Programación de sistemas

Genéricos

Departamento de Ingeniería Telemática

Introducción a los genéricos (*generics*)

- Servicio proporcionado por Java (J2SE 5.0 y superiores)
- Permiten comprobar los tipos a los que pertenecen las instancias durante la compilación
  - Los fallos se producen **en tiempo de compilación** (se garantiza que siempre hay un tipo compatible durante la ejecución)

Introducción a los genéricos (*generics*)

Sin Genéricos

```
List list = new LinkedList();
list.insert("test");
Integer value = (Integer) list.extract();
```

Error en tiempo de ejecución
"ClassCastException".
Se detiene el programa

Con Genéricos

```
List<Integer> list = new LinkedList<Integer>();
list.insert("test");
Integer value = list.extract();
```

Error en tiempo de compilación
"The method insert(Integer) in the type List<Integer> is not applicable for the arguments (String)"
Se detecta a tiempo antes de ejecutar el programa

BY-NC-SA

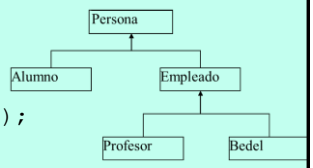
Introducción a los genéricos (*generics*)

Creación de una clase que utiliza genéricos

```
public class LinkedList<E>{
    public void insert (E e){…}
    public E extract(){…}
…
}
```

Uso de una clase que utiliza genéricos

```
LinkedList<Empleado> employees = new
    LinkedList<Empleado>();
employees.insert(new Empleado());
Empleado myEmployee = employees.extract();
employees.insert(new Bedel());
employees.insert(new Persona());
employees.insert(new Object());
```



Errores de compilación

Programación de sistemas

Árboles

Departamento de Ingeniería Telemática

Contenidos

- **Concepto de árbol**
- **Terminología**
- **Implementación**
- Casos especiales
  - Árboles binarios de búsqueda
  - Montículos (*heaps*)

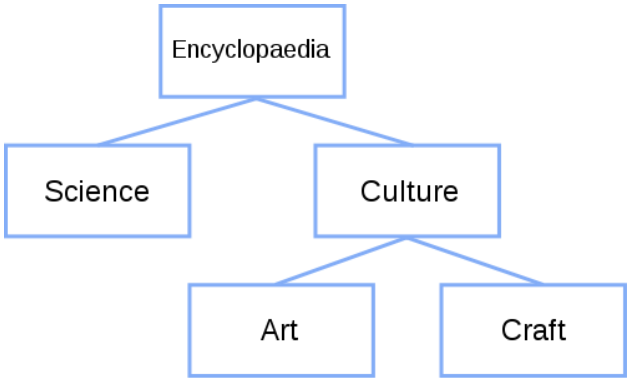
Concepto de árbol

Un **árbol** es una estructura de datos **no lineal** que almacena los elementos **jerárquicamente**

7

Ejemplos

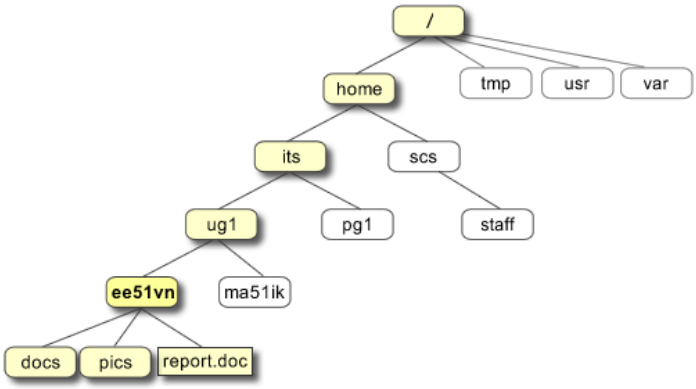
1. **Clasificación de la información en una enciclopedia**



8

Ejemplos

2. **Sistema de ficheros**



Ejemplos

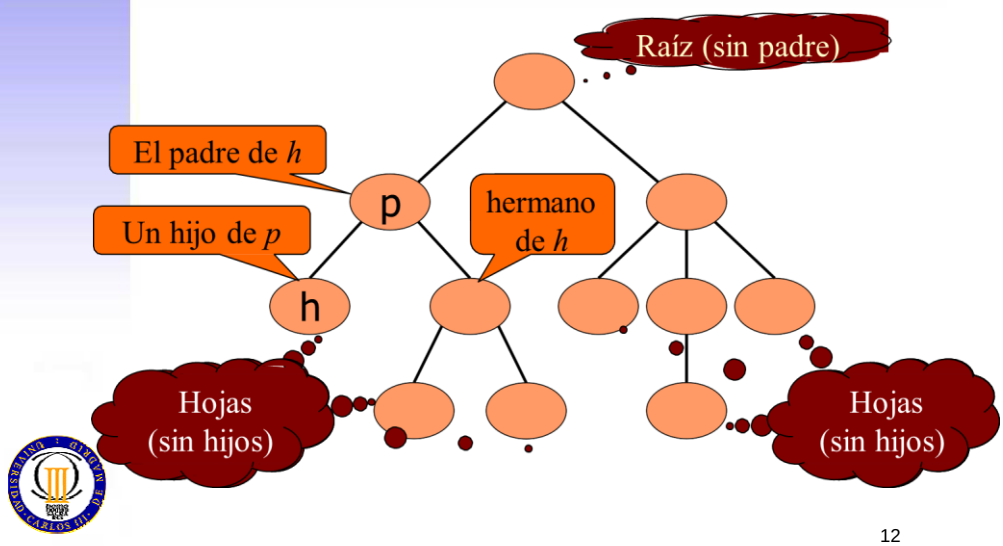
3. **Estructura organizativa de una empresa**
4. **Estructura de rangos del ejército**
5. **Estructura de un libro**

…

Definición no recursiva

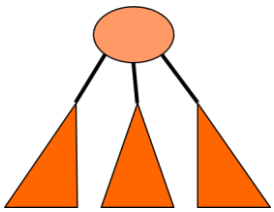
- Un árbol consiste en un conjunto de **nodos** y un conjunto de **aristas**, de forma que:
  - Se distingue un nodo llamado **raíz**
  - A cada nodo h (**hijo**), excepto la raíz, le llega una arista de otro nodo p (**padre**)
  - Para cada nodo hay un **camino** (secuencia de aristas) único desde la raíz.
  - Los nodos que no tienen hijos se denominan **hojas**

Definición no recursiva



Definición recursiva

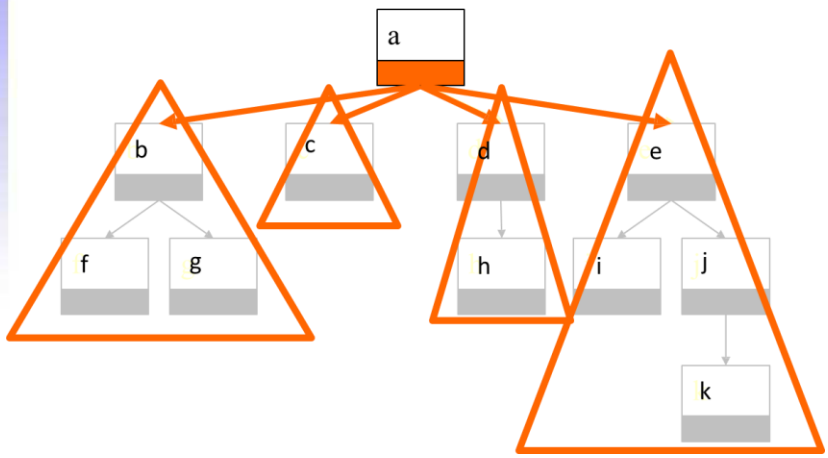
- Un árbol es:
  - Vacío
  - Un nodo y cero o más árboles conectados al nodo mediante una arista



* A los árboles que se conectan al nodo raíz los denominaremos también "subárboles"

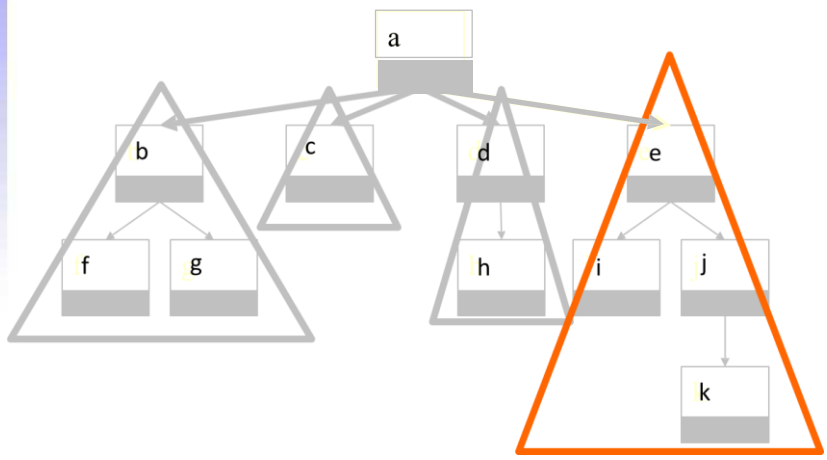
13

Definición recursiva

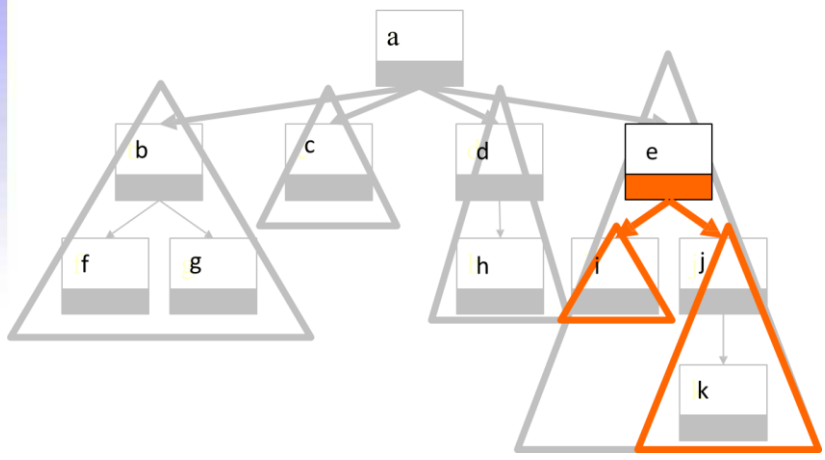


14

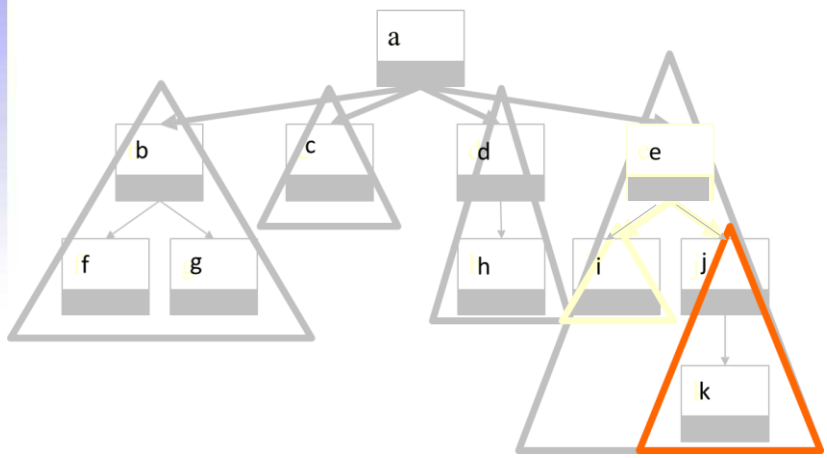
Definición recursiva



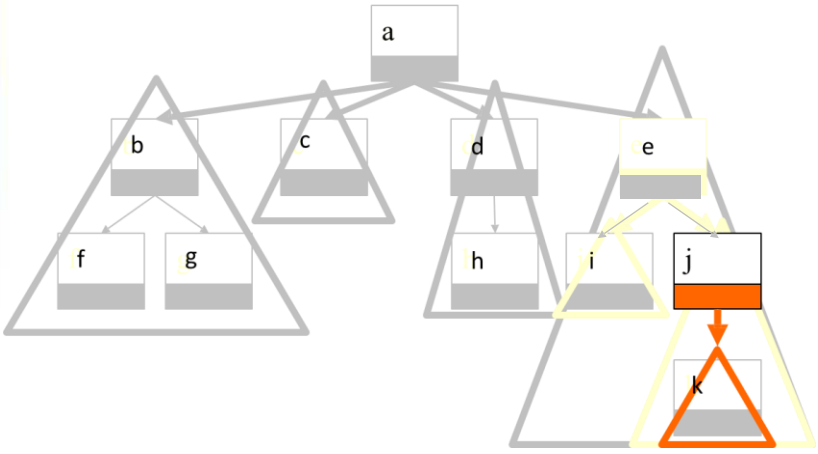
Definición recursiva



Definición recursiva



Definición recursiva



Terminología

- Un nodo es **externo**, si no tiene hijos (es *hoja*)
  - Según la definición recursiva: si todos los subárboles conectados a ese nodo están vacíos
- Un nodo es **interno**, si tiene uno o más hijos
  - Según la definición recursiva: si alguno de los subárboles conectados a ese nodo no está vacío
- Un nodo es **ascendiente** de otro, si es padre de él o ascendiente de su padre.
- Un nodo es **descendiente** de otro, si este último es ascendiente del primero
- Los descendientes de un nodo forman un **subárbol** en el que ese nodo hace de raíz

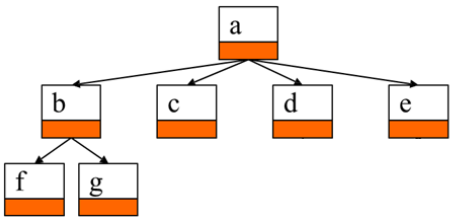
19

Terminología

- Un **camino** de un nodo a otro, es una secuencia de aristas consecutivas que llevan del primero al segundo
  - La **longitud** del camino es el número de aristas
- La **profundidad** de un nodo es la longitud del camino de la raíz a ese nodo
- La **altura** de un árbol es el valor de la profundidad del nodo más profundo
- El **tamaño** de un árbol es el número de nodos que contiene

20

Ejemplo

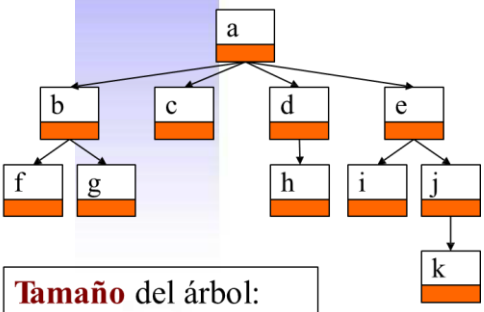


Tamaño del árbol: 7
Altura del árbol: 2

Nodo	Altura	Profundidad	Tamaño	Int./Ext.
a	**2**	0	**7**	Interno
b	1	1	3	Interno
c	0	1	1	Externo
d	0	1	1	Externo
e	0	1	1	Externo
f	0	2	1	Externo
g	0	2	1	Externo

Ejercicio 1

- Completa la tabla para el siguiente árbol

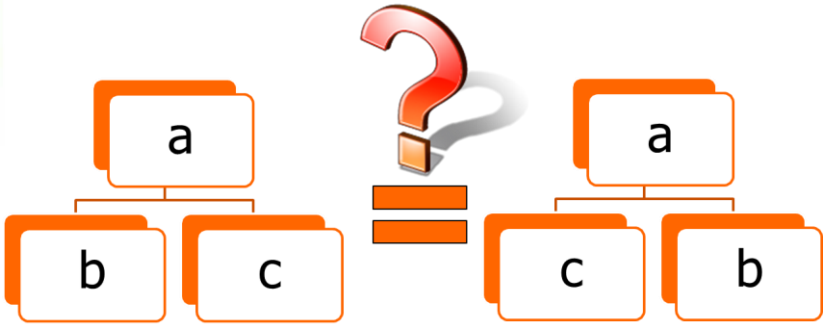


Tamaño del árbol:
Altura del árbol:

Nodo	Altura	Profundidad	Tamaño	Int./Ext.
a				Interno
b	1	1	3	Interno
c	0	1	1	Externo
d				
e				
f	0	2	1	Externo
g	0	2	1	Externo
h				
i				
j				
k				

Terminología: Árbol ordenado

- Un árbol es **ordenado**, si para cada nodo existe un orden lineal para todos sus hijos



Terminología: Árbol binario

- Un árbol **binario** es un árbol ordenado en el que cada nodo tiene 2 árboles (izquierdo y derecho).
  - Árbol binario según la definición recursiva de árbol
  - Los árboles izquierdo y/o derecho pueden estar vacíos

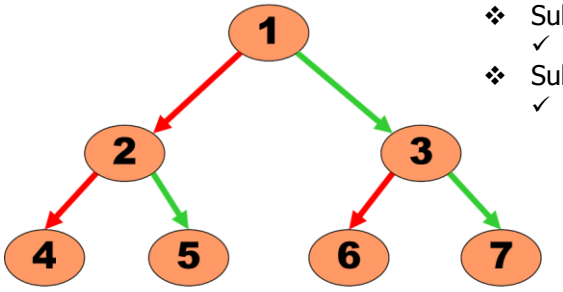
* En general, suponemos árboles binarios para simplificar la implementación de los árboles

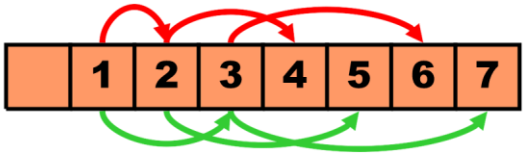
La interfaz BTree

```
public interface BTree<E> {

    static final int LEFT = 0;
    static final int RIGHT = 1;

    boolean isEmpty();
    E getInfo() throws BTreeException;
    BTree<E> getLeft() throws BTreeException;
    BTree<E> getRight() throws BTreeException;

    void insert(BTree<E> tree, int side) throws BTreeException;
    BTree<E> extract(int side) throws BTreeException;

    String toStringPreOrder();
    String toStringInOrder();
    String toStringPostOrder();
    String toString(); // preorder

    int size();
    int height();

    boolean equals(BTree<E> tree);
    boolean find(BTree<E> tree);
}
```


Una interfaz varias implementaciones

- Implementación basada en arrays

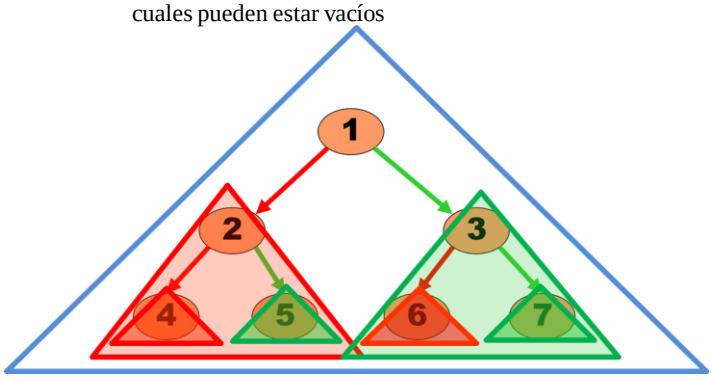




- Subárbol izquierdo
  - ✓ Posición nodo raíz * 2
- Subárbol derecho
  - ✓ Posición nodo raíz * 2 +1

Una interfaz varias implementaciones

- Implementación basada en nodos enlazados
  - *Linked Binary Node (LBNode)*
  - *Linked Binary Tree (LBTree)*
    - ✓ Cada árbol (*LBTree)* tiene un nodo raíz (atributo *LBNode*)
    - ✓ Cada nodo raíz *LBNode* apunta a dos árboles (atributos *LBTree*), los cuales pueden estar vacíos



27

La clase LBNode

```
public class LBNode<E> {

    private E info;
    private BTree<E> left;
    private BTree<E> right;

    public LBNode(E info, BTree<E> left, BTree<E> right) {…}
    public E getInfo() {…}
    public void setInfo(E info) {…}
    public BTree<E> getLeft() {…}
    public void setLeft(BTree<E> left) {…}
    public BTree<E> getRight() {…}
    public void setRight(BTree<E> right){…}
}
```

28

Ejercicio 2

- Completa la implementación de la clase **`LBNode`**.

29

La clase LBTree

```
public class LBTree<E> implements BTree<E> {

    private LBNode<E> root;

    public LBTree() {
        root = null;
    }

    public LBTree(E info) {
        root = new LBNode<E>(info, new LBTree<E>(), new LBTree<E>());
    }

    public boolean isEmpty() {
        return (root == null);
    }
    …
```

30

La clase LBTree

```
...
    public E getInfo() throws BTreeException {
        if (isEmpty()) {
            throw new BTreeException("empty trees do not have info");
        }
        return root.getInfo();
    }

    public BTree<E> getLeft() throws BTreeException {
        if (isEmpty()) {
            throw new BTreeException("empty trees do not have a left child");
        }
        return root.getLeft();
    }

    public BTree<E> getRight() throws BTreeException {
        if (isEmpty()) {
            throw new BTreeException("empty trees do not have a right child");
        }
        return root.getRight();
    }
...
```


Algoritmos básicos

- Tamaño: **size()**
- Altura: **height()**
- Recorridos
  - Pre-orden: **toStringPreOrder()**
  - In-orden: **toStringInOrder()**
  - Post-orden: **toStringPostOrder()**

La clase LBTree: size()

```
public int size() {
    if (isEmpty()) {
        return 0;
    } else {
        return 1
            + root.getLeft().size()
            + root.getRight().size();
    }
}
```

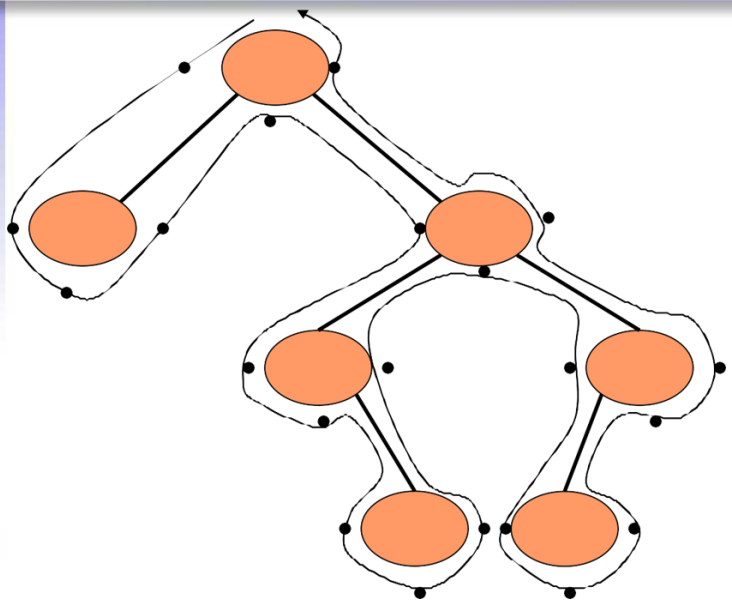
33

La clase LBTree: height()

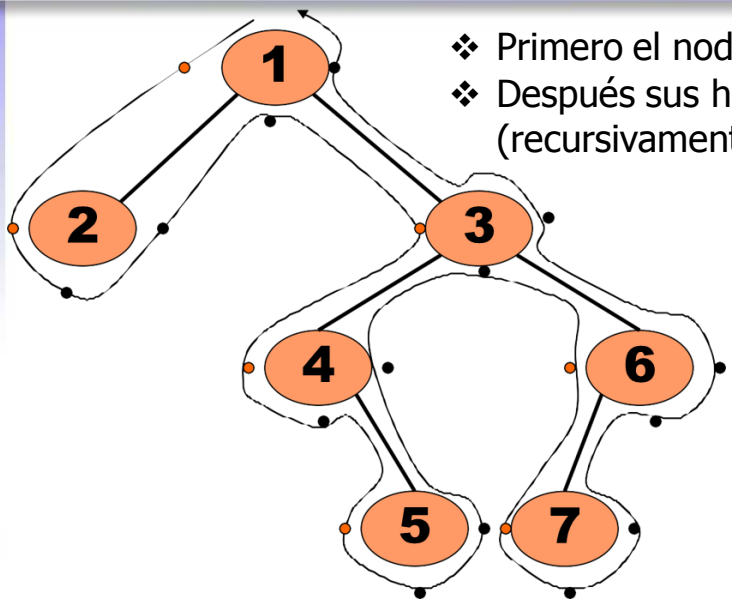
```
public int height() {
    if (isEmpty()) {
        return -1;
    } else {
        int leftHeight = root.getLeft().height();
        int rightHeight = root.getRight().height();
        if (leftHeight > rightHeight) {
            return 1 + leftHeight;
        } else {
            return 1 + rightHeight;
        }
    }
}
```

34

Recorrido de Euler



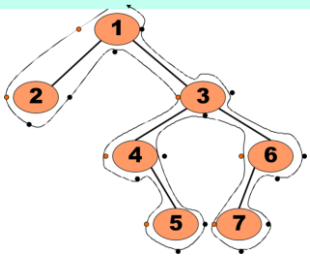
Recorrido Pre-orden



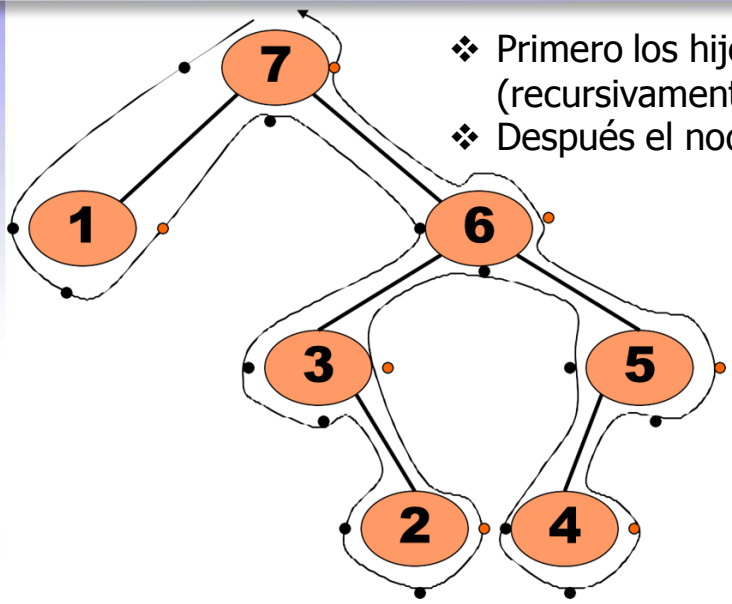
- Primero el nodo raíz
- Después sus hijos (recursivamente)

La clase LBTree: toStringPreOrder()

```
public String toStringPreOrder() {
    if (isEmpty()) {
        return "";
    } else {
        return root.getInfo().toString() + " " +
            root.getLeft().toStringPreOrder() +
            root.getRight().toStringPreOrder();
    }
}
```



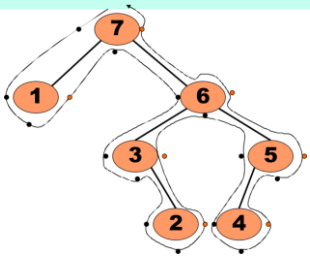
Recorrido Post-orden



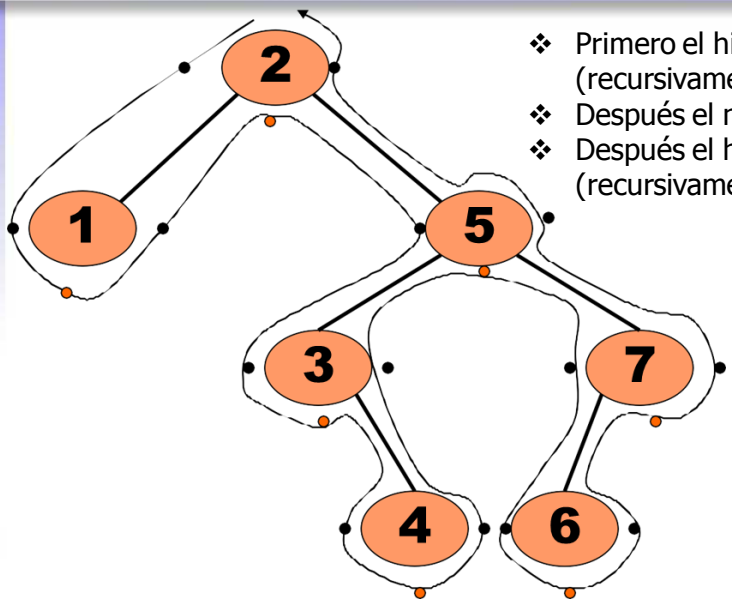
- Primero los hijos (recursivamente)
- Después el nodo raíz

La clase LBTree: toStringPostOrder()

```
public String toStringPostOrder() {
    if (isEmpty()) {
        return "";
    } else {
        return root.getLeft().toStringPostOrder() +
            root.getRight().toStringPostOrder() +
            root.getInfo().toString() + " ";
    }
}
```



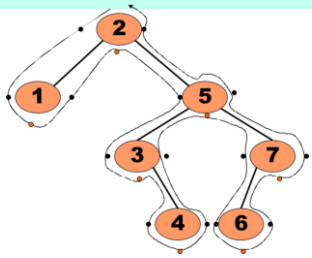
Recorrido In-orden (simétrico)



- Primero el hijo izquierdo (recursivamente)
- Después el nodo raíz
- Después el hijo derecho (recursivamente)

La clase LBTree: toStringInOrder()

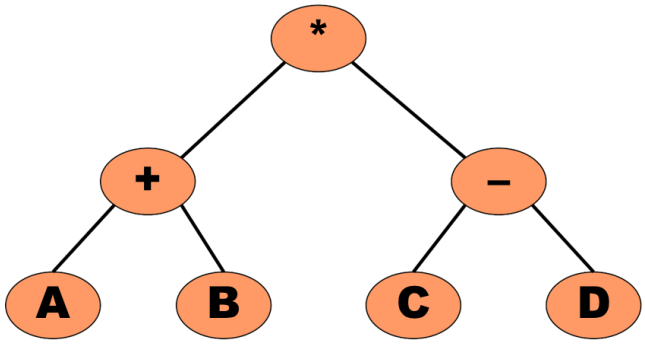
```
public String toStringInOrder() {
    if (isEmpty()) {
        return "";
    } else {
        return root.getLeft().toStringInOrder() +
            root.getInfo().toString() + " " +
            root.getRight().toStringInOrder();
    }
}
```



41

Ejercicio 3

- Dado el siguiente árbol binario, indica qué recorrido (pre-orden, in-orden, post-orden) produce el resultado (A+B)*(C-D).



42

Ejemplo

Infijo	Prefijo	Postfijo
A+B	+AB	AB+
A+B–C	–+ABC	AB+C–
(A+B)*(C–D)	*+AB–CD	AB+CD–*