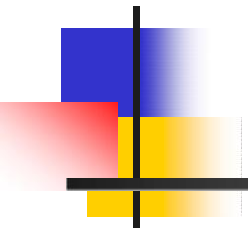


Programació Avançada

Aclariments Pràctica 4

Pràctica 4

Objectius:

Versió 2 de teoria

Objectiu 1: Implementació de l'estructura de dades Arbre de Cerca Binària (Acb). Generecitat usant la notació <E>

Objectiu 2: Implementació de mètodes aplicant la tècnica del Divideix i Venç.

Objectiu 3: Col·leccions de Java: Cua

Lliurament:

- 1.- Llistat imprès dels fonts: **Desembre – Inici Pràctica 5**
- 2.- Penjar el projecte al Moodle: **30/Novembre 8h**

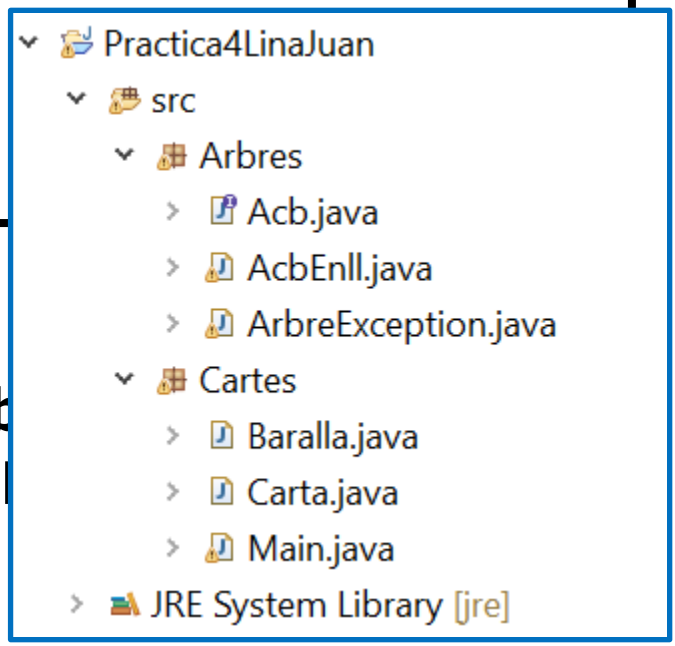
Pràctica 4

Objectius:

Objectiu 1: Implementació de l'estructura de dades Arbre de Cerca Binària (Acb). Generecitat usant la notació <E>

Objectiu 2: Implementació de mètodes aplicant la tècnica del Divideix i Vençe.

Objectiu 3: Col·leccions de Java: Cua



Lliurament:

- 1.- Llistat imprès dels fonts: **Desemb**
- 2.- Penjar el projecte al Moodle: **30/**

Pràctica 4

Enunciat. Implementar la col.lecció ACB

- Es vol implementar una estructura de dades que permeti obtenir la informació que emmagatzema de forma ordenada, ascendentment o descendentment segons vulgui l'usuari.
- Es vol que la nostra estructura sigui **recorrible**, obtenint ordenadament ascendentment o descendentment segons indiqui l'usuari, **un a un** els diferents elements que emmagatzema l'estructura.

Pràctica 4

Enunciat. Implementar la col.lecció ACB

- Es vol que la nostra estructura sigui **recorrible**, obtenint ordenadament ascendentment o descendentment segons indiqui l'usuari, **un a un** els diferents elements que emmagatzema l'estructura.

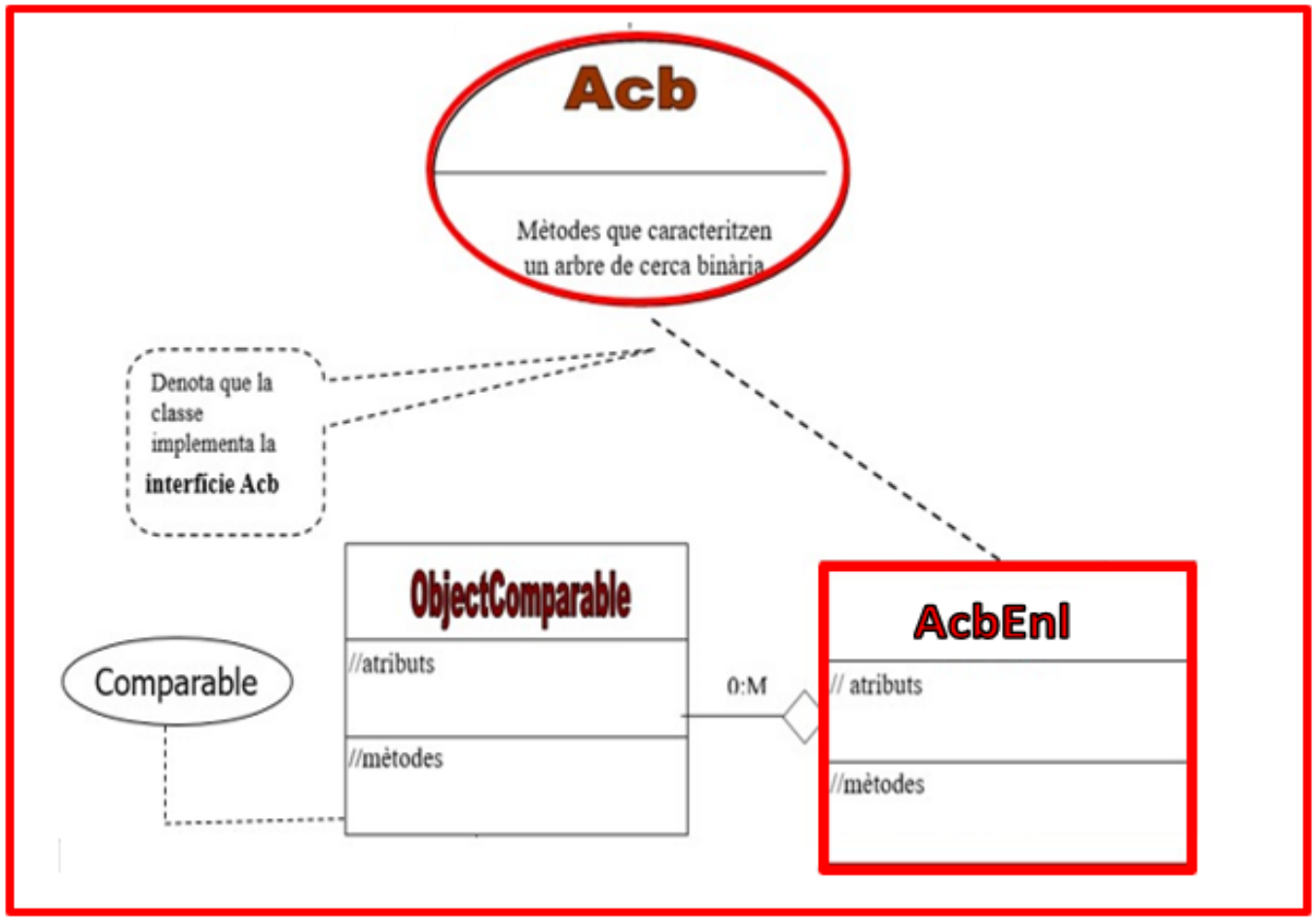
```
Acb<E> arbre;
arbre = new AcbEnll();
Comparable<E> c;
arbre.inserir(...);
...
arbre.iniRecorregut(true);
while (!arbre.finalRecorregut()) {
      c = arbre.segRecorregut();
      // fer el que sigui amb c
}
```

Caldrà indicar si es volen obtenir les dades ordenades en un sentit u altre

Pràctica 4

Part 1

Generecitat usant notació <E>



No muntarem herència a partir de la classe AbEnll

Pràctica 4

Classe privada NodeA

```
private class NodeA{
        NodeA esq, dret;
        E inf;
        //mètodes
}
```

Atributs (no es poden afegir atributs)

```
private NodeA arrel; //per referenciar l'arrel de l'arbre
private Queue<E> cua; //Nou atribut!
// Per aconseguir que l'arbre sigui recorrible
```

Mètodes (es poden afegir mètodes privats)

```
// Constructor i mètodes de la interfície  que s'ha d'implementar
// Per aconseguir que l'arbre sigui recorrible cal afegir els següents
// mètodes:
```

Obligatòriament s'ha d'implementar la **versió 2 de teoria**
Usant la interfície Comparable de java

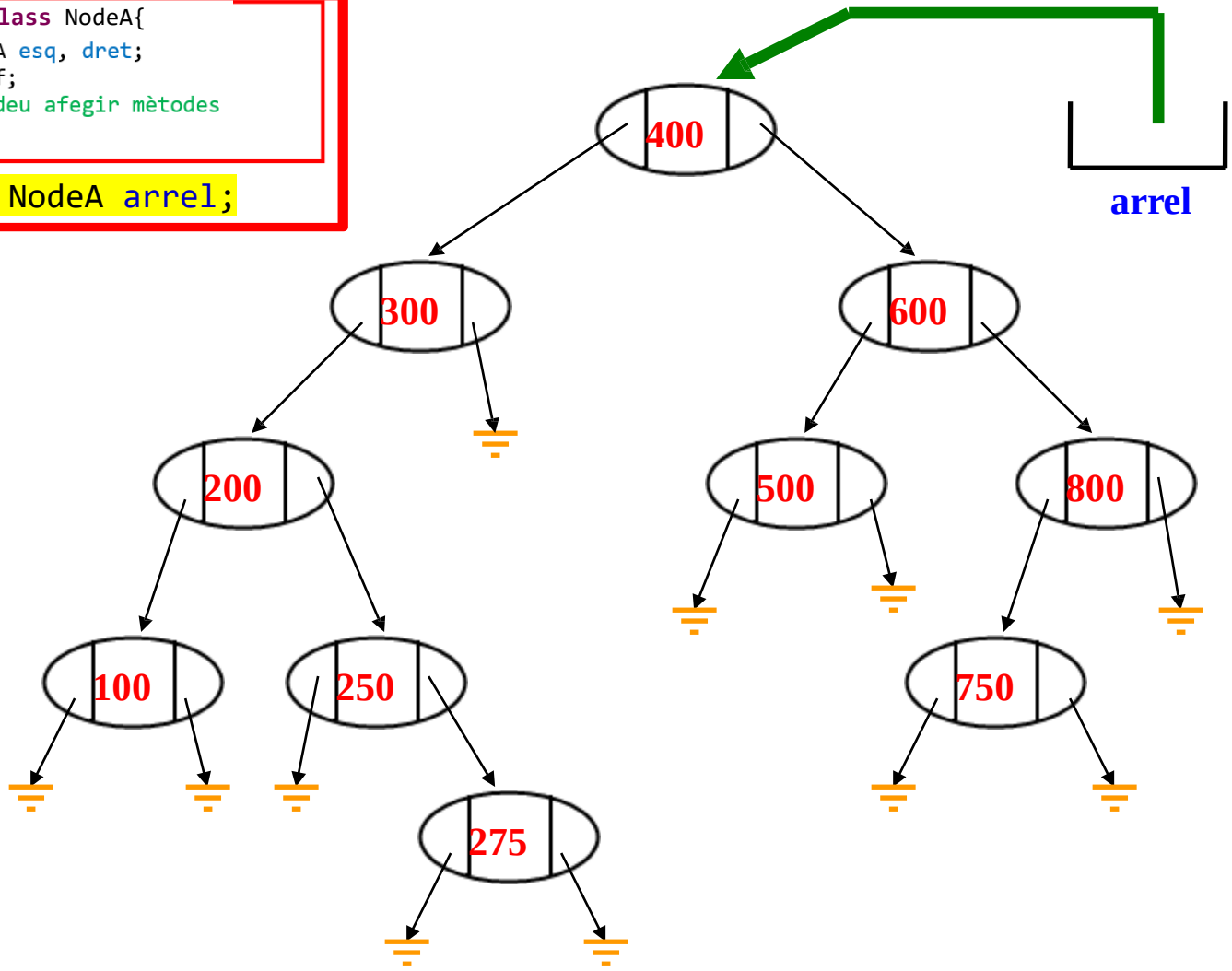
Pràctica 4

```
public  void  iniRecorregut (boolean  sentit)
/* prepara l'arbre per a ser recorregut en inordre. Després d'invocar
aquest mètode, la invocació del mètode segRecorregut retornarà el
primer element en inordre de l'arbre. Aquest mètode ha de emplenar la cua amb els
elements de l'arbre aplicant un recorregut en inordre. Cal tenir present el
paràmetre alhora d'emplenar la cua */
public  boolean finalRecorregut( )
/* retorna true si ja s'ha arribat al final del recorregut en inordre
de l'arbre. Això és si:
      - l'arbre és buit
      - la darrera vegada que es va invocar segRecorregut aquest mètode
      ja va retornar el darrer element en inordre de l'arbre.
Tot això és el mateix que dir que retorna true quan no té sentit
invocar el mètode segRecorregut */
public E segRecorregut () throws ArbreException
/*retorna el següent element en inordre, si n'hi ha.
Llença una excepció si:
      - abans d'invocar-lo no s'ha invocat el mètode iniRecorregut
      - la darrera vegada que es va invocar ja va retornar
      el darrer element del recorregut (finalRecorregut retornaria true)
      - s'invoca quan entre la invocació de iniRecorregut i la del mètode s'ha
      produït una modificació de l'arbre, això és, s'ha fet ús del mètode inserir,
      esborrar, buidar*/
```

Representació

```
private class NodeA{
      NodeA esq, dret;
      E inf;
      //podeu afegir mètodes
}
```

```
private NodeA arrel;
```



Els mètodes de recorregut

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

- **iniRecorregut** => informa a l'objecte de la intenció de sotmetre'l a un recorregut en inordre. L'objecte respon preparant-se per a ser recorregut. **Omplena** la Cua amb el recorregut.

- **finalRecorregut** => informa de si el recorregut ha finalitzat (tots els elements han estat lliurats).

- **segRecorregut** => 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 AcbEnll s'efectuaria de la següent manera:

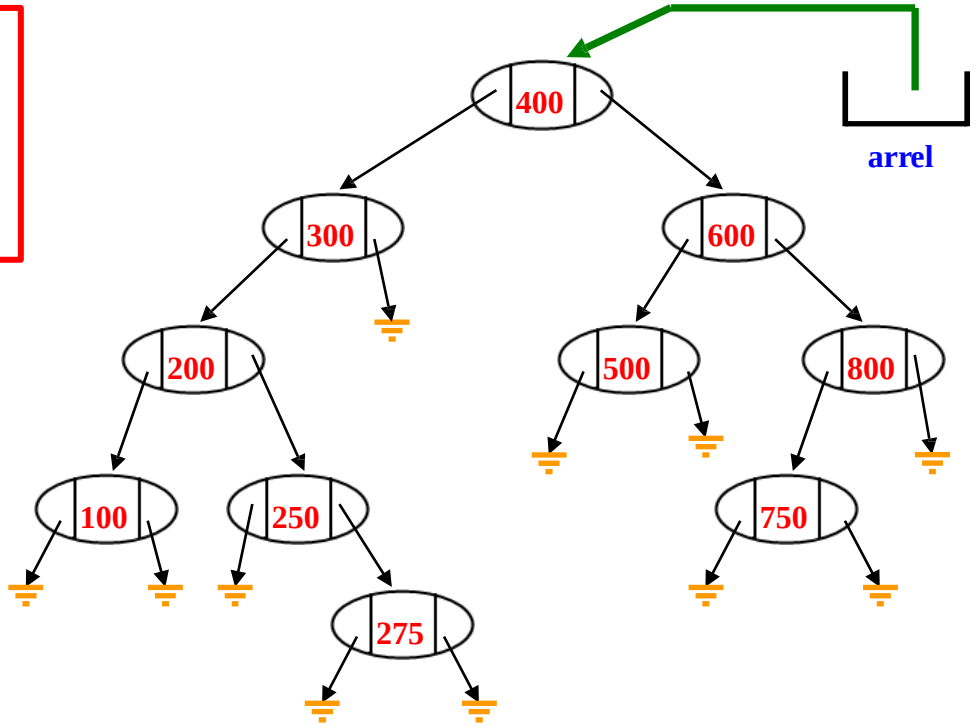
```
Comparable<E> c;
AcbEnll<E> arbre=new AcbEnll();
. . .
arbre.iniRecorregut(true);
while (!arbre.finalRecorregut()){
      c=arbre.segRecorregut();
      //fer el que sigui amb c
}
```

Els mètodes de recorregut:

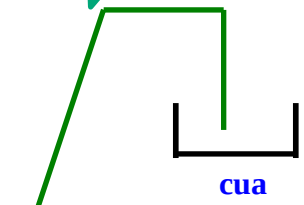
`iniRecorregut` (amb true)

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

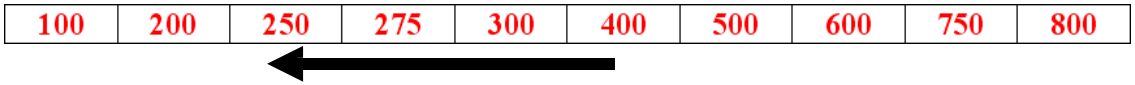
Suposem un arbre que emmagatzema enters



Tingueu en compte el sentit indicat en el paràmetre



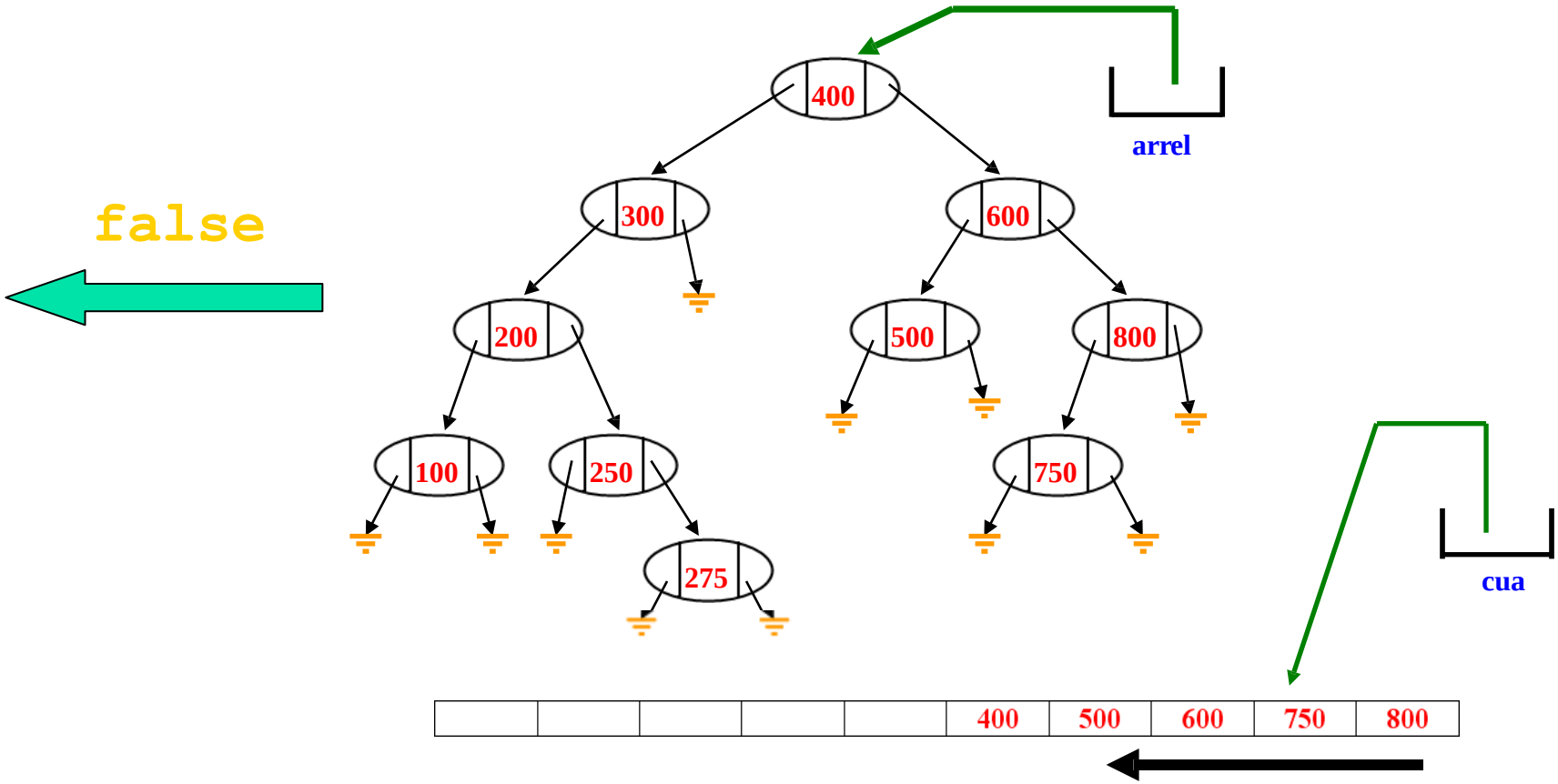
Atribut cua de la classe



Els mètodes de recorregut:

finalRecorregut

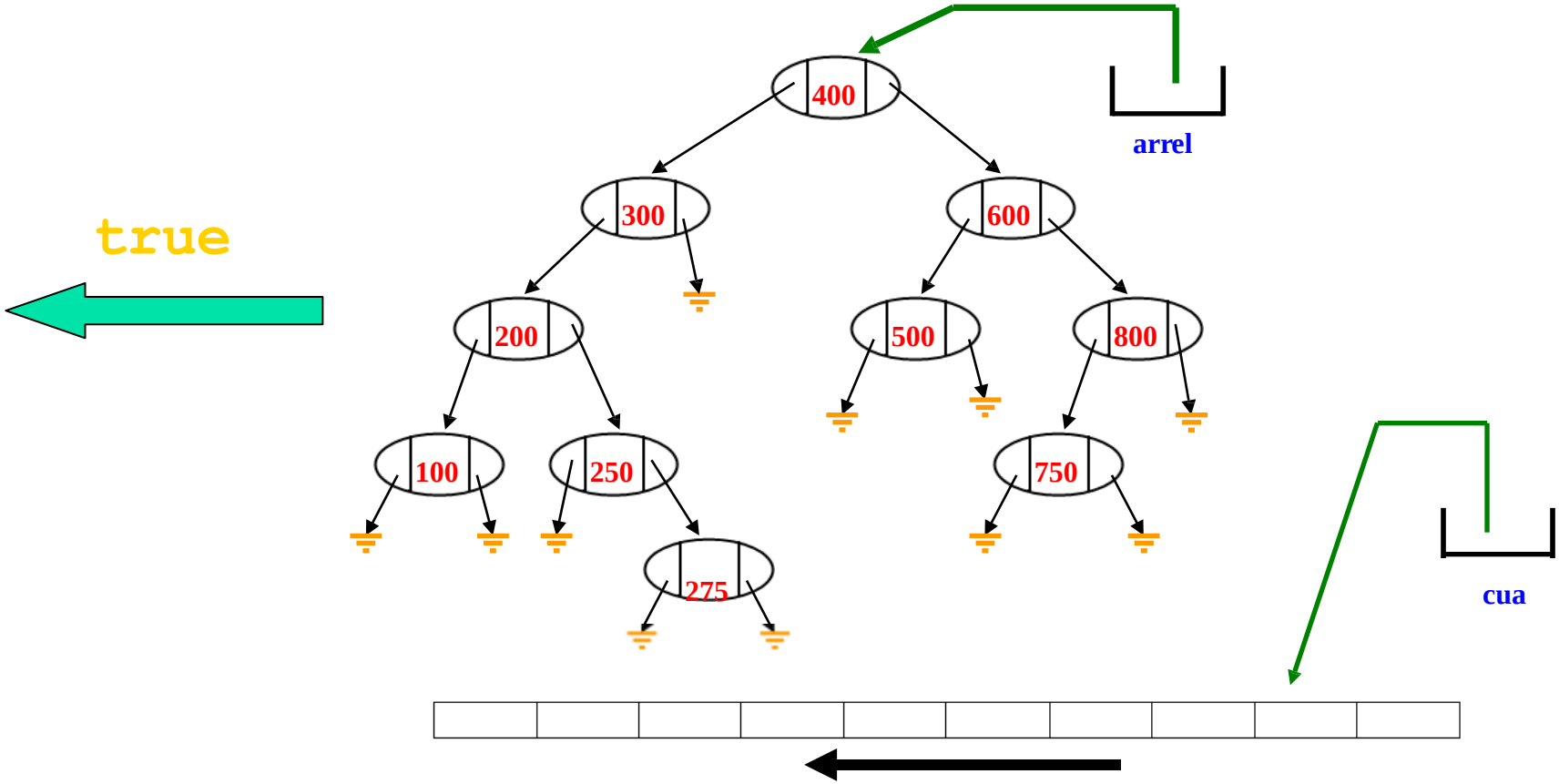
Aquest mètode es limita a indicar si el recorregut ha finalitzat o no (=> és buida o no la cua)



Els mètodes de recorregut:

finalRecorregut

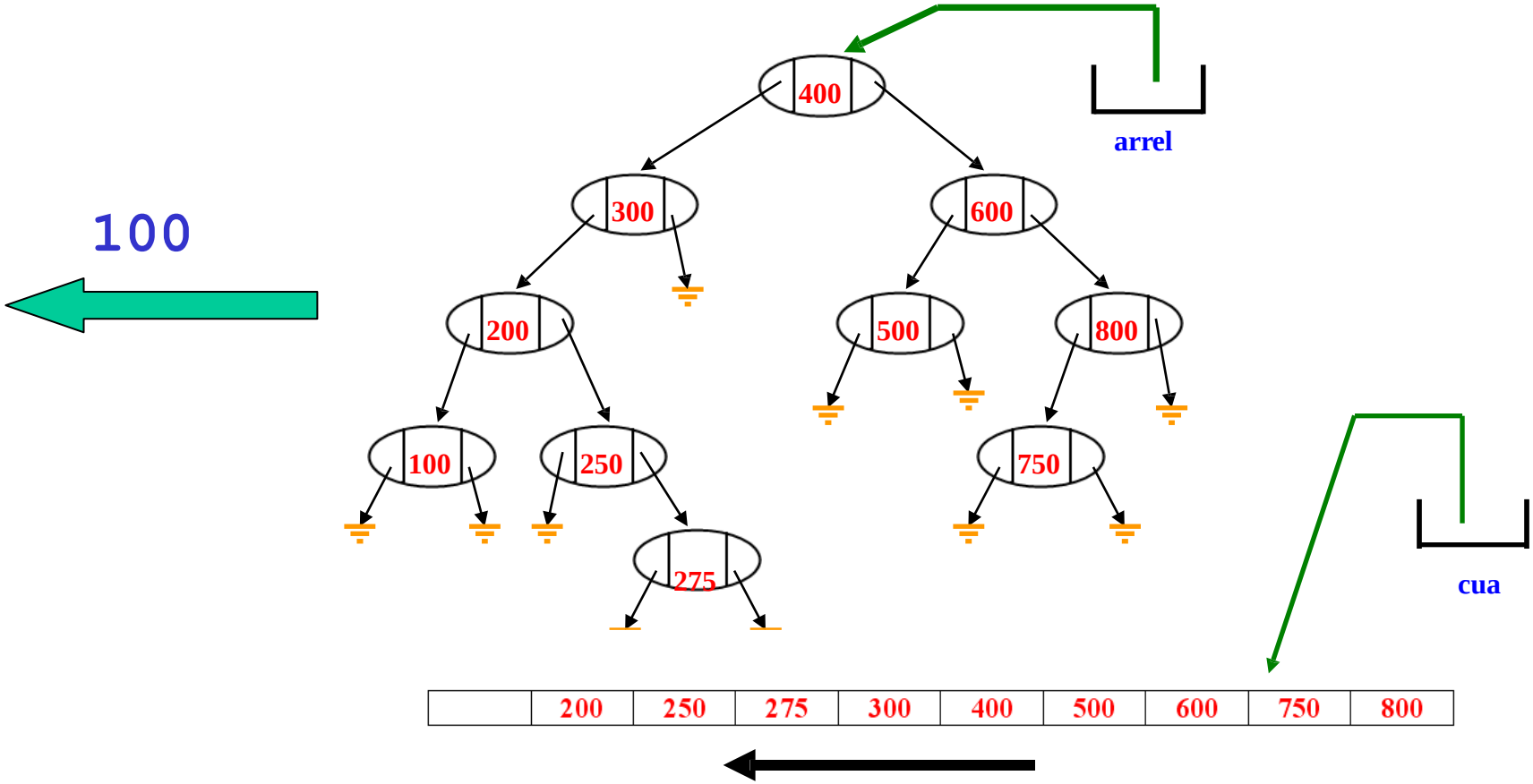
Aquest mètode és limita a indicar si el recorregut ha finalitzat o no (=> és buida o no la cua)



Els mètodes de recorregut:

segRecorregut

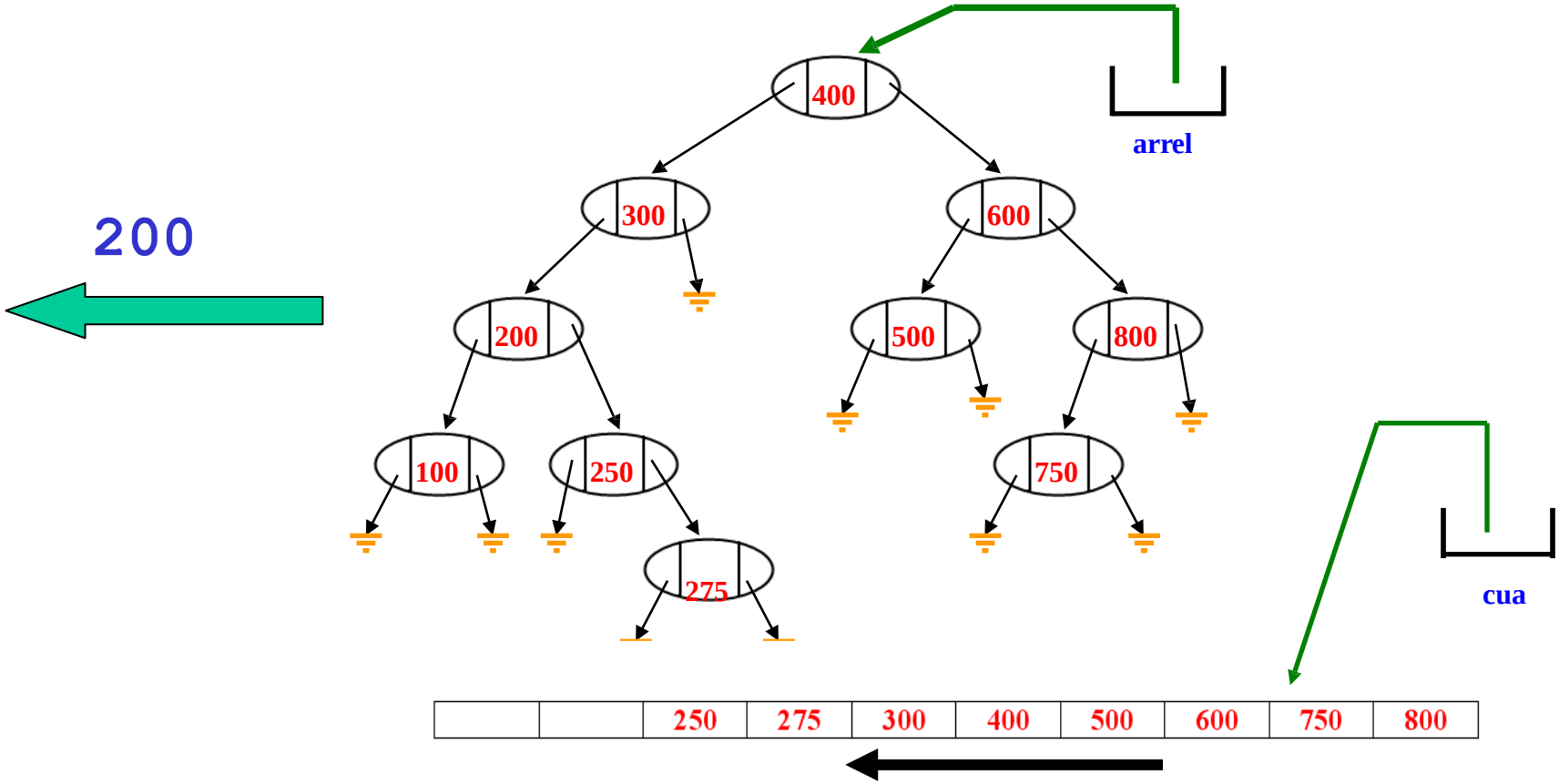
Desencua el primer element de la cua i el retorna



Els mètodes de recorregut:

segRecorregut

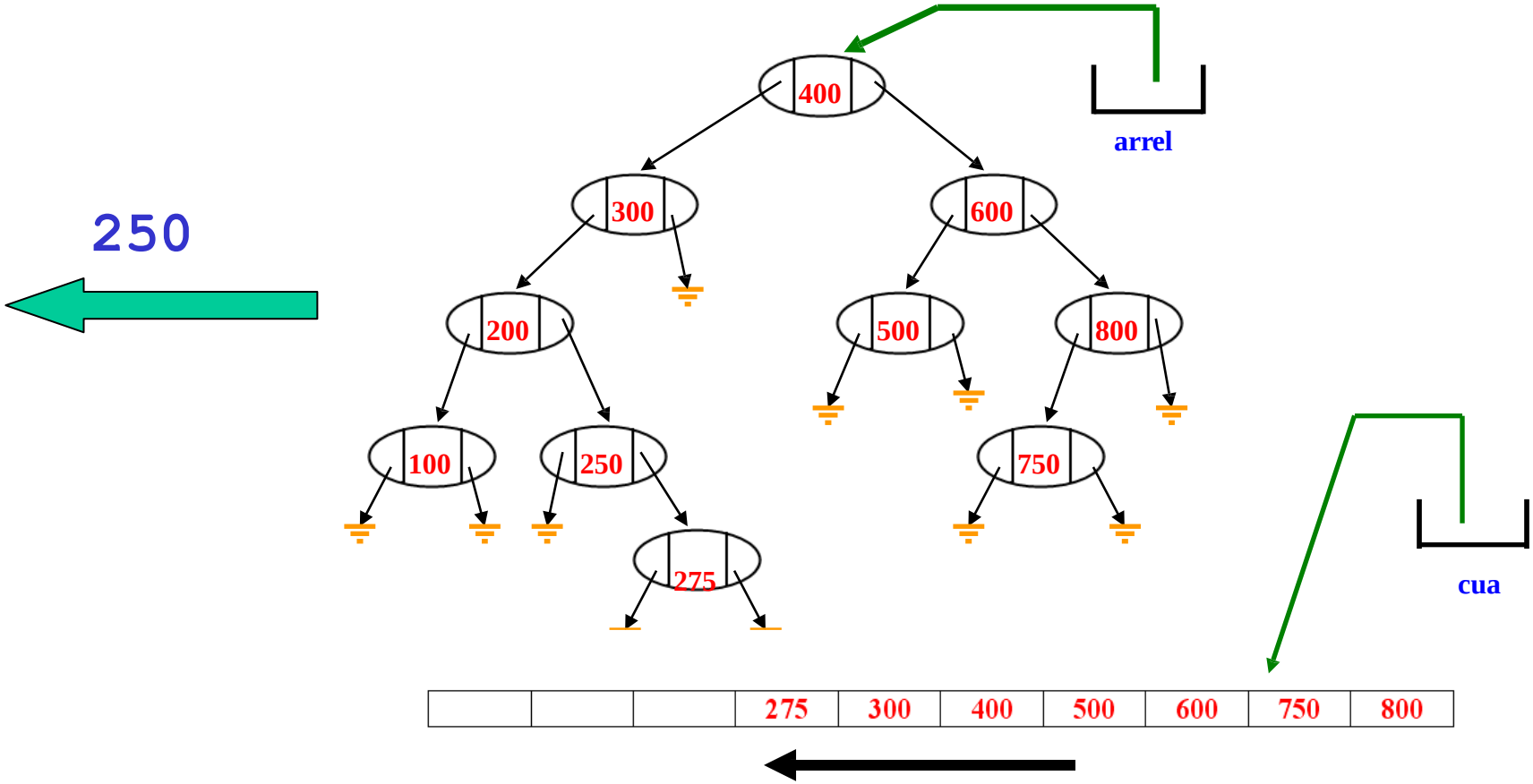
Desencua el primer element de la cua i el retorna



Els mètodes de recorregut:

segRecorregut

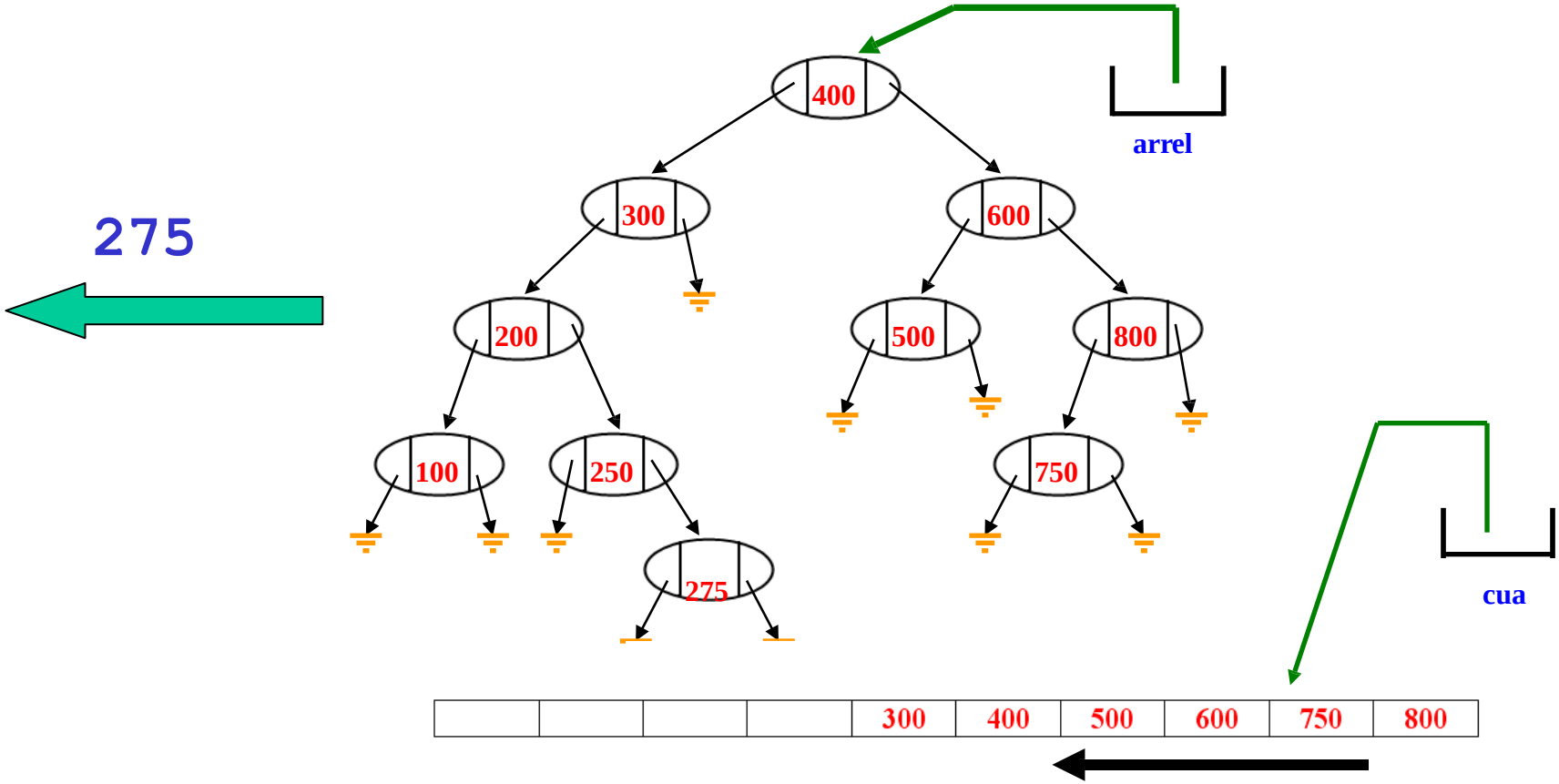
Desencua el primer element de la cua i el retorna



Els mètodes de recorregut:

segRecorregut

Desencua el primer element de la cua i el retorna



Els mètodes de recorregut:

segRecorregut

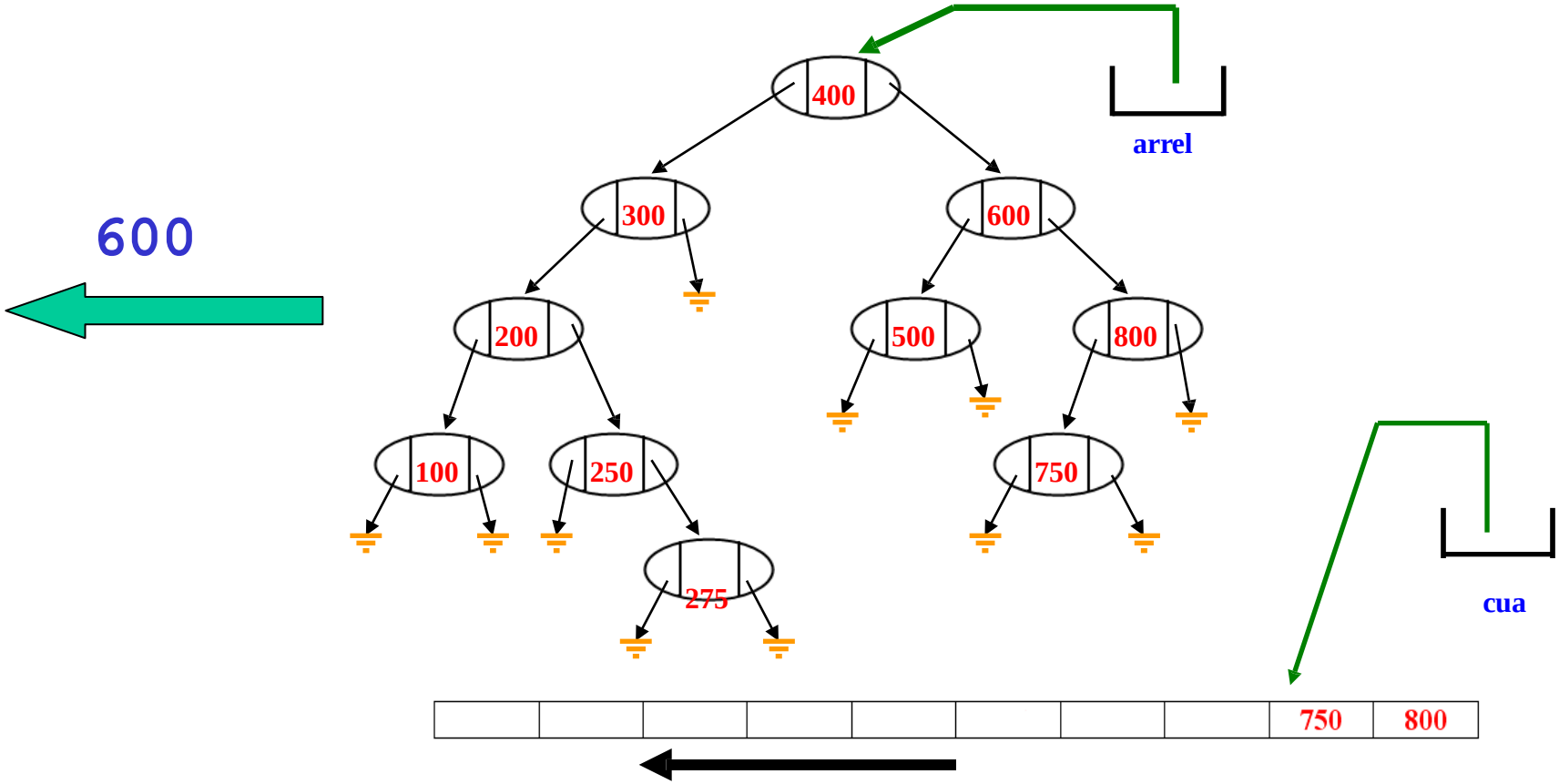
Desencua el primer element de la cua i el retorna

punts suspensius

Els mètodes de recorregut:

segRecorregut

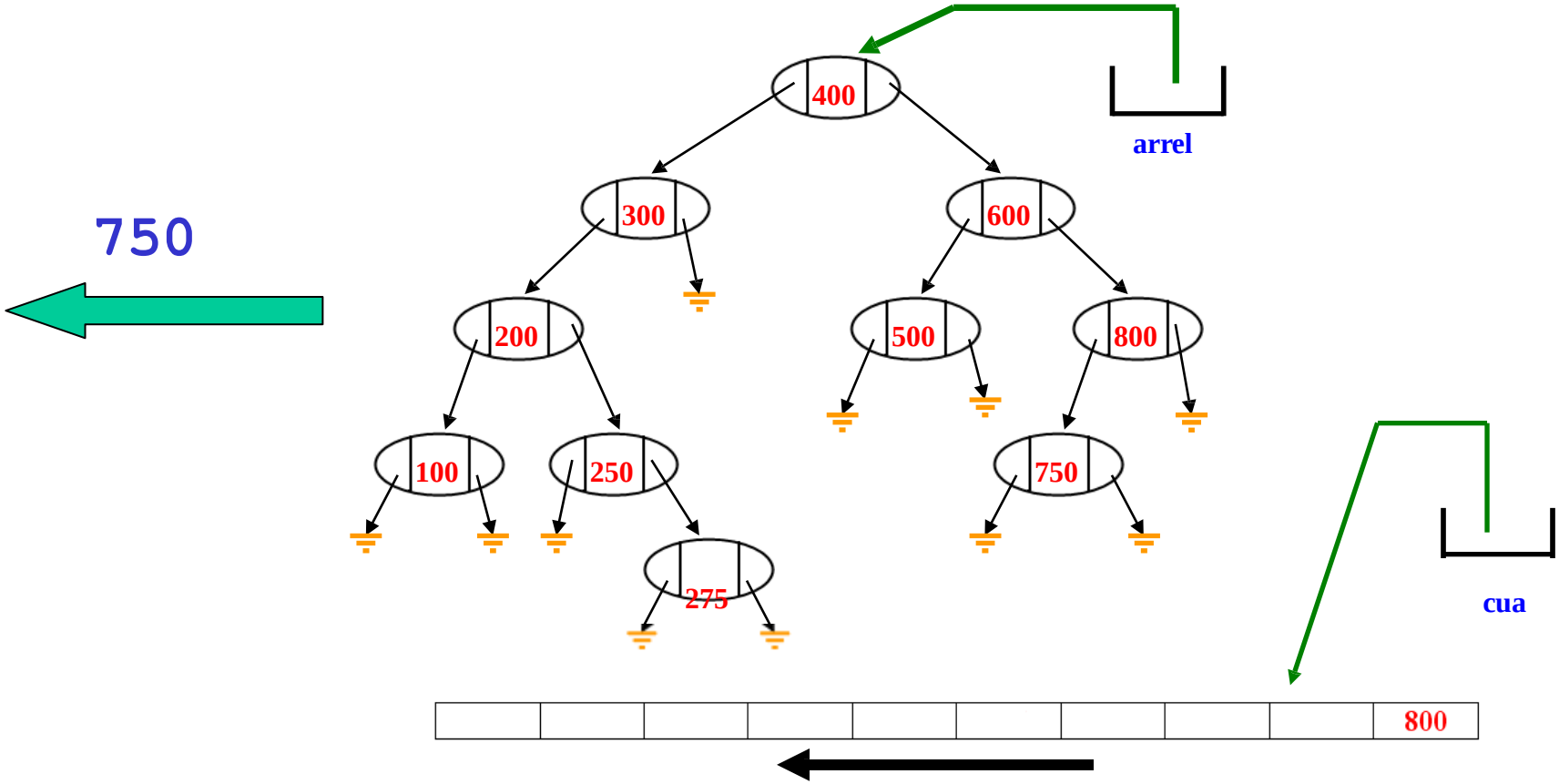
Desencua el primer element de la cua i el retorna



Els mètodes de recorregut:

segRecorregut

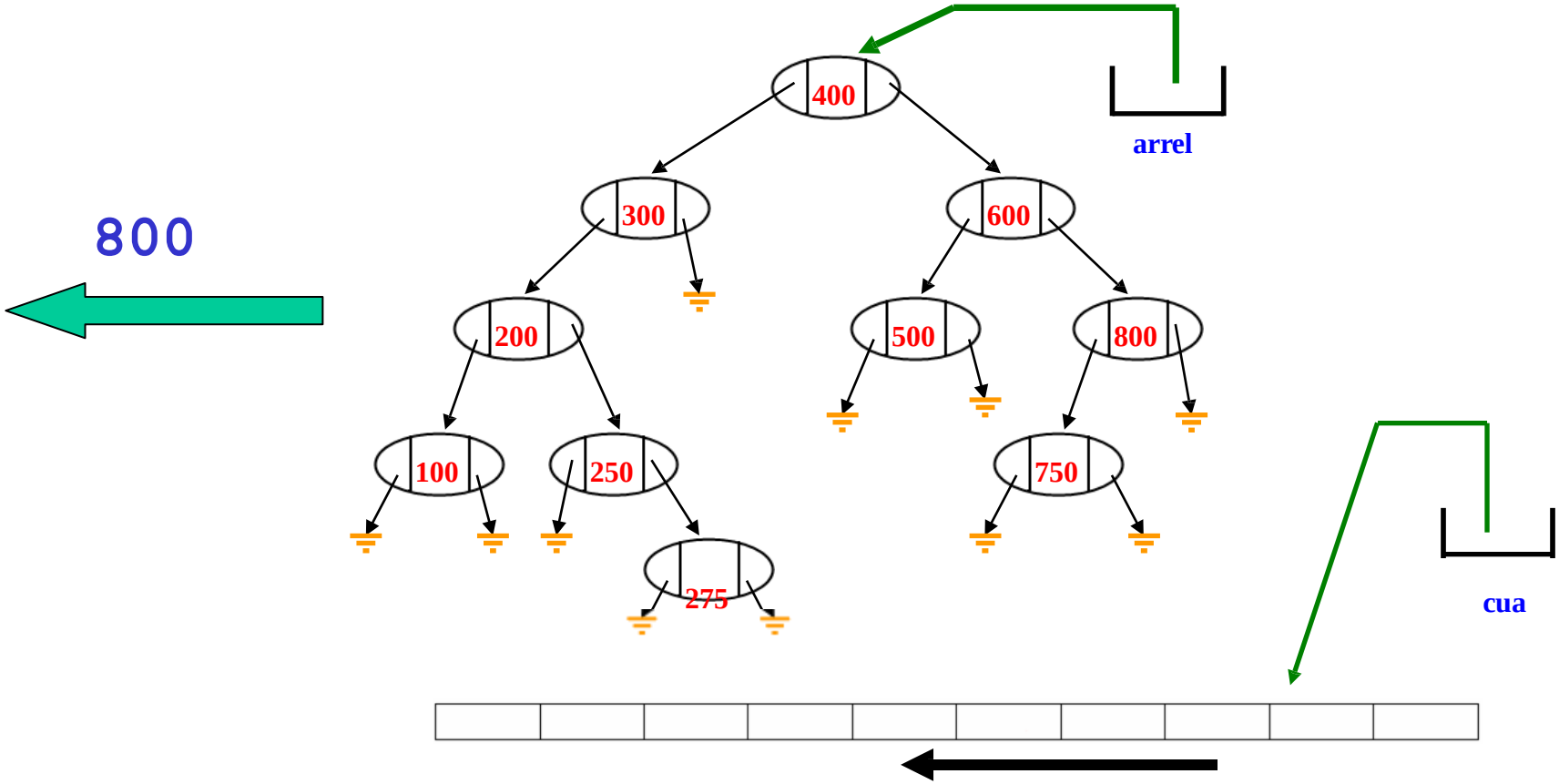
Desencua el primer element de la cua i el retorna



Els mètodes de recorregut:

segRecorregut

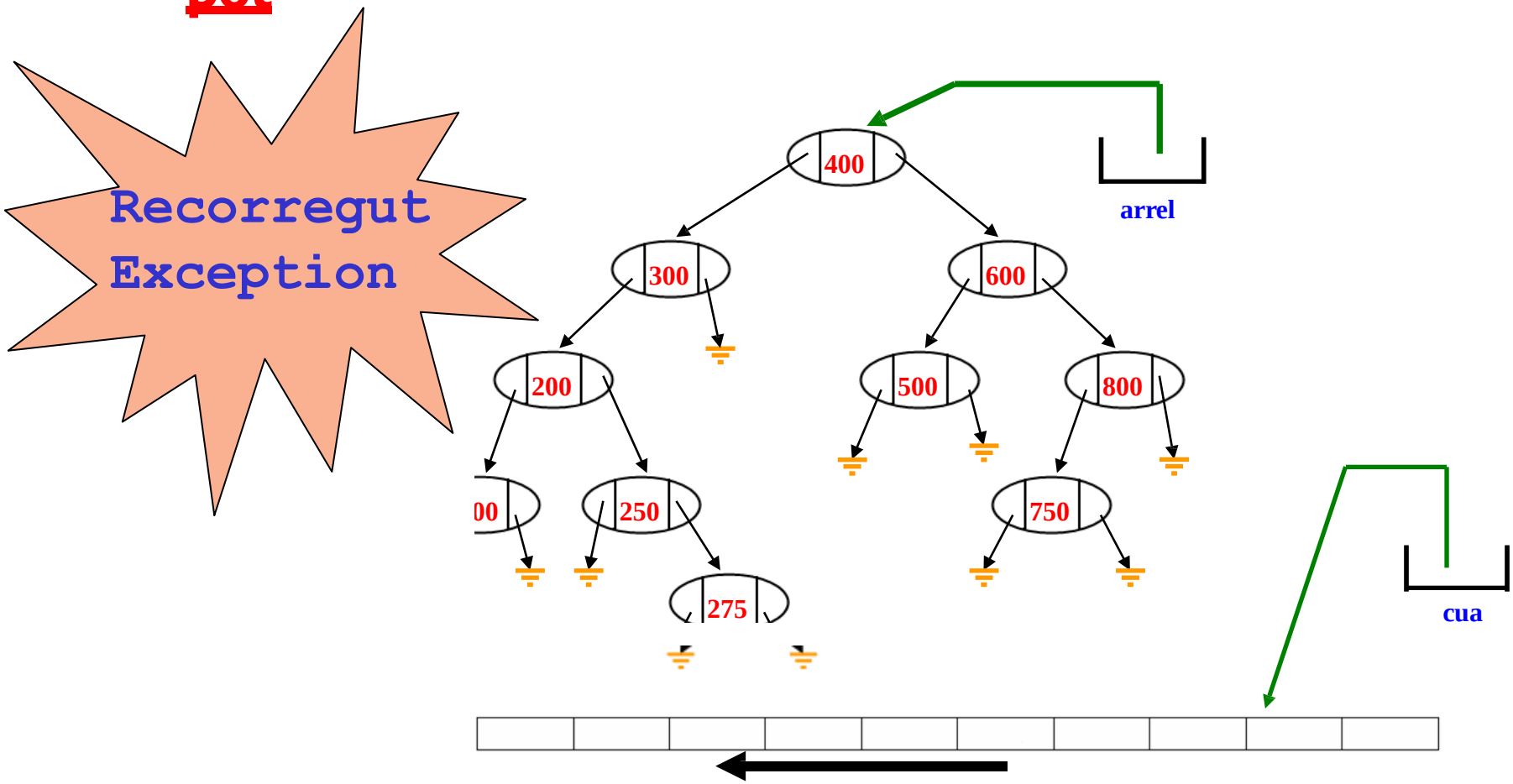
Desencua el primer element de la cua i el retorna



Els mètodes de recorregut:

segRecorregut

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



Els mètodes de recorregut: llençament d'excepcions

El mètode segRecorregut 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:

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

Pràctica 4. Implementació

Part 2

Més mètodes
Tècnica DIV

1.- Redefinició del mètode clone(). Aquest mètode arriba via herència, la implementació per defecte és realitzar una copia de referències. El que es demana és que es faci un duplicat. Signatura:

```
public Object clone()
```

Adjuntat un document amb informació de com procedir per redefinir el mètode clone(). Document que s'ha treballat a classe de teoria i que s'indiquen les pautes que s'han de seguir per redefinir-lo. No es permet invocar a cap mètode de la classe en la implementació. **Prohibit** usar l'operador **new**

2.- Calcula i retorna la cardinalitat, el nombre de nodes que té l'arbre.

```
public int cardinalitat()
```

Pràctica 4. Usuaris

Part 3

- Per provar el correcte funcionament de l'arbre es crearà una baralla de cartes i es visualitzarà ordenada usant un ACB recorrible
- **IMPLEMENTEU** la classe Carta i la classe Baralla

La classe Carta

Els objectes d'aquesta classe han de tenir dos atributs privats, un per emmagatzemar la numeració i altre pel pal. Referent als mètodes, a més del constructor afegiu els que considereu més adients. **Imprescindible** que hi hagi la redefinició del mètode `toString`.

La classe Carta **ha d'implementar la interfície Comparable**, les cartes s'han de poder comparar. La relació d'ordre vindrà donada, primer de tot **tenint en compte el pal i a igualtat de pal la relació vindrà donada per la numeració**. Considereu que el pal de COPA < ESPASA < OROS < BASTONS.

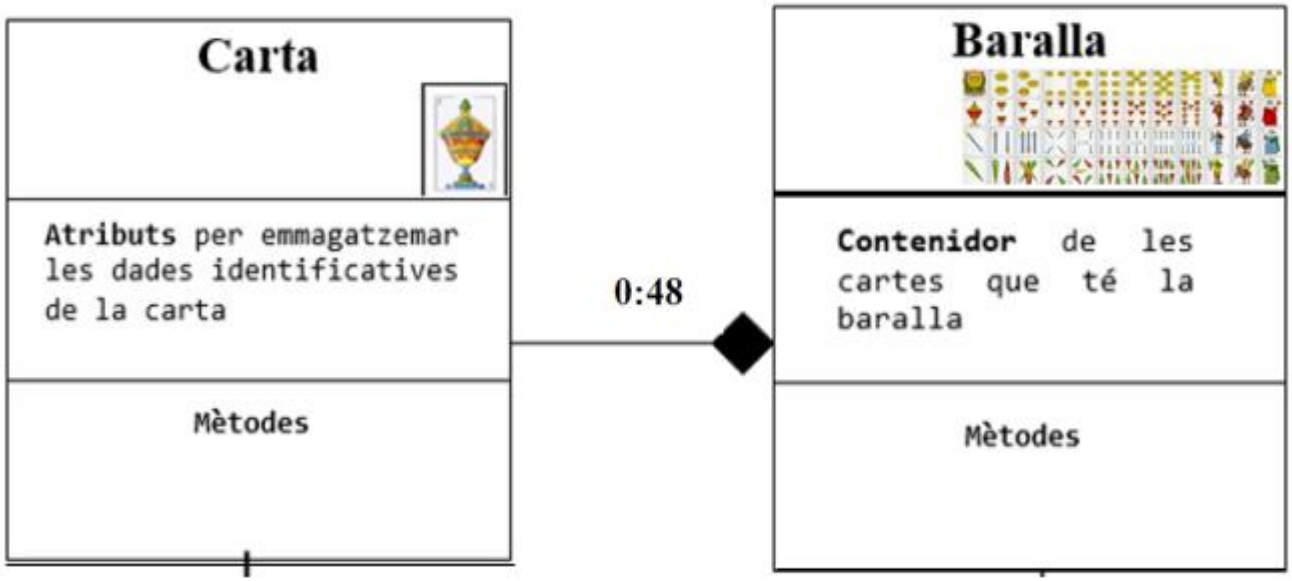
La classe Baralla és un contenidor d'objectes de tipus Carta

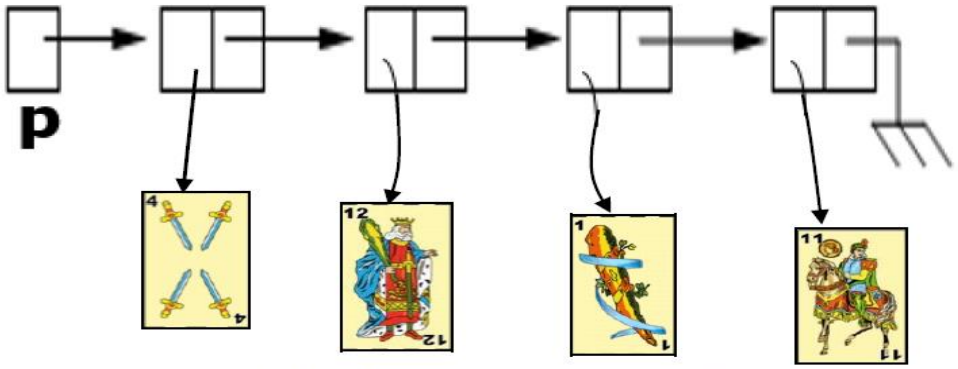
Heu de determinar vosaltres quins són els atributs i mètodes que calen per representar un objecte Baralla. **Imprescindible** que les cartes que formen la baralla estiguin emmagatzemades en una seqüència enllaçada lineal sense node capçalera. Un node per a cadascuna de les cartes.

El constructor **obligatòriament** ha de crear una baralla sencera desordenada.

Pràctica 4. Usuaris

Part 3





En la construcció la seqüència ha de tenir 48 nodes corresponents a les 48 cartes que té la baralla

La classe node serà privada dins de la classe Baralla.

Pràctica 4. Usuaris

Part 3

Escriu un programa main que:

- 1.- Creí una baralla de cartes barrejades i a continuació les carregi en un Acb invocant al mètode `inserir` (a partir d'aquest moment la baralla ja no s'usarà més).
- 2.- Visualitza un llistat ordenat de totes les cartes que conté l'arbre, en sentit ascendent.
- 3.- Fes una copia de l'arbre creat invocant al mètode **`clone`**, per usar més endavant.
- 4- Repetidament mostra un menú on l'usuari pugui seleccionar cartes que vol eliminar de l'arbre, d'una en una fins que ja no quedin cartes o bé no vulgui eliminar-ne més. Després de cada eliminació s'ha de mostrar novament el llistat ordenat ascendent de cartes que té l'arbre. Per controlar si a l'arbre li queden o no cartes **imprescindible invocar al mètode que calcula la cardinalitat** de l'arbre.

```
Menú  Opcions
============
1.- Eliminar carta
2.- Acabar
Tria que vols fer (1 o  2)
```

- 5.- Un cop l'usuari ja no vol o ja no queden cartes, al prémer l'opció Acabar del menú es visualitza l'arbre clonat, visualitzant a pantalla el seu contingut ordenat descendentment. I a continuació cal eliminar cartes del clonat, tal com s'indica en el punt següent.
- 6.- Elimina de l'arbre clonat cartes aleatòries, tantes com el número aleatori que surti aleatòriament (useu Random). Mostreu el seu contingut després de cada eliminació. Compte, si s'intenta eliminar una carta que no està a l'arbre, no és una eliminació!
- 7.- Finalment indica quin dels dos arbres té més cartes, el clonat o l'arbre inicial del que s'han eliminat tantes cartes com ha volgut l'usuari. **Imprescindible invocar al mètode que calcula la cardinalitat per fer aquest tractament.**