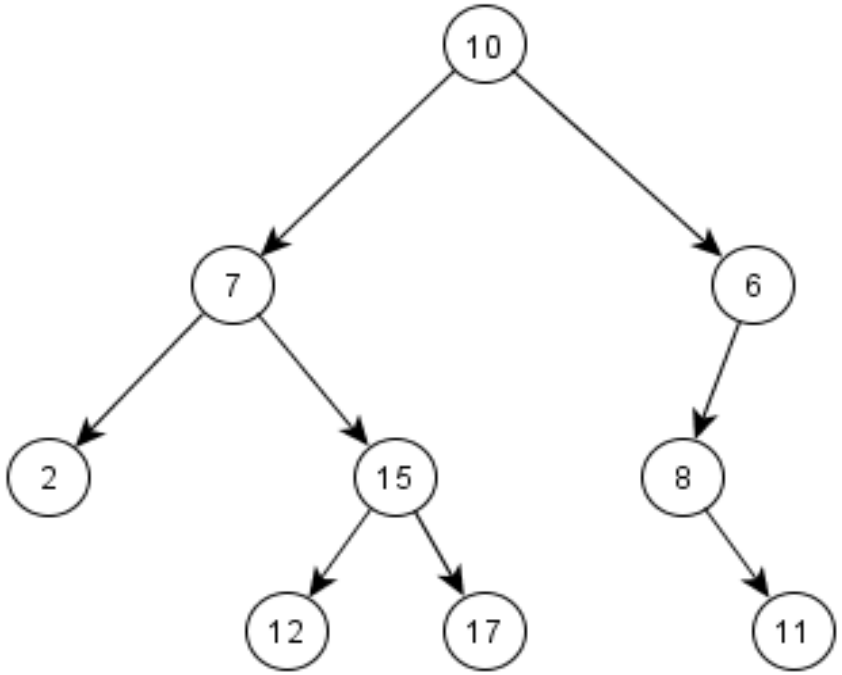
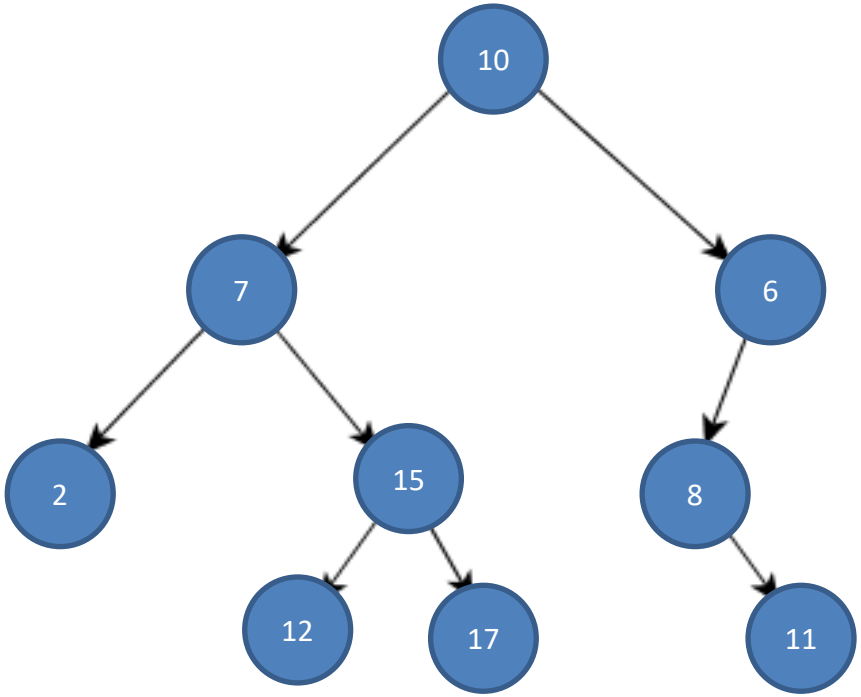
Arbre binaire

Structure de données qui peut se représenter de la façon suivante :



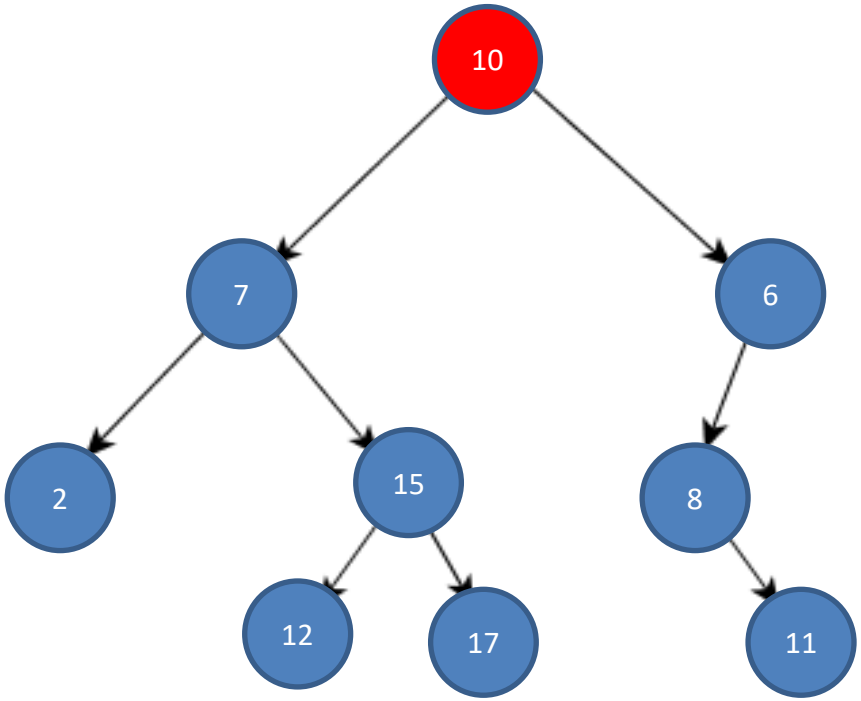
Arbres binaires

Cet arbre contient **9** nœuds :



Arbre binaire

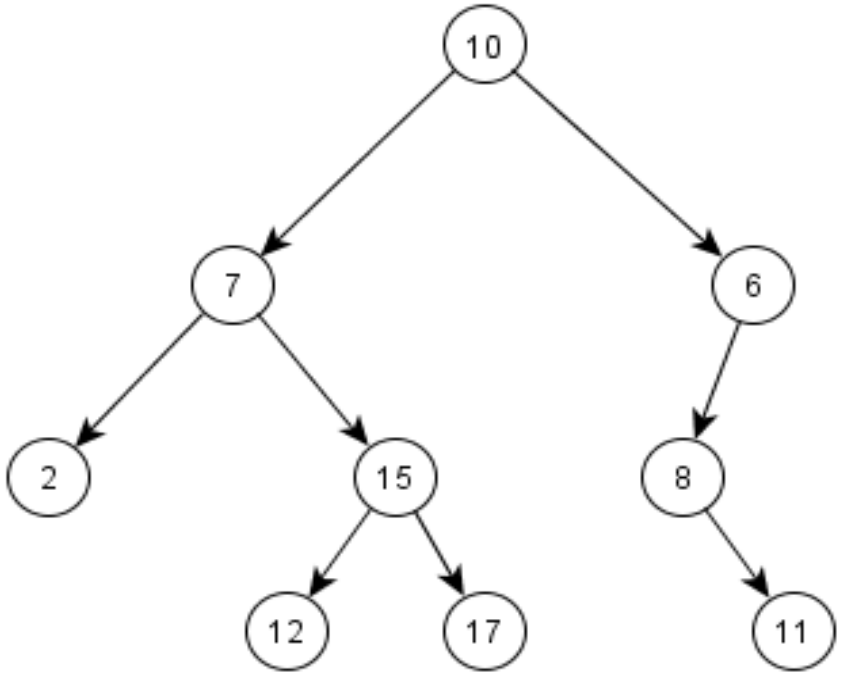
La racine est un nœud



Arbres binaires

Chaque nœud a maximum 2 enfants :

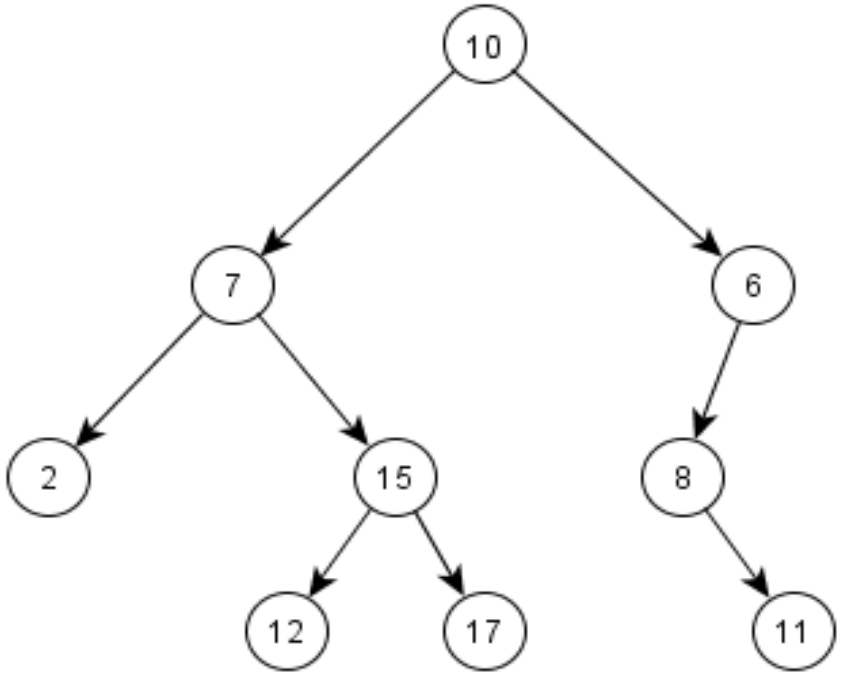
fils gauche et **fils droit**



Arbres binaires

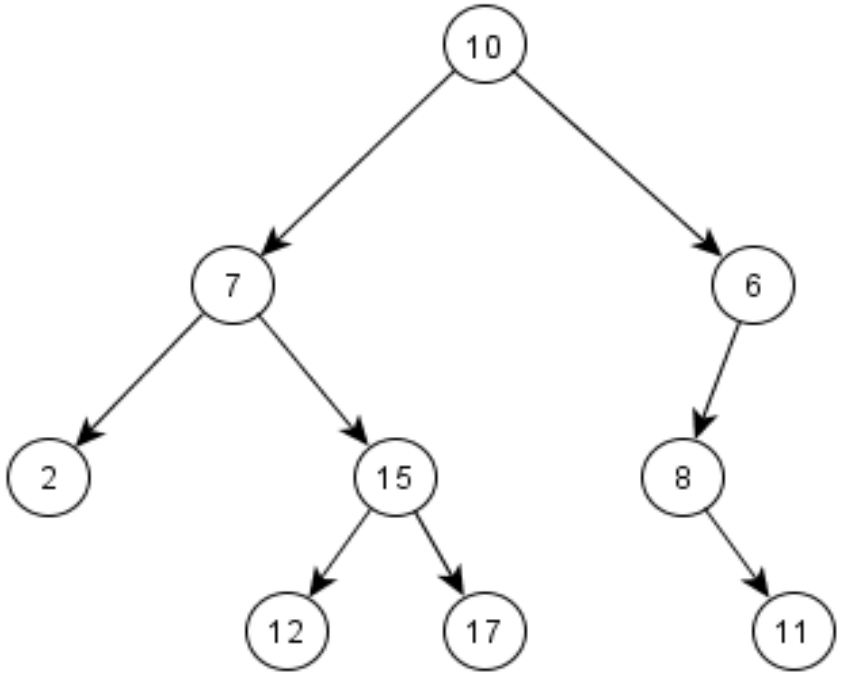
Ces fils gauche et fils droit sont des arbres

→ **sous-arbres**



Arbre binaire

Les feuilles sont des nœuds qui n'ont pas de fils



Implémentation

Un **nœud** contient :

- un élément
- un nœud qui correspond au sous-arbre de gauche
- un nœud qui correspond au sous-arbre de droite

Classe Noeud

```
class Noeud {
    E element;
    Noeud gauche;
    Noeud droit;

    public Noeud (E element);
    public Noeud (Noeud g, E element, Noeud d);
}
```

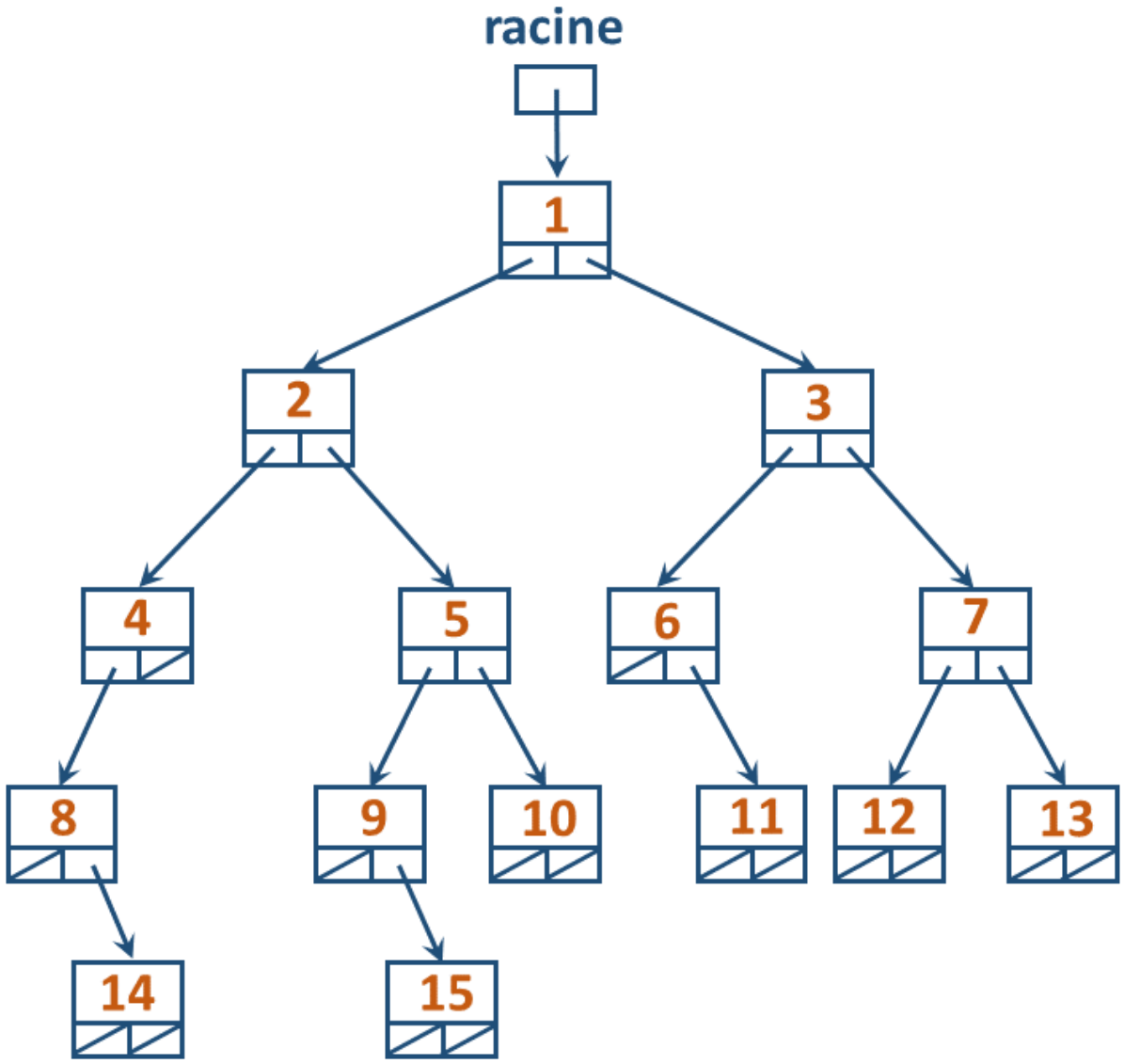
Implémentation

Un arbre binaire est constitué d'un nœud **racine**

L'arbre vide a sa racine à null :

racine

Implémentation



Implémentation

L'implémentation propose 3 constructeurs :

```
ArbreBinaire ()

ArbreBinaire (E element)

ArbreBinaire (ArbreBinaire filsGauche,
              E element,
              ArbreBinaire filsDroit)
```

Programmation

arbre = SD récursive

$\Rightarrow$ la plupart des méthodes sont elles-mêmes **récursives**

Programmation

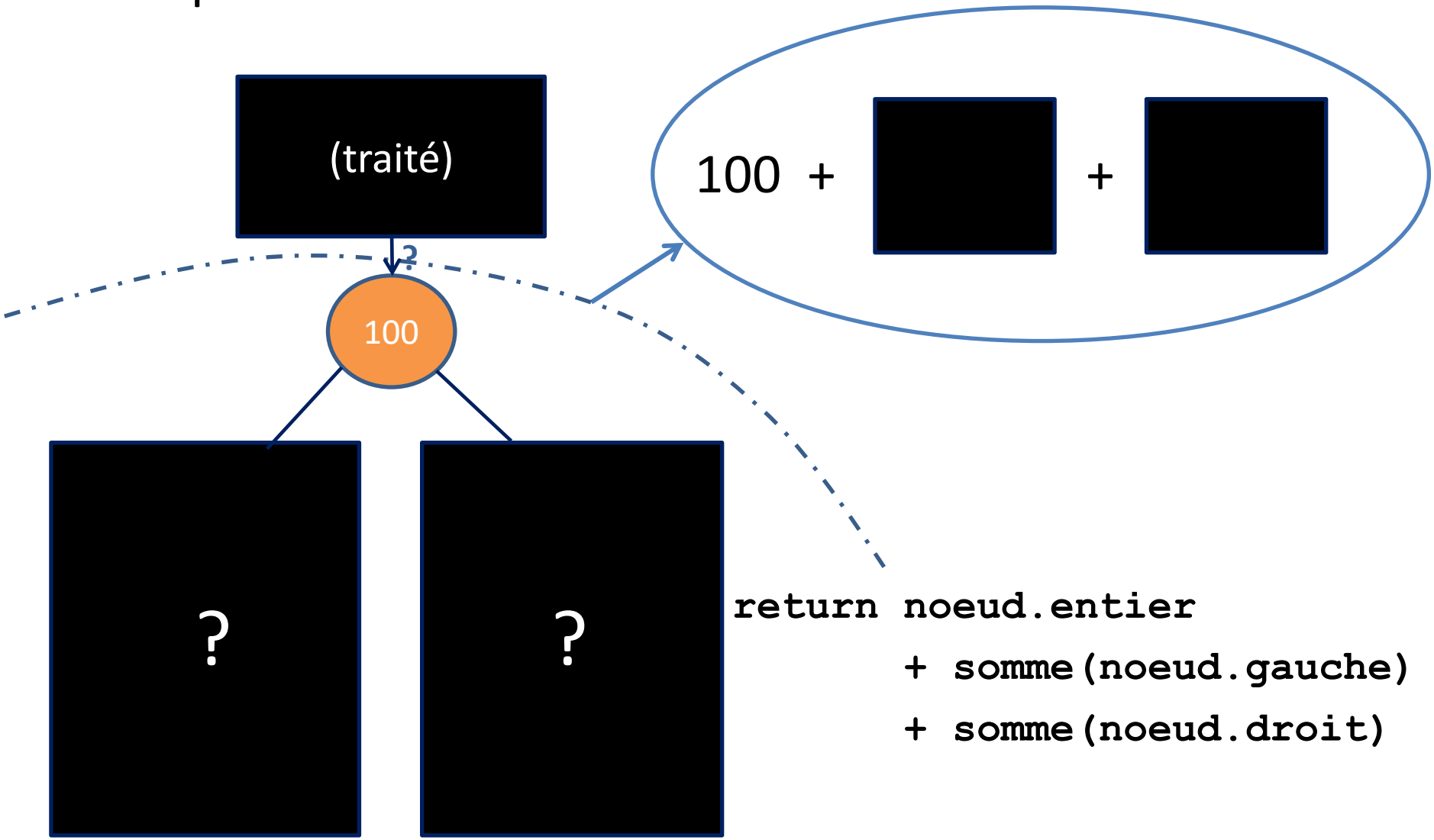
Exemple :

```
public int somme () {
     return somme(racine);
}

private int somme (Noeud n) {
     if (n == null) return 0;
     return n.entier
            + somme(n.gauche)
            + somme(n.droit);
}
```

Programmation

Exemple :




```
return noeud.entier
       + somme(noeud.gauche)
       + somme(noeud.droit)
```

Programmation

Exemple : [100 + [20 + [5] + [30]] + [200]]