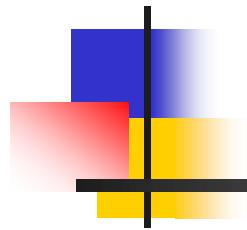


# Programació Avançada

## Estructures no lineals



### El T.A.D. ACB

Definició. Usos. Operacions.

Definició de la classe Acb:

la interfície

Implementació enllaçada amb  
punters (**2 versions**).



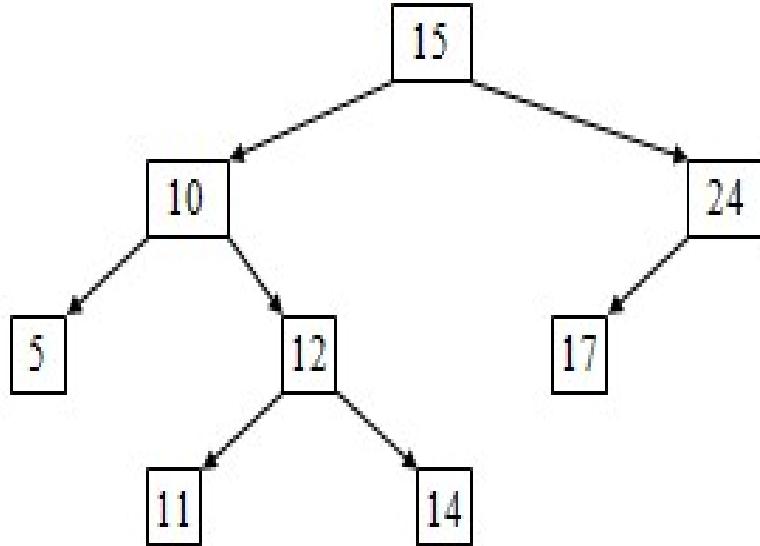
# Arbre de Cerca binaria ACB. Definició

- Un ACB és un arbre binari on els seus nodes estan **ordenats segons el següent criteri:**
  - els elements que emmagatzemen els nodes que formen el subarbre esquerre de **qualsevol node n** tenen un valor més petit que l'element del node n, i
  - els elements que emmagatzemen els nodes que formen el subarbre dret de **qualsevol node n** tenen un valor més gran que l'element del node n.
- **Una conseqüència** d'aquesta definició és que en un ACB no poden haver nodes amb el mateix element, **no poden haver repetits!**



JAVA

# Exemple



Adoneu-vos que el recorregut en **inordre** de l'arbre:  
5,10,11,12,14,15,17,24 obté els **elements ordenats**.



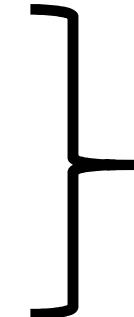
JAVA

# Usos

- La seva utilitat és la mateixa que la proporcionada per les llistes associatives, per tant és una alternativa a aquestes.
- L'avantatge dels ACB en front de les llistes és que habitualment milloren l'eficiència, la complexitat en promig és de l'ordre  $\theta(\lg_2 n)$  essent n el nombre d'elements que té l'estructura i en el pitjor dels casos és  $\theta(n)$ , la mateixa que la de les llistes, denoteu que un ACB pot degenerar en una llista associativa (solució arbres AVL).



# Operacions

- Construir un arbre buit
  - Consultar l'arrel de l'arbre
  - Consultar el fill esquerra de l'arbre
  - Consultar el fill dret de l'arbre
  - Determinar si l'arbre és buit
  - Inserir
  - Esborrar
  - HiEs (Membre)
  - Buidar
- 

**noves**



JAVA

# Objectes Comparables

- Els arbres ACB **només poden** emmagatzemar objectes comparables.
- Implementació de la interfície Comparable.

## La interfície Comparable<T>

- La interfície **Comparable** és una de les més utilitzades (implementades).
- Els objectes de les classes que la implementen poden comparar-se entre ells (comparació d'ordre)
- Es troba definida en el paquet [java.lang](#)

## La interfície Comparable<T> El mètode compareTo

- **Comparable** defineix un únic mètode anomenat **compareTo**  
**public int compareTo(T o)**
- El resultat de **compareTo** és
  - **ClassCastException** si *this* i *o* no es poden comparar
  - Negatiu si *this* s'ha de considerar menor que *o*
  - Zero si *this* i *o* s'han de considerar iguals
  - Positiu si *this* s'ha de considerar major que *o*



JAVA

# La interfície



Creador

```
public interface Acb<T> {  
    Object Arrel() throws Exception;  
    Ab FillEsquerre() throws Exception;  
    Ab FillDret() throws Exception;  
    boolean AbBuit();  
    void buidar();  
    //OPERACIONES PRÓPTES DEL ACB  
    void Inserir(Comparable<T> e) throws RuntimeException;  
    /*  
     * afegeix a l'ACB l'element especificat, el situa en el lloc adient segons  
     * l'ordenació, si no és possible es crea i llença una excepció  
     */  
    void Esborrar(Comparable<T> e) throws RuntimeException;  
    /*  
     * elimina de l'ACB l'element especificat, si no és possible es crea i  
     * llença una excepció  
     */  
    boolean Membre(Comparable<T> e);  
    /*  
     * comprova l'aparició o no de l'element a l'arbre, si no hi és retorna un  
     * valor de false  
     */  
}
```

Mateix contingut que la interfície Ab

Tot Acb és un Ab, reaprofitem codi!



JAVA

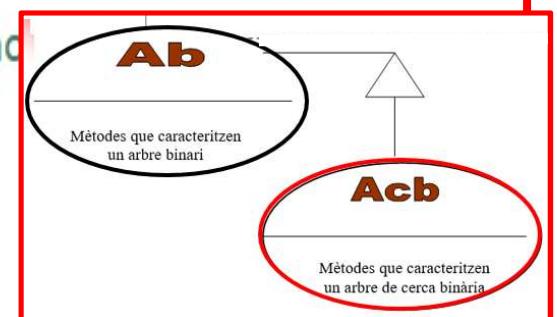
# La interfície.

Reaprofitem codi



```
public interface Acb<T> extends Ab { Herència entre interfícies
    void inserir(Comparable<T> e) throws RuntimeException;
    /*
     * afegeix a l'ACB l'element especificat, el situa en el lloc adient segons
     * l'ordenació, si no és possible es crea i llença una excepció
     */
    void esborrar(Comparable<T> e) throws RuntimeException;
    /*
     * elimina de l'ACB l'element especificat, si no és possible es crea i
     * llença una excepció
     */
    boolean membre(Comparable<T> e);
    /*
     * comprova l'aparició o no de l'element a l'arbre, si no
     * valor de false
     *
     */
}
```

No són de  
tractament  
obligat!





# Implementem .....



Creador

**Ab**

Mètodes que caracteritzen  
un arbre binari

**Acb**

Mètodes que caracteritzen  
un arbre de cerca binària

Denota que la  
classe  
implementa la  
interfície Acb

**AbEnII**

// atributs

//mètodes

**ObjectComparable**

//atributs

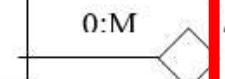
//mètodes

**Acb?**

// atributs

//mètodes

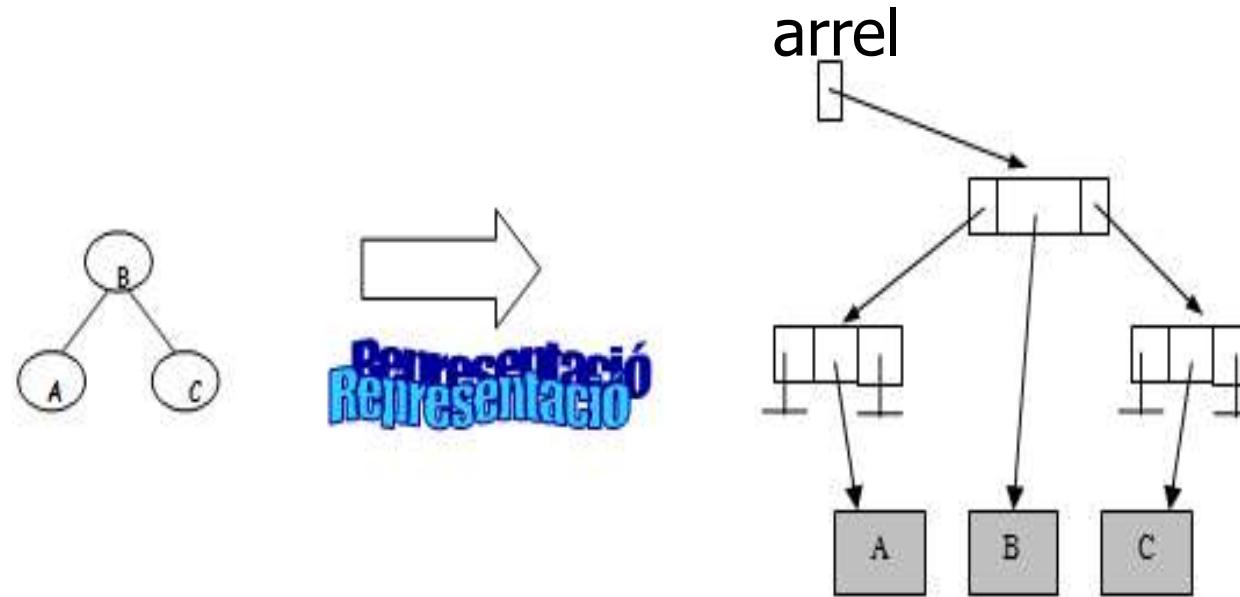
0:M





JAVA

# Implementació enllaçada

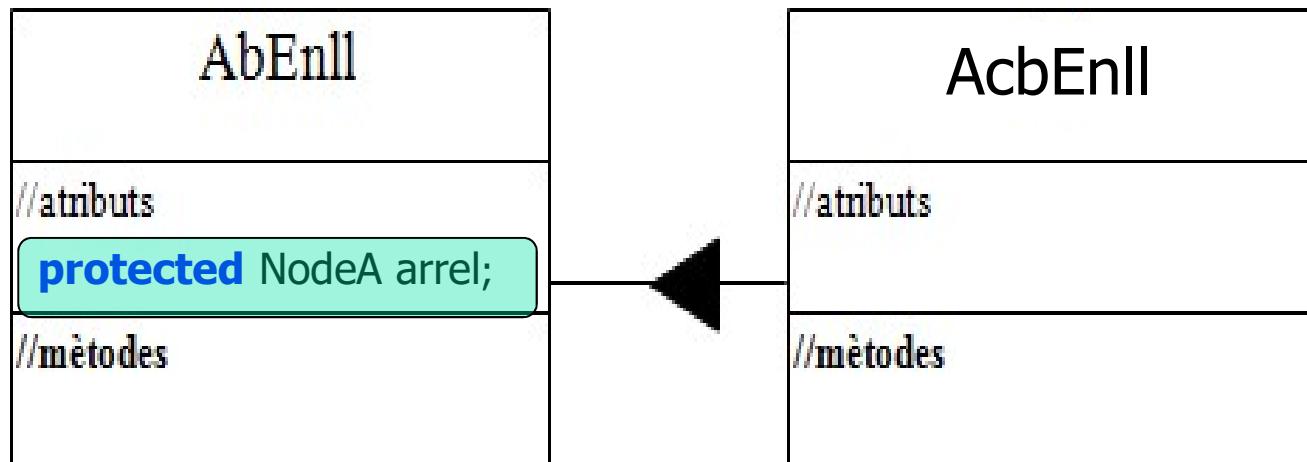


- ✓ Identicament als arbres binaris!!!!
- ✓ En la implementació no caldrà afegir **cap** atribut
- ✓ Recordeu que el vàrem declarar amb visibilitat **protected**



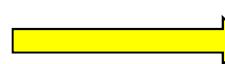
JAVA

# Consideració ...



Implementarem els mètodes “nous”  
Redefinirem quins mètodes?

Un Object pot  
emmagatzemar  
un Comparable? SI  
S'haurà de fer càsting en la recuperació  
perquè  
s'emmagatzema com a objecte!



```
class NodeA {
    Object inf;
    NodeA esq, drt;

    NodeA() {
        this(null);
    }
    NodeA(Object o) {
        this(o, null, null);
    }
    NodeA(Object o, NodeA e, NodeA d) {
        inf = o;
        esq = e;
        drt = d;
    }
}
```

NodeA.java



JAVA

# Operacions

- Construir un arbre buit
- Consultar l'arrel de l'arbre
- Consultar el fill esquerra de l'arbre
- Consultar el fill dret de l'arbre
- Determinar si l'arbre és buit
- inserir
- esborrar
- membre
- buidar

Operacions a  
redefinir



# Implementació enllaçada



## Consideracions:

- Herència classe AbEnll. Herència atribut de nom **arrel** de tipus NodeA i amb visibilitat **protected**.
- **Implementació de les operacions:**
  - inserir
  - esborrar
  - membre
- Les operacions heretades **no** s'han de redefinir. **El constructor no s'hereta.**



# Implementació enllaçada



## Implementació 1:

Recursivitat afegint mètodes static a la pròpia classe.

### Estructura

```
public Queue<Object> preordre() {  
    Queue<Object> c = new LinkedList<Object>();  
    if (arrel != null)  
        preordreR(c, arrel);  
    return c;  
}  
  
private static void preordreR(Queue<Object> c, NodeA a) {  
    // sabem que el paràmetre a no és null  
    try {  
        c.add(a.inf);  
    } catch (IllegalStateException e) {}  
    if (a.esq != null)  
        preordreR(c, a.esq);  
    if (a.drt != null)  
        preordreR(c, a.drt);  
}
```

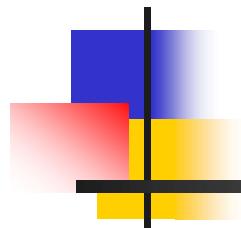
## Implementació 2:

Recursivitat afegint un mètode  
a la classe NodeA. **Estructura**

```
public Queue<Object> inordre(){  
    Queue<Object> c = new LinkedList<Object>();  
    if (arrel!=null) arrel.inordre(c);  
    return c;  
}  
  
public void inordre(Queue<Object> c){  
    if (esq!=null) esq.inordre(c);  
    try {  
        c.add(inf);  
    } catch (IllegalStateException e) {}  
    if (drt!=null) drt.inordre(c);  
}  
  
class NodeA {  
    Object inf;  
    NodeA esq, drt;  
  
    NodeA() {  
        this(null);  
    }  
    NodeA(Object o) {  
        this(o, null, null);  
    }  
    NodeA(Object o, NodeA e, NodeA d) {  
        inf = o;  
        esq = e;  
        drt = d;  
    }  
}
```

# Programació Avançada

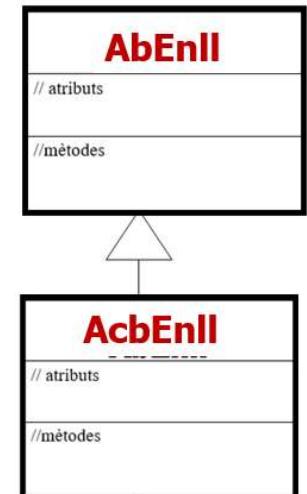
## Estructures no lineals



El T.A.D. ACB

Implementació 1

Recursivitat a la pròpia classe  
afegint mètode **static**





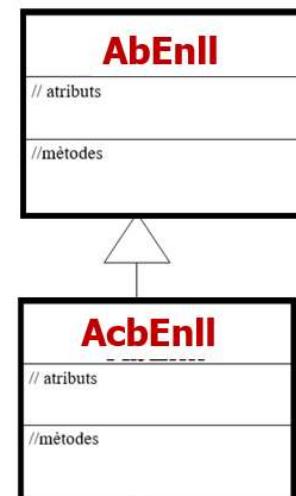
JAVA

# Implementació enllaçada



```
public class AcbEnll<T> extends AbEnll implements Acb<T> {  
    // NO s'afegeixen atributs  
    // l'atribut NodeA arrel ve d'herència i la classe hi té accés  
    public AcbEnll() {  
        super();  
    } // opcional  
  
    // Implementació de mètodes  
}
```

**No caldria posar  
el constructor!**





# Operació membre

## ■ Cercar. Com?

■ Els passos que s'han de seguir a l'hora d'inserir un element en un ACB, són:

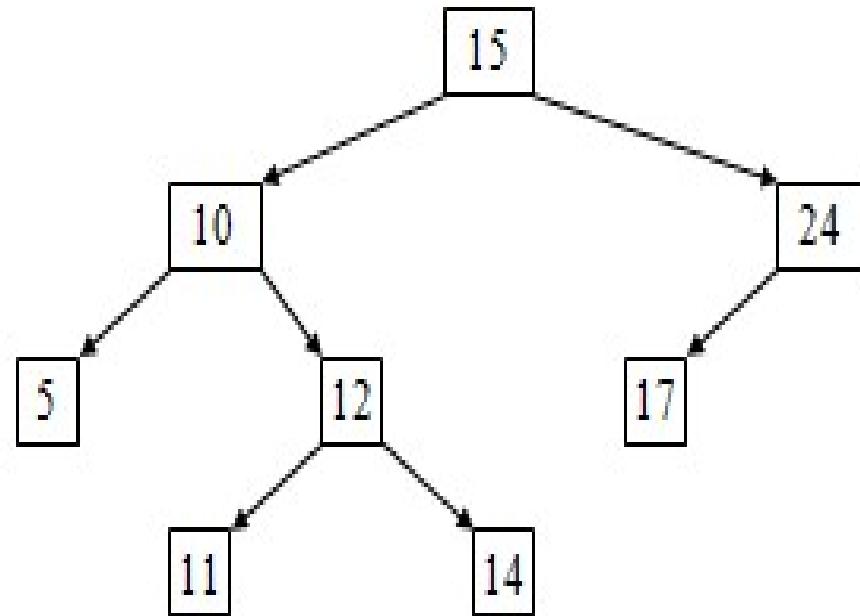
- Comparar amb l'arrel. Si és igual la cerca finalitza amb èxit. Si l'element és més gran, fer la cerca pel subarbre esquerra. Si és més petit fer-la pel subarbre dret. **Si s'arriba a un arbre buit la cerca ha fracassat.**
- Aquest procés és, essencialment, **una cerca dicotòmica.**



# Operació membre

## ■ Membre. Com?

- 



Localitzar el valor 13. **Procés a seguir?**

- El procés anterior finalitza quan s'ha localitzat l'element o arribem a una fulla, en aquest cas no hi és!



JAVA

# Operació membre



Creador

```
@Override  
public boolean membre(Comparable<T> e) {  
    return membreRec(this.arrel, e);  
}
```

```
private static boolean membreRec(NodeA a, Comparable<T> e) {  
    if (a==null) return false; // cas base
```

// comparar

```
int cmp = e.compareTo((T)a.inf);
```

```
if (cmp==0) return true; // trobat!
```

// en funció de si és menor o major triar subarbre

```
if (cmp<0) return membreRec(a.esq,e);
```

```
else return membreRec(a.drt,e);
```

```
}
```

Implementació recursiva. Aquesta es fa en un nou mètode **static privat afegit a la classe**.

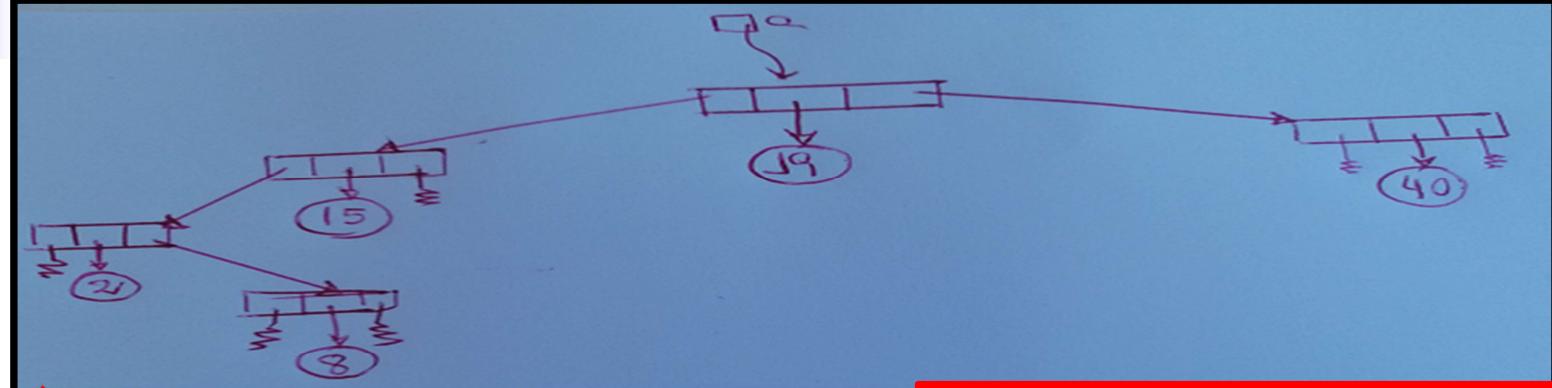
```
class NodeA {  
    Object inf;  
    NodeA esq, drt;  
  
    NodeA() {  
        this(null);  
    }  
    NodeA(Object o) {  
        this(o, null, null);  
    }  
    NodeA(Object o, NodeA e, NodeA d) {  
        inf = o;  
        esq = e;  
        drt = d;  
    }  
}
```

NodeA.java

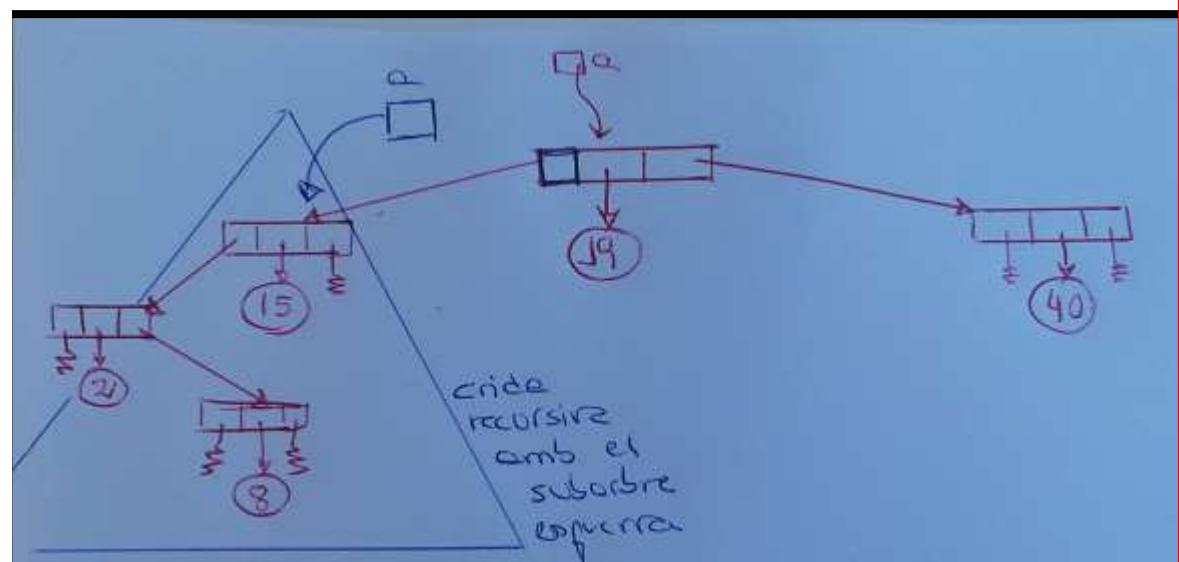


JAVA

# Operació membre



Localitzem el 8



```
private static boolean membreRec(NodeA a, Comparable<T> e) {  
    if (a==null) return false; // cas base  
  
    // comparar  
    int cmp = e.compareTo((T)a.inf);  
  
    if (cmp==0) return true; // trobat!  
  
    // en funció de si és menor o major triar subarbre  
    if (cmp<0) return membreRec(a.esq,e);  
    else return membreRec(a.drt,e);  
}
```



# Operació inserir

## Inserir. Com?

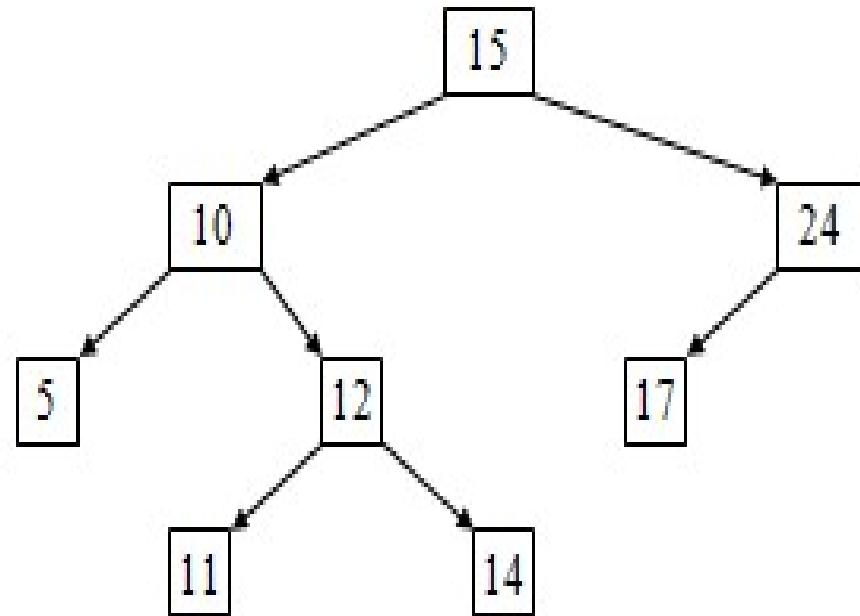
- Els passos que s'han de seguir a l'hora d'inserir un element en un ACB, són:
  - **buscar el lloc adient**, per això es comença comparant l'element a inserir amb l'arrel de l'arbre. En funció del resultat de la comparació es repeteix el procés, amb el subarbre esquerre, si l'element a inserir és menor que l'arrel o, amb el dret si l'element a inserir és major que l'arrel. Si l'element a inserir ja forma part de l'arbre el nou element no pot ser inserit (no poden haver-ne de repetits) i per tant l'operació no serà possible.
  - El procés anterior finalitza quan s'ha localitzat el lloc que li pertoca, **sempre serà una fulla**.



# Operació inserir

## ■ Inserir. Com?

- 



Inserim el valor 13. Procés a seguir?

- El procés anterior finalitza quan s'ha localitzat el lloc que li pertoca, **sempre serà una fulla**.



# Operació inserir



Observeu que l'únic cas en que es modifica l'atribut **arrel** és quan l'arbre era buit

```
@Override  
public void inserir(Comparable<T> e) throws RuntimeException {  
    this.arrel = inserirRecursiu(this.arrel, e);  
}  
  
private static NodeA inserirRecursiu(NodeA a, Comparable<T> e){  
    if (a==null) {  
        a = new NodeA();  
        a.esq = null; a.inf=e; a.drt = null;  
    }  
    else {  
        int cmp = e.compareTo((T)a.inf);  
        if (cmp == 0) throw new RuntimeException("Repetit "+e);  
        if (cmp < 0) {  
            a.esq = inserirRecursiu(a.esq, e);  
        }  
        else {  
            a.drt = inserirRecursiu(a.drt,e);  
        }  
    }  
    return a;  
}
```

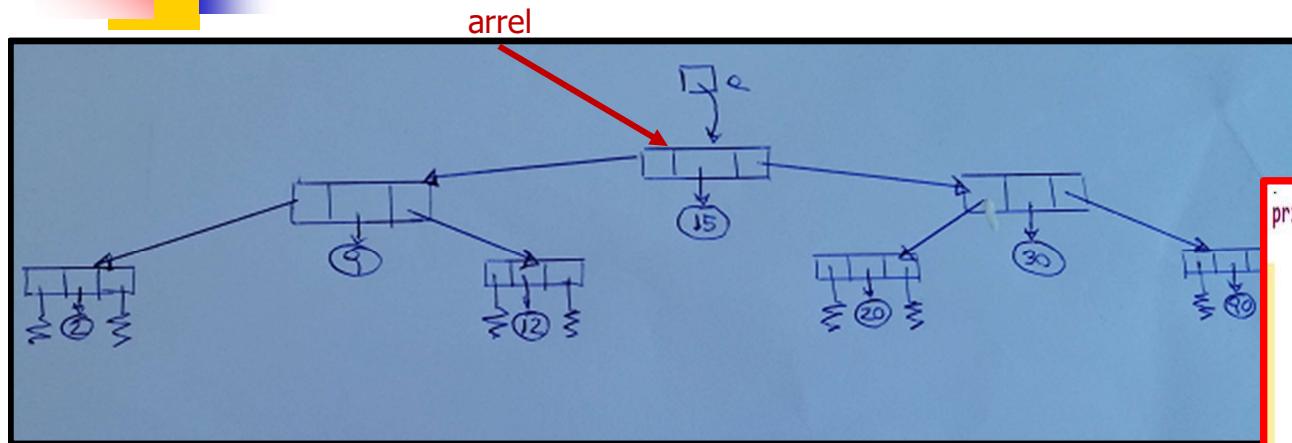


JAVA

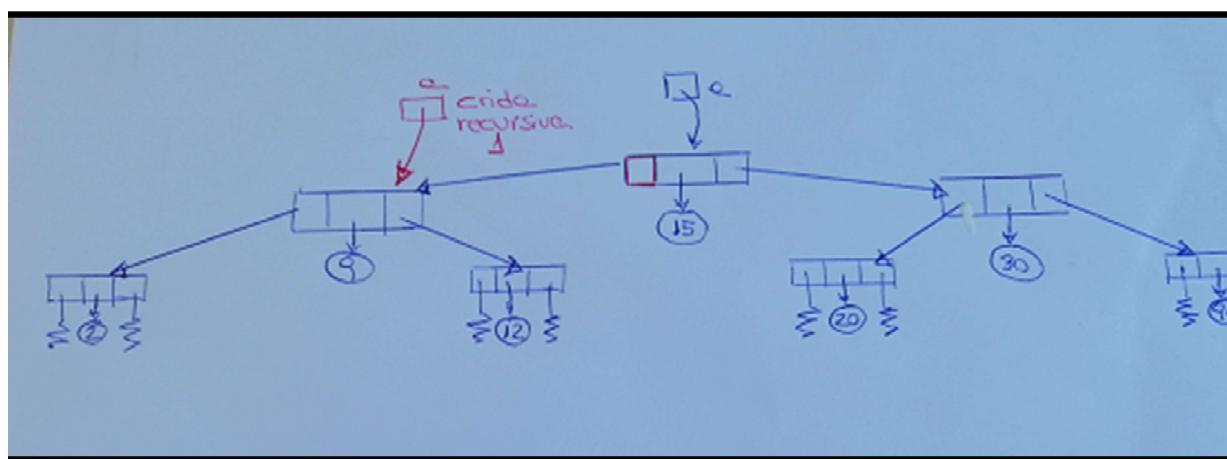
# Operació inserir



Exemple: Inserim el valor 13. Crida principal



```
private static NodeA inserirRecursiu(NodeA a, Comparable<T> e){  
    if (a==null) {  
        a = new NodeA();  
        a.esq = null; a.inf=e; a.drt = null;  
    }  
    else {  
        int cmp = e.compareTo((T)a.inf);  
        if (cmp == 0) throw new RuntimeException("Repetit "+e);  
        if (cmp < 0) {  
            a.esq = inserirRecursiu(a.esq, e);  
        }  
        else {  
            a.drt = inserirRecursiu(a.drt,e);  
        }  
    }  
    return a;  
}
```



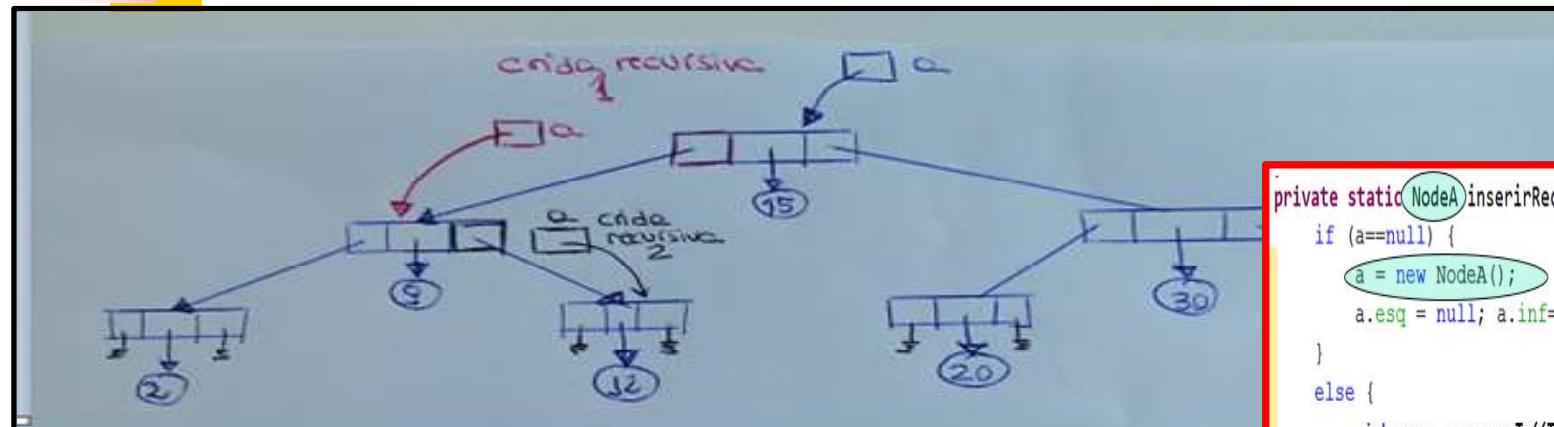


JAVA

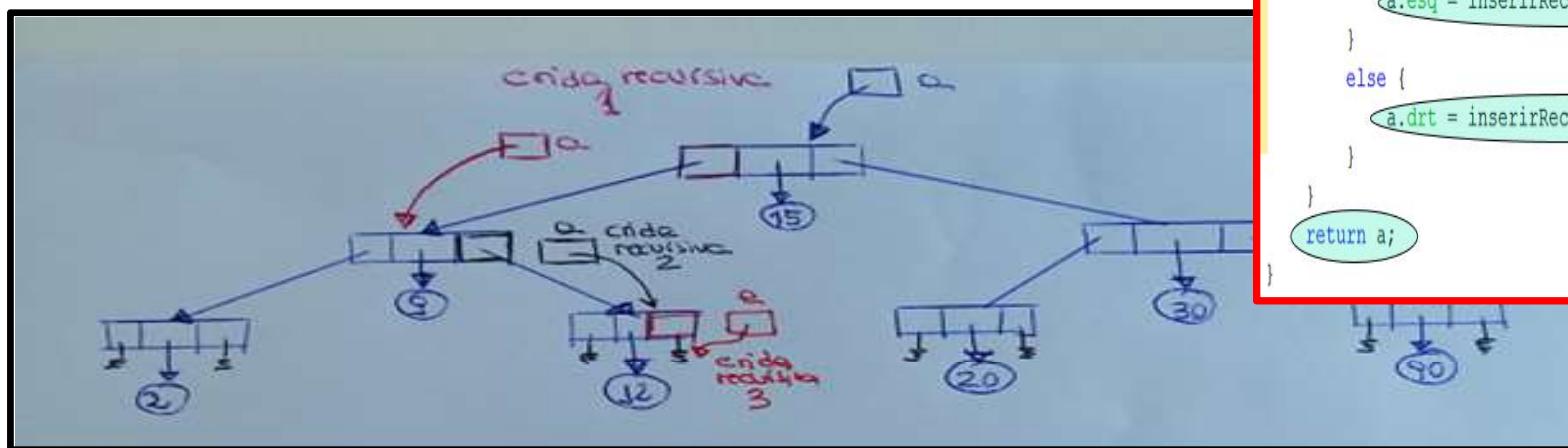
# Operació inserir



Exemple: Inserim el valor 13. **Crida principal**



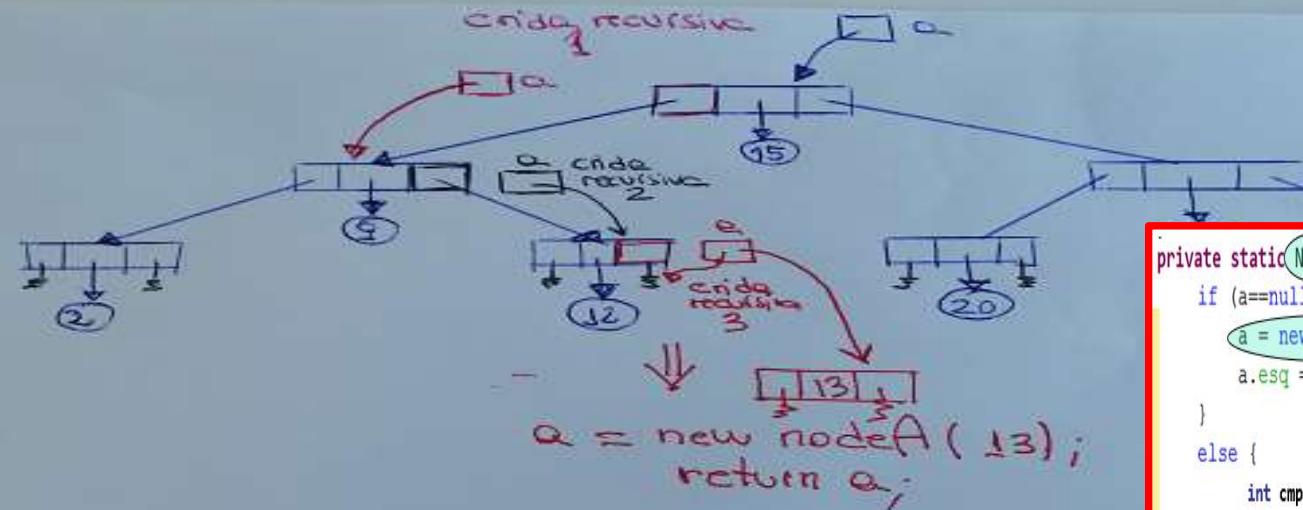
```
private static NodeA inserirRecursiu(NodeA a, Comparable<T> e){  
    if (a==null) {  
        a = new NodeA();  
        a.esq = null; a.inf=e; a.drt = null;  
    }  
    else {  
        int cmp = e.compareTo((T)a.inf);  
        if (cmp == 0) throw new RuntimeException("Repetit "+e);  
        if (cmp < 0) {  
            a.esq = inserirRecursiu(a.esq, e);  
        }  
        else {  
            a.drt = inserirRecursiu(a.drt,e);  
        }  
    }  
    return a;  
}
```



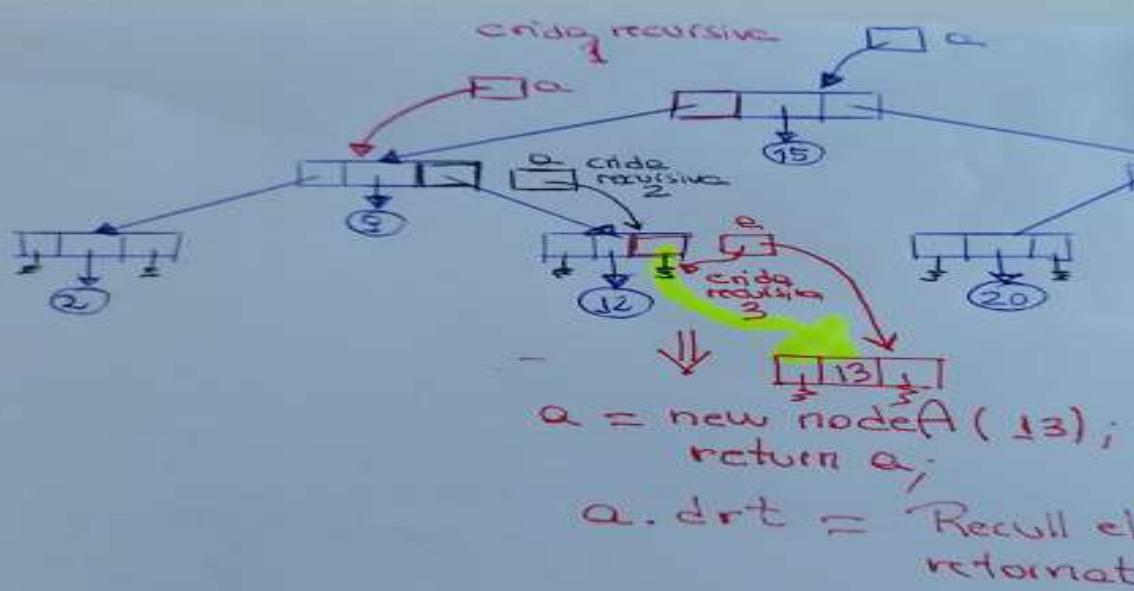


JAVA

# Operació inserir



```
private static NodeA inserirRecursiu(NodeA a, Comparable<T> e){  
    if (a==null) {  
        a = new NodeA();  
        a.esq = null; a.inf=e; a.drt = null;  
    }  
    else {  
        int cmp = e.compareTo((T)a.inf);  
        if (cmp == 0) throw new RuntimeException("Repetit "+e);  
        if (cmp < 0) {  
            a.esq = inserirRecursiu(a.esq, e);  
        }  
        else {  
            a.drt = inserirRecursiu(a.drt,e);  
        }  
    }  
    return a;  
}
```





JAVA

# Operació esborrar

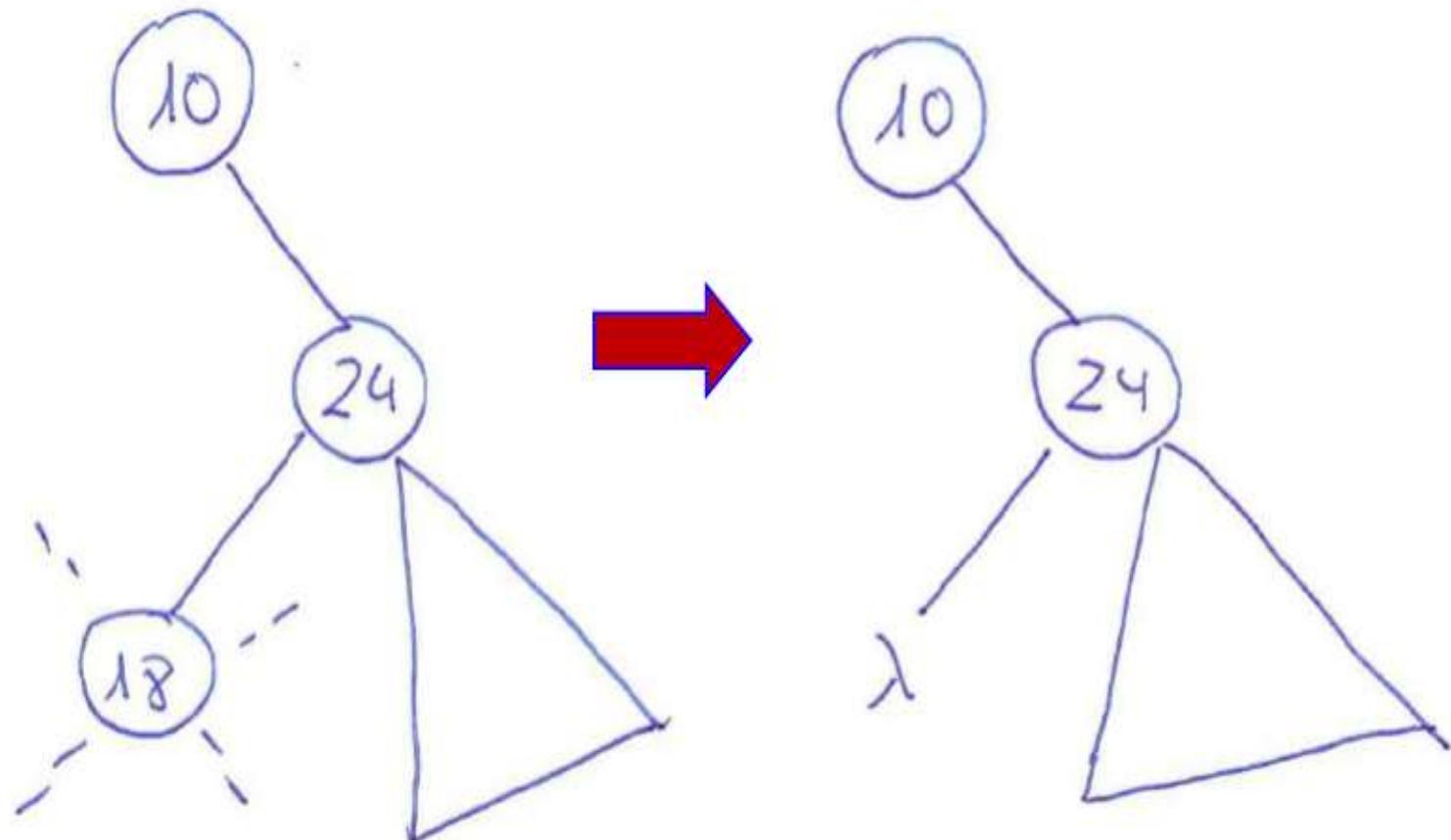
## ■ **Esborrar. Com?**

- S'ha de repetir el **mateix procés de cerca** que en el cas d'inserir un element (per localitzar l'element a esborrar), però ara no necessàriament l'element a esborrar ha de ser una fulla. Poden donar-se **tres situacions diferents**, l'element a esborrar és:
  - una **fulla**
  - un node intern amb un **únic fill**, el dret o l'esquerre
  - un node intern amb **dos fills**



JAVA

# Esborrar fulla



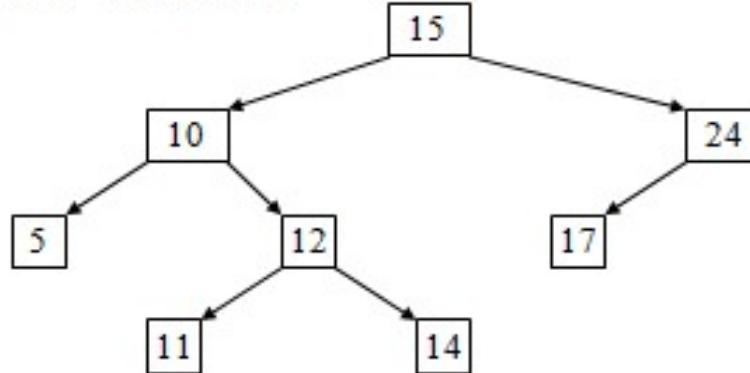


JAVA

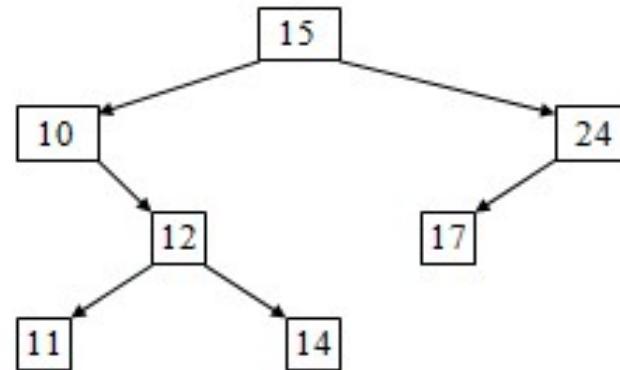
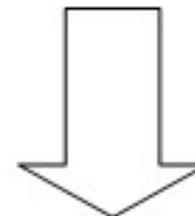
# Esborrar fulla

**Exemple:** el node a esborrar és una fulla

El node a  
esborrar  
és una  
fulla



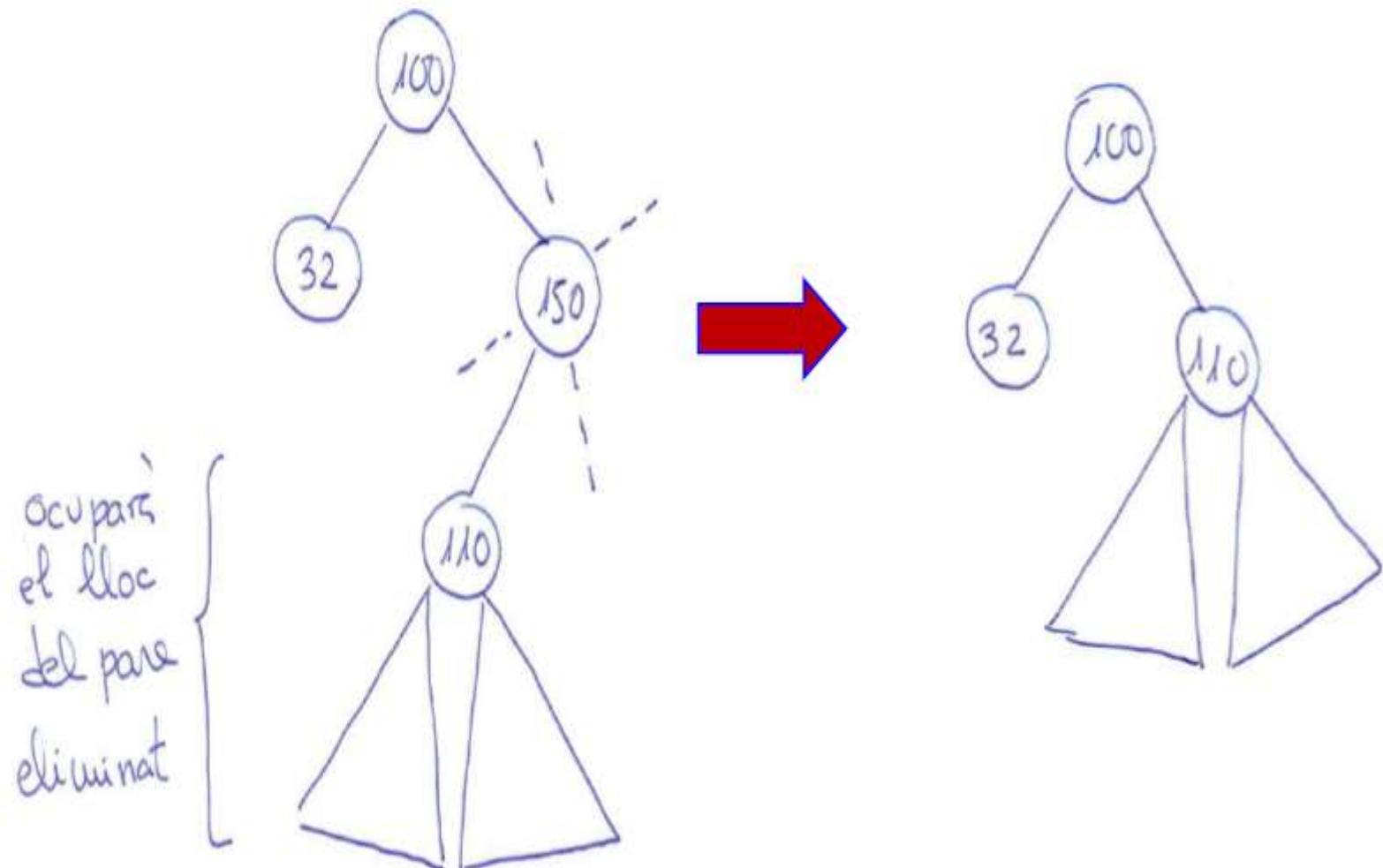
El pare del node a eliminar no  
tindrà fill esquerre





JAVA

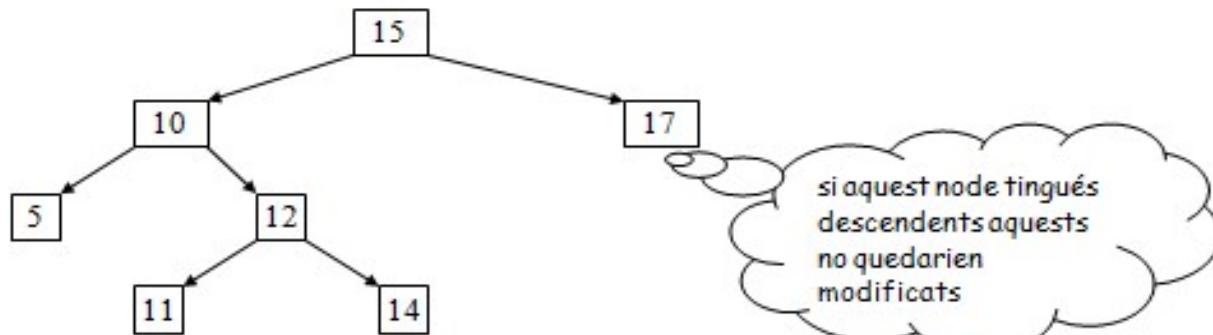
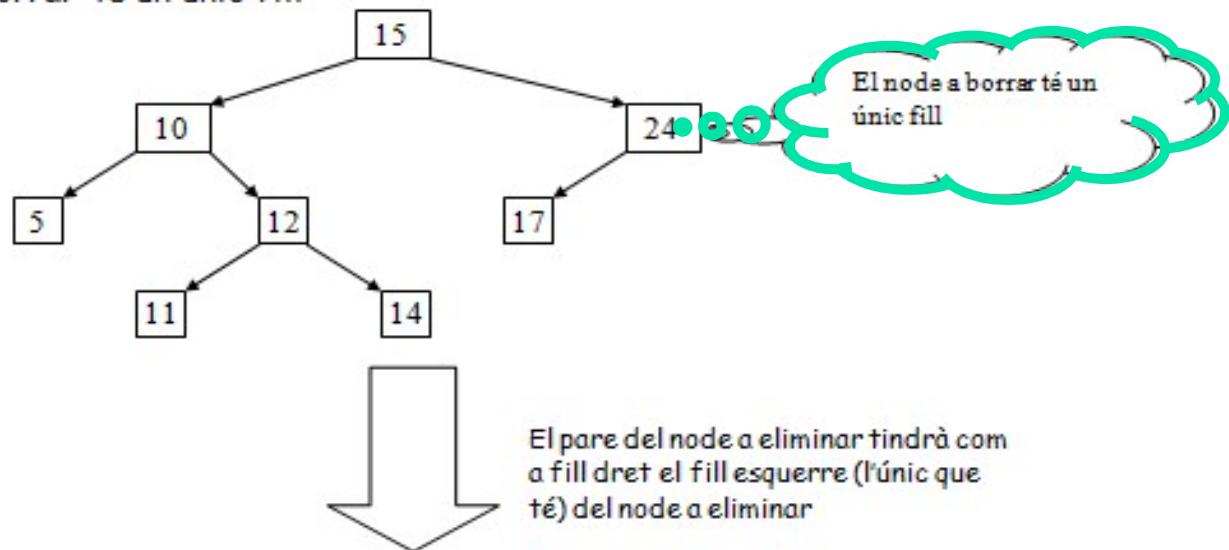
# Esborrar node amb un sol fill





# Esborrar node amb un sol fill

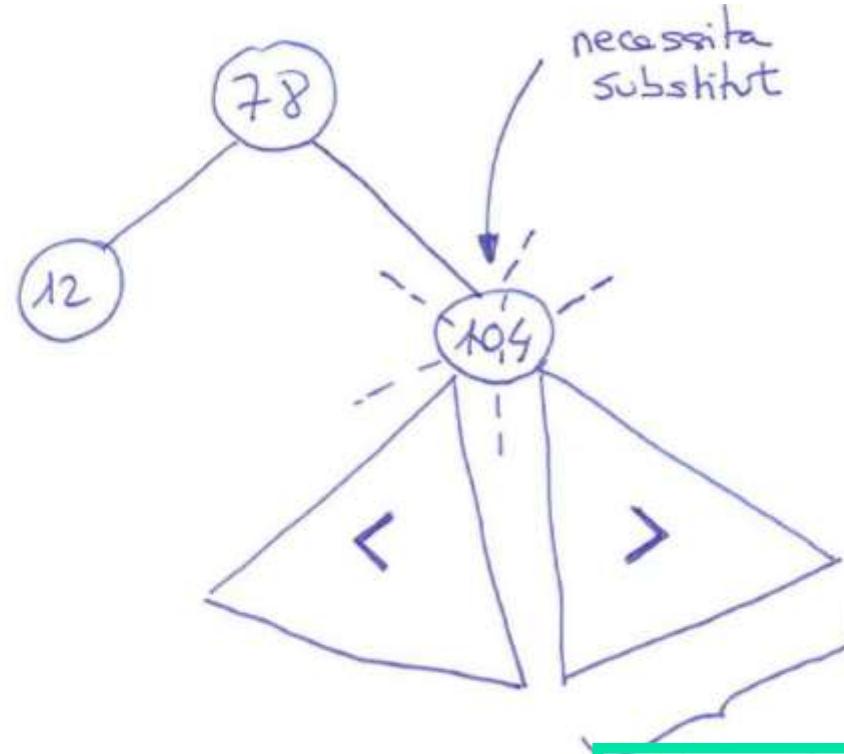
Exemple: el node a esborrar té un únic fill





JAVA

# Esborrar node amb 2 fills



el més petit  
d'aquest subarbre  
serà el substitut  
[buscar-lo, substituir  
l'eliminat i eliminar-lo]

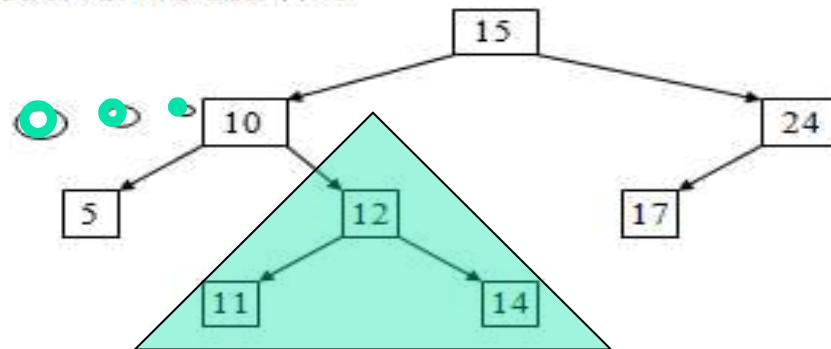


JAVA

# Esborrar node amb 2 fills

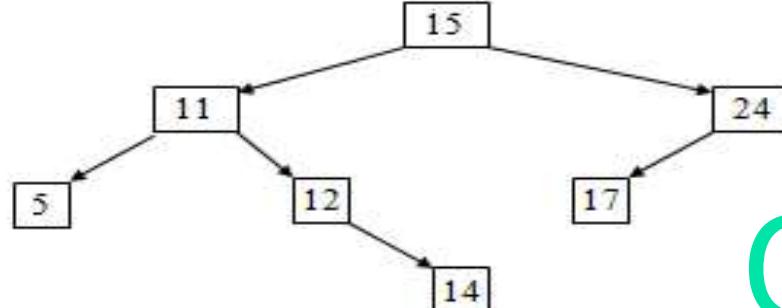
Exemple: el node a esborrar té dos fills

El node a  
borrar té  
dos fills



Opció 1

Es substitueix el node a eliminar pel node  
més petit del seu subarbre dret (no tindrà  
fill esquerre) i s'elimina aquest últim.



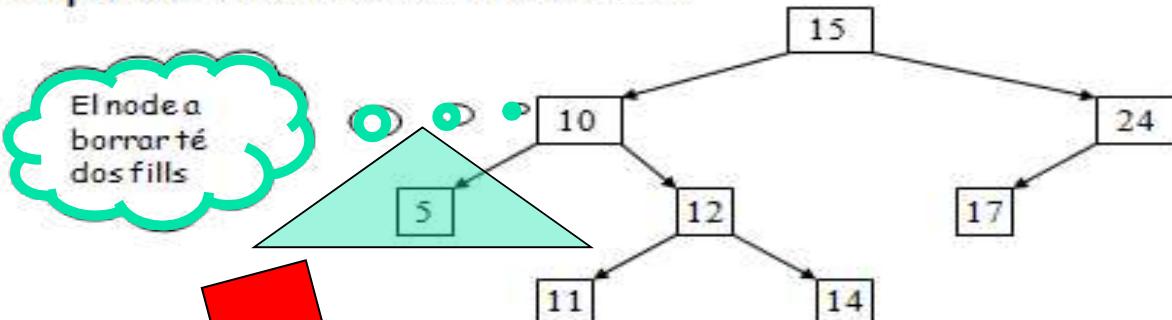
Opció 1



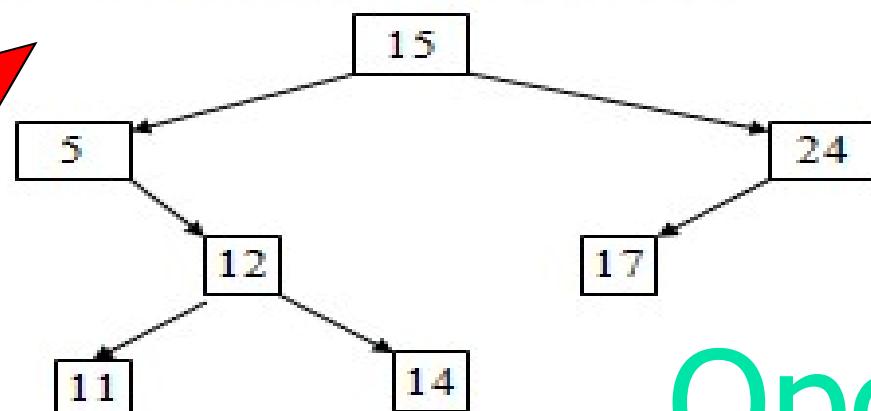
JAVA

# Esborrar node amb 2 fills

Exemple: el node a esborrar té dos fills



Es substitueix el node a eliminar pel node més gran del seu subarbre esquerre (no tindrà fill dret) i s'elimina aquest últim.



Opció 2



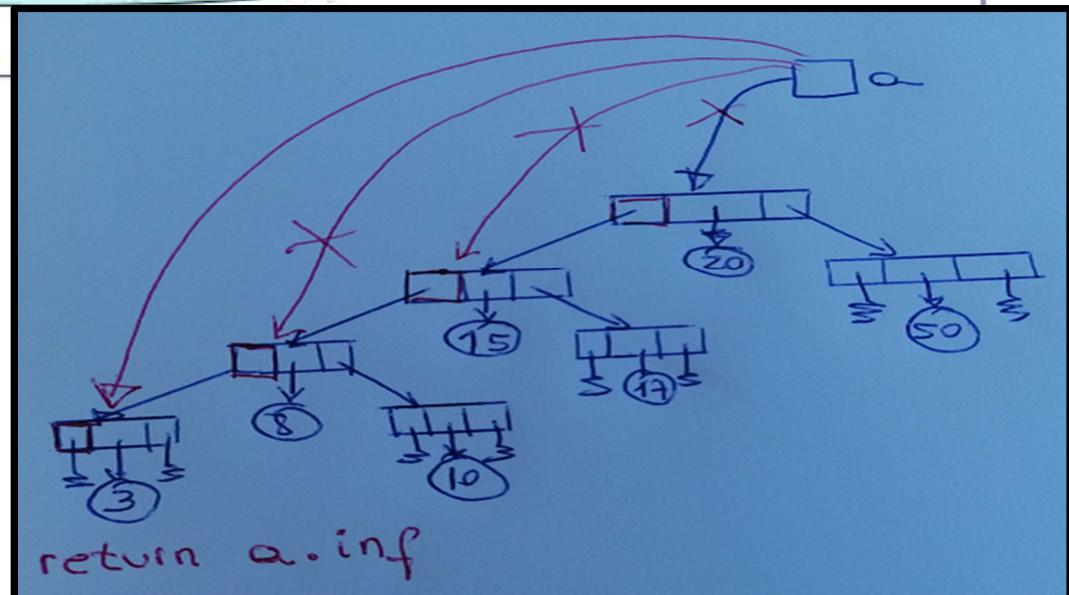
JAVA



# Esborrar node amb 2 fills

```
public static Comparable<T> buscarMinim(NodeA a){  
    // l'invocador garanteix que a!=null  
  
    // baixar per l'esquerra tant com es pugui  
    while (a.esq!=null) {  
        a = a.esq;  
    }  
  
    // qui ja no té fill esquerre és el mínim  
    // (no té ningú més petit que ell)  
    return Comparable<T>)(a.inf)  
}
```

**IDEA:** baixar per l'esquerra tant com sigui possible





# Operació esborrar



```
private static NodeA esborrarMinim(NodeA a){  
    // l'invocador garanteix que a!=null  
  
    if (a.esq==null) return a.drt;  
    else {  
        a.esq = esborrarMinim(a.esq);  
        return a;  
    }  
}
```

Implementació recursiva

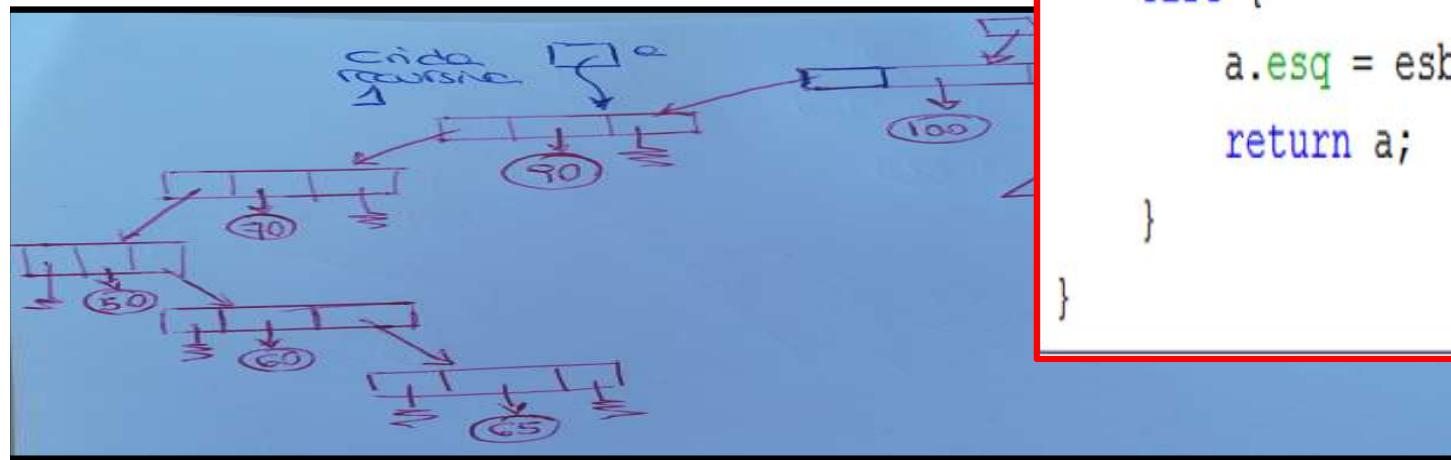
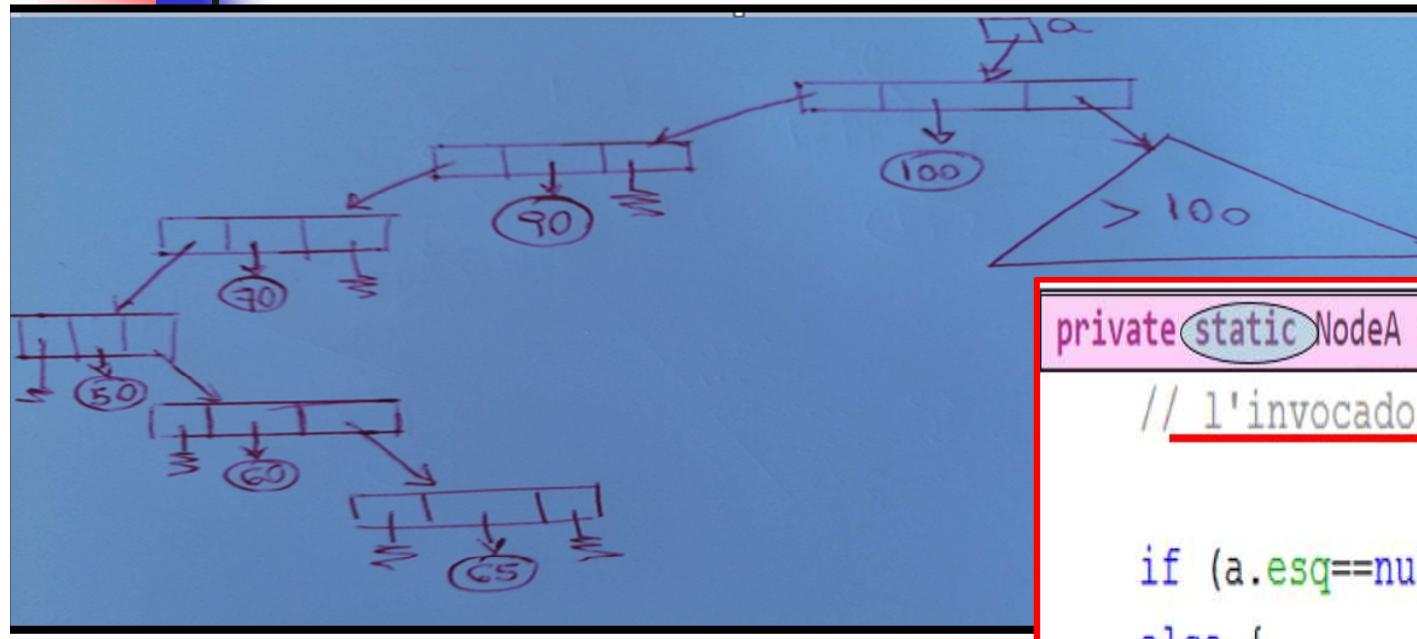


JAVA

# Operació esborrar



Volem esborrar el 50



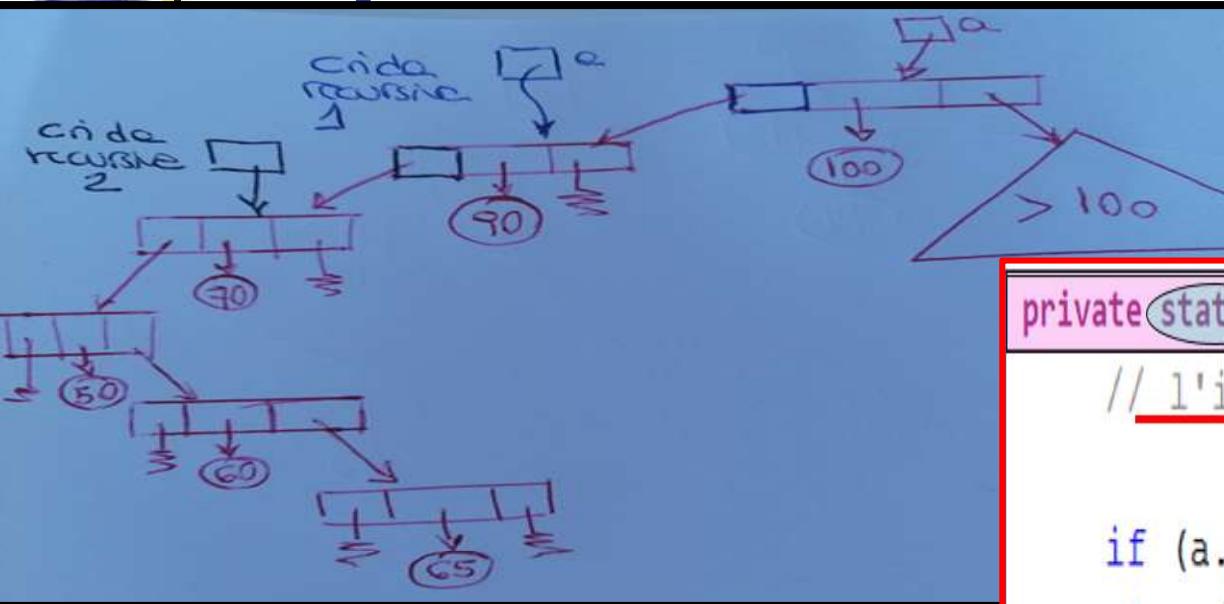
```
private static NodeA esborrarMinim(NodeA a){  
    // l'invocador garanteix que a!=null  
  
    if (a.esq==null) return a.drt;  
    else {  
        a.esq = esborrarMinim(a.esq);  
        return a;  
    }  
}
```



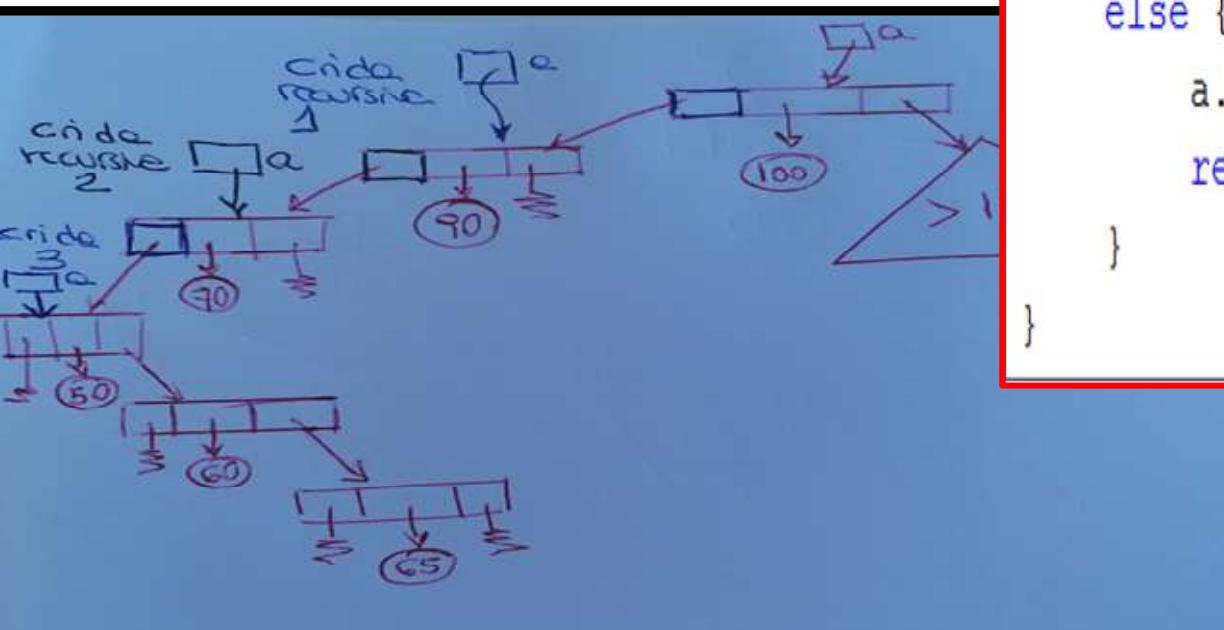
# Operació esborrar



Creador

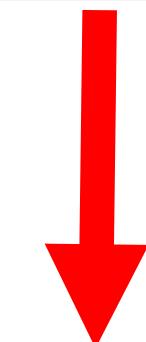
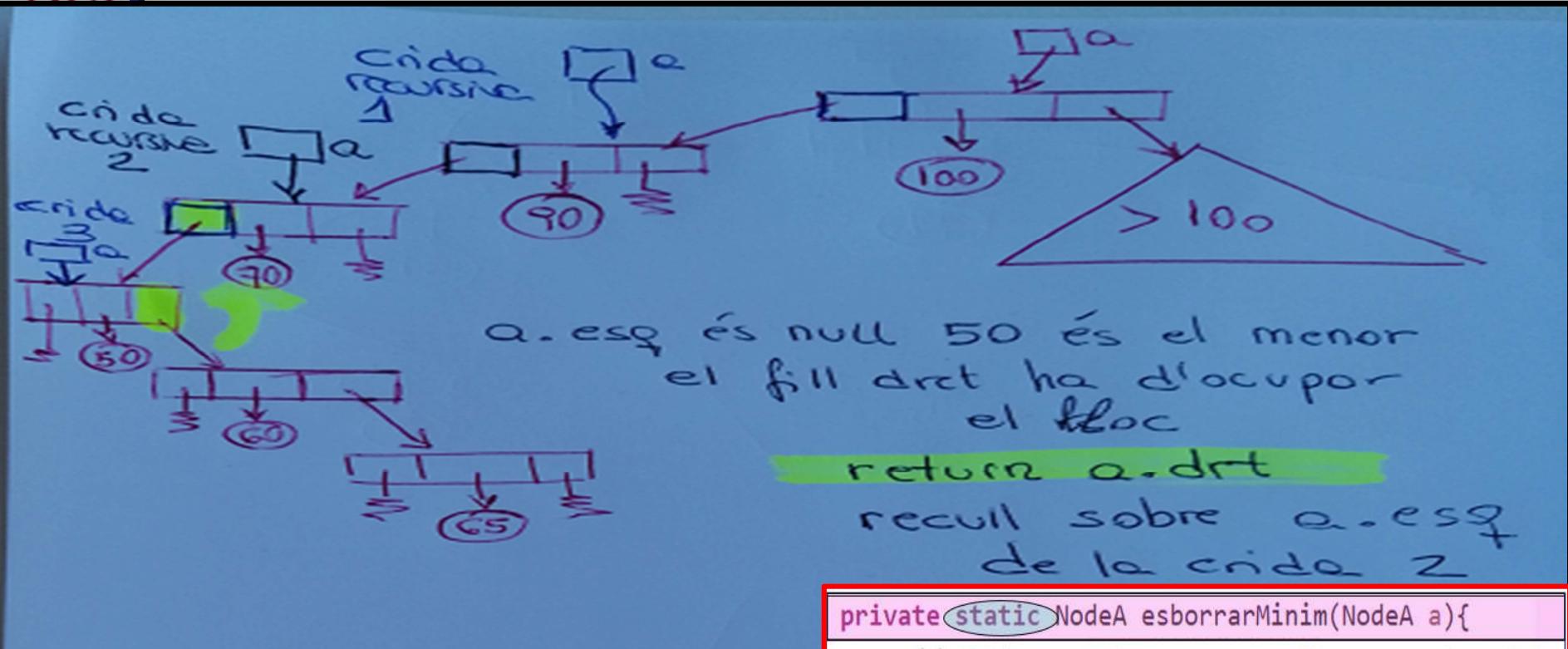


```
private static NodeA esborrarMinim(NodeA a){  
    // l'invocador garanteix que a!=null  
  
    if (a.esq==null) return a.drt;  
    else {  
        a.esq = esborrarMinim(a.esq);  
        return a;  
    }  
}
```





# Operació esborrar

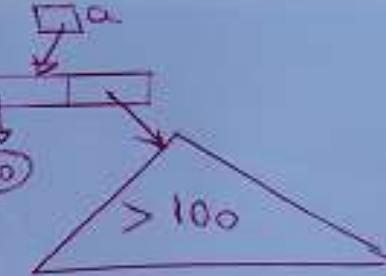
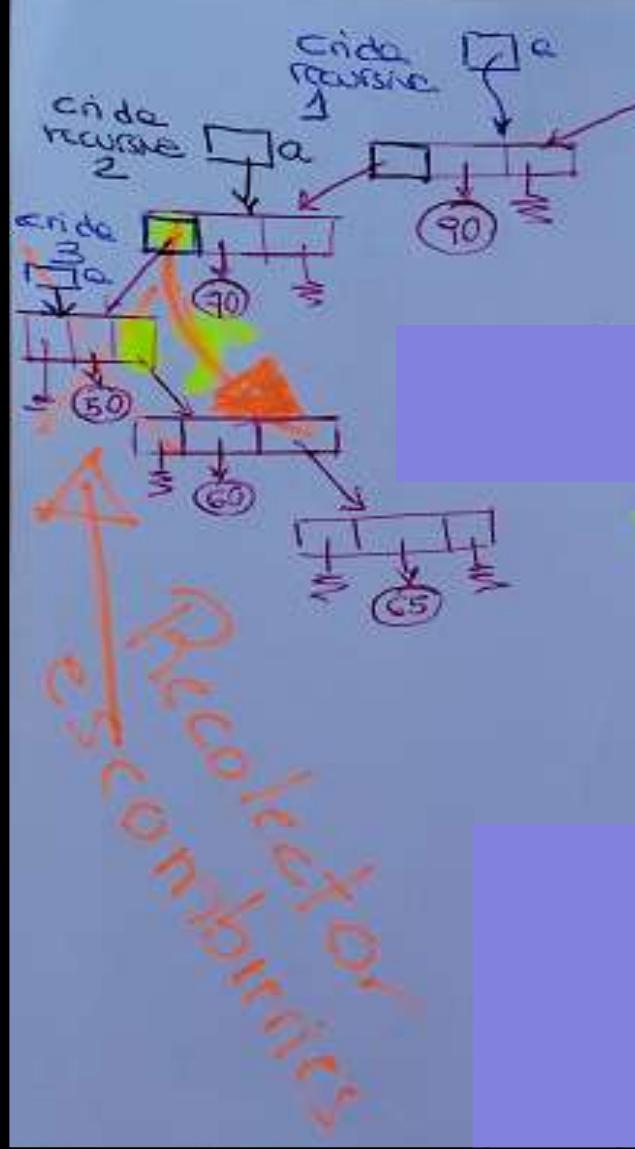


resultat

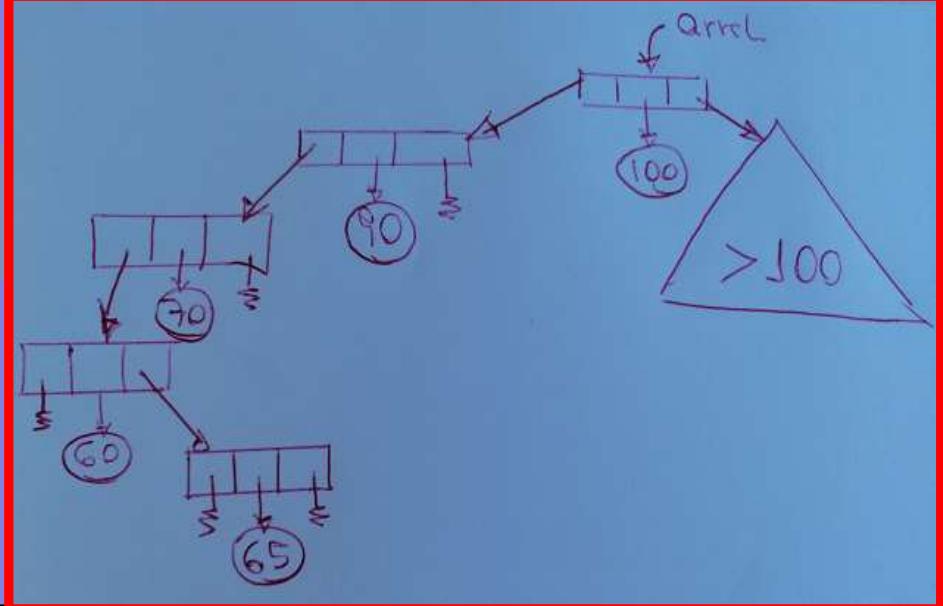


JAVA

# Operació esborrar



## Resultat final





JAVA

# Operació esborrar



```
@Override  
public void esborrar(Comparable<T> e) throws RuntimeException {  
    this.arrel = esborrarRec(this.arrel, e);  
}
```



JAVA

# Operació esborrar



```
private static NodeA esborrarRec(NodeA a, Comparable<T> e){
    if (a==null) throw new RuntimeException("inexistent "+e);

    int cmp = e.compareTo((T)a.inf);

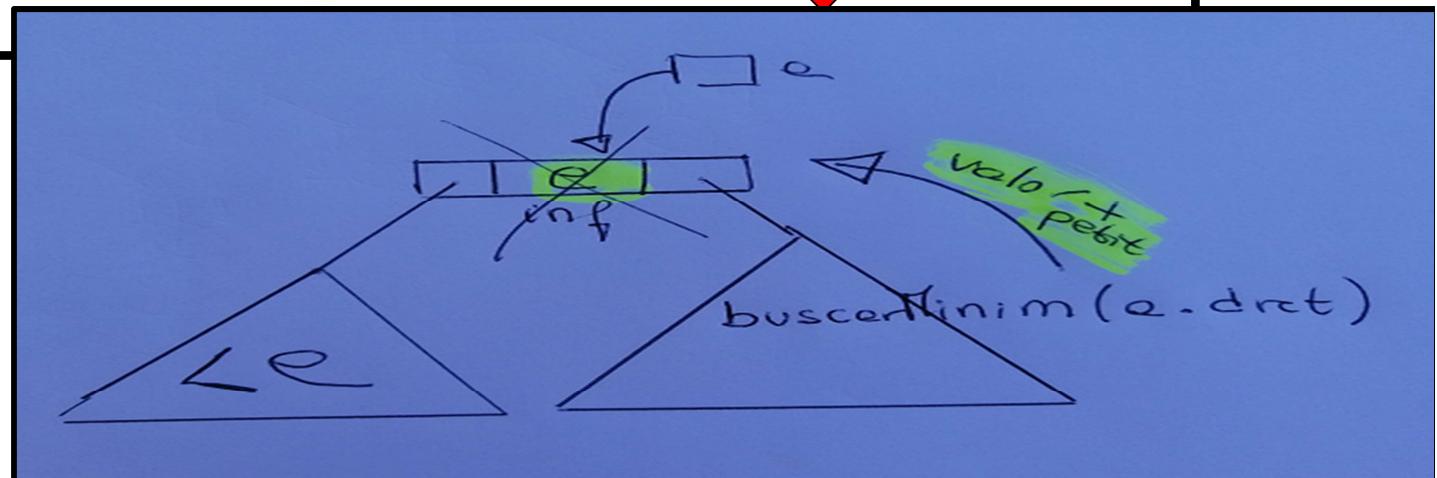
    if (cmp<0) { // cal eliminar per l'esquerra
        a.esq = esborrarRec(a.esq, e);
    }
    else {
        if (cmp>0) { // cal eliminar per la dreta
            a.drt = esborrarRec(a.drt, e);
        }
        else { // ja el tenim! Eliminem-lo
            if (a.esq==null && a.drt==null) a=null; // era una fulla
            else {
                if (a.esq!=null && a.drt!=null) { // té dos fills
                    a.inf = buscarMinim(a.drt);
                    a.drt = esborrarMinim(a.drt);
                }
                else { // només té un fill
                    if (a.esq==null) a=a.drt;
                    else a=a.esq;
                }
            }
        }
    }
    return a;
}
```



JAVA

Detall  
trobat  
element

```
else { // ja el tenim! Eliminem-lo
    if (a.esq==null && a.drt==null) a=null; // era una fulla
    else {
        if (a.esq!=null && a.drt!=null) { // té dos fills
            a.inf = buscarMinim(a.drt);
            a.drt = esborrarMinim(a.drt);
        }
        else { // només té un fill
            if (a.esq==null) a=a.drt;
            else a=a.esq;
        }
    }
}
```

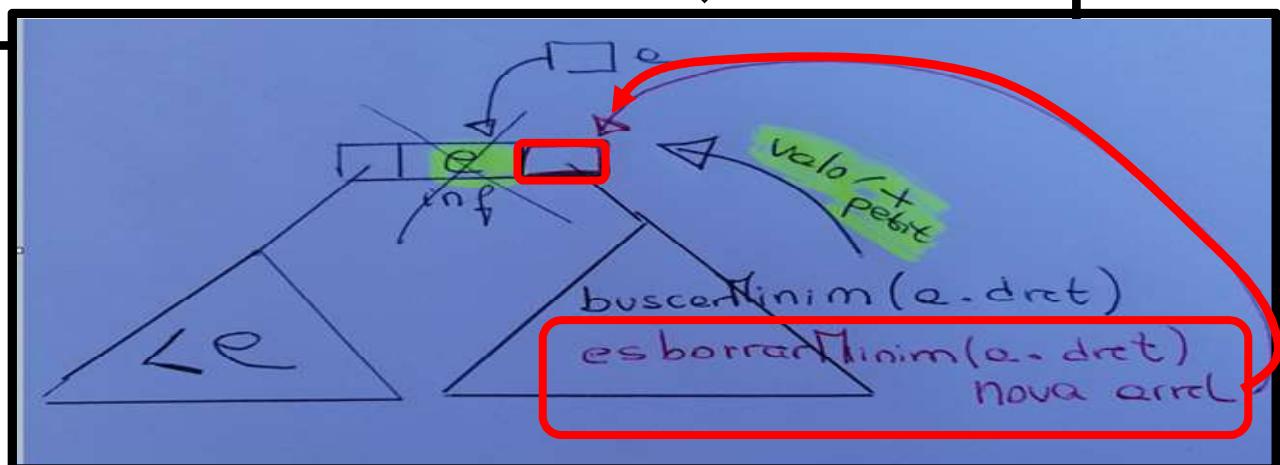




JAVA

Detall  
trobat  
element

```
else { // ja el tenim! Eliminem-lo
    if (a.esq==null && a.drt==null) a=null; // era una fulla
    else {
        if (a.esq!=null && a.drt!=null) { // té dos fills
            a.inf = buscarMinim(a.drt);
            a.drt = esborrarMinim(a.drt);
        }
        else { // només té un fill
            if (a.esq==null) a=a.drt;
            else a=a.esq;
        }
    }
}
```





JAVA

# Exercici



Torneu a escriure el mètode:

```
private static NodeA esborrarMinim(NodeA a){  
    //invocador garanteix que a != null
```

**Sense fer ús  
de la  
recusivitat.**

```
private static NodeA esborrarMinim(NodeA a){  
    // l'invocador garanteix que a!=null  
  
    if (a.esq==null) return a.drt;  
    else {  
        a.esq = esborrarMinim(a.esq);  
        return a;  
    }  
}
```



JAVA

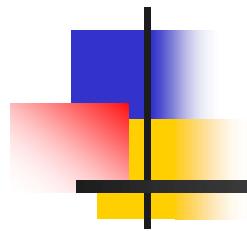
# Exercici - Solució

```
private NodeA esborrarMinim(NodeA a) {
    //invocador garanteix que a != null
    NodeA aux = a;
    if (aux.esq == null)
        return aux.drt;
    // Imprescindible treballar amb un auxiliar perquè cal renornar el node a
    while (aux.esq.esq != null)
        aux = aux.esq;
    aux.esq = aux.esq.drt;
    return a;
}
```

**Sense fer ús de la recusivitat.**

# Programació Avançada

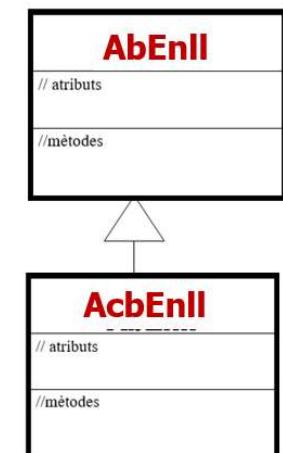
## Estructures no lineals



El T.A.D. ACB

Implementació 2

Recursivitat a la classe NodeA





JAVA

# Consideració ...



## Utilització de la nostra interfície Comparable

```
public interface Comparable {  
    boolean MenorQue(Comparable c);  
    boolean MajorQue(Comparable c);  
}
```

Comparable.java

Aquesta **nostra interfície** coincideix amb nom però:

- Té dos mètodes
- No està parametrizada



JAVA

# Consideració ...



Al no estar parametrizada la  
interfície Comparable tampoc ho és la interfície Acb

```
public interface Acb extends Ab {  
    void inserir(Comparable e) throws RuntimeException;  
    /*  
     * afegeix a l'ACB l'element especificat, el situa en el lloc adient segons  
     * l'ordenació, si no és possible es crea i llença una excepció  
     */  
    void esborrar(Comparable e) throws RuntimeException;  
    /*  
     * elimina de l'ACB l'element especificat, si no és possible es crea i  
     * llença una excepció  
     */  
    boolean membre(Comparable e);  
    /*  
     * comprova l'aparició o no de l'element a l'arbre, si no hi és retorna un  
     * valor de false  
     */  
}
```



# Implementació enllaçada



```
public class AcbEnll extends AbEnll implements Acb {  
    // NO s'afegeixen atributs  
    // l'atribut NodeA arrel ve d'herència i la classe hi té accés  
    public AcbEnll(){super();} //opcional  
    public void inserir(Comparable e) throws RuntimeException  
    {  
        if (arrel==null) arrel=new NodeA(e,null, null);  
        else arrel.inserir(e);  
    }  
}
```

Abans de fer la crida és imprescindible comprovar que **arrel** no sigui null per evitar el null pointer

La recursivitat es fa en els mètodes de la classe **NodeA**



# Implementació enllaçada



```
public void esborrar(Comparable e) throws RuntimeException{
    if (arrel==null) throw new Exception("l'arbre és buit");
    arrel=arrel.esborrar(e);
}

public boolean membre(Comparable e){
    if (arrel==null) return false;
    return (arrel.hiEs(e));
}
```

Abans de fer la crida és imprescindible comprovar que **arrel** no sigui null per evitar el null pointer



# Implementació enllaçada



Abans de fer la crida és imprescindible comprovar que l'objecte sobre el que es fa la crida recursiva no sigui null per evitar el null pointer

```
class NodeA {  
    Object inf;  
    NodeA esq, drt;  
  
    public boolean hiEs(Comparable e) {  
        if (e.MenorQue((Comparable) inf)) {  
            if (esq == null)  
                return false;  
            return (esq.hiEs(e));  
        } else if (e.MajorQue((Comparable) inf)) {  
            if (drt == null)  
                return false;  
            return (drt.hiEs(e));  
        } else  
            return (true);  
    }  
}
```



JAVA



# Implementació enllaçada

Aquest  
mètode no  
retornar res

```
public void inserir(Comparable e) throws Exception {  
    /* l'arbre no buit, localitzem el lloc */  
    if (e.MenorQue((Comparable) inf))  
        if (esq != null)  
            esq.inserir(e);  
        else // trobat lloc  
            esq = new NodeA(e, null, null);  
    else if (e.MajorQue((Comparable) inf))  
        if (drt != null)  
            drt.inserir(e);  
        else // trobat lloc  
            drt = new NodeA(e, null, null);  
    else /* l'element ja hi és */  
        throw new Exception("l'element ja hi és");  
} // fi mètode
```

Modifica el  
propi  
atribut



# Implementació enllaçada



```
public NodeA esborrar(Comparable e) throws Exception {  
    if (((Comparable) inf).MajorQue(e))  
        if (esq != null) {  
            esq = esq.esborrar(e);  
            return this;  
        } else  
            throw new Exception("no hi és");  
    else if (((Comparable) inf).MenorQue(e))  
        if (drt != null) {  
            drt = drt.esborrar(e);  
            return this;  
        } else  
            throw new Exception("no hi és");  
    else /* l'hem trobat */
```



JAVA

# Implementació enllaçada



```
if (esq != null && drt != null) // node intern
{
    inf = drt.buscarMinim();
    drt = drt.esborrar((Comparable)inf); // crida recursiva
    return this;
} else if (esq == null && drt == null)
    return null; // fulla
else if (esq == null)
    return drt;
else
    return esq;
}
```

Crida a  
mètodes de la  
propia classe  
NodeA

Crida recursiva directa, a  
ell mateix, sabem que  
**NO** ha d'eliminar un  
node amb dos fills.



JAVA

# Implementació enllaçada



```
private Comparable buscarMinim() {  
    if (esq == null)  
        return (Comparable) inf;  
    NodeA aux = esq;  
    while (aux.esq != null)  
        aux = aux.esq;  
    return (Comparable) aux.inf;  
}
```