Programmation avec des objets : les Listes

1. Listes: principe
2. Programmation de Cellules chaînées
3. Utilisation d'une sentinelle
4. Listes doublement chaînées

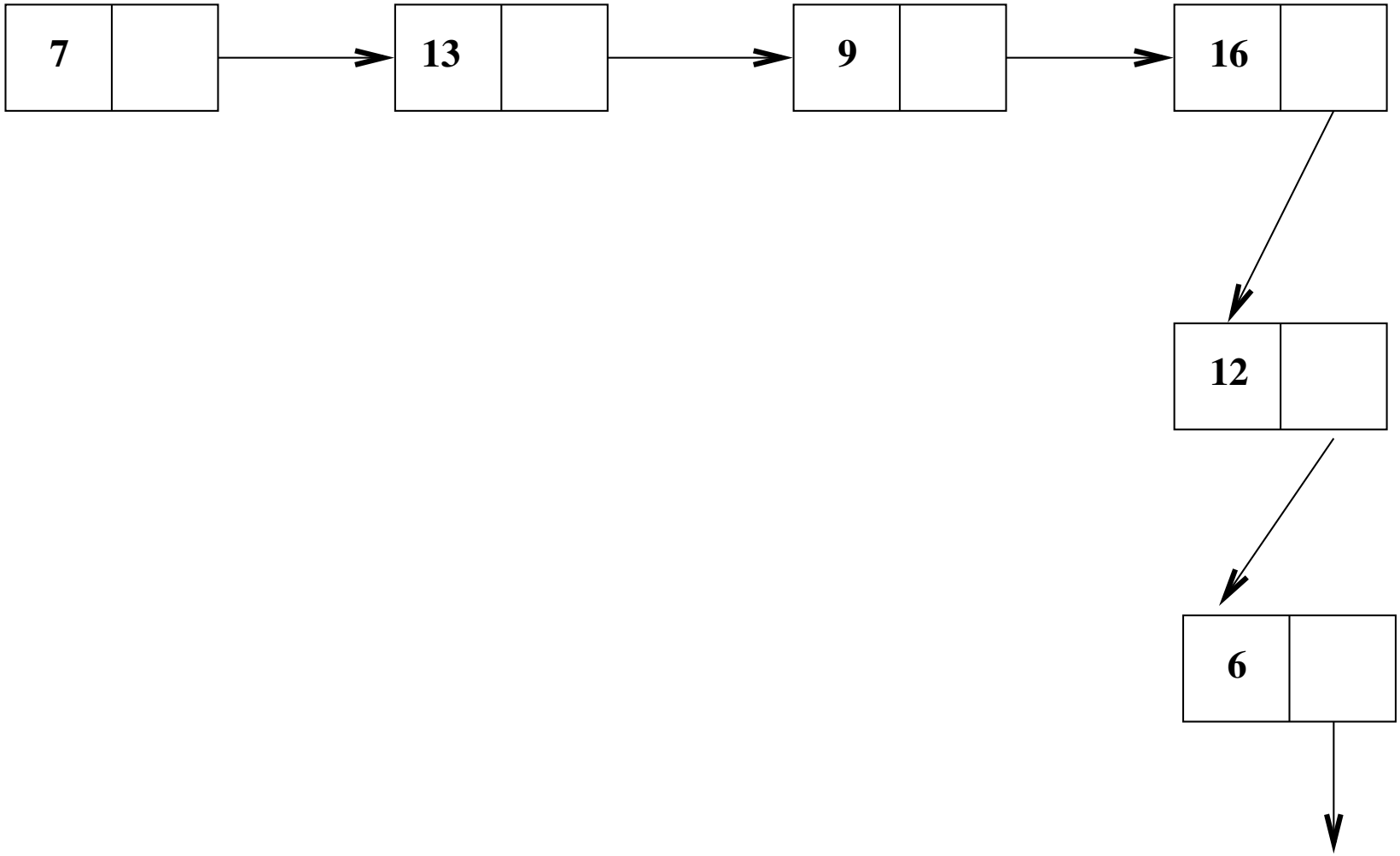
Pourquoi des listes?

- Contenir une suite d'objets
- On ne connaît pas à l'avance le nombre d'éléments
- On souhaite que le système récupère la place que l'on n'utilise plus

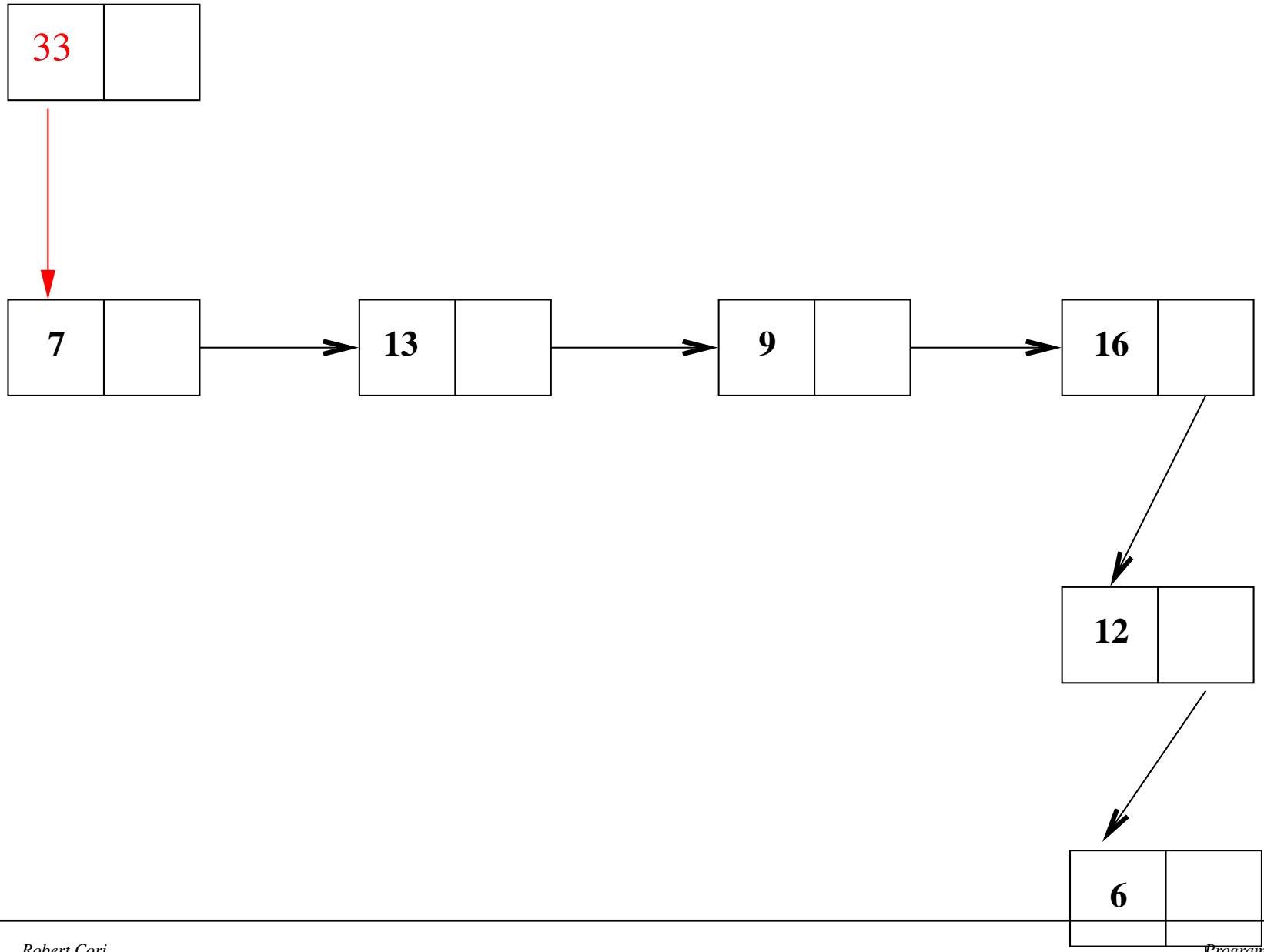
Principe des listes

- On crée des cellules au fur et à mesure des besoins
- Chaque cellule pointe (contient une référence) sur la suivante
- La derniere pointe sur l'indéfini
- Pour ajouter un élément, on crée une nouvelle cellule qui pointe sur la première
- Pour supprimer un élément, on modifie le pointage
- Le glaneur de cellules récupère les cellules sur lesquelles aucune autre ne pointe (en Java n'est pas à la charge du programmeur).

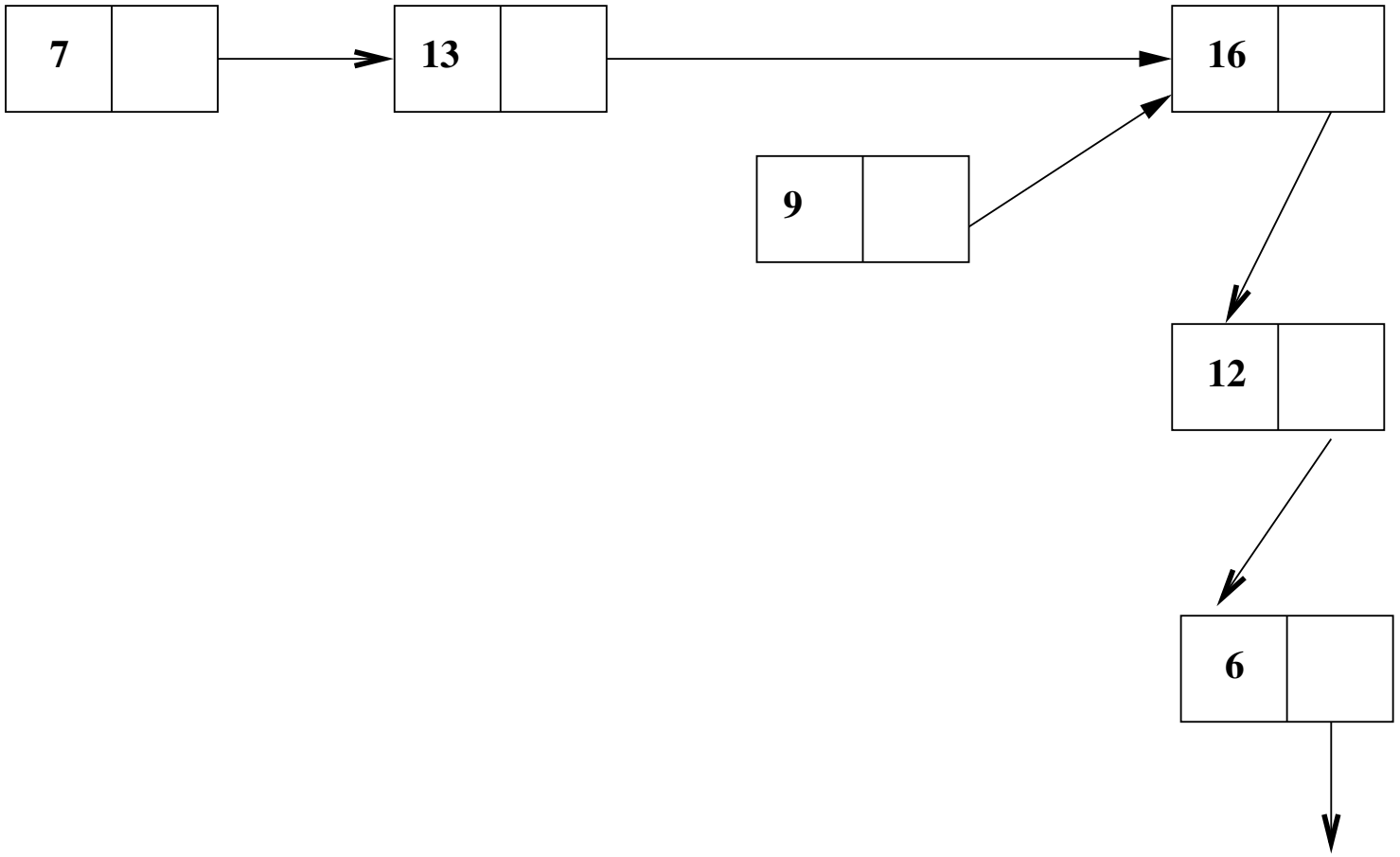
Une liste chaînée



Ajouter 33



Supprimer 9



Opérations sur les listes

Constante: La liste vide NIL

Opérations

Construire:	Liste	$\times$	Entier	$\rightarrow$	Liste
Valeur	Liste	$\rightarrow$	Entier		
Reste	Liste	$\rightarrow$	Liste		
Vide?	Liste	$\rightarrow$	Booléens		
Longueur	Liste	$\rightarrow$	Entiers		
Appartient	Liste	$\times$	Entier	$\rightarrow$	Booléens

Valeur, Reste ne sont pas définis pour NIL

Equations des Listes

Vide? (NIL)	=	true
Vide? (Cons(x, a))	=	false
Valeur (Cons(x,a))	=	a
Reste (Cons (x,a))	=	x
Longueur (NIL)	=	0
Longueur (Cons(x,a))	=	1 + Longeur(x)
Appartient (NIL, a)	=	false
Appartient (Cons(x,a) , a)	=	true
Appartient (Cons(x,b) , a)	=	Appartient(x,a) *(Si $a \neq b$)*

Représentation simple d'une liste en Java

- Une cellule contient une information (un entier par ex.) et pointe sur la cellule suivante.
- Pointer sur une cellule revient à contenir une *référence* sur l'objet cellule correspondante.
- On a ainsi la formule:

Liste = NIL Ou (Entier, Liste)

- Noter que `null` désigne l'objet vide en Java

Attention à `null` !!

- On ne peut pas appliquer une méthode à `null`
- `nullPointerException`
- faire un test avant d'appeler une méthode

La classe liste simple: premières méthodes

```
class Liste{
    private int val;
    private Liste suiv;
    public Liste (int a) {
        val = a; suiv = null;
    }
    public Liste (int a, Liste x){
        val = a; suiv = x;
    }
    public static boolean estVide(Liste x){
        return (x == null);
    }
```

Classe Liste (valeur, reste, longueur, appartient)

```
public int valeur(){
    return val;
}
public Liste reste(){
    return suiv;
}
 public int longueur(){
    if (suiv == null) return 1;
    else return 1 + reste().longueur();
}
public boolean appartient (int a){
    if (val == a) return true;
    if (suiv == null) return false;
    else return reste().appartient(a);
}
```

Classe Liste (ajouts)

```
public void  ajoutDebut(int a){
    Liste x = new Liste(this.val, suiv);
    this.val = a;
    this.suiv = x;
}
public void ajoutFin (int a){
    if (suiv == null) {
        Liste x = new Liste (a);
        suiv = x;}
    else reste().ajoutFin(a);
}
```

Classe Liste (impression)

```
public String toString() {
    if (suiv == null)
        return valeur() + "->> " ;
    else
        return valeur() + " -> " + reste().toString();
}
```

Classe Liste (suppression)

```
public void supprimer (int a){
    Liste x = this;
    if (x.val == a)
          if  (suiv == null)
              System.out.println("oh la liste nulle");
          else {
              val = suiv.val;
              suiv = suiv.suiv;
           }
    else if (suiv != null) suiv.supprimerAux(this, a);
}
public void supprimerAux (Liste x, int a){
    if (val == a) x.suiv = this.suiv;
    else if (suiv != null) suiv.supprimerAux(this, a);
}
```

Listes avec sentinelle

```
class Liste{
    private int val;
    private Liste suiv;
    Liste () {// sentinelle
        val = 0; suiv = null;
    }
    private Liste (int a, Liste x){
        val = a; suiv = x;}
    Liste (int a) {
        Liste x = new Liste(a, null);
        val = 0;
        suiv= x;
    }
    public boolean estVide(){
        return suiv == null;
    }
```

Listes avec sentinelle (suite)

```
static boolean estVide(Liste x){
    return (x.suiv == null);
}
public int valeur(){
    if (!estVide()) return suiv.val;
    System.out.println("valeur sur liste vide");
    return -1;
}
public Liste reste(){
    if (!estVide()) return new Liste(0, suiv.suiv);;
    System.out.println(" reste sur liste vide");
    return null;}
```

Listes avec sentinelle (suite)

```
 public int longueur(){
    if (estVide()) return 0;
    else return 1 + reste().longueur();
}
public boolean appartient (int a){
    if (estVide()) return false;
    else if (valeur() == a) return true;
    else return reste().appartient(a);
}
public String toString() {
    if (estVide()) return " ";
    else
        return valeur() + " -> " + reste().toString();
}
```

Listes avec sentinelle (ajouts, suppression)

```
public void  ajoutDebut(int a){
     Liste x = new Liste(a, suiv);
     suiv = x;
}
public void ajoutFin (int a){
     Liste x;
     for ( x = this; x.suiv != null; x = x.suiv);
     x.suiv = new Liste(a, null);
}
public void supprimer (int a){
   Liste x;
     for (x = this; x.suiv!= null && x.suiv.val != a
           ;x = x.suiv);
     if (x.suiv != null) x.suiv = x.suiv.suiv;
}
```

Listes doublement chaînées

```
class Liste{
    private int val;
    private Liste suiv;
    private Liste pred;
    public Liste (int a) {
       val = a; suiv = null; pred = null;}
    public Liste (int a, Liste x){
        val = a; suiv = x; pred = null;
        if ( x != null)  x.pred = this;}

    public static boolean estVide(Liste x){
        return (x == null);}
    public int valeur(){
        return val;}
```

Listes doublement chaînées(suite)

```
public Liste reste(){
    return suiv;}
public int longueur(){
    if (suiv == null) return 1;
    else return 1 + reste().longueur();}
public void  ajoutDebut(int a){
    int b = this.valeur();
    Liste u = this.reste();
    Liste x = new Liste(b, u);
    x.pred = this;
    this.val = a;
    this.suiv = x;
}
```

Listes doublement chaînées(suite)

```
public void ajoutFin (int a){
    if (suiv == null) {
        Liste x = new Liste (a);
        x.pred = this;
        suiv = x;
    }
    else reste().ajoutFin(a);
}
```

Listes doublement chaînées(suite)

```
public boolean appartient (int a){
    if (val == a) return true;
    if (suiv == null) return false;
    else return reste().appartient(a);
}
```

```
public String toString() {
    if (suiv == null)
        return valeur() + "->> " ;
    else
        return valeur() + " -> " + reste().toString();
}
```

Listes doublement chaînées(suite)

```
public void supprimer (int a){
    Liste x = this;
    if (x.val == a)
        if (x.suiv == null)
        System.out.println(" on trouve la liste vide");
        else {
            x.val = x.suiv.val;
            x.suiv= x.suiv.suiv;
        }
    else{
        while (x!= null && x.val != a) x = x.suiv;
        if (x != null){
        Liste y = x.pred, z = x.suiv;
        y.suiv = z;
        if (z != null) z.pred = y;
        }
    }
}
```

Listes doublement chaînées(fin)

```
public void afficheEnvers(){
    Liste x = this;
    while (x.suiv != null) x = x.suiv;
    while (x != null) {
        System.out.print(" <- " + x.val);
        x = x.pred;
    }
        System.out.println("<-");
}
```

Fonctions sur les listes

```
static Liste reverse (Liste x){
   // attention destruction de la liste de depart
   if (Liste.estVide( x.reste())) return x;
   else {
       Liste u = reverse(x.reste());
       int a = x.valeur();
       u.ajoutFin(a); return u;}
    }
static Liste reverseAux(Liste u, Liste v){
     if (u.estVide()) return v;
     else {
         v.ajoutDebut(u.valeur());
         return reverseAux(u.reste(), v);
     }}
static Liste reverse1(Liste u) {
    return reverseAux(u, newListe());}
```

Remarques sur les listes

- Attention aux effets de bords
- Liste d'appel modifiée ou conservée?
- Fonction sur la liste vide déclanche une exception
- Destructif ou pas
- Pas besoin de libérer de la place le glaneur de cellules (garbage collector) le fera
- Attention à ne jamais effectuer: `null.valeur` ou `null.reste()` .....