

## BÀI 12. MỘT SỐ CẤU TRÚC DỮ LIỆU TRONG JAVA

---

1

### Nội dung

- Danh sách liên kết (Linked List)
- Ngăn xếp (Stack)
- Hàng đợi (Queue)
- Cây (Tree)

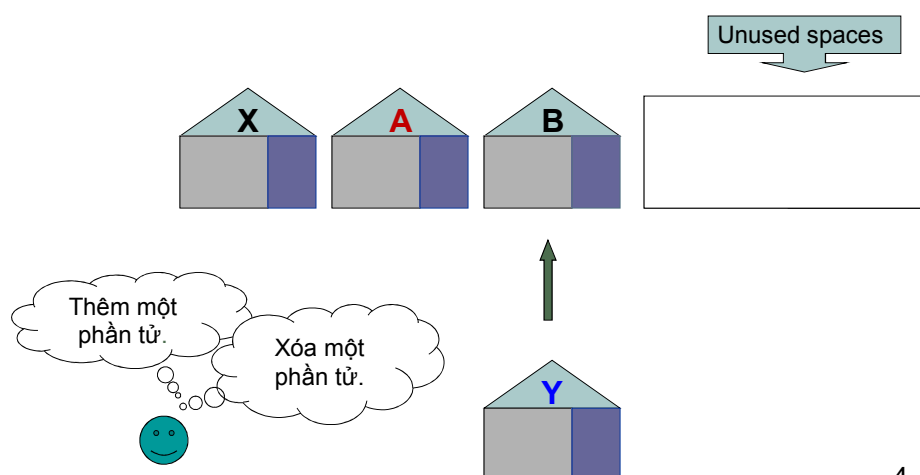
2

## 1. DANH SÁCH LIÊN KẾT (LINKED-LIST)

3

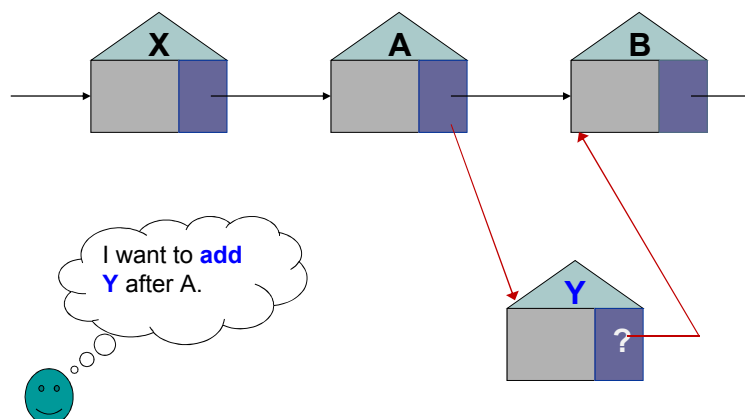
## Mảng vs Danh sách liên kết(DSLK)

- Hạn chế của mảng



4

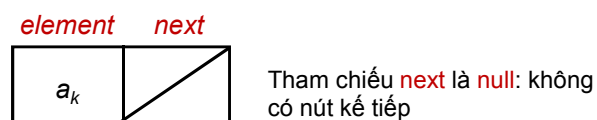
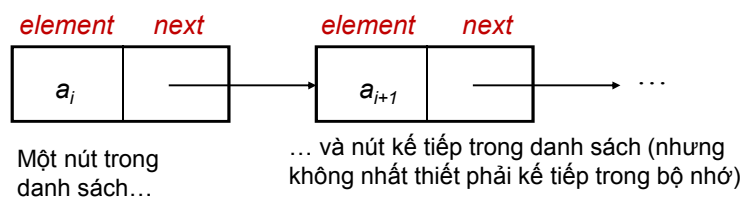
## Mảng vs Danh sách liên kết



5

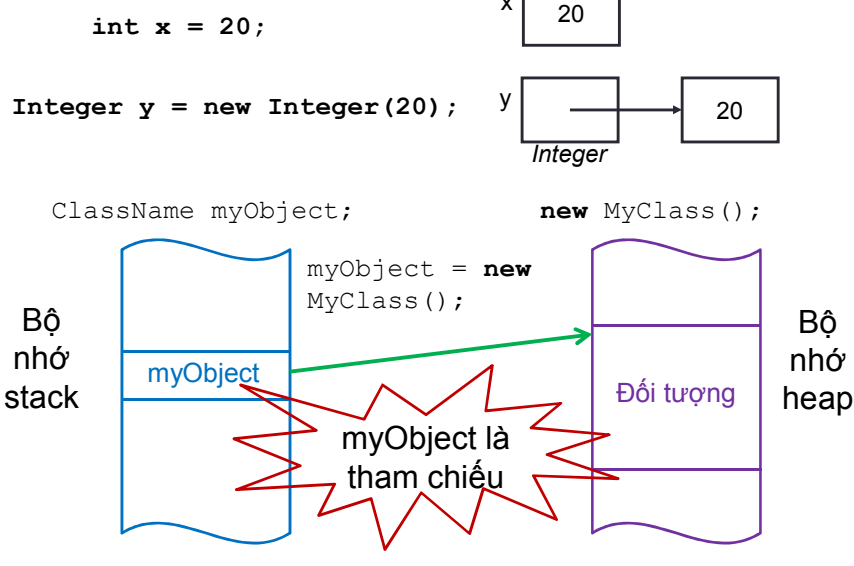
## Ý tưởng xây dựng DSLK

- Mỗi phần tử trong danh sách, gọi là một **nút**, chứa một tham chiếu trở đến nút tiếp theo
- Các phần tử không nằm kế tiếp nhau trên bộ nhớ:
  - Mảng: Các phần tử nằm kế tiếp nhau trên bộ nhớ



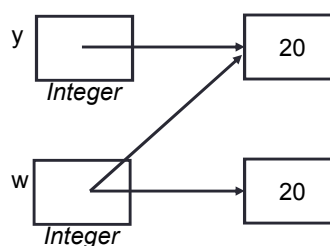
6

## Nhắc lại: Tham chiếu



## Nhắc lại: Tham chiếu(tiếp)

```
Integer y = new Integer(20);
Integer w;
w = new Integer(20);
if (w == y)
    System.out.println("1. w == y");
w = y;
if (w == y)
    System.out.println("2. w == y");
```



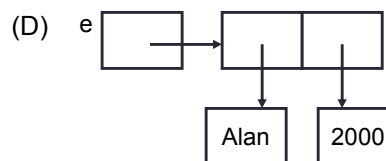
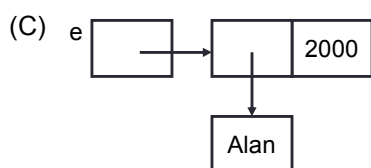
- Kết quả hiển thị là gì?

## Nhắc lại (tham chiếu)

- Mô tả nào là đúng về e trên bộ nhớ

```
class Employee {
    private String name;
    private int salary;
}
```

Employee e = new Employee("Alan", 2000);



9

## Xây dựng DSLK trên Java

- Sử dụng kỹ thuật lập trình tổng quát
- Giao diện `IList<E>` định nghĩa các phương thức

```
import java.util.*;

public interface IList <E> {
    public boolean isEmpty();
    public int size();
    public E getFirst() throws NoSuchElementException;
    public boolean contains(E item);
    public void addFirst(E item);
    public E removeFirst()
        throws NoSuchElementException;
    public void print();
}
```

IList.java

10

## ListNode

ListNode.java

```
class ListNode <E> {
    /* data attributes */
    private E element;
    private ListNode <E> next;

    /* constructors */
    public ListNode(E item) { this(item, null); }
    public ListNode(E item, ListNode <E> n) {
        element = item;
        next = n;
    }

    /* get the next ListNode */
    public ListNode <E> getNext() { return next; }

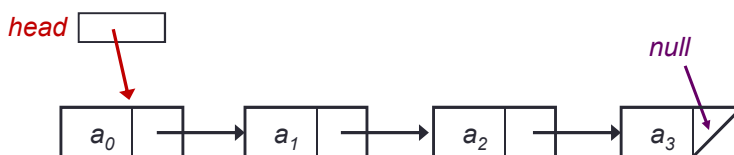
    /* get the element of the ListNode */
    public E getElement() { return element; }

    /* set the next reference */
    public void setNext(ListNode <E> n) { next = n; }
}
```

11

## Xây dựng DSLK

- Giả sử danh sách có 4 phần tử  $\langle a_0, a_1, a_2, a_3 \rangle$
- **head** trỏ đến phần tử đầu tiên trong danh sách
  - Khi duyệt danh sách: bắt đầu từ **head**



BasicLinkedList.java

```
import java.util.*;
class BasicLinkedList <E> implements IList<E> {
    private ListNode <E> head = null;
    private int num_nodes = 0;

    //Khai báo các phương thức ...
}
```

12

## Xây dựng DSLK

BasicLinkedList.java

```
import java.util.*;
class BasicLinkedList <E> implements IList<E> {
    private ListNode <E> head = null;
    private int num_nodes = 0;

    public boolean isEmpty() { return (num_nodes == 0); }

    public int size() { return num_nodes; }

    public E getFirst() throws NoSuchElementException {
        if (head == null)
            throw new NoSuchElementException("can't get from an empty list");
        else return head.getElement();
    }

    public boolean contains(E item) {
        for (ListNode <E> n = head; n != null; n = n.getNext())
            if (n.getElement().equals(item)) return true;
        return false;
    }
}
```

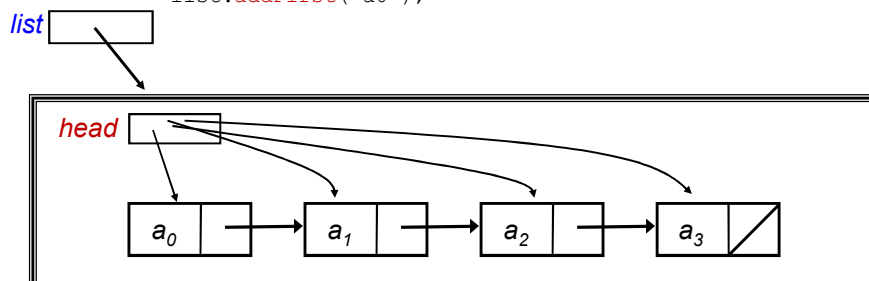
13

## addFirst(): thêm 1 phần tử vào DS

- Thêm ở đầu


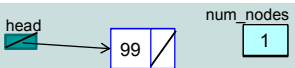

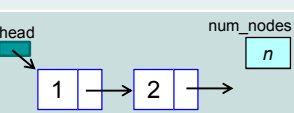
```
BasicLinkedList <String> list = new
    BasicLinkedList <String>();

list.addFirst("a3");
list.addFirst("a2");
list.addFirst("a1");
list.addFirst("a0");
```



14


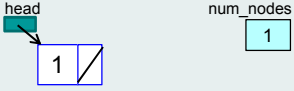
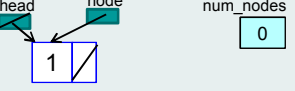
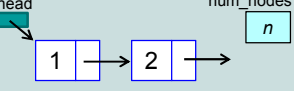
## addFirst()

DSLK	Trước khi thêm	Sau khi thêm: list.addFirst(99)
Rỗng		
1 nút		
nhiều nút		

```
public void addFirst(E item) {
    head = new ListNode<E> (item, head);
    num_nodes++;
}
```

15

## removeFirst(): Xóa một phần tử

DSLK	Before: list	After: list.removeFirst()
rỗng		can't remove
1 nút		
nhiều nút		

```
public E removeFirst() throws NoSuchElementException {
    ListNode<E> node;
    if (head == null)
        throw new NoSuchElementException("can't remove");
    else {
        node = head; head = head.getNext(); num_nodes--;
        return node.getElement();
    }
}
```



## print() Hiển thị danh sách

BasicLinkedList.java

```
public void print() throws NoSuchElementException {
    if (head == null)
        throw new NoSuchElementException("Nothing to
                                         print...");

    ListNode <E> ln = head;
    System.out.print("List is: " + ln.getElement());
    for (int i=1; i < num_nodes; i++) {
        ln = ln.getNext();
        System.out.print(", " + ln.getElement());
    }
    System.out.println(".");
}
```

17

## Collections Framework: LinkedList

- Là lớp triển khai của giao diện `List` trong Collections Framework
  - Danh sách 2 chiều
- Các phương thức triển khai từ `List`: `add()`, `clear()`, `contains()`, `remove()`, `size()`, `toArray()`...
- Các phương thức riêng của `LinkedList`
  - `void addFirst(E e)`: thêm vào đầu danh sách
  - `void addLast(E e)`: thêm vào cuối danh sách
  - `Iterator descendingIterator()`: trả về `Iterator` để duyệt danh sách từ cuối lên
  - `E element()`: trả về đối tượng ở đầu danh sách
  - `E get(int index)`: trả về đối tượng ở vị trí xác định bởi `index`
  - `listIterator(int index)`: trả về `Iterator` để duyệt từ vị trí `index`

18

## LinkedList – Các phương thức

- E getFirst()
- E getLast()
- E removeFirst()
- E removeLast()
- void push(E e): tương tự addFirst()
- E pop(): tương tự removeFirst()
- E peek(): tương tự getFirst()
- E peekFirst(): tương tự getFirst()
- E peekLast(): tương tự getLast()

19

## LinkedList – Ví dụ

TestLinkedListAPI.java

```
import java.util.*;

public class TestLinkedListAPI {

    static void printList(LinkedList <Integer> alist) {
        System.out.print("List is: ");
        for (int i = 0; i < alist.size(); i++)
            System.out.print(alist.get(i) + "\t");
        System.out.println();
    }

    // Print elements in the list and also delete them
    static void printListv2(LinkedList <Integer> alist) {
        System.out.print("List is: ");
        while (alist.size() != 0) {
            System.out.print(alist.element() + "\t");
            alist.removeFirst();
        }
        System.out.println();
    }
}
```

20

## LinkedList – Ví dụ(tiếp)

TestLinkedListAPI.java

```
public static void main(String [] args) {
    LinkedList <Integer> alist = new LinkedList <Integer> ();
    for (int i = 1; i <= 5; i++)
        alist.add(new Integer(i));

    printList(alist);

    System.out.println("First element: " + alist.getFirst());

    System.out.println("Last element: " + alist.getLast());

    alist.addFirst(888);
    alist.addLast(999);
    printListv2(alist);
    printList(alist);
}
}
```

21

## Bài tập

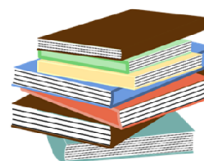
- Viết lại hai phương thức contain() và print() bằng kỹ thuật đệ quy
- Hãy tạo danh sách liên kết 2 chiều

22

## 2. NGĂN XẾP (STACK)

---

**Last-In-First-Out (LIFO)**



23

## Ngăn xếp(stack) là gì?

- Ngăn xếp: tập hợp các phần tử với cách thức truy cập Last-In-First-Out (LIFO)
- Các phương thức:
  - push(): thêm 1 phần tử vào đỉnh ngăn xếp
  - pop(): lấy và xóa 1 phần tử ra khỏi ngăn xếp
  - peek(): lấy một phần tử ở đỉnh ngăn xếp
- Ứng dụng:
  - Hệ thống: Gọi phương thức, hàm, xử lý ngắt
  - Đệ quy
  - Kiểm tra cặp dấu ngoặc “”, “”, ( ), { }...
  - Tính toán biểu thức...

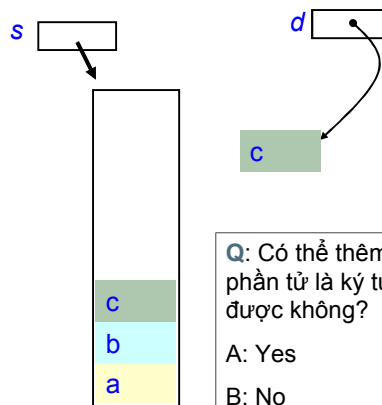
24

## Hoạt động của ngăn xếp

```

→ Stack s = new Stack();
→ s.push ("a");
→ s.push ("b");
→ s.push ("c");
→ d = s.peek ();
→ s.pop ();
→ s.push ("e");
→ s.pop ();

```



Q: Có thể thêm vào phần tử là ký tự 'f' được không?

A: Yes

B: No

25

## Xây dựng ngăn xếp trong Java

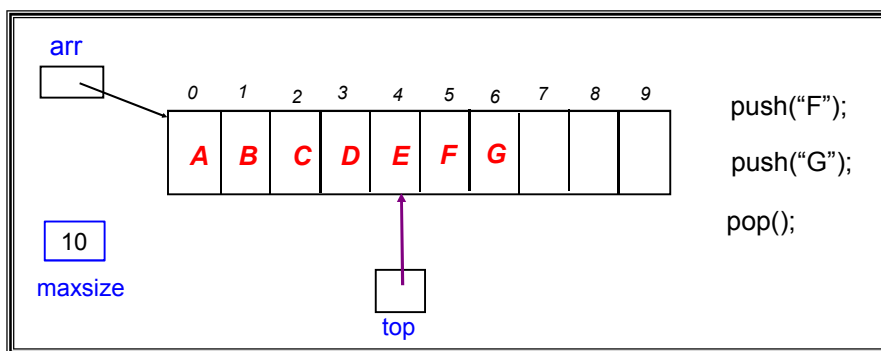
- Sử dụng mảng (Array)
- Sử dụng danh sách liên kết (Linked List)
- Lớp Stack trong Collections Framework

26

## Ngăn xếp – Sử dụng mảng

- Tham chiếu **top** trỏ vào đỉnh của ngăn xếp

### StackArr



27

## Ngăn xếp – Sử dụng mảng

```
import java.util.*;

public interface IStack <E> {
    // check whether stack is empty
    public boolean empty();

    // retrieve topmost item on stack
    public E peek() throws EmptyStackException;

    // remove and return topmost item on stack
    public E pop() throws EmptyStackException;

    // insert item onto stack
    public void push(E item);
}
```

IStack.java

28

## Ngăn xếp – Sử dụng mảng

```
import java.util.*;

class StackArr <E> implements IStack <E> {
    private E[] arr;
    private int top;
    private int maxSize;
    private final int INITSIZE = 1000;

    public StackArr() {
        arr = (E[]) new Object[INITSIZE]; // creating array of type E
        top = -1; // empty stack - thus, top is not on a valid array element
        maxSize = INITSIZE;
    }

    public boolean empty() {
        return (top < 0);
    }
}
```

StackArr.java

29

## Ngăn xếp – Sử dụng mảng

```
public E peek() throws EmptyStackException {
    if (!empty()) return arr[top];
    else throw new EmptyStackException();
}

public E pop() throws EmptyStackException {
    E obj = peek();
    top--;
    return obj;
}
```

StackArr.java

30

## Ngăn xếp – Sử dụng mảng (tiếp)

```

public void push(E obj) {
    if (top >= maxSize - 1) enlargeArr(); //array is full, enlarge it
    top++;
    arr[top] = obj;
}

private void enlargeArr() {
    // When there is not enough space in the array
    // we use the following method to double the number
    // of entries in the array to accommodate new entry
    int newSize = 2 * maxSize;
    E[] x = (E[]) new Object[newSize];

    for (int j=0; j < maxSize; j++) {
        x[j] = arr[j];
    }
    maxSize = newSize;
    arr = x;
}

```

StackArr.java

31

## Ngăn xếp – Sử dụng DSLK

```

import java.util.*;

class StackLL <E> implements IStack<E> {
    private BasicLinkedList <E> list;

    public StackLL() {
        list = new BasicLinkedList <E> ();
    }

    public boolean empty() { return list.isEmpty(); }

    public E peek() throws EmptyStackException {
        try {
            return list.getFirst();
        } catch (NoSuchElementException e) {
            throw new EmptyStackException();
        }
    }
}

```

StackLL.java

32



## Ngăn xếp – Sử dụng DSLK(tiếp)

StackLL.java

```
public E pop() throws EmptyStackException {
    E obj = peek();
    list.removeFirst();
    return obj;
}

public void push(E o) {
    list.addFirst(o);
}
}
```

33

## Ngăn xếp – Kế thừa từ DSLK

StackLLE.java

```
import java.util.*;

class StackLLE <E> extends BasicLinkedList <E> implements IStack<E> {
    public boolean empty() { return isEmpty(); }

    public E peek() throws EmptyStackException {
        try {
            return getFirst();
        } catch (NoSuchElementException e) {
            throw new EmptyStackException();
        }
    }

    public E pop() throws EmptyStackException {
        E obj = peek();
        removeFirst();
        return obj;
    }

    public void push (E o) { addFirst(o); }
}
```

34

## Ngăn xếp – Lớp Stack

- Là một lớp kế thừa từ lớp Vector trong Collections Framework
- Các phương thức kế thừa từ Vector: `add()`, `clear()`, `contains()`, `remove()`, `size()`, `toArray()`...
- Các phương thức riêng của Stack:
  - `boolean empty()`
  - `E peek()`
  - `E pop()`
  - `E push()`
  - `int search (Object)`

35

## Ngăn xếp – Ví dụ

StackDemo.java

```
import java.util.*;
public class StackDemo {
    public static void main (String[] args) {

        StackArr <String> stack = new StackArr <String>();
        //StackLL <String> stack = new StackLL <String>();
        //StackLLE <String> stack = new StackLLE <String>();
        //Stack <String> stack = new Stack <String>();

        System.out.println("stack is empty? " + stack.empty());
        stack.push("1");
        stack.push("2");
        System.out.println("top of stack is " + stack.peek());
        stack.push("3");
        System.out.println("top of stack is " + stack.pop());
        stack.push("4");
        stack.pop();
        stack.pop();
        System.out.println("top of stack is " + stack.peek());
    }
}
```

36

## Ứng dụng – Kiểm tra dấu ngoặc

- Trên biểu thức, câu lệnh sử dụng dấu ngoặc phải đảm bảo các dấu ngoặc đủ cặp mở-đóng

Ví dụ: `{a, (b+f[4])*3, d+f[5]}`



- Một số ví dụ về sử dụng dấu ngoặc sai nguyên tắc:

`(...)...)` Thừa dấu đóng

`(...(...` Thừa dấu mở

`{...(...)}...` Không đúng cặp

37

## Ứng dụng – Kiểm tra dấu ngoặc

Khởi tạo ngăn xếp

for mỗi ký tự trong biểu thức

```
{
  if là dấu mở then
    push()
  if nếu là dấu đóng then
    pop()
    if ngăn xếp rỗng hoặc dấu đóng không đúng cặp
    then báo lỗi
}
```

if stack không rỗng then báo lỗi

Example

`{ a - ( b + f [ 4 ] ) * 3 * d + f [ 5 ] }`



[	]
(	)
{	}

Ngăn xếp

38

## Bài tập

- Sử dụng ngăn xếp để tính giá trị biểu thức

39

### 3. HÀNG ĐỢI (QUEUE)

---

First-In-First-Out (FIFO)



40

## Hàng đợi (queue) là gì?

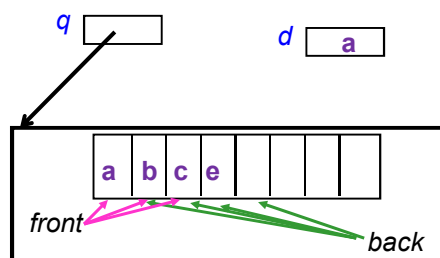
- Hàng đợi: Tập hợp các phần tử với cách thức truy cập First-In-First-Out(FIFO)
- Các phương thức:
  - offer(): đưa một phần tử vào hàng đợi
  - poll(): đưa một phần tử ra khỏi hàng đợi
  - peek(): lấy một phần tử trong hàng đợi
- Ứng dụng:
  - Hàng đợi chờ tài nguyên phục vụ
  - Duyệt theo chiều rộng trên cây
  - ...

41

## Hoạt động của hàng đợi

```

Queue q = new Queue ();
→ q.offer ("a");
→ q.offer ("b");
→ q.offer ("c");
→ d = q.peek ();
→ q.poll ();
→ q.offer ("e");
→ q.poll ();
  
```

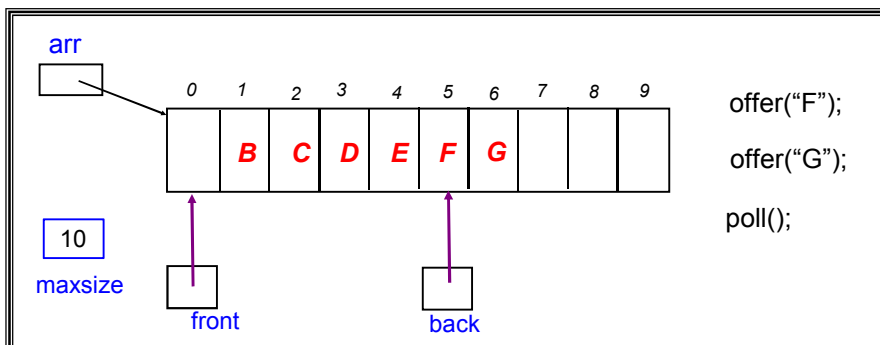


42

## Hàng đợi – Sử dụng mảng

- Sử dụng hai tham chiếu **front** và **back**

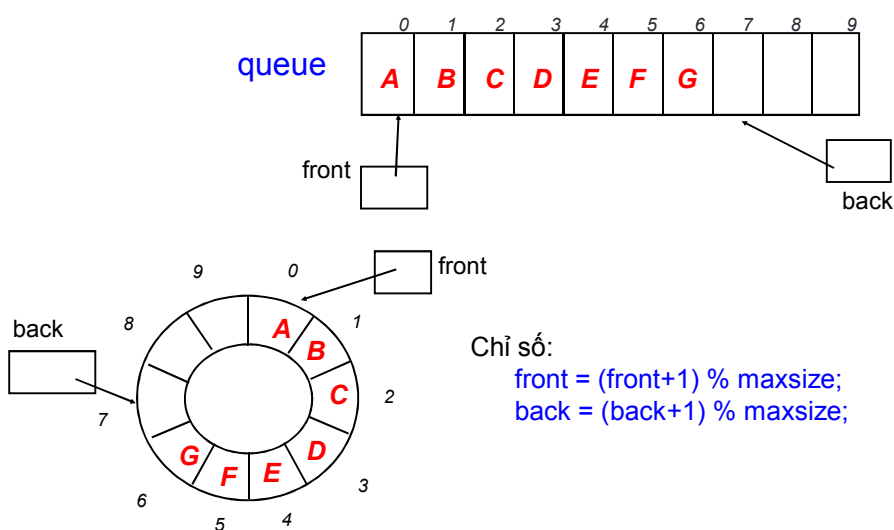
**QueueArr**



Làm thế nào để sử dụng lại các vị trí trống?

43

## Hàng đợi – Sử dụng mảng



44

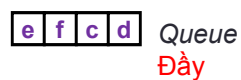
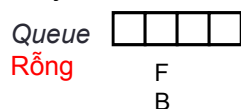
## Hàng đợi – Sử dụng mảng

- Câu hỏi: khi nào  $\text{back} == \text{front}$ 
  - a) Hàng đợi đầy
  - b) Hàng đợi rỗng
  - c) Cả A và B đều đúng
  - d) Cả A và B đều sai

45

## Hàng đợi – Sử dụng mảng

- Nhập nhằng giữa 2 trường hợp hàng đợi rỗng và hàng đợi đầy



- Giải pháp 1: sử dụng giá trị size lưu số phần tử trong hàng đợi
  - $\text{size} = 0$ : hàng đợi rỗng
- Giải pháp 2: khi hàng đợi chỉ còn một chỗ trống thì coi là đầy
  - Hàng đợi rỗng:  $F == B$
  - Hàng đợi đầy:  $F == (B+1) \% \text{maxsize}$



46

## Hàng đợi – Sử dụng mảng

```
import java.util.*;

public interface IQueue <E> {

    // return true if queue has no elements
    public boolean isEmpty();

    // return the front of the queue
    public E peek();

    // remove and return the front of the queue
    public E poll();

    // add item to the back of the queue
    public boolean offer(E item);
}
```

IQueue.java

47

## Hàng đợi – Sử dụng mảng(tiếp)

```
import java.util.*;

class QueueArr <E> implements IQueue <E> {
    private E [] arr;
    private int front, back;
    private int maxSize;
    private final int INITSIZE = 1000;

    public QueueArr() {
        arr = (E []) new Object[INITSIZE]; // create array of E
                                           // objects
        front = 0; // the queue is empty
        back = 0;
        maxSize = INITSIZE;
    }

    public boolean isEmpty() {
        return (front == back);
    }
}
```

QueueArr.java

48



## Hàng đợi – Sử dụng mảng (tiếp)

QueueArr.java

```

public E peek() { // return the front of the queue
    if (isEmpty()) return null;
    else return arr[front];
}

public E poll() { // remove and return the front of the queue
    if (isEmpty()) return null;
    E obj = arr[front];
    arr[front] = null;
    front = (front + 1) % maxSize; // "circular" array
    return obj;
}

public boolean offer(E o) { // add item to the back of the queue
    if ((back+1)%maxSize == front) // array is full
        return false;
    arr[back] = o;
    back = (back + 1) % maxSize; // "circular" array
    return true;
}
}

```

49

## Hàng đợi – Ví dụ

QueueDemo.java

```

import java.util.*;
public class QueueDemo {
    public static void main (String[] args) {

        QueueArr <String> queue= new QueueArr <String> ();

        System.out.println("queue is empty? " + queue.isEmpty());
        queue.offer("1");
        System.out.println("operation: queue.offer(\"1\")");
        System.out.println("queue is empty? " + queue.isEmpty());
        System.out.println("front now is: " + queue.peek());
        queue.offer("2");
        System.out.println("operation: queue.offer(\"2\")");
        System.out.println("front now is: " + queue.peek());
        queue.offer("3");
        System.out.println("operation: queue.offer(\"3\")");
        System.out.println("front now is: " + queue.peek());
    }
}

```

50

## Hàng đợi – Ví dụ (tiếp)

QueueDemo.java

```
queue.poll();
System.out.println("operation: queue.poll()");
System.out.println("front now is: " + queue.peek());
System.out.print("checking whether queue.peek().equals(\"1\"): ");
System.out.println(queue.peek().equals("1"));
queue.poll();
System.out.println("operation: queue.poll()");
System.out.println("front now is: " + queue.peek());
queue.poll();
System.out.println("operation: queue.poll()");
System.out.println("front now is: " + queue.peek());
}
```

51

## Hàng đợi trong Collections Framework

- Giao diện `Queue`: Kế thừa từ giao diện `Collection` trong Collections Framework
  - Giao diện con: `Deque`
- Các phương thức cần triển khai:
  - `boolean add(E)`
  - `E element()`: lấy phần tử đầu tiên trong hàng đợi
  - `boolean offer(E)`
  - `E peek()`
  - `E poll()`
  - `E remove()`: lấy và xóa phần tử đầu tiên trong hàng đợi

52

## Hàng đợi trong Collections Framework

- Lớp `PriorityQueue`: hàng đợi có ưu tiên dựa trên sự sắp xếp lại các nút
- Lớp `DelayQueue`: Hàng đợi có hỗ trợ thiết lập thời gian chờ cho phương thức `poll()`
- Giao diện `BlockingQueue`:
  - `offer()`, `add()`, `put()`: chờ đến khi hàng đợi có chỗ thì thực thi
  - `poll()`, `remove()`, `take()`: chờ đến khi hàng đợi không rỗng thì thực thi
- Lớp `LinkedBlockingQueue`: xây dựng hàng đợi dựa trên nút của DSLK
- Lớp `ArrayBlockingQueue`: xây dựng hàng đợi dựa trên mảng

53

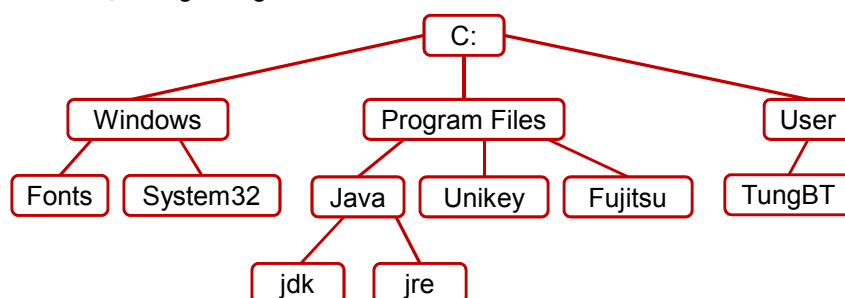
## 4. CÂY(TREE)

---

54

## Cây

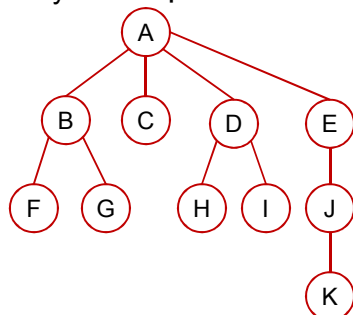
- Một tập các nút tổ chức theo cấu trúc phân cấp
- Mỗi quan hệ giữa các nút: cha-con
- Ứng dụng:
  - Cây thư mục
  - Sơ đồ tổ chức
  - Hệ thống thông tin tên miền...



55

## Cây – Các khái niệm cơ bản

- Gốc: nút không có nút cha (A)
- Nút nhánh: các nút có tối thiểu 1 nút con (B, D, E, J)
- Nút lá: nút không có nút con (C, F, G, H, I, K)
- Kích thước: tổng số nút trên cây (11)
- Độ sâu của một nút: số nút trên đường đi từ nút gốc
- Độ cao của cây: độ dài đường đi từ gốc tới nút sâu nhất
- Cây con: một nút nhánh và tất cả con cái của nó



Nút	Độ sâu	Chiều cao
A	0	3
B	1	1
C	1	0
E	1	2
F	2	0
J	2	1
K	3	0

56

## Cây và các thuật toán đệ quy

- Các thuật toán đệ quy có thể cài đặt đơn giản và làm việc hiệu quả trên cấu trúc cây

- Tính kích thước:

`size (Cây) = 1 + size(Cây con trái) + size (Cây con phải)`

- Tính chiều cao:

`height(Cây) =`

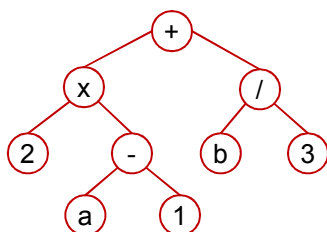
`1 + Max(height(Cây con trái), height(Cây con phải))`

- ...

57

## Cây nhị phân

- Là cây mà mỗi nút không có quá 2 con: nút con trái và nút con phải
  - Cây nhị phân đầy đủ: mỗi nút có đúng 2 nút con
- Cây con trái: gồm nút con trái và toàn bộ con cái
- Cây con phải: gồm nút con phải và toàn bộ con cái
- Định nghĩa đệ quy: cây nhị phân là cây có một nút gốc và hai cây con trái và con phải là cây nhị phân
- Ứng dụng: cây nhị phân biểu thức, cây nhị phân tìm kiếm



Cây biểu diễn biểu thức:  
 $2x(a - 1) + b/3$

58

## Xây dựng cây nhị phân

```
public interface IBinaryTree<E> {
    //Check whether tree is empty
    public boolean isEmpty();
    //Remove all of nodes
    public void clear();
    //Return the size of the tree
    public int size();
    //Return the height of the tree
    public int height();
    //Visit tree using in-order traversal
    public void visitInOrder();
    //Visit tree using pre-order traversal
    public void visitPreOrder();
    //Visit tree using pos-order traversal
    public void visitPosOrder();
}
```

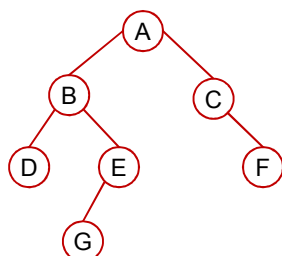
IBinaryTree.java

59

## Xây dựng cây nhị phân trên Java

- Giải pháp 1: sử dụng mảng để lưu trữ các nút của cây

- Chỉ số nút:  $i$
- Chỉ số nút cha (nếu có):  $(i-1)/2$
- Chỉ số nút con trái (nếu có):  $2*i + 1$
- Chỉ số nút con phải (nếu có):  $2*i + 2$



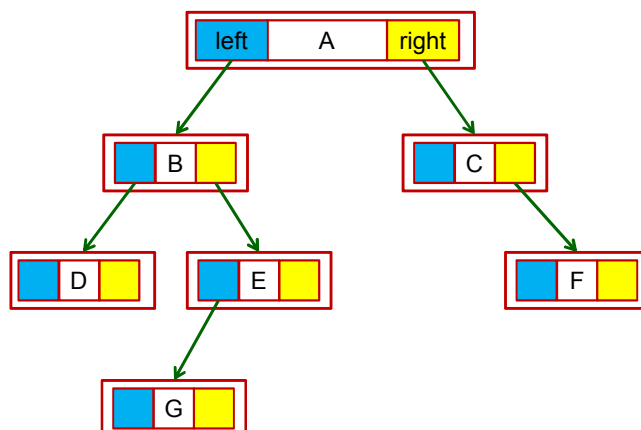
- Không hiệu quả

index	item	left	right
0	A	1	2
1	B	3	4
2	C	-1	6
3	D	-1	-1
4	E	9	-1
5	null	-1	-1
6	F	-1	-1
7	null	-1	-1
8	null	-1	-1
9	G	-1	-1
10	null	-1	-1

60

## Xây dựng cây nhị phân trên Java

- Giải pháp 2: Sử dụng danh sách liên kết
- Mỗi nút có 2 tham chiếu trỏ đến con trái và con phải



61

## Xây dựng cây nhị phân trên Java

```

public class BinaryNode<E> {
    private E element;
    private BinaryNode<E> left;
    private BinaryNode<E> right;

    //Constructors
    public BinaryNode() {
        this(null, null, null);
    }

    public BinaryNode(E item) {
        this(item, null, null);
    }

    public BinaryNode(E item, BinaryNode<E> l, BinaryNode<E> r) {
        element = item;
        left = l;
        right = r;
    }
}

```

BinaryNode.java

62

## Xây dựng cây nhị phân trên Java(tiếp)

BinaryNode.java

```
//getter methods...
//Return true if has left child
public static <E> boolean hasLeft(BinaryNode<E> t){
    return t.left != null;
}

// Return true if has right child
public static <E> boolean hasRight(BinaryNode<E> t){
    return t.right != null;
}

// Add left child
public void addLeft(BinaryNode<E> l){
    left = l;
}

//Add right child
public void addRight(BinaryNode<E> r){
    right = r;
}
```

63

## Xây dựng cây nhị phân trên Java(tiếp)

BinaryTree.java

```
public class BinaryTree<E> implements IBinaryTree<E>{
    private BinaryNode<E> root;

    //Constructors
    public BinaryTree(){
        root = null;
    }

    public BinaryTree(E rootItem){
        root = new BinaryNode<E>(rootItem, null, null);
    }

    //getter methods
    public BinaryNode<E> getRoot(){ return root; }

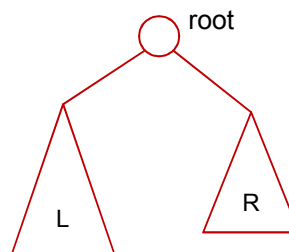
    //setter methods
    public void setRoot(BinaryNode<E> r){ root = r;}
    public boolean isEmpty() { return root == null; }
    public void clear() { root = null; }
```

64



## Tính kích thước của cây

- Sử dụng đệ quy
- Trường hợp cơ sở:  
nếu  $root == null$  thì  $S_T = 0$
- Bước đệ quy:  
 $S_T = 1 + S_L + S_R$ 
  - $S_T$ : Kích thước của cây
  - $S_L$ : Kích thước của cây con trái
  - $S_R$ : Kích thước của cây con phải



BinaryNode.tree

```
//Return the size of the binary tree
public int size(){
    return size(root);
}

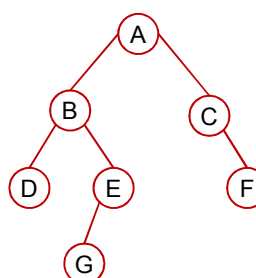
private int size(BinaryNode<E> n){
    if(n == null) return 0;
    else return 1 + size(n.getLeft()) + size(n.getRight());
}
```

65

## Đệ quy khi tính kích thước của cây

Ngăn xếp gọi phương thức

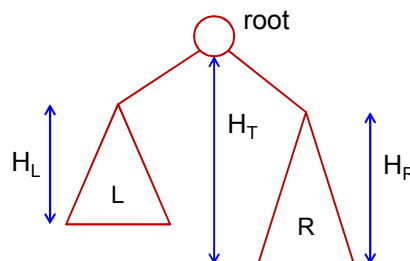
size <sub>G</sub> ()	return 1
size <sub>F</sub> ()	return 1 + size <sub>G</sub> ()
size <sub>C</sub> ()	return 1 + size <sub>E</sub> ()
size <sub>A</sub> ()	return 1 + size <sub>B</sub> ()



66

## Tính chiều cao của cây

- Sử dụng đệ quy
- Trường hợp cơ sở:  
nếu  $root == null$  thì  $H_T = -1$
- Bước đệ quy  
 $H_T = 1 + \max(H_L, H_R)$ 
  - $H_T$ : Chiều cao của cây
  - $H_L$ : Chiều cao của cây con trái
  - $H_R$ : Chiều cao của cây con phải



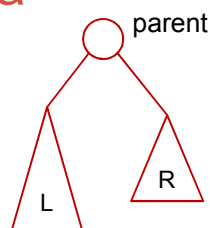
BinaryTree.java

```
//Return the size of the binary tree rooted at n
public int height() {
    return height(root);
}
private int height(BinaryNode<E> n) {
    if (n == null) return -1;
    else return 1 + Math.max(height(n.getLeft()),
                             height(n.getRight()));
}
```

67

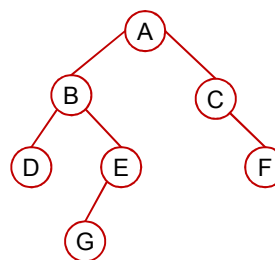
## Duyệt cây theo thứ tự giữa

- Duyệt cây theo thứ tự giữa (in order): sử dụng đệ quy
  - Nếu có con trái, duyệt con trái
  - Duyệt nút cha
  - Nếu có con phải, duyệt con phải
- Ví dụ: D, B, G, E, A, C, F



BinaryTree.java

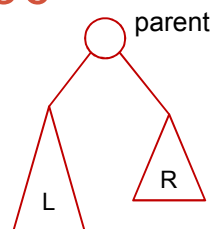
```
public void visitInOrder() {
    visitInOrder(root);
}
private void visitInOrder(BinaryNode<E> n) {
    if (n.hasLeft())
        visitInOrder(n.getLeft());
    System.out.println(n.getElement());
    if (n.hasRight())
        visitInOrder(n.getRight());
}
```



68

## Duyệt cây theo thứ tự trước

- Duyệt cây theo thứ tự trước(pre order): sử dụng đệ quy
  - Duyệt nút cha
  - Nếu có con trái, duyệt con trái
  - Nếu có con phải, duyệt con phải
- Ví dụ: A, B, D, E, G, C, F

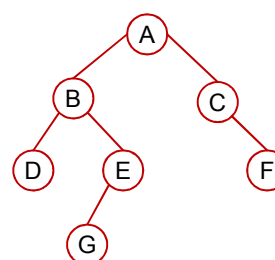


```

public void visitPreOrder() {
    visitPreOrder(root);
}
private void visitPreOrder(BinaryNode<E> n) {
    System.out.println(n.getElement());
    if(n.hasLeft())
        visitPreOrder(n.getLeft());
    if(n.hasRight())
        visitPreOrder(n.getRight());
}

```

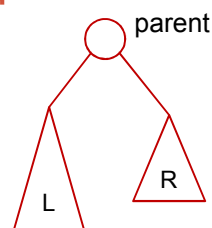
BinaryTree.java



69

## Duyệt cây theo thứ tự sau

- Duyệt cây theo thứ tự trước(pre order): sử dụng đệ quy
  - Nếu có con trái, duyệt con trái
  - Nếu có con phải, duyệt con phải
  - Duyệt nút cha
- Ví dụ: D, G, E, B, F, C, A

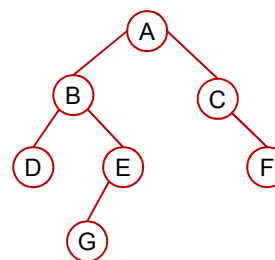


```

public void visitPosOrder() {
    visitPosOrder(root);
}
private void visitPosOrder(BinaryNode<E> n) {
    if(n.hasLeft())
        visitPosOrder(n.getLeft());
    if(n.hasRight())
        visitPosOrder(n.getRight());
    System.out.println(n.getElement());
}

```

BinaryTree.java



70

## Thử nghiệm

BinaryTreeDemo.java

```
public class BinaryTreeDemo {

    public static void main(String[] args) {
        IBinaryTree<String> tree = new BinaryTree<String>("A");

        BinaryNode<String> left = new BinaryNode<String>("B");
        tree.getRoot().addLeft(left);
        left.addLeft(new BinaryNode<String>("D"));
        left.addRight(new BinaryNode<String>("E"));
        BinaryNode<String> right;
        right = left.getRight();
        right.addLeft(new BinaryNode<String>("G"));

        right = new BinaryNode<String>("C");
        tree.getRoot().addRight(right);
        right.addRight(new BinaryNode<String>("F"));
    }
}
```

71

## Thử nghiệm (tiếp)

BinaryTreeDemo.java

```
System.out.println("The size of tree:" + tree.size());

System.out.println("The height of tree:" +
    tree.height());

System.out.println("Visit tree by in-order");
tree.visitInOrder();

System.out.println("Visit tree by pre-order");
tree.visitPreOrder();

System.out.println("Visit tree by pos-order");
tree.visitPosOrder();
    }
}
```

72

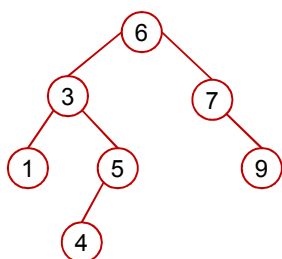
## Bài tập

- Viết các phương thức tìm kiếm trên cây
  - Gợi ý: thực hiện tương tự các phương thức duyệt cây

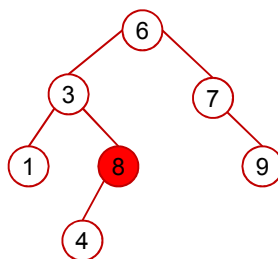
73

## Cây nhị phân tìm kiếm

- Cây nhị phân tìm kiếm:
  - Là cây nhị phân
  - Mọi con trái nhỏ hơn cha
  - Mọi con phải lớn hơn cha
- Cho phép tìm kiếm với độ phức tạp  $O(\log(n))$ 
  - Tìm kiếm trên cây nhị phân thường:  $O(n)$



Cây nhị phân tìm kiếm



Không phải là cây nhị phân tìm kiếm

74

## Xây dựng cây nhị phân tìm kiếm

IBinarySearchTree.java

```
public interface IBinarySearchTree<E>{
    //Insert into a subtree
    public void insert(E item) throws DuplicateItemException;

    //Find a node
    public BinaryNode<E> find(E item);

    //Visit tree using in-order traversal
    public void visitInOrder();

    //Visit tree using pre-order traversal
    public void visitPreOrder();

    //Visit tree using pos-order traversal
    public void visitPosOrder();
}
```

75

## Xây dựng cây nhị phân tìm kiếm

BinarySearchTree.java

```
public class BinaryTree<E> extends BinaryTree<E>
    implement IBinaryTree<E>{
    private BinaryNode<E> root;
    private Comparator<E> comparator;

    //Constructors
    public BinarySearchTree(Comparator c){
        root = null;
        comparator = c;
    }

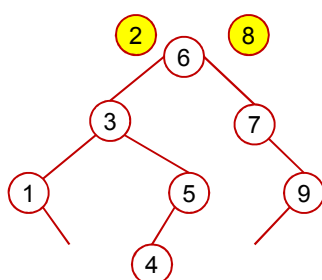
    public BinarySearchTree(BinaryNode<E> r, Comparator c){
        root = r;
        comparator = c;
    }

    //declares other methods...
}
```

76

## Thêm một nút vào cây

- Sử dụng đệ quy
- Trường hợp cơ sở: nếu nút đang duyệt là `null` thêm nút mới vào vị trí đang duyệt
- Bước đệ quy:
  - Nếu nút mới lớn hơn, thêm vào cây con trái
  - Nếu nút mới nhỏ hơn, thêm vào cây con phải
  - Nếu nút mới bằng thông báo lỗi trùng nút



77

## Thêm một nút vào cây

BinarySearchTree.java

```

public void insert(E item) throws DuplicateItemException {
    root = insert(item, root);
}

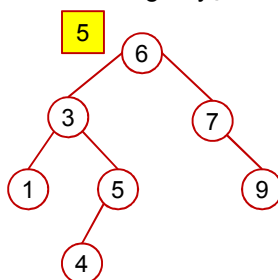
private BinaryNode<E> insert(E item, BinaryNode<E> t)
    throws DuplicateItemException {
    if(t == null)
        t = new BinaryNode<E>(item);
    else if(comparator.compare(item, t.getElement()) < 0)
        t.addLeft(insert(item, t.getLeft()));
    else if(comparator.compare(item, t.getElement()) > 0)
        t.addRight(insert(item, t.getRight()));
    else
        throw new DuplicateItemException(item.toString());
    return t;
}

```

78

## Tìm kiếm trên cây nhị phân

- Sử dụng vòng lặp : khi nút đang duyệt còn khác `null` thực hiện thủ tục đệ quy
- Bước cơ sở: Nếu nút đang duyệt mang giá trị đang tìm kiếm trả lại nút đang duyệt
- Bước đệ quy:
  - Nếu giá trị nhỏ hơn nút đang duyệt, tìm trên cây con trái
  - Nếu giá trị lớn hơn nút đang duyệt, tìm trên cây con phải



79

## Tìm kiếm trên cây nhị phân

BinarySearchTree.java

```

public BinaryNode<E> find(E item) {
    return find(item, root);
}

private BinaryNode<E> find(E item, BinaryNode<E> t) {
    while(t != null){
        if(comparator.compare(item, t.getElement()) < 0)
            t = t.getLeft();
        else if (comparator.compare
                    (item, t.getElement()) > 0)
            t = t.getRight();
        else return t;
    }
    return null;
}
  
```

80



## Ví dụ thử nghiệm

BinarySearchTreeDemo.java

```
public class BinarySearchTreeDemo {
    public static class IntComparator implements
        Comparator<Integer>{

        @Override
        public int compare(Integer o1, Integer o2) {
            return o1.compareTo(o2);
        }
    }

    public static void main(String[] args) throws
        DuplicateItemException{
        Comparator<Integer> c = new IntComparator();
        BinarySearchTree<Integer> tree = new
            BinarySearchTree<Integer>(c);
        tree.insert(6);
        tree.insert(3);
        tree.insert(7);
        tree.insert(1);
        tree.insert(5);
        tree.insert(4);
        tree.insert(9);
    }
}
```

81

## Ví dụ thử nghiệm(tiếp)

BinarySearchTreeDemo.java

```
tree.visitInOrder();

if (tree.find(5) != null)
    System.out.println("Found item!");
else
    System.out.println("Could not found item!");

if (tree.find(8) != null)
    System.out.println("Found item!");
else
    System.out.println("Could not found item!");
}
```

82

## Bài tập

Viết phương thức:

- Tìm nút có giá trị lớn nhất
- Tìm nút có giá trị nhỏ nhất
- Xóa một nút khỏi cây

83

## TreeSet<E>

- `TreeSet<E>` là một lớp trong Collections Framework cài đặt cây tổng quát
- Các nút trên cây là có thứ tự theo định nghĩa của người dùng
- Các phương thức chung: `size()`, `remove()`...
- Các phương thức
  - `TreeSet()`
  - `TreeSet(Comparator comp)` : khởi tạo với bộ so sánh để sắp xếp
  - `TreeSet(Collection c)` : khởi tạo với các nút trong 1 đối tượng Collection
  - `boolean add(E e)` : trả về true nếu thêm 1 nút
  - `boolean remove(E e)` : trả về true nếu xóa được 1 nút

84

## TreeSet<E> - Các phương thức

- `void clear()` : xóa toàn bộ cây
- `boolean contain(Objects o)` : trả về true nếu tìm thấy
- `Iterator<E> descendingIterator()` : trả về Iterator để theo thứ tự giảm dần
- `Iterator<E> iterator()` : trả về Iterator để theo thứ tự tăng dần
- `E first()` : trả về phần tử nhỏ nhất
- `E last()` : trả về phần tử lớn nhất
- `E floor(E e)` : trả về nút lớn nhất còn nhỏ hơn hoặc bằng e trong cây
- `E higher(E e)` : trả về nút nhỏ nhất còn lớn hơn hoặc bằng e trong cây

85

## Tài liệu tham khảo

- Bài giảng sử dụng hình ảnh và mã nguồn minh họa cho các nội dung về DSLK, ngăn xếp và hàng đợi từ bài giảng của Đại học QG Singapore (NUS)
- Nội dung về cây được tham khảo từ sách *"Data Structures & Problem Solving Using Java"*, Mark Allen Weiss

86