UNIVERSITÉ PARIS 1
PANTHÉON SORBONNE

INF6
ALGORITHME AVANCÉ

Manuele Kirsch Pinheiro

UNIVERSITÉ PARIS 1
PANTHÉON SORBONNE

Contenu prévisionnel

- Piles et files
- Listes
- Récursivité
  - Récursivité dans le calcul
  - Récursivité structurelle
- Arbres binaires
  - Parcours en profondeur et en largeur
- Généralisation de la notion d'arbre
  - Insertion et suppression de nœuds
- Arbre de recherche
  - Recherche
  - Rééquilibrage

UNIVERSITÉ PARIS 1
PANTHÉON SORBONNE

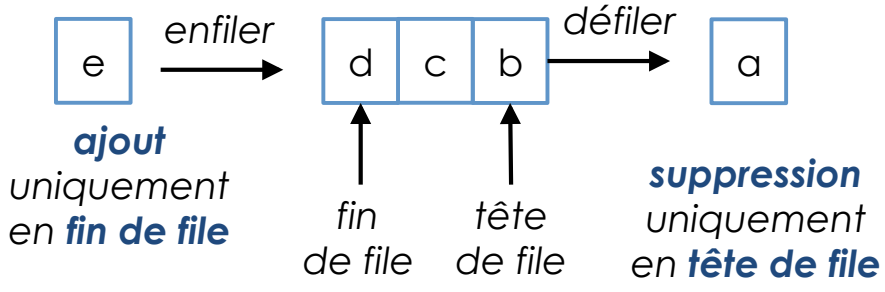
FILES

UNIVERSITÉ PARIS 1
PANTHÉON SORBONNE

Files

- **File**
  - Séquence d'**éléments accessibles** par les **deux extrémités**, suivant le modèle **FIFO** (***First In, First Out***) « *premier entré, premier sorti* »
  - Les éléments sont **insérés** dans une extrémité (queue ou **fin de file**) et en sont **extraits** par l'autre extrémité (**tête de file**)

Exemple :
- file d'attente au guichet au bureau de poste

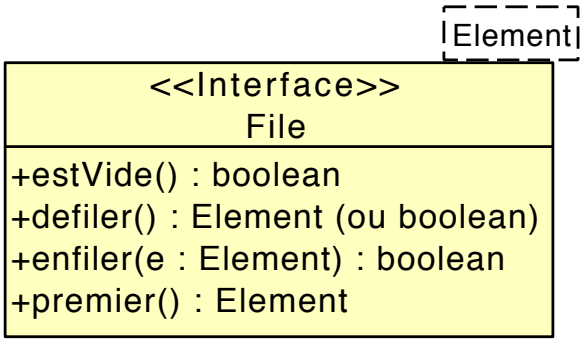


Files

- **Opérations**

  Opérations abstraites définies pour les files

  - **estVide** : indique si la file est vide ou pas
  - **enfiler** : ajoute un nouvel élément en fin de file (queue)
  - **défiler** : supprime l'élément en tête de file
  - **premier** : retourne l'élément en tête de file, sans l'enlever de la file



Files

- **Opérations**
  - **défiler enlève** le **premier** élément ajouté à la file
  - **enfiler** ne permet d'**ajouter** des éléments qu'à la **fin** de la file
  - On ne peut pas défiler une **file vide**, ni avoir son premier élément (tête)
    - soit f une file et e un élément

      $\nexists f$ *tel que* $f = défiler(fileVide)$

      $\nexists e$ *tel que* $e = premier(fileVide)$

    Exception File Vide

  - Comme pour la pile, les files peuvent être limitées à une taille fixe ou avoir une taille variable
    - si taille fixe limitée, *enfiler (filePleine, e)* ⇒ *Exception* File Pleine

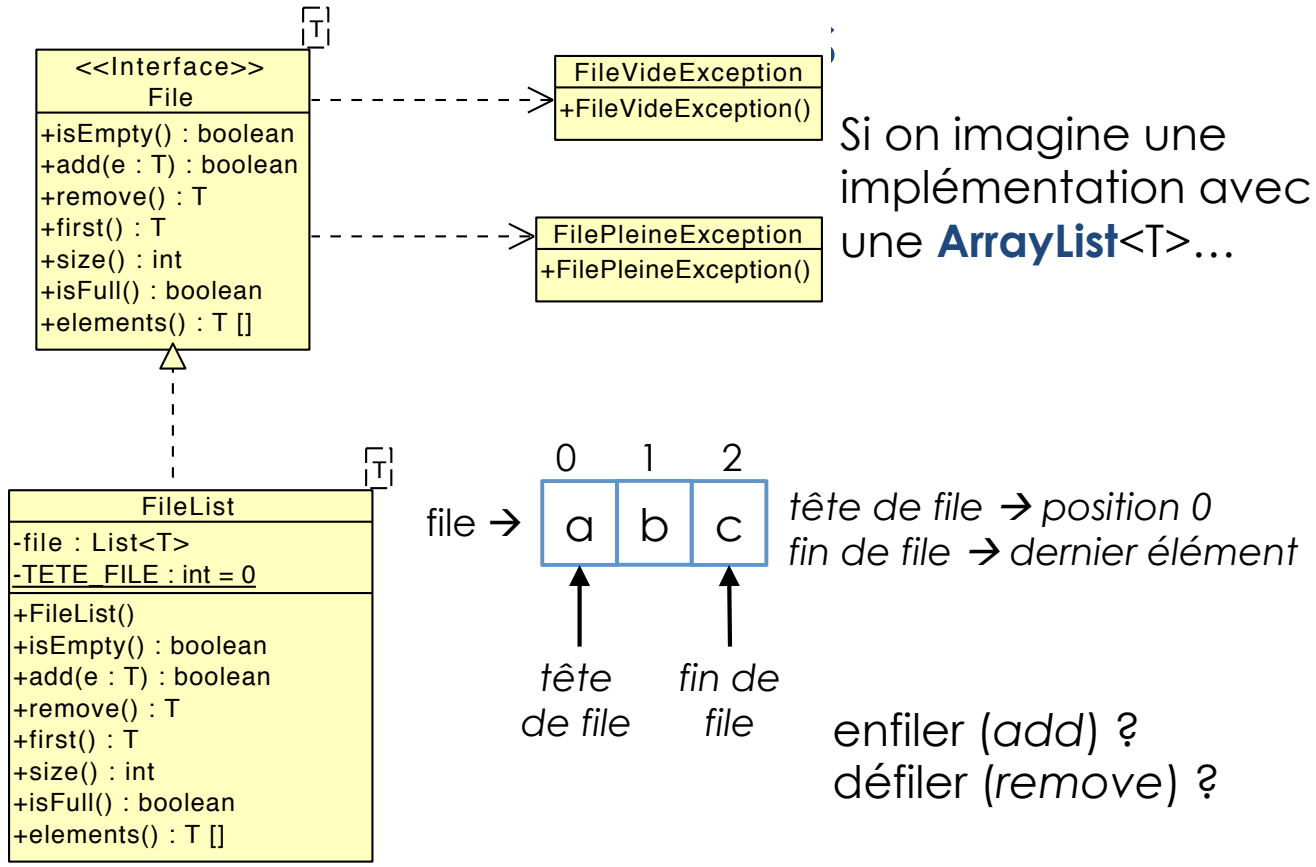
Files

- Opérations
  - Autres opérations sont envisageables

Opérations de base :
- estVide (*isEmpty*)
- enfiler (*add*)
- défiler (*remove*)
- premier (*first*)

Autres opérations utiles :
- taille (*size*)
- estPleine (*isFull*)
- éléments (*elements*)
- dernier (*last*)

- Implémentations
  - Multiples implémentations Java sont possibles



Files

```
public class FileList<T> implements File<T> {
    private List<T> file;
    private static final int TETE_FILE = 0;

    public FileList() {
        this.file = new ArrayList<T>();
    }
```

```
public boolean add (T e) {
        //on ajoute en fin de file
        return this.file.add(e);
}
```

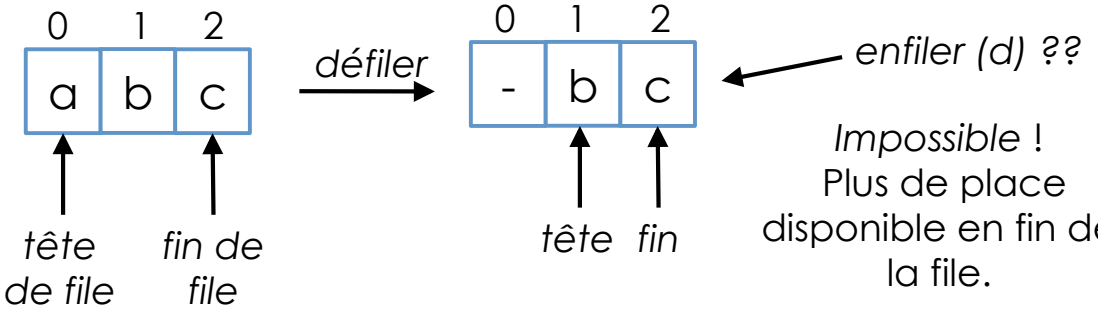
```
public boolean isEmpty() {
      return (this.file.size() == 0);
    }
```

```
public T remove () throws
FileVideException {
        if ( ! this.isEmpty() ) {
            //on supprime la tête de file
            return this.file.remove(TETE_FILE);
        } else {
            throw new FileVideException();
        }
    }
```


Files

- **File avec un tableau**
  - **Problème** : gestion des places qui se libèrent en tête de file
    - Il faut « compacter » le tableau

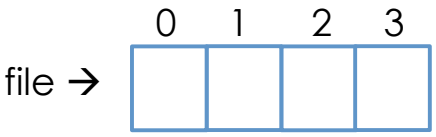
taille = 3



Files

• File avec un tableau : opération *Enfiler*

taille = 4

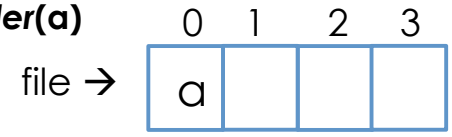


tête = 0 *pos prochain élément à enlever*
fin = -1 *pos dernier élément ajouté*

si tête > fin alors est Vide

cas 1) premier élément (file vide)

***enfiler*(a)**

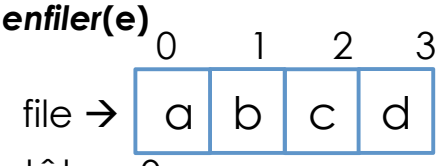


tête = 0
fin = ~~-1~~ **0**

taille = 4

cas 3) file pleine

***enfiler*(e)**



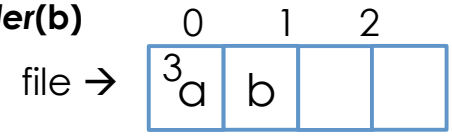
file pleine !

tête = 0
fin = ~~-1 0 1 2~~ **3**

fin >= taille

cas 2) file non vide

***enfiler*(b)**



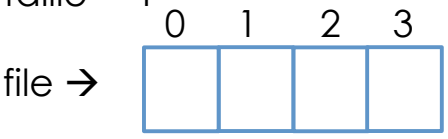
tête = 0
fin = ~~-1 0~~ **1**

taille = 4

Files

• File avec un tableau : positions vides

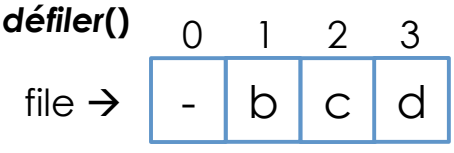
taille = 4



tête = 0 *pos prochain élément à enlever*
fin = -1 *pos dernier élément ajouté*

si tête > fin alors est Vide

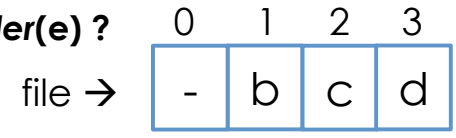
***défiler*()**



tête = ~~0~~ **1**
fin = ~~-1 0 1 2~~ **3**

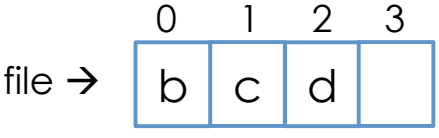
cas 4) places libres avant (défiler)

***enfiler*(e) ?**



tête = ~~0~~ **1**
fin = ~~-1 0 1 2~~ **3**

taille = 4



tête = ~~0 1~~ **0**
fin = ~~-1 0 1 2~~ ~~3~~ **2**

enfiler (e) : booléen

entrée :
Elément e
sortie :
booléen réponse

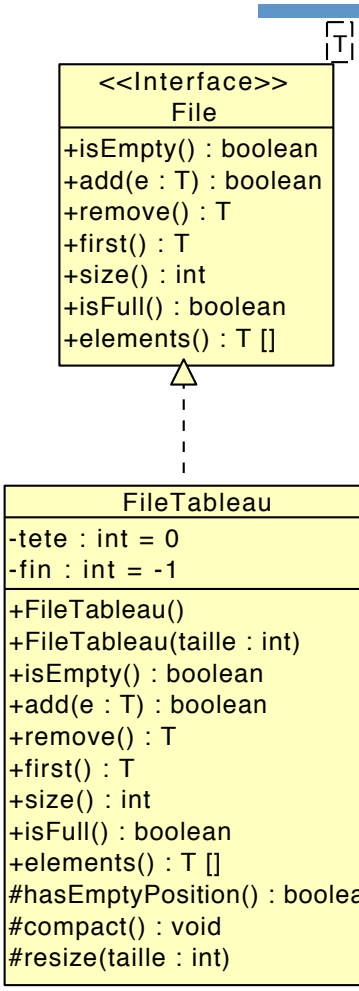
Si (fin + 1) < fille.taille
alors // il y a de la place
fin = fin + 1
file[fin] = e
réponse = VRAI
sinon // vérifier places libres
Si placesLibres()
alors
compacter()
réponse = enfiler(e)
sinon
réponse = FAUX
fin Si
fin Si
retourner réponse
Fin enfiler

placesLibres ( ) : booléen

sortie :
booléen réponse
Si tête > 0
alors // il y a de la place en tête
réponse = VRAI
sinon
réponse = FAUX
fin Si
retourner réponse
Fin placesLibres

défiler ( ) : Elément

sortie :
Elément s
Si tête > fin
alors
Lancer Exception File Vide
sinon
s = file[tête]
tête = tête + 1
fin Si
retourner s
Fin défiler



plusieurs moyens de compacter les places vides en tête de file…

compacter ( )

Entier i
i = 0
Tant que i ≤ (fin – tête)
faire
file [i] = file [tête + i]
i = i + 1
fin tant que
fin = i – 1
tête = 0
Fin compacter

compacter ( )

Entier i, j
i = 0
j = tête
Tant que i < file.taille ET j ≤ fin
faire
file [i] = file [j]
i = i + 1 ; j = j + 1
fin tant que
fin = i – 1
tête = 0
Fin compacter