtHead(a3);

        c1.insertAtHead(a4);

        c1.insertAtHead(a5);

        c1.printList();

        System.out.println(c1.count());

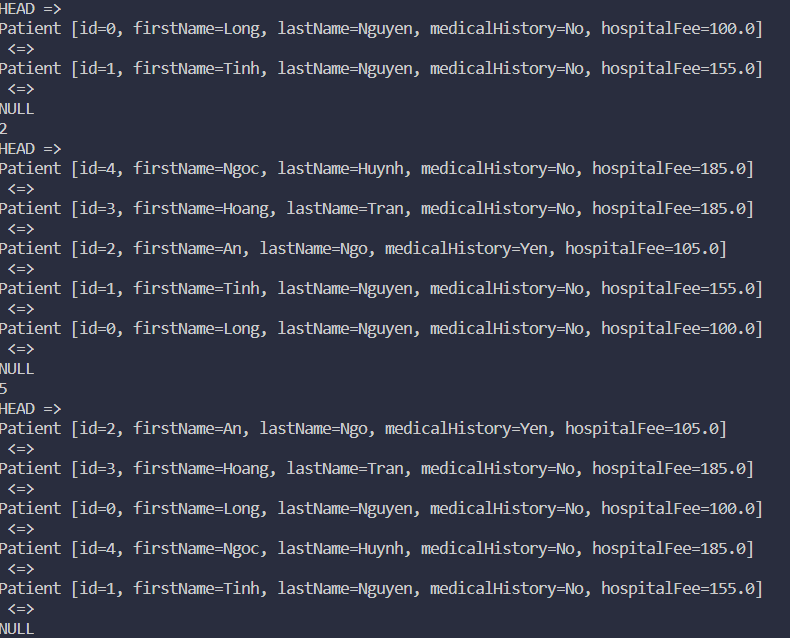
        c1.sortByName();

        c1.printList();

    }

}

**Kết quả:**

****

**Câu 3:** Cho danh sách liên kết đơn SinglyLinkedList chứa danh sách hàng hóa, mỗi phần tử chứa dữ liệu là mặt hang bao gồm các thông tin Tên mặt hang, giá mặt hang, Số lượng còn trong kho. Hãy thực hiện các yêu cầu sau:

* Khai báo danh dách liên kết lưu danh sách các mặt hàng theo thông tin như mô tả trên.
* Viết hàm hiển thị danh sách mặt hàng ở câu a.
* Viết hàm trả về số lượng các phần tử đang có trong danh sách câu a

**CLASS SINGLY**

public class Singly {

    private String name;

    private float price;

    private int quantity;

    public Singly(String \_name, float \_price, int \_quantity) {

*this*.name = \_name;

*this*.price = (float) \_price;

*this*.quantity = \_quantity;

    }

    public String getName() {

*return* name;

    }

    public void setName(String name) {

*this*.name = name;

    }

    public float getPrice() {

*return* price;

    }

    public void setPrice(float price) {

*this*.price = price;

    }

    public int getQuantity() {

*return* quantity;

    }

    public void setQuantity(int quantity) {

*this*.quantity = quantity;

    }

    @Override

    public String toString() {

*return* "Singly [name=" + name + ", price=" + price + ", quantity=" + quantity + "]";

    }

}

**CLASS SINGLYNODE**

public class SinglyNode {

    private Singly singly;

    private SinglyNode next;

    public SinglyNode(Singly singly) {

*this*.singly = singly;

    }

    public Singly getSingly() {

*return* singly;

    }

    public void setSingly(Singly singly) {

*this*.singly = singly;

    }

    public SinglyNode getNext() {

*return* next;

    }

    public void setNext(SinglyNode next) {

*this*.next = next;

    }

    @Override

    public String toString() {

*return* "SinglyNode [singly=" + singly + ", next=" + next + "]";

    }

}

**CLASS SINGLYLINKEDLIST**

public class SinglyLinkedList {

    private SinglyNode head;

    private int size = 0;

    public SinglyLinkedList() {

    }

    public void addFirst(Singly singly) {

        SinglyNode newNode = *new* SinglyNode(singly);

        newNode.setNext(head);

        head = newNode;

        size++;

    }

    public void printList() {

*if* (size == 0) {

            System.out.println("Chưa có dữ liệu!");

        } *else* {

            SinglyNode current = head;

            System.out.println("HEAD =>");

*while* (current != null) {

                System.out.println("" + current.getSingly().toString());

                System.out.println("->");

                current = current.getNext();

            }

            System.out.println("NULL");

        }

    }

    public int count() {

*return* *this*.size;

    }

}

**MAIN**

public class Main {

    public static void main(String[] args) {

        Singly s1 = *new* Singly("Trung", 10000, 20);

        Singly s2 = *new* Singly("Ca", 10000, 30);

        Singly s3 = *new* Singly("Thit", 20000, 10);

        Singly s4 = *new* Singly("Gao", 15000, 50);

        SinglyLinkedList list = *new* SinglyLinkedList();

        list.addFirst(s1);

        list.addFirst(s2);

        list.addFirst(s3);

        list.addFirst(s4);

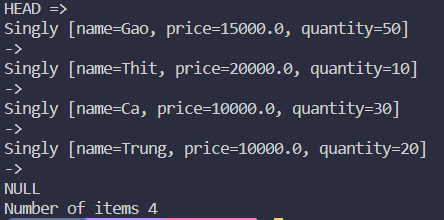
        list.printList();

        System.out.println("Number of items: " + list.count());

    }

}

**Kết quả:**

****

**Câu 4:** Cho một danh sách liên kết đôi đã lưu thông tin về sản phẩm trong một công ty, bao gồm:

* Mã sản phẩm (kiểu số nguyên)
* Tên sản phẩm (kiểu chuỗi)
* Chủng loại (bằng Giấy, bằng Kim loại, bằng Nhựa)
* Năm sản xuất (kiểu số nguyên)
* Số năm bảo hành (kiểu số nguyên)

Hai con trỏ Head, Tail đang trỏ đến phần tử đầu tiên và cuối cùng trong danh sách trên. Hãy thực hiện các yêu cầu sau :

* Viết hàm thêm vào sản phẩm.
* Viết hàm xóa các sản phẩm đã hết hạn bảo hành ra khỏi danh sách khi thỏa điều kiện : Năm sản xuất + Số năm bảo hành > Năm hiện tại.
* Viết hàm hiển thị danh sách các sản phẩm có trong công ty.
* Viết hàm trả về số lượng sản phẩm có trong danh sách

**Class HangHoa**

public class HangHoa {

    private int maSanPham;

    private String tenSanPham;

    private String chungLoai;

    private int namSanXuat;

    private int namBaoHanh;

    private final String[] chuanLoaiHang = { "Giay", "Kim Loai", "Nhua" };

    public HangHoa(int \_maSanPham, String \_tenSanPham, String \_chungLoai, int \_namSanXuat, int \_namBaoHanh) {

*if* (checkChungLoai(\_chungLoai)) {

*this*.maSanPham = \_maSanPham;

*this*.tenSanPham = \_tenSanPham;

*this*.chungLoai = \_chungLoai;

*this*.namSanXuat = \_namSanXuat;

*this*.namBaoHanh = \_namBaoHanh;

        } *else* {

            System.out.println("Lỗi chủng loại!! (Giay, Kim Loai, Nhua)");

*return*;

        }

    }

    public int getMaSanPham() {

*return* maSanPham;

    }

    public void setMaSanPham(int maSanPham) {

*this*.maSanPham = maSanPham;

    }

    public String getTenSanPham() {

*return* tenSanPham;

    }

    public void setTenSanPham(String tenSanPham) {

*this*.tenSanPham = tenSanPham;

    }

    public String getChungLoai() {

*return* chungLoai;

    }

    public void setChungLoai(String chungLoai) {

*if* (checkChungLoai(chungLoai)) {

*this*.chungLoai = chungLoai;

        } *else* {

            System.out.println("Lỗi chủng loại!! (Giay, Kim Loai, Nhua)");

        }

    }

    public boolean checkChungLoai(String e) {

*for* (int i = 0; i < *this*.chuanLoaiHang.length; i++) {

*if* (e.toUpperCase().equals(chuanLoaiHang[i].toUpperCase())) {

*return* true;

            }

        }

*return* false;

    }

    public int getNamSanXuat() {

*return* namSanXuat;

    }

    public void setNamSanXuat(int namSanXuat) {

*this*.namSanXuat = namSanXuat;

    }

    public int getNamBaoHanh() {

*return* namBaoHanh;

    }

    public void setNamBaoHanh(int namBaoHanh) {

*this*.namBaoHanh = namBaoHanh;

    }

    @Override

    public String toString() {

*return* "HangHoa [maSanPham=" + maSanPham + ", tenSanPham=" +

                tenSanPham + ", chungLoai=" + chungLoai

                + ", namSanXuat=" + namSanXuat + ", namBaoHanh="

                + namBaoHanh + "]";

    }

}

**Class HangHoaNode**

public class HangHoaNode {

    private HangHoa hangHoa;

    private HangHoaNode next;

    private HangHoaNode previous;

    public HangHoaNode(HangHoa hangHoa) {

*this*.hangHoa = hangHoa;

    }

    public HangHoa getHangHoa() {

*return* hangHoa;

    }

    public void setHangHoa(HangHoa hangHoa) {

*this*.hangHoa = hangHoa;

    }

    public HangHoaNode getNext() {

*return* next;

    }

    public void setNext(HangHoaNode next) {

*this*.next = next;

    }

    public HangHoaNode getPrevious() {

*return* previous;

    }

    public void setPrevious(HangHoaNode previous) {

*this*.previous = previous;

    }

    @Override

    public String toString() {

*return* "HangHoaNode [hangHoa=" + hangHoa + ", next=" + next

                + ", previous=" + previous + "]";

    }

}

**Class HangHoaDoubleLinkedList**

import java.util.Calendar;

public class HangHoaDoubleLinkedList {

    private HangHoaNode head;

    private HangHoaNode tail;

    private int size = 0;

    public void insertAtHead(HangHoa hangHoa) {

        HangHoaNode newNode = *new* HangHoaNode(hangHoa);

*if* (head == null) {

            tail = newNode;

        } *else* {

            newNode.setNext(head);

            head.setPrevious(newNode);

        }

        head = newNode;

        size++;

    }

    public void insertAtTail(HangHoa hangHoa) {

        HangHoaNode newNode = *new* HangHoaNode(hangHoa);

*if* (tail == null) {

            head = newNode;

        } *else* {

            newNode.setPrevious(tail);

            tail.setNext(newNode);

        }

        tail = newNode;

        size++;

    }

    public boolean isEmpty() {

*return* head == null;

    }

    public void removeHangHoa() {

        HangHoaNode current = head;

        head = null;

        tail = null;

        size = 0;

        Calendar calendar = Calendar.getInstance();

        int year = calendar.get(Calendar.YEAR);

*while* (current != null) {

            int hieuSo = (current.getHangHoa().getNamBaoHanh() +

                    current.getHangHoa().getNamSanXuat()) - year;

            System.out.println(hieuSo < 0);

*if* (hieuSo >= 0) {

                insertAtHead(current.getHangHoa());

            }

            current = current.getNext();

        }

    }

    public int count() {

*return* size;

    }

    public void printList() {

        HangHoaNode current = head;

        System.out.println("HEAD =>");

*while* (current != null) {

            System.out.println("" +

                    current.getHangHoa().toString());

            System.out.println(" <=> ");

            current = current.getNext();

        }

        System.out.println("NULL");

    }

}

**Main**

public class Main {

    public static void main(String[] args) {

        HangHoa hangHoa1 = *new* HangHoa(0, "Roi", "kim loai", 2018, 2);

        HangHoa hangHoa2 = *new* HangHoa(1, "Kiem", "kim loai", 2020, 5);

        HangHoa hangHoa3 = *new* HangHoa(2, "Ban", "kim loai", 2021, 4);

        HangHoaNode c = *new* HangHoaNode(hangHoa2);

        HangHoaDoubleLinkedList a = *new* HangHoaDoubleLinkedList();

        a.insertAtHead(hangHoa1);

        a.insertAtHead(hangHoa2);

        a.insertAtHead(hangHoa3);

        a.printList();

        a.removeHangHoa();

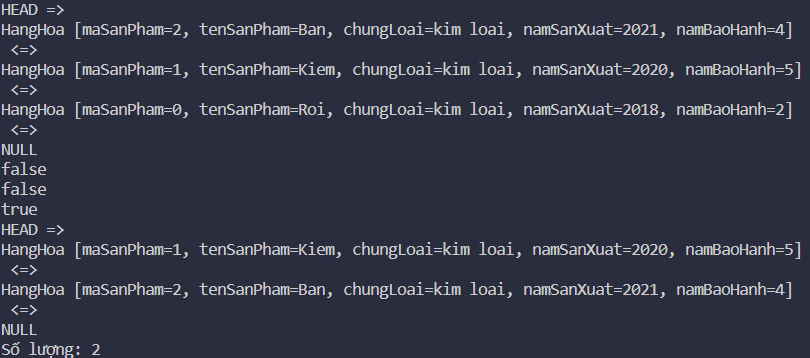
        a.printList();

        System.out.println("Số lượng: " + a.count());

    }

}

**Kết quả:**

****

**Chương 3: Ngăn xếp Stack**

**Câu 5:** Hãy cho biết nội dung của stack sau mỗi thao tác trong dãy:

EAS\*Y\*\*QUE\*\*\*ST\*\*\*I\*ON

Với một chữ cái tượng trưng cho thao tác thêm chữ cái tương ứng vào stack, dấu \* tượng trưng cho thao tác lấy nội dung một phần tử trong stack in lên màn hình. Hãy cho biết sau khi hoàn tất chuỗi thao tác, những gì xuất hiện trên màn hình?

**Main**

import java.util.Stack;

public class Main {

    public static void main(String[] args) {

        String s = "EAS\*Y\*\*QUE\*\*\*ST\*\*\*I\*ON";

        char[] ch = *new* char[s.length()];

*for* (int i = 0; i < s.length(); i++) {

            ch[i] = s.charAt(i);

        }

*for* (char c *:* ch) {

            System.out.print(c);

        }

        Stack<String> stack = *new* Stack<String>();

*for* (char a *:* ch) {

            String m = "" + a;

            stack.push(m);

        }

        System.out.println();

        System.out.println("Size: " + stack.size());

        System.out.println("Stack ban dau: " + stack);

        Stack<String> stack1 = *new* Stack<String>();

*while* (stack.size() > 0) {

            String str = stack.pop();

*if* (str.equals("\*")) {

*continue*;

            } *else* {

                stack1.push(str);

            }

        }

        System.out.println("Stack pop \*: " + stack1);

        String string = "";

*while* (stack1.size() > 0) {

            String str = stack1.pop();

            stack.push(str);

            string += str;

        }

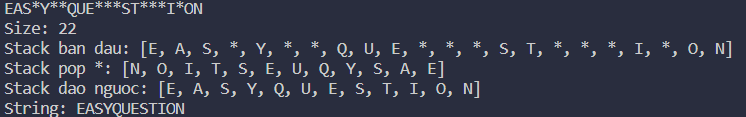
        System.out.println("Stack dao nguoc: " + stack);

        System.out.println("String: " + string);

    }

}

**Kết quả:**

****

**Câu 6:** Giả sử cho hàm push(a) là hàm thực hiện nạp a vào ngăn xếp và hàm pop() là hàm lấy phần tử ra khỏi ngăn xếp. Giả sử cho dãy thao tác sau đây, biết rằng ngăn xếp ban đầu được khởi tạo rỗng : push(5), push(3), pop(), push(2), push(8), pop(), pop(), pop(), push(9), push(1), pop(), push(7), push(6), pop(), pop(), push(4), pop(), pop(). Hãy viết ra dãy phần tử của danh của ngăn xếp (chỉ rõ vị trí đầu ngăn xếp) sau khi thực hiện mỗi thao tác.

**Giải:**

Đầu tiên ta có :

push(5), push(3) => C = [5, 3], pop() => c = [5],

push(2), push(8) => C = [5, 2, 8],

pop(), pop(), pop() => c = []

push(9), push(1) => c = [9, 1]

pop() => c = [9]

push(7), push(6) => c = [9, 7, 6]

pop(), pop() => c = [9]

push(4) => c = [9, 4]

pop(), pop() => c = [] Kết luận cuối cùng c rỗng

**Câu 7:** Viết chương trình chuyển đổi một số nguyên N trong hệ thập phân (hệ 10) sang biểu diễn ở các hệ sau, sử dụng ngăn xếp:

* Hệ nhị phân (hệ 2)
* Hệ bát phân (hệ 8)

**Class MyStack**

import java.util.EmptyStackException;

public class MyStack {

    private int[] stack;

    private int top;

    public MyStack(int capacity) {

        stack = *new* int[capacity];

    }

    public void push(int value) {

*if* (top == stack.length) {

            int[] newStack = *new* int[stack.length \* 2];

            System.arraycopy(stack, 0, newStack, 0, stack.length);

            stack = newStack;

        }

        stack[top++] = value;

    }

    public boolean isEmpty() {

*return* top == 0;

    }

    public int pop() {

*if* (isEmpty()) {

*throw* *new* EmptyStackException();

        }

        int value = stack[top - 1];

        stack[top - 1] = 0;

        top--;

*return* value;

    }

    public void printStack() {

*for* (int i = top - 1; i >= 0; i--) {

            System.out.print(" " + stack[i]);

        }

    }

}

**Main**

public class Main {

    public static void NhiPhan(int k) {

        MyStack s = *new* MyStack(20);

        System.out.print(k + " in binary system is: ");

*while* (k > 0) {

            s.push(Integer.valueOf(k % 2));

            k = k / 2;

        }

*while* (!s.isEmpty()) {

            System.out.print(s.pop());

        }

    }

    public static void BatPhan(int k) {

        MyStack s = *new* MyStack(20);

        System.out.print(k + " in oct system is: ");

*while* (k > 0) {

            s.push(Integer.valueOf(k % 8));

            k = k / 8;

        }

*while* (!s.isEmpty()) {

            System.out.print(s.pop());

        }

    }

    public static void main(String[] args) {

        NhiPhan(100);

        System.out.println();

        BatPhan(100);

    }

}

**Kết quả:**

****

**Câu 8:** Viết chương trình nhập vào một số nguyên không âm bất kỳ, sau đó hiển thị lên màn hình số đảo ngược thứ tự của số nguyên vừa nhập vào (ví dụ: nếu nhập số 12567, hiển thị lên màn hình số 76521) bằng cách:

* Sử dụng ngăn xếp
* Sử dụng hàng đợi

**Class MyStack**

import java.util.EmptyStackException;

public class MyStack {

    private int[] stack;

    private int top;

    public MyStack(int capacity) {

        stack = *new* int[capacity];

    }

    public void push(int value) {

*if* (top == stack.length) {

            int[] newStack = *new* int[stack.length \* 2];

            System.arraycopy(stack, 0, newStack, 0, stack.length);

            stack = newStack;

        }

        stack[top++] = value;

    }

    public boolean isEmpty() {

*return* top == 0;

    }

    public int pop() {

*if* (isEmpty()) {

*throw* *new* EmptyStackException();

        }

        int value = stack[top - 1];

        stack[top - 1] = 0;

        top--;

*return* value;

    }

    public void printStack() {

*for* (int i = top - 1; i >= 0; i--) {

            System.out.print(" " + stack[i]);

        }

    }

}

**Main**

import java.util.PriorityQueue;

import java.util.Queue;

public class Main {

    public static void reseverStack(int number) {

        MyStack stack = *new* MyStack(20);

        System.out.print(number + " So dao nguoc (Stack) : ");

*while* (number > 0) {

            stack.push(Integer.valueOf(number % 10));

            number = number / 10;

            System.out.print(stack.pop());

        }

    }

    public static void reseverQueue(int number) {

        Queue<Integer> N = *new* PriorityQueue<Integer>();

        System.out.print(number + " So dao nguoc (Hang Doi) : ");

*while* (number > 0) {

            N.add(number % 10);

            number = number / 10;

            System.out.print(N.peek());

        }

        System.out.println();

    }

    public static void main(String[] args) {

        reseverStack(1234567);

        System.out.println();

        reseverQueue(1234567);

    }

}

**Kết quả:**

****

**Câu 9:** Giả sử cho hàm push(a) là hàm thực hiện nạp a vào ngăn xếp và hàm pop() là hàm lấy phần tử ra khỏi ngăn xếp. Giả sử cho dãy thao tác sau đây, biết rằng ngăn xếp ban đầu được khởi tạo rỗng chứa số nguyên và x, y, z là những biến số nguyên. Hiển thị output của chương trình.

x = 3;y = 5;z = 2;

s.makeEmpty();

s.push(x);s.push(4);s.pop();

s.push(y);s.push(3); s.push(z);

s.pop();s.push(2);s.push(x);

while(! s.isEmpty( )) System.out.println(s.pop() + " ");

**Giải:**

import java.util.Stack;

public class Main {

    public static void main(String[] args) {

        Stack<Integer> s = *new* Stack<>();

        int x = 3;

        int y = 5;

        int z = 2;

        s.clear();

        s.push(x);

        s.push(4);

        s.pop();

        s.push(y);

        s.push(3);

        s.push(z);

        s.pop();

        s.push(2);

        s.push(x);

*while* (!s.isEmpty())

            System.out.println(s.pop() + " ");

    }

}

**Kết quả:**

****

**Câu 10:** Giả sử cho hàm push(a) là hàm thực hiện nạp a vào ngăn xếp và hàm pop() là hàm lấy phần tử ra khỏi ngăn xếp. Giả sử cho dãy thao tác sau đây, biết rằng ngăn xếp ban đầu được khởi tạo rỗng chứa số nguyên và x, y, z là những biến số nguyên. Hiển thị output của chương trình.

x = 3; y = 1;

s.makeEmpty();

s.push(5);s.push(7);s.pop();

x += y;

s.pop();

s.push(x);s.push(y); s.push(2);s.pop();s.pop();

while (! s.isEmpty( ))

{

y = s.pop();

System.out.println(y);

}

System.out.println("x = " + x);

System.out.println("y = " + y);

**Giải:**

import java.util.Stack;

public class Main {

    public static void main(String[] args) {

        Stack<Integer> s = *new* Stack<>();

        int x = 3;

        int y = 1;

        s.clear();

        s.push(5);

        s.push(7);

        s.pop();

        x += y;

        s.pop();

        s.push(x);

        s.push(y);

        s.push(2);

        s.pop();

        s.pop();

*while* (!s.isEmpty()) {

            y = s.pop();

            System.out.println(y);

        }

        System.out.println("x = " + x);

        System.out.println("y = " + y);

    }

}

**Kết quả:**

****

**Câu 11:** Khai báo cấu trúc dữ liệu là ngăn xếp dùng để chứa các số nguyên gồm các phương thức sau đây:

* boolean isEmpty() - return true if the stack is empty and false otherwise.
* void clear() - clear the stack.
* void push(int x) - insert a node with value x at the top of the stack.
* int pop() - remove the top element on the stack and return it's value; throw EmptyStackException for empty stack.
* int top() - return value of a node at the top of the stack; throw EmptyStackException for empty stack.
* void traverse() - display all values in the stack from the top to the bottom.
* Use a stack to convert an integer number in decimal system to binary system and display on the screen.

**Class MyStack**

import java.util.EmptyStackException;

public class MyStack {

    private int[] stack;

    private int top;

    public MyStack(int capacity) {

        stack = *new* int[capacity];

    }

    public void push(int value) {

*if* (top == stack.length) {

            int[] newStack = *new* int[stack.length \* 2];

            System.arraycopy(stack, 0, newStack, 0, stack.length);

            stack = newStack;

        }

        stack[top++] = value;

    }

    public int pop() {

*if* (isEmpty()) {

*throw* *new* EmptyStackException();

        }

        int value = stack[top - 1];

        stack[top - 1] = 0;

        top--;

*return* value;

    }

    public int top() {

*if* (isEmpty()) {

*throw* *new* EmptyStackException();

        }

*return* stack[top - 1];

    }

    public void clear() {

*while* (top != 0) {

            stack[top - 1] = 0;

            top--;

        }

    }

    public int size() {

*return* top;

    }

    public void printStack() {

*if* (top == 0) {

            System.out.println("Null");

        } *else* {

*for* (int i = top - 1; i >= 0; i--) {

                System.out.print(" " + stack[i]);

            }

        }

    }

    public void traverse() {

*if* (top == 0) {

            System.out.println("Null");

        } *else* {

*for* (int i = 0; i < top; i++) {

                System.out.print(" " + stack[i]);

            }

        }

    }

    public boolean isEmpty() {

*return* top == 0;

    }

}

**Main**

public class Main {

    public static void main(String[] args) {

        MyStack stack = *new* MyStack(20);

        stack.push(20);

        stack.push(9);

        stack.push(7);

        stack.push(10);

        stack.push(5);

        stack.push(58);

        stack.printStack();

        System.out.println();

        stack.traverse();

        stack.clear();

        System.out.println();

        stack.printStack();

    }

}

**Kết quả:**

****

**Chương 4: Hàng đợi Queue**

**Câu 12:** Giả sử cho hàm enq(a) là hàm thực hiện nạp a vào hàng đợi và hàm deq() là hàm thực hiện lấy phần tử ra khỏi hàng đợi. Giả sử cho dãy các thao tác sau đây, biết rằng hàng đợi ban đầu được khởi tạo rỗng:

enq(5), enq(3), deq(), enq(2), enq(8), deq(), enq(9), enq(1), deq(), enq(7), enq(6), deq(), deq(). enq(4), deq(), deq().

Hãy viết ra dãy các phần tử của hàng đợi (chỉ rõ vị trí đầu và cuối của hàng đợi) sau khi thực hiện mỗi thao tác.

**Class MyQueue**

public class MyQueue {

    public int[] queue;

    private int back = 0;

    private int front = 0;

    public MyQueue(int capacity) {

        queue = *new* int[capacity];

    }

    public void enq(int e) {

*if* (queue.length == back) {

            int[] newArray = *new* int[queue.length \* 2];

            System.arraycopy(queue, 0, newArray, 0, queue.length);

            queue = newArray;

        }

        queue[back] = e;

        back++;

    }

    public void printQueue() {

*for* (int i = front; i < back; i++) {

            System.out.println("" + queue[i]);

        }

    }

    public int size() {

*return* back - front;

    }

    public int deq() {

*if* (*this*.size() == 0) {

            System.out.println("The Queue is Empty");

        }

        int e = queue[front];

        queue[front] = 0;

        front++;

*if* (size() == 0) {

            front = 0;

            back = 0;

        }

*return* e;

    }

    public int getBack() {

*return* back;

    }

    public void setBack(int back) {

*this*.back = back;

    }

    public int getFront() {

*return* front;

    }

    public void setFront(int front) {

*this*.front = front;

    }

}

**Main**

public class Main {

    public static void main(String[] args) {

        MyQueue myQueue = *new* MyQueue(20);

        myQueue.enq(5);

        myQueue.enq(3);

        myQueue.deq();

        myQueue.enq(2);

        myQueue.enq(8);

        myQueue.deq();

        myQueue.enq(9);

        myQueue.enq(1);

        myQueue.deq();

        myQueue.enq(7);

        myQueue.enq(6);

        myQueue.deq();

        myQueue.deq();

        myQueue.enq(4);

        myQueue.deq();

        myQueue.deq();

        myQueue.printQueue();

    }

}

**Kết quả:**

****

**Câu 13:** Viết chương trình nhập vào một số nguyên không âm bất kỳ, sau đó hiển thị lên màn hình số đảo ngược thứ tự của số nguyên vừa nhập vào (ví dụ: nếu nhập số 12567, hiển thị lên màn hình số 76521) bằng cách:

* Sử dụng hàng đợi

**Giải:**

import java.util.LinkedList;

import java.util.PriorityQueue;

import java.util.Queue;

public class Main {

    public static void reseverQueue(int number) {

        Queue<Integer> myQueue = *new* PriorityQueue<Integer>();

        System.out.print(number + " So dao nguoc: ");

*while* (number > 0) {

            myQueue.add(number % 10);

            number = number / 10;

            System.out.print(myQueue.peek());

        }

        System.out.println();

    }

    private static void reverseQueue1(int number) {

        Queue<Integer> myQueue = *new* LinkedList<>();

*while* (number > 0) {

            myQueue.add(number % 10);

            number = number / 10;

        }

        System.out.println("myQueue : " + myQueue);

        int numberResever = 0;

        System.out.print("In ra myQueue: ");

*for* (Integer i *:* myQueue) {

            numberResever = numberResever \* 10 + i;

            System.out.print(i + " ");

        }

        System.out.println();

        System.out.println("So dao nguoc: " + numberResever);

    }

    public static void main(String[] args) {

        System.out.println("Cach 1: ");

        reseverQueue(1234567);

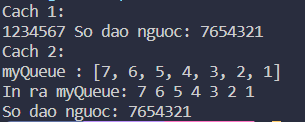
        System.out.println("Cach 2:");

        reverseQueue1(1234567);

    }

}

**Kết quả:**

****

**Câu 14:** Khai báo cấu trúc dữ liệu là hàng đợi dùng để chứa các số nguyên gồm các phương thức sau đây:

1. boolean isEmpty() - return true if the queue is empty and false otherwise. 2. void clear() - clear the queue.

3. void enqueue(int x) - insert a node with value x at the end of the queue.

4. int dequeue() - remove the first element on the queue and return it's value; throw Exception for empty queue.

5. int first() - return value of the first node of the queue; throw Exception for empty queue.

4. void traverse() - display all values in the queue from the front to the rear.

6. Use a queue to convert a real number less than 1 in decimal system to binary system and display on the screen. Note: You can write some constructors and other methods as you see they are necessary

**Class MyQueue**

import java.util.EmptyStackException;

public class MyQueue {

    public int[] queue;

    private int back = 0;

    private int front = 0;

    public MyQueue(int capacity) {

        queue = *new* int[capacity];

    }

    public void enqueue(int e) {

*if* (queue.length == back) {

            int[] newArray = *new* int[queue.length \* 2];

            System.arraycopy(queue, 0, newArray, 0, queue.length);

            queue = newArray;

        }

        queue[back] = e;

        back++;

    }

    public int dequeue() {

*if* (isEmpty()) {

*throw* *new* EmptyStackException();

        }

        int e = queue[front];

        queue[front] = 0;

        front++;

*if* (size() == 0) {

            front = 0;

            back = 0;

        }

*return* e;

    }

    public int first() {

*if* (isEmpty()) {

*throw* *new* EmptyStackException();

        }

*return* queue[front];

    }

    public void printQueue() {

*for* (int i = front; i < back; i++) {

            System.out.println("" + queue[i]);

        }

    }

    public void traverse() {

*for* (int i = back; i >= front; i--) {

            System.out.println("" + queue[i]);

        }

    }

    public int size() {

*return* back - front;

    }

    public void clear() {

*for* (int i = front; i < back; i++) {

            queue[i] = 0;

        }

        front = 0;

        back = 0;

    }

    public boolean isEmpty() {

*return* size() == 0;

    }

    public int getBack() {

*return* back;

    }

    public void setBack(int back) {

*this*.back = back;

    }

    public int getFront() {

*return* front;

    }

    public void setFront(int front) {

*this*.front = front;

    }

}

**Main**

public class Main {

    public static void main(String[] args) {

        MyQueue myQueue = *new* MyQueue(10);

        myQueue.enqueue(5);

        myQueue.enqueue(2);

        myQueue.enqueue(10);

        myQueue.enqueue(8);

        myQueue.enqueue(6);

        myQueue.printQueue();

        System.out.println();

        myQueue.traverse();

        System.out.println();

        System.out.println(myQueue.dequeue());

        myQueue.printQueue();

        System.out.println();

        myQueue.clear();

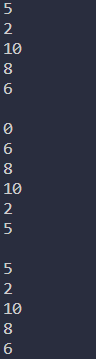
        myQueue.printQueue();

        System.out.println();

    }

}

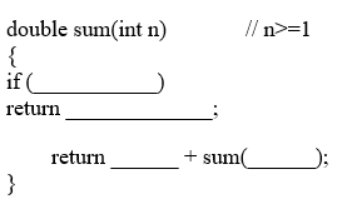
**Kết quả:**

****

**Chương 5: Đệ Quy**

**Câu 15:** Complete the Java code to recursively evaluate the sum:

*sum = 1 + 1/2 + 1/3 +...+1/n, n >= 1.*

****

**Giải:**

public class Main {

    public static int fact(int n) {

*if* (n <= 1)

*return* 1;

*return* n \* fact(n - 1);

    }

    public static void main(String[] args) {

        System.out.println("5! = " + fact(5));

        System.out.println("10! = " + fact(10));

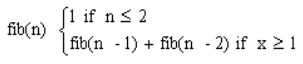
    }

}

**Kết quả:**

****

**Câu 17:** Implement Fibonacci recursively as **f**

****

**Giải:**

public class Main {

    public static int fibo(int n) {

*if* (n <= 2)

*return* 1;

*return* fibo(n - 1) + fibo(n - 2);

    }

    public static void main(String[] args) {

*for* (int i = 1; i <= 10; i++) {

            System.out.print(fibo(i) + " ");

        }

    }

}

**Kết quả:**

****

**Câu 18:** Write a recursive function that computes and returns the sum of all elements in an array, where the array and its size are given as parameters.

//return the sum of all elements in a[]

int findsum(int a[], int n)

**Giải:**

public class Main {

    public static int findsum(int[] array, int length) {

*if* (length == 1)

*return* array[0];

*return* array[length - 1] + findsum(array, length - 1);

    }

    public static void main(String[] args) {

        int[] numbers = { 1, 2, 3, 8 };

        System.out.println(findsum(numbers, numbers.length));

    }

}

**Kết quả:**

****

**Câu 19:** Write a recursive function that determines whether an array is a palindrome, where the array and its size are given as parameters.

//returns 1 if a[] is a palindrome, 0 otherwise. The string a is palindrome if it is the same as its reverse.

int ispalindrome(char a[], int n)

**Giải:**

public class Main {

    static boolean ispalindrome(char c[], int n) {

*if* (n == 1) {

*return* true;

        }

        int m = c.length;

*if* (c[n - 1] != c[m - n]) {

*return* false;

        } *else* {

*return* ispalindrome(c, n - 1);

        }

    }

    public static void main(String[] args) {

        char[] n = { 'a', 'b', 'c', 'c', 'b', 'a' };

        boolean check = ispalindrome(n, n.length);

*if* (check) {

            System.out.println("is Palindrome");

        } *else* {

            System.out.println("is not Palindrome");

        }

    }

}

**Kết quả:**

****

**Chương 6: Cây**

**Câu 20:** Cho một cây nhị phân tìm kiếm T, mỗi node là một số nguyên

* Hãy vẽ cây nhị phân tìm kiếm T biết rằng khi duyệt cây theo thứ tự Left-RightRoot thì ta được dãy như sau:

o 5, 3, 7, 9, 8, 11, 6, 20, 19, 37, 25, 21, 15, 12

* Khai báo cấu trúc dữ liệu cây nhị phân tìm kiếm
* Viết hàm thêm các node vào cây
* Viết hàm hiển thị node theo cách duyệt sâu ( inOrder, preOrder, postOrder)
* Viết hàm đếm số node lá trên cây
* Viết hàm tìm phần tử có khóa x trên cây

**Class Tree**

public class Tree {

    private TreeNode root;

    public void insert(int value) {

*if* (root == null) {

            root = *new* TreeNode(value);

        } *else* {

            root.insert(value);

        }

    }

    public TreeNode search(int value) {

*if* (root != null) {

*return* root.search(value);

        }

*return* null;

    }

    public void traverseInOrder() {

*if* (root != null) {

            root.traverseInOrder();

        }

    }

    public void traversePreOrder() {

*if* (root != null) {

            root.traversePreOrder();

        }

    }

    public void traversePostOrder() {

*if* (root != null) {

            root.traversePostOrder();

        }

    }

    public int max() {

*if* (root == null) {

*return* Integer.MAX\_VALUE;

        } *else* {

*return* root.max();

        }

    }

    public int count() {

*if* (root == null) {

*return* 0;

        } *else* {

*return* root.count();

        }

    }

    public int min() {

*if* (root == null) {

*return* Integer.MIN\_VALUE;

        } *else* {

*return* root.min();

        }

    }

}

**Class TreeNode**

public class TreeNode {

    private int data;

    private TreeNode leftChild;

    private TreeNode rightChild;

    public TreeNode(int data) {

*this*.data = data;

    }

    public void insert(int value) {

*if* (value == data) {

*return*;

        }

*if* (value < data) {

*if* (leftChild == null) {

                leftChild = *new* TreeNode(value);

            } *else* {

                leftChild.insert(value);

            }

        }

*if* (value > data) {

*if* (rightChild == null) {

                rightChild = *new* TreeNode(value);

            } *else* {

                rightChild.insert(value);

            }

        }

    }

    public int max() {

*if* (getRightChild() == null) {

*return* data;

        } *else* {

*return* getRightChild().max();

        }

    }

    public int min() {

*if* (getLeftChild() == null) {

*return* data;

        } *else* {

*return* getLeftChild().min();

        }

    }

    public TreeNode search(int value) {

*if* (value == data) {

*return* *this*;

        }

*if* (value < data) {

*if* (leftChild != null) {

*return* leftChild.search(value);

            }

        }

*else* {

*if* (rightChild != null) {

*return* rightChild.search(value);

            }

        }

*return* null;

    }

    public void traversePreOrder() {

        System.out.print(data + ", ");

*if* (leftChild != null) {

            leftChild.traversePreOrder();

        }

*if* (rightChild != null) {

            rightChild.traversePreOrder();

        }

    }

    public void traverseInOrder() {

*if* (leftChild != null) {

            leftChild.traverseInOrder();

        }

        System.out.print(data + ", ");

*if* (rightChild != null) {

            rightChild.traverseInOrder();

        }

    }

    public void traversePostOrder() {

*if* (leftChild != null) {

            leftChild.traversePostOrder();

        }

*if* (rightChild != null) {

            rightChild.traversePostOrder();

        }

        System.out.print(data + ", ");

    }

    public int count() {

        TreeNode right = getRightChild();

        TreeNode left = getLeftChild();

        int c = 1;

*if* (right != null)

            c += right.count();

*if* (left != null)

            c += left.count();

*return* c;

    }

    public int getData() {

*return* data;

    }

    public void setData(int data) {

*this*.data = data;

    }

    public TreeNode getLeftChild() {

*return* leftChild;

    }

    public void setLeftChild(TreeNode leftChild) {

*this*.leftChild = leftChild;

    }

    public TreeNode getRightChild() {

*return* rightChild;

    }

    public void setRightChild(TreeNode rightChild) {

*this*.rightChild = rightChild;

    }

    @Override

    public String toString() {

*return* "TreeNode [data=" + data + ", leftChild=" + leftChild

                + ", rightChild=" + rightChild + "]";

    }

}

**Main**

public class Main {

    public static void main(String[] args) {

        TreeNode node01 = *new* TreeNode(25);

        TreeNode node02 = *new* TreeNode(20);

        TreeNode node03 = *new* TreeNode(15);

        TreeNode node04 = *new* TreeNode(27);

        Tree intTree = *new* Tree();

        intTree.insert(25);

        intTree.insert(20);

        intTree.insert(15);

        intTree.insert(27);

        intTree.insert(30);

        intTree.insert(29);

        intTree.insert(26);

        intTree.insert(22);

        intTree.insert(32);

        System.out.println("Goc Trai Phai");

        intTree.traversePreOrder();

        System.out.println("\nTrai Goc Phai");

        intTree.traverseInOrder();

        System.out.println("\nTrai Phai Goc");

        intTree.traversePostOrder();

        System.out.println();

        System.out.println(intTree.search(15));

        System.out.println("MIN: " + intTree.min());

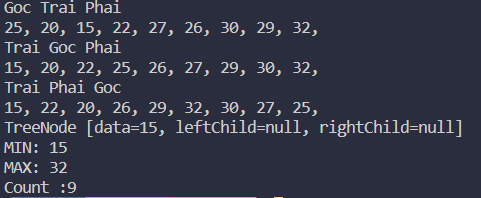
        System.out.println("MAX: " + intTree.max());

        System.out.println("Count :" + intTree.count());

    }

}

**Kết quả:**

****