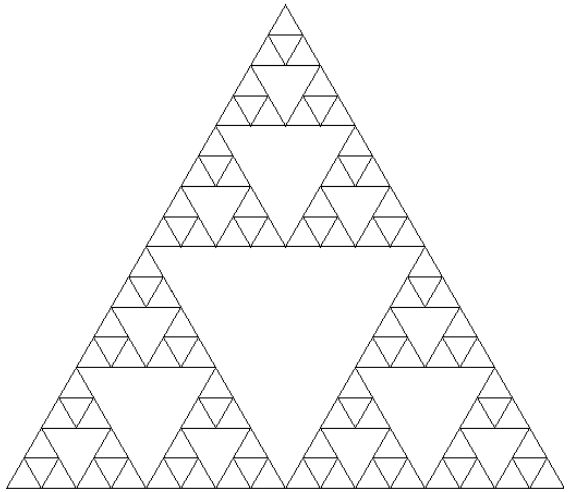
Eléments sur la récursivité et la programmation par objets

2018-2019

HMIN11M

1- Récursivité / introduction



- Une expression récursive est une expression qui se fait référence à elle-même
- En programmation, un algorithme qui s'appelle lui-même

1- Récursivité / introduction

- Constructions apparentées
  - Mise en abyme dans les arts, image qui contient une image similaire

    la vache qui rit

  - Concept décrit par une référence à lui-même

    un troupeau de moutons est un mouton ajouté à un troupeau de moutons

1- Récursivité / introduction

- Constructions apparentées
  - Une fonction définie dans ses propres termes

    Fonction factorielle

    $0! = 1$

    $n! = n * (n-1)!$ Pour $n > 0$
  - Une suite mathématique (u_n) définie par récurrence

    $u_0 = 1$

    $u_{n+1} = \sqrt{(1 + u_n)}$, pour $n > 0$

1- Récursivité / introduction

Examinons la définition de la fonction factorielle

Définition non récursive

0! = 1

n! est le produit des entiers de 1 à n, pour n >0

Définition récursive : on cherche à réduire le problème

0! = 1 est le **cas de base**

n! = n * (n-1)! pour n >0 est le **cas général**

1- Récursivité / introduction

- Pour formuler certains algorithmes et certaines structures dans des cas où l'expression est naturellement récursive

  – Fonction factorielle
  – Structures de données (une liste est un élément suivi d'une sous-liste, un arbre binaire est un élément et deux sous-arbres)

2- Fonctions récursives en Java

Factorielle n, notée aussi **n!**

Cas de base : si n = 0, c'est **1**

Cas général : c'est **n * factorielle(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));
      }}
```

2- Fonctions récursives en Java

Somme des n premiers entiers

Cas de base : si n = 0, c'est **0**

Cas général : c'est **n + la somme des n-1 premiers entiers**

```
public class FonctionsRécursives{
    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));
      }}
```

2- Fonctions récursives en Java

Puissance entière (x et n sont des entiers) : x^n

Cas de base : si n = 0, c'est **1**

Cas général : c'est **$x * x^{n-1}$**

```
public class FonctionsRécursives{
    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));      }
}
```

3- Méthodes récursives

Une 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)
    {
        this.listePersonnes.add(p);
    }
……
```

Classe Personne

```
public class Personne {

    private String nom = "nom inconnu";
    private String prenom = "prenom inconnu";
    private int age;
    private String ticket; // nom du film

.....}
```

Somme des âges des spectateurs (itératif)

```
public int sommeAge()
    {
        int somme = 0;
        for (Personne p : this.listePersonnes)
            somme += p.getAge();
        return somme;
    }
```

Somme des âges des spectateurs (récursif)

```
public int sommeAgeRecursive()
{return this.sommeAgeRecAux(0);}

public int sommeAgeRecAux(int debut)
{
    if (debut==this.listePersonnes.size())
        return 0;
    else
        return this.listePersonnes.get(debut).getAge()
                        +sommeAgeRecAux(debut+1);
}
```

4- Définition de structures récursives

- Une file d'attente de personnes devant un cinéma est :
  - soit vide
  - soit une personne, suivie d'une file d'attente

```
public class FileAttente {
    private Personne premier = null;
    private FileAttente suiteFile;

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

File d’attente

```
public class FileAttente {

    private Personne premier;
    private FileAttente suiteFile;

    public FileAttente() {
    }

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

```
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");

        FileAttente f0 = new FileAttente();
        FileAttente f1 = new FileAttente(p0, f0);
        FileAttente f2 = new FileAttente(p1, f1);
        FileAttente f3 = new FileAttente(p2, f2);

    }
```

Nombre d’éléments

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

toString

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

Age moyen

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

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

Spectateurs pour un film

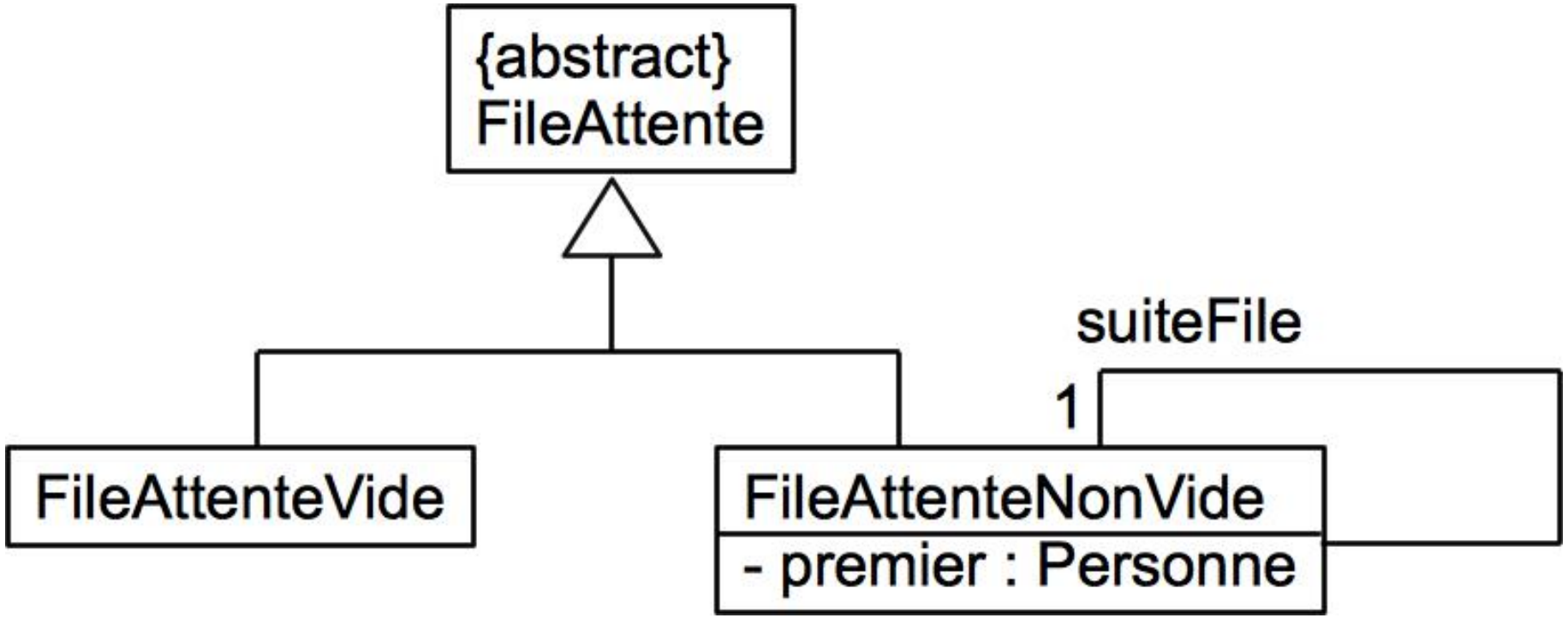
```
public FileAttente film(String f)
    {
        if (this.estVide())
            return new FileAttente();
        else
            if (this.premier.getTicket().equals(f))
                return new FileAttente(this.premier,
                                       this.suiteFile.film(f));
        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

```
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 f0 = new FileAttente();
        FileAttente f1 = new FileAttente(p4, f0);
        FileAttente f2 = new FileAttente(p3, f1);
        FileAttente f3 = new FileAttente(p2, f2);
        FileAttente f4 = new FileAttente(p1, f3);
        FileAttente f5 = new FileAttente(p0, f4);

        System.out.println(f5);
        System.out.println(f5.film("blanche-neige"));
    }
```

4- Définition de structures récursives avec spécialisation

- Utilisation de la spécialisation et de l'héritage pour distribuer le comportement
- On introduit une classe abstraite pour représenter les files d'attente
- Une file d'attente concrète de personnes devant un cinéma est :
  - soit une file d'attente vide (cas de base)
  - soit une file d'attente non vide (cas général)

Schéma des classes



File d’attente

```
public abstract class FileAttente {

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

}
```

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;}

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

File d’attente non vide

```
public class FileAttenteNonVide extends FileAttente {
…..
      public int sommeAge() {
                  return this.premier.getAge()+this.suiteFile.sommeAge();
       }

      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);
      }

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

Main

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

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

        System.out.println(f5);
        System.out.println(f5.sommeAge());
        System.out.println(f5.film("blanche-neige"));
       }
```

6- Conclusion

- Identifier les algorithmes et structures naturellement récursifs
- Penser récursivement en décomposant le problème en :
  - Ses cas particuliers, ses cas de base, sur les données de petite taille
  - Ses cas généraux, qui se ramènent à des sous-problèmes identiques (à des problèmes d'ordre inférieur)