



## Sessió 3

---

### Consideracions Aclariments



## Objectius

---

- Implementar un **Arbre de cerca binària (Acb)**
  - **Encadenat** vol dir que la gestió de l'espai es fa utilitzant "nodes" que es poden encadenar entre ells.
  - **Recorrible**, vol dir que es pot fer un recorregut de l'arbre per obtenir el seu contingut ordenat ascendentment o descendentment.

## La interfície Ab<E>

Defineix les operacions típiques dels Arbres Binaris.

- **arrel** proporciona l'arrel de l'arbre binari. Exception si és buit.
- **fillEsquerre** proporciona el subarbre esquerre, si en té. Si no en té el resultat és un arbre buit. Exception si el **this** és buit.
- **fillDret** proporciona el subarbre dret, si en té. Si no en té el resultat és un arbre buit. Exception si el **this** és buit.
- **abBuit** preguntar si és buit.
- **buidar**

## La interfície Acb<E extends Comparable>

Defineix les operacions típiques dels Arbres de Cerca binària. És una **extensió** de la interfície Ab.

- **inserir** un objecte comparable en la posició escaient. Excepció si no és possible (element repetit).
- **esborrar** un objecte comparable, si hi és. Retorna true si l'objecte hi era (i, per tant, l'ha esborrat). Retorna false si no hi era (i no l'ha pogut esborrar). **Ull, no llença excepcions!**
- **membre** donat un objecte comparable determina si aquest es troba o no en l'arbre.

## La classe NodeA<E>

És **imprescindible** que aquesta classe sigui només d'ús de les classes que heu d'implementar en aquesta pràctica. I que s'ajusti a la següent signatura:

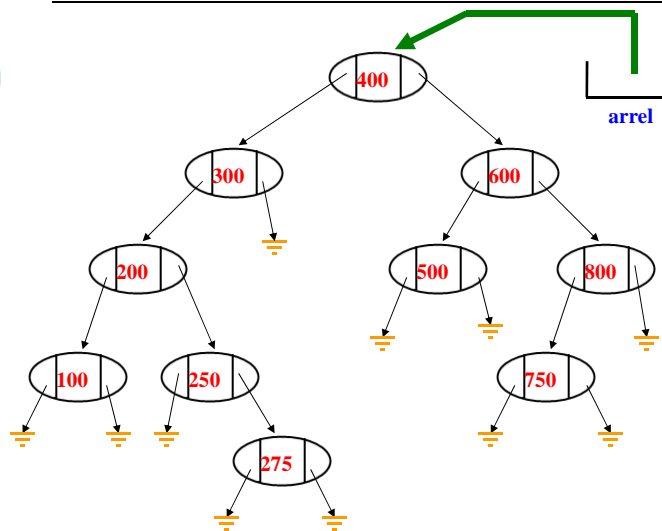
```
class NodeA<E>{  
    E inf;  
    Node<E> esq, dret;  
}
```

## Les classes AbEnll i AcbRecorrible

Internament, l'arbre s'organitza com una jerarquia de nodes. Seran suficients els següents atributs (bo i que, si voleu, en podeu usar més, sempre i quan siguin privats):

1. Una referència al node que fa d'arrel de l'arbre. (A la classe AbEnll)
2. Una Queue que pugui emmagatzemar el resultat d'una visita de l'arbre. (a la classe AcbRecorrible). **Podeu usar la vostra Cua de la Pràctica 2.**
3. Un int per determinar el sentit dels recorreguts, si s'efectuaran ascendentment o descendentment. (a la classe AcbRecorrible)

# Representació



## Els mètodes de recorregut

Els objectes de la classe **AcbRecorrible** proporcionen els següents mètodes que en permeten el recorregut:

- o **inilNordre** => informa a l'objecte de la intenció de sotmetre'l a un recorregut en inordre. L'objecte respon preparant-se per a ser recorregut.
- o **finalNordre** => informa de si el recorregut ha finalitzat (tots els elements han estat lliurats).
- o **segNordre** => proporciona un element del recorregut en inordre de l'arbre, que és el primer o el que segueix al donat la darrera invocació d'aquest mateix mètode.

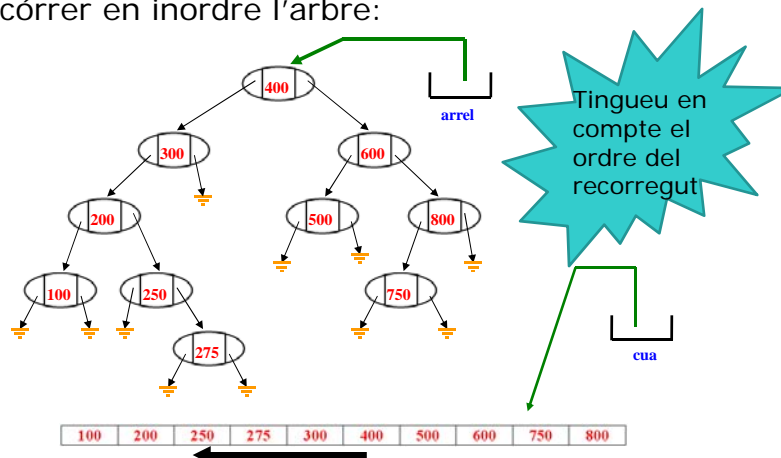
## Els mètodes de recorregut

Un recorregut d'un arbre de la classe AcbRecorrible s'efectuaria de la següent manera:

```
Comparable c;  
..  
arbre.iniInordre();  
while (!arbre.finalInordre()) {  
    c = arbre.segInordre();  
    // fer el que sigui amb c  
}
```

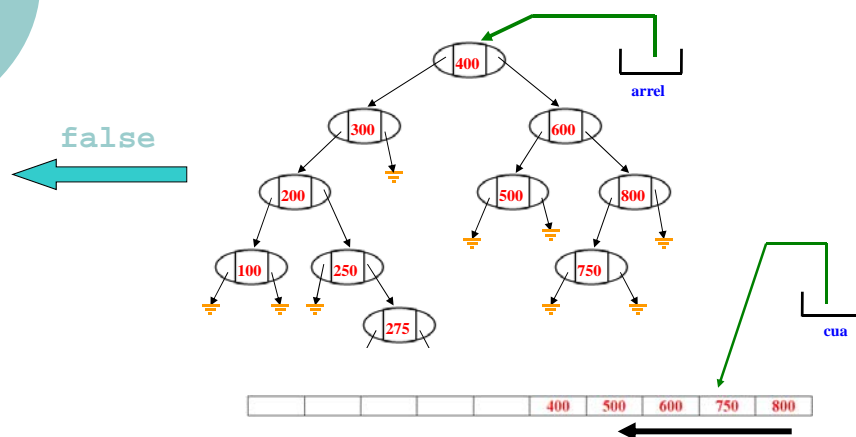
### Els mètodes de recorregut: inilnordre

Aquest mètode **omple la cua** amb el resultat de recórrer en inordre l'arbre:



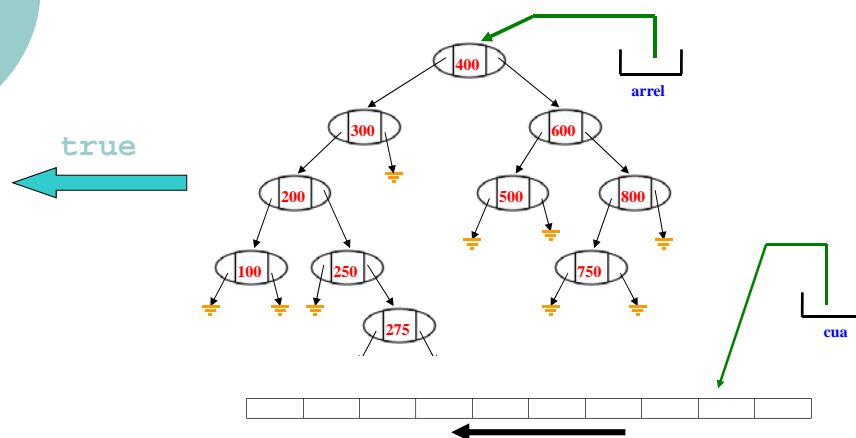
## Els mètodes de recorregut: finalInordre

Aquest mètode és limita a indicar si el recorregut ha finalitzat o no (= > és buida o no la cua)



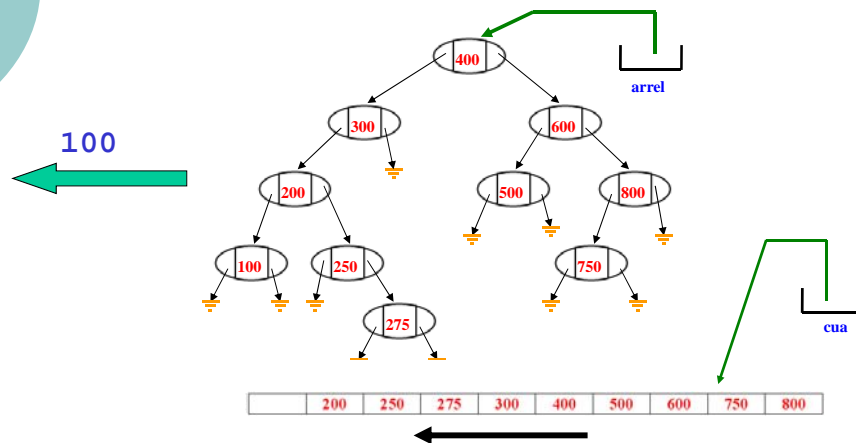
## Els mètodes de recorregut: finalInordre

Aquest mètode és limita a indicar si el recorregut ha finalitzat o no (= > és buida o no la cua)



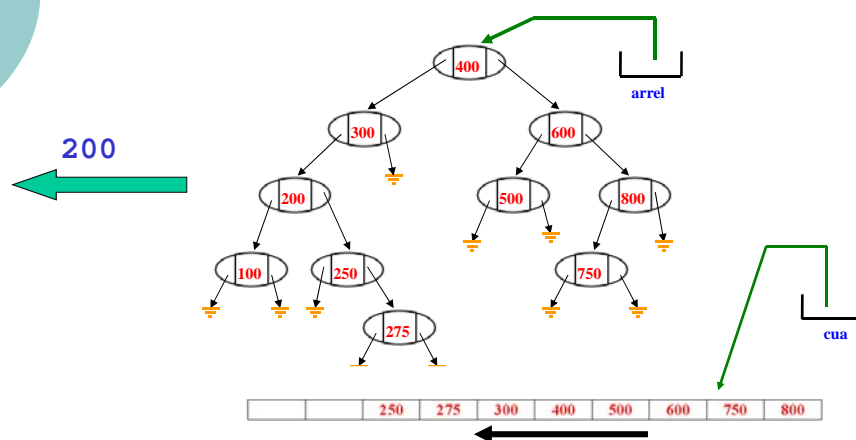
## Els mètodes de recorregut: segnordre

Desencua el primer element de la cua i el retorna



## Els mètodes de recorregut: segnordre

Desencua el primer element de la cua i el retorna



# Els mètodes de recorregut: seglinordre

Desencua el primer element de la cua i el retorna

```
graph TD; 400((400)) --> 300((300)); 400 --> 600((600)); 300 --> 200((200)); 300 --> null1[ ]; 200 --> 100((100)); 200 --> 250((250)); 250 --> 275((275)); 600 --> 500((500)); 600 --> 800((800)); 800 --> 750((750)); 100 --> null2[ ]; 275 --> null3[ ]; 500 --> null4[ ]; 750 --> null5[ ]; style null1 fill:none,stroke:none; style null2 fill:none,stroke:none; style null3 fill:none,stroke:none; style null4 fill:none,stroke:none; style null5 fill:none,stroke:none;
```

250

arrel

cua

			275	300	400	500	600	750	800
--	--	--	-----	-----	-----	-----	-----	-----	-----

# Els mètodes de recorregut: seglinordre

Desencua el primer element de la cua i el retorna

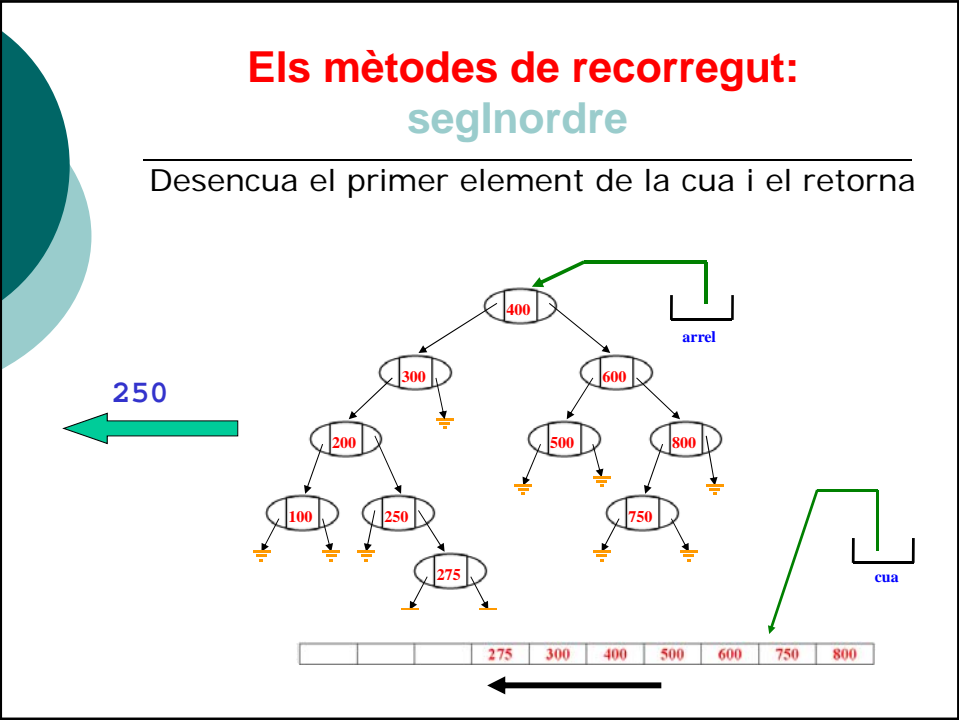
```
graph TD; 400((400)) --> 300((300)); 400 --> 600((600)); 300 --> 200((200)); 300 --> null1[ ]; 200 --> 100((100)); 200 --> 250((250)); 250 --> 275((275)); 600 --> 500((500)); 600 --> 800((800)); 800 --> 750((750)); 500 --> null2[ ]; 750 --> null3[ ]; 275 --> null4[ ]; 100 --> null5[ ]; 100 --> null6[ ]; 250 --> null7[ ]; 275 --> null8[ ]; 750 --> null9[ ]; 750 --> null10[ ]; 500 --> null11[ ]; 500 --> null12[ ]; 800 --> null13[ ]; 800 --> null14[ ]; 200 --> null15[ ]; 200 --> null16[ ]; 300 --> null17[ ]; 300 --> null18[ ]; 600 --> null19[ ]; 600 --> null20[ ]; 400 --> null21[ ]; 400 --> null22[ ]; 275 --> array[275]; 300 --> array[300]; 400 --> array[400]; 500 --> array[500]; 600 --> array[600]; 750 --> array[750]; 800 --> array[800];
```

250

arrel

cua

275 300 400 500 600 750 800



# Els mètodes de recorregut: segnordre

---

Desencua el primer element de la cua i el retorna

Diagram illustrating the dequeue operation in a binary tree. The tree structure is as follows:

- Root: 400 (labeled 'arrel')
- Level 1: 300 (left), 600 (right)
- Level 2: 200 (left of 300), 500 (left of 600), 800 (right of 600)
- Level 3: 100 (left of 200), 250 (right of 200), 750 (right of 800)
- Level 4: 275 (right of 250)

The value 275 is highlighted with a green arrow pointing to it from the root. A green arrow points from the root to the 'cua' label. A black arrow points from the value 275 to the value 300 in the array below.

				300	400	500	600	750	800
--	--	--	--	-----	-----	-----	-----	-----	-----

# Els mètodes de recorregut: segnordre

---

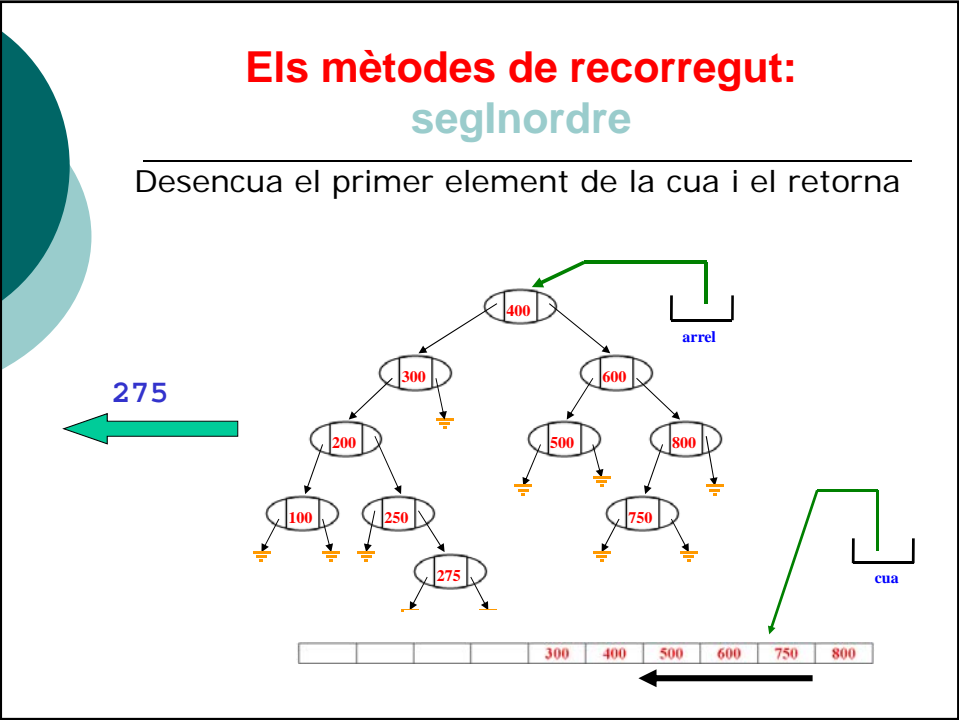
Desencua el primer element de la cua i el retorna

Diagram illustrating the dequeue operation in a binary tree. The tree structure is as follows:

- Root: 400 (labeled 'arrel')
- Level 1: 300 (left), 600 (right)
- Level 2: 200 (left of 300), 500 (left of 600), 800 (right of 600)
- Level 3: 100 (left of 200), 250 (right of 200), 750 (right of 800)
- Level 4: 275 (right of 250)

The value 275 is highlighted with a green arrow pointing to it from the root. A green arrow points from the root to the 'cua' label. A black arrow points from the value 275 to the value 300 in the array below.

				300	400	500	600	750	800
--	--	--	--	-----	-----	-----	-----	-----	-----





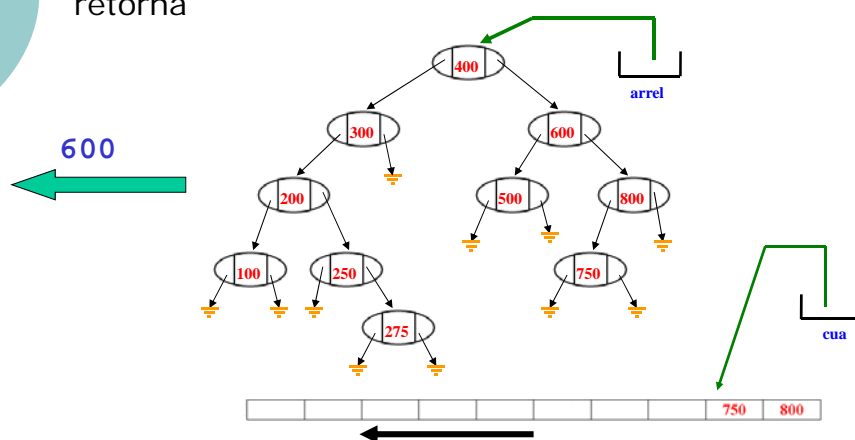
## Els mètodes de recorregut: seglnordre

Desencua el primer element de la cua i el retorna

● ● ●  
punts suspensius

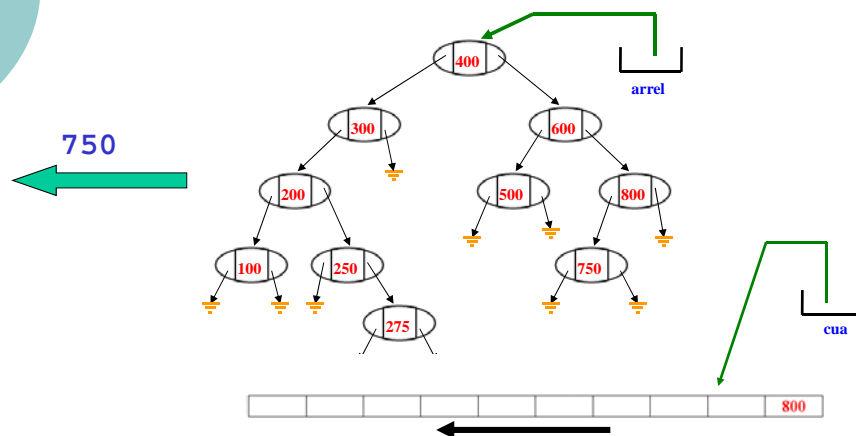
## Els mètodes de recorregut: seglnordre

Desencua el primer element de la cua i el retorna



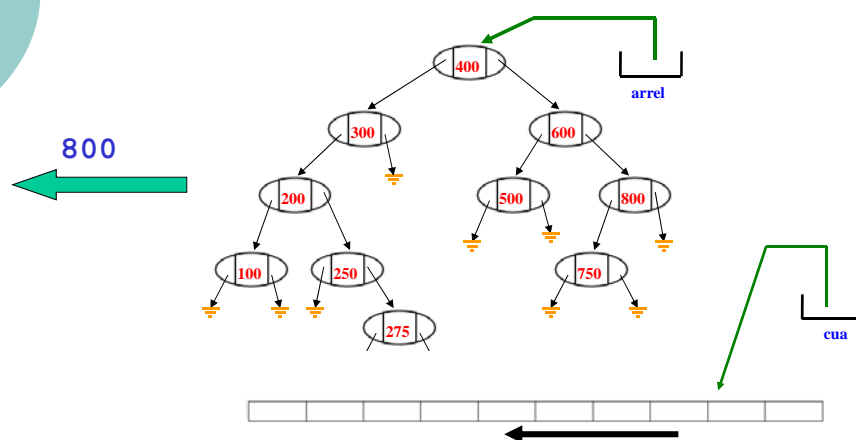
## Els mètodes de recorregut: segl'nordre

Desencua el primer element de la cua i el retorna



## Els mètodes de recorregut: segl'nordre

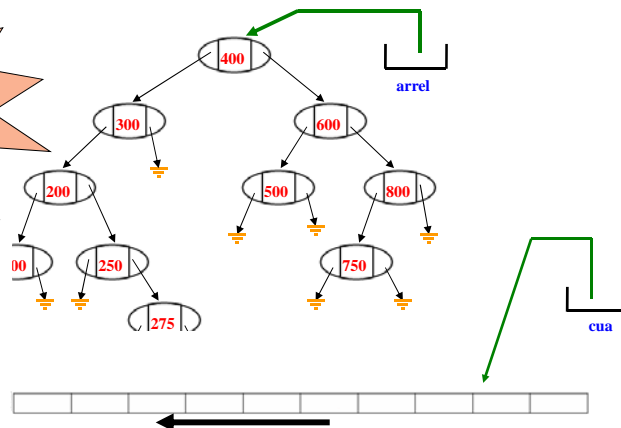
Desencua el primer element de la cua i el retorna



## Els mètodes de recorregut: segInordre

Desencua el primer element de la cua i el retorna, si pot

Recorregut  
Exception



## Els mètodes de recorregut: llençament d'excepcions

El mètode `segInordre` llença excepcions de la classe `ArbreException` quan l'estat de l'arbre no és apte per a invocar-lo. L'estat de l'arbre no és apte:

- o si `iniInordre` no ha estat invocat
- o si entre la invocació de `iniInordre` i `segInordre` s'ha produït una invocació de:
  - `buidar`
  - `inserir`
  - `esborrar`
  - `setOrdre`
- o Si la darrera invocació de `segInordre` ja va exhaurir el recorregut i no s'ha tornat a invocar `iniInordre`.

## El mètode inserir

- Afegeix un nou element (objecte que implementa la interfície Comparable) a l'arbre AcbRecorrible. La inserció es fa en la posició escaient.
- Llença una excepció de la classe `ArbreException` si l'element que es volia inserir ja existeix en l'arbre (n'hi ha un que és igual a ell).
- Totes les comparacions (menor, major, igual, ... ) s'han de fer invocant el mètode `compareTo` de la interfície Comparable de l'API de Java. El mètode `equals` **no** s'ha d'utilitzar per res.

## El mètode esborrar

- Intenta d'eliminar un objecte que sigui igual (`compareTo`  $\rightarrow$  0) a l'objecte donat com a paràmetre. Retorna `true` si tal objecte existeix i `false` en cas contrari.
- A diferència del que s'ha treballat a classe de teoria, aquest mètode **no** llença excepcions de cap mena. Si no troba l'objecte que es pretén esborrar es limita a retornar `false`.
- Totes les comparacions (menor, major, igual, ... ) s'han de fer invocant el mètode `compareTo`. El mètode `equals` **no** s'ha d'utilitzar per res.

## El mètode membre

- Determina si l'arbre conté un objecte que sigui igual (`compareTo`  $\rightarrow$  0) al donat com a paràmetre.
- Totes les comparacions (menor, major, igual, ... ) s'han de fer invocant el mètode `compareTo`. El mètode `equals` **no** s'ha d'utilitzar per res.

## Useu l'estructura de dades ...

- Cal provar el **correcte** funcionament de la nostra estructura.
- Per això **cal** fer la gestió d'arxius de text, la indicada en l'enunciat de la pràctica.
- **Us he** adjuntat uns fitxers de text per si voleu fer-los servir en les vostres proves de comprovació de funcionament.