Compte Rendu TP6

BOURENNANE Amine

Exercice 1

1. On peut regarder avec un champs auxiliaire `size` qui contient la taille de la liste.
2.

```
public class Fifo<E> implements Iterable<E> {
    private final E[] elements;
    private int size;
    private int head;
    private int tail;

    public Fifo(int capacity) {
        if (capacity <= 0)
            throw new IllegalArgumentException("need capacity > 0");
        @SuppressWarnings("unchecked")
        E[] elements = (E[]) new Object[capacity];
        this.elements = elements;
    }
}
```

3. On regarde si la taille n'est pas nulle et que les pointeurs de tête et de queue ne sont pas égaux, on peut renvoyer une exception.

```
public void offer(E elem) {
    Objects.requireNonNull(elem);
    if (tail == head && size != 0)
        throw new IllegalStateException("no more capacity");
    elements[tail] = elem;
    tail = (tail + 1) % elements.length;
    size++;
}
```

4. On peut aussi renvoyer une exception.

```
public E poll() {
    if (tail == head && size == 0)
        throw new IllegalStateException("is empty");
    var last_head = elements[head];
    elements[head] = null;
    head = (head + 1) % elements.length;
    size--;
```

```
    return last_head;
}
```

5.

```
@Override
public String toString() {
    var ret = new StringJoiner(", ", "[", "]");
    var h =  head;
    for (var i = 0; i < size; i++) {
        ret.add(elements[h].toString());
        h = (h + 1) % elements.length;
    }
    return ret.toString();
}
```

6. Un *memory leak* en java c'est quand une adresse qui n'est plus utile est toujours présente et n'est pas supprimé dans le **CarbageCollector**
7.

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

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

8. Le principe d'un iterateur est qu'il garantit le parcour de la collection en O(n) où n est la taille de la collection.
9.

```
public Iterator<E> iterator() {
    return new Iterator<>() {
        private int i = 0;
        private int h = head;

        @Override
        public boolean hasNext() {
            return i < size;
        }

        @Override
        public E next() {
            if (!hasNext())
                throw new NoSuchElementException("no next");
            var elem = elements[h];
```

```
            i++;
            h = (h + 1) % elements.length;
            return elem;
        }
    };
}
```

10. L'interface *Iterable* permet de faire un **for element** en appelant en secret la méthode iterator de la classe et parcours les éléments.

Exercice 2

1. On peut déplacer la tête à 0 et concaterner la queue à la suite de celle-ci.

```
public class ResizeableFifo<E> implements Iterable<E> {
    private E[] elements;
    private int size;
    private int head;
    private int tail;

    public ResizeableFifo(int capacity) {
        if (capacity <= 0)
            throw new IllegalArgumentException("need capacity > 0");
        @SuppressWarnings("unchecked")
        E[] elements = (E[]) new Object[capacity];
        this.elements = elements;
    }

    public void offer(E elem) {
        Objects.requireNonNull(elem);
        if (size == elements.length)
            resizeElements();
        elements[tail] = elem;
        tail = (tail + 1) % elements.length;
        size++;
    }

    private void resizeElements() {
        @SuppressWarnings("unchecked")
        E[] copy = (E[]) new Object[size * 2];
        System.arraycopy(elements, head, copy, 0, size - head);
        System.arraycopy(elements, 0, copy, head, tail);
        elements = copy;
        head = 0;
        tail = size;
    }
}
```

2. Il faut réimplémenter les méthodes : `clear()`, `retainAll()`, `removeAll()`, `addAll()`, `containsAll()`, `remove()`, `add()`, `contains()`, `toArray()`, `peek()` ; les méthodes qui doivent

être modifiées sont offer et poll