4 TAD Collection

Collection
Listes Chainées (Simple, Double)
Ensemble

TAD

- Algorithmique de base
  - Résoudre problème = décomposer en ss-pbs
  - Résoudre ss-problème = combinaison Boucles & Conditions
  - Structures
    - Données séquentielles (tableaux)

TAD

- Algorithmique de base
  - Résoudre problème = décomposer en ss-pbs
  - Résoudre ss-problème = combinaison Boucles & Conditions
  - Structures
    - Données séquentielles (tableaux)
- Algorithmique avancée
  - Compromis entre
    - Structure de données la plus petite & intelligente possible
    - Algo prenant le moins de temps possible et/ou moins de place possible
    - $\Rightarrow$ Complexité (Module suivant)
  - Structures
    - Données linéaires (listes…)
    - Données arborescentes (arbres & graphes)
    - Récursives = structures qui s'auto-référencent!

TAD

- Algorithmique de base
  - Résoudre problème = décomposer en ss-pbs
  - Résoudre ss-problème = combinaison Boucles & Conditions
  - Structures
    - Données séquentielles (tableaux)
- Algorithmique avancée
  - Compromis entre
    - Structure de données la plus petite & intelligente possible
    - Algo prenant le moins de temps possible et/ou moins de place possible
    - $\Rightarrow$ Complexité (Module suivant)
  - Structures
    - Données linéaires (listes…)
    - Données arborescentes (arbres & graphes)
    - Récursives = structures qui s'auto-référencent!
- https://en.wikipedia.org/wiki/Category:Abstract_data_types

TAD Collection

- **TAD Collection**
  - Ensemble fini de plusieurs éléments (+sieurs occurrences) de type Element

TAD Collection

- **TAD Collection**
  - Ensemble fini de plusieurs éléments (+sieurs occurrences) de type Element
  - Méthodes
    - vide () : ∅ → Collection

TAD Collection

- **TAD Collection**
  - Ensemble fini de plusieurs éléments (+sieurs occurrences) de type Element
  - Méthodes
    - vide () : $\varnothing \rightarrow$ Collection
    - cons () : Collection $\times$ Element $\rightarrow$ Collection

TAD Collection

- **TAD Collection**
  - Ensemble fini de plusieurs éléments (+sieurs occurrences) de type Element
  - Méthodes
    - vide () : $\varnothing \rightarrow$ Collection
    - cons () : Collection $\times$ Element $\rightarrow$ Collection
    - tete () : Collection $\rightarrow$ Element

TAD Collection

- **TAD Collection**
  - Ensemble fini de plusieurs éléments (+sieurs occurrences) de type Element
  - Méthodes
    - vide () : $\varnothing \rightarrow$ Collection
    - cons () : Collection $\times$ Element $\rightarrow$ Collection
    - tete () : Collection $\rightarrow$ Element
    - queue () : Collection $\rightarrow$ Collection

TAD Collection

- **TAD Collection**
  - Ensemble fini de plusieurs éléments (+sieurs occurrences) de type Element
  - Méthodes
    - vide () : $\varnothing \rightarrow$ Collection
    - cons () : Collection $\times$ Element $\rightarrow$ Collection
    - tete () : Collection $\rightarrow$ Element
    - queue () : Collection $\rightarrow$ Collection
    - estVide () : Collection $\rightarrow$ Bool

TAD Collection

- **TAD Collection**
  - Ensemble fini de plusieurs éléments (+sieurs occurrences) de type Element
  - Méthodes
    - vide () : $\varnothing \rightarrow$ Collection
    - cons () : Collection $\times$ Element $\rightarrow$ Collection
    - tete () : Collection $\rightarrow$ Element
    - queue () : Collection $\rightarrow$ Collection
    - estVide () : Collection $\rightarrow$ Bool
    - supp () : Collection $\times$ Element $\rightarrow$ Collection

TAD Collection

- **TAD Collection**
  - Ensemble fini de plusieurs éléments (+sieurs occurrences) de type Element
  - Méthodes
    - vide () : $\varnothing \rightarrow$ Collection
    - cons () : Collection $\times$ Element $\rightarrow$ Collection
    - tete () : Collection $\rightarrow$ Element
    - queue () : Collection $\rightarrow$ Collection
    - estVide () : Collection $\rightarrow$ Bool
    - supp () : Collection $\times$ Element $\rightarrow$ Collection
    - contient () : Collection $\times$ Element $\rightarrow$ Booleen

TAD Collection

- **TAD Collection**
  - Ensemble fini de plusieurs éléments (+sieurs occurrences) de type Element
  - Méthodes
    - vide () : $\varnothing \rightarrow$ Collection
    - cons () : Collection $\times$ Element $\rightarrow$ Collection
    - tete () : Collection $\rightarrow$ Element
    - queue () : Collection $\rightarrow$ Collection
    - estVide () : Collection $\rightarrow$ Bool
    - supp () : Collection $\times$ Element $\rightarrow$ Collection
    - contient () : Collection $\times$ Element $\rightarrow$ Booleen
    - taille () : Collection $\rightarrow$ Entier

TAD Collection

- **TAD Collection**
  - Ensemble fini de plusieurs éléments (+sieurs occurrences) de type Element
  - Méthodes
    - vide () : $\varnothing \rightarrow$ Collection
    - cons () : Collection $\times$ Element $\rightarrow$ Collection
    - tete () : Collection $\rightarrow$ Element
    - queue () : Collection $\rightarrow$ Collection
    - estVide () : Collection $\rightarrow$ Bool
    - supp () : Collection $\times$ Element $\rightarrow$ Collection
    - contient () : Collection $\times$ Element $\rightarrow$ Booleen
    - taille () : Collection $\rightarrow$ Entier
  - Axiomes (c $\in$ Collection; x,y $\in$ Element)

TAD Collection

- **TAD Collection**
  - Ensemble fini de plusieurs éléments (+sieurs occurrences) de type Element
  - Méthodes
    - vide () : $\varnothing \rightarrow$ Collection
    - cons () : Collection $\times$ Element $\rightarrow$ Collection
    - tete () : Collection $\rightarrow$ Element
    - queue () : Collection $\rightarrow$ Collection
    - estVide () : Collection $\rightarrow$ Bool
    - supp () : Collection $\times$ Element $\rightarrow$ Collection
    - contient () : Collection $\times$ Element $\rightarrow$ Booleen
    - taille () : Collection $\rightarrow$ Entier
  - Axiomes (c $\in$ Collection; x,y $\in$ Element)
    - estVide(vide()) = Vrai
    - tete(cons(x,c))=x

TAD Collection

- **TAD Collection**
  - Ensemble fini de plusieurs éléments (+sieurs occurrences) de type Element
  - Méthodes
    - vide () : $\varnothing \rightarrow$ Collection
    - cons () : Collection $\times$ Element $\rightarrow$ Collection
    - tete () : Collection $\rightarrow$ Element
    - queue () : Collection $\rightarrow$ Collection
    - estVide () : Collection $\rightarrow$ Bool
    - supp () : Collection $\times$ Element $\rightarrow$ Collection
    - contient () : Collection $\times$ Element $\rightarrow$ Booleen
    - taille () : Collection $\rightarrow$ Entier
  - Axiomes (c $\in$ Collection; x,y $\in$ Element)
    - estVide(vide()) = Vrai
    - estVide(cons(c,x)) = Faux
    - tete(cons(x,c))=x
    - queue(cons(x,c))=c

TAD Collection

- **TAD Collection**
  - Ensemble fini de plusieurs éléments (+sieurs occurrences) de type Element
  - Méthodes
    - vide () : $\varnothing \rightarrow$ Collection
    - cons () : Collection $\times$ Element $\rightarrow$ Collection
    - tete () : Collection $\rightarrow$ Element
    - queue () : Collection $\rightarrow$ Collection
    - estVide () : Collection $\rightarrow$ Bool
    - supp () : Collection $\times$ Element $\rightarrow$ Collection
    - contient () : Collection $\times$ Element $\rightarrow$ Booleen
    - taille () : Collection $\rightarrow$ Entier
  - Axiomes (c $\in$ Collection; x,y $\in$ Element)
    - estVide(vide()) = Vrai
    - estVide(cons(c,x)) = Faux
    - contient(vide(),x) = Faux
    - tete(cons(x,c))=x
    - queue(cons(x,c))=c

TAD Collection

- **TAD Collection**
  - Ensemble fini de plusieurs éléments (+sieurs occurrences) de type Element
  - Méthodes
    - vide () : $\varnothing \rightarrow$ Collection
    - cons () : Collection $\times$ Element $\rightarrow$ Collection
    - tete () : Collection $\rightarrow$ Element
    - queue () : Collection $\rightarrow$ Collection
    - estVide () : Collection $\rightarrow$ Bool
    - supp () : Collection $\times$ Element $\rightarrow$ Collection
    - contient () : Collection $\times$ Element $\rightarrow$ Booleen
    - taille () : Collection $\rightarrow$ Entier
  - Axiomes (c $\in$ Collection; x,y $\in$ Element)
    - estVide(vide()) = Vrai
    - estVide(cons(c,x)) = Faux
    - contient(vide(),x) = Faux
    - supp(cons(c,x),x) = c
    - tete(cons(x,c))=x
    - queue(cons(x,c))=c

TAD Collection

- **TAD Collection**
  - Ensemble fini de plusieurs éléments (+sieurs occurrences) de type Element
  - Méthodes
    - vide () : $\varnothing \rightarrow$ Collection
    - cons () : Collection $\times$ Element $\rightarrow$ Collection
    - tete () : Collection $\rightarrow$ Element
    - queue () : Collection $\rightarrow$ Collection
    - estVide () : Collection $\rightarrow$ Bool
    - supp () : Collection $\times$ Element $\rightarrow$ Collection
    - contient () : Collection $\times$ Element $\rightarrow$ Booleen
    - taille () : Collection $\rightarrow$ Entier
  - Axiomes (c $\in$ Collection; x,y $\in$ Element)
    - estVide(vide()) = Vrai
    - estVide(cons(c,x)) = Faux
    - contient(vide(),x) = Faux
    - supp(cons(c,x),x) = c
    - tete(cons(x,c))=x
    - queue(cons(x,c))=c
  - Axiomes sur *size* ?

TAD Collection

- **TAD Collection**
  - Ensemble fini de plusieurs éléments (+sieurs occurrences) de type Element
  - Méthodes
    - vide () : $\varnothing \rightarrow$ Collection
    - cons () : Collection $\times$ Element $\rightarrow$ Collection
    - tete () : Collection $\rightarrow$ Element
    - queue () : Collection $\rightarrow$ Collection
    - estVide () : Collection $\rightarrow$ Bool
    - supp () : Collection $\times$ Element $\rightarrow$ Collection
    - contient () : Collection $\times$ Element $\rightarrow$ Booleen
    - taille () : Collection $\rightarrow$ Entier
  - Axiomes (c $\in$ Collection; x,y $\in$ Element)
    - estVide(vide()) = Vrai
    - estVide(cons(c,x)) = Faux
    - contient(vide(),x) = Faux
    - supp(cons(c,x),x) = c

    tete(cons(x,c))=x

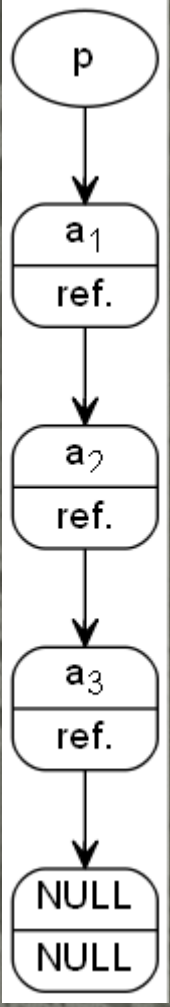
    queue(cons(x,c))=c
  - Axiomes sur *size* ?
  - https://en.wikipedia.org/wiki/Collection_(abstract_data_type)

Collection par Tableau

- Représentation séquentielle = Tableau
  - Utilisation d'un tableau contenant des E
  - Avantage(s)
    - Accès très rapide car direct en mémoire
  - Inconvénient(s)
    - Taille max fixée au départ !
    - Manipulation de la taille effective du tableau
    - Insertion assez couteuse
      - Au pire des cas, déplacement de tous les éléments du tableau

Collection par Liste Chainée

- Représentation chainée = Liste
  - Création d'une structure de données contenant
    E + un pointeur / référence sur E suivant
  - Avantage(s)
    - Minimisation de la mémoire utilisée
    - Pas besoin de fixer une taille initiale
    - Ajout facile
  - Inconvénient(s)
    - Accès plus compliqués

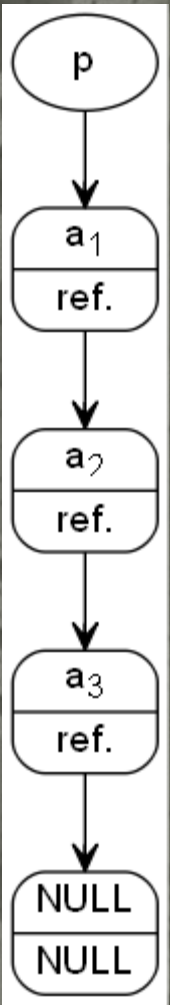


Collection par Liste Chainée

- **Représentation chainée = Liste**
  - **En Algo**

TAD Collection

vide () : ∅ → Collection
cons () : Collection × Element → Collection
tete () : Collection → Element
queue () : Collection → Collection
estVide () : Collection → Bool
taille () : Collection → Entier
contient () : Collection × Element → Booleen
supp () : Collection × Element → Collection



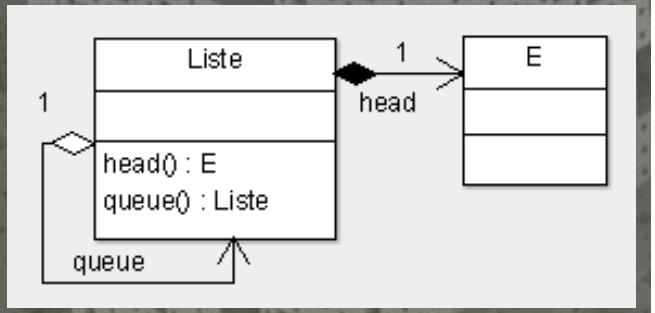
Collection par Liste Chainée

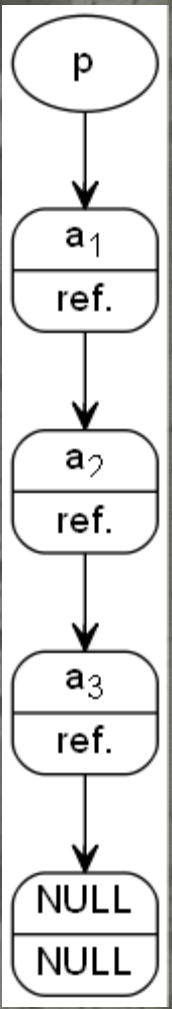
- **Représentation chainée = Liste**
  - **En Algo**

TAD Collection

vide () : ∅ → Collection
cons () : Collection × Element → Collection
tete () : Collection → Element
queue () : Collection → Collection
estVide () : Collection → Bool
taille () : Collection → Entier
contient () : Collection × Element → Booleen
supp () : Collection × Element → Collection

- **En UML**





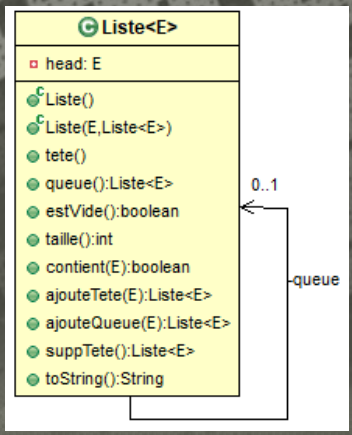
Collection par Liste Chainée

- **Représentation chainée = Liste**
  - **En Algo**

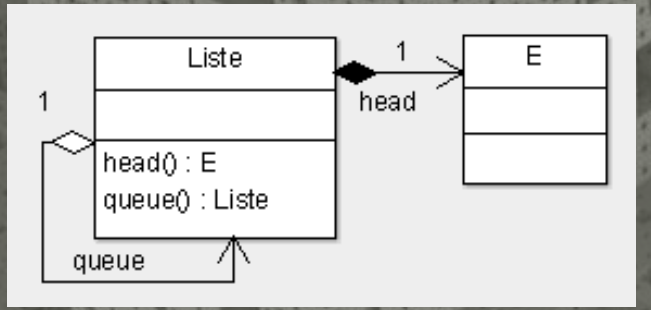
TAD Collection

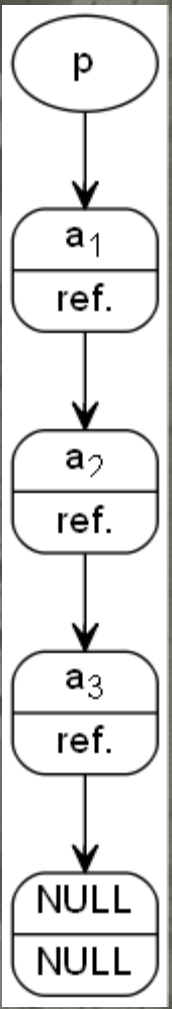
vide () : ∅ → Collection
cons () : Collection × Element → Collection
tete () : Collection → Element
queue () : Collection → Collection
estVide () : Collection → Bool
taille () : Collection → Entier
contient () : Collection × Element → Booleen
supp () : Collection × Element → Collection

  - **En java**



- **En UML**





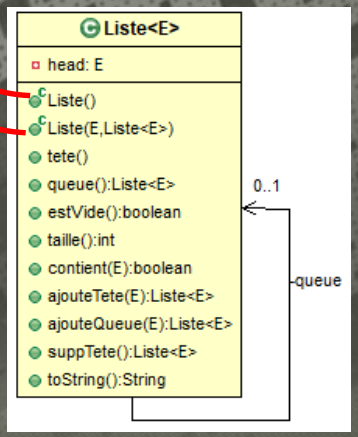
Collection par Liste Chainée

- **Représentation chainée = Liste**
  - **En Algo**

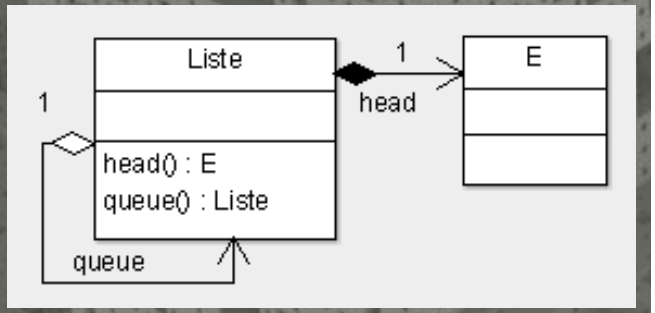
TAD Collection

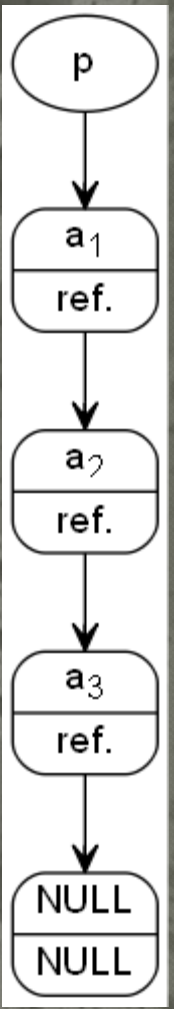
vide () : ∅ → Collection
cons () : Collection × Element → Collection
tete () : Collection → Element
queue () : Collection → Collection
estVide () : Collection → Bool
taille () : Collection → Entier
contient () : Collection × Element → Booleen
supp () : Collection × Element → Collection

  - **En java**



- **En UML**





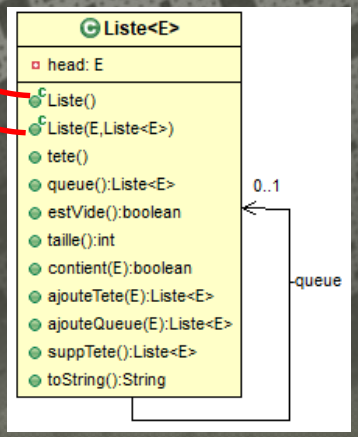
Collection par Liste Chainée

- **Représentation chainée = Liste**
  - **En Algo**

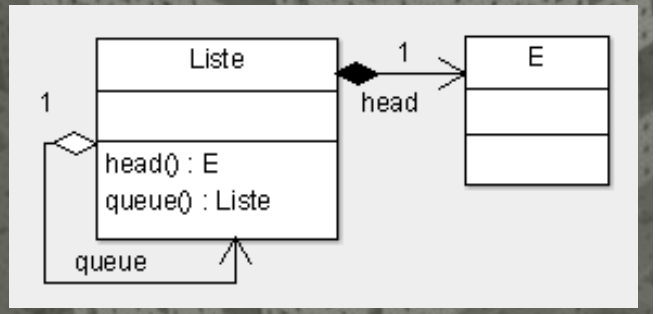
TAD Collection

vide () : ∅ → Collection
cons () : Collection × Element → Collection
tete () : Collection → Element
queue () : Collection → Collection
estVide () : Collection → Bool
taille () : Collection → Entier
contient () : Collection × Element → Booleen
supp () : Collection × Element → Collection

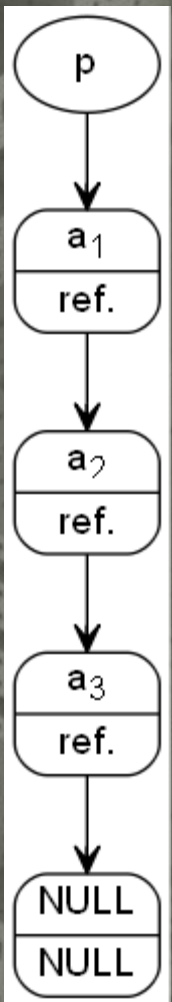
  - **En java**



- **En UML**




```
public class Liste<E> { //
    private E tete;
    private Liste<E> queue;

    public Liste() {
        tete = null;
        queue = null;
    }
    public Liste(E n, Liste<E> suivante) {
        tete=n ;
        queue = suivante ;
    }
    public E tete() {
        return tete;
    }
    public Liste<E> queue() {
        return queue;
    }
```



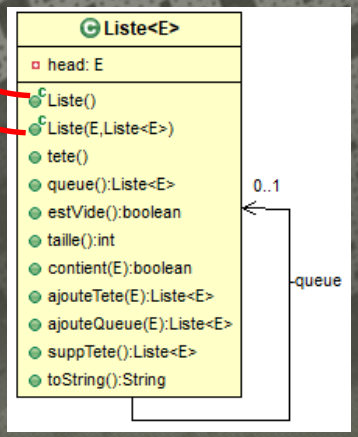
Collection par Liste Chainée

- **Représentation chainée = Liste**
  - **En Algo**

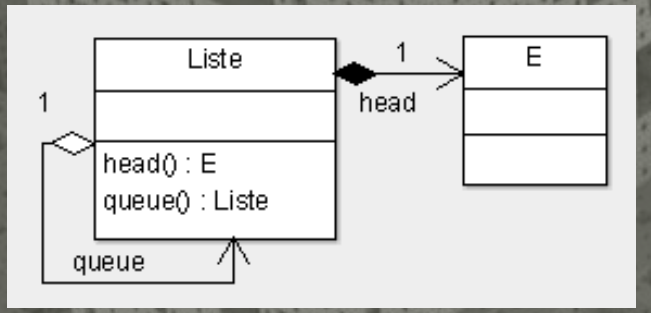
TAD Collection

vide () : ∅ → Collection
cons () : Collection × Element → Collection
tete () : Collection → Element
queue () : Collection → Collection
estVide () : Collection → Bool
taille () : Collection → Entier
contient () : Collection × Element → Booleen
supp () : Collection × Element → Collection

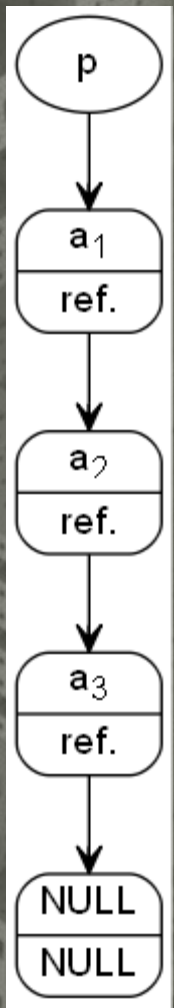
  - **En java**



- **En UML**



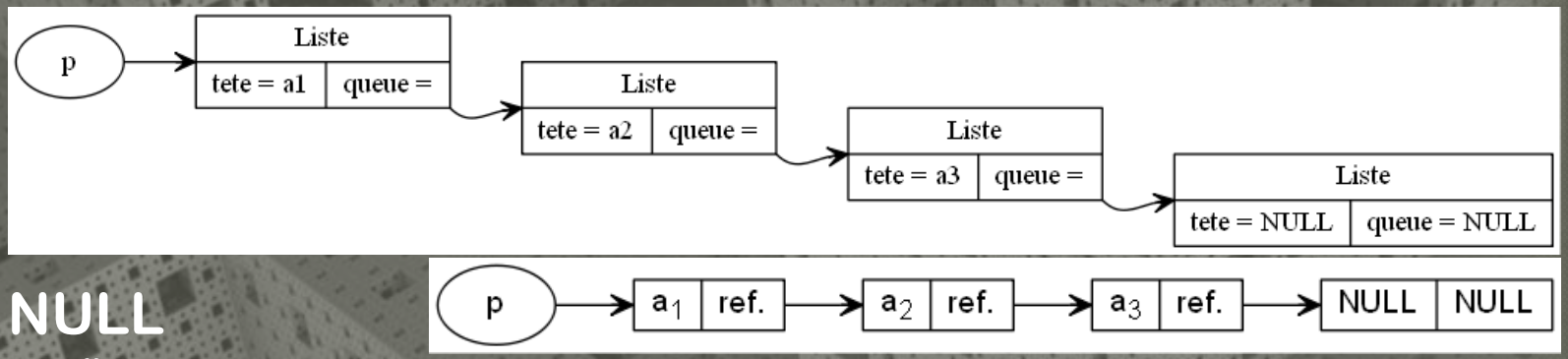

```
public class Liste<E> { //
    private E tete;
    private Liste<E> queue;

    public Liste() {
        tete = null;
        queue = null;
    }
    public Liste(E n, Liste<E> suivante) {
        tete=n ;
        queue = suivante ;
    }
    public E tete() {
        return tete;
    }
    public Liste<E> queue() {
        return queue;
    }
```




```
Liste<Integer> liste_vide = new Liste<Integer>() ;
Liste<Integer> liste_un = new Liste<Integer>(1, liste_vide) ;
```

Collection par Liste Chainée

- Soit



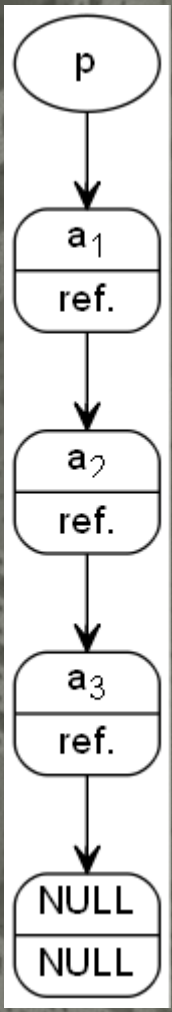
- p ≠ NULL
- p.tete() = a_1
- p.queue() ≠ NULL
- p.queue(). tete()=a_2
- p.queue().queue() ≠ NULL
- p.queue().queue(). tete()=a_3
- p.queue().queue().queue() ≠ NULL
- p.queue().queue().queue().estVide() =TRUE !
- p.queue().queue().queue().tete() = NULL
- p.queue().queue().queue().queue() = NULL

Collection par Liste Chainée

- Est vide ?

```
public class Liste<E> { //
    private E tete;
    private Liste<E> queue;

    public Liste() {
        tete = null;
        queue = null;
    }
    public Liste(E n, Liste<E> suivante) {
        tete=n ;
        queue = suivante ;
    }
```

```
public E tete() {
    return tete;
}
public Liste<E> queue() {
    return queue;
}
```



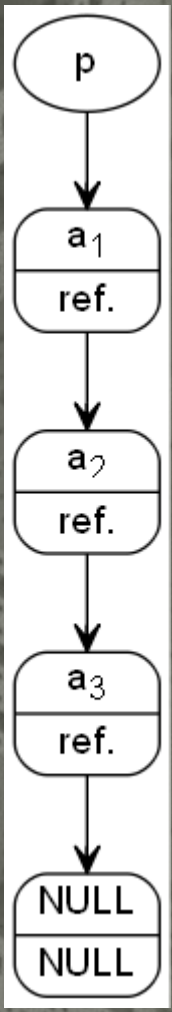
Collection par Liste Chainée

- Est vide ?

```
public boolean estVide() {
    return (tete==null) ;
}
```

```
public class Liste<E> { //
    private E tete;
    private Liste<E> queue;

    public Liste() {
        tete = null;
        queue = null;
    }
    public Liste(E n, Liste<E> suivante) {
        tete=n ;
        queue = suivante ;
    }
```

```
public E tete() {
    return tete;
}
public Liste<E> queue() {
    return queue;
}
```



Collection par Liste Chainée

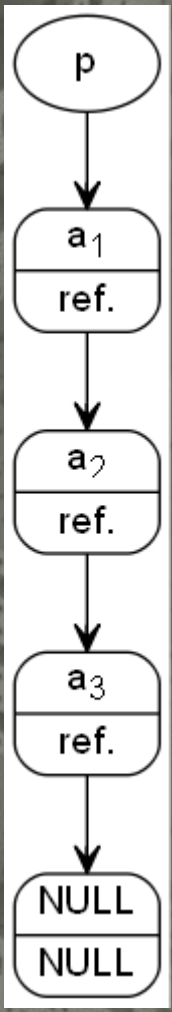
- Est vide ?

```
public boolean estVide() {
    return (tete==null) ;
}
```

- Longueur de la liste ?

```
public class Liste<E> { //
    private E tete;
    private Liste<E> queue;

    public Liste() {
        tete = null;
        queue = null;
    }
    public Liste(E n, Liste<E> suivante) {
        tete=n ;
        queue = suivante ;
    }
```

```
public E tete() {
    return tete;
}
public Liste<E> queue() {
    return queue;
}
```



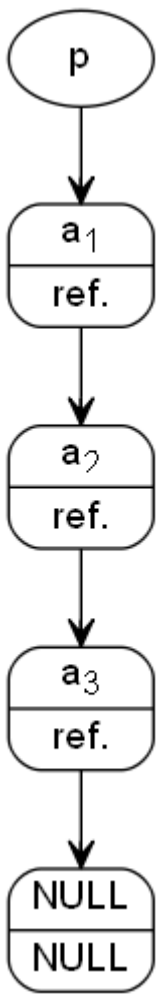
Collection par Liste Chainée

- Est vide ?

```
public boolean estVide() {
    return (tete==null) ;
}
```

```
public class Liste<E> { //
    private E tete;
    private Liste<E> queue;

    public Liste() {
        tete = null;
        queue = null;
    }
    public Liste(E n, Liste<E> suivante) {
        tete=n ;
        queue = suivante ;
    }
```

```
public E tete() {
    return tete;
}
public Liste<E> queue() {
    return queue;
}
```



- Longueur de la liste ?

```
Function taille(l : Liste) : Entier
  In      : l
  Do      : calcul la longueur de la liste l
1 begin
2   if estVide (l) then
3     return 0
4   else
5     return 1 + taille(queue(l))
```

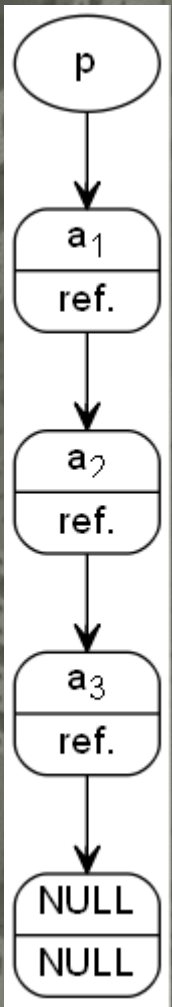
Collection par Liste Chainée

- Est vide ?

```
public boolean estVide() {
    return (tete==null) ;
}
```

```
public class Liste<E> { //
    private E tete;
    private Liste<E> queue;

    public Liste() {
        tete = null;
        queue = null;
    }
    public Liste(E n, Liste<E> suivante) {
        tete=n ;
        queue = suivante ;
    }
```

```
public E tete() {
    return tete;
}
public Liste<E> queue() {
    return queue;
}
```



- Longueur de la liste ?

```
Function taille(l : Liste) : Entier
  In      : l
  Do      : calcul la longueur de la liste l
1 begin
2   if estVide (l) then
3     return 0
4   else
5     return 1 + taille(queue(l))
```

```
public int taille() {
    if (estVide())
        return 0;
    else
        return 1 + queue.taille();
}
```

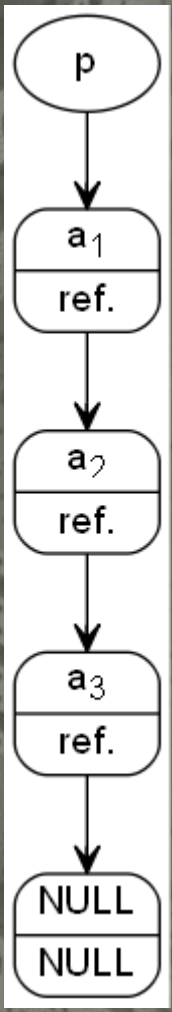
Collection par Liste Chainée

- Est vide ?

```
public boolean estVide() {
    return (tete==null) ;
}
```

```
public class Liste<E> { //
    private E tete;
    private Liste<E> queue;

    public Liste() {
        tete = null;
        queue = null;
    }
    public Liste(E n, Liste<E> suivante) {
        tete=n ;
        queue = suivante ;
    }
```

```
public E tete() {
    return tete;
}
public Liste<E> queue() {
    return queue;
}
```



- Longueur de la liste ?

```
Function taille(l : Liste) : Entier
  In      : l
  Do      : calcul la longueur de la liste l
1 begin
2   if estVide (l) then
3     return 0
4   else
5     return 1 + taille(queue(l))
```

```
public int taille() {
    if (estVide())
        return 0;
    else
        return 1 + queue.taille();
}
```

- Conversion en chaine ?

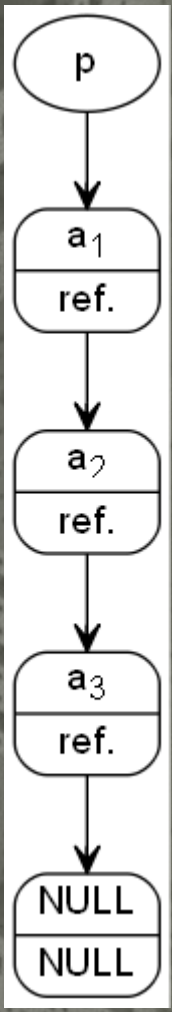
Collection par Liste Chainée

- Est vide ?

```
public boolean estVide() {
    return (tete==null) ;
}
```

```
public class Liste<E> { //
    private E tete;
    private Liste<E> queue;

    public Liste() {
        tete = null;
        queue = null;
    }
    public Liste(E n, Liste<E> suivante) {
        tete=n ;
        queue = suivante ;
    }
```

```
public E tete() {
    return tete;
}
public Liste<E> queue() {
    return queue;
}
```



- Longueur de la liste ?

```
Function taille(l : Liste) : Entier
  In      : l
  Do      : calcul la longueur de la liste l
1 begin
2   if estVide (l) then
3     return 0
4   else
5     return 1 + taille(queue(l))
```

```
public int taille() {
    if (estVide())
        return 0;
    else
        return 1 + queue.taille();
}
```

- Conversion en chaine ?

```
Function toString(l : Liste) : Chaine
  In      : l
  Do      : Convertit l en chaine
1 begin
2   if estVide (l) then
3     return "null"
4   else
5     return "(" + tete(l) + "," + toString(queue(l)) + ")"
```

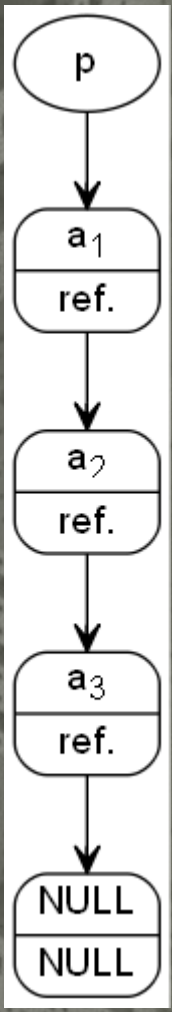
Collection par Liste Chainée

- Est vide ?

```
public boolean estVide() {
    return (tete==null) ;
}
```

```
public class Liste<E> { //
    private E tete;
    private Liste<E> queue;

    public Liste() {
        tete = null;
        queue = null;
    }
    public Liste(E n, Liste<E> suivante) {
        tete=n ;
        queue = suivante ;
    }
```

```
public E tete() {
    return tete;
}
public Liste<E> queue() {
    return queue;
}
```



- Longueur de la liste ?

```
Function taille(l : Liste) : Entier
  In      : l
  Do      : calcul la longueur de la liste l
1 begin
2   if estVide (l) then
3     return 0
4   else
5     return 1 + taille(queue(l))
```

```
public int taille() {
    if (estVide())
        return 0;
    else
        return 1 + queue.taille();
}
```

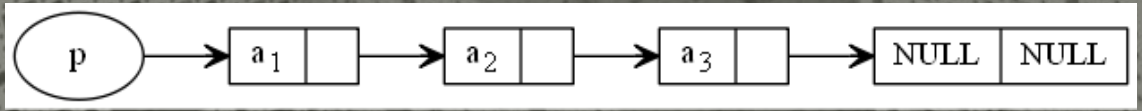
- Conversion en chaine ?

```
Function toString(l : Liste) : Chaine
  In      : l
  Do      : Convertit l en chaine
1 begin
2   if estVide (l) then
3     return "null"
4   else
5     return "(" + tete(l) + "," + toString(queue(l)) + ")"
```

```
@Override
public String toString() {
    if (estVide()){
        return "null";
    }
    else {
        return "("+tete()+ ", " + queue()+")";
    }
}
```

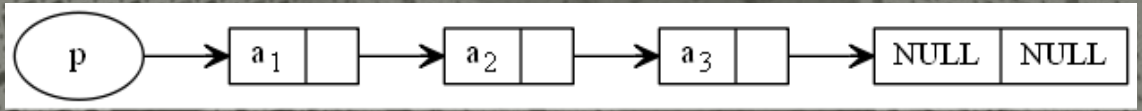
Liste Chainée

- Ajout d'un élément (en tête)
  - Soit sous forme fonctionnelle
    - Les objets manipulés ne sont pas modifiés
    - Avant



Liste Chainée

- **Ajout d'un élément (en tête)**
  - Soit sous forme fonctionnelle
    - Les objets manipulés ne sont pas modifiés
    - Avant

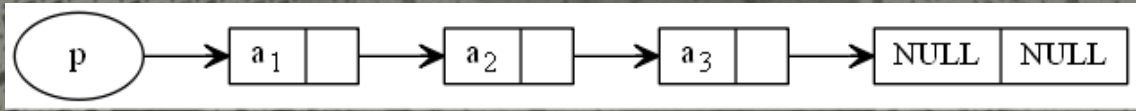


    - Après

```
public Liste<E> addTete(E v) {
    return  new Liste<E> (v,this) ;
}
```

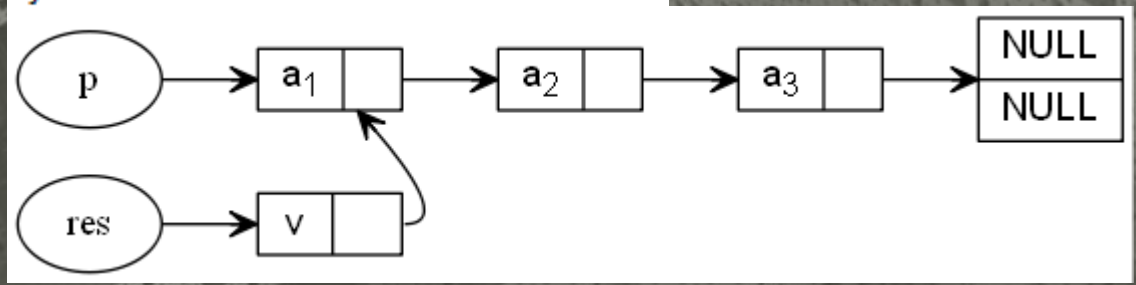
Liste Chainée

- **Ajout d'un élément (en tête)**
  - Soit sous forme fonctionnelle
    - Les objets manipulés ne sont pas modifiés
    - Avant



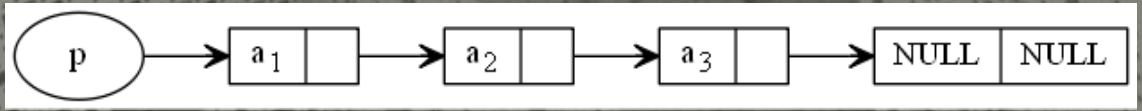
    - Après

```
public Liste<E> addTete(E v) {
    return  new Liste<E> (v,this) ;
}
```



Liste Chainée

- Ajout d'un élément (en tête)
  - Soit sous forme fonctionnelle
    - Les objets manipulés ne sont pas modifiés
    - Avant

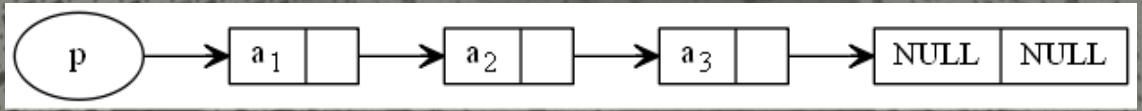


    - Après

```
public Liste<E> addTete(E v) {
    return new Liste<E> (v,this) ;
}
```

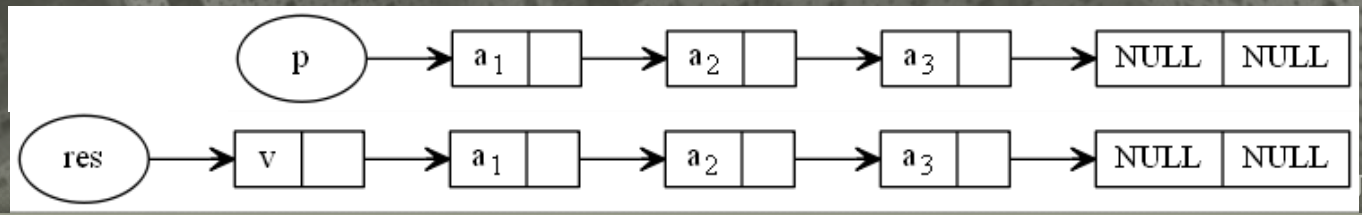
Liste Chainée

- **Ajout d'un élément (en tête)**
  - Soit sous forme fonctionnelle
    - Les objets manipulés ne sont pas modifiés
    - Avant



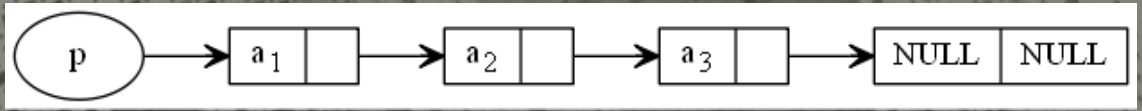
    - Après

```
public Liste<E> addTete(E v) {
    return new Liste<E> (v,this) ;
}
```



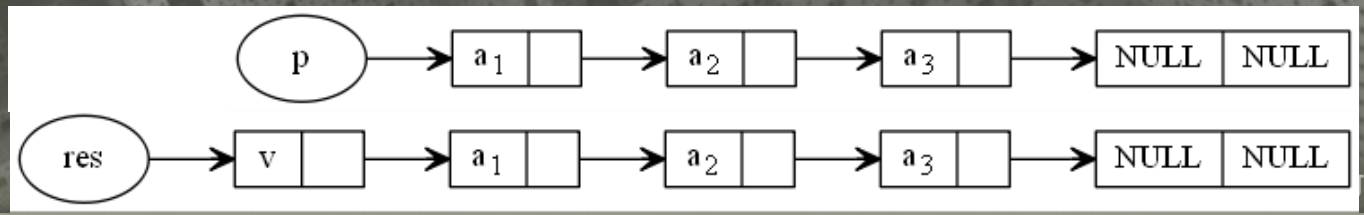
Liste Chainée

- **Ajout d'un élément (en tête)**
  - Soit sous forme fonctionnelle
    - Les objets manipulés ne sont pas modifiés
    - Avant



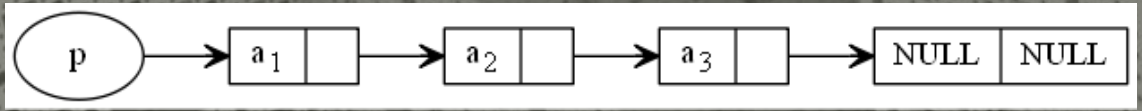
    - Après

```
Function ajouteTete(l : Liste, v : E) : Liste
  In      : l, v
  Do      : Renvoie une nouvelle liste identique à l avec v ajouter en tête
1 begin
2   if estVide (l) then
3     return cons (v, vide ())
```



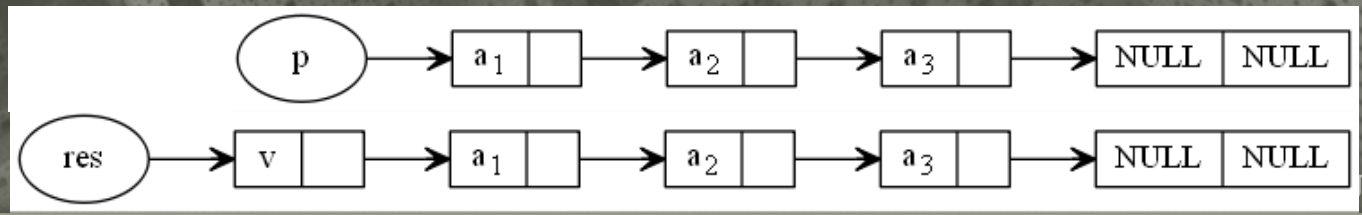
Liste Chainée

- **Ajout d'un élément (en tête)**
  - Soit sous forme fonctionnelle
    - Les objets manipulés ne sont pas modifiés
    - Avant



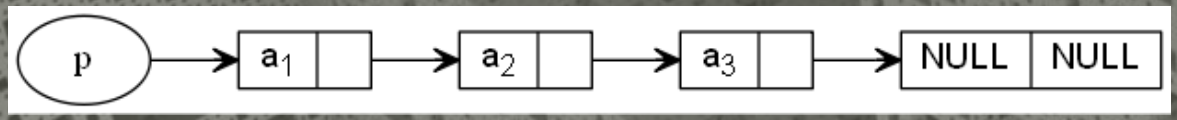
    - Après

```
Function ajouteTete(l : Liste, v : E) : Liste
  In      : l, v
  Do      : Renvoie une nouvelle liste identique à l avec v ajouter en tête
1 begin
2   if estVide (l) then
3     return cons (v, vide ())
4   else
5     return cons (v, ajouteTete (queue (l),tete (l)))
```

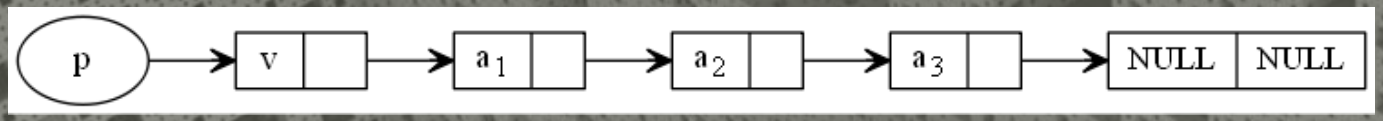


Liste Chainée

- Ajout d'un élément (en tête)
  - Soit sous forme procédurale
    - les objets manipulés sont modifiés
    - Avant

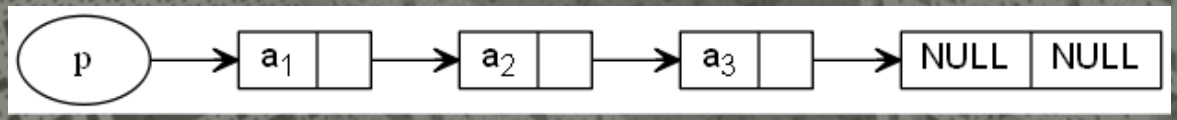


    - Après

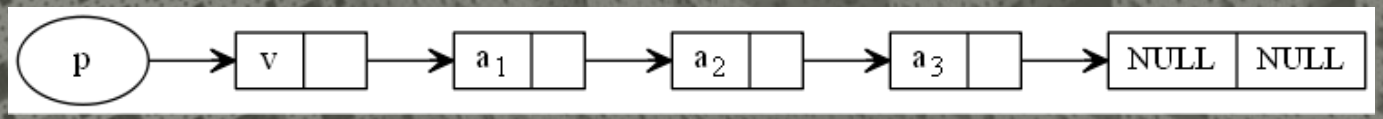


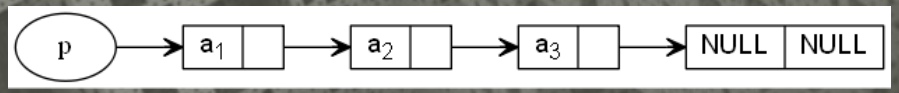
Liste Chainée

- Ajout d'un élément (en tête)
  - Soit sous forme procédurale
    - les objets manipulés sont modifiés
    - Avant



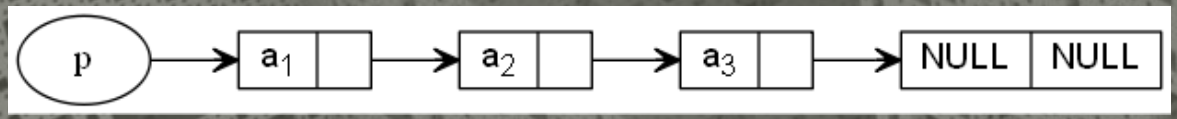
    - Après



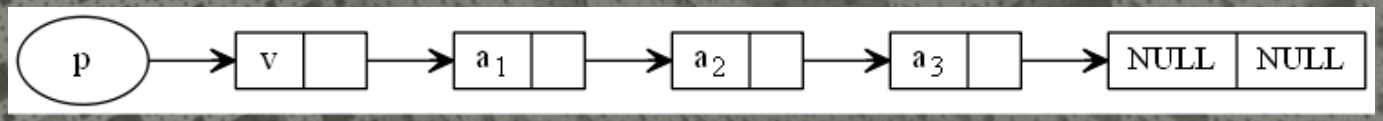


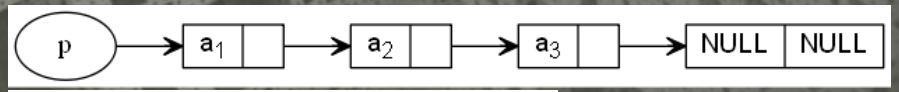
Liste Chainée

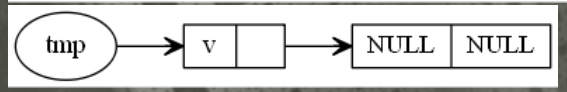
- **Ajout d'un élément (en tête)**
  - Soit sous forme procédurale
    - les objets manipulés sont modifiés
    - Avant



    - Après

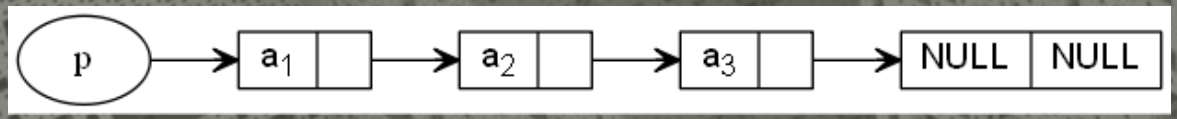




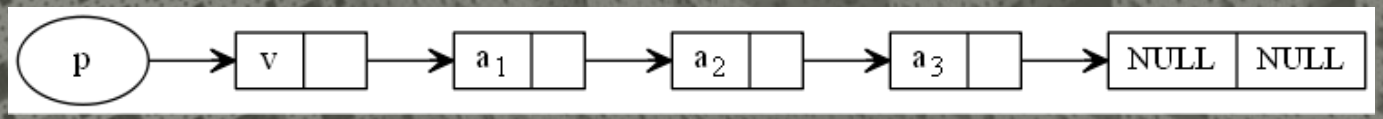


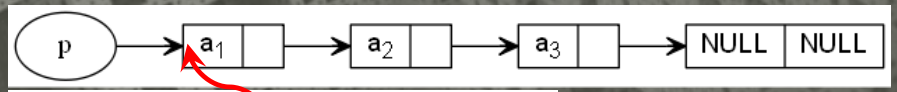
Liste Chainée

- Ajout d'un élément (en tête)
  - Soit sous forme procédurale
    - les objets manipulés sont modifiés
    - Avant



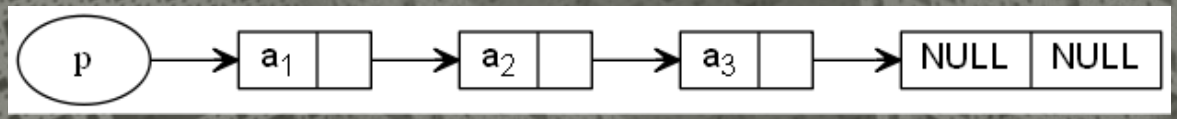
    - Après



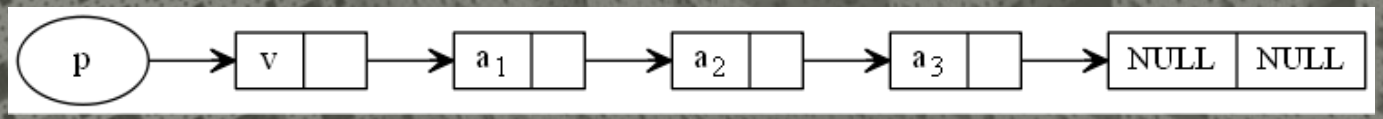


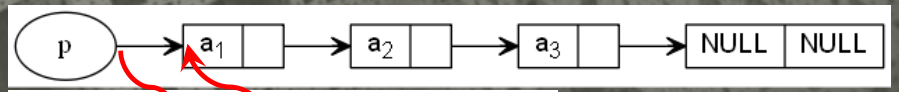
Liste Chainée

- Ajout d'un élément (en tête)
  - Soit sous forme procédurale
    - les objets manipulés sont modifiés
    - Avant



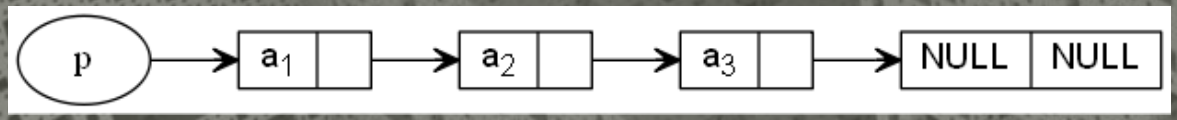
    - Après



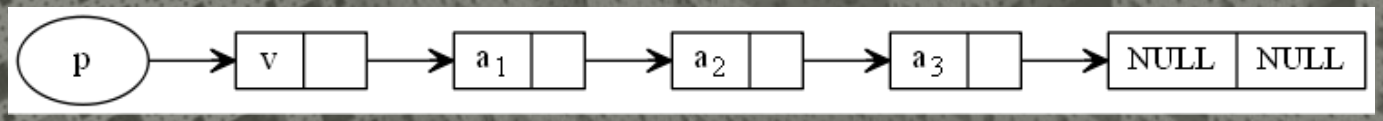


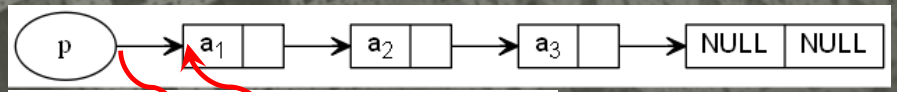
Liste Chainée

- **Ajout d'un élément (en tête)**
  - Soit sous forme procédurale
    - les objets manipulés sont modifiés
    - Avant



    - Après

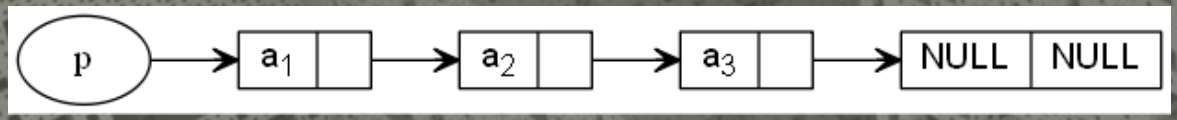




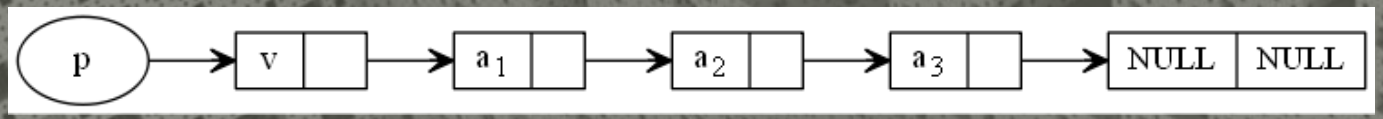

```
public void addTail_Imp(E v) {
    Liste<E> tmp= new Liste<E>(v,new Liste<E>()) ;
    tmp.queue=this ;
    this=tmp ;
}
```

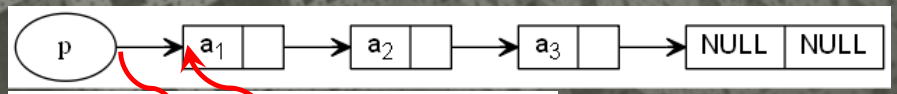
Liste Chainée

- Ajout d'un élément (en tête)
  - Soit sous forme procédurale
    - les objets manipulés sont modifiés
    - Avant



    - Après

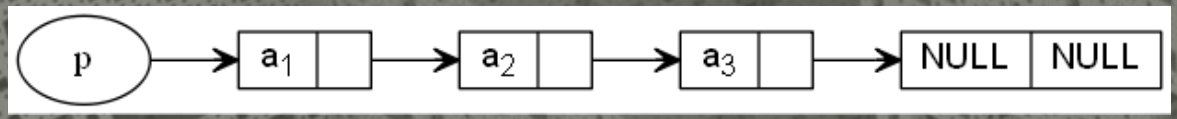




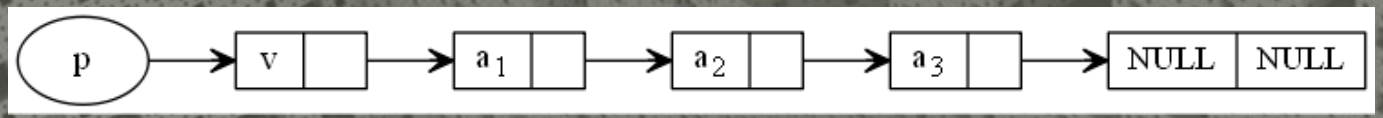

```
public void addTail_Imp(E v) {
    Liste<E> tmp= new Liste<E>(v,new Liste<E>()) ;
    tmp.queue=this ;
    this=tmp ;
}
```

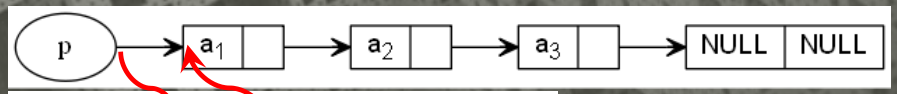
Liste Chainée

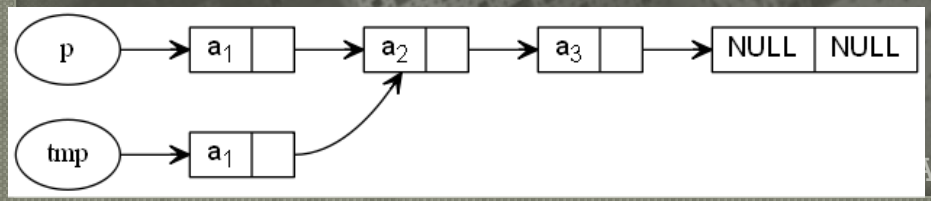
- **Ajout d'un élément (en tête)**
  - Soit sous forme procédurale
    - les objets manipulés sont modifiés
    - Avant



    - Après



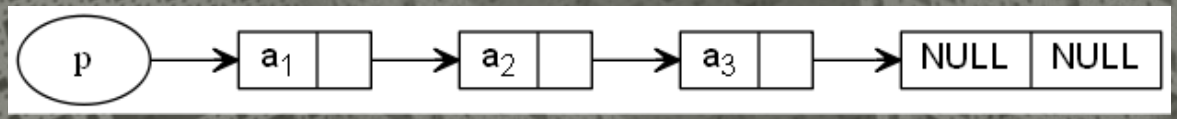




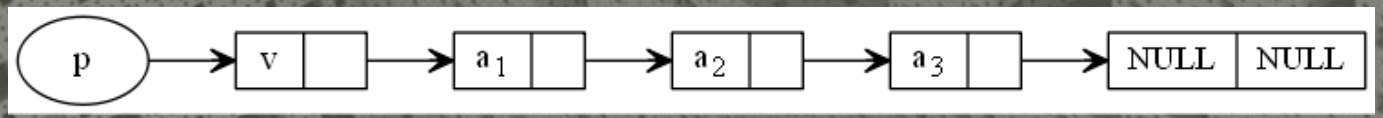

```
public void addTail_Imp(E v) {
    Liste<E> tmp= new Liste<E>(v,new Liste<E>()) ;
    tmp.queue=this ;
    this=tmp ;
}
```

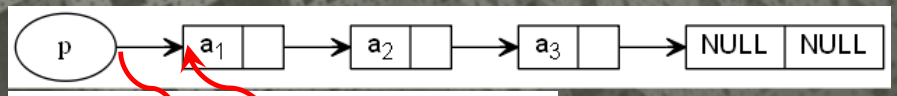
Liste Chainée

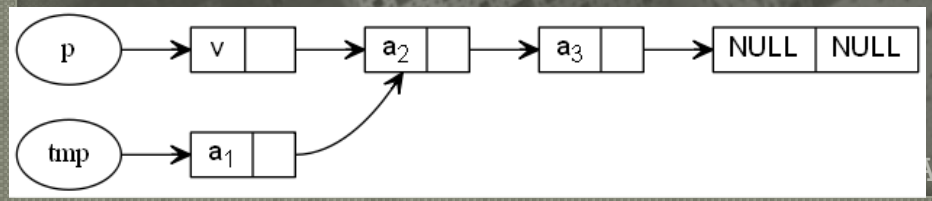
- **Ajout d'un élément (en tête)**
  - Soit sous forme procédurale
    - les objets manipulés sont modifiés
    - Avant



    - Après



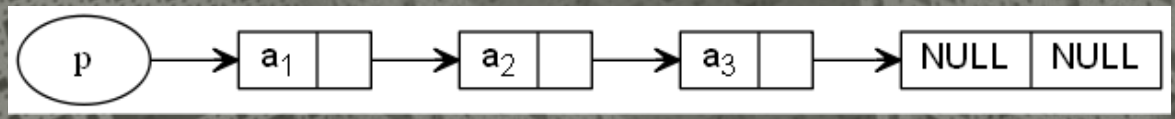




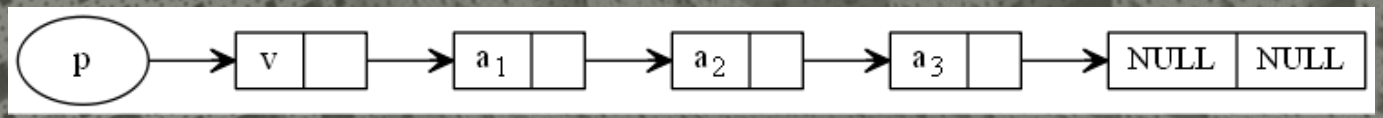

```
public void addTail_Imp(E v) {
    Liste<E> tmp= new Liste<E>(v,new Liste<E>()) ;
    tmp.queue=this ;
    this=tmp ;
}
```

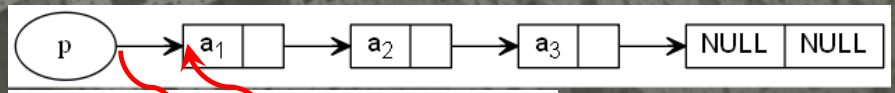
Liste Chainée

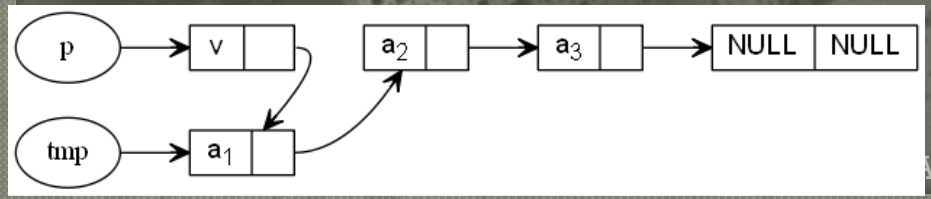
- **Ajout d'un élément (en tête)**
  - Soit sous forme procédurale
    - les objets manipulés sont modifiés
    - Avant



    - Après



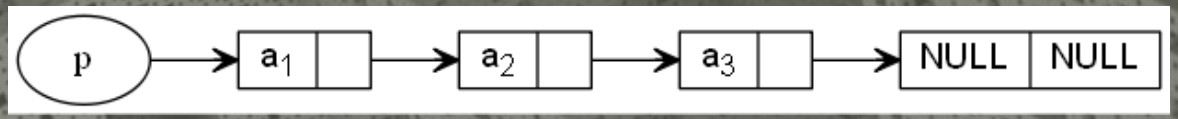





```
public void addTail_Imp(E v) {
    Liste<E> tmp= new Liste<E>(v,new Liste<E>()) ;
    tmp.queue=this ;
    this=tmp ;
}
```

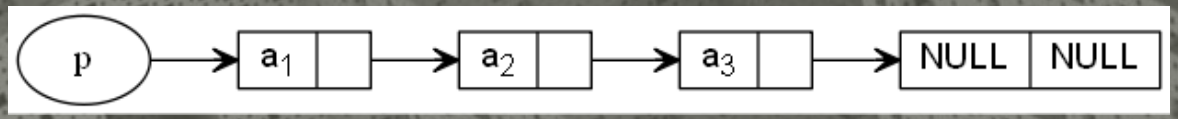
Liste Chainée

- **Ajout d'un élément (en queue)**
  - Sous forme fonctionnelle
    - Avant

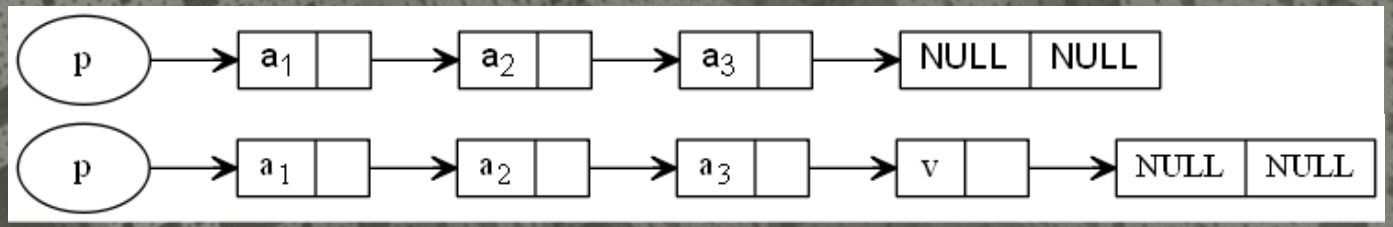


Liste Chainée

- **Ajout d'un élément (en queue)**
  - Sous forme fonctionnelle
    - Avant

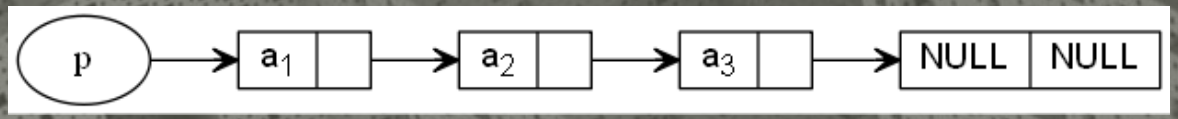


    - Après

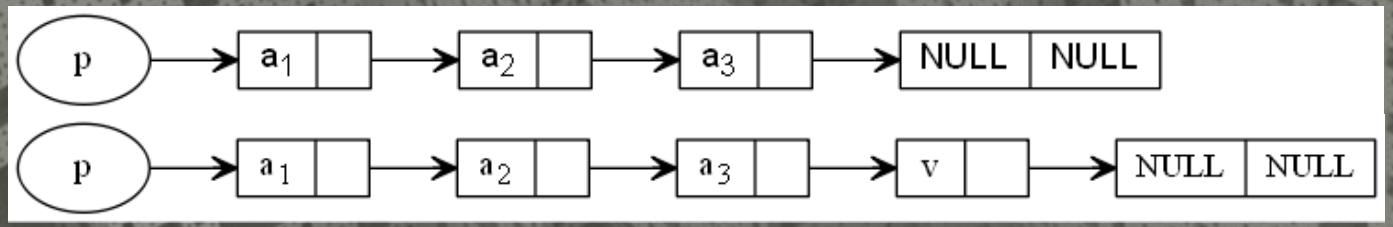


Liste Chainée

- **Ajout d'un élément (en queue)**
  - Sous forme fonctionnelle
    - Avant



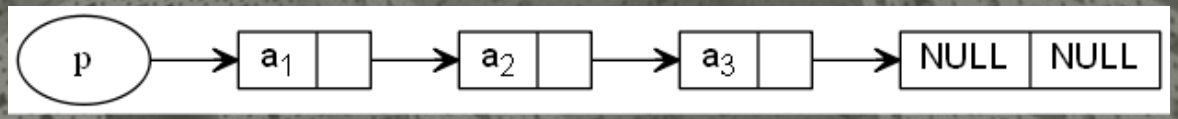
    - Après



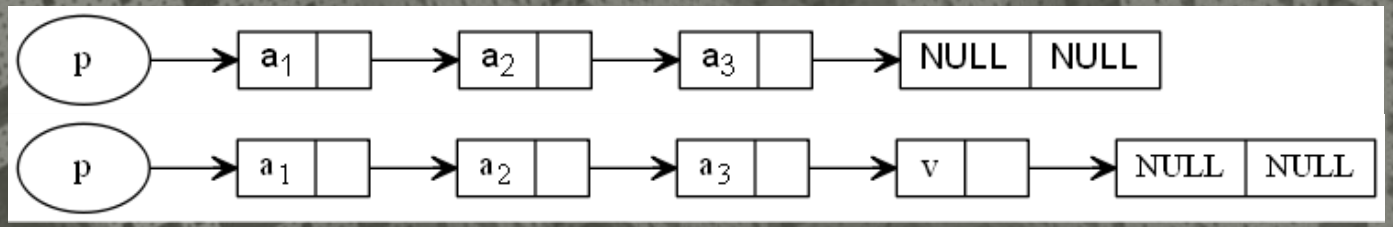
    - Recopie de la liste pendant le parcours

Liste Chainée

- **Ajout d'un élément (en queue)**
  - Sous forme fonctionnelle
    - Avant



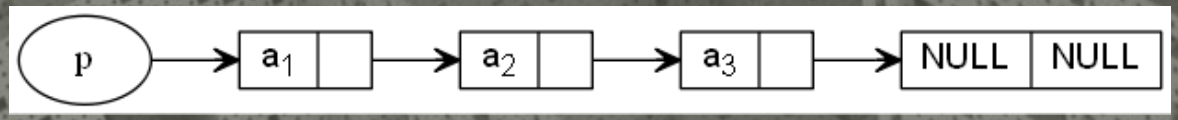
    - Après



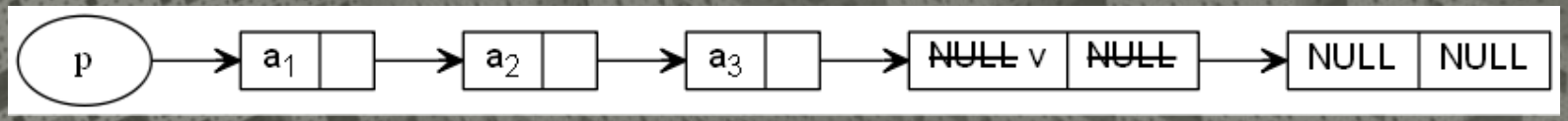
    - Recopie de la liste pendant le parcours
    - ⇒ new Liste(…., …)

Liste Chainée

- **Ajout d'un élément (en queue)**
  - Sous forme procédurale
    - Avant

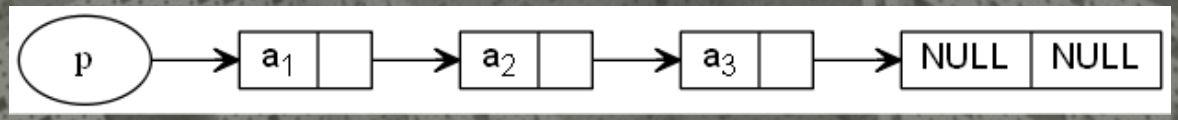


    - Après

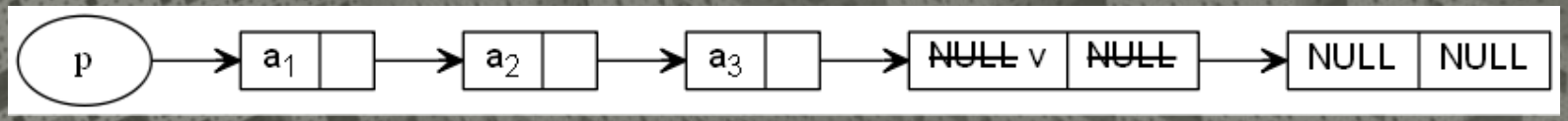


Liste Chainée

- **Ajout d'un élément (en queue)**
  - Sous forme procédurale
    - Avant



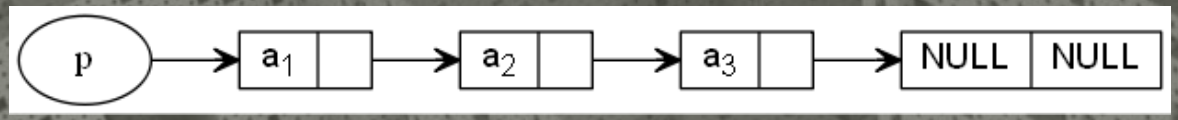
    - Après



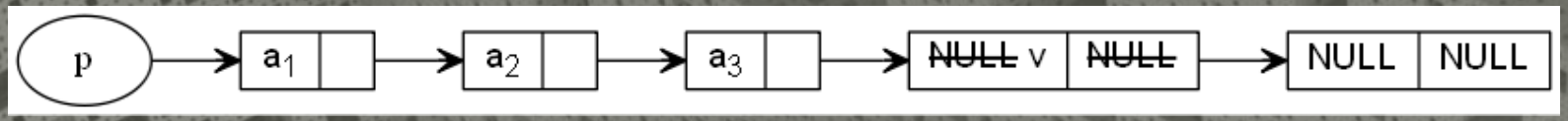

```
Procedure ajouteQueueProc(l : Liste, v : E)
  In     : v
  In/Out: l
  Do     : Ajoute en queue v dans l de manière procédurale
1 begin
2   if estVide (l) then
3     l.tete ← v
4     l.queue ← vide()
```

Liste Chainée

- **Ajout d'un élément (en queue)**
  - Sous forme procédurale
    - Avant



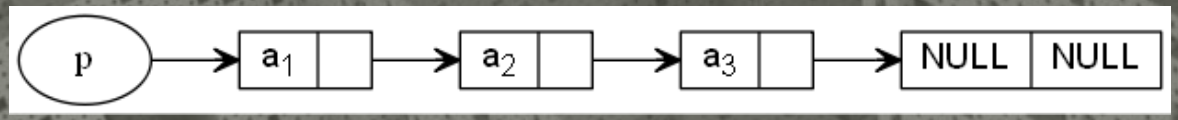
    - Après



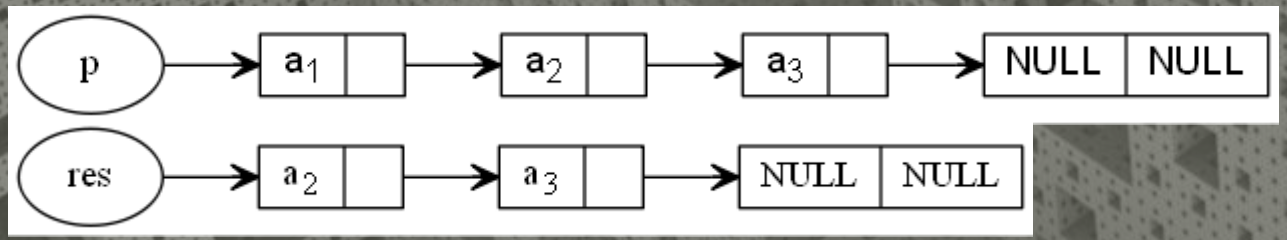

```
Procedure ajouteQueueProc(l : Liste, v : E)
  In     : v
  In/Out: l
  Do     : Ajoute en queue v dans l de manière procédurale
1 begin
2   if estVide (l) then
3     l.tete ← v
4     l.queue ← vide()
5   else
6     ajouteQueueProc (queue (l),v)
```

Liste Chainée

- **Suppression en tête d'un élément**
  - Sous forme fonctionnelle
    - Avant

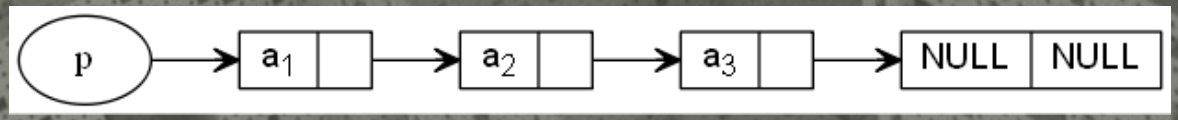


    - Après

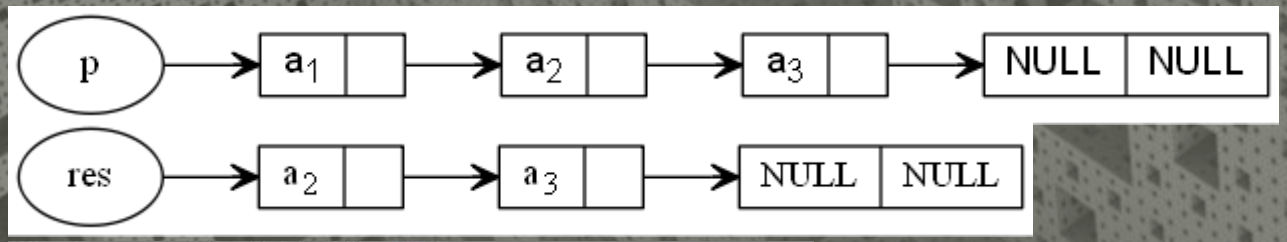


Liste Chainée

- **Suppression en tête d'un élément**
  - Sous forme fonctionnelle
    - Avant



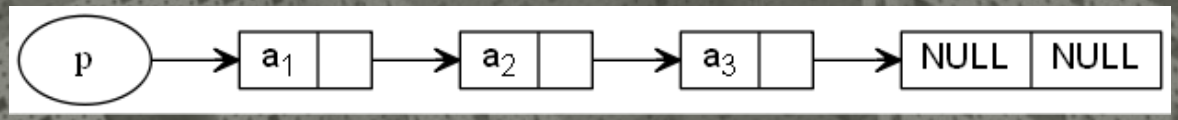
    - Après



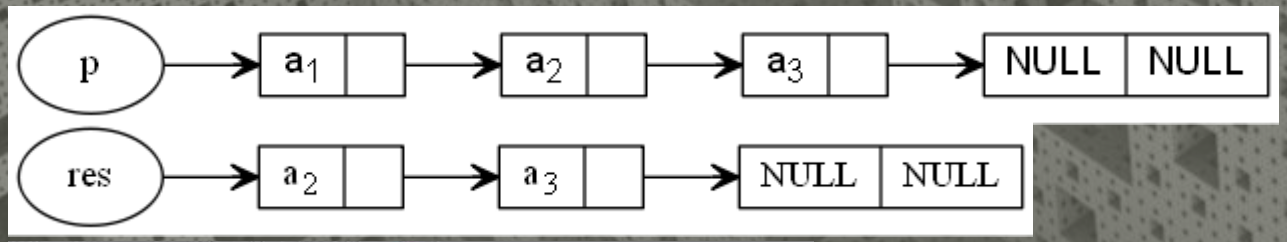

```
Function suppTete(l : Liste) : Liste
  In      : l
  Do      : Supprime la tête de l de manière fonctionnelle
1 begin
2   if estVide (l) then
3     return vide ()
```

Liste Chainée

- **Suppression en tête d'un élément**
  - Sous forme fonctionnelle
    - Avant



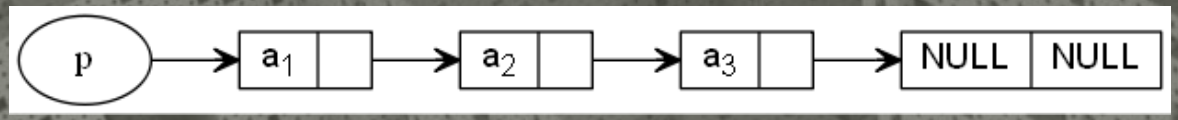
    - Après



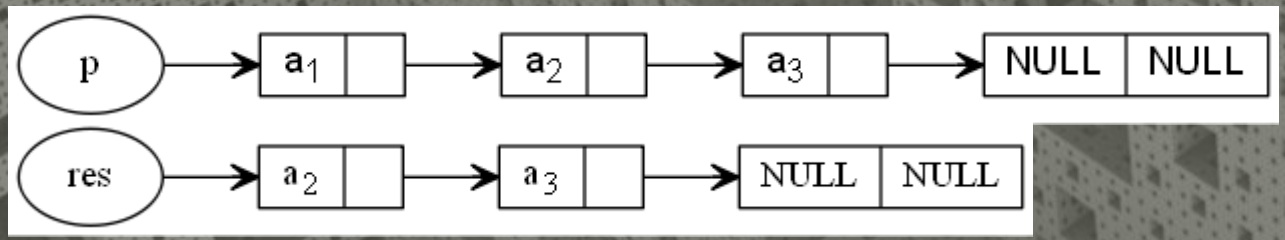

```
Function suppTete(l : Liste) : Liste
  In      : l
  Do      : Supprime la tête de l de manière fonctionnelle
1 begin
2   if estVide (l) then
3     return vide ()
4   else if estVide (queue (l)) then
5     return vide ()
```

Liste Chainée

- **Suppression en tête d'un élément**
  - Sous forme fonctionnelle
    - Avant



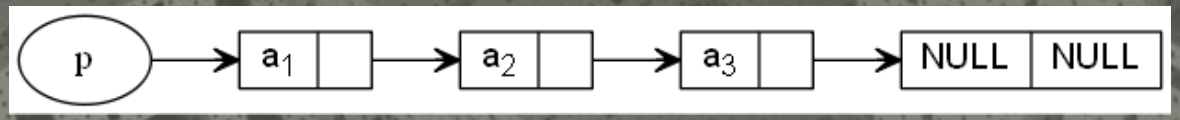
    - Après



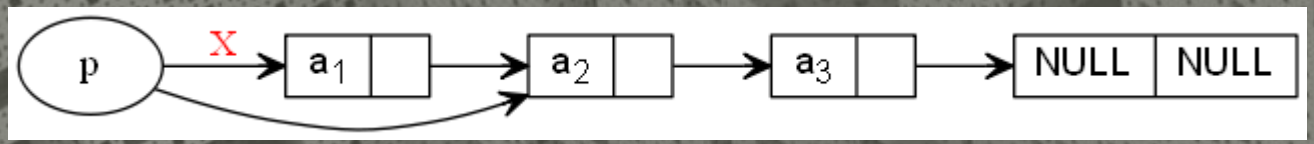

```
Function suppTete(l : Liste) : Liste
  In      : l
  Do      : Supprime la tête de l de manière fonctionnelle
1 begin
2   if estVide (l) then
3     return vide ()
4   else if estVide (queue (l)) then
5     return vide ()
6   else
7     return ajouteTete (queue (queue (l)) , tete (queue (l)))
```

Liste Chainée

- **Suppression en tête d'un élément**
  - Sous forme procédurale
    - Avant

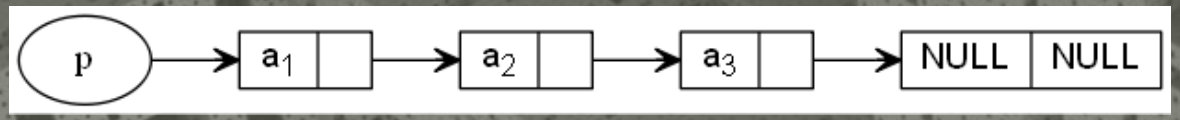


    - Après
      - Idéalement

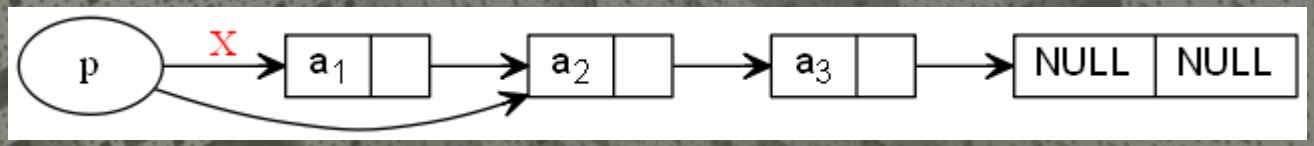


Liste Chainée

- **Suppression en tête d'un élément**
  - Sous forme procédurale
    - Avant



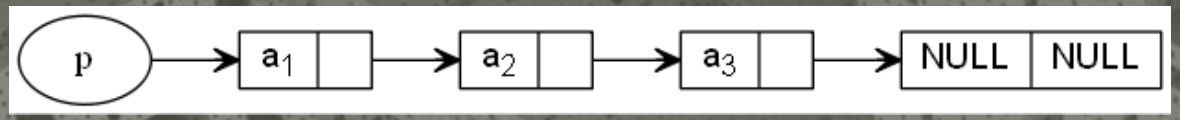
    - Après
      - Idéalement



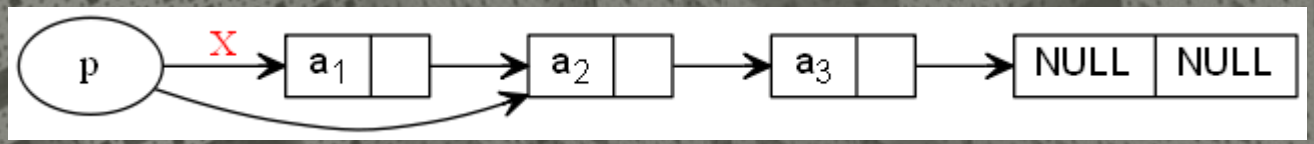
      - Impossible car implique modification de `this`

Liste Chainée

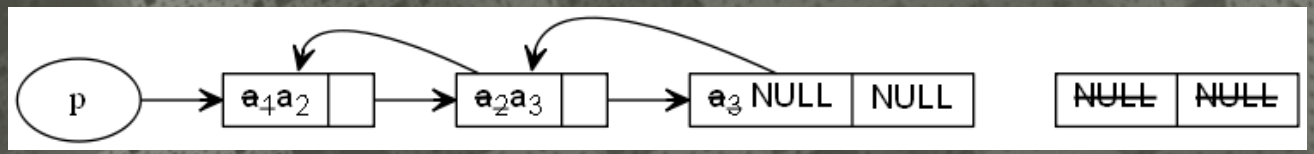
- **Suppression en tête d'un élément**
  - Sous forme procédurale
    - Avant



    - Après
      - Idéalement

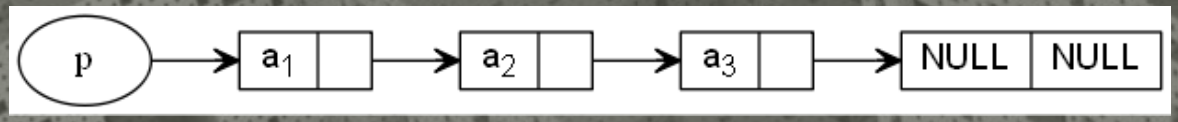


      - Impossible car implique modification de `this`



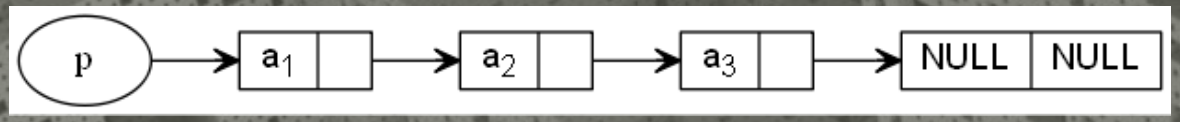
Liste Chainée

- **Suppression en queue d'un élément**
  - Sous forme fonctionnelle
    - Avant

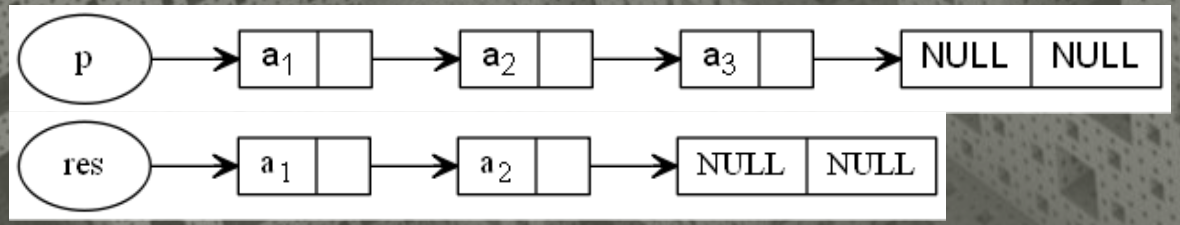


Liste Chainée

- **Suppression en queue d'un élément**
  - Sous forme fonctionnelle
    - Avant

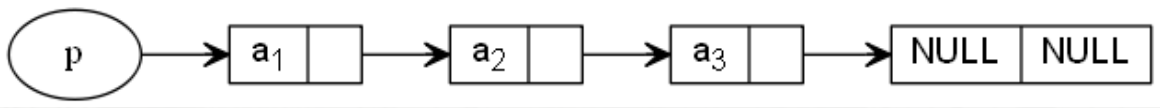


    - Après



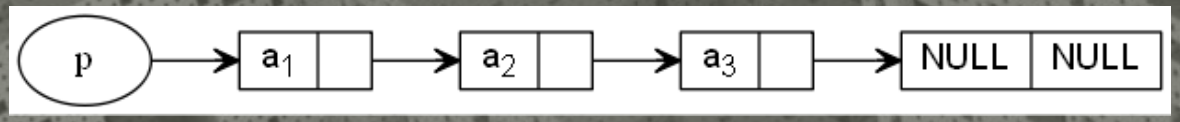
Liste Chainée

- **Suppression en queue d'un élément**
  - Sous forme procédurale
    - Avant

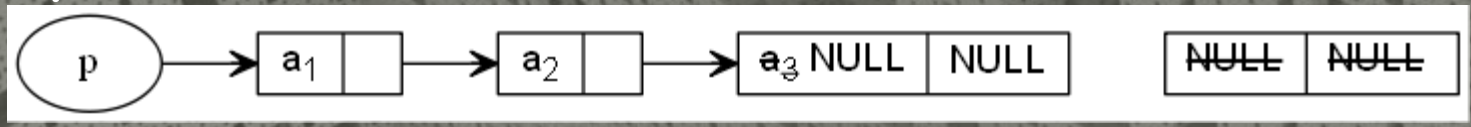


Liste Chainée

- **Suppression en queue d'un élément**
  - Sous forme procédurale
    - Avant

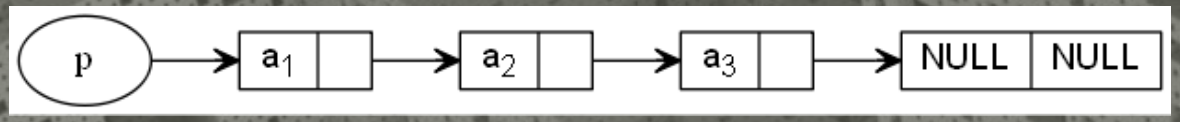


    - Après

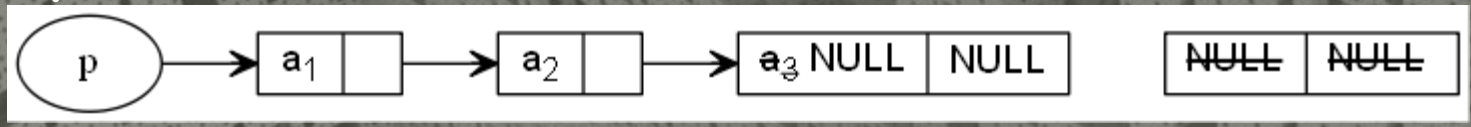


Liste Chainée

- **Suppression en queue d'un élément**
  - Sous forme procédurale
    - Avant



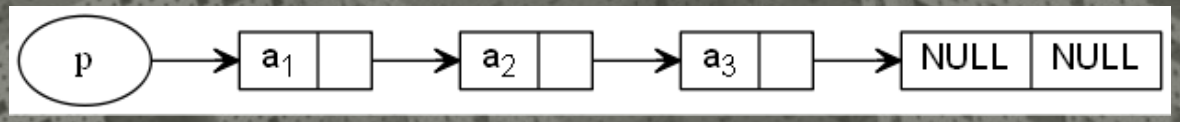
    - Après



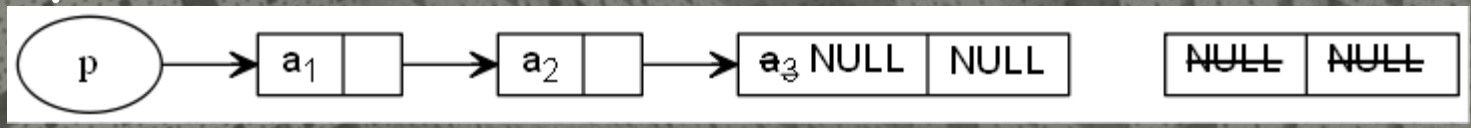

```
Procedure suppQueueProc(l : Liste)
  In/Out: l
  Do      : Supprime en queue dans l de manière procédurale
1 begin
2 |  if estVide (l) then ∅
```

Liste Chainée

- **Suppression en queue d'un élément**
  - Sous forme procédurale
    - Avant



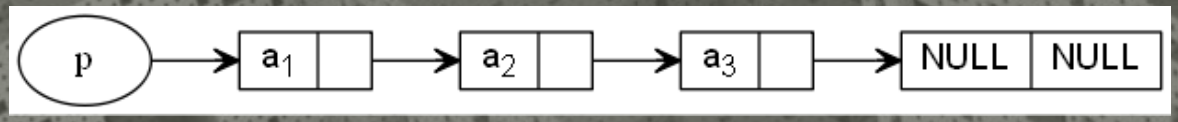
    - Après



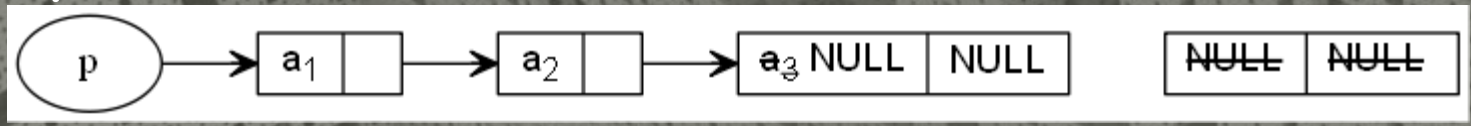

```
Procedure suppQueueProc(l : Liste)
  In/Out: l
  Do      : Supprime en queue dans l de manière procédurale
1 begin
2   if estVide (l) then ∅
3   else if estVide (queue (l)) then
4     l.tete ← null
5     l.queue ← null
```

Liste Chainée

- **Suppression en queue d'un élément**
  - Sous forme procédurale
    - Avant



    - Après




```
Procedure suppQueueProc(l : Liste)
  In/Out: l
  Do      : Supprime en queue dans l de manière procédurale
1 begin
2     if estVide (l) then ∅
3     else if estVide (queue (l)) then
4         l.tete ← null
5         l.queue ← null
6     else
7         suppQueueProc (queue (l))
```

Inconvénients Liste Chainée

- Insertion en queue compliqué

Inconvénients Liste Chainée

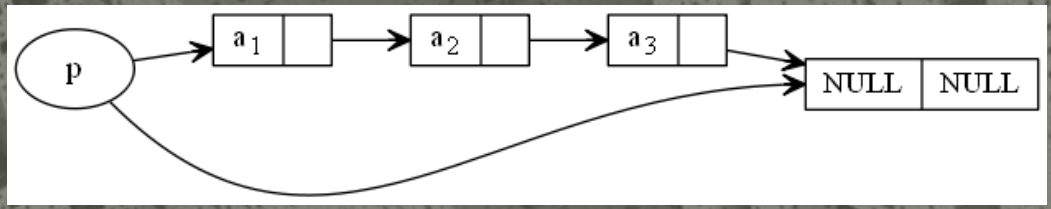
- **Insertion en queue compliqué**
  - **Obliger de parcourir toute la liste pour insérer**

Inconvénients Liste Chainée

- **Insertion en queue compliqué**
  - Obliger de parcourir toute la liste pour insérer
  - $\Rightarrow$ Ajout d'une référence sur le dernier élément

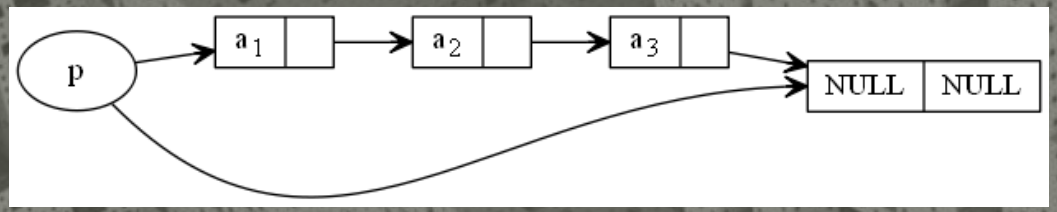
Inconvénients Liste Chainée

- **Insertion en queue compliqué**
  - Obliger de parcourir toute la liste pour insérer
  - $\Rightarrow$ Ajout d'une référence sur le dernier élément



Inconvénients Liste Chainée

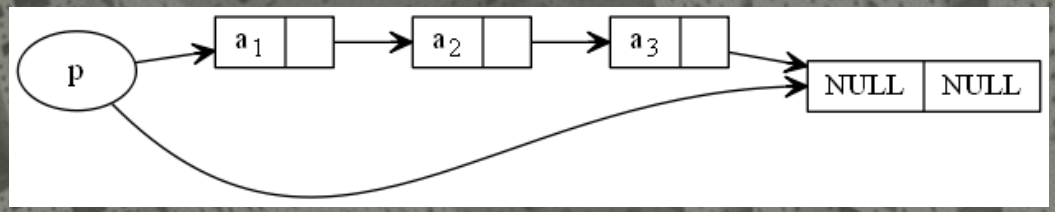
- Insertion en queue compliqué
  - Obliger de parcourir toute la liste pour insérer
  - $\Rightarrow$ Ajout d'une référence sur le dernier élément



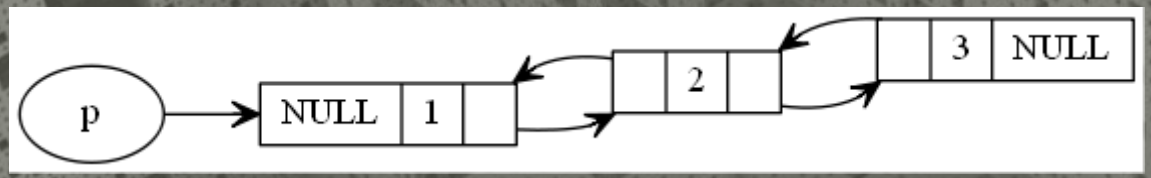
- Impossible d'aller au précédent

Inconvénients Liste Chainée

- Insertion en queue compliqué
  - Obliger de parcourir toute la liste pour insérer
  - ⇒ Ajout d'une référence sur le dernier élément



- Impossible d'aller au précédent
  - ⇒ Liste Doublement Chainée (LinkedList en java)



java.util

Interface List<E>

All Superinterfaces:

Collection<E>, Iterable<E>

boolean	`add(E e)` Appends the specified element to the end of this list (optional operation).
void	`add(int index, E element)` Inserts the specified element at the specified position in this list (optional operation).
boolean	`addAll(Collection<? extends E> c)` Appends all of the elements in the specified collection to the end of this list, in the order that they are returned by the specified collection's iterator (optional operation).
boolean	`addAll(int index, Collection<? extends E> c)` Inserts all of the elements in the specified collection into this list at the specified position (optional operation).
void	`clear()` Removes all of the elements from this list (optional operation).
boolean	`contains(Object o)` Returns `true` if this list contains the specified element.
boolean	`containsAll(Collection<?> c)` Returns `true` if this list contains all of the elements of the specified collection.
boolean	`equals(Object o)` Compares the specified object with this list for equality.
E	`get(int index)` Returns the element at the specified position in this list.
int	`hashCode()` Returns the hash code value for this list.
int	`indexOf(Object o)` Returns the index of the first occurrence of the specified element in this list, or -1 if this list does not contain the element.
boolean	`isEmpty()` Returns `true` if this list contains no elements.
Iterator<E>	`iterator()` Returns an iterator over the elements in this list in proper sequence.
int	`lastIndexOf(Object o)` Returns the index of the last occurrence of the specified element in this list, or -1 if this list does not contain the element.
ListIterator<E>	`listIterator()` Returns a list iterator over the elements in this list (in proper sequence).
ListIterator<E>	`listIterator(int index)` Returns a list iterator over the elements in this list (in proper sequence), starting at the specified position in the list.
E	`remove(int index)` Removes the element at the specified position in this list (optional operation).
boolean	`remove(Object o)` Removes the first occurrence of the specified element from this list, if it is present (optional operation).
boolean	`removeAll(Collection<?> c)` Removes from this list all of its elements that are contained in the specified collection (optional operation).
boolean	`retainAll(Collection<?> c)` Retains only the elements in this list that are contained in the specified collection (optional operation).
E	`set(int index, E element)` Replaces the element at the specified position in this list with the specified element (optional operation).
int	`size()` Returns the number of elements in this list.
List<E>	`subList(int fromIndex, int toIndex)` Returns a view of the portion of this list between the specified `fromIndex`, inclusive, and `toIndex`, exclusive.
Object[]	`toArray()` Returns an array containing all of the elements in this list in proper sequence (from first to last element).
<T> T[]	`toArray(T[] a)` Returns an array containing all of the elements in this list in proper sequence (from first to last element); the runtime type of the returned array is that of the specified array.

Collection Java

java.util

Interface Collection<E>

Type Parameters:

E - the type of elements in this collection

All Superinterfaces:

Iterable<E>

All Known Subinterfaces:

BeanContext, BeanContextServices, BlockingDeque<E>, BlockingQueue<E>, Deque<E>, List<E>, NavigableSet<E>, Queue<E>, Set<E>, SortedSet<E>, TransferQueue<E>

All Known Implementing Classes:

AbstractCollection, AbstractList, AbstractQueue, AbstractSequentialList, AbstractSet, ArrayBlockingQueue, ArrayDeque, ArrayList, AttributeList, BeanContextServicesSupport, BeanContextSupport, ConcurrentLinkedDeque, ConcurrentLinkedQueue, ConcurrentSkipListSet, CopyOnWriteArrayList, CopyOnWriteArraySet, DelayQueue, EnumSet, HashSet, JobStateReasons, LinkedBlockingDeque, LinkedBlockingQueue, LinkedHashSet, LinkedList, LinkedTransferQueue, PriorityBlockingQueue, PriorityQueue, RoleList, RoleUnresolvedList, Stack, SynchronousQueue, TreeSet, Vector

Collection Java

java.util

Interface Collection<E>

Methods

Modifier and Type	Method and Description
boolean	`add(E e)` Ensures that this collection contains the specified element (optional operation).
boolean	`addAll(Collection<? extends E> c)` Adds all of the elements in the specified collection to this collection (optional operation).
void	`clear()` Removes all of the elements from this collection (optional operation).
boolean	`contains(Object o)` Returns `true` if this collection contains the specified element.
boolean	`containsAll(Collection<?> c)` Returns `true` if this collection contains all of the elements in the specified collection.
boolean	`equals(Object o)` Compares the specified object with this collection for equality.
int	`hashCode()` Returns the hash code value for this collection.
boolean	`isEmpty()` Returns `true` if this collection contains no elements.
Iterator<E>	`iterator()` Returns an iterator over the elements in this collection.

Collection Java

java.util

Interface Collection<E>

Methods

Modifier and Type	Method and Description
boolean	`add(E e)` Ensures that this collection contains the specified element (optional operation).
boolean	`addAll(Collection<? extends E> c)` Adds all of the elements in the specified collection to this collection (optional operation).
void	`clear()` Removes all of the elements from this collection (optional operation).
boolean	`contains(Object o)` Returns `true` if this collection contains the specified element.
boolean	`containsAll(Collection<?> c)` Returns `true` if this collection contains all of the elements in the specified collection.
boolean	`equals(Object o)` Compares the specified object with this collection for equality.
int	`hashCode()` Returns the hash code value for this collection.
boolean	`isEmpty()` Returns `true` if this collection contains no elements.
Iterator<E>	`iterator()` Returns an iterator over the elements in this collection.
boolean	`remove(Object o)` Removes a single instance of the specified element from this collection, if it is present (optional operation).
boolean	`removeAll(Collection<?> c)` Removes all of this collection's elements that are also contained in the specified collection (optional operation).
boolean	`retainAll(Collection<?> c)` Retains only the elements in this collection that are contained in the specified collection (optional operation).
int	`size()` Returns the number of elements in this collection.
Object[]	`toArray()` Returns an array containing all of the elements in this collection.
<T> T[]	`toArray(T[] a)` Returns an array containing all of the elements in this collection; the runtime type of the returned array is that of the specified array.

TAD Ensemble

- **Ensemble**
  - collection d'éléments où chaque élément ne peut apparaitre qu'une seule fois
  - Set en java
  - Implémentation possibles
    - Tableaux de booléens
    - Tableaux ou liste d'éléments
    - …
    - Axiomes
      - EstDans(x, Supprime(x,e)) = Faux

Sources

- Florent Hivert, Algorithmique
- Elise Bouzon, Algo et Structures récursives