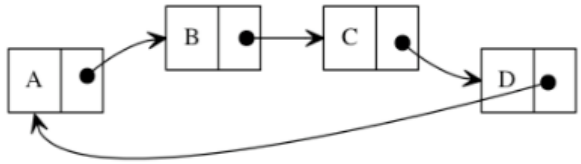
PART 1 - Circular linkedlist (Implem)

On s'intéresse à l'implémentation d'une `liste simplement chaînée circulaire`, c'est-à-dire une liste pour laquelle la dernière position de la liste fait référence, comme position suivante, à la première position de la liste.



L'ajout d'un nouvel élément dans la file (méthode `enqueue`) se fait en fin de liste et le retrait (méthode `remove`) se fait a un *index* particulier de la liste. Une (seule) référence sur la fin de la liste (`last`) est nécessaire pour effectuer toutes les opérations sur cette file.

Il vous est donc demander d'implémenter cette liste simplement chaînée circulaire à partir de la classe `CircularLinkedList.java` où vous devez completer (*TODO STUDENT*):

- la méthode d'ajout (`enqueue`);
- la méthode de retrait (`remove`) [L'exception `IndexOutOfBoundsException` est lancée quand la valeur de l'index n'est pas comprise en 0 et size()-1];
- l'itérateur (`ListIterator`) qui permet de parcourir la liste en FIFO.

Attention: L'itérateur devra lancer des exceptions dans les cas suivants:

- étant donnée que le `remove` est optionnel dans l'API , l'iterateur devra juste lancer un `UnsupportedOperationException` en cas d'appel du `remove`;
- étant donnée qu'on ne peut modifier l'itérateur alors qu'on est en train d'itérer, l'iterateur devra lancer un `ConcurrentModificationException` dans ce cas dans le `next` et le `hasNest`;
- si le `next` est appelé alors qu'il n'y a plus de prochain élément, l'iterateur devra lancer un `NoSuchElementException`.

```
import java.util.ConcurrentModificationException;
import java.util.Iterator;
import java.util.NoSuchElementException;

public class CircularLinkedList<Item> implements Iterable<Item> {
 private long nOp = 0; // count the number of operations
 private int n;           // size of the stack
 private Node  last;    // trailer of the list

 // helper linked list class
 private class Node {
     private Item item;
     private Node next;
 }

 public CircularLinkedList() {
     last = null;
     n = 0;
 }

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

 public int size() { return n; }

 private long nOp() { return nOp; }

 /**
  * Append an item at the end of the list
  * @param item the item to append
  */
```

```java
 public void enqueue(Item item) {
     // TODO STUDENT: Implement add method
 }

 /**
  * Removes the element at the specified position in this list.
  * Shifts any subsequent elements to the left (subtracts one from their
indices).
  * Returns the element that was removed from the list.
  */
 public Item remove(int index) {
     // TODO STUDENT: Implement remove method
 }

 /**
  * Returns an iterator that iterates through the items in FIFO order.
  * @return an iterator that iterates through the items in FIFO order.
  */
 public Iterator<Item> iterator() {
     return new ListIterator();
 }

 /**
  * Implementation of an iterator that iterates through the items in FIFO
order.
  *
  */
 private class ListIterator implements Iterator<Item> {
     // TODO STUDENT: Implement the ListIterator
 }

}
```

Le projet IntelliJ est disponible ici.

Your answer passed the tests! Your score is 100.0%. [Submission #5d90e65c6779dd2b0302646d]

Test	Status	Grade	Comment
class src.CircularLinkedListTestExtreme		**25/25**	
→ testConcurrentModificationNext(src.CircularLinkedListTestExtreme)	✅ Success	5/5	
→ testIteratorList(src.CircularLinkedListTestExtreme)	✅ Success	15/15	
→ testOutOfBound(src.CircularLinkedListTestExtreme)	✅ Success	5/5	
class src.CircularLinkedListTestRandom		**25/25**	
→ runAsExpected[0](src.CircularLinkedListTestRandom)	✅ Success	0.5/0.5	
→ runAsExpected[1](src.CircularLinkedListTestRandom)	✅ Success	0.5/0.5	
→ runAsExpected[2](src.CircularLinkedListTestRandom)	✅ Success	0.5/0.5	

→ runAsExpected[3](src.CircularLinkedListTestRandom)	✅ Success	0.5/0.5	
→ runAsExpected[4](src.CircularLinkedListTestRandom)	✅ Success	0.5/0.5	
→ runAsExpected[5](src.CircularLinkedListTestRandom)	✅ Success	0.5/0.5	
→ runAsExpected[6](src.CircularLinkedListTestRandom)	✅ Success	0.5/0.5	
→ runAsExpected[7](src.CircularLinkedListTestRandom)	✅ Success	0.5/0.5	
→ runAsExpected[8](src.CircularLinkedListTestRandom)	✅ Success	0.5/0.5	
→ runAsExpected[9](src.CircularLinkedListTestRandom)	✅ Success	0.5/0.5	
→ runAsExpected[10](src.CircularLinkedListTestRandom)	✅ Success	0.5/0.5	
→ runAsExpected[11](src.CircularLinkedListTestRandom)	✅ Success	0.5/0.5	
→ runAsExpected[12](src.CircularLinkedListTestRandom)	✅ Success	0.5/0.5	
→ runAsExpected[13](src.CircularLinkedListTestRandom)	✅ Success	0.5/0.5	
→ runAsExpected[14](src.CircularLinkedListTestRandom)	✅ Success	0.5/0.5	
→ runAsExpected[15](src.CircularLinkedListTestRandom)	✅ Success	0.5/0.5	
→ runAsExpected[16](src.CircularLinkedListTestRandom)	✅ Success	0.5/0.5	
→ runAsExpected[17](src.CircularLinkedListTestRandom)	✅ Success	0.5/0.5	
→ runAsExpected[18](src.CircularLinkedListTestRandom)	✅ Success	0.5/0.5	
→ runAsExpected[19](src.CircularLinkedListTestRandom)	✅ Success	0.5/0.5	
→ runAsExpected[20](src.CircularLinkedListTestRandom)	✅ Success	0.5/0.5	
→ runAsExpected[21](src.CircularLinkedListTestRandom)	✅ Success	0.5/0.5	
→ runAsExpected[22](src.CircularLinkedListTestRandom)	✅ Success	0.5/0.5	
→ runAsExpected[23](src.CircularLinkedListTestRandom)	✅ Success	0.5/0.5	
→ runAsExpected[24](src.CircularLinkedListTestRandom)	✅ Success	0.5/0.5	

→ runAsExpected[25](src.CircularLinkedListTestRandom)	✅ Success	0.5/0.5	
→ runAsExpected[26](src.CircularLinkedListTestRandom)	✅ Success	0.5/0.5	
→ runAsExpected[27](src.CircularLinkedListTestRandom)	✅ Success	0.5/0.5	
→ runAsExpected[28](src.CircularLinkedListTestRandom)	✅ Success	0.5/0.5	
→ runAsExpected[29](src.CircularLinkedListTestRandom)	✅ Success	0.5/0.5	
→ runAsExpected[30](src.CircularLinkedListTestRandom)	✅ Success	0.5/0.5	
→ runAsExpected[31](src.CircularLinkedListTestRandom)	✅ Success	0.5/0.5	
→ runAsExpected[32](src.CircularLinkedListTestRandom)	✅ Success	0.5/0.5	
→ runAsExpected[33](src.CircularLinkedListTestRandom)	✅ Success	0.5/0.5	
→ runAsExpected[34](src.CircularLinkedListTestRandom)	✅ Success	0.5/0.5	
→ runAsExpected[35](src.CircularLinkedListTestRandom)	✅ Success	0.5/0.5	
→ runAsExpected[36](src.CircularLinkedListTestRandom)	✅ Success	0.5/0.5	
→ runAsExpected[37](src.CircularLinkedListTestRandom)	✅ Success	0.5/0.5	
→ runAsExpected[38](src.CircularLinkedListTestRandom)	✅ Success	0.5/0.5	
→ runAsExpected[39](src.CircularLinkedListTestRandom)	✅ Success	0.5/0.5	
→ runAsExpected[40](src.CircularLinkedListTestRandom)	✅ Success	0.5/0.5	
→ runAsExpected[41](src.CircularLinkedListTestRandom)	✅ Success	0.5/0.5	
→ runAsExpected[42](src.CircularLinkedListTestRandom)	✅ Success	0.5/0.5	
→ runAsExpected[43](src.CircularLinkedListTestRandom)	✅ Success	0.5/0.5	
→ runAsExpected[44](src.CircularLinkedListTestRandom)	✅ Success	0.5/0.5	
→ runAsExpected[45](src.CircularLinkedListTestRandom)	✅ Success	0.5/0.5	
→ runAsExpected[46](src.CircularLinkedListTestRandom)	✅ Success	0.5/0.5	

Information

Author(s)	Pierre Schaus, John Aoga
Deadline	01/10/2019 23:55:00
Status	Succeeded
Grade	100.0%
Grading weight	1.0
Attempts	3

→ runAsExpected[47](src.CircularLinkedListTestRandom)	✅ Success	0.5/0.5	
→ runAsExpected[48](src.CircularLinkedListTestRandom)	✅ Success	0.5/0.5	
→ runAsExpected[49](src.CircularLinkedListTestRandom)	✅ Success	0.5/0.5	
TOTAL		**50/50**	
TOTAL WITHOUT IGNORED		**50/50**	

Question 1: Implementation de la fonction ajout: void enqueue(Item item)

```
/**
 * Append an item at the end of the list
 * @param item the item to append
 */
public void enqueue(Item item) {
    // TODO STUDENT: Implement add method
}
```

Copier le contenu de la fonction `public void enqueue(Item item)` ci-desssous.

```
if (n==0){
            last=new Node();
            last.item=item;
            last.next=last;
        }
        else{
            Node premier=last.next;
            last.next=new Node();
            last.next.item=item;
            last.next.next=premier;
            last=last.next;
        }
        nOp++;
        n++;
```

Submission limit	No limitation

Submitting as

Gauthier de Moffarts d'H.

Teams management

Project Generator

Download IntelliJ Project

For evaluation

Best submission

29/09/2019 19:14:04 - 100.0%

Submission history

29/09/2019 19:14:04 - 100.0%

27/09/2019 13:48:51 - 0.0%

27/09/2019 11:59:24 - 0.0%

Question 2: Implementation de la fonction ajout: Item remove(int index)

```
/**
 * Removes the element at the specified position in this list.
 * Shifts any subsequent elements to the left (subtracts one from
their indices).
 * Returns the element that was removed from the list.
 */
public Item remove(int index) {
    // TODO STUDENT: Implement remove method
}
```

Copier le contenu de la fonction `public Item remove(int index)` ci-desssous.

```
if(index < 0 || index > n-1){
            throw new IndexOutOfBoundsException();
        }
        Node iter = last;
        for(int i = 0; i < index; i++){
            iter=iter.next;//il va s'arreter pile avant le node a retirer
        }
        Item theItem = iter.next.item;
        iter.next=iter.next.next;//plus de lien vers le node a supprimer sauf le last si c'est lui!!/!\
        if (index==n-1){
            last=iter;
        }
        if (n==1){
            last=null;
        }
        nOp++;
        n--;
        return theItem;
```

Question 3: Implementation de l'iterateur: ListIterator

```
/**
 * Implementation of an iterator that iterates through the items in
FIFO order.
 *
 */
private class ListIterator implements Iterator<Item> {
    // TODO STUDENT: Implement the ListIterator
}
```

Copier le contenu de la class `private class ListIterator implements Iterator<Item>` ci-desssous.

```java
private Node iter;
        int fait;
        public ListIterator(){
            if(last!=null){
                iter=last.next;
            } else{
                iter=null;
            }
            nOp=0;
            fait=0;
        }

        public boolean hasNext()throws ConcurrentModificationException{
            if(nOp>0){
                throw new ConcurrentModificationException();
            }
            /*
            if ( iter == last ){
                return false;
            }
            return true; on ne peut pas faire ceci car on compare des objets et on est pas en c!!! */
            else if(fait==n){
                return false;
            }
            else{
                return true;
            }
        }

        public Item next() throws  NoSuchElementException{
            if(nOp>2){
                throw new ConcurrentModificationException();
            }
            else if ( !hasNext() ){
                throw new  NoSuchElementException();
            }
            else{
                Item donnee = iter.item;
                iter=iter.next;
                fait++;
                return donnee;
            }
        }

        public void remove() throws UnsupportedOperationException {
            throw new UnsupportedOperationException();
        }
```