
INF2010 – ASD

Listes, piles et files

Plan

- Concept de liste
- Classes et interfaces standards
- Implémentation d'une liste par tableau
- Implémentation d'une liste chaînée
- Piles et files

Plan

- Concept de liste
- Classes et interfaces standards
- Implémentation d'une liste par tableau
- Implémentation d'une liste chaînée
- Piles et files

Liste

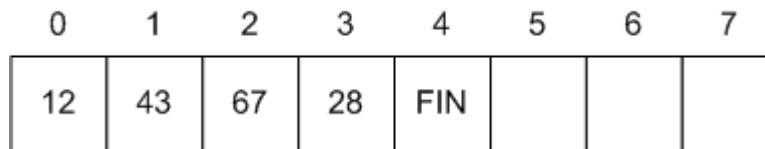
- La liste est une structure de donnée abstraite permettant d'accéder librement à une collection d'objets.
- La liste nous sera utile pour implémenter des structures de données plus complexes, telles que file (FIFO), pile (LIFO), dictionnaire, etc.
- Le concept de liste nous aidera également à faire nos premiers pas dans l'étude de la complexité asymptotique des algorithmes

Liste

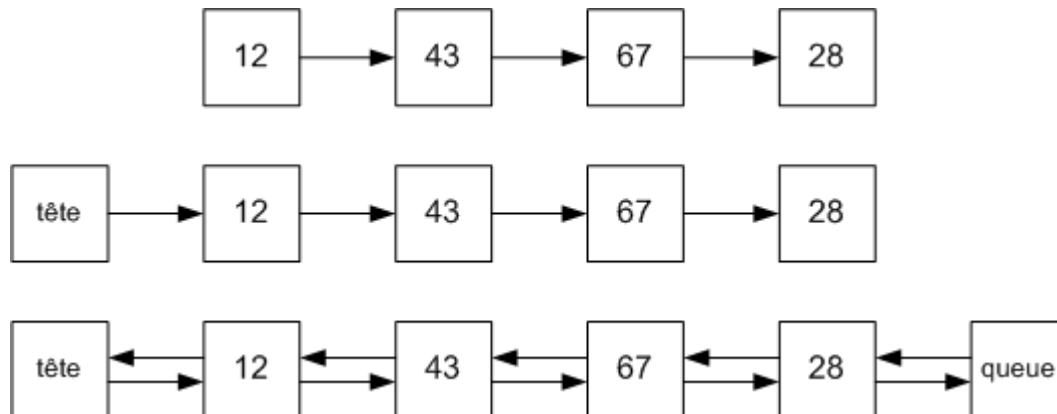
- On associe généralement aux listes les méthodes de base que sont:
 - `add()` : insertion d'un nouvel objet dans la liste
 - `remove()`: retrait d'un objet de la liste
 - `get()`: obtient un objet
 - `set()`: modifie un objet
 - `empty()`: informe que la liste est vide ou non
 - `size()`: donne le nombre d'éléments dans la liste
- Nous allons voir dans ce qui suit comment implémenter une liste en Java

Liste

- On considère généralement deux type d'implémentation de la liste:
 - Liste par tableau



- Liste chaînée



Plan

- Concept de liste
- **Classes et interfaces standards**
- Implémentation d'une liste par tableau
- Implémentation d'une liste chaînée
- Piles et files

Collections

- Les objets et les interfaces que nous allons manipuler implémentent ou dérivent de l'interface `Collection`.
- L'interface `Collection` dérive elle-même de l'interface `Iterable`.

Collections

```
public interface Collection<AnyType> extends
Iterable<AnyType>
{
    int size( );
    boolean isEmpty( );
    void clear( );
    boolean contains( AnyType x );
    boolean add( AnyType x );
    boolean remove( AnyType x );
    java.util.Iterator<AnyType> iterator( );
}
```

« for intelligent » (collections)

- Il est à noter qu'il est possible d'effectuer une boucle intelligente sur un type « Iterable »

```
public static <AnyType> void  
print(Collection<AnyType> coll)  
{  
    for( AnyType item : coll )  
        System.out.println( item );  
}
```

Interface « Iterator »

```
public interface Iterator<AnyType>
{
    boolean hasNext( );
    AnyType next( );
    void remove( );
}
```

Complexite:

- hasNext, next $\rightarrow O(1)$
- remove
 - $O(n)$ sur ArrayList
 - $O(1)$ sur LinkedList

Exemple (iterator)

```
public static <AnyType> void
print(Collection<AnyType> coll )
{
    Iterator<AnyType> itr = coll.iterator( );
    while( itr.hasNext( ) )
    {
        AnyType item = itr.next( );
        System.out.println( item );
    }
}
```

Interface « List »

```
public interface List<AnyType> extends
Collection<AnyType>
{
    AnyType get(int idx);
    AnyType set(int idx, AnyType newVal);
    void add(int idx, AnyType x);
    void remove(int idx);

    ListIterator<AnyType> listIterator(int pos);
}
```

Interface « List »

- Complexité des implantations
 - ArrayList
 - Get $\rightarrow O(1)$
 - Set $\rightarrow O(1)$
 - Add $\rightarrow O(n)$, En fin de liste $\rightarrow O(1)$
 - Remove $\rightarrow O(n)$, En fin de liste $\rightarrow O(1)$
 - LinkedList
 - Get $\rightarrow O(n)$
 - Set $\rightarrow O(n)$
 - Add $\rightarrow O(n)$
 - Remove $\rightarrow O(n)$
 - addFirst, addLast, getFirst, getLast $\rightarrow O(1)$
-

Exemple

- `removeEvensVer1` $\rightarrow O(n^2)$ sur toutes les listes implantées par “array” ou “liste chaînée”

```
public static void
removeEvensVer1(List<Integer> lst)
{
    int i = 0;
    while( i < lst.size() )
        if( lst.get( i ) % 2 == 0 )
            lst.remove( i );
        else
            i++;
}
```

Exemple (2)

- Exception: ConcurrentModificationException

```
public static void  
removeEvensVer2 (List<Integer> lst)  
{  
    for (Integer x : lst)  
        if ( x%2 == 0 )  
            lst.remove( x );  
}
```

- removeEvensVer2
 - $O(n^2)$ sur ArrayList et LinkedList
 - Introduit une exception car l'itérateur est corrompu par le retrait

Exemple (3)

```
public static void
removeEvensVer3(List<Integer> lst )
{
    Iterator<Integer> itr = lst.iterator();

    while( itr.hasNext() )
        if( itr.next() % 2 == 0 )
            itr.remove();
}
```

- removeEvensVer3
 - $O(n^2)$ sur ArrayList
 - $O(n)$ sur LinkedList

Interface « ListIterator »

```
public interface ListIterator<AnyType> extends
Iterator<AnyType>
{
    boolean hasPrevious( );
    AnyType previous( );

    void add(AnyType x);
    void set(AnyType newVal);

    //... Sous-ensemble de l'interface totale ...
}
```

Interface « ListIterator » (2)

- Complexité de l'implantation par liste chaînée
 - hasPrevious $\rightarrow O(1)$
 - previous $\rightarrow O(1)$
 - add $\rightarrow O(1)$
 - set $\rightarrow O(1)$

Interface « ListIterator » (3)

- **Interface**("ListIterator") \supset **interface**("Iterator")

{« hasNext () »,		{« hasNext () »,
« next () »,		« next () »,
« remove () »,		« remove () » }
« hasPrevious () », \supset		
« previous () »,		
« add (AnyType x) »,		
« set (AnyType newVal) » }		

- **“ListIterator” est un sous-type de “Iterator”**

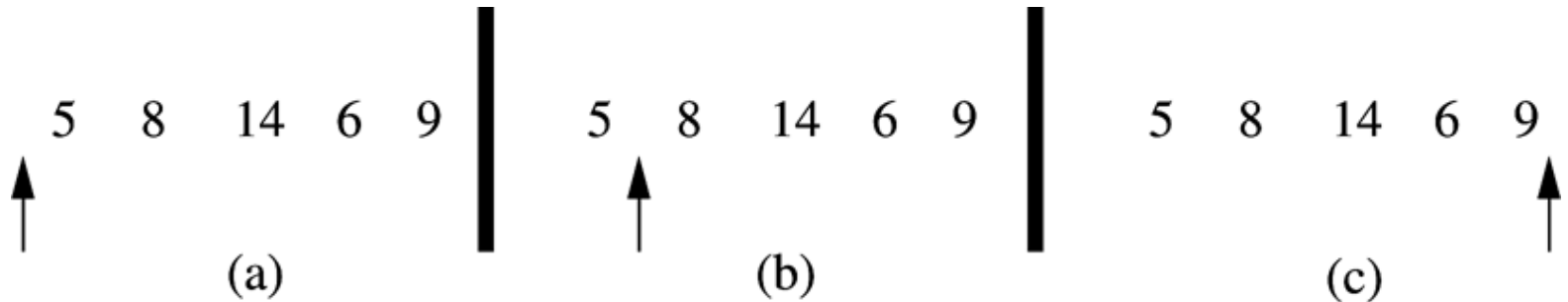
Interface « ListIterator » (4)

- Tout objet instance d'une classe de type T1 sous-type de T2 peut remplacer un objet instance d'une classe de type T2 (mais pas l'inverse)
- Tout objet instance d'une classe de type "ListIterator" peut remplacer un objet instance d'une classe de type "Iterator" (mais pas l'inverse)

Interface « ListIterator » (5)

- Tout objet instance d'une classe qui implante ("implements") "ListIterator" peut remplacer un autre objet instance d'une classe qui implante "Iterator"
- Le sous-typage est explicite en Java avec le mot clef « **extends** »
 - `ListIterator extends Iterator`

Exemple



(a): next \rightarrow 5, previous \rightarrow UNDEF, add \rightarrow avant 5

(b): next \rightarrow 8, previous \rightarrow 5, add \rightarrow entre 5 et 8

(c): next \rightarrow UNDEF, previous \rightarrow 9, add \rightarrow apres 9

Plan

- Concept de liste
- Classes et interfaces standards
- **Implémentation d'une liste par tableau**
- Implémentation d'une liste chaînée
- Piles et files

MyArrayList

```
public class MyArrayList<AnyType> implements Iterable<AnyType>
{
    private static final int DEFAULT_CAPACITY = 10;

    private int theSize;
    private AnyType [] theItems;

    public MyArrayList()
    { clear(); }

    public void clear()
    {
        theSize = 0;
        ensureCapacity( DEFAULT_CAPACITY );
    }
}
```

MyArrayList (2)

```
public int size()
{ return theSize; }

public boolean isEmpty()
{ return size() == 0; }

public void trimToSize()
{ ensureCapacity(size()); }

public AnyType get(int idx)
{
    if( idx < 0 || idx >= size( ) )
        throw new ArrayIndexOutOfBoundsException( );
    return theItems[ idx ];
}
```

MyArrayList (3)

```
public AnyType set( int idx, AnyType newVal )
{
    if( idx < 0 || idx >= size( ) )
        throw new ArrayIndexOutOfBoundsException( );
    AnyType old = theItems[ idx ];
    theItems[ idx ] = newVal;
    return old;
}


public void ensureCapacity( int newCapacity )
{
    if( newCapacity < theSize )
        return;

    AnyType [ ] old = theItems;
    theItems = (AnyType [ ]) new Object[ newCapacity ];
    for(int i = 0; i < size( ); i++ )
        theItems[ i ] = old[ i ];
}
```



MyArrayList(4)

```
public boolean add( AnyType x )
{
    add( size( ), x );
    return true;
}
```

```
public void add(int idx, AnyType x ) 
{
    if( theItems.length == size( ) )
        ensureCapacity( size( ) * 2 + 1 ); // pour eviter size == 0

    for(int i = theSize; i > idx; i-- )
        theItems[ i ] = theItems[ i - 1 ];

    theItems[ idx ] = x;

    theSize++;
}
```

MyArrayList(5)

```
public AnyType remove(int idx)
{
    AnyType removedItem = theItems[ idx ];

    for(int i = idx; i < size( ) - 1; i++ )
        theItems[ i ] = theItems[ i + 1 ];

    theSize--;

    return removedItem;
}
```



```
public java.util.Iterator<AnyType> iterator( )
{
    return new ArrayListIterator( );
}
```




MyArrayList(6)

```
private class ArrayListIterator
implements java.util.Iterator<AnyType>
{
    private int current = 0;

    public boolean hasNext()
    { return current < size(); }

    public AnyType next()
    {
        if( !hasNext() )
            throw new java.util.NoSuchElementException();
        return theItems[ current++ ];
    }

    public void remove( )
    { MyArrayList.this.remove(--current); }
}
} // Fin de la class MyArrayList
```



Plan

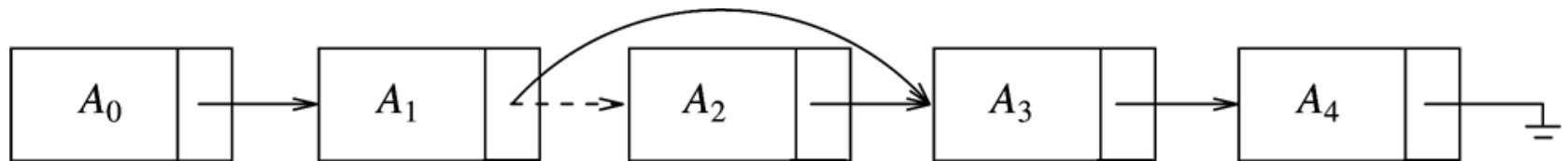
- Concept de liste
- Classes et interfaces standards
- Implémentation d'une liste par tableau
- **Implémentation d'une liste chaînée**
- Piles et files

Liste chaînée

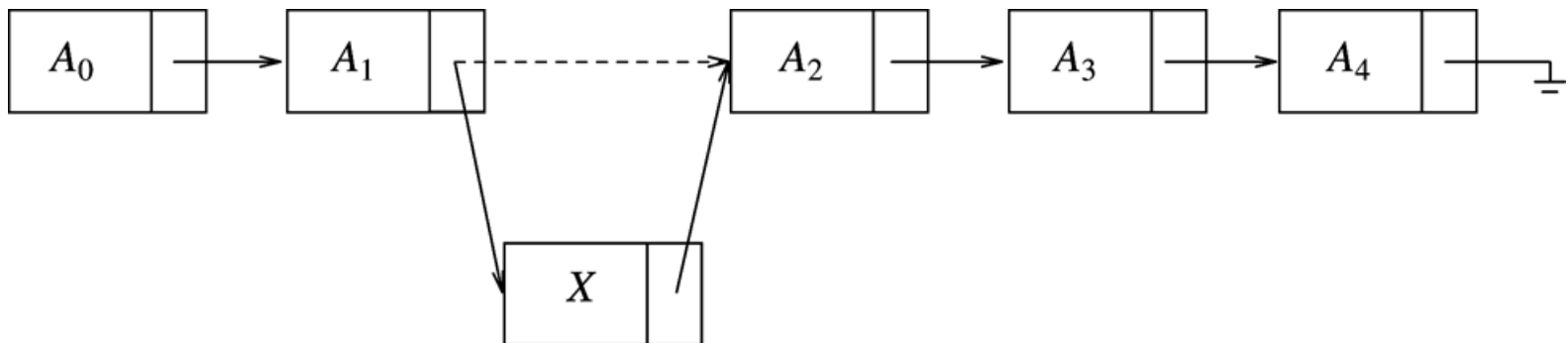


Listes chaînées (opérations)

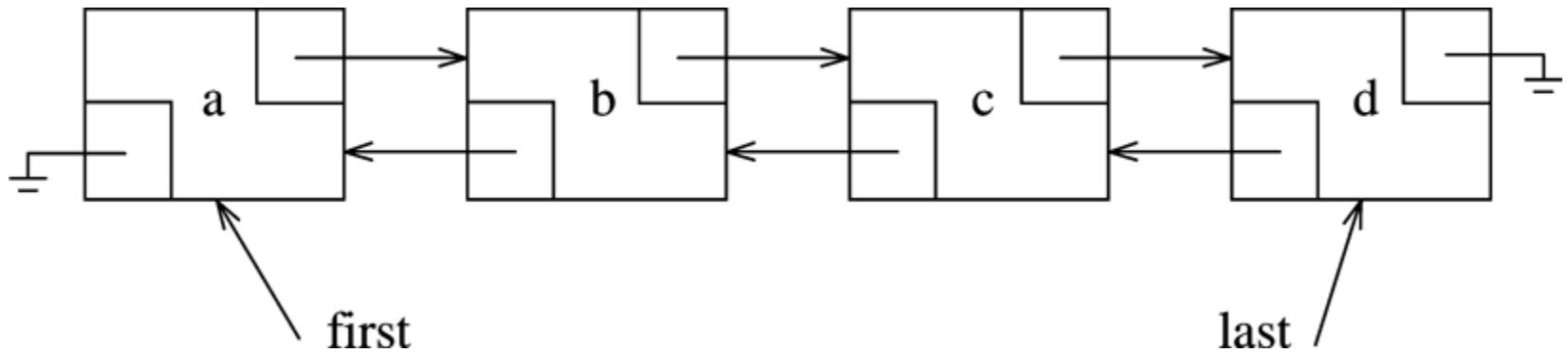
Élimination



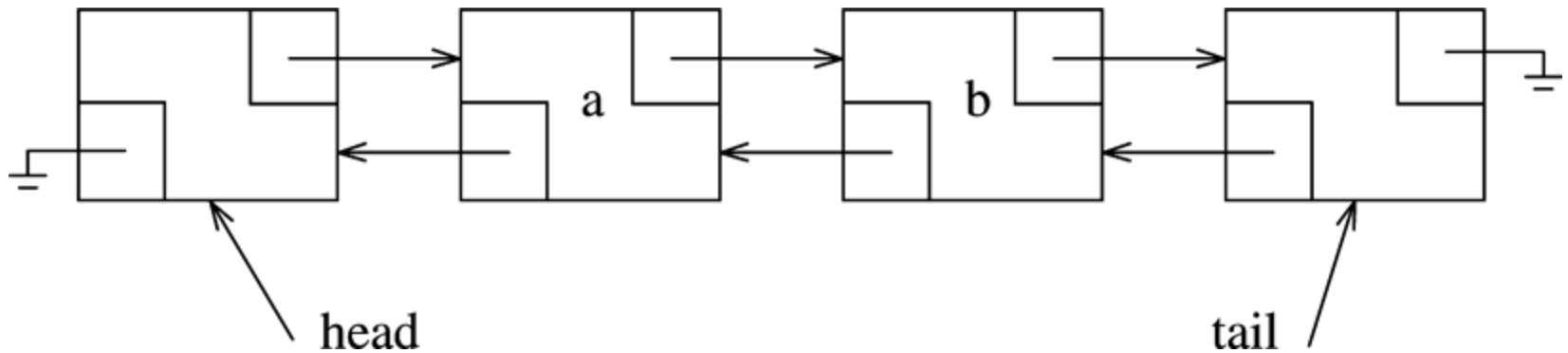
Insertion



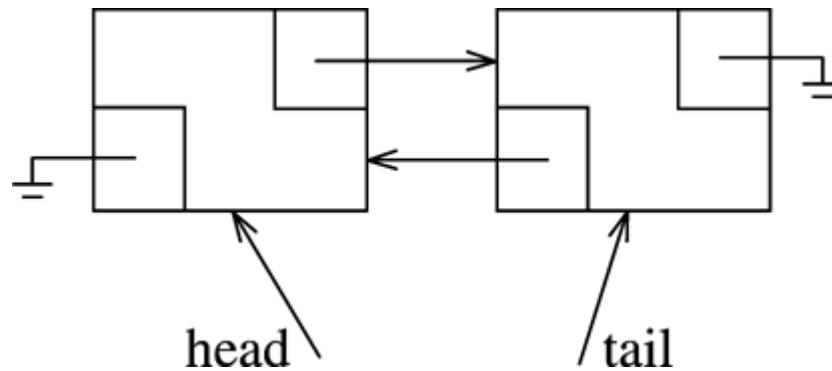
Listes doublement chaînées



« LinkedList »




« LinkedList » vide



MyLinkedList

```
public class MyLinkedList<AnyType> implements Iterable<AnyType>
{
    private int theSize;
    private int modCount = 0; // compteur de modifications
    private Node<AnyType> beginMarker;
    private Node<AnyType> endMarker;

    private static class Node<AnyType> 
    {
        public Node(AnyType d, Node<AnyType> p, Node<AnyType> n)
        { data = d; prev = p; next = n; }

        public AnyType data;
        public Node<AnyType> prev;
        public Node<AnyType> next;
    }
}
```

MyLinkedList (2)

```
public MyLinkedList( )  
{ clear( ); }  
  
public int size( )  
{ return theSize; }  
  
public boolean isEmpty( )  
{ return size() == 0; }  
  
public boolean add( AnyType x )  
{ add( size(), x ); return true; }  
  
public void add(int idx, AnyType x )  
{ addBefore( getNode( idx ), x ); }  
  
public AnyType get(int idx )  
{ return getNode( idx ).data; }
```

MyLinkedList (3)

```
public AnyType set(int idx, AnyType newVal)
{
    Node<AnyType> p = getNode( idx );
    AnyType oldVal = p.data;
    p.data = newVal;
    return oldVal;
}

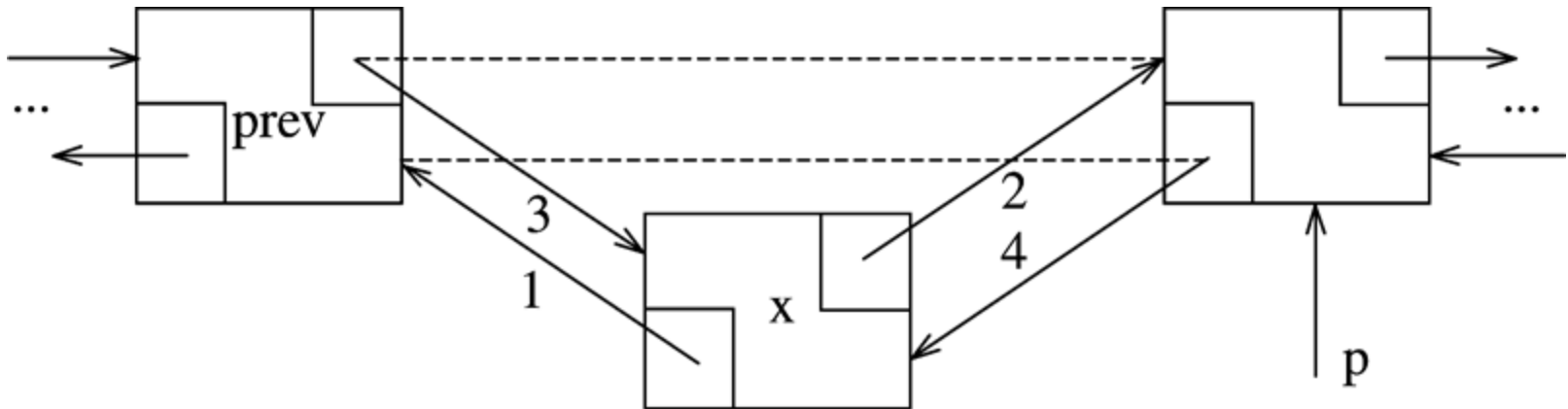
public AnyType remove(int idx )
{
    return remove( getNode( idx ) );
}
```

clear



```
public void clear()
{
    beginMarker = new Node<AnyType>(null, null, null);
    endMarker = new Node<AnyType>(null, beginMarker, null);
    beginMarker.next = endMarker;

    theSize = 0;
    modCount++;
}
```

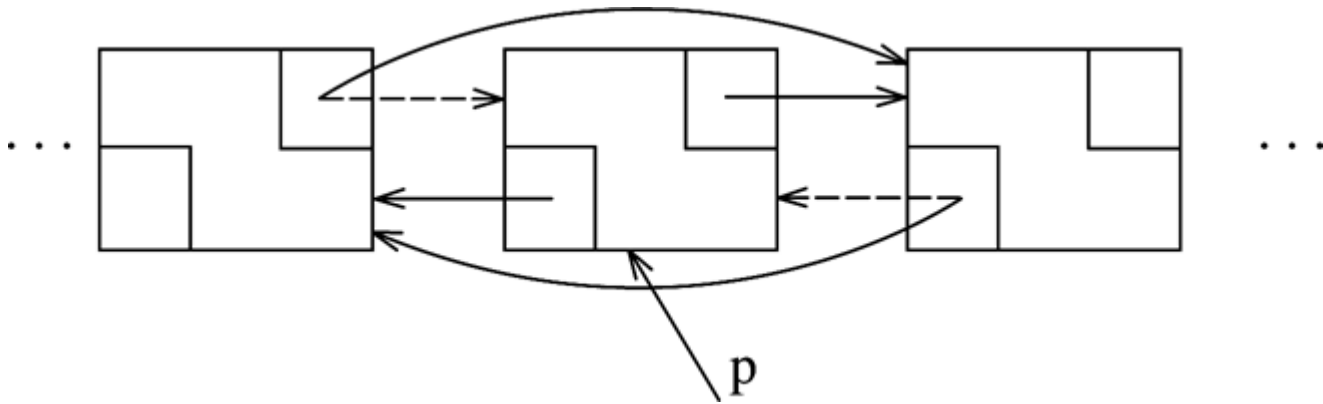

addBefore



addBefore (2)

```
private void addBefore(Node<AnyType> p, AnyType x)   
{  
    Node<AnyType> newNode = new Node<AnyType>(x, p.prev, p);  
  
    newNode.prev.next = newNode;  
    p.prev = newNode;  
  
    theSize++;  
  
    modCount++;   
}
```

remove



MyLinkedList (5)

```
private AnyType remove(Node<AnyType> p)
{
    p.next.prev = p.prev;
    p.prev.next = p.next;

    theSize--;

    modCount++;
    return p.data;
}
```



getNode


```
private Node<AnyType> getNode(int idx)
{
    Node<AnyType> p;
    if( idx < 0 || idx > size() )
        throw new IndexOutOfBoundsException();
    if( idx < size( ) / 2 ) {
        p = beginMarker.next;
        for(int i = 0; i < idx; i++ )
            p = p.next;
    }
    else {
        p = endMarker;
        for(int i = size( ); i > idx; i-- )
            p = p.prev;
    }

    return p;
}
```

LinkedListIterator

```
private class LinkedListIterator
Implements java.util.Iterator<AnyType>
{
    private Node<AnyType> current = beginMarker.next;
    private int expectedModCount = modCount;
    private boolean okToRemove = false;

    public boolean hasNext( )
    {
        return current != endMarker;
    }
}
```



Note: `expectedModCount` vérifie que l'itérateur est cohérent par rapport aux modifications

LinkedListIterator (2)

```
public AnyType next()
{
    if( modCount != expectedModCount )
        throw new java.util.ConcurrentModificationException();

    if( !hasNext() )
        throw new java.util.NoSuchElementException();

    AnyType nextItem = current.data;
    current = current.next;
    okToRemove = true; ←
    return nextItem;
}
```

Note: Il y a une erreur dans le livre. C'est bien true

LinkedListIterator (3)

```
public void remove( )
{
    if( modCount != expectedModCount )
        throw new java.util.ConcurrentModificationException( );

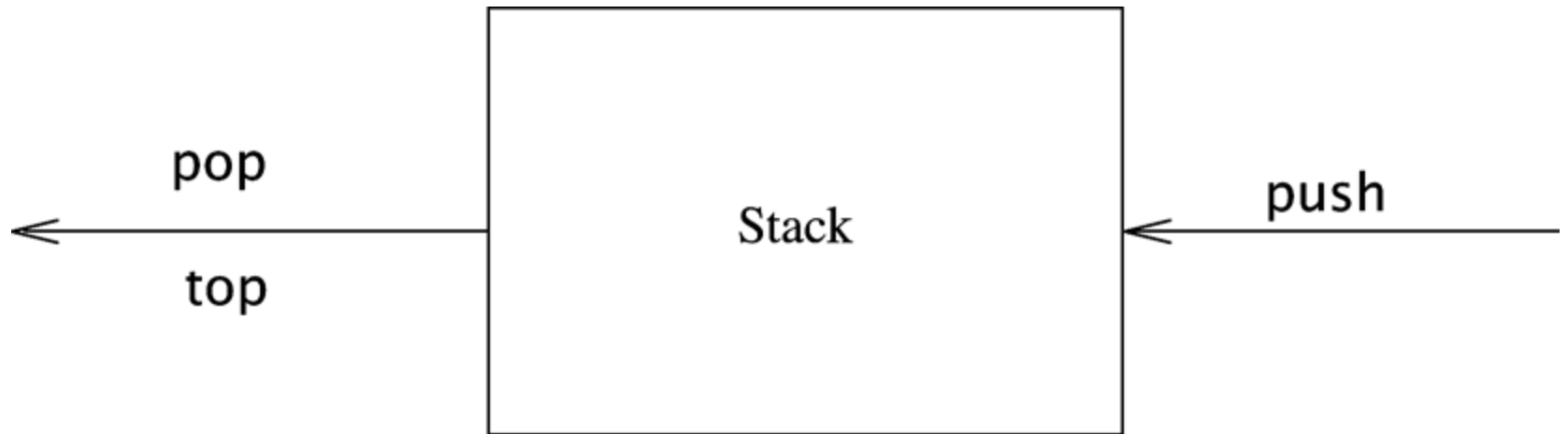
    if( !okToRemove )
        throw new IllegalStateException( );

    MyLinkedList.remove( current.prev );
    okToRemove = false;
}
} // Fin classe LinkedListIterator
```

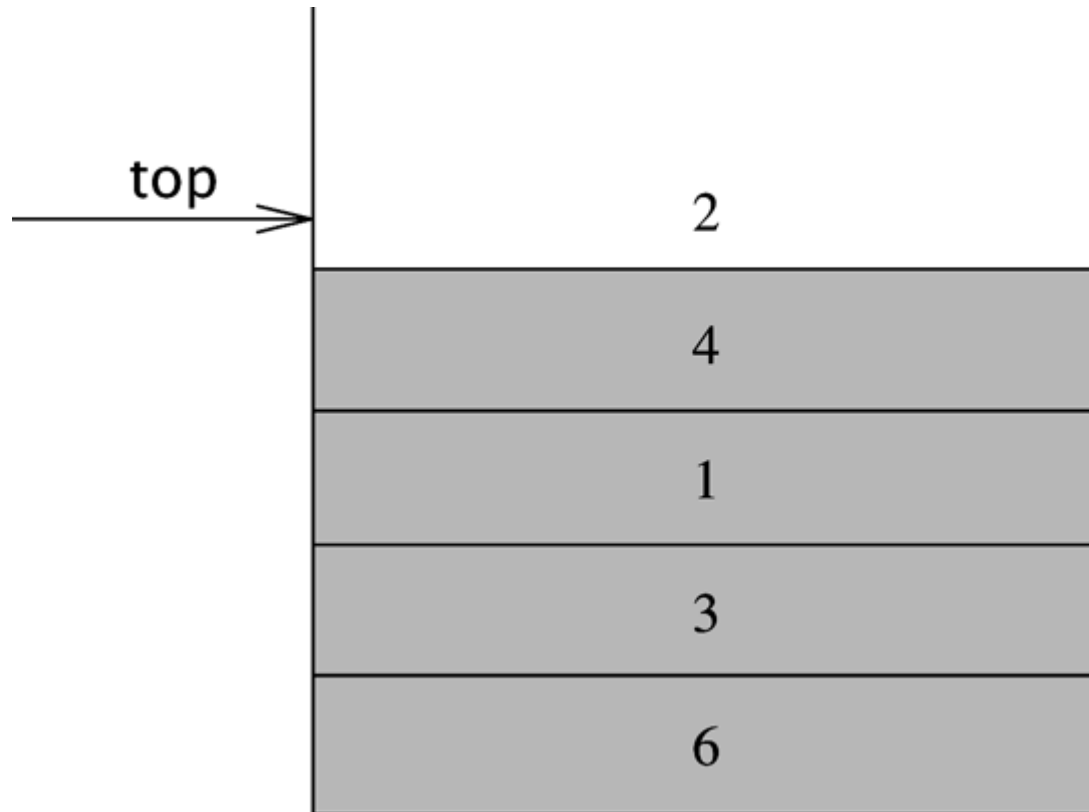

Plan

- Concept de liste
- Classes et interfaces standards
- Implémentation d'une liste par tableau
- Implémentation d'une liste chaînée
- **Piles et files**

Piles



Piles (2)



Piles (3)

- Implantation
 - ArrayList
 - LinkedList
- Applications
 - Analyse d'expressions
 - Évaluation d'expressions
 - Notation post fixe
 - Modèle de computation (activation de routines)

Expressions post-fixes

```
import java.io.*;
import java.util.*;

public class evalPostFix {

    public static void eval(char[] expr)
    {

        int index = 0;
        Integer iVal1 = null;
        Integer iVal2 = null;
        Integer auxVal = null;
        Stack st = new Stack();
```

Expressions post-fixes (2)

```
for(index=0; index<expr.length; index++) {
    System.out.println("C: " + expr[index]);

    if(expr[index] == '+') {
        iVal1 = (Integer) st.pop();
        iVal2 = (Integer) st.pop();
        auxVal = new Integer(
            iVal1.intValue()+iVal2.intValue());
        st.push(auxVal);
    } else {
        auxVal = new Integer(Character.digit(expr[index], 10));
        st.push(auxVal);
    }
} // Fin boucle for

System.out.println("SIZE: " + st.size());
System.out.println("TOP: " + st.peek());}
```

Expressions post-fixes (3)

```
public static void main(String[] args)
{
    char[] expr = {'3', '1', '4', '+',
                  '+', '0', '5', '+', '+'};

    eval(expr);
}

// Fin de la classe evalPostFix
```

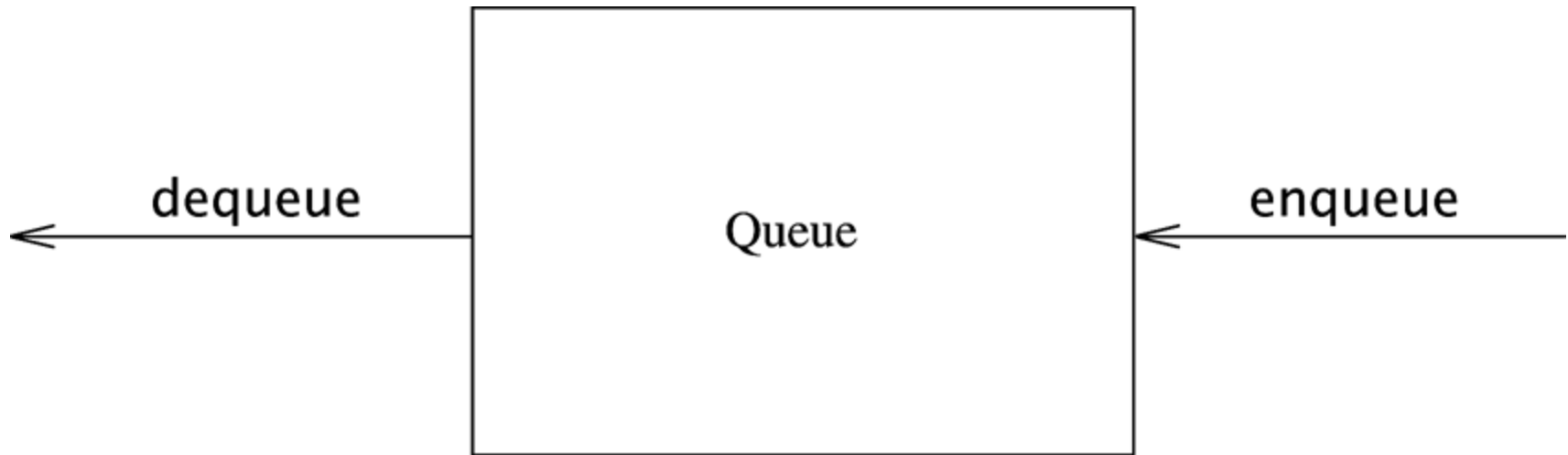
Sortie:

```
C: 3
C: 1
C: 4
C: +
C: +
C: 0
C: 5
C: +
C: +
SIZE: 1
TOP: 13
```

Exercises

- Ajouter les opérations '-' (binaire), '*' et '/'
- Ajouter '(' et ')'– Suggestion: utiliser des méthodes récursives

Files



Files (2)

- Implantation
 - ArrayList
 - LinkedList
- Applications
 - Tampons de messages (« buffers »)