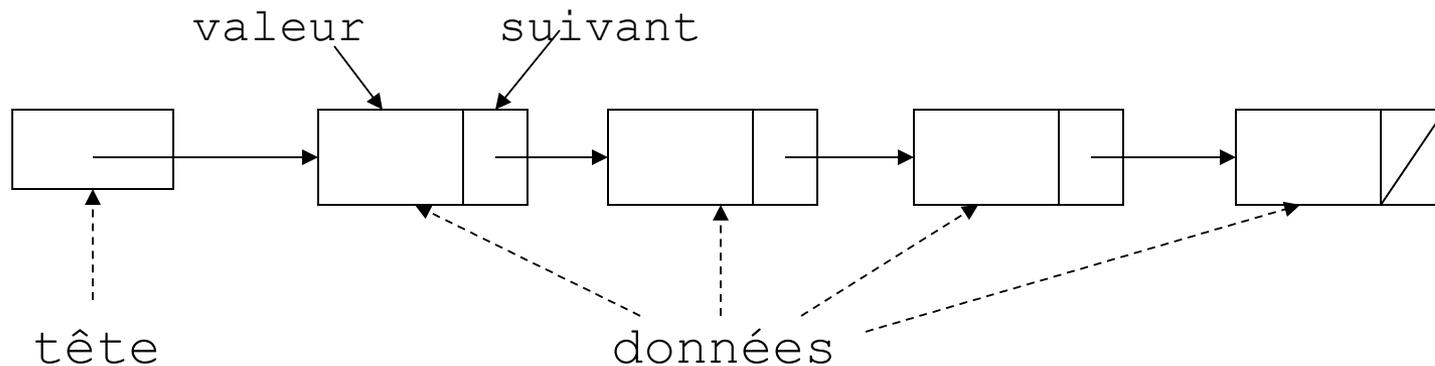


Chapitre 8 : types récur­sifs, les listes

Listes et tableaux
Opérations sur les listes
Exemples
Arbres binaires

Les listes

Une liste est une structure de données qui relie des objets de type identique (et quelconque)



Différences entre listes et tableaux

Accès aux éléments :

- dans un tableau, on accède à chaque élément par son indice. L'accès est immédiat
- dans une liste, on accède aux éléments séquentiellement à partir du premier. On doit donc parcourir tout ou partie de la liste avant de trouver l'élément voulu

Taille:

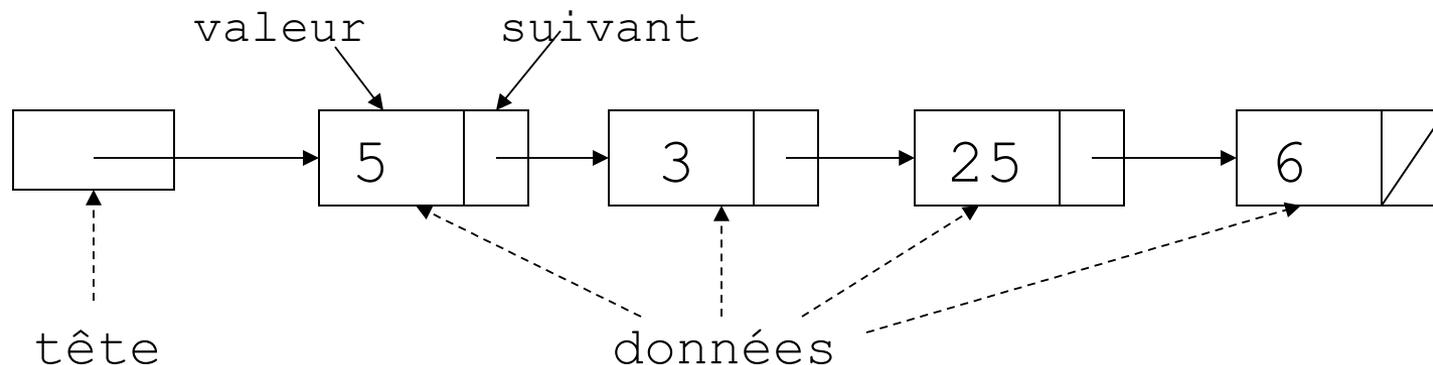
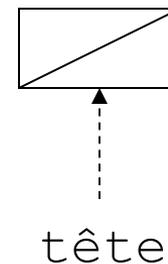
- un tableau a une taille fixe
- une liste a une taille flexible. On peut toujours rajouter un élément à une liste

Définition

Les listes sont définies récursivement.

Un liste l d'éléments de type T est soit :

- vide (notée $l = []$)
- constituée d'un premier élément de type T (noté e) et d'une liste d'éléments de type T (notée r) $\Rightarrow l = (e, r)$



Représentation java

```
// exemple : liste d'entiers
class ListeInt{
    int valeur;
    ListeInt suivant;

    ListeInt(int premier, ListeInt reste){
        valeur = premier;
        suivant = reste;
    }
}
```

Cette représentation impose la présence d'au moins un élément
Pour créer une liste vide en java, on utilise la référence nulle

```
ListeInt l = null;
```

Listes doublement chaînées

```
// exemple : liste d'entiers
```

```
class ListeInt{
```

```
    int valeur;
```

```
    ListeInt precedent;
```

```
    ListeInt suivant;
```

```
ListeInt(int premier, ListeInt avant,
```

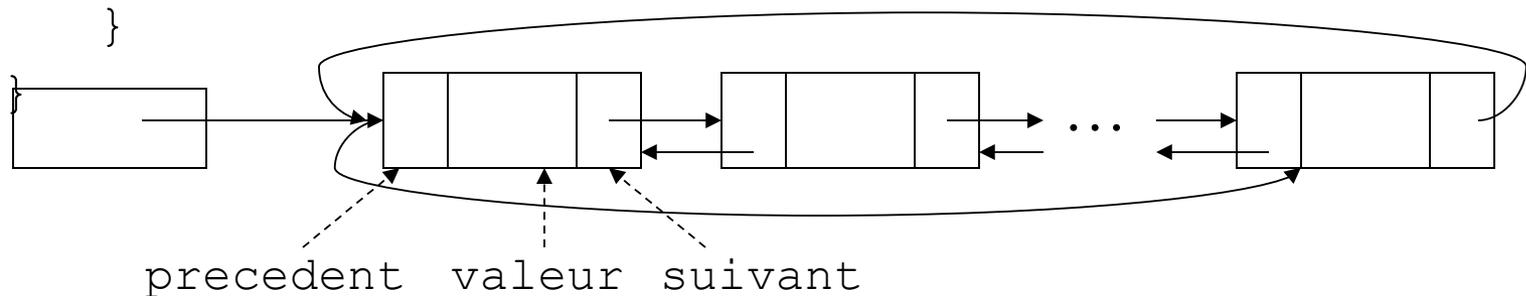
```
        ListeInt apres){
```

```
    valeur = premier;
```

```
    precedent = avant;
```

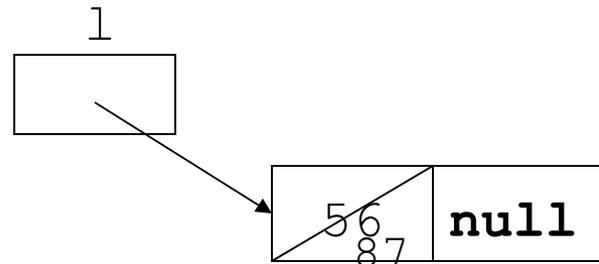
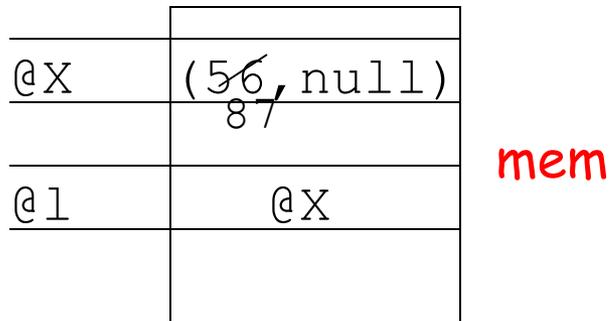
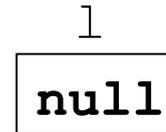
```
    suivant = apres;
```

```
}
```



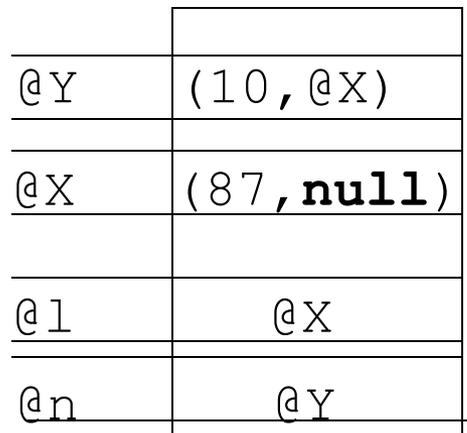
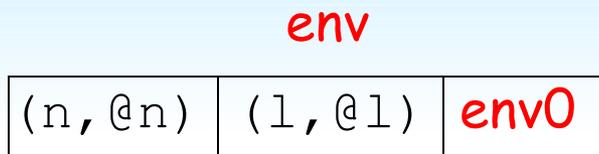
Manipulation de listes en Java (1/3)

```
ListeInt l = null;  
l = new ListeInt(56, null);  
l.valeur = 87;
```

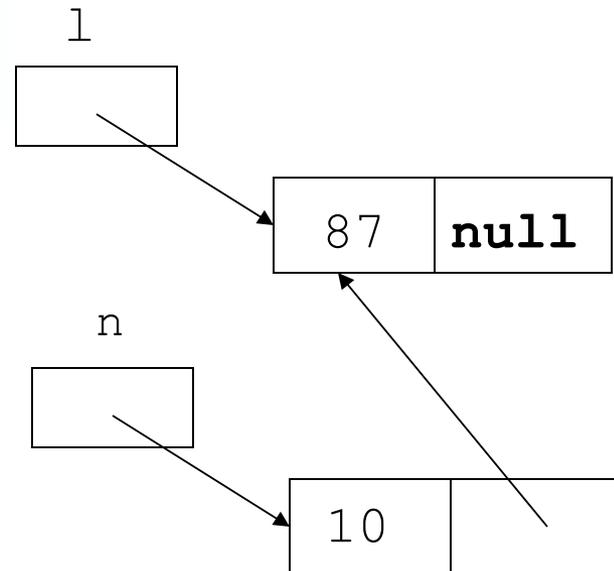


Manipulation de listes en Java (2/3)

```
ListeInt n = new ListeInt(10,1);
```



mem



Manipulation de listes en Java (3/3)

```
l = n;
```

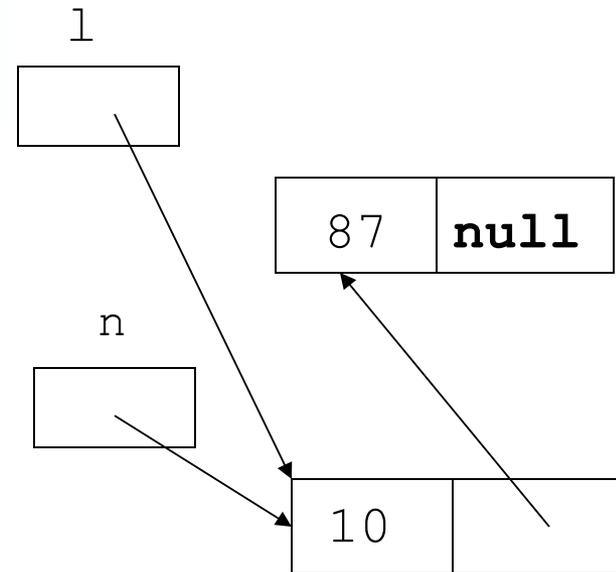
```
// copie des références et non des objets
```

env

(n, @n)	(l, @l)	env0
---------	---------	------

@Y	(10, @X)
@X	(87, null)
@l	@Y
@n	@Y

mem



Comparaisons

```
l == n
```

```
// vaut true si l et n réfèrent le même objet
```

```
l == null
```

```
// vaut true si la liste l est vide
```

La classe Liste d'entiers (1/3)

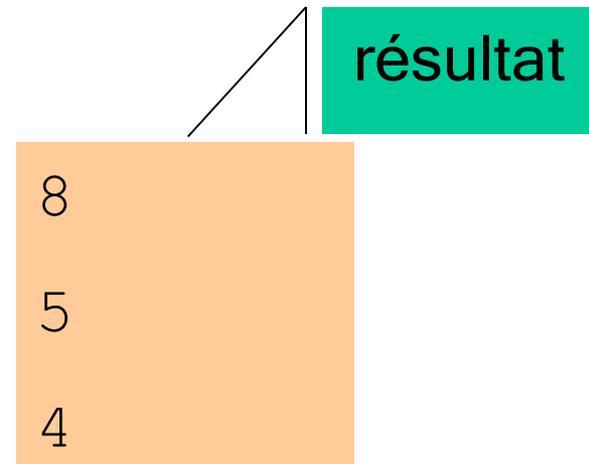
```
class ListeInt{
  private int valeur;
  private ListeInt suivant;
  public ListeInt(int premier, ListeInt reste){
    valeur = premier;
    suivant = reste;
  }
  // insertion d'un élément en tête de liste
  public ListeInt cons(int e){....}
  // récupération de la valeur de la tête de liste
  public int tete(){....}
  // récupération de la liste privée de son premier élément
  public ListeInt queue(){....}
}
```

La classe Liste d'entiers (2/3)

```
// insertion d'un élément en tête de liste  
public ListeInt cons(int e) {  
    return new ListeInt(e, this);  
}  
// récupération de la valeur de la tête de liste  
public int tete() {  
    return this.valeur;  
}  
// récupération de la liste privée de son premier  
// élément  
public ListeInt queue() {  
    return this.suivant;  
}
```

La classe Liste d'entiers (3/3)

```
import static java.lang.System.*;
public class MainListe{
    public static void main(String[] args){
        ListeInt = new ListeInt( 4,null );
        l=l.cons(5);
        l=l.cons(8);
        do{
            out.println(l.tete());
            l=l.queue();
        }while( l!= null );
    }
}
```



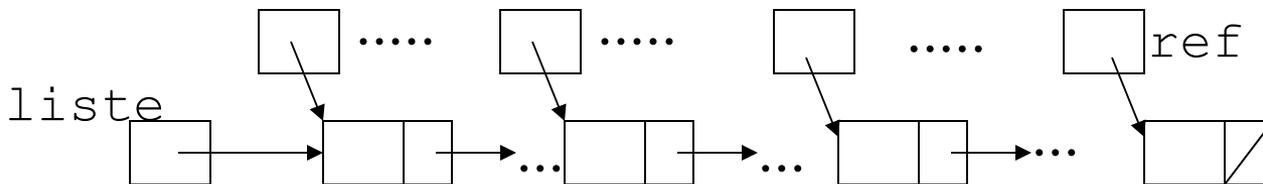
Autre opération: modification

```
// modifier la valeur de tête d'une liste  
public void setTete( int valeur ){  
    this.valeur = valeur;  
}
```

```
// modifier la queue d'une liste  
public void setQueue( ListeInt queue ){  
    this.suivant = queue;  
}
```

Autre opération: longueur

```
// longueur d'une liste  
public int longueur() {  
    int lg = 0;  
    ListeInt ref = this;  
    while( ref!=null ) {  
        lg++;  
        ref=ref.queue();  
    }  
    return lg;  
}
```



Autre opération: appartenance

```
// appartenance d'une valeur à une liste  
public boolean contient( int valeur ){  
    ListeInt ref = this;  
    while(ref != null) {  
        if(ref.tete() == valeur)  
            return true;  
        else  
            ref=ref.queue();  
    }  
    return false;  
}
```

Autre opération: concaténation

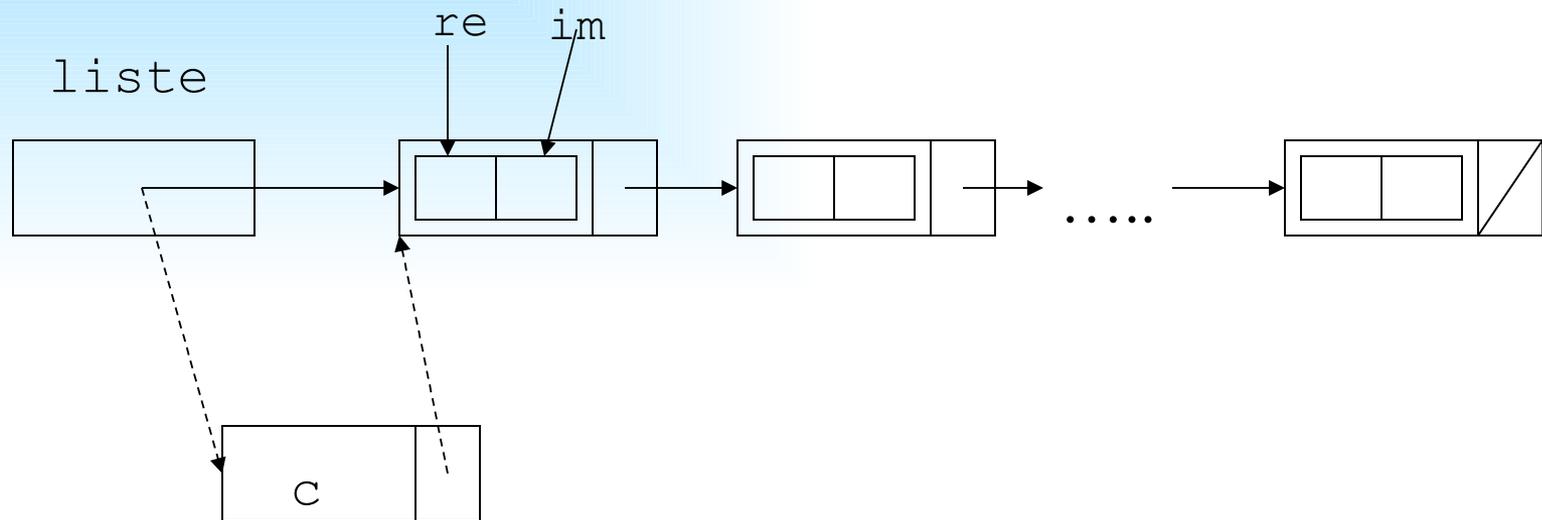
```
// concaténer la liste réceptrice avec une autre liste  
public void concatener(ListeInt l){  
    ListeInt ref = this;  
    while(ref.queue() != null){  
        // parcours jusqu'à la dernière cellule de la liste  
        ref = ref.queue();  
    }  
    ref.setQueue( l );  
}
```

Autre liste: liste de complexes

```
class Complexe{
    private float re,im;
    public Complexe( float re,float im){
        this.re = re; this.im = im;
    }
}

class ListeComplexe{
    Complexe valeur;
    ListeComplexe suivant;
    public ListeComplexe(Complexe c,ListeComplexe l){
        this.valeur = c; this.suivant = l;
    }
    public ListeComplexe ajouter
        (Complexe c, ListeComplexe liste){
        liste = new ListeComplexe(c,this);
    }
}
```

ajouter(Complexe c, ListeComplexe liste)



Exemple : liste des mots d'un texte

Comment rassembler en une unique structure, des données dont le nombre sera déterminé dynamiquement ?

Exemple : programme qui parcourt un texte et ajoute chaque mot nouveau à une liste. Si ce mot est déjà présent, il incrémente son nombre d'occurrences.

On considère (pour simplifier) que le texte ne compte qu'un mot par ligne.

Exemple (1/10): typage

```
class Donnee{  
    private String mot;  
    private int occ = 0;  
    public Donnee( String mot, int occ){  
        this.mot = mot;  
        this.occ = occ;  
    }  
    public void incrementerOcc(){  
        occ++;  
    }  
}
```

Exemple (1/10): typage

```
class Liste{
  private Donnee valeur;
  private Liste suivant;
  public Liste(Donnee premier, Liste reste){
    valeur = premier;
    suivant = reste;
  }
  // insertion d'un élément en tête de liste
  public Liste cons(Donnee e){....}
  // récupération de la valeur de la tête de liste
  public Donnee tete(){....}
  // récupération de la liste privée de son premier élément
  public Liste queue(){....}
}
```

Exemple (2/10) : algorithme

début

ouvrir le fichier texte en lecture;

lire un mot;

créer une liste initialisée avec le mot lu

tant que

il reste des mots à lire

faire

lire un mot;

traiter ce mot;

fin faire;

afficher la liste obtenue;

fin.

Exemple (3/10) : raffinement

2 méthodes apparaissent (à inclure dans la classe `Liste`):

- Ajouter une donnée à la liste.

```
Liste traiter(String mot);
```

- Afficher la liste

```
void println();
```

Exemple (4/10) : programme principal

```
package listes;
import java.util.Scanner;
import java.io.File;
import java.io.FileNotFoundException;
public class Mots{
    public void main() throws FileNotFoundException{
        Scanner in = new Scanner( new
                                   File("D:\\ProjetsBlueJ\\texte.txt") );
        String mot = in.next();
        Liste liste = new Liste(new Donnee(mot,1), null);
        while( in.hasNext() ){
            mot = in.next();
            liste = liste.traiter( mot );
        }
        liste.println();
    }
}
```

Exemple (5/10) : algorithme de traiter (version récursive)

début

si le mot lu est le premier de la liste

alors

incrémenter le nombre d'occurrences de ce mot;

sinon

si la liste ne possède qu'un seul élément

ajouter la donnée en tête de la liste;

sinon

retourner la liste après avoir ajouter le mot dans la queue de la liste;

fin si;

fin si;

fin;

Exemple (6/10) : code de traiter (version récursive)

```
// ne s'applique qu'à une liste non vide
public Liste traiter( String mot ) {
    Donnee donnee = this.valeur;
    if (mot.equals(donnee.mot)) {
        donnee.incrementerOcc();
        return this;
    }
    else
        if (this.suivant == null)
            return this.cons(new Donnee(mot, 1));
        else
            return new Liste(donnee, this.suivant.traiter(mot));
}
```

Exemple (7/10) : algorithme de traiter (version itérative)

// on suppose la liste non vide

début

curseur<-liste;

tant que non vide(curseur) **faire**

si mot lu = premier mot du curseur

alors incrémenter première cellule du curseur;

sinon curseur<-suivant du curseur;

fin si;

fin tant que;

ajouter nouvelle cellule en tête de liste;

fin;

Exemple (8/10) : code de traiter (version itérative)

```
public Liste traiter( String mot ) {
    Liste ref = this;
    Donnee donneeCourante=ref.valeur;
    String motCourant = null;
    while( ref!=null ) {
        motCourant = donneeCourante.mot;
        if( mot.equals(motCourant)) {
            ref.valeur.incrementerOcc();
            return this;
        }
        else
            ref = ref.suivant;
            donneeCourante = ref.valeur;
    }
    return this.cons(new Donnee(mot,1));
}
```

Exemple (9/10) : code de println

```
// méthode de la classe Liste

public void println(){
    System.out.println(" mot      nombre d'occurences");
    Liste ref = this;
    Donnee donnee = this.valeur;
    String mot = null;
    int occ = 0;
    while(ref != null) {
        mot = donnee.mot;
        occ = donnee.occ;
        System.out.printf("%s=>%d\n", mot, occ);
        ref = ref.queue();
        donnee = ref.valeur;
    }
}
```

Exemple (10/10) : résultats

Fichier texte

un
arbre
binaire
est
soit
vide
soit
possede
un
sous
arbre
binaire
gauche
et
un
sous
arbre
binaire
droite

---- *affichage de la liste*
mot nombre d'occurences
 arbre=>3
binaire=>3
 est=>1
 soit=>2
 vide=>1
possede=>1
 sous=>2
gauche=>1
 et=>1
droite=>1
 un=>3

Arbre binaire

Structures naturellement adaptées à une grande variété d'algorithmes

- Recherche du meilleur coup à jouer (go, échecs)
- Recherche d'une donnée associée à une clé

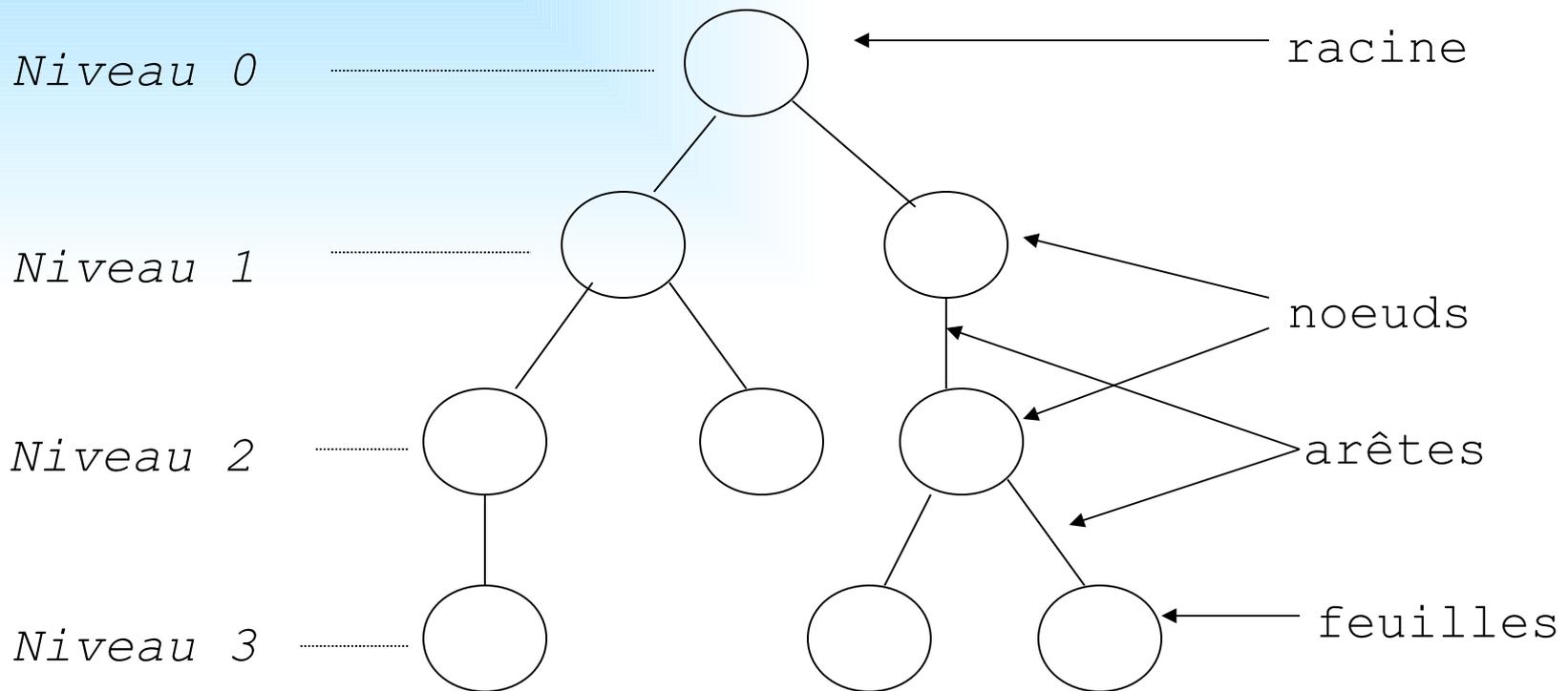
Définition récursive

Un arbre binaire est :

Soit vide

Soit un nœud qui possède un sous-arbre gauche et un sous-arbre droite eux mêmes arbres binaires

Représentation graphique



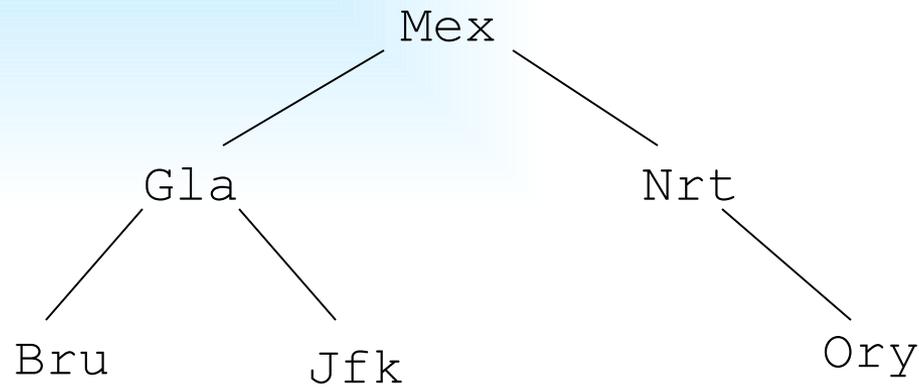
Exemple (1/9) : arbre binaire de recherche

L'ensemble des informations concernant les aéroports internationaux est classé dans un arbre binaire selon l'ordre lexicographique de leur sigle.

Ce choix (arbre plutôt que liste) améliore aussi bien la *recherche* que l'*ajout* d'un nouvel aéroport

La longueur d'un chemin de l'arbre est plus courte que la longueur de la séquence

Exemple (2/9) : un arbre



Propriété conservée :

```
cle(fils_gauche) < cle < cle(fils_droite)
```

Exemple (3/9) : typage

```
public class Element{
    private String sigle;
    private String infos;

    public Element(String sigle,String infos){
        this.sigle = sigle;
        this.infos = infos;
    }

    public String getInfos(){
        return infos;
    }

    public String getSigle(){
        return sigle;
    }
}
```

Exemple (4/9) : typage

```
public class ArbreBinaire{  
    Element valeur;  
    ArbreBinaire filsGauche;  
    ArbreBinaire filsDroite;  
  
    public ArbreBinaire(Element valeur,  
                        ArbreBinaire filsGauche,  
                        ArbreBinaire filsDroite){  
        this.valeur = valeur;  
        this.filsGauche = filsGauche;  
        this.filsDroite = filsDroite;  
    }  
}
```

Exemple (4/9) : recherche

Soient a l'arbre binaire de recherche, ag et ad respectivement ses fils gauche et droite.

Soit un couple (sigle,info) noté $\langle s, d \rangle$, de type `Element` racine de a :

$$a = \langle \langle s, d \rangle, ag, ad \rangle$$

Exemple (5/9) : algorithme de recherche

Soit $a = \langle \langle s, d \rangle, ag, ad \rangle$

```
rechercher(s' : Sigle, a : ArbreBinaire) =
```

```
    si s=s' alors d
```

```
    sinon si s>s'
```

```
        alors rechercher(s', ag)
```

```
        sinon rechercher(s', ad)
```

```
    fin si;
```

```
    fin si;
```

```
fin si;
```

Exemple (6/9) : code de recherche

```
public String rechercher( String sigle ){
    if( this.sigle.equals(sigle) )
        return valeur.getInfos();
    else
        if( this.sigle>sigle)
            return rechercher(sigle, filsGauche);
        else
            return rechercher(sigle, filsDroite);
}
```

Exemple (7/9) : algorithme de inserer

inserer

```
(s' : Sigle, d' : Donnee, <<s, d>, ag, ad> : ArbreBinaire) =  
  si s/=s'  
    alors si s>s'  
      alors <<s, d>, inserer(s', d', ag), ad>  
      sinon <<s, d>, ag, inserer(s', d', ad)>  
      fin si;  
    fin si;  
fin si;
```

Exemple (8/9) : inserer

// on suppose que l'arbre n'est pas vide

```
public ArbreBinaire inserer(Element valeur){
    String MonSigle = this.valeur.getSigle();
    String UnSigle  = valeur.getSigle();
    if(!MonSigle.equals(UnSigle))
        if(MonSigle.compareTo(UnSigle)>0)
            if(filsGauche==null)
                return new ArbreBinaire(this.valeur,
                                        new ArbreBinaire(valeur,null,null),
                                        filsDroite);
            else
                return new ArbreBinaire(this.valeur,
                                        filsGauche.inserer(valeur),
                                        filsDroite);
```

.....

Exemple (9/9) : insérer (suite)

else

```
if (filsDroite==null)
```

```
    return new ArbreBinaire(this.valeur, filsGauche,
```

```
        new  
        ArbreBinaire(valeur, null, null));
```

```
else return new  
    ArbreBinaire(this.valeur, filsGauche,
```

```
        filsDroite.insérer(valeur));
```

else

```
    return this;
```

```
}
```