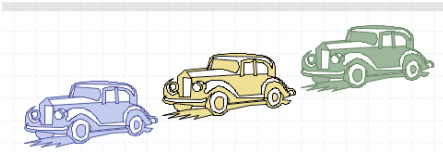
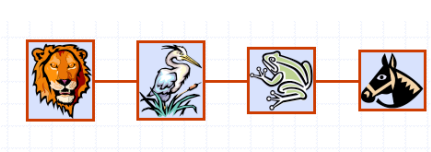


## FILES



## LISTES CHAÎNÉES



## Type Abstrait de Données FILE (§4.7)

- Garde en mémoire des objets arbitraires
- Les insertions et suppressions se font dans l'ordre "premier arrivé, premier sorti (ou servi!)"
- Principales opérations:
  - `enqueue(element)`  
■ `ajouter(objet)`: insère un objet à la fin de la file
  - `dequeue()`  
■ `objet enlever()`: retire et retourne l'objet au début de la file

## Type abstrait de données FILE (suite)

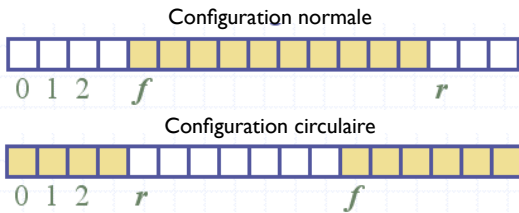
- opérations auxiliaires
  - objet `devant()`: retourne l'objet n devant de la file sans le retirer
  - entier `taille()`: retourne le nombre d'objets de la file
  - booléen `estVide()`: indique si la file est vide ou non
- Exceptions
  - `ExceptionFileVide` si on exécute `devant()` ou `enlever()` sur une file vide

## Applications des piles

- Applications directes
  - Listes d'attentes
  - Accessibilité à des ressources partagées (imprimante)
- Applications indirectes
  - Apparaît comme structure de données auxiliaire dans certains algorithmes

## Première implémentation d'une file

- On utilise une liste de longueur  $N$ , d'une façon circulaire
- Deux variables gardent en mémoire le devant et le derrière de la file
  - $f$  est l'indice du devant de la file
  - $r$  est l'indice du derrière de la file
- La position  $r$  de la liste est toujours vide

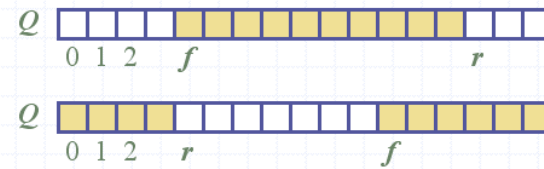


## Opérations sur une file

- On utilise l'opérateur modulo pour calculer la taille de la file

**Algorithme *taille()***  
retourner  $(N - f + r) \bmod N$

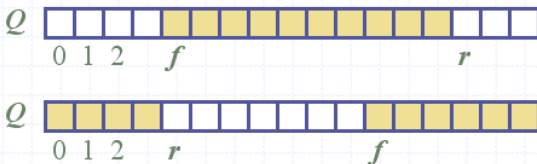
**Algorithme *estVide()***  
retourner  $(f = r)$



## Opérations sur une file (suite)

- L'opération **ajouter(o)** envoie une exception si la liste est pleine
- Cette exception est liée à l'implémentation

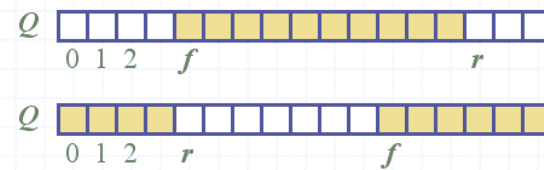
**Algorithme *ajouter(o)***  
si  $taille() = N - 1$  alors  
  throw *ExceptionFilePleine*  
sinon  
   $Q[r] \leftarrow o$   
   $r \leftarrow (r + 1) \bmod N$



## Opérations sur une file (suite)

- L'opération **enlever()** envoie une exception si la liste est vide
- Cette exception est intrinsèque au TAD pile

**Algorithme *enlever()***  
si *estVide()* alors  
  throw *ExceptionFileVide*  
sinon  
   $o \leftarrow Q[f]$   
   $f \leftarrow (f + 1) \bmod N$   
  retourner  $o$



## Implémentation de notre file

- Interface JAVA correspondant à notre TAD file
- On doit définir une classe `ExceptionFileVide`
- Il n'existe pas de classe JAVA intrinsèque pour les files

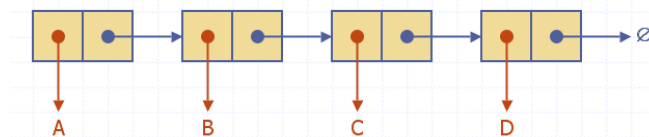
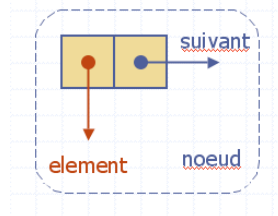
```
public interface Pile {
    public int taille();
    public boolean estVide();
    public Object devant()
        throws EmptyQueueException;
    public void ajouter(Object o);
    public Object enlever()
        throws EmptyQueueException;
}
```

## Complexité et limitations

- Si  $N$  est la longueur de la liste utilisée dans l'implémentation
  - Complexité en espace:  $O(N)$
  - Complexité en temps des opérations:  $O(1)$
- Limitations
  - La longueur maximale de la liste doit être définie a priori et ne peut être changée
  - Essayer d'ajouter un nouvel élément dans une liste pleine cause une exception (liée à l'implémentation)

## Listes chaînées (§3.3)

- Une liste simplement chaînée est une structure de données concrète constituée d'une séquence de noeuds
- Chaque noeud garde en mémoire une référence à un objet et un lien (pointeur) vers un autre noeud



© 2004, Goodrich, Tamassia

## La classe "Node"

```
public class Node {
    // Instance variables:
    private Object element;
    private Node next;
    /** Creates a node with null references to its element and next node. */
    public Node() {
        this(null, null);
    }
    /** Creates a node with the given element and next node. */
    public Node(Object e, Node n) {
        element = e;
        next = n;
    }
    // Accessor methods:
    public Object getElement() {
        return element;
    }
    public Node getNext() {
        return next;
    }
    // Modifier methods:
    public void setElement(Object newElem) {
        element = newElem;
    }
    public void setNext(Node newNext) {
        next = newNext;
    }
}
```

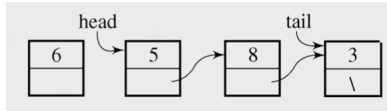
© 2004, Goodrich, Tamassia

## Insérer un élément en tête de liste

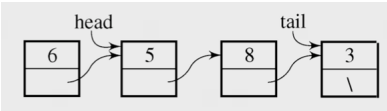
- Créer un nouveau noeud



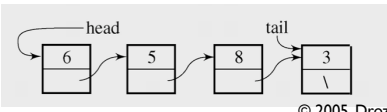
- Insérer un nouvel élément



- Faire pointer le nouveau noeud sur l'ancienne tête de liste



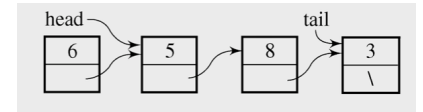
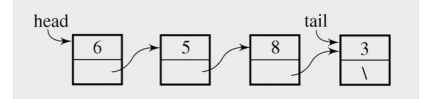
- La tête devient le nouveau noeud



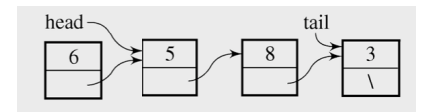
© 2005, Drozdek

## Enlever l'élément en tête de liste

- La tête devient le prochain élément de la liste



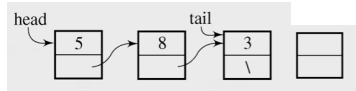
- Détacher l'ancienne tête de la liste



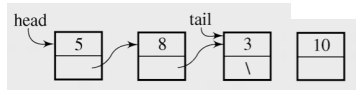
© 2005, Drozdek

## Insérer à la fin de la liste

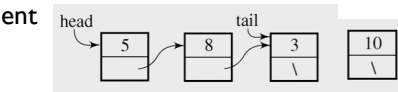
- Créer un nouveau noeud



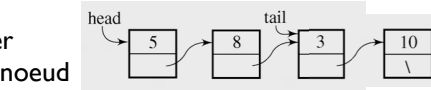
- Insérer un nouvel élément



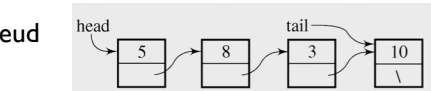
- Faire pointer le nouvel élément sur null



- Faire pointer l'ancien dernier élément sur notre nouveau noeud



- "tail" devient le nouveau noeud

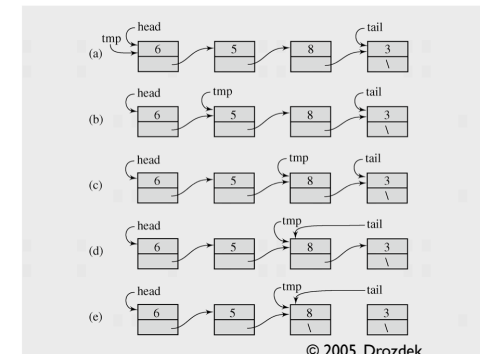


© 2005, Drozdek

## Enlever un élément à la fin de la liste

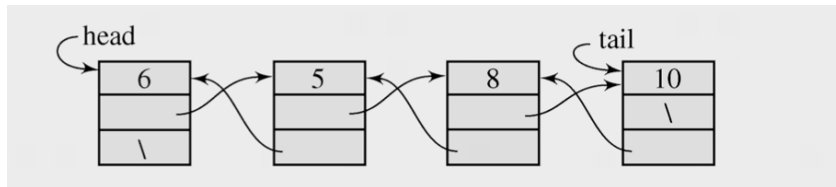
- Dans une liste simplement chaînée, on ne peut enlever efficacement un élément à la fin de la liste
- Cela vient du fait que pour accéder au noeud avant le noeud final, on doit passer à travers toute la liste.

FIGURE 3.7 Deleting a node from the end of a singly linked list.

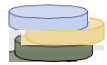


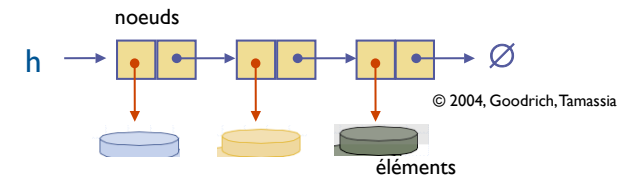
© 2005, Drozdek

## Liste doublement chaînée (§3.4)



## Implémenter une pile avec une liste chaînée

- On peut implémenter une pile avec une liste simplement chaînée 
- L'objet au dessus de la pile est gardé en mémoire dans le premier noeud de la liste
- La complexité en espace est  $O(n)$ , où  $n$  est la taille de la pile et chaque opération peut s'exécuter en  $O(1)$



## Implémenter une file avec une liste chaînée

- On peut implémenter une file avec une liste simplement chaînée
  - L'élément devant la file est gardé en mémoire dans le premier noeud de la liste
  - L'élément en fin de file est gardé en mémoire dans le dernier noeud de la liste
- La complexité en espace est  $O(n)$ , où  $n$  est la taille de la file et chaque opération peut s'exécuter en  $O(1)$

