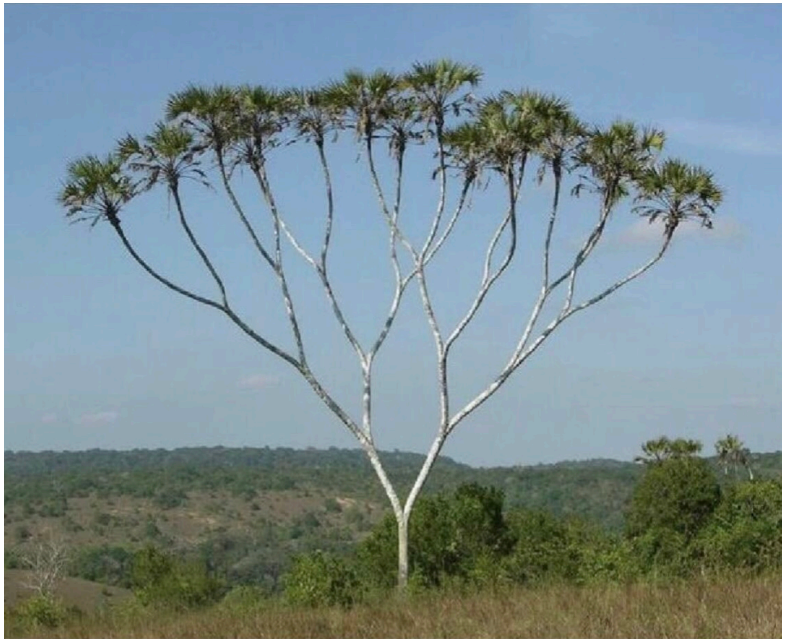
Arbre binaire

Définitions

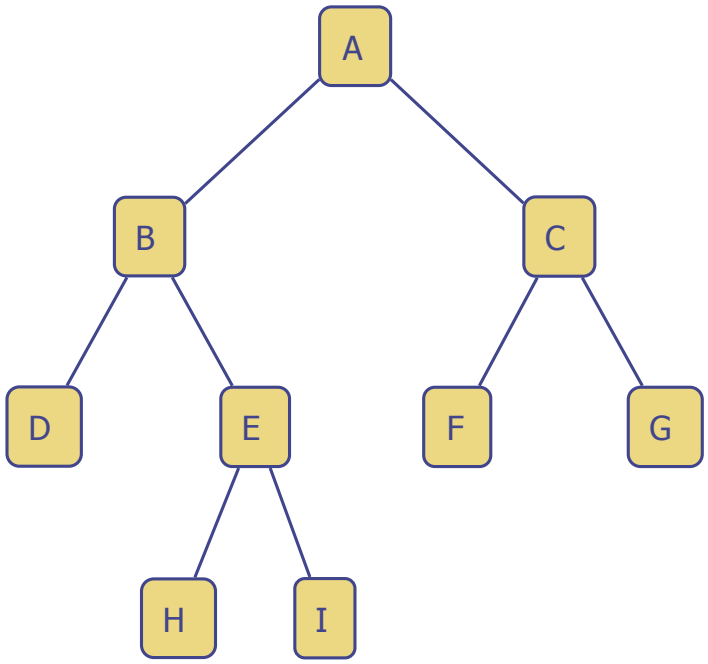
ADT BinaryTree

Propriétés de l'arbre binaire



Arbre binaire

- Un **arbre binaire** est un arbre avec les propriétés suivantes :
  - Chaque nœud interne possède au plus deux enfants (exactement deux pour un arbre binaire **plein**)
  - Les enfants d'un nœud sont une paire ordonnée : l'un est à gauche et l'autre à droite (**D** est à gauche et **E** à droite de **B**)
- Une définition récursive : un **arbre binaire** est soit :
  - un arbre vide
  - un arbre non vide dont la racine, r, possède une paire ordonnée d'enfants, respectivement les sous-arbres binaires gauche et droit de r
- Applications:
  - expressions arithmétiques
  - processus de décision

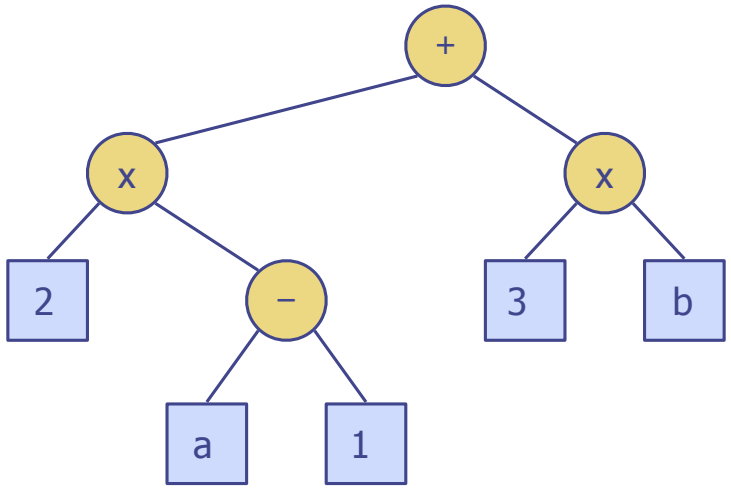


Arbre binaire pour une expression arithmétique

Les nœuds internes sont les opérateurs

Les noeuds externes sont les opérandes

Exemple : (2 x (a - 1) + (3 x b))

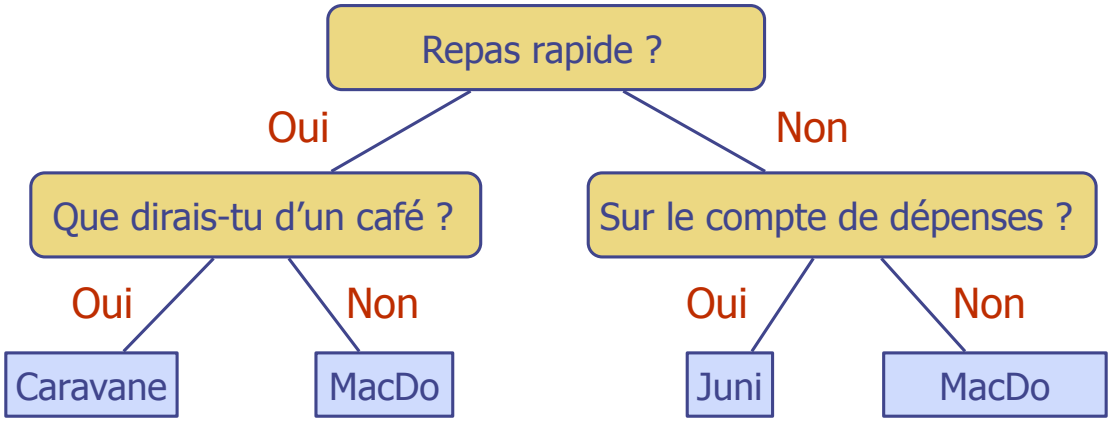


Arbre de décision

Les nœuds internes sont des questions à réponses oui/non

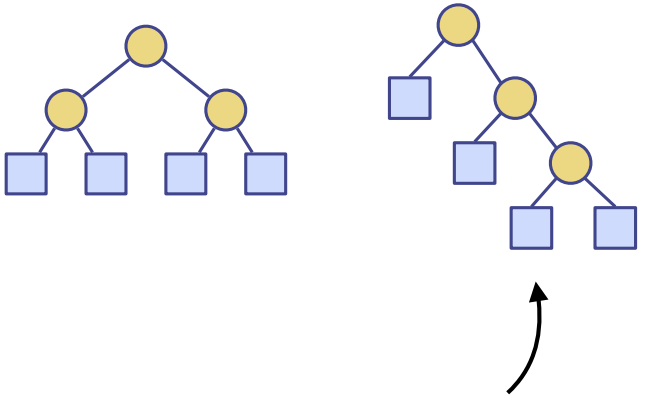
Les nœuds externes sont des décisions

Exemple : décider quoi manger

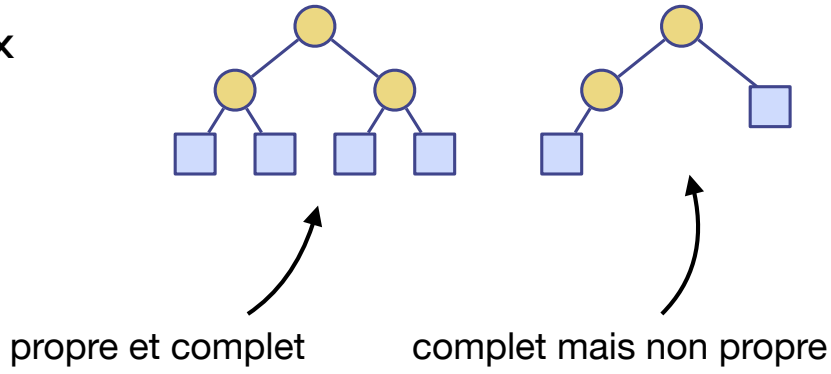


Arbres binaires propre/stricte et complet

Un **arbre binaire propre** ("full" en anglais) est un arbre binaire dont tous les noeuds internes possèdent 2 enfants. Un arbre binaire propre est aussi dit **stricte**.



Un **arbre binaire complet** ("complete" en anglais) est un arbre binaire dont tous les niveaux sauf possiblement le dernier sont complètement remplis et tous les noeuds sont le plus à gauche possible. *Un arbre binaire complet n'est pas nécessairement propre* !



ADT `BinaryTree`

- L'ADT **BinaryTree** étend l'ADT **Tree**, en plus de fournir les méthodes suivantes :
  - `left( p )` : retourne la positions de l'enfant à gauche de p, null si p n'a pas d'enfant à gauche
  - `right( p )` : retourne la position de l'enfant à droite de p, null si p n'a pas d'enfant à droite
  - `sibling( p )` : retourne la position de l'autre enfant issu du même parent

`BinaryTree.java`

Allons voir le code...

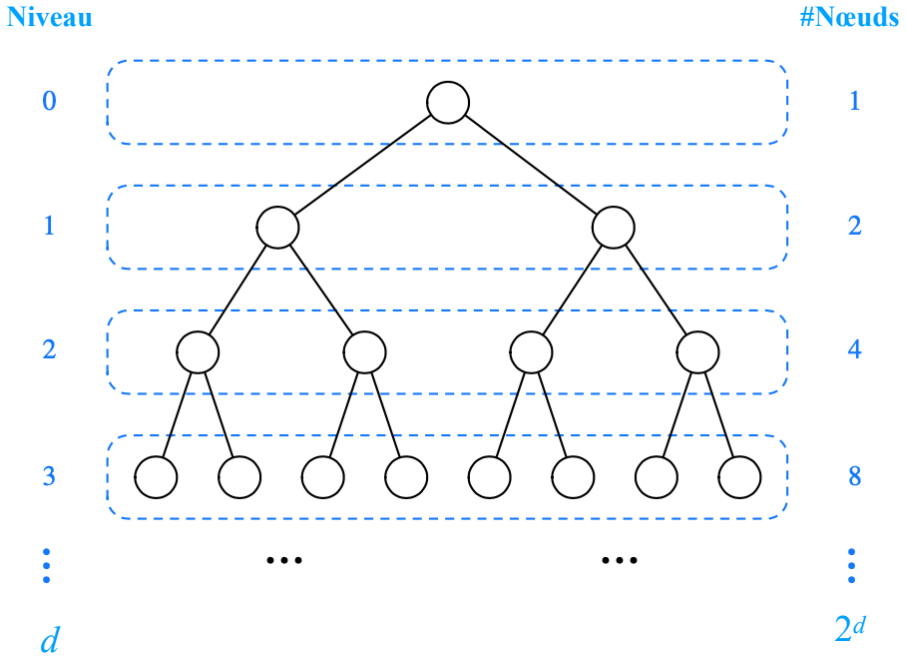
`AbstractBinaryTree.java`

Propriétés d'un arbre binaire

Les arbres binaires ont plusieurs propriétés intéressantes concernant les relations entre leurs hauteurs et le nombre de nœuds.

Le **niveau** d'un nœud fait référence à la distance de ce nœud par rapport au nœud racine, basée sur le nombre d'arêtes (ou d'étapes) qu'il faut parcourir pour l'atteindre.

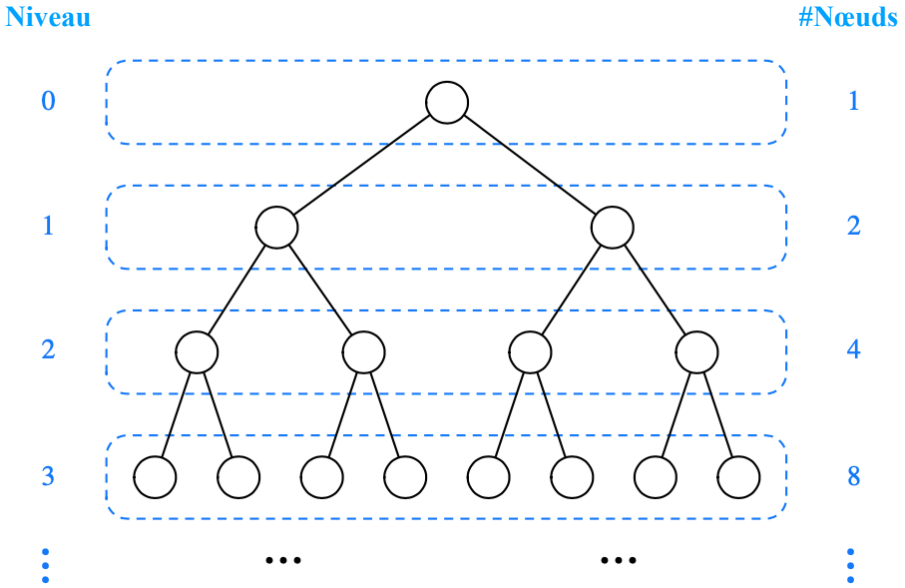
Dans un arbre binaire, le niveau 0 a au plus un nœud (la racine), le niveau 1 a au plus 2 nœuds (les enfants de la racine), le niveau 2 a au plus 4 nœuds, et ainsi de suite. En général, le niveau d a au plus 2^d nœuds.



Nombre maximum de nœuds dans les niveaux d'un arbre binaire

Propriétés d'un arbre binaire

Nous voyons que le nombre maximum de nœuds sur les niveaux d'un arbre binaire augmente de manière exponentielle au fur et à mesure que nous descendons l'arbre. De cette simple observation, on peut déduire les propriétés suivantes reliant la hauteur d'un arbre binaire T à son nombre de nœuds.



h: profondeur maximale d'un des nœuds
Profondeur d'un nœud: nombre d'ancêtres (niveau)

Soit T un arbre binaire non vide et

- n nombre de nœuds
- E nombre de nœuds externes
- I nombre de nœuds internes
- h hauteur de T

T possède alors les propriétés suivantes :

- $h + 1 \leq n \leq 2^{h+1} - 1$
- $1 \leq E \leq 2^h$
- $h \leq I \leq 2^h - 1$
- $log(n + 1) - 1 \leq h \leq n - 1$

Si T est propre (tous les noeuds internes possèdent 2 enfants), alors T possède aussi les propriétés :

- $2h + 1 \leq n \leq 2^{h+1} - 1$
- $h + 1 \leq E \leq 2^h$
- $h \leq I \leq 2^h - 1$
- $log(n + 1) - 1 \leq h \leq (n - 1) / 2$

👉 hauteur h d'un arbre binaire est borné par log(n) et n.

👉 on peut stocker jusqu'à n nœuds dans un arbre binaire de hauteur log(n) !

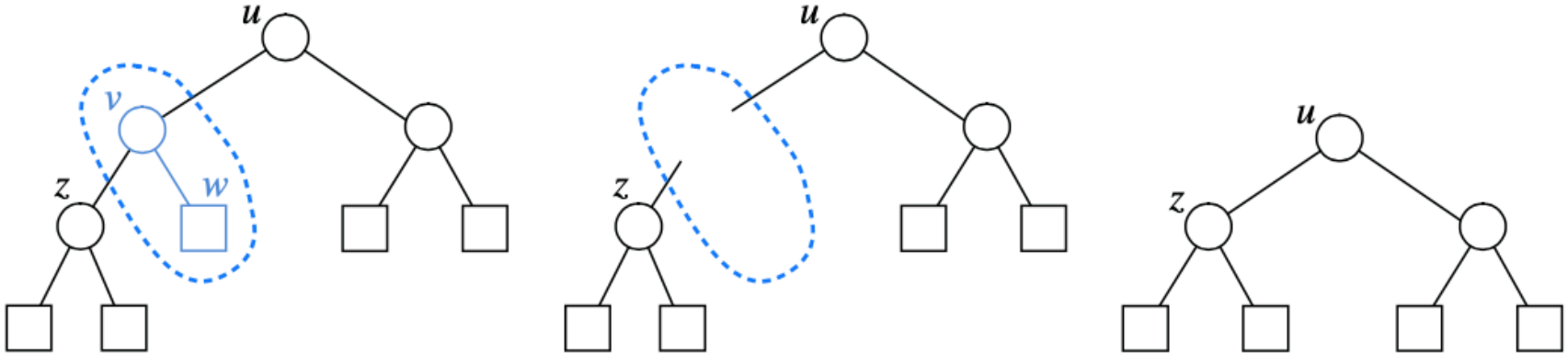
La relation entre les nombres de nœuds internes et externes dans un arbre binaire propre

Dans un arbre binaire propre non vide T avec E nœuds externes et I nœuds internes, on a $E = I + 1$

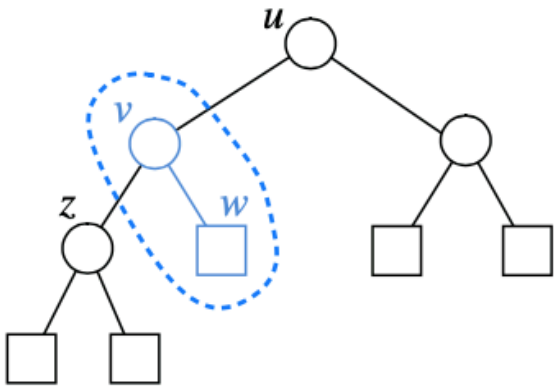
Imaginez enlever tous les nœuds de T et les diviser en deux piles, une pour les nœuds internes et une pour les nœuds externes. Les piles sont initialement vides. À la fin, nous trouverons que la pile de nœuds externes a un nœud de plus que la pile de nœuds internes.

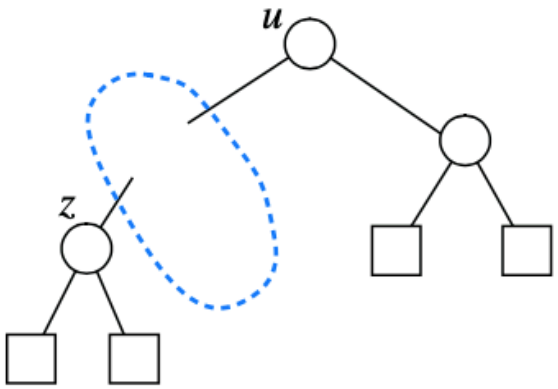
Considèrez deux cas :

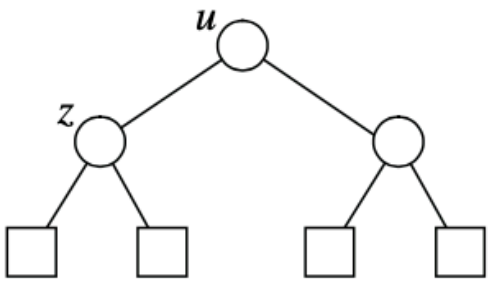
1. Si T n'a qu'un seul nœud v, nous enlevons v et le plaçons sur la pile de nœuds externes. Ainsi, la pile de nœuds externes a un nœud et la pile de nœuds internes est vide ;
2. Sinon (T a plus d'un nœud), nous enlevons de T un nœud externe (arbitraire) w et son parent v, qui est un nœud interne. Nous plaçons w sur la pile des nœuds externes et v sur la pile des nœuds internes. Si v a un parent u, alors nous reconnectons u avec l'ancien frère z de w, comme dans la figure. Cette opération enlève un nœud interne et un nœud externe, et laisse l'arbre binaire propre. En répétant cette opération, nous nous retrouvons finalement avec un arbre composé d'un seul nœud. Notez que le même nombre de nœuds externes et internes ont été enlevés et placés sur leurs piles respectives par la séquence d'opérations menant à cet arbre final. Maintenant, nous enlevons le nœud de l'arbre final et nous le plaçons sur la pile de nœuds externes. Ainsi, la pile de nœuds externes a un nœud de plus que la pile de nœuds internes.

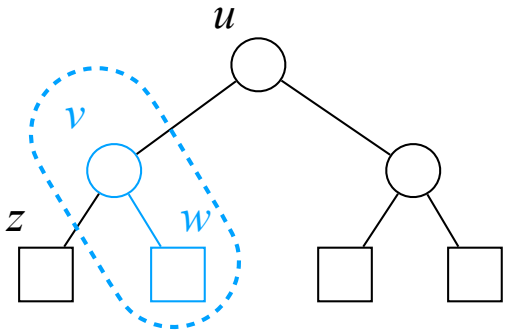


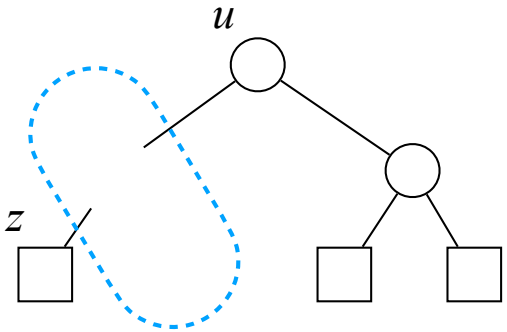
Convaincu.e ?

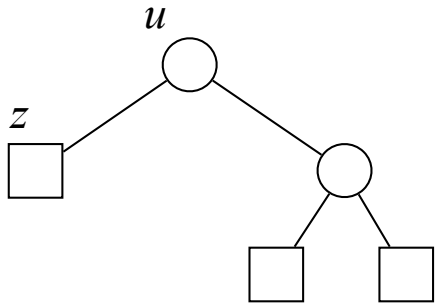


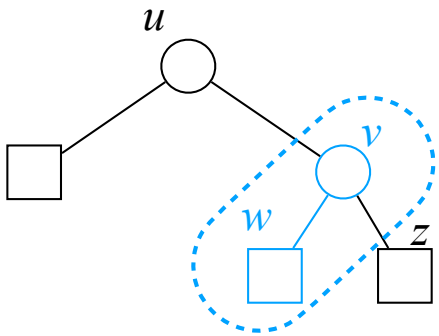


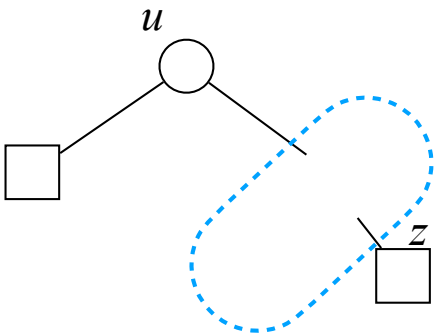


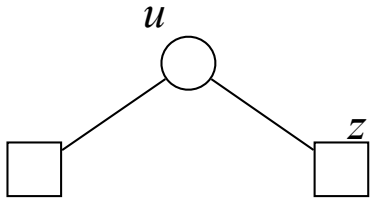












Conclusions du module

- Nous avons donné 2 définitions d'un arbre binaire
- Nous avons regardé 2 applications : expression arithmétique et arbre de décision
- Nous avons défini l'interface (ADT) pour un arbre binaire (BinaryTree) et une classe abstraite pour commencer son implémentation (AbstractBinaryTree)
- Nous avons regardé des propriétés d'un arbre binaire et d'un arbre binaire propre