Panduan dalam mengimplementasikan Stack dan Queue. Silakan Anda membuat kode implementasi dari latihan ini, dan hasilnya simpan sebagai library program Anda.

Jika Anda ingin bisa/sukses dalam kuliah SDA ini, rajinlah mengimplementasikan ADT dan mengerjakan soal-soal yang diberikan.

Diberikan interface **Stack** berikut:

```
public interface Stack<E> {

	/**
	 * Kembalikan true jika Stack kosong, dan False
	 * jika sebaliknya.
	 */
	public boolean isEmpty();

	/**
	 * dalam kondisi Stack apapun, force Stack kembali
	 * menjadi kondisi Kosong !
	 */
	public void makeEmpty();

	/**
	 * peek ! kembalikan elemen yang ada pada TOP tanpa
	 * menghapus elemen tersebut. Perhatikan ketika kondisi
	 * stack kosong.
	 */
	public E top();

	/**
	 * lakukan pop ! tanpa mengembalikan elemen yang sudah
	 * dikeluarkan. Perhatikan ketika kondisi stack kosong
	 */
	public void pop();

	/**
	 * gabungan top() dan pop()
	 * hapus elemen pada posisi TOP, lalu method akan mengembalikan
	 * elemen yang baru saja dihapus dari TOP tersebut.
	 */
	public E topAndPop();

	/**
	 * tambah elemen baru pada TOP -> push
	 * khusus untuk implementasi Array: lakukan array doubling jika sudah hampir
	 * penuh !
	 */
	public void push(E x);
}
```

Implementasikan Stack menggunakan **Array** (Space tidak muat, silakan isi di tempat yang kosong):

```
import java.util.EmptyStackException;         //digunakan untuk handle Stack Kosong
import java.util.Arrays;

public class MyArrayStack<E> implements Stack<E> {

	private E [] array;
	private int topOfStack;
	private static final int DEFAULT_CAPACITY = 5;

	public MyArrayStack() {
		array = (E[]) new Object[DEFAULT_CAPACITY];
		topOfStack = -1;
	}

	public boolean isEmpty() {

	}

	public void makeEmpty() {

	}

	public E top() {









	}

	public void pop() {









	}

	public E topAndPop() {









	}

	public void push(E x) {









	}
```

```
	/**
	 * method untuk melakukan Array Doubling.
	 * akan digunakan oleh push(E x) ketika array sudah hampir penuh
	 */
	private void doubleArray() {

	}
}
```

Implementasikan Stack menggunakan **LinkedList**:

```
import java.util.EmptyStackException;

public class MyLinkedListStack<E> implements Stack<E> {

	private ListNode<E> tos;		//asumsi: kelas ListNode sudah ada

	public MyLinkedListStack() {
		tos = null;
	}

	public boolean isEmpty() {

	}

	//dan seterusnya ...
	//implementasikan method-method yang sama yang ada pada interface Stack
	//tetapi, kali ini, kita menggunakan implementasi LinkedList

	//method doubleArray() tidak diperlukan ! mengapa ?




}
```

Diberikan interface **Queue** berikut:

```
public interface Queue<E> {

	/**
	 * kembalikan true jika Queue kosong, dan sebaliknya
	 */
	public boolean isEmpty();

	/**
	 * force Queue menjadi kosong
	 */
	public void makeEmpty();
```

```
	/**
	 * kembalikan nilai elemen yang berada di Front/Head
	 * tanpa mengeluarkannya/menghapusnya dari Queue
	 * perhatikan ketika Queue kosong
	 */
	public E getFront();

	/**
	 * proses dequeue() pada posisi front/head
	 * perhatikan ketika Queue kosong
	 */
	public void dequeue();

	/**
	 * proses gabungan getFront() dan dequeue()
	 */
	public E getFrontAndDequeue();

	/**
	 * proses enqueue()
	 * untuk implementasi Array: perhatikan ketika Queue
	 * hampir penuh !
	 */
	public void enqueue(E x);
}
```

Implementasikan Queue menggunakan **Array**:

*jika butuh melempar exception karena Queue kosong, silakan gunakan kelas **EmptyQueueException**. Kelas ini diasumsikan sudah ada (atau silakan Anda membuat kelas exception sendiri yang bernama **EmptyQueueException**).

```
import java.util.Arrays;

public class MyArrayQueue<E> implements Queue<E> {
	private E [] array;
	private int front;
	private int back;
	private int size;
	private static final int DEFAULT_CAPACITY = 5;

	public MyArrayQueue() {
		array = (E[]) new Object[DEFAULT_CAPACITY];
		size  = 0;
		front = back = 0;
	}

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

```
	//...dan seterusnya...



	// digunakan untuk arrayDoubling() saat enqueue() !
	private void arrayDoubling() {

		...
	}
}
```

Implementasikan Queue menggunakan **LinkedList**:

```
public class MyLinkedListQueue<E> implements Queue<E> {

	private ListNode<E> front;
	private ListNode<E> back;

	public MyLinkedListQueue() {
		front = back = null;
	}

	public boolean isEmpty() {

	}

	public void makeEmpty() {

	}



	//lanjutkan untuk enqueue(), dequeue(), getFront(), ...



}
```