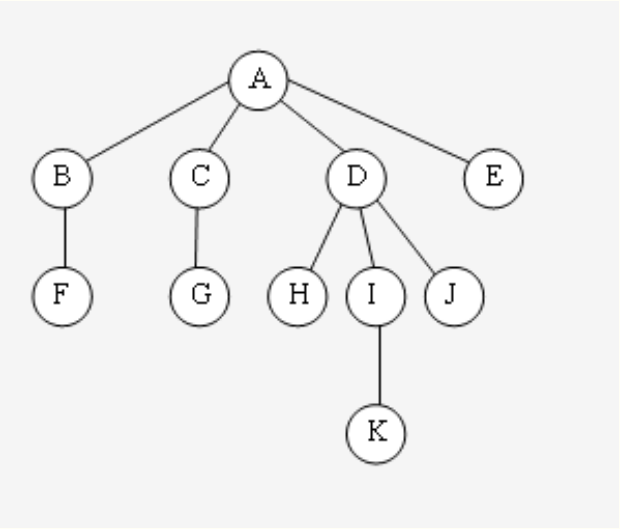
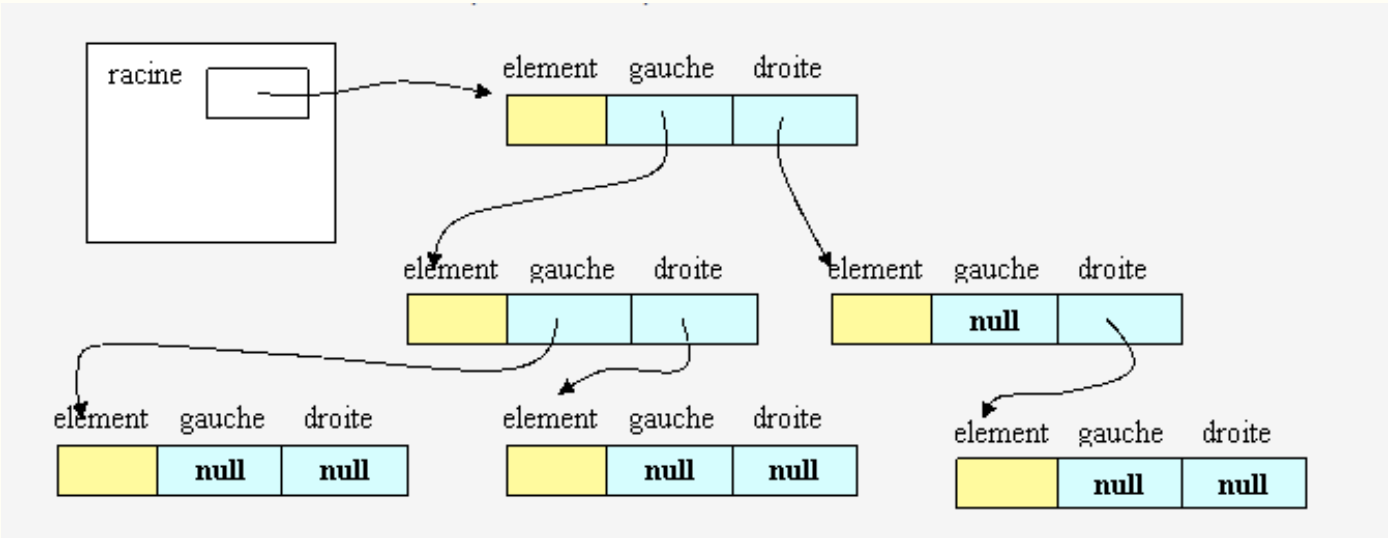
Structures de données arborescentes

- arbres généraux,
- arbres binaires,
- arbres binaires de recherche,
- …

Les arbres

- Arbres généraux et arbres binaires ordonnés





- Structures très adaptées dans certains cas
  (systèmes de fichiers, généalogie, expressions arithmétiques, …)
- différents types de parcours (profondeur, largeur) est ordre (préfixe, infixe ou post fixe)
- recherche, insertion, suppression, équilibrage, …

Les Arbres (suites)

Définition **[arbre]** *ensemble fini d'éléments appelés nœuds, liés par une* « **relation de parenté** » *qui a les propriétés suivantes :*

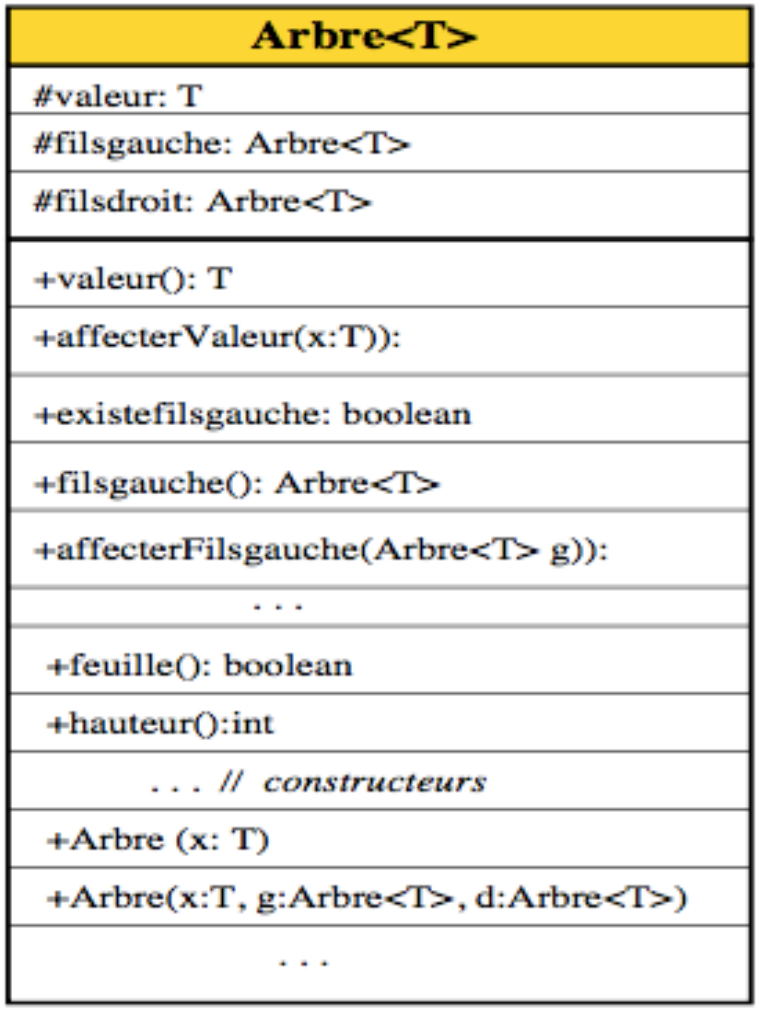
◁ *il existe un unique élément n'ayant pas de père, appelé racine de l'arbre ;*

◁ *tout élément, à part la racine a un et un seul père ;*

◁ *tout élément est un descendant de la racine.*

Structures bien adaptées pour certains types de problèmes (e.g. évaluation d'expressions arithmétiques, logiciel de jeu d'échec, représentation de la relation d'héritage dans les objets, répertoires informatiques, . . .)

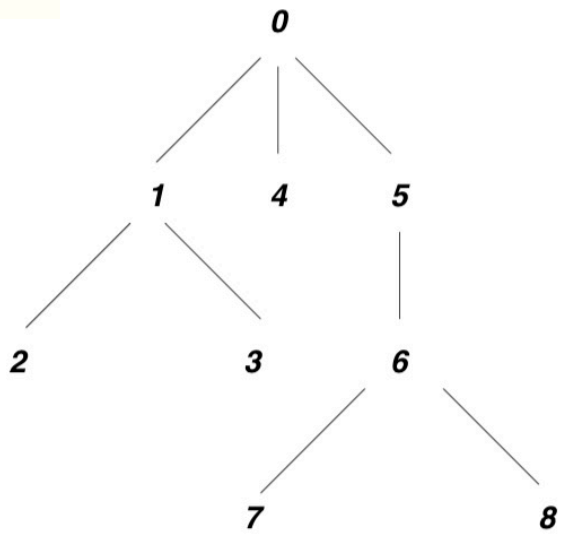
Le schéma d'un arbre binaire :

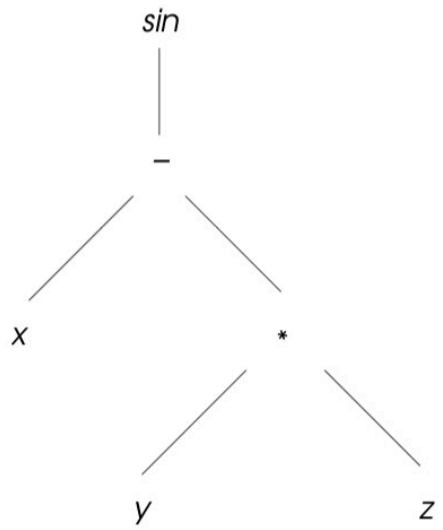


Vocabulaire sur les arbres

- Feuille : nœud terminal (pas de fils),
- Profondeur d'un nœud (nombre d'ascendants),
- Hauteur d'un arbre (profondeur maximale),
- Taille d'un arbre (nombre de nœuds),
- Sous-arbre, (ensemble des descendants d'un nœud),
- Arbre n-aire, (0 ou n branches issues de chaque nœud),
- Arbre équilibré, (la profondeur de toute feuille diffère au maximum de 1 de la profondeur des autres feuilles),
- Arbre étiqueté.

Exemples d'arbres





Un arbre ("ordonné") d'entiers et un arbre de représentation de l'expression arithmétique $\sin(x - yz)$

Les différents parcours d'arbres

Les types de parcours (1/2)

- *"En profondeur d'abord"* : de la racine, descendre au plus bas en suivant la branche la plus à gauche de chaque nœud, puis remonter pour explorer les autres branches ;
- *"En largeur d'abord"* : de la racine, explorer tous les nœuds d'un niveau avant de passer au niveau suivant.

Figure __ Parcours en profondeur et en largeur.

Une implémentation des "Arbres binaires" en Java

```
class Arbre {
  protected <T> valeur;
  protected Arbre filsGauche, filsDroit;
  public <T> valeur() { return valeur; }

  public boolean existeFilsGauche() { return filsGauche != null; }
  public boolean existeFilsDroit() { return filsDroit != null; }

  public Arbre filsGauche() { return filsGauche; }
  public Arbre filsDroit() { return filsDroit; }

  public void affecterValeur(<T> c) { valeur = c; }

  public void affecterFilsGauche(Arbre g) { filsGauche = g; }
  public void affecterFilsDroit(Arbre d) { filsDroit = d;}

  public boolean feuille() {return (filsDroit==null &&
  filsGauche==null); }
}
```

Exercice à suivre: la hauteur de l'arbre et ses constructeurs

Quelques méthodes sur les Arbres binaires

```
public int hauteur() {
    int g = existeFilsGauche() ? filsGauche.hauteur() : 0;
  . . .
    return  . . .  ;
}
// Constructeurs
public Arbre(T val) {
    valeur = val;
    filsGauche = filsDroit = null;
}
public Arbre(T val, Arbre<T> g, Arbre<T> d) {
    valeur = val;
    filsGauche = g; filsDroit = d;
}
```

Rmq. nouvelle instruction conditionnelle

La méthode "hauteur" sur les arbres binaires

```
public int hauteur() {
    int g = existeFilsGauche() ? filsGauche.hauteur() : 0;
    int d = existeFilsDroit() ? filsDroit.hauteur() : 0;
    return 1 + Math.max(g, d);
}
```

```
// Constructeurs
public Arbre(T val) {
    valeur = val;
    filsGauche = filsDroit = null;
}
public Arbre(T val, Arbre<T> g, Arbre<T> d) {
    valeur = val;
    filsGauche = g; filsDroit = d;
}
```

Affichage préfixe d'un Arbre binaire

```
// Affichage dans l'ordre préfixe
public void afficherPrefixe() {
    ?? ;
    ?? ;
    ?? ;
}
```

Affichage préfixe d'un Arbre binaire

```
// Affichage
public void afficherPrefixe() {
    System.out.print(valeur+"\t");
    if (existeFilsGauche()) filsGauche.afficherPrefixe();
    if (existeFilsDroit())   filsDroit.afficherPrefixe();
}
```

Autres méthodes pour l'affichage des Arbres binaires

```
// Affichage
public void afficherPrefixe() {
    System.out.print(valeur+"\t");
    if (existeFilsGauche()) filsGauche.afficherPrefixe();
    if (existeFilsDroit())   filsDroit.afficherPrefixe();
}
public void afficherInfixe() {
    if (existeFilsGauche()) filsGauche.afficherInfixe();
    System.out.print(valeur+"\t");
    if (existeFilsDroit())filsDroit.afficherInfixe();
}
public void afficherPostfixe() {
    if (existeFilsGauche()) filsGauche.afficherPostfixe();
    if (existeFilsDroit())filsDroit.afficherPostfixe();
    System.out.print(valeur+"\t");
}
```

Les Arbres binaires de recherche

Définition 5 (Arbre binaire de recherche [ABR]). *Un* ABR *est un arbre dont les étiquettes des nœuds appartiennent à un ensemble totalement ordonné et les conditions équivalentes suivantes sont vérifiées :*

◁ *La liste des étiquettes en ordre infixe est croissante.*

◁ *Pour tout nœud* `x` *d'étiquette* `e`*, les étiquettes des nœuds de la branche gauche de* `x` *sont inférieures ou égales à* `e` *et les étiquettes des nœuds de la branche droite de* `x` *sont supérieures ou égales à* `e`*.*

Motivations :

◁ La recherche d'un élément dans un arbre ordonné est plus rapide.

Exemple d'arbres binaires de recherche

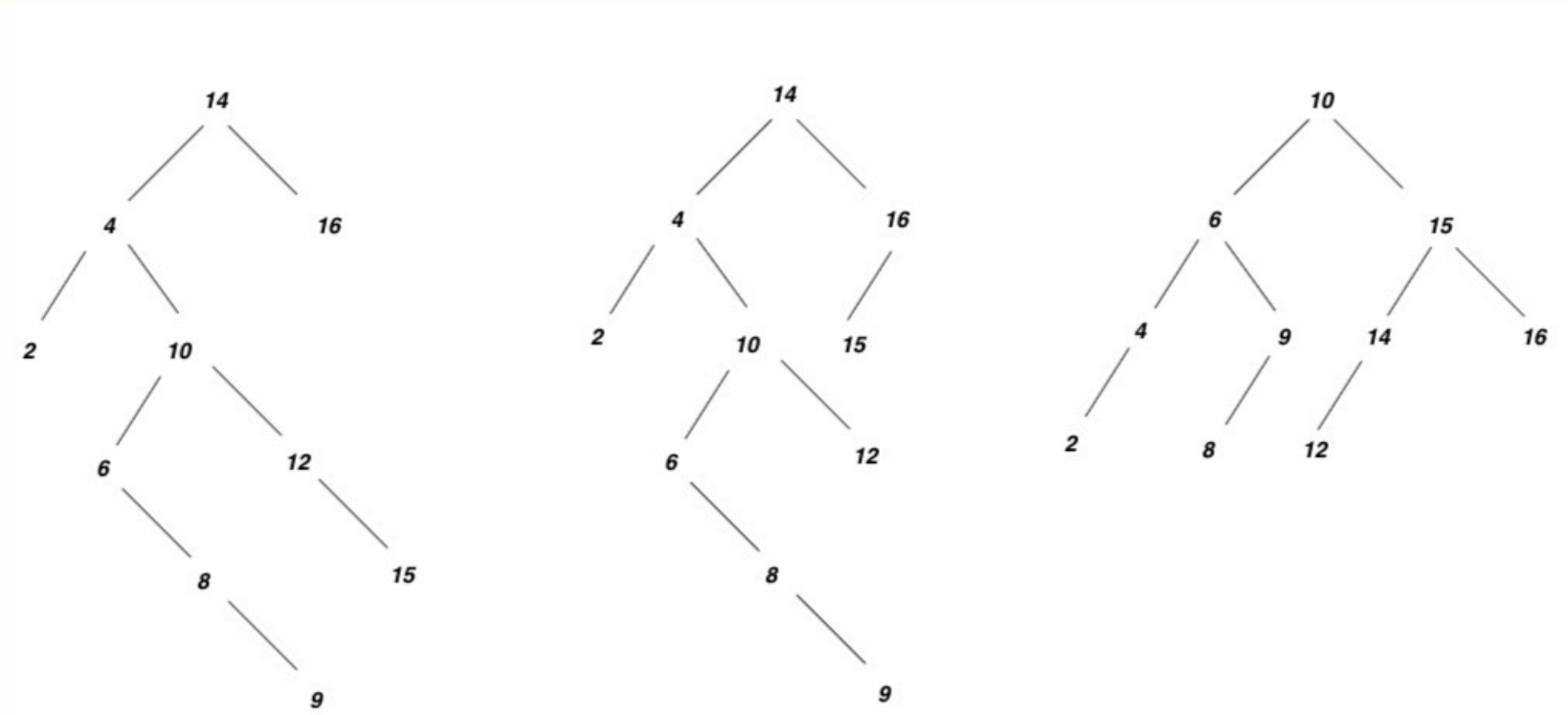


Figure *Arbres : un non* ABR, *et deux* ABRs.

Vérifier si un Arbre binaire est un ABR

```
public boolean superieur(char x) {
// vrai si x est supérieur à tous les éléments de l'arbre

  if (feuille()) return (x>=valeur);

  else return (((this.existeFilsGauche())?
    (this.filsGauche).superieur(x):true) &
    ((this.existeFilsDroit())?
          (this.filsDroit).superieur(x):true));

}

public boolean inferieur(char x) {//similaire a superieur ... }


public boolean binrech() {

  if (feuille()) return true;

  else return( (existeFilsGauche()?
   (filsGauche.superieur(valeur) && filsGauche.binrech()):true)
             & (existeFilsDroit()?
   (filsDroit.inferieur(valeur) && filsDroit.binrech()):true));

}
```

Conclusion

- Les grands apports des objets
(programmation plus facile, sûreté, encapsulation, …)
- Les grands apports des classes
(héritage, ré-utilisation, …)
- Focalisation sur les questions essentielles
(structurations des données, …)
- Etude de quelques structures de données
(Listes, Arbres, Tables de hachage …)