JAVA AVANCEE

Faites la queue

TP5

Saravanane MANICOME

IINFO 2020 - 2021

Saravanane MANICOME

Exercice 1

Question 1

Pour palier ce problème on peut ajouter un champ size dans la classe Fifo.

Question 2

```
public class Fifo<E> implements Iterable<E> {
    private int size = 0;
    private int headIndex = 0;
    private int tailIndex = 0;
    private final int capacity;
    private final E[] array;

    @SuppressWarnings("SafeIgnore")
    public Fifo(int capacity) {
        if (capacity <= 0) {
            throw new IllegalArgumentException("arraySize must be greater than 0");
        }

        this.capacity = capacity;
        array = (E[]) new Object[capacity];
    }
}
```

Question 3

```
public void offer(E value) {
    requireNonNull(value);
    if(size == capacity) {
        throw new IllegalStateException("Capacity limit reached");
    }
    array[tailIndex] = value;
    tailIndex = (tailIndex + 1) % capacity;
    size++;
}
```

Pour savoir si la file est pleine on s'appuie sur le champ size que l'on compare avec la capacité de la file.
Si la file est pleine on jette une IllegalStateException.

Saravanane MANICOME

Question 4

```java
public E poll() {
    if (size == 0) {
        throw new IllegalStateException("Fifo is empty");
    }

    E value = array[headIndex];
    headIndex = (headIndex + 1) % capacity;
    size--;
    return value;
}
```

Si la file est pleine on jette une IllegalStateException.

Question 5

```java
@Override
public String toString() {
    StringJoiner stringJoiner = new StringJoiner( delimiter: ", ", prefix: "[", suffix: "]");
    for (int i = 0; i < size; i++) {
        var index = (headIndex + i) % capacity;
        stringJoiner.add(array[index].toString());
    }
    return stringJoiner.toString();
}
```

Question 6

Un memory leak est une fuite de mémoire générée lorsque qu'un objet ne peut être considéré par le garbage collector dû à une référence à cette objet toujours présente.

Question 7

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

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

Saravanane MANICOME

Question 8

Comme son nom l'indique un itérateur permet d'itérer les éléments contenu dans un ensemble de données, selon une logique donnée sans calculer au préalable le résultat.

Question 9

```
public Iterator<E> iterator() {
    return new Iterator<E>() {
        private int futur = headIndex;
        private int size = Fifo.this.size;

        @Override
        public boolean hasNext() {
            return size > 0;
        }

        public E next() {
            if(size == 0) {
                throw new NoSuchElementException();
            }
            var value :E = array[futur];
            futur = ( futur + 1) % capacity;
            size--;
            return value;
        }
    };
}
```

Question 10

```
public class Fifo<E> implements Iterable<E>
```

Saravanane MANICOME

Exercice 2

Question 1

```java
@SuppressWarnings("SafeIgnore")
private void grow() {
    size++;
    if(size != capacity)
        return;

    Iterator<E> iterator = iterator();
    var tmp :E[] = (E[]) new Object[capacity * 2];
    for (int i = 0; iterator.hasNext(); i++) {
        tmp[i] = iterator.next();
    }

    array = tmp;
    capacity = capacity * 2;
    headIndex = 0;
    tailIndex = size;
}
```

Question 2

1. La seule méthode supplémentaire à implanter est peek.
2. La signature de offer doit être modifiée pour retouner true si l'ajout a fonctionné et poll doit retourner null si la file est vide.
3. De mon côté, je n'ai rien trouvé à enlever

```java
public class ResizeableFifo<E> extends AbstractQueue<E>
```

Saravanane MANICOME

```java
public boolean offer(E value) {
    requireNonNull(value);
    if(size == capacity) {
        throw new IllegalStateException("Capacity limit reached");
    }
    array[tailIndex] = value;
    tailIndex = (tailIndex + 1) % capacity;
    grow();
    return true;
}
```

```java
public E poll() {
    if (size == 0) {
        return null;
    }

    E value = array[headIndex];
    headIndex = (headIndex + 1) % capacity;
    size--;
    return value;
}
```

```java
@Override
public E peek() {
    if(size == 0)
        throw new NoSuchElementException();
    return array[headIndex];
}
```

Saravanane MANICOME

Question 2

```
public boolean contains(Object value) {
    Objects.requireNonNull(value);
    Entry<T> entry = entries[getIndex(value.hashCode())];
    for (Entry<T> i = entry; i != null ; i = i.next()) {
        if(i.value().equals(value))
            return true;
    }

    return false;
}
```

On utilise un Object plutôt que E car nous ne souhaitons qu'utiliser les méthodes equals et hashcode qui sont présentent dans Object et toutes les classes qui l'héritent.

Question 3

```
public void add(T value) {
    Objects.requireNonNull(value);
    if(!contains(value)) {
        int index = getIndex(value.hashCode());
        Entry<T> entry = entries[index];
        entries[index] = new Entry<T>(value, entry);
        increaseSize();
    }
}

@SuppressWarnings("SafeIgnore")
private void increaseSize() {
    size++;
    if(size < (bucketsNumber / 2))
        return;

    Object[] valueList = new Object[size];
    AtomicInteger index = new AtomicInteger( initialValue: 0);
    forEach((T value) -> {
        valueList[index.getAndIncrement()] = value;
    });

    bucketsNumber = bucketsNumber * 2;
    entries = (Entry<T>[]) new Entry[bucketsNumber];

    size = 0;
    for(Object value : valueList) {
        add((T) value);
    }
}
```

Saravanane MANICOME

Conclusion

L’implantation de la file avec l’usage de type paramétré rajoute de la complexité par rapport au TP similaire que j’ai fais au DUT. Finalement le fonctionnement est le même mais avec quelques modifications.
En sommes pour moi ce TP est une formalité, bien le fait qu’il puisse y avoir des défauts. Je n’ai pas eu de mal à valider les tests ou comprendre ce qui était demandé.