

자료구조: 2022년 1학기 [강의]

Array Queue



© J.-H. Kang, CNU

강지훈

jhkang@cnu.ac.kr

충남대학교 컴퓨터융합학부

큐 (Queue)

큐 (Queue)

■ 특수한 성격의 리스트:

"서비스를 받기 위해 서는 줄"

- 줄에 먼저 서는 것이 먼저 서비스 받는다.
- 버스를 타기 위해서 서는 줄
- 식당에서 배식을 위해 서는 줄
- 컴퓨터 안에서 일(job)들이 처리 서비스를 받기 위해 서는 줄

■ 큐의 또 다른 용어:

- 선입선출 (先入先出) 리스트
- FIFO (First-In-First-Out) List

큐


■ 순서 리스트 (ordered list):

- 큐 안에 있는 원소의 순서가 삽입과 삭제의 위치를 결정


■ 즉, 삽입과 삭제는 큐의 양 끝에서:

- 새로운 원소의 **삽입**은 항상 큐의 **뒤쪽**에서
- 서비스 받기 위한 원소의 **삭제**는 큐의 **앞쪽**에서

$$Q = (a_0, \dots, a_{n-1})$$



front 원소



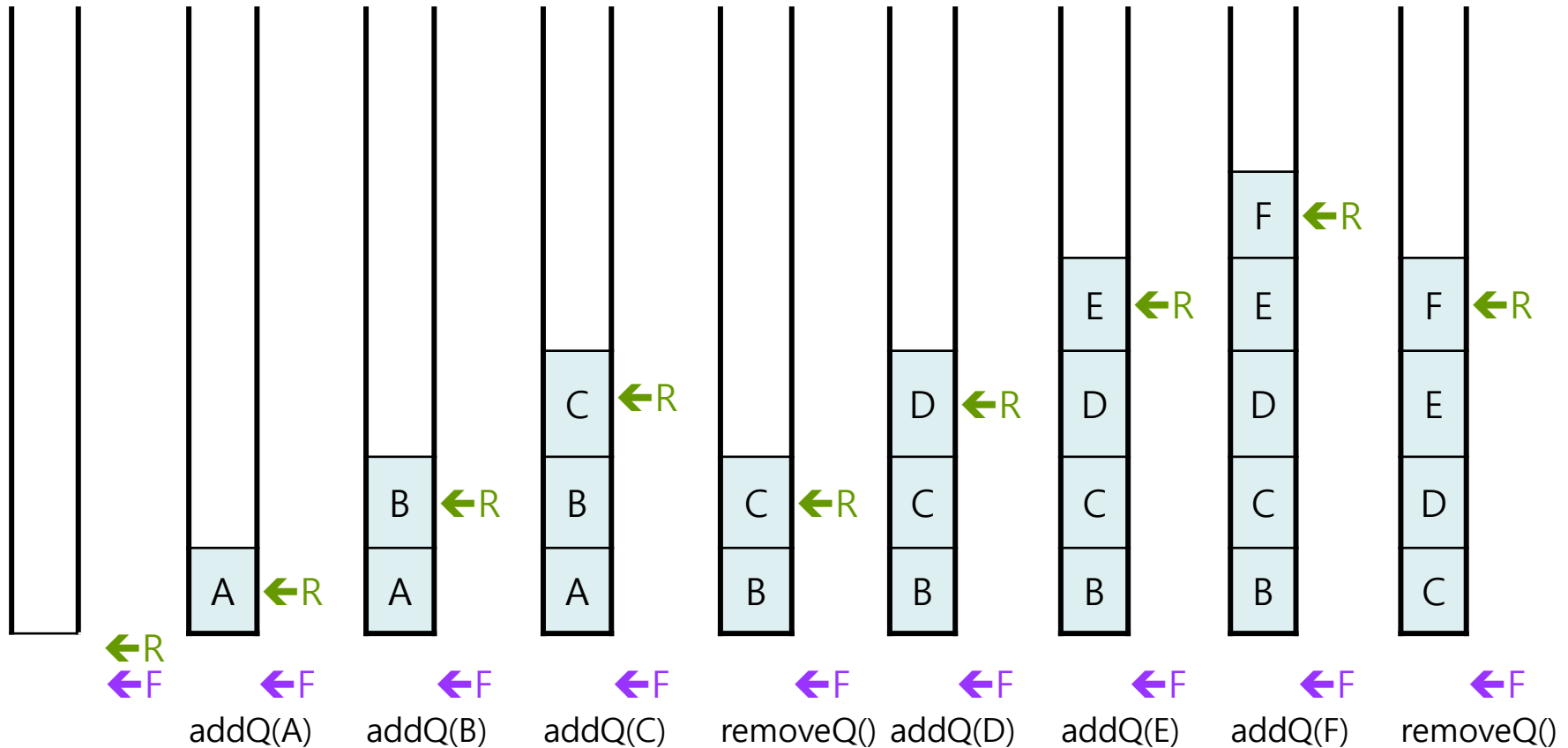
rear 원소

a_{i+1} 은 a_i 의 **뒤**에 있다 ($0 \leq i \leq n-2$)

□ 예: 큐의 작동 원리

"←R" 큐에서 **rear** 원소를 가리킨다

"←F" 큐에서 **front** 원소를 가리킨다



Class "Queue<T>"



□ Queue<T>: 공개함수

■ Queue<T> 객체 사용법

- public Queue () ;
- public boolean isEmpty () ;
- public boolean isFull () ;
- public int size () ;
- public T front () ;
- public T rear () ;
- public boolean enqueue (T anElement) ;
- public T dequeue () ;
- public void clear () ;



Class "ArrayQueue<T>"



□ **ArrayQueue<T>: 공개함수**

■ **ArrayQueue<T> 객체 사용법**

- **public** `ArrayQueue () ;`
- **public** `ArrayQueue (int givenCapacity) ;`

- **public boolean** `isEmpty () ;`
- **public boolean** `isFull () ;`
- **public int** `size () ;`

- **public T** `front () ;`
- **public T** `rear () ;`

- **public boolean** `enqueue (T anElement) ;`
- **public T** `dequeue () ;`
- **public void** `clear () ;`



Class "ArrayQueue" 의 구현



□ Class “ArrayQueue<T>”의 초기 형태는 이렇게!

```
public class ArrayQueue<T>
{
    public ArrayQueue ( )
    {
        // 수정해야 함
    }
    public ArrayQueue (int givenCapacity)
    {
        // 수정해야 함
    }

    public boolean isEmpty () {
        return true ; // 수정해야 함
    }
    public boolean isFull () {
        return true ; // 수정해야 함
    }
    public int size () {
        return 0 ; // 수정해야 함
    }
    public T front () {
        return null ; // 수정해야 함
    }
    public T rear () {
        return null ; // 수정해야 함
    }
    public boolean enqueue (E anElement) {
        return true ; // 수정해야 함
    }
    public T dequeue () {
        return null ; // 수정해야 함
    }
    public void clear () {
        return ; // 수정해야 함
    }
}
```

} // End of Class “ArrayQueue<T>”



□ ArrayQueue: 기본 작동 방법

front	rear	Q[0]	Q[1]	Q[2]	Q[3]	설명
-1	-1					Empty queue
-1	0	J ₁				enQueue (J ₁)
-1	1	J ₁	J ₂			enQueue (J ₂)
-1	2	J ₁	J ₂	J ₃		enQueue (J ₃)
0	2		J ₂	J ₃		deQueue -> J ₁
1	2			J ₃		deQueue -> J ₂
1	3			J ₃	J ₄	enQueue (J ₄)
2	3				J ₄	deQueue -> J ₃



□ ArrayQueue: 비공개 인스턴스 변수

```
public class ArrayQueue<T>
{
    // 비공개 상수
    private static final int DEFAULT_CAPACITY = 50 ;

    // 비공개 인스턴스 변수
    private int          _capacity ;
    private int          _frontPosition ;
    private int          _rearPosition ;
    private T[]          _elements ;
}
```

ArrayQueue: Getter / Setter

```

public class ArrayQueue<T>
{
    // 비공개 상수
    private static final int DEFAULT_CAPACITY = 50 ;

    // 비공개 인스턴스 변수
    private int        _capacity ;
    private int        _frontPosition ;
    private int        _rearPosition ;
    private T[]        _elements ;

    // Getter/Setter
    private int capacity() {
        return this._capacity ;
    }
    private void setCapacity (int newCapacity) {
        this._capacity = newCapacity ;
    }
    private int frontPosition() {
        return this._frontPosition ;
    }
    private void setFrontPosition (int newFrontPosition) {
        this._frontPosition = newFrontPosition ;
    }
    private int rearPosition() {
        return this._rearPosition ;
    }
    private void setRearPosition (int newRearPosition) {
        this._rearPosition = newRearPosition ;
    }
    private T[] elements() {
        return this._elements ;
    }
    private void setElements (T[] newElements) {
        this._elements = newElements ;
    }
}

```



□ ArrayQueue: 생성자

```
public class ArrayQueue<T>
{
```

```
// 생성자
```

```
public ArrayQueue ()
```

```
{
```

```
    this (ArrayQueue.DEFAULT_CAPACITY);
```

```
}
```

```
public ArrayQueue (int givenCapacity)
```

```
{
```

```
    this.setCapacity (givenCapacity);
```

```
    @SuppressWarnings("Unchecked");
```

```
    this.setElements ( (T[ ]) new Object[this.capacity()] );
```

```
    this.setFrontPosition (-1);
```

```
    this.setRearPosition (-1);
```

```
}
```

- static 으로 선언된 상수/변수/함수를 언급할 때에는 class의 이름으로.
- (static 으로 선언되지 않은 것을 언급할 때에는 this)



□ ArrayQueue: 상태 알아보기

```
public class ArrayQueue<T>  
{
```

```
.....
```

```
// 상태 알아보기
```

```
public boolean isEmpty ()
```

```
{
```

```
    return (this.frontPosition() == this.rearPosition());
```

```
}
```

```
public boolean isFull ()
```

```
{
```

```
    return (this.rearPosition() == (this.capacity()-1) );
```

```
}
```

```
public int size()
```

```
{
```

```
    return (this.rearPosition() - this.frontPosition());
```

```
}
```

- 큐의 크기 (큐에 들어있는 원소의 개수)를 얻는 getter
- Setter는 존재하지 않는다. 이유는?

❑ ArrayQueue: front(), rear()

```
public T front()
{
    T frontElement = null ;
    if ( ! this.isEmpty() ) {
        frontElement = this.elements()[this.frontPosition()+1] ;
    }
    return frontElement ;
}
```

```
public T rear()
{
    T rearElement = null ;
    if ( ! this.isEmpty() ) {
        rearElement = this.elements()[this.rearPosition()] ;
    }
    return rearElement ;
}
```

❑ ArrayQueue : enqueue()

// 원소 추가 함수

```
public boolean enqueue (T anElement)
{
    if ( this.isFull() ) {
        return false ;
    }
    else {
        this.setRearPosition (this.rearPosition()+1) ;
        this.elements()[this.rearPosition()] = anElement ;
        return true ;
    }
}
```

❑ **ArrayQueue : dequeue()**

```
public T dequeue()
{
    T frontElement = null ;
    if ( ! this.isEmpty() ) {
        this.setFrontPosition (this.frontPosition()+1) ;
        frontElement = this.elements()[this.frontPosition()] ;
        this.elements()[this.frontPosition()] = null ;
    }
    return frontElement ;
}
```

❑ ArrayQueue : clear()

```
public void clear ()  
{  
    for ( i = this.frontPosition()+1 ; i <= this.rearPosition() ; i++ ) {  
        this.elements()[i] = null ;  
    }  
    this.setFrontPosition (-1) ;  
    this.setRearPosition  (-1) ;  
}
```



CircularArrayQueue 로의 구현



□ ArrayQueue 구현의 문제점

- 원소의 위치가 점진적으로 왼쪽에서 오른쪽으로 이동!!

front	rear	Q[0]	Q[1]	Q[2]	Q[3]	설명
-1	-1					Empty queue
-1	0	J ₁				enqueue (J ₁)
-1	1	J ₁	J ₂			enqueue (J ₂)
-1	2	J ₁	J ₂	J ₃		enqueue (J ₃)
0	2		J ₂	J ₃		dequeue → J ₁
1	2			J ₃		dequeue → J ₂
1	3			J ₃	J ₄	enqueue (J ₄)
2	3				J ₄	dequeue → J ₃



□ Circular Array Queue로 구현

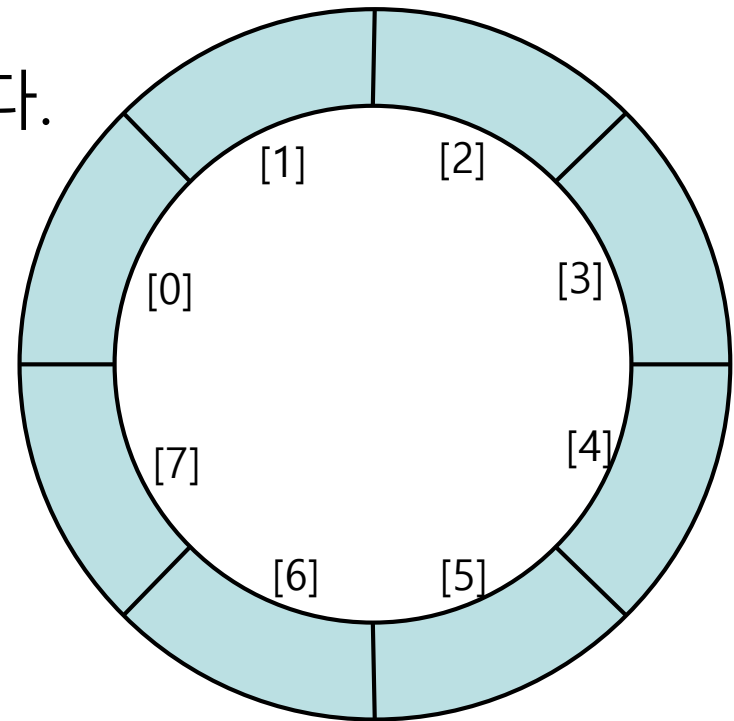
■ 반지 형태의 환형 배열:

- 배열의 끝이 배열의 맨 처음과 붙어 있는 것으로 본다.

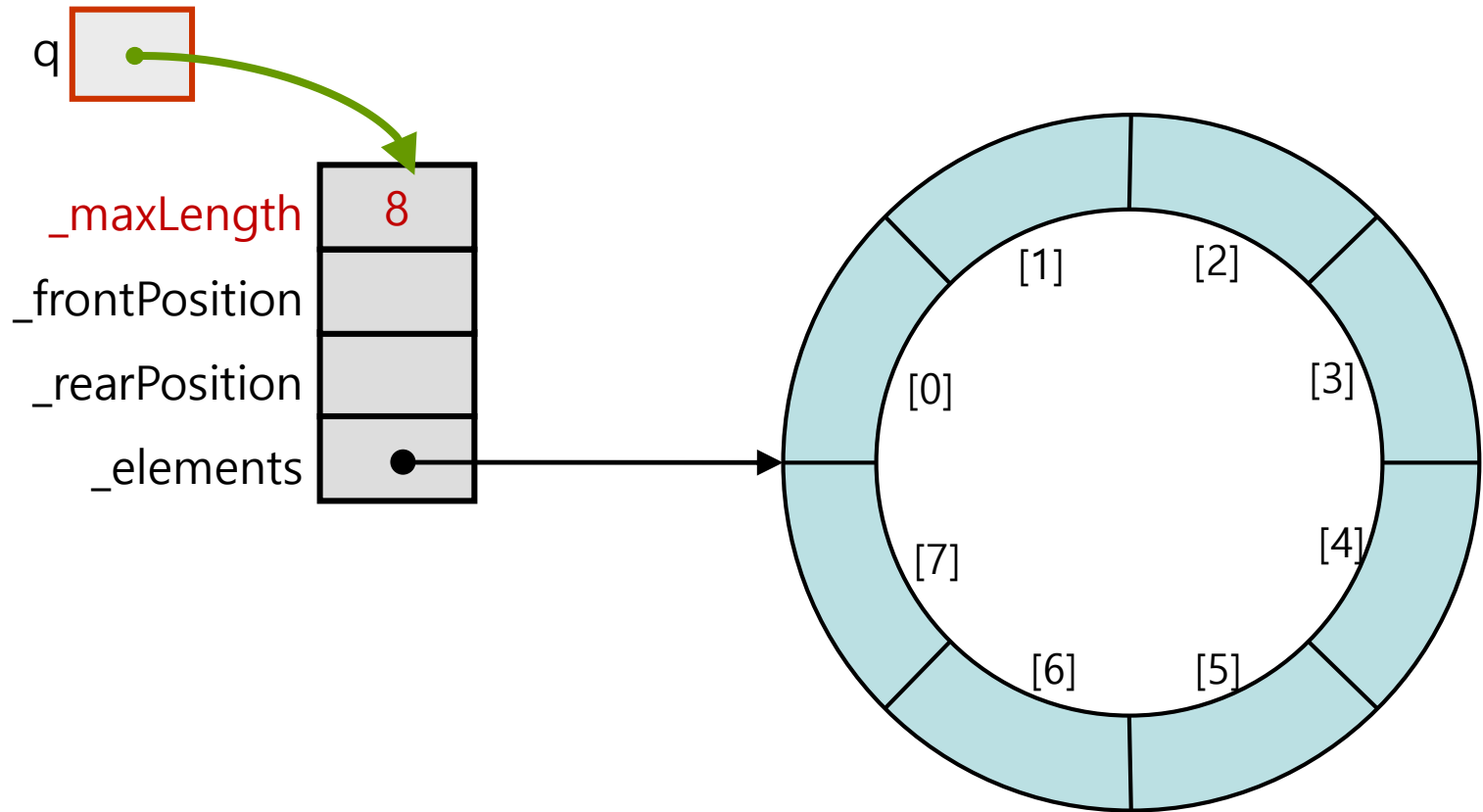
■ Public 함수의 사용법은 바뀌지 않는다!!!

■ 함수의 구현만 약간 변경될 뿐이다.

- 함수 코드를 일부 수정



□ 배열을 반지 형태로 생각하자



□ 다음 위치 계산은?

- 크기가 8인 Circular Queue 에서,
 - [7]번째 다음은 [0]번째

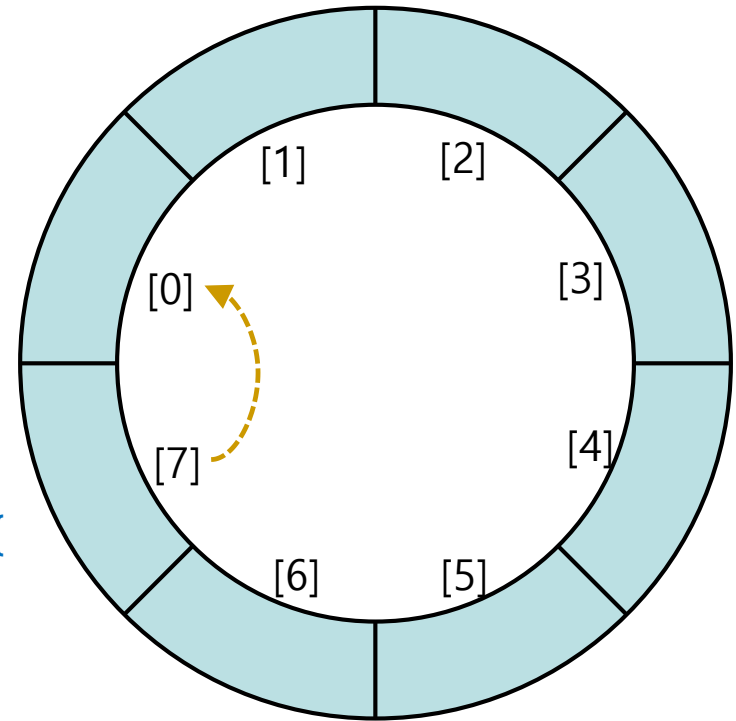
- 일반적으로 다음 위치 계산은 어떻게?

```
this._rear++;
```

```
if (this._rearPosition == this._maxLength) {  
    this._rearPosition = 0 ;  
}
```

- 이렇게도...

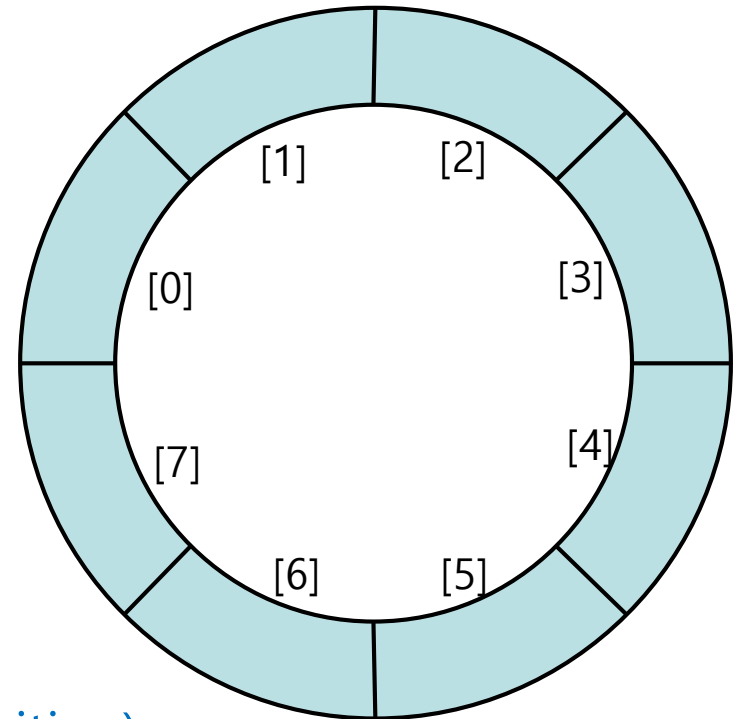
```
this._rearPosition = (this._rearPosition+1) % this._maxLength ;
```



□ 초기화 / Empty 조건

■ 초기화는?

- `this._frontPosition = 0 ;`
- `this._rearPosition = 0 ;`



■ 큐 Empty 조건은?

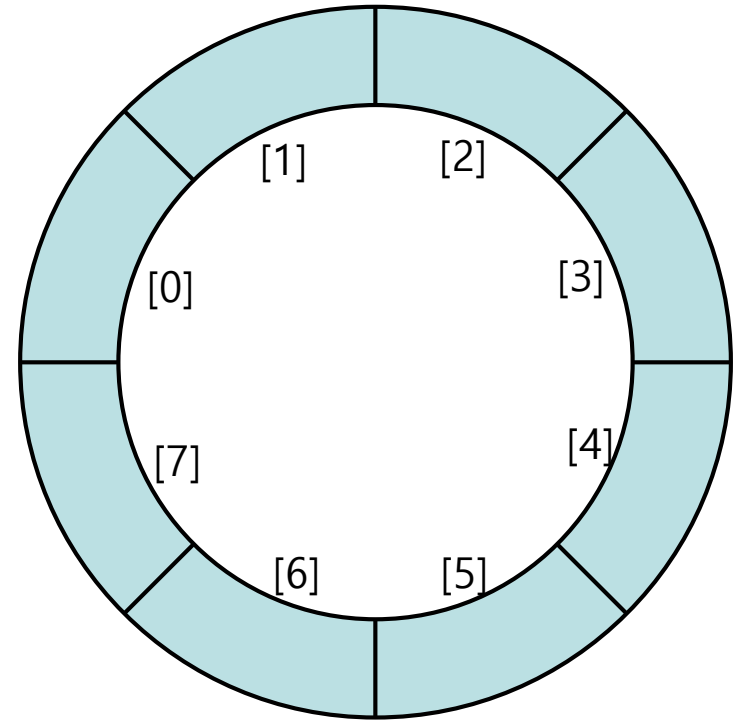
- `(this._frontPosition == this._rearPosition)`
- 초기 상태도 empty임

■ 큐 Full 조건은?

□ Full 조건 [1]

■ 큐 Full 조건은?

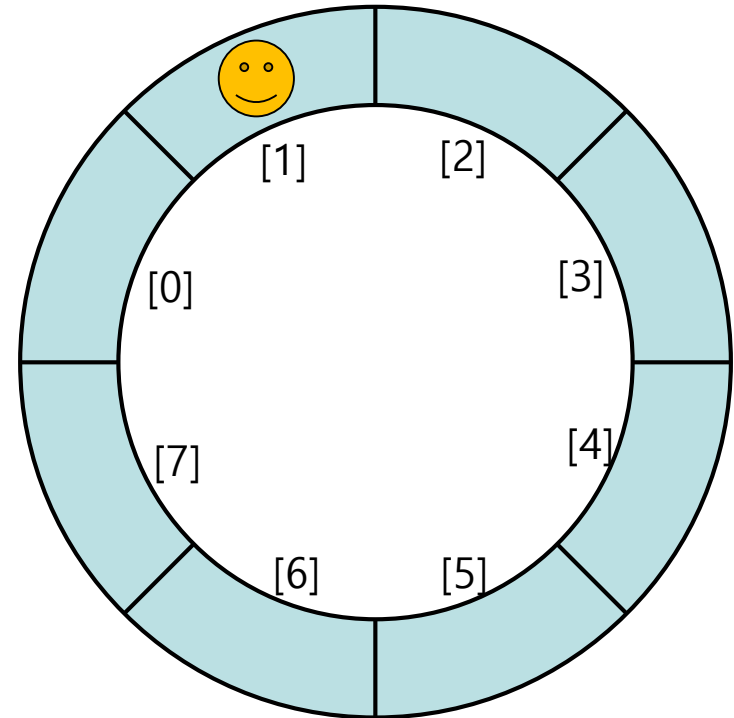
- 초기 시작은 (큐 empty 상태),
 - ◆ `this._frontPosition == 0`
 - ◆ `this._rearPosition == 0`



□ Full 조건 [2]

■ 큐 Full 조건은?

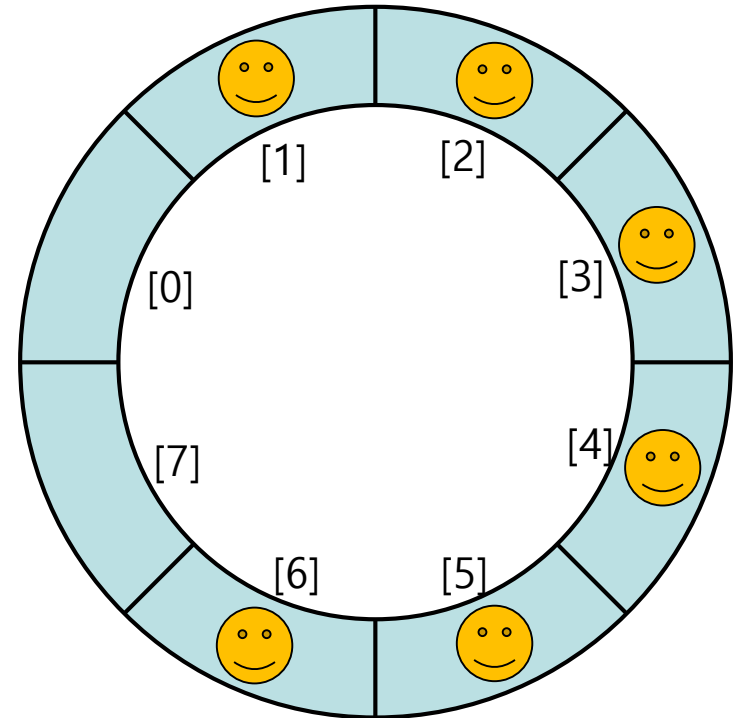
- 초기 시작은,
 - ◆ `this._frontPosition == 0`
 - ◆ `this._rearPosition == 0`
- 1개가 차면,
 - ◆ `this._frontPosition == 0`
 - ◆ `this._rearPosition == 1`



□ Full 조건 [2]

■ 큐 Full 조건은?

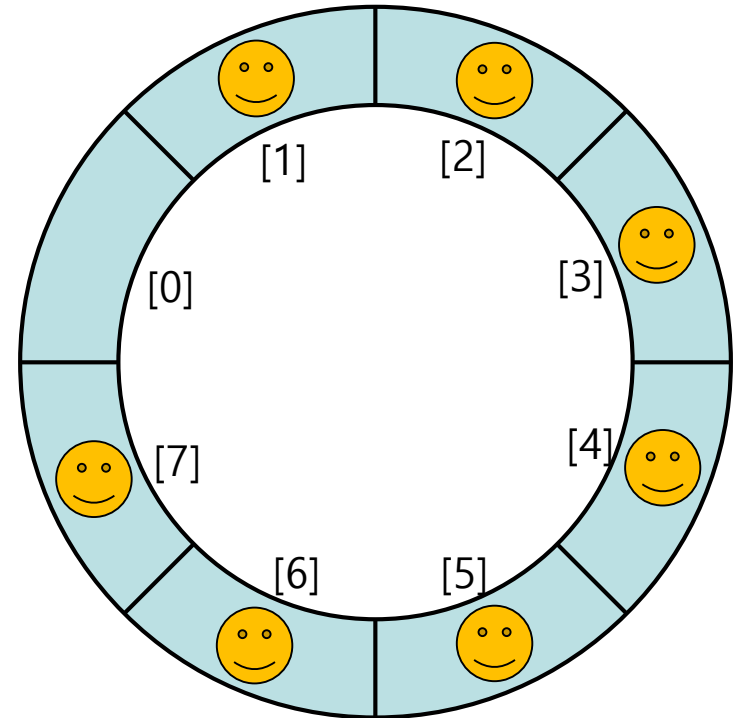
- 초기 시작은,
 - ◆ `this._frontPosition == 0`
 - ◆ `this._rearPosition == 0`
- 6개가 차면,
 - ◆ `this._frontPosition == 0`
 - ◆ `this._rearPosition == 6`



□ Full 조건 [2]

■ 큐 Full 조건은?

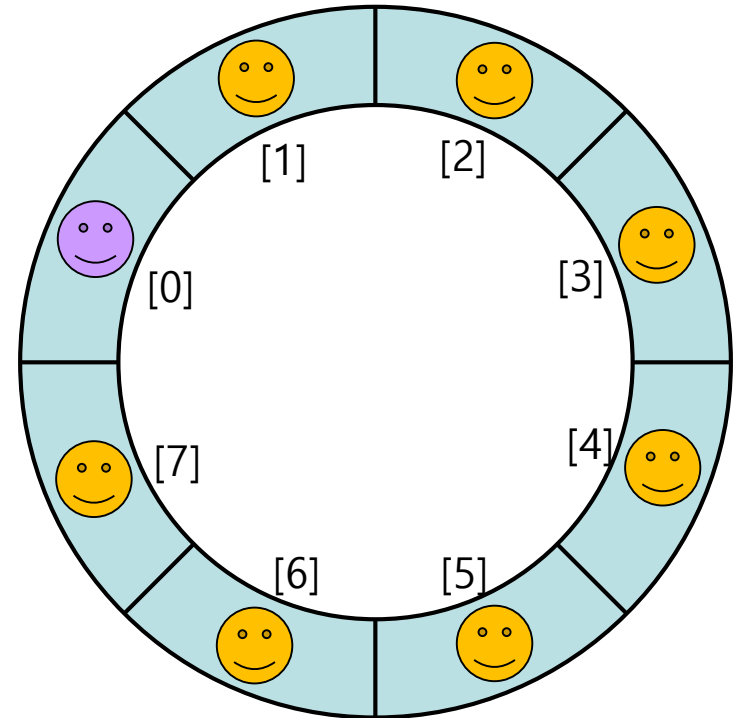
- 초기 시작은,
 - ◆ `this._frontPosition == 0`
 - ◆ `this._rearPosition == 0`
- 6개가 차면,
 - ◆ `this._frontPosition == 0`
 - ◆ `this._rearPosition == 6`
- 7개가 차면,
 - ◆ `this._frontPosition == 0`
 - ◆ `this._rearPosition == 7`



□ Full 조건 [3]

■ 큐 Full 조건은?

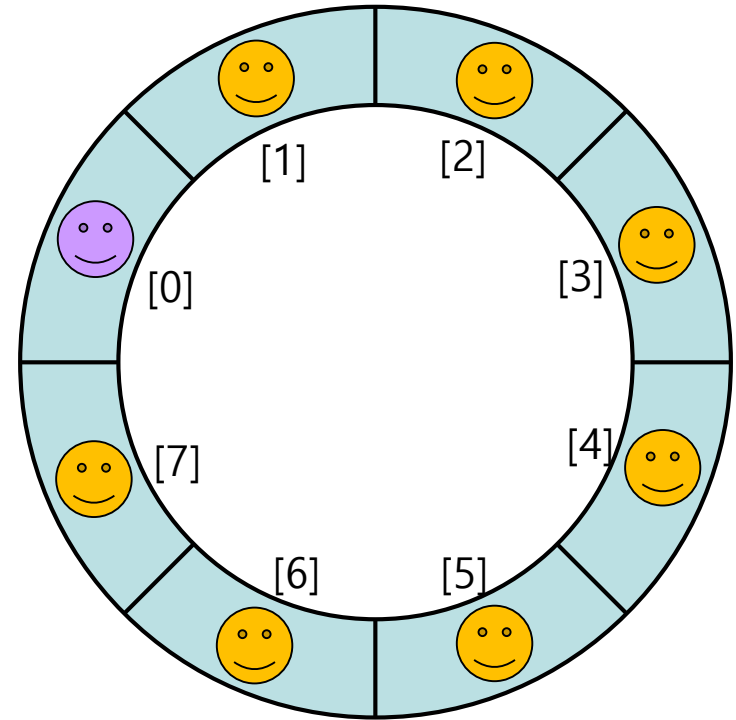
- 초기 시작은,
 - ◆ `this._frontPosition == 0`
 - ◆ `this._rearPosition == 0`
- 6개가 차면,
 - ◆ `this._frontPosition == 0`
 - ◆ `this._rearPosition == 6`
- 7개가 차면,
 - ◆ `this._frontPosition == 0`
 - ◆ `this._rearPosition == 7`
- 8개가 차면,
 - ◆ `this._frontPosition == 0`
 - ◆ `this._rearPosition == 0`



□ Full 조건 [4]

■ 큐 Full 조건은?

- 초기 시작은,
 - ◆ `this._frontPosition == 0`
 - ◆ `this._rearPosition == 0`
- 6개가 차면,
 - ◆ `this._frontPosition == 0`
 - ◆ `this._rearPosition == 6`
- 7개가 차면,
 - ◆ `this._frontPosition == 0`
 - ◆ `this._rearPosition == 7`
- 8개가 차면,
 - ◆ `this._frontPosition == 0`
 - ◆ `this._rearPosition == 0`



■ 모두 채우면 큐가 empty 인지 full 인지 구분 불가!

□ CircularArrayQueue<T>의 공개함수

■ CircularArrayQueue<T> 객체 사용법

- public CircularArrayQueue () ;
- public CircularArrayQueue (int givenCapacity) ;

- public int capacity() ; // 큐에 넣을 수 있는 최대 개수
- public boolean isEmpty () ;
- public boolean isFull () ;
- public int size () ;

- public T front () ;
- public T rear () ;

- public boolean enqueue (T anElement) ;
- public T dequeue () ;
- public void clear () ;

Class "CircularArrayQueue" 의 구현



❑ CircularArrayQueue: 비공개 인스턴스 변수

```
public class CircularArrayQueue<T>
{
    // 비공개 상수
    private static final int DEFAULT_CAPACITY = 50 ;

    // 비공개 인스턴스 변수
    private int      _maxLength ; // capacity + 1
    private int      _frontPosition ;
    private int      _rearPosition ;
    private T[]      _elements ;
}
```

□ CircularArrayQueue: Getter/Setter

```
public class CircularArrayQueue<T>
{
    // 비공개 상수
    private static final int DEFAULT_CAPACITY = 50 ;

    // 비공개 인스턴스 변수
    private int _maxLength ; // capacity + 1
    private int _frontPosition ;
    private int _rearPosition ;
    private T[] _elements ;

    // Getter/Setter
    private int maxLength() {
        return this._maxLength ;
    }
    private void setMaxLength (int newMaxLength) {
        this._maxLength = newMaxLength ;
    }
    private int frontPosition() {
        return this._frontPosition ;
    }
    private void setFrontPosition (int newFrontPosition) {
        this._frontPosition = newFrontPosition ;
    }
    private int rearPosition() {
        return this._rearPosition ;
    }
    private void setRearPosition (int newRearPosition) {
        this._rearPosition = newRearPosition ;
    }
    private T[] elements() {
        return this._elements ;
    }
    private void setElements (T[] newElements) {
        this._elements = newElements ;
    }
}
```



□ CircularArrayQueue 의 생성자

```
public class CircularArrayQueue<T>
{
    // 생성자
    public CircularArrayQueue ()
    {
        this (CircularArrayQueue.DEFAULT_CAPACITY) ;
    }

    public CircularArrayQueue (int givenCapacity)
    {
        this.setMaxLength (givenCapacity + 1) ;
        @SuppressWarnings("Unchecked") ;
        this.setElements ( (T[ ]) new Object[this.maxLength()] ) ;
        this.setFrontPosition (0) ;
        this.setRearPosition (0) ;
    }
}
```

□ CircularArrayQueue: 상태 알아보기

```
public class CircularArrayQueue<T>
{
    .....

    // 상태 알아보기
    public boolean isEmpty () {
        return (this.frontPosition() == this.rearPosition()) ;
    }

    public boolean isFull () {
        int nextRearPosition = (this.rearPosition() + 1) % this.maxLength() ;
        return (nextRearPosition == this.frontPosition()) ;
    }

    public int size() {
        if ( this.frontPosition() <= this.rearPosition() ) {
            return (this.rearPosition() - this.frontPosition())
        }
        else {
            return ( (this.rearPosition() + this.maxLength()) - this.frontPosition() ) ;
        }
    }
}
```

❑ CircularArrayQueue: frontElement()

```
public T front()
{
    T frontElement = null ;
    if ( ! this.isEmpty() ) {
        frontElement = this.elements()[this.frontPosition()+1] ;
    }
    return frontElement ;
}
```

```
public T rear()
{
    T rearElement = null ;
    if ( ! this.isEmpty() ) {
        rearElement = this.elements()[this.rearPosition()] ;
    }
    return rearElement ;
}
```

❑ CircularArrayQueue: enqueue()

// 원소 추가 함수

```
public boolean enqueue (T anElement)
{
    if ( this.isFull() ) {
        return false ;
    }
    else {
        this.setRearPosition (
            (this.rearPosition()+1) % this.maxLength() ) ;
        this.elements[this.rearPosition()] = anElement ;
        return true ;
    }
}
```


□ CircularArrayQueue: dequeue()

```
public T dequeue()  
{  
    T frontElement = null ;  
    if ( ! this.isEmpty() ) {  
        this.setFrontPosition (  
            (this.frontPosition()+1) % this.maxLength() ) ;  
        frontElement = this.elements()[this.frontPosition()] ;  
        this.elements()[this.frontPosition()] = null ;  
    }  
    return frontElement ;  
}
```

❑ CircularArrayQueue: clear()

```
public void clear ()
{
    int clearSize = this.size() ;
    int clearPosition = this.frontPosition() ;
    for ( int i = 0 ; i < clearSize ; i++ ) {
        clearPosition = (clearPosition+1) % this.maxLength() ;
        this.elements()[clearPositon] = null ;
    }
    this.setFrontPosition (0) ;
    this.setRearPosition (0) ;
}
```

End of "Array Queue"



