Penser récursivement, penser par objets

HAI8403I
Modélisation et Programmation par Objets

Université de Montpellier

Plan

Introduction à la récursivité

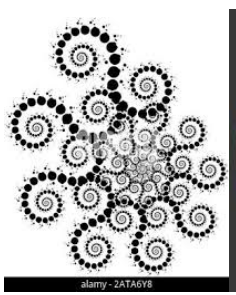
- Une expression récursive est une expression intégrant une référence à elle-même
- Un algorithme récursif est un algorithme qui s'appelle lui-même

Owarimonogatari First, Ougi Formula, dir. Tomoyuki Itamura, Akiyuki Shimbô (as Akiyuki Shinbo), 2015, photogramme

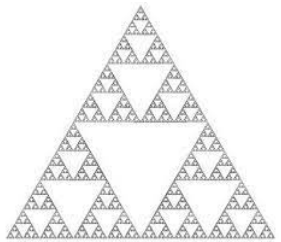
Introduction à la récursivité

Visions apparentées

- une phrase qui inclut un groupe nominal, un verbe et une autre phrase
- courbes fractales
- mises en abîme dans le domaine de l'art ou du graphisme
- certains motifs biologiques (fleur de tournesol, coquilles de nautile)
- définitions de concepts telles que : "un troupeau de moutons est vide ou bien un mouton ajouté à un troupeau de moutons".
- définitions de fonctions mathématiques dans leurs propres termes, ou de suites mathématiques définies par récurrence.







Introduction à la récursivité

Définitions de fonctions mathématiques dans leurs propres termes

Fonction factorielle définie dans ses propres termes :

$$\begin{aligned} 0! &= 1 \\ \textit{pour } n > 0, \quad n! &= n * (n-1)! \end{aligned} \tag{1}$$

Introduction à la récursivité

Définitions de suites mathématiques définies par récurrence

Suite mathématique (u_n) définie par récurrence :

$$u_0 = 1 \ pour \ n > 0, \quad u_{n+1} = \sqrt{1 + u_n} \tag{2}$$

Récursivité et réduction de problème

Principe de réduction

- Problème sur une donnée d'une certaine taille
- Réduit au même problème sur une donnée de taille inférieure

Récursivité et réduction de problème

Fonction factorielle

Définition non récursive :

$$0\,! = 1$$
$$n\,! \;\; est \;\; le \;\; produit \;\; des \;\; entiers \;\; de \;\; 1 \;\; à \;\; n, \;\; pour \;\; n > 0$$

Cas de base

$$0\,! = 1$$

Cas général : $n!$ se pense en terme d'une factorielle d'un nombre plus petit

$$pour \;\; n > 0, \quad n\,! = n * (n\text{-}1) * (n\text{-}2) * \ldots * 1$$
$$pour \;\; n > 0, \quad n\,! = n * (n\text{-}1)\,!$$

Récursivité en modélisation et en programmation

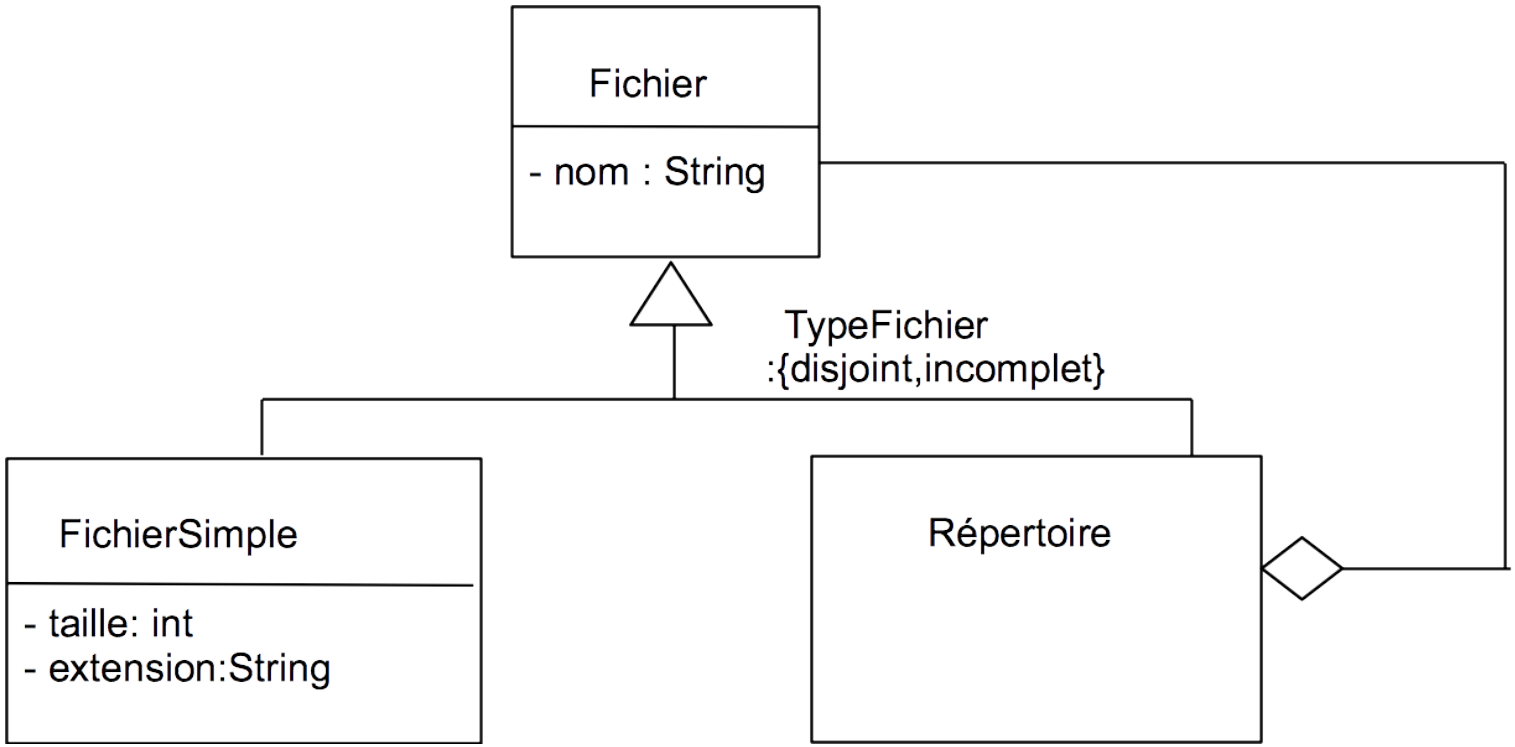
Formulation élégante de certains algorithmes et certaines structures de données dans des cas où leur définition/expression est naturellement récursive :

- Fonction factorielle
- Liste : une liste est un élément suivi d'une sous-liste
- Arbre binaire : un arbre binaire est constitué d'un nœud/élément et de deux sous-arbres

Récursivité en modélisation et en programmation

En UML, on la trouvera fréquemment :

- sous forme de classes munies d'associations (ou d'attributs) ayant comme extrémité (comme type) cette même classe ou une super-classe.



Récursivité en modélisation et en programmation

En Java, on la trouvera :

- pour définir des structures de données
- dans des méthodes statiques
- dans des méthodes d'instances
  - dans un cadre général
  - plus particulièrement sur des structures de données récursives

Plan

Méthodes statiques récursives

Factorielle

Cas de base : si $n = 0$, cela vaut 1

Cas général : si $n > 0$ cela vaut $n * (n - 1)!$

```
public class FonctionsRécursives{

  public static int factorielle(int n)
  {
                if (n==0) return 1;
                else return (n * factorielle(n-1));
  }

  public static void main(String[] args)
  {
        System.out.println("factorielle 0 "+factorielle(0));
        System.out.println("factorielle 4 "+factorielle(4));
  }
}
```

Méthodes statiques récursives

Somme des n premiers entiers
Cas de base : si $n = 0$, cela vaut 0
Cas général : cela vaut n + (la somme des $n-1$ premiers entiers)

```
....
   public static int somme(int n)
   {
      if (n==0) return 0;
      else return (n+somme(n-1));
   }
   public static void main(String[] args)
   {
      System.out.println("somme 0 premiers entiers "+somme(0));
      System.out.println("somme 4 premiers entiers "+somme(4));
   }
}
```

Méthodes statiques récursives

Puissance entière x^n
Cas de base : si $n = 0$, cela vaut 1
Cas général : cela vaut $x * x^{n-1}$

```
....
   public static int puissance(int x, int n){
      if (n==0) return 1;
      else return x * puissance(x, n-1);
   }

   public static void main(String[] args) {
      System.out.println("2 ^0 "+puissance(2,0));
      System.out.println("2 ^3 "+puissance(2,3));
    }
}
```

Plan

Méthodes d'instances récursives

File d'attente de personnes devant un cinéma

```
public class FileAtt {
    private ArrayList<Personne> listePersonnes
                = new ArrayList<Personne>();
    public FileAtt(){}
    public void entre(Personne p)
    {
     if (!  this.listePersonnes.contains(p))
          this.listePersonnes.add(p);
    }
.........}
public class Personne { // représente un spectateur
    private String nom = "nom inconnu";
    private String prenom = "prenom inconnu";
    private int age;
    private String ticket; // titre du film
.........}
```

Méthodes d'instances récursives

Construction d'une file d'attente de spectateurs

```
public static void main(String[] argv){
  Personne p0 = new Personne("Alice","Livre",18,"cendrillon");
  Personne p1 = new Personne("Theo","Laforet",18,"blanche-neige");
  Personne p2 = new Personne("Hector","Dulac",23,"blanche-neige");
  Personne p3 = new Personne("Arthur","Dumoulin",16,"cendrillon");
  Personne p4 = new Personne("Alex","Bosquet",16,"blanche-neige");

  FileAtt f5 = new FileAtt();
  f5.entre(p0); f5.entre(p1);
  f5.entre(p2); f5.entre(p3); f5.entre(p4);

  // methodes que l'on va definir
  System.out.println(f5);
  System.out.println(f5.ageMoyen());
  System.out.println(f5.spectateurFilmRec("blanche-neige"));
}
```

Méthodes d'instances récursives

Méthode itérative de calcul de l'âge moyen

```
public class FileAtt {
    private ArrayList<Personne> listePersonnes
                = new ArrayList<Personne>();

     public int ageMoyen()
    {
       if (this.listePersonnes.isEmpty())
          return 0;
       int somme = 0;
       for (Personne p : this.listePersonnes)
          somme += p.getAge();
       return somme/this.listePersonnes.size();
    }
}
```

Méthodes d'instances récursives

Méthode récursive de calcul de l'âge moyen
On calcule tout d'abord la somme des âges

avec un curseur qui avance sur la liste :

- cas général : la somme est donnée par l'âge de la personne désignée par le curseur auquel on ajoute la somme des âges des personnes qui sont dans la suite de la file d'attente.
- cas de base : tout est examiné, on retourne 0.

Le curseur est un paramètre de la méthode

Méthodes d'instances récursives

Méthode récursive (auxiliaire) de calcul de la somme des âges

```
public int sommeAgeRecAux(int curseur)
 {
    if (curseur==this.listePersonnes.size()) // cas de base
       return 0;
    else // cas general
       return this.listePersonnes.get(curseur).getAge()
              +sommeAgeRecAux(curseur+1);
 }
```

Il faut une méthode qui appelle cette méthode auxiliaire et lance le calcul

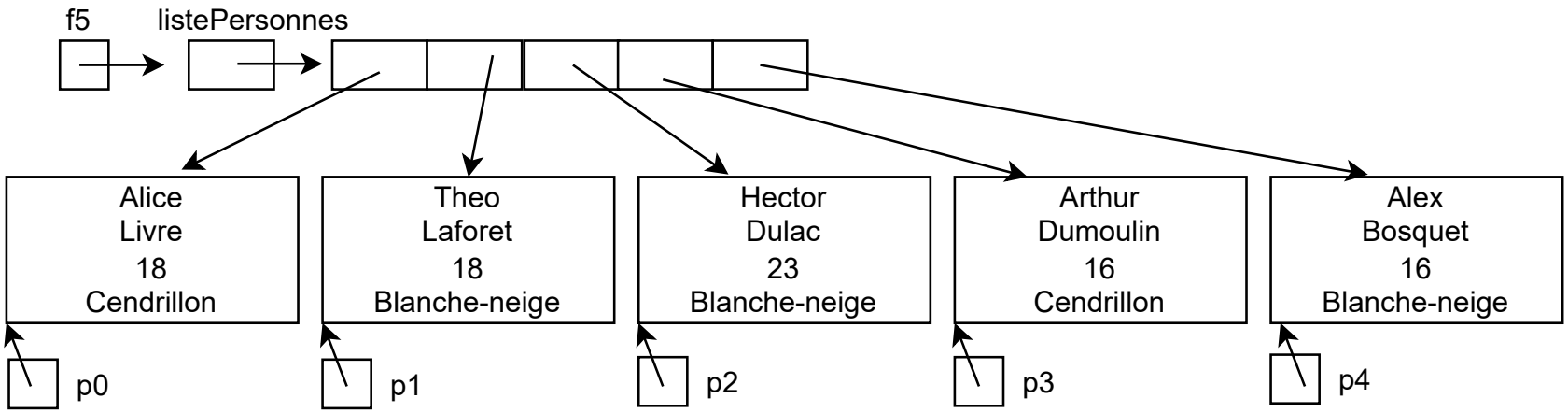
Méthodes d'instances récursives

Méthode principale qui calcule l'âge moyen
appelant la méthode récursive de calcul de la somme des âges

Le curseur vaut 0 lors de l'appel

```
public int ageMoyen()
{
  if (this.listePersonnes.isEmpty()) return 0;

  return this.sommeAgeRecAux(0)/this.listePersonnes.size();
}
```

Méthodes d'instances récursives

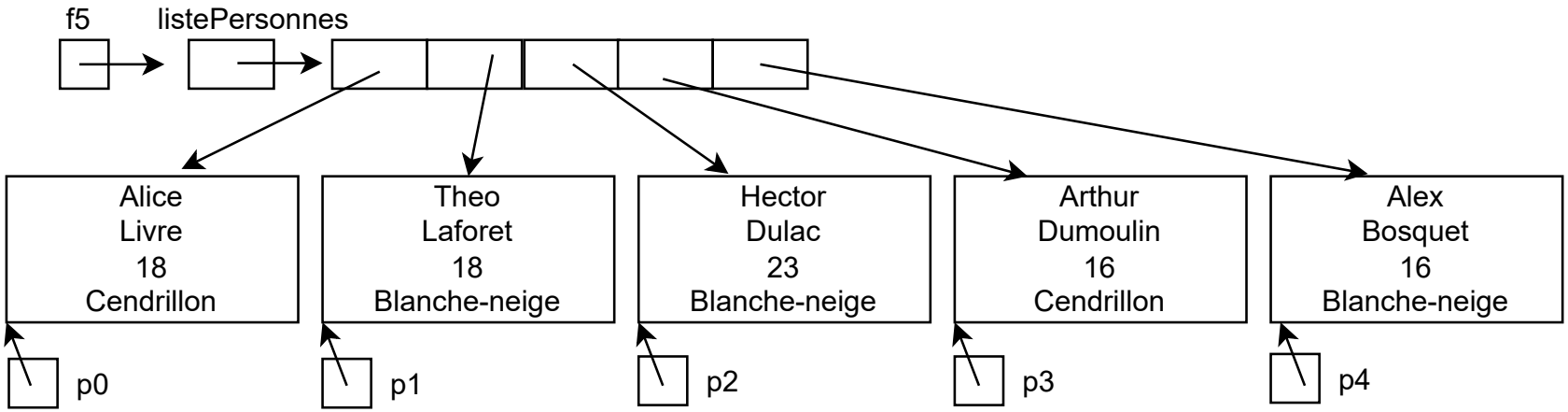


this.sommeAgeRecAuc(0)

= listePersonnes.get(0).getAge()+sommeAgeRecAux(1)

= **18**+sommeAgeRecAux(1)

Méthodes d'instances récursives



this.sommeAgeRecAuc(0)

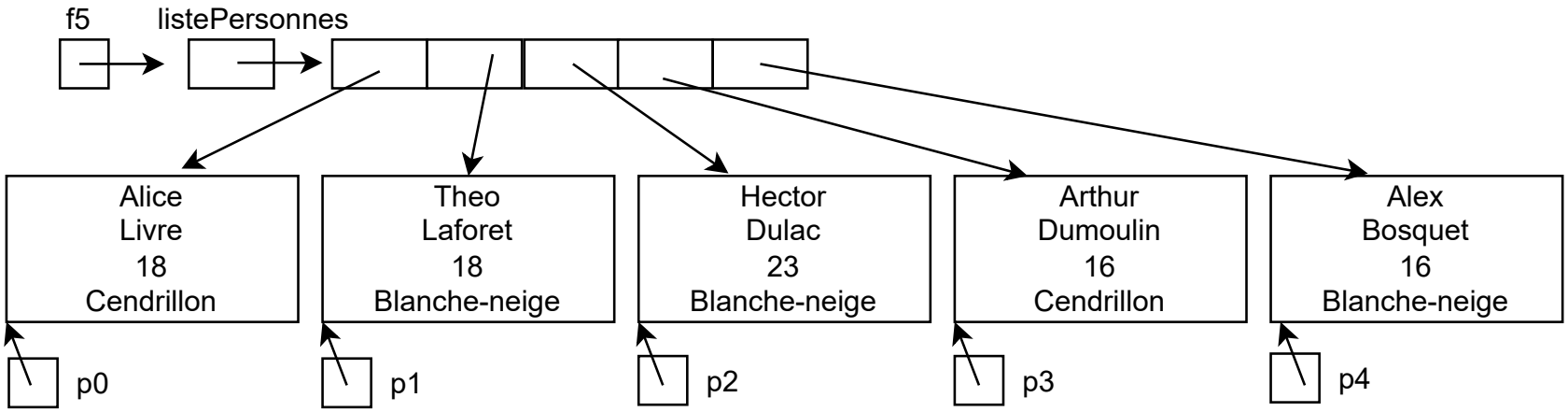
= listePersonnes.get(0).getAge()+sommeAgeRecAux(1)

= **18**+sommeAgeRecAux(1)

= listePersonnes.get(1).getAge()+sommeAgeRecAux(2)

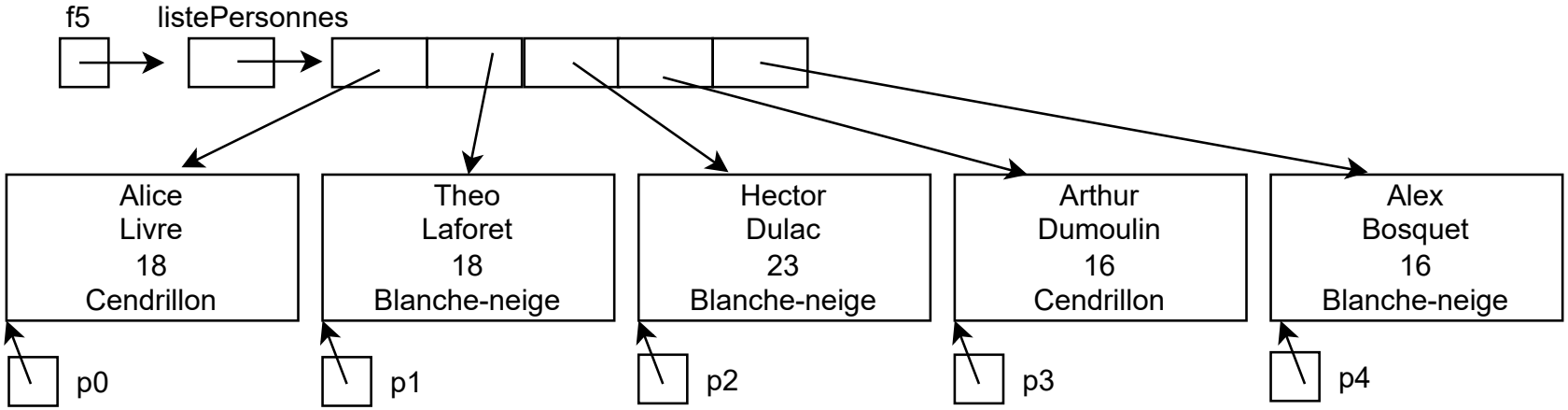
= **18**+sommeAgeRecAux(2)

Méthodes d'instances récursives



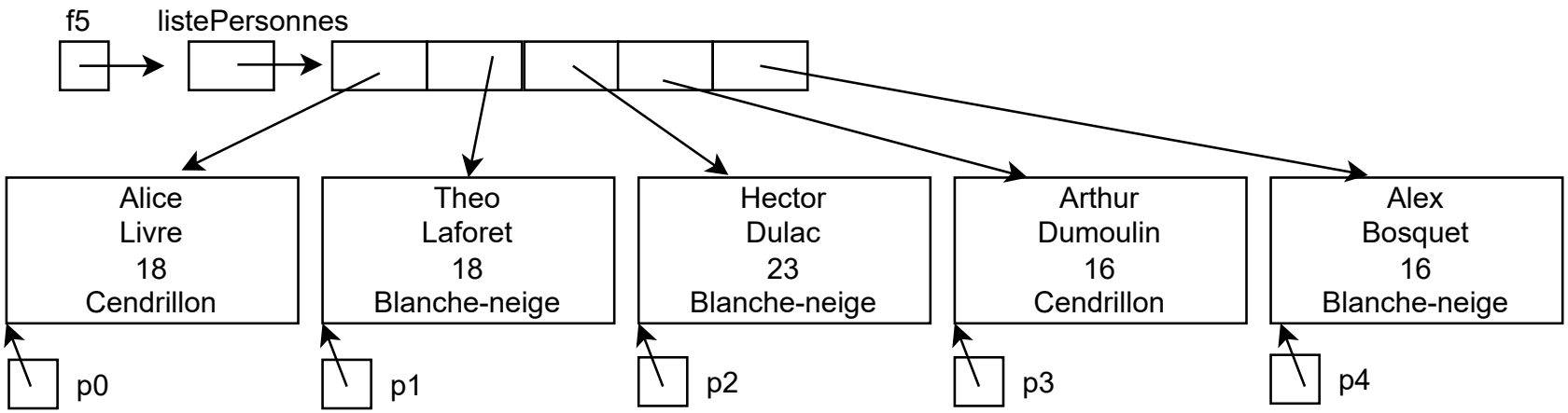
this.sommeAgeRecAuc(0)
= listePersonnes.get(0).getAge()+sommeAgeRecAux(1)
= **18**+sommeAgeRecAux(1)
= listePersonnes.get(1).getAge()+sommeAgeRecAux(2)
= **18**+sommeAgeRecAux(2)
= listePersonnes.get(2).getAge()+sommeAgeRecAux(3)
= **23**+sommeAgeRecAux(3)

Méthodes d'instances récursives



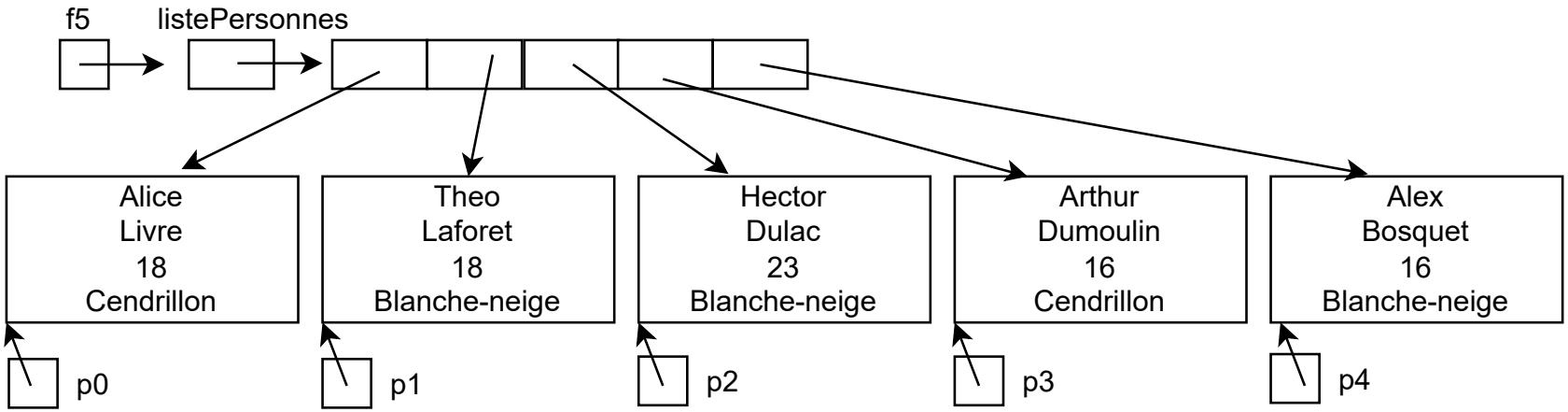
this.sommeAgeRecAuc(0)
= listePersonnes.get(0).getAge()+sommeAgeRecAux(1)
= **18**+sommeAgeRecAux(1)
= listePersonnes.get(1).getAge()+sommeAgeRecAux(2)
= **18**+sommeAgeRecAux(2)
= listePersonnes.get(2).getAge()+sommeAgeRecAux(3)
= **23**+sommeAgeRecAux(3)
= listePersonnes.get(3).getAge()+sommeAgeRecAux(4)
= **16**+sommeAgeRecAux(4)

Méthodes d'instances récursives



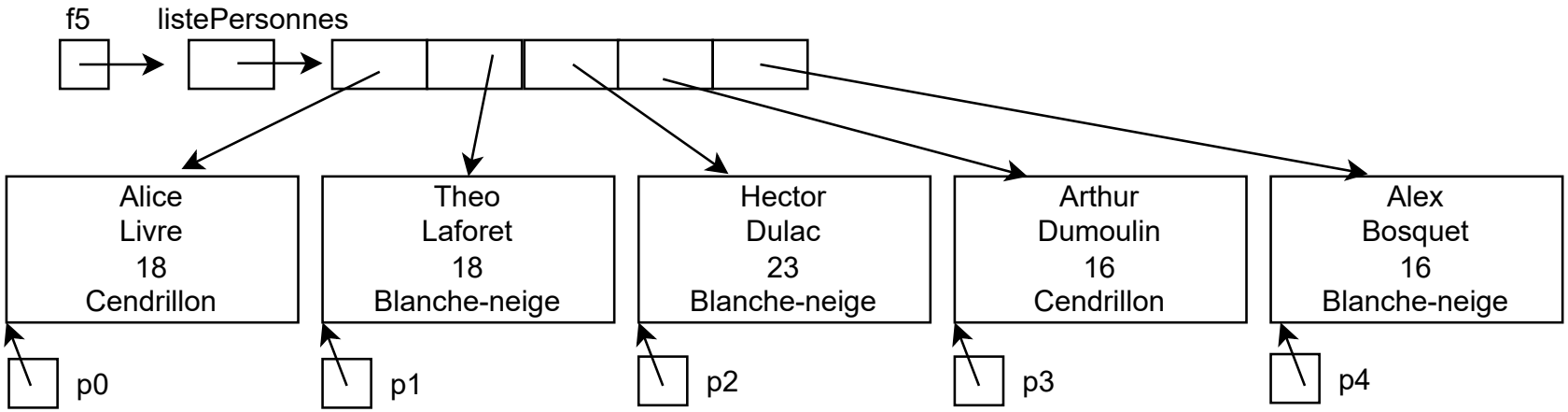
this.sommeAgeRecAuc(0)
= listePersonnes.get(0).getAge()+sommeAgeRecAux(1)
= **18**+sommeAgeRecAux(1)
= listePersonnes.get(1).getAge()+sommeAgeRecAux(2)
= **18**+sommeAgeRecAux(2)
= listePersonnes.get(2).getAge()+sommeAgeRecAux(3)
= **23**+sommeAgeRecAux(3)
= listePersonnes.get(3).getAge()+sommeAgeRecAux(4)
= **16**+sommeAgeRecAux(4)
= listePersonnes.get(4).getAge()+sommeAgeRecAux(5)

Méthodes d'instances récursives



this.sommeAgeRecAuc(0)
= listePersonnes.get(0).getAge()+sommeAgeRecAux(1)
= **18**+sommeAgeRecAux(1)
= listePersonnes.get(1).getAge()+sommeAgeRecAux(2)
= **18**+sommeAgeRecAux(2)
= listePersonnes.get(2).getAge()+sommeAgeRecAux(3)
= **23**+sommeAgeRecAux(3)
= listePersonnes.get(3).getAge()+sommeAgeRecAux(4)
= **16**+sommeAgeRecAux(4)
= listePersonnes.get(4).getAge()+sommeAgeRecAux(5)
= **16**+sommeAgeRecAux(5)
=0

Méthodes d'instances récursives



this.sommeAgeRecAuc(0)

= **18**+sommeAgeRecAux(1)

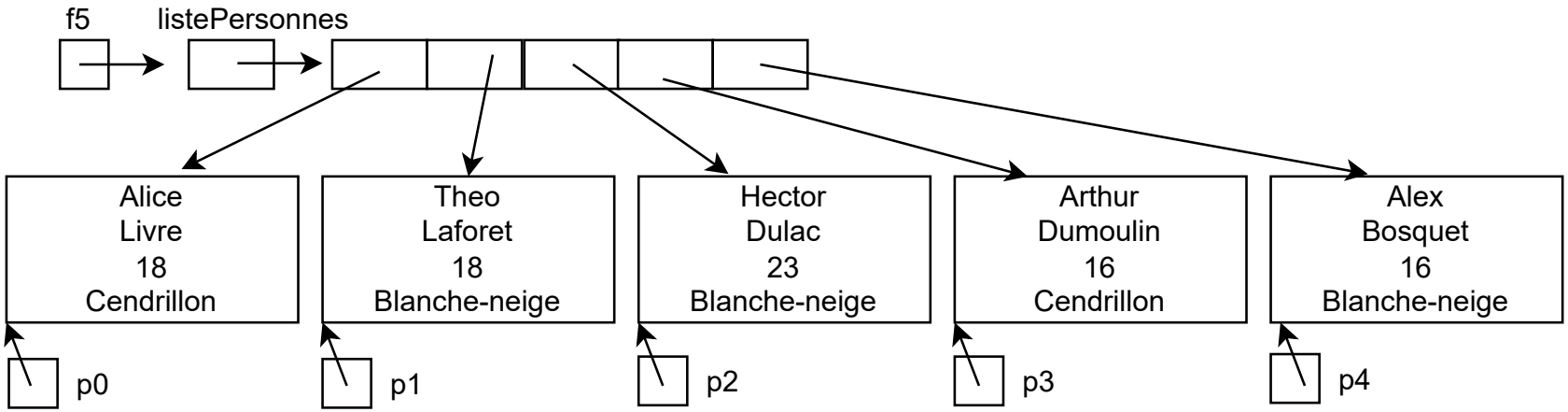
= **18**+sommeAgeRecAux(2)

= **23**+sommeAgeRecAux(3)

= **16**+sommeAgeRecAux(4)

= **16**+0

Méthodes d'instances récursives



this.sommeAgeRecAuc(0)

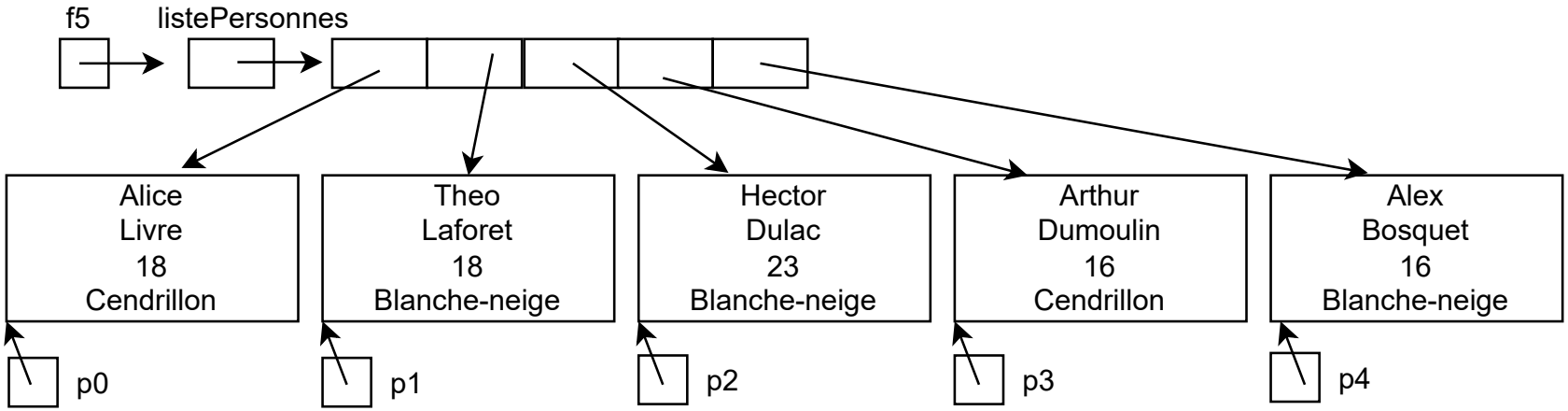
= **18**+sommeAgeRecAux(1)

= **18**+sommeAgeRecAux(2)

= **23**+sommeAgeRecAux(3)

= **16**+16+0

Méthodes d'instances récursives



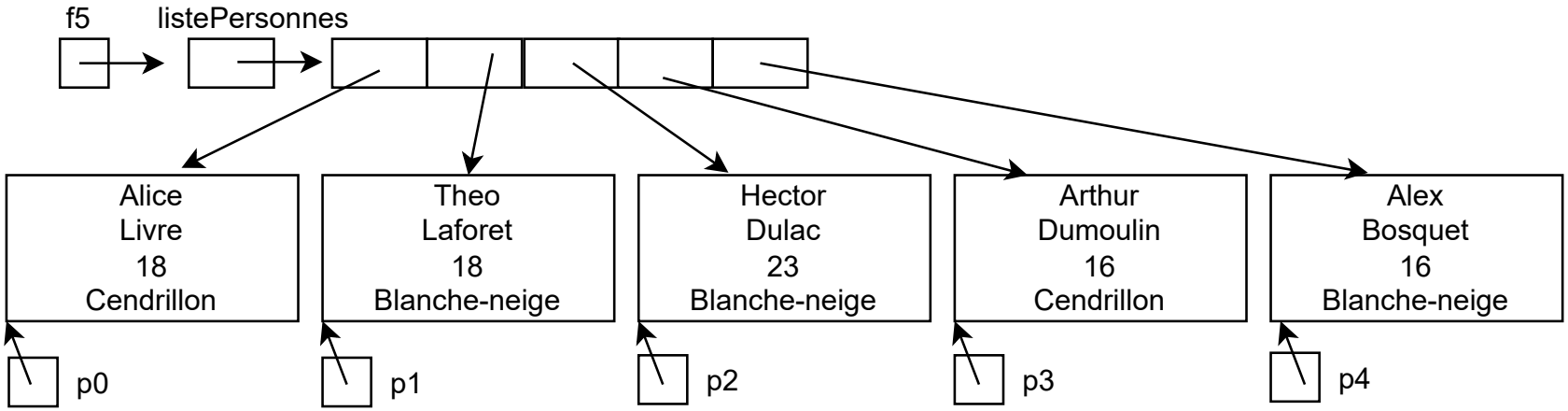
this.sommeAgeRecAuc(0)

= **18**+sommeAgeRecAux(1)

= **18**+sommeAgeRecAux(2)

= **23**+**16**+16+0

Méthodes d'instances récursives

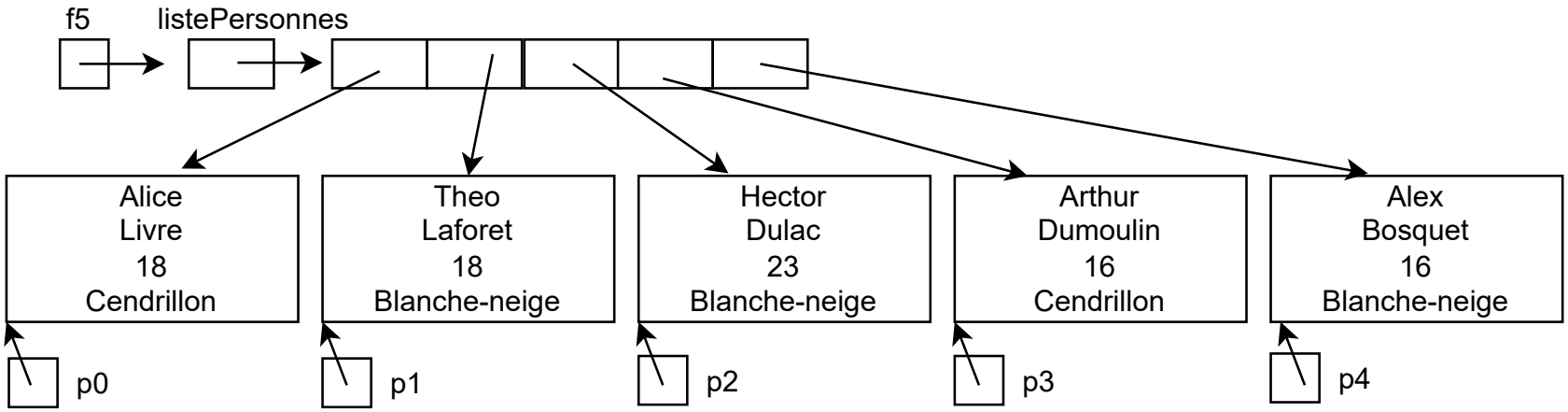


this.sommeAgeRecAuc(0)

= **18**+sommeAgeRecAux(1)

= **18**+23+16+16+0

Méthodes d'instances récursives



this.sommeAgeRecAuc(0)

= **18**+18+23+16+26+0
= 91

Méthodes d'instances récursives

Méthode qui place dans une file `fres` les spectateurs allant voir un certain film passé en paramètre.

Cas de base : tout est examiné, il n'y a rien à faire.
Cas général : si la personne désignée par le curseur `debut` va voir le film attendu, elle est placée dans `fres`.

```
public void spectateurFilmAux(int debut, String f, FileAtt fres){

   if (debut<this.listePersonnes.size()){

      if (this.listePersonnes.get(debut).getTicket().equals(f))
         fres.entre(this.listePersonnes.get(debut));

      spectateurFilmAux(debut+1, f, fres);

   }
}
```

Méthodes d'instances récursives

Méthode principale qui retourne les spectateurs allant voir un certain film passé en paramètre

```
public FileAtt spectateurFilmRec(String f) {

    FileAtt listeSpectFilm = new FileAtt();

    spectateurFilmAux(0, f, listeSpectFilm);

    return listeSpectFilm;
}
```

Pour s'exercer : la dérouler !

Plan

File d'attente récursive

Une file d'attente de personnes est :

- soit vide (cas de base)
- soit une personne, suivie d'une file d'attente (cas général)

```
public class FileAttente {

    private Personne premier = null;
    private FileAttente suiteFile;

    public FileAttente() {}

    public FileAttente(Personne premier, FileAttente suiteFile) {
        this.premier = premier;
        this.suiteFile = suiteFile;
    }
}
```

File d'attente récursive

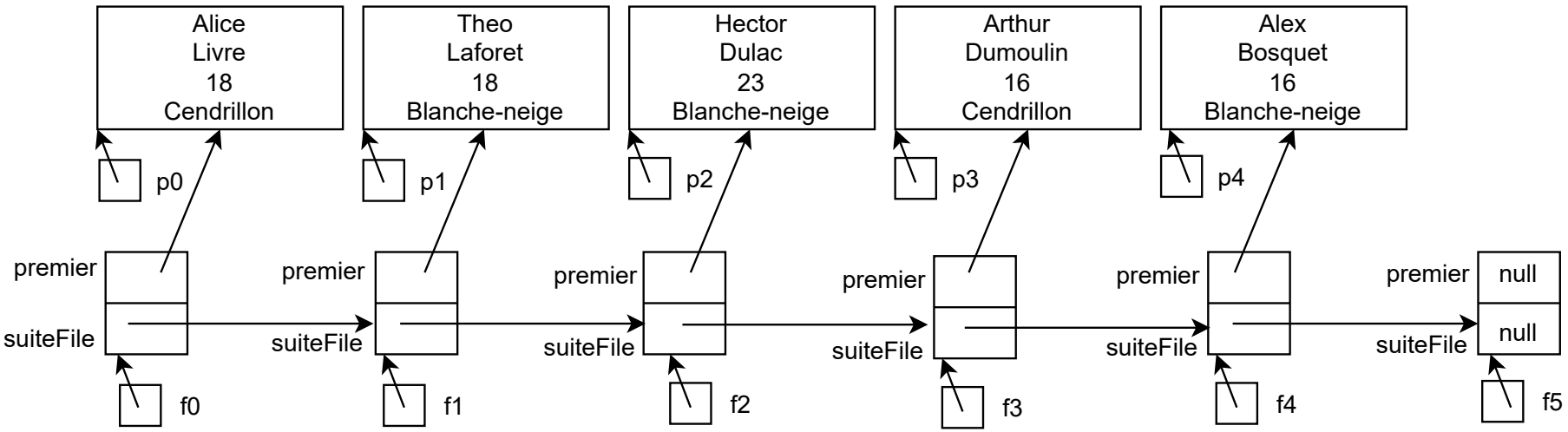
Construction d'une file d'attente de spectateurs selon le principe

- soit vide (cas de base)
- soit un spectateur, suivi d'une file d'attente (cas général)

File d'attente récursive

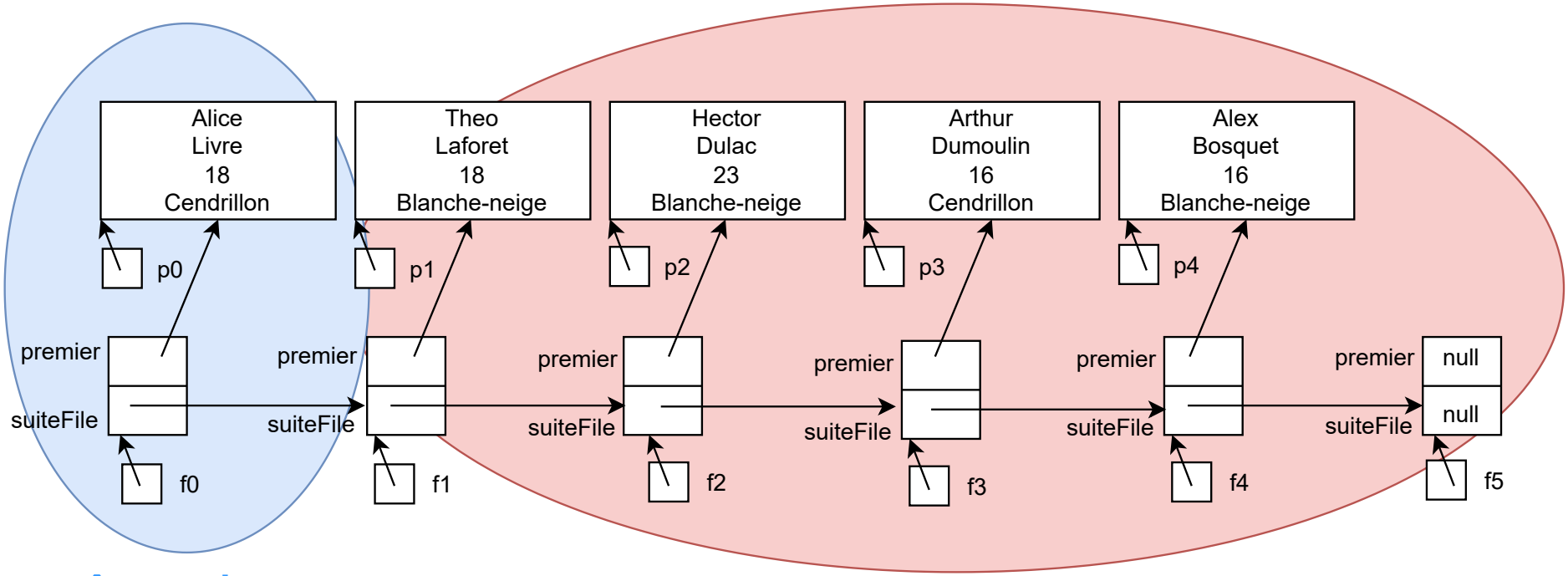
```
public static void main(String[] argv){
  Personne p0 = new Personne("Alice","Livre",18,"cendrillon");
  Personne p1 = new Personne("Theo","Laforet",18,"blanche-neige");
  Personne p2 = new Personne("Hector","Dulac",23,"blanche-neige");
  Personne p3 = new Personne("Arthur","Dumoulin",16,"cendrillon");
  Personne p4 = new Personne("Alex","Bosquet",16,"blanche-neige");

  FileAttente f5 = new FileAttente(); // cas de base
  FileAttente f4 = new FileAttente(p4, f5); // cas general
  FileAttente f3 = new FileAttente(p3, f4);
  FileAttente f2 = new FileAttente(p2, f3);
  FileAttente f1 = new FileAttente(p1, f2);
  FileAttente f0 = new FileAttente(p0, f1);

  System.out.println(f0);  System.out.println(f0.sommeAge());
  System.out.println(f0.spectateurFilm("blanche-neige"));
}
```

File d'attente récursive



File d'attente récursive

Somme des âges des spectateurs



âge du premier + Somme des âges de la suite

File d'attente récursive

Somme des âges des spectateurs et âge moyen

```
public class FileAttente {
...
  public boolean estVide(){return premier==null;}

  public int sommeAge(){
    if (this.estVide())
      return 0;
    else
      return this.premier.getAge() + this.suiteFile.sommeAge();
  }

  public double ageMoyen(){
      if (this.estVide()) return 0;
      return this.sommeAge()/this.nombreElements();
    }
}
```

File d'attente récursive

Nombre d'éléments de la file

```
public class FileAttente {
...
public int nbElements()
{
    if (this.estVide())
        return 0;
    else
        return 1 + suiteFile.nbElements();
}
...
}
```

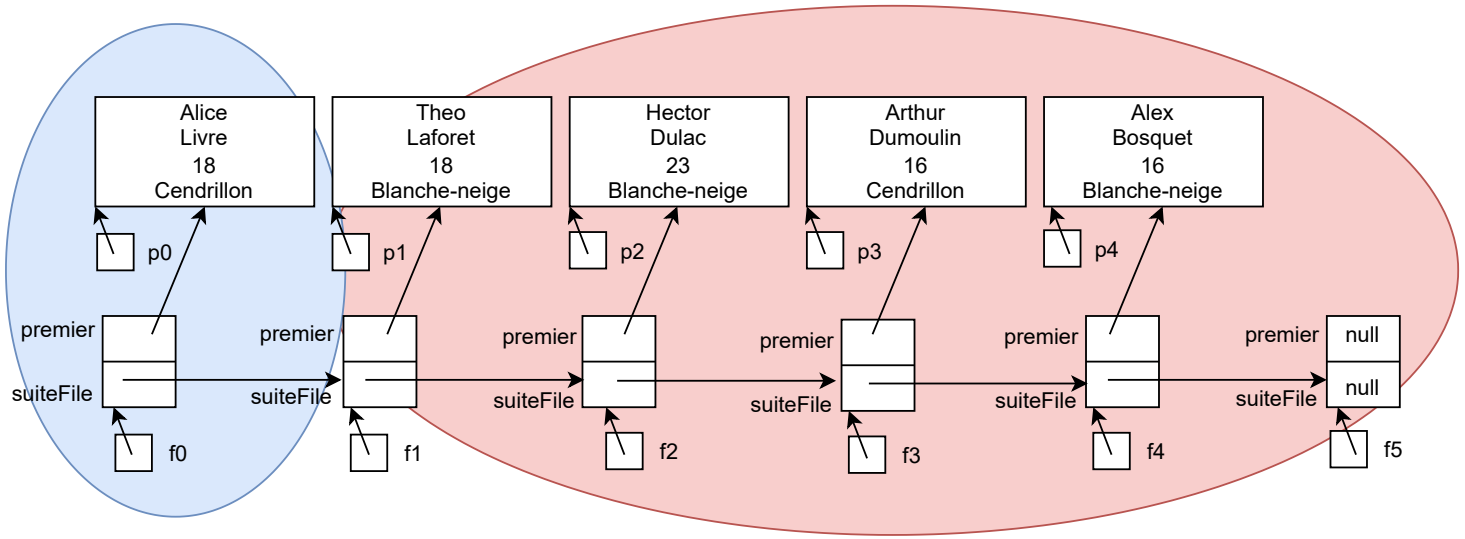
File d'attente récursive

Méthode toString

```
public class FileAttente {
...
public String toString()
{
    if (this.estVide())
        return "end";
    else
        return this.premier + "\n"+this.suiteFile;
}
...
}
```

File d'attente récursive

Sous-file d'attente des spectateurs qui vont voir Blanche-Neige.

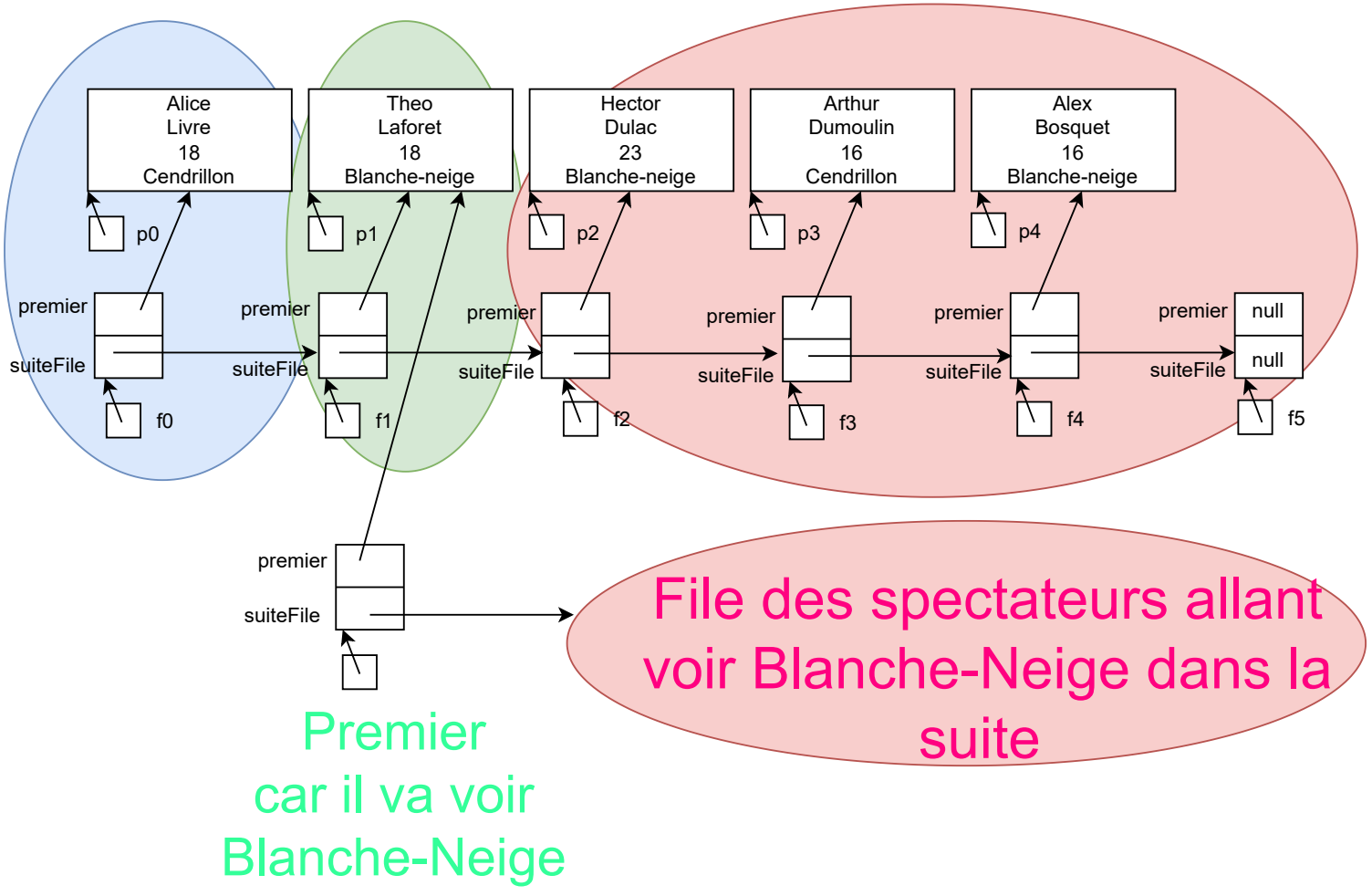


le premier ne va pas voir Blanche-Neige

File des spectateurs allant voir Blanche-Neige dans la suite

File d'attente récursive

Sous-file d'attente des spectateurs qui vont voir Blanche-Neige.



File d'attente récursive

Sous-file d'attente des spectateurs qui vont voir un certain film dont le nom est passé en paramètre ; Le film est inscrit sur le ticket du spectateur.

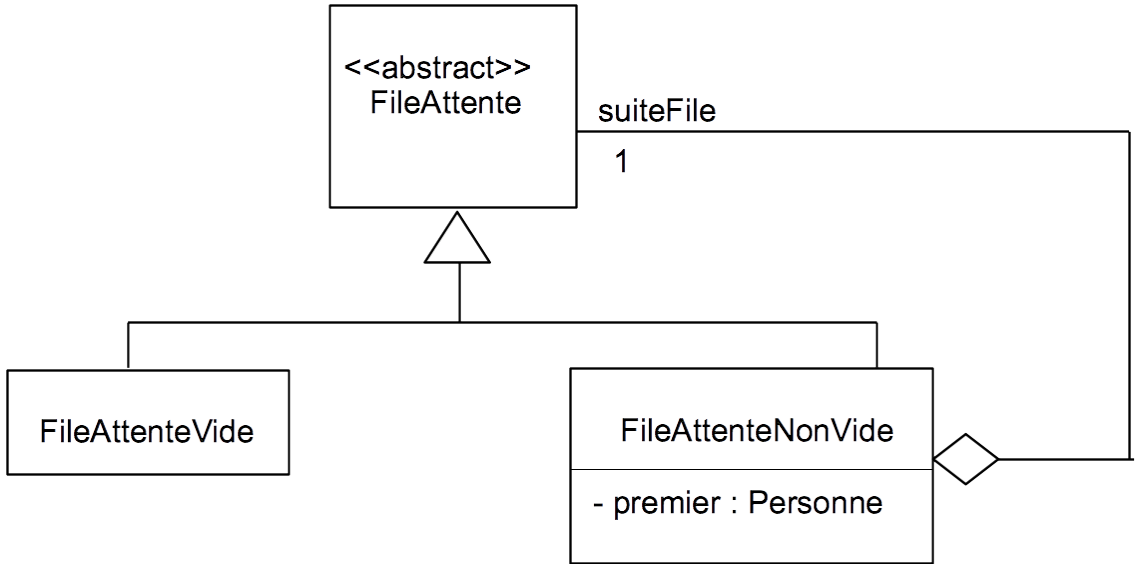
```
public class FileAttente {
...
public FileAttente film(String f)
{
   if (this.estVide())
     return new FileAttente();
   else

   if (this.premier.getTicket().equals(f))
     return new FileAttente(this.premier, this.suiteFile.film(f));

   else return this.suiteFile.film(f);
}
...
}
```

Plan

Récursivité et spécialisation

Représentation des files d'attente récursives avec de la spécialisation :

- Une classe abstraite `FileAttente` représente le concept général.
- Une sous-classe `FileAttenteVide` représente les files d'attente vides (cas de base)
- Une sous-classe `FileAttenteNonVide` représente les files d'attente non vides (cas général)



Classe abstraite

```
public abstract class FileAttente {

     public abstract boolean estVide();
     public abstract int nbElements();
     public abstract int sommeAge();
     public abstract FileAttente film(String f);

}
```

Noter que dans ce cas on peut faire aussi une interface, mais parfois on aura des attributs à stocker, dans ce cas une interface et une classe abstraite peuvent être utilisées.

File d'attente vide

```
public class FileAttenteVide extends FileAttente {

     public boolean estVide(){return true;}

     public int nbElements(){return 0;}

     public int sommeAge() {return 0;}

     public FileAttente film(String f) {
          return new FileAttenteVide();
     }

     public String toString(){
          return ".";
     }
}
```

File d'attente non vide

```
public class FileAttenteNonVide extends FileAttente {
     private Personne premier;
     private FileAttente suiteFile;

     public FileAttenteNonVide(Personne premier,
                               FileAttente suiteFile) {
          this.premier = premier;
          this.suiteFile = suiteFile;
     }

     public boolean estVide(){return false;}
...
}
```

File d'attente non vide

```
public class FileAttenteNonVide extends FileAttente {
     private Personne premier;
     private FileAttente suiteFile;

...

     public int nbElements(){
      return 1+suiteFile.nbElements();
     }

     public int sommeAge() {
          return this.premier.getAge()+this.suiteFile.sommeAge();
     }
}
```

File d'attente non vide

```
public class FileAttenteNonVide extends FileAttente {
     private Personne premier;
     private FileAttente suiteFile;

...

     public FileAttente film(String f) {
          if (this.premier.getTicket().equals(f))
               return new FileAttenteNonVide
                              (this.premier,
                              this.suiteFile.film(f));
          else
               return this.suiteFile.film(f);
     }
}
```

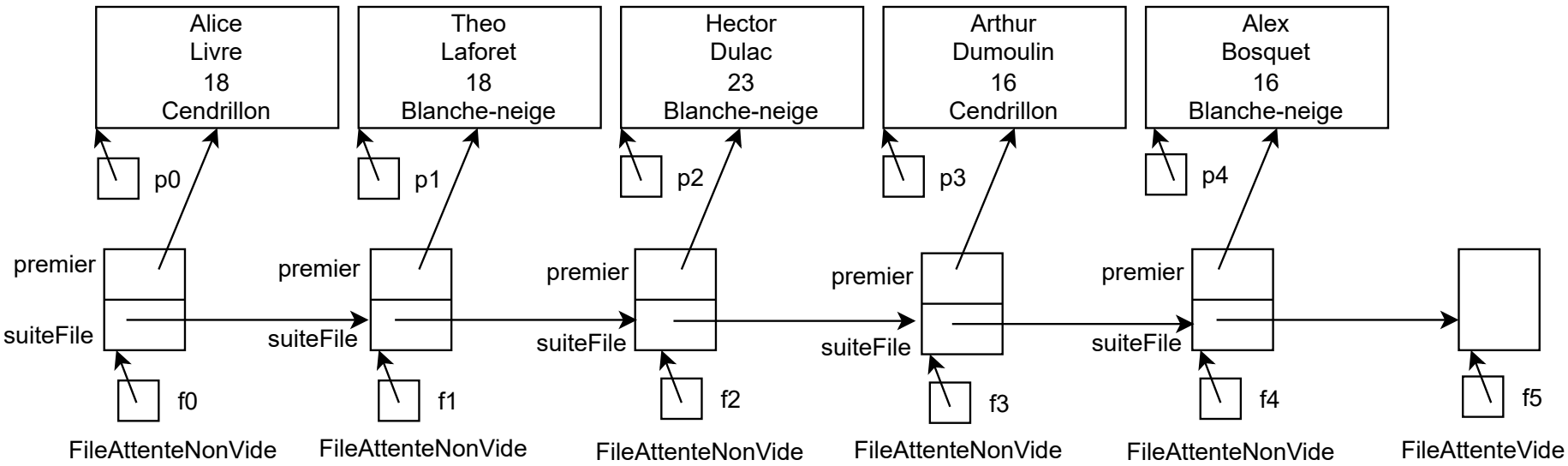
File d'attente non vide

```
public class FileAttenteNonVide extends FileAttente {
     private Personne premier;
     private FileAttente suiteFile;

...

     public String toString(){
          return this.premier.getNom() + " "+
                    this.suiteFile.toString();
     }
}
```

Création de files d'attente

```
public static void main(String[] args) {
  Personne p0 = new Personne("Alice","Livre",18,"cendrillon");
  Personne p1 = new Personne("Theo","Laforet",18,"blanche-neige");
  Personne p2 = new Personne("Hector","Dulac",23,"blanche-neige");
  Personne p3 = new Personne("Arthur","Dumoulin",16,"cendrillon");
  Personne p4 = new Personne("Alex","Bosquet",16,"blanche-neige");

  FileAttente f5 = new FileAttenteVide(); // cas de base
  FileAttente f4 = new FileAttenteNonVide(p4, f5); // cas general
  FileAttente f3 = new FileAttenteNonVide(p3, f4);
  FileAttente f2 = new FileAttenteNonVide(p2, f3);
  FileAttente f1 = new FileAttenteNonVide(p1, f2);
  FileAttente f0 = new FileAttenteNonVide(p0, f1);

  System.out.println(f0);
  System.out.println(f0.sommeAge());
  System.out.println(f0.spectateurFilm("blanche-neige"));
}
```

File d'attente récursive



Plan

Synthèse

Penser récursif et objets

- Les cas particuliers, les cas de base, sur les données de petite taille
- Les cas généraux, qui se ramènent à des sous-problèmes identiques (à des problèmes d'ordre inférieur)
- Avec la spécialisation : répartir les cas de base et les cas généraux dans les différentes classes qui les représentent et qui sont organisées dans une hiérarchie d'héritage.
- Pour aller plus loin ...
  - Récursivité terminale, non terminale, optimisation
  - Techniques de dé-récursivation