

Files

Introduction aux files (d'attente)

L'interface `ca.umontreal.IFT2015.adt.Queue` et `java.util.Queue`

`ArrayQueue`

`LinkedQueue`

`CircularQueue`

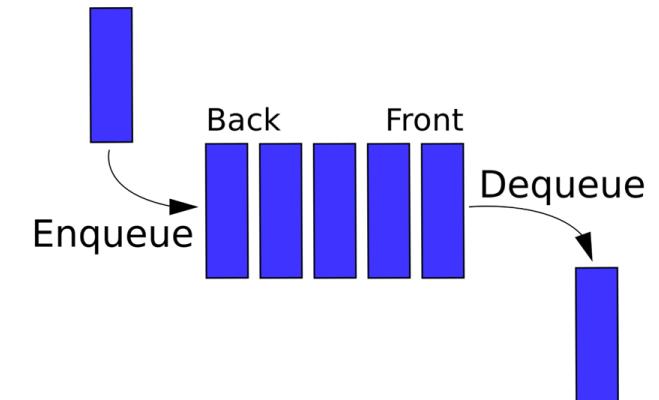
Application : Le problème de Josephus

`Deque`

`DoublyLinkedList` implémente `Deque`

Introduction aux files (d'attente)

La File est caractérisé par deux opérations :
enqueue (enfiler) et dequeue (défiler)



Pile de disques
Queue d'écoute
Inventé en 1925 par Eric Waterworth.

La File est caractérisée par sa politique de premier entré premier sorti (first-in-first-out; FIFO). On parle souvent de "buffer", par exemple d'entrées ou de sorties, de files "d'attentes".

Quelques applications

Applications directes :

- Listes d'attente, bureaucratie
- Accès aux ressources partagées (par exemple, imprimante)
- Multiprogrammation

Applications indirectes :

- Structure de données auxiliaires pour les algorithmes
- Composant d'autres structures de données

L'interface `ca.umontreal.IFT2015.adt.Queue` et l'interface `java.util.Queue`

<code>enqueue(E e)</code>	ajoute l'élément e à la fin de la file
<code>dequeue()</code>	retire et retourne le premier élément de la file, <code>null</code> si vide
<code>first()</code>	retourne le premier élément de la file, <code>null</code> si vide
<code>size()</code>	retourne le nombre d'éléments dans la file
<code>isEmpty()</code>	retourne un booléen indiquant si la file est vide

`java.util.Queue`

Method Summary

Method	Return Value	$\text{first} \leftarrow Q \leftarrow \text{last}$
<code>enqueue(5)</code>	–	(5)
<code>enqueue(3)</code>	–	(5, 3)
<code>size()</code>	2	(5, 3)
<code>dequeue()</code>	5	(3)
<code>isEmpty()</code>	false	(3)
<code>dequeue()</code>	3	()
<code>isEmpty()</code>	true	()
<code>dequeue()</code>	null	()
<code>enqueue(7)</code>	–	(7)
<code>enqueue(9)</code>	–	(7, 9)
<code>first()</code>	7	(7, 9)
<code>enqueue(4)</code>	–	(7, 9, 4)

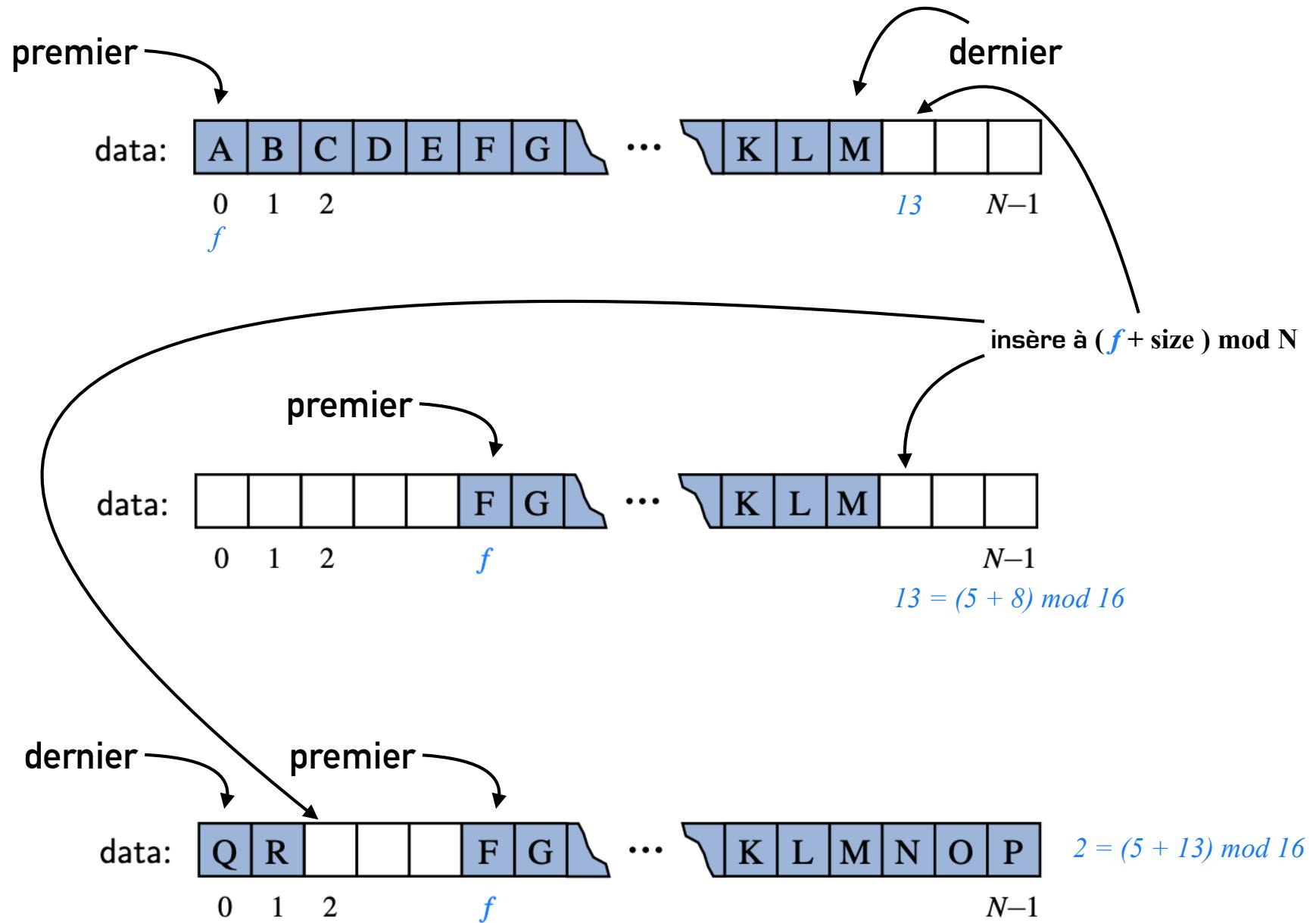
All Methods Instance Methods Abstract Methods

Modifier and Type	Method and Description
boolean	<code>add(E e)</code> Inserts the specified element into this queue if it is possible to do so immediately without violating capacity restrictions, returning <code>true</code> upon success and throwing an <code>IllegalStateException</code> if no space is currently available.
E	<code>element()</code> Retrieves, but does not remove, the head of this queue.
boolean	<code>offer(E e)</code> Inserts the specified element into this queue if it is possible to do so immediately without violating capacity restrictions.
E	<code>peek()</code> Retrieves, but does not remove, the head of this queue, or returns <code>null</code> if this queue is empty.
E	<code>poll()</code> Retrieves and removes the head of this queue, or returns <code>null</code> if this queue is empty.
E	<code>remove()</code> Retrieves and removes the head of this queue.

Voir le code...

Queue.java

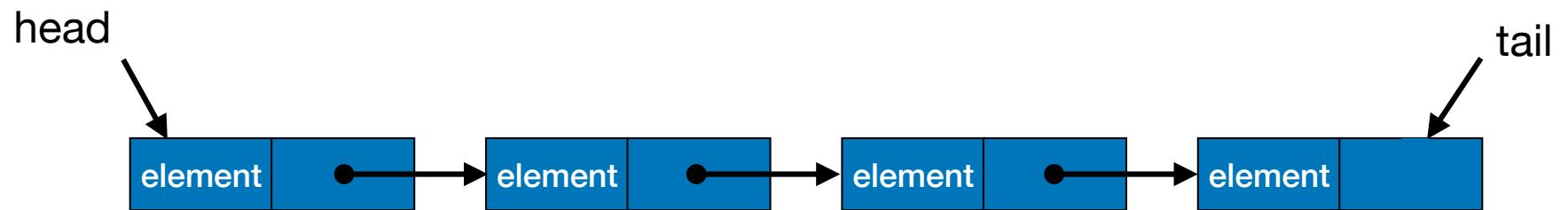
Implémentation dans un tableau circulaire



Voir le code de...

ArrayQueue.java

SinglyLinkedList



enqueue(e)	list.addLast(e)
dequeue()	list.removeFirst()
first()	list.first()
size()	list.size()
isEmpty()	list.isEmpty()

Voir le code de...

LinkedQueue.java

Une file d'attente circulaire

Une file circulaire est une spécialisation de file à laquelle on ajoute la méthode `rotate()`, *ne pas confondre liste, tableau et file circulaires !*

On peut facilement l'implémenter avec `CircularlyLinkedList` pour produire la classe `LinkedCircularQueue`

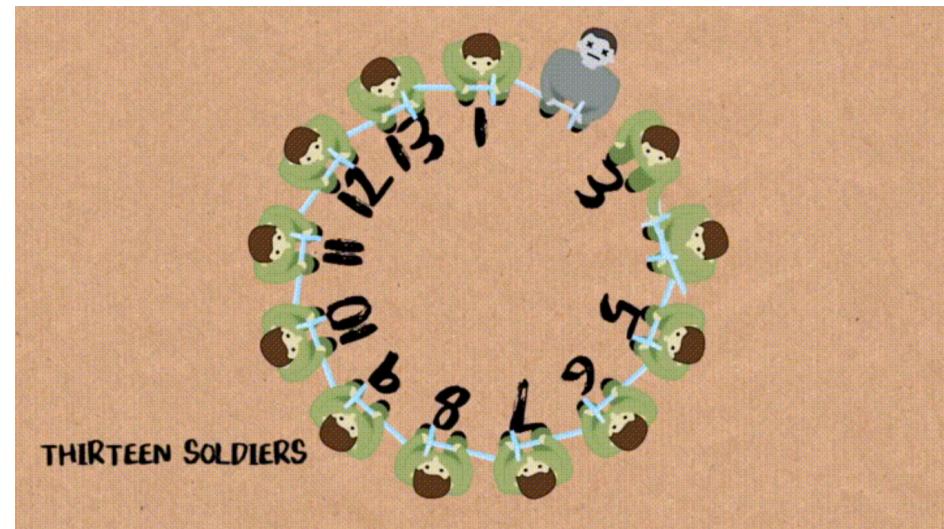
Un des avantages de `LinkedCircularQueue` sur `LinkedList` est qu'un appel à `rotate()` est une implémentation plus efficace que la combinaison `enqueue(dequeue())` puisqu'aucun noeud n'est créé, supprimé ni relié.

`CircularQueue.java`

Voir le code de...

LinkedCircularQueue.java

Problème de Josephus



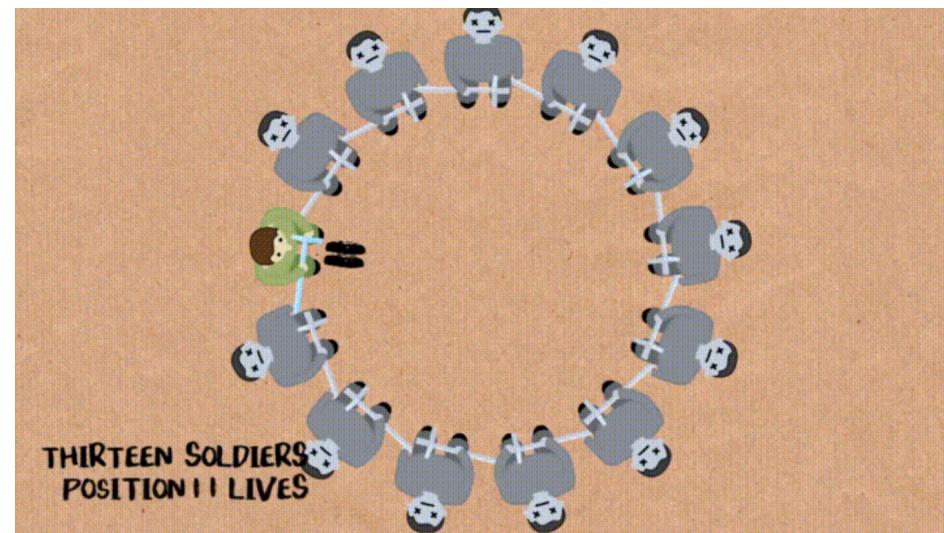
Un groupe de personnes est disposé en cercle, ici de 1 à 13. Ils décident de s'éliminer les uns après les autres selon une règle précise : en comptant jusqu'à un certain nombre, une personne est éliminée, puis on recommence à compter à partir de la personne suivante, jusqu'à ce qu'il ne reste plus qu'une seule personne.

Le défi est de trouver quelle place dans le cercle il faut occuper pour être le dernier survivant. C'est un problème de logique amusant qui fait intervenir des concepts de comptage et de cycles.

Pour le cas où on compte une personne à la fois. L'ordre de sortie pour 13 personnes est la suivante :

2, 4, 6, 8, 10, 12, 1, 5, 9, 13, 7, 3

Le gagnant est donc : 11.



Players	Winner
1	1
2	1
3	3
4	1
5	3
6	5
7	7
8	1
9	3
10	5
11	7
12	9
13	11
14	13
15	15
16	1



Voir le code de...

Josephus.java
QueueApp.java

```
String[] group1 = {"Alice", "Bob", "Toto", "Titi", "Foo", "Bar"};
System.out.println( "Winner is: " + Josephus.execute( Josephus.sit( group1 ), 3 ) );
```

```
[java] [Alice,Bob,Toto,Titi,Foo,Bar]
[java] [Titi,Foo,Bar,Alice,Bob,Toto]
[java]     Titi is out
[java] [Foo,Bar,Alice,Bob,Toto]
[java] [Bob,Toto,Foo,Bar,Alice]
[java]     Bob is out
[java] [Toto,Foo,Bar,Alice]
[java] [Alice,Toto,Foo,Bar]
[java]     Alice is out
[java] [Toto,Foo,Bar]
[java] [Toto,Foo,Bar]
[java]     Toto is out
[java] [Foo,Bar]
[java] [Bar,Foo]
[java]     Bar is out
[java] Winner is: Foo
```

De Queue à Deque

Une **Deque** (Double-Ended Queue) est une structure de données qui permet d'ajouter et de retirer des éléments à **ses deux extrémités** (à la fois au début et à la fin). Contrairement à une file classique où les éléments sont ajoutés à la fin et retirés au début, une deque offre une plus grande flexibilité.

Peut être utilisée comme une **pile** (LIFO) ou comme une **file** (FIFO).

<code>addFirst(e)</code>	ajoute l'élément e au début du deque
<code>addLast(e)</code>	ajoute l'élément e à la fin du deque
<code>removeFirst()</code>	retire et retourne le premier élément du deque, <code>null</code> si vide
<code>removeLast()</code>	retire et retourne le dernier élément du deque, <code>null</code> si vide
<code>first()</code>	retourne le premier élément du deque, <code>null</code> si vide
<code>last()</code>	retourne le dernier élément du deque, <code>null</code> si vide
<code>size()</code>	retourne le nombre d'éléments dans le deque
<code>isEmpty()</code>	retourne un booléen indiquant si le deque est vide ou non

Method	Return Value	D
<code>addLast(5)</code>	–	(5)
<code>addFirst(3)</code>	–	(3, 5)
<code>addFirst(7)</code>	–	(7, 3, 5)
<code>first()</code>	7	(7, 3, 5)
<code>removeLast()</code>	5	(7, 3)
<code>size()</code>	2	(7, 3)
<code>removeLast()</code>	3	(7)
<code>removeFirst()</code>	7	()
<code>addFirst(6)</code>	–	(6)
<code>last()</code>	6	(6)
<code>addFirst(8)</code>	–	(8, 6)
<code>isEmpty()</code>	false	(8, 6)
<code>last()</code>	6	(8, 6)

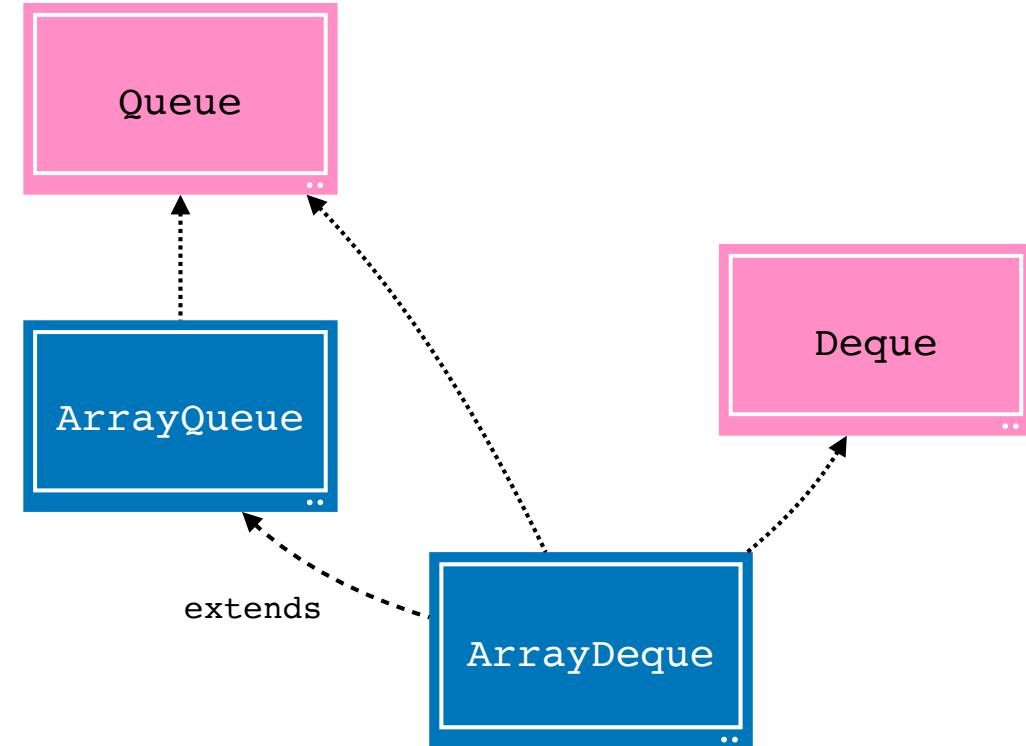
`java.util.Deque`

<https://docs.oracle.com/javase/8/docs/api/java/util/Deque.html>

Voir le code de...

Deque.java

ArrayDeque.java



Arithmétique modulo pour addFirst et addLast

addFirst[e]:

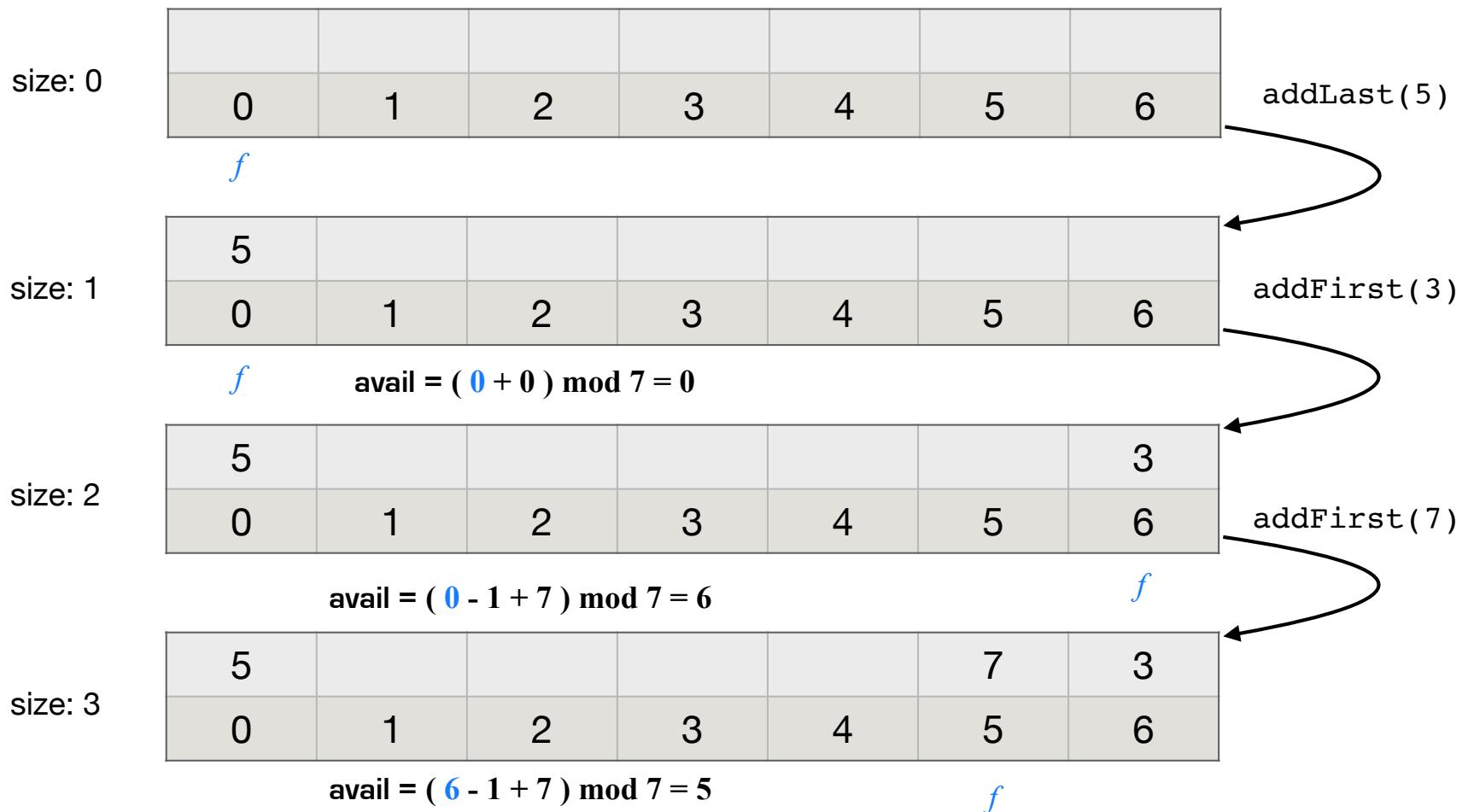
$$\text{avail} = (\text{f} - 1 + N) \bmod N$$

$\text{f} = \text{avail}$

addLast[e]:

enqueue(e):

$$\text{avail} = (\text{f} + \text{size}) \bmod N$$



Le dernier élément du Deque

$$\text{last} = (\text{f} + \text{size} - 1) \bmod N$$

$$\text{last} = (5 + 3 - 1) \bmod 7$$

$$\text{last} = 0$$

f: 5
size: 3

5					7	3
0	1	2	3	4	5	6

f

DoublyLinkedList implémente toutes les opérations de Deque

```
package ca.umontreal.IFT2015.adt.list;

import java.util.NoSuchElementException;
import java.lang.IndexOutOfBoundsException;
import java.util.Iterator;

import ca.umontreal.IFT2015.adt.queue.Deque;

/**
 * DoublyLinkedList is an implementation of the list ADT
 *   additional operation removeLast
 *   use doubly linked Node
 *   use header and trailer sentinel (dummy Nodes)
 *
 * Based on Goodrich, Tamassia, Goldwasser
 *
 * @author    Francois Major
 * @version   1.0
 * @since    1.0
 */
public class DoublyLinkedList<E> implements List<E>, Deque<E> {
```

Remarques conclusives

- On a regardé ce qu'est une file (d'attente) et quelques applications.
- On a défini l'ADT et une interface Queue (qu'on a comparé avec l'interface `java.util.Queue`).
- On a implémenté l'interface Queue avec un Array (circulaire) puis avec une `SinglyLinkedList` et une `CircularlyLinkedList`
- Toutes les opérations s'exécutent en $O(1)$
- On a vu que l'avantage de `CircularLinkedList` est l'utilisation de la méthode `rotate()` plutôt que `enqueue()` / `dequeue()` pour remettre le premier élément de la liste en file d'attente.
- On a regardé une application : Le problème de Josephus.
- On a vu le Deque, une version étendue de la file (d'attente) avec insertion et suppression au début et à la fin, et ses implémentations dans un tableau...
- ... et dans une liste doublement chaînée, qui, en fait, l'implémente.