큐(Queue)

1. ADT Queue
2. Array Queue (Queue on Array)
3. Linked Queue (Queue by Link)
4. Reusing List
5. 큐 사용 프로그램 예

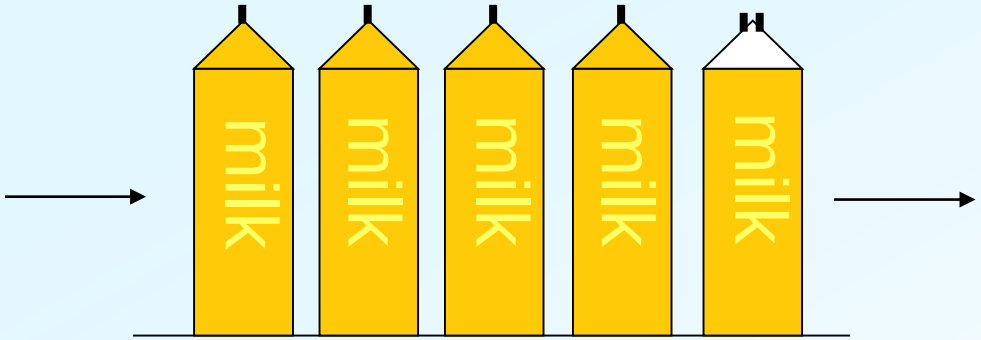
1. ADT Queue

Stack vs. Queue

Stack: 최근에 삽입된 원소를 제거한다
Queue: 가장 먼저 삽입된 원소를 제거한다

Stack과 Queue는
쌍으로 생각하면서 공부할 것

stack



queue

도입을 위한 예

- 오래된 우유부터 먹기
- 식당에서 기다리는 사람들의 열
- Lotto 복권을 사려는 사람들의 열

✓ Stack: LIFO (Last-In-First-Out)

✓ Queue: FIFO (First-In-First-Out)

ADT Queue

맨 뒤에 원소 x를 추가한다

맨 앞의 원소를 알려주면서 삭제한다

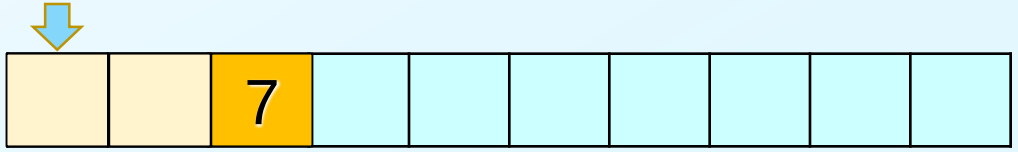
맨 앞의 원소를 알려준다

유일하게 접근 가능한 원소는
가장 먼저 삽입한 원소

FIFO(First-In-First-Out)란 별칭을 갖고 있다

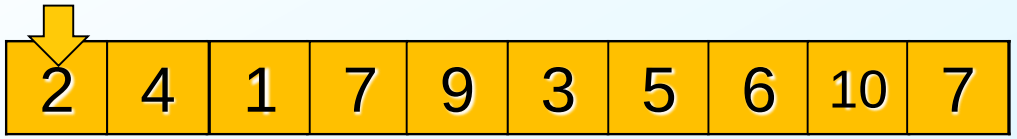
2. Array Queue

A Naïve Version

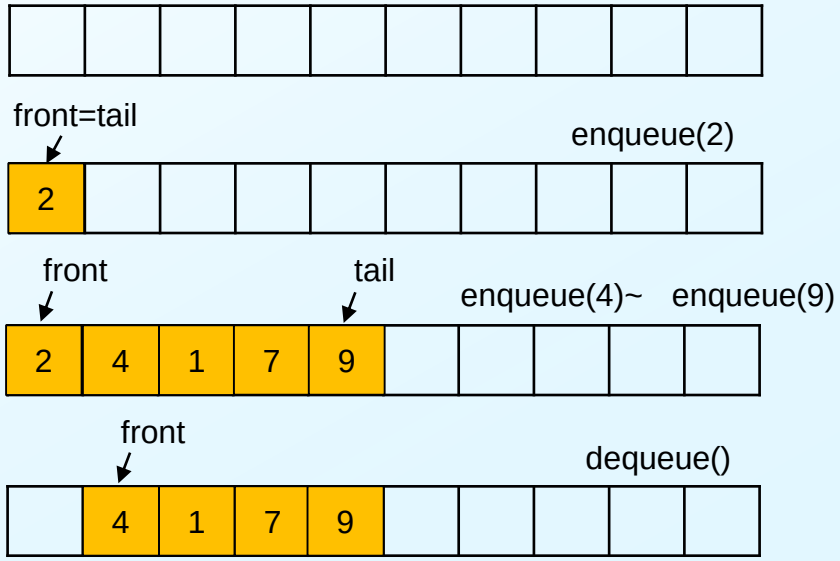


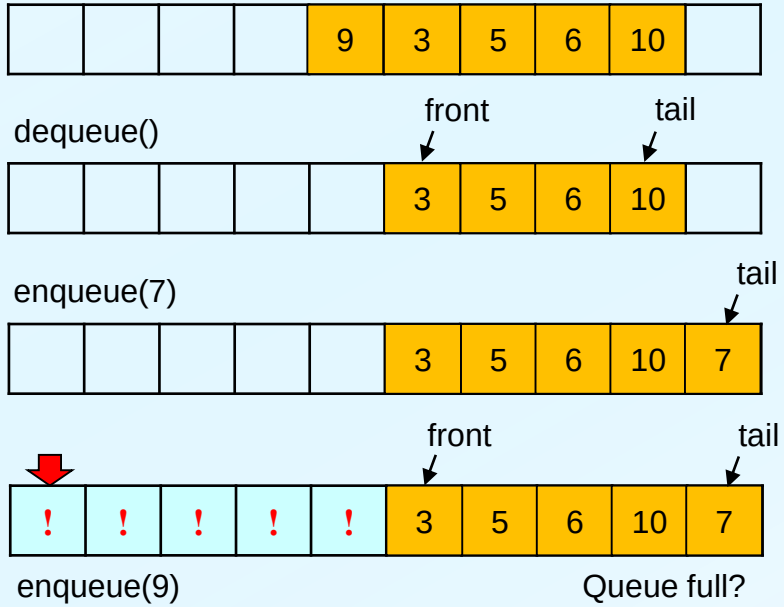
← Animation

queue.enqueue(5)
queue.enqueue(2)
queue.enqueue(7)
queueFront ← queue.front()
queueFront ← queue.dequeue()
queueFront ← queue.dequeue()



Animation

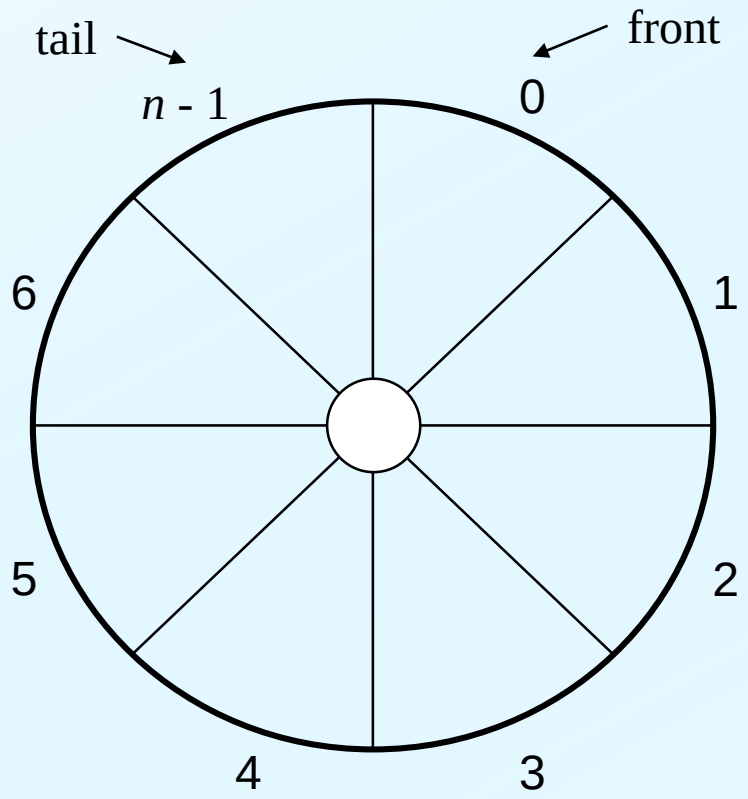




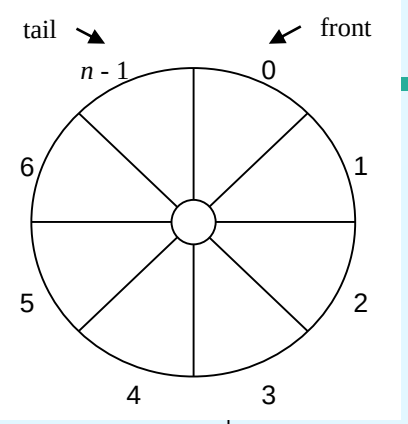
단순히 구현하면 공간이 있는데도 큐가 꽉 찬 것처럼 보인다

원형 큐circular queue로 구현하면 이 문제는 해결된다

원형 큐 Circular Queue



Array Queue 객체 구조

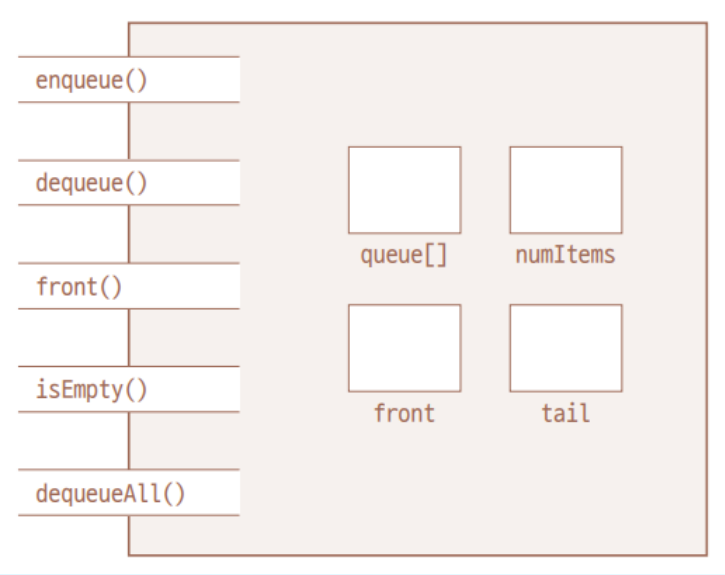


필드:

- queue[] ◀ 큐 원소들이 저장되는 배열
- numItems ◀ 큐에 있는 원소의 총 수
- front ◀ 맨 앞 원소의 인덱스
- tail ◀ 맨 뒤 원소의 인덱스

작업:

- enqueue() ◀ 큐의 맨 뒤에 원소를 삽입한다
- dequeue() ◀ 큐의 맨 앞에 있는 원소를 알려주고 삭제한다
- front() ◀ 큐의 맨 앞에 있는 원소를 알려준다
- isEmpty() ◀ 큐가 빈 큐인지를 알려준다
- dequeueAll() ◀ 큐를 깨끗이 청소한다

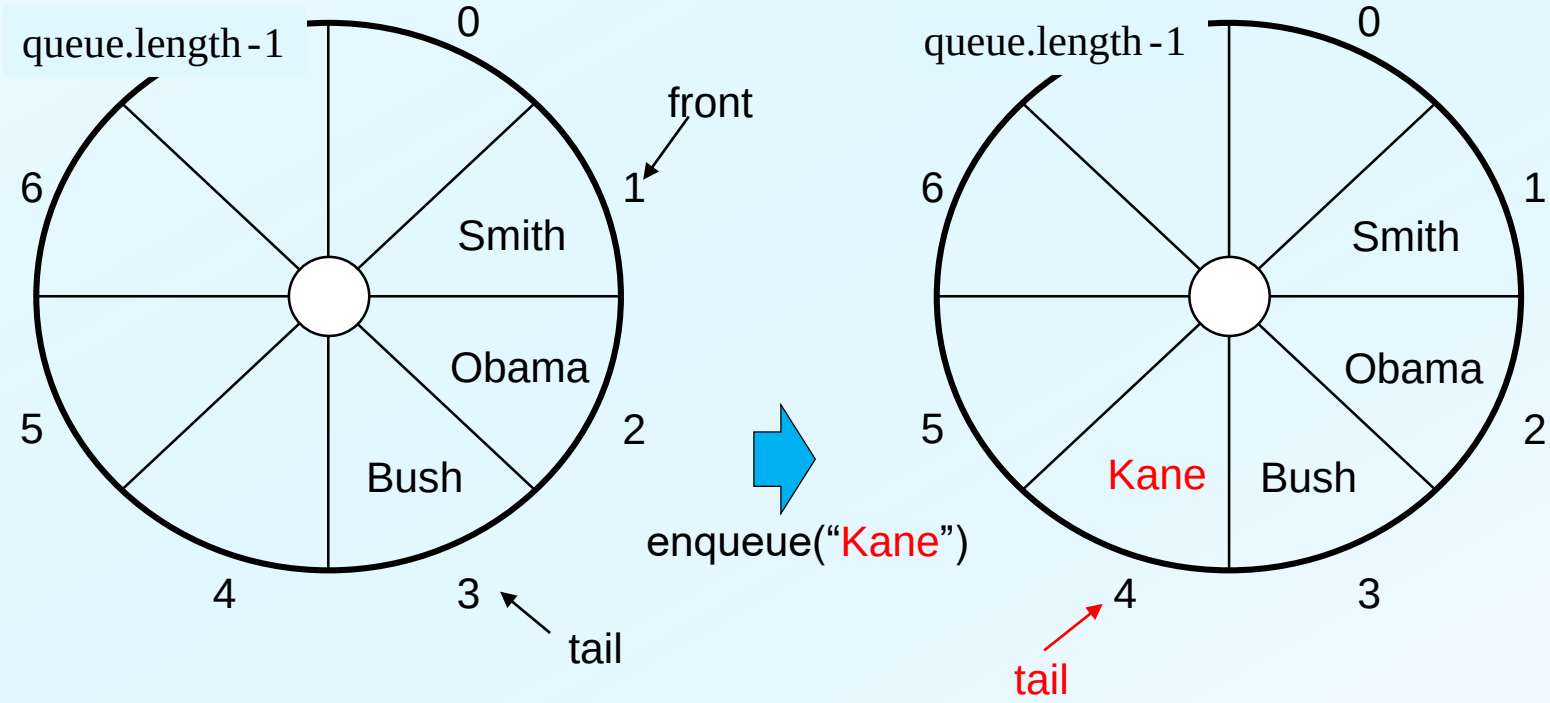


삽입 Insertion

핵심 작업

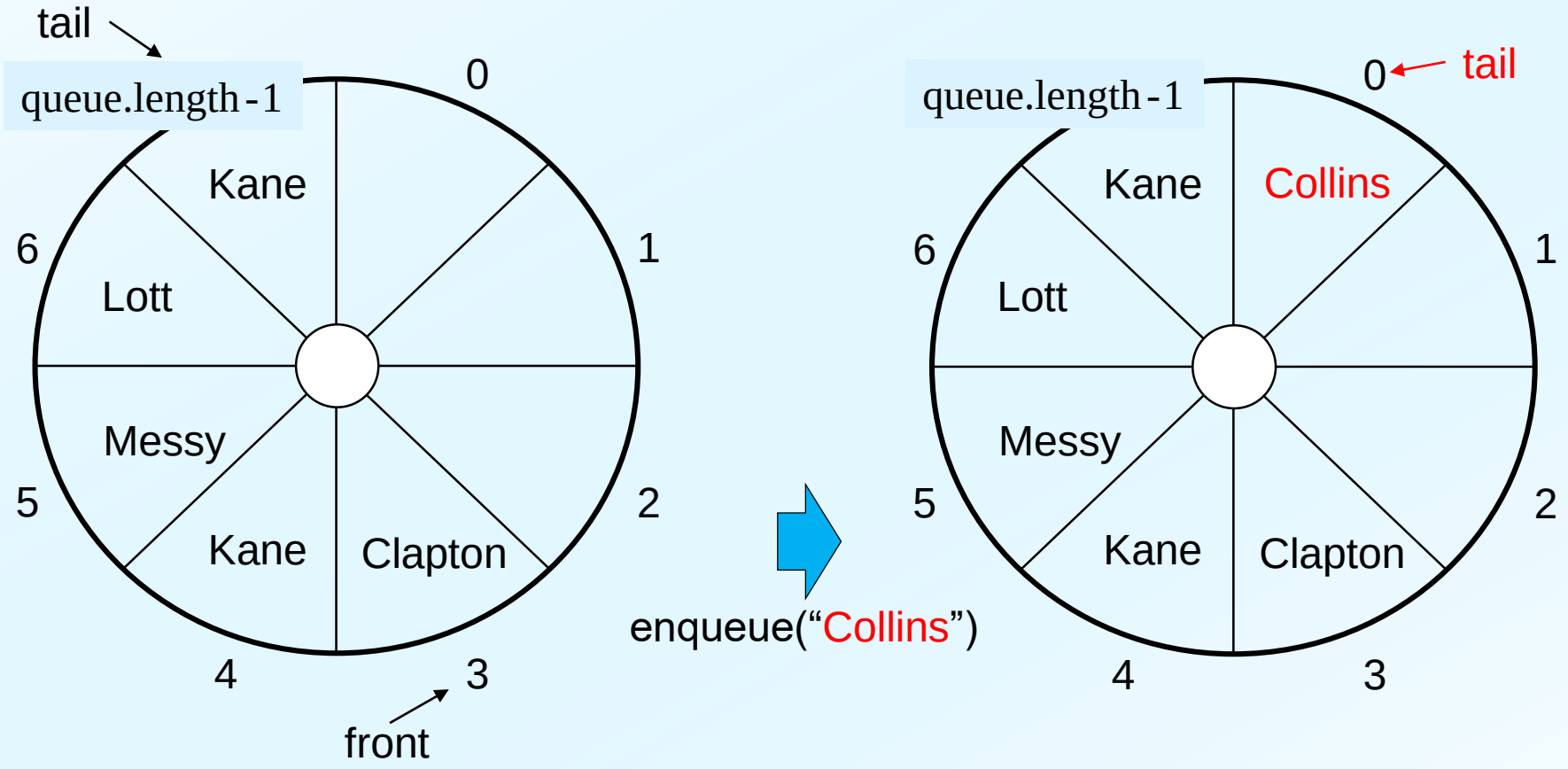
```
tail ← (tail+1) % queue.length
queue[tail] ← x
numItems++
```

```
enqueue(x):  ◀ Insert x
    if (numItems >= queue.length)
        /*에러 처리*/  ◀ 배열 용량 초과
    else
        tail ← (tail+1) % queue.length
        queue[tail] ← x
        numItems++
```



경계를 넘는 예

tail ← (tail+1) % queue.length

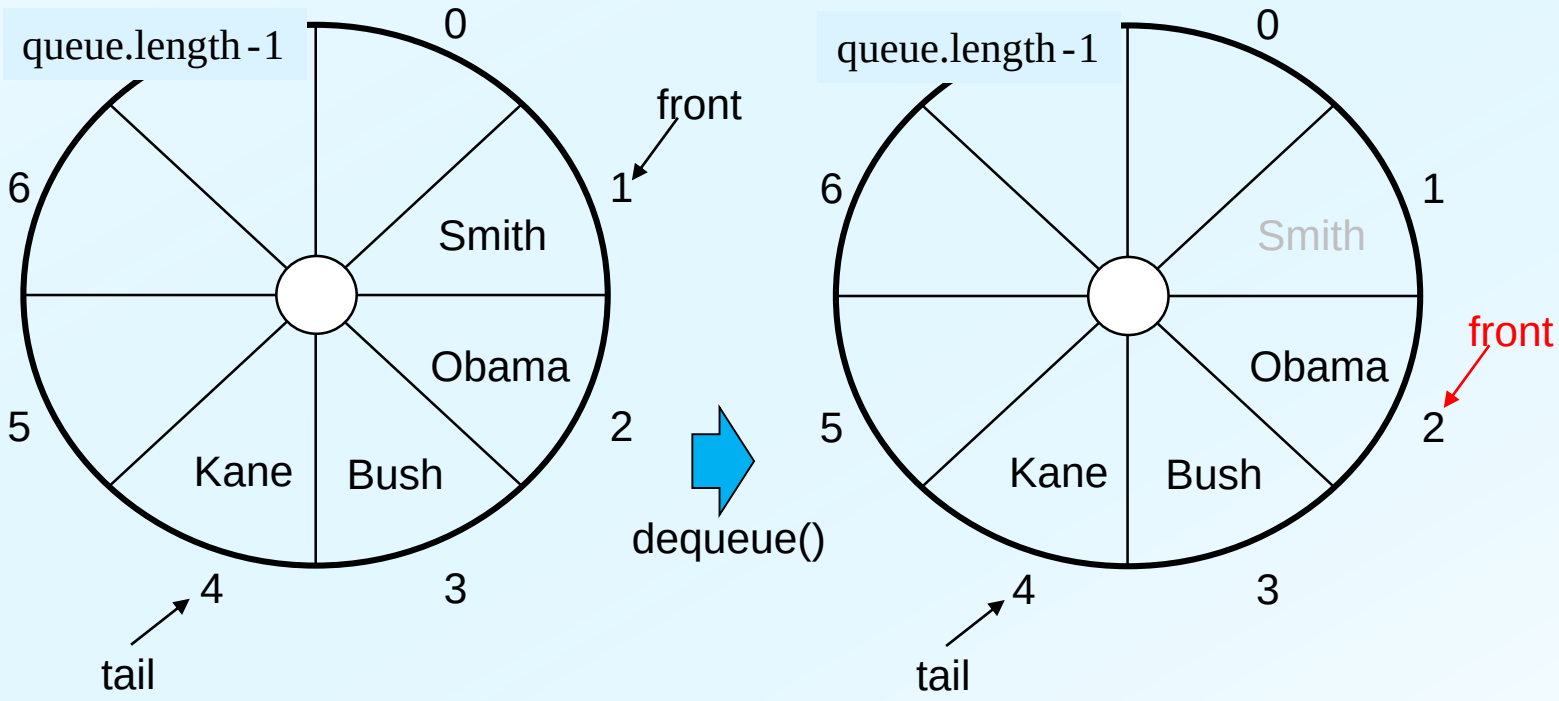


삭제 Deletion

핵심 작업

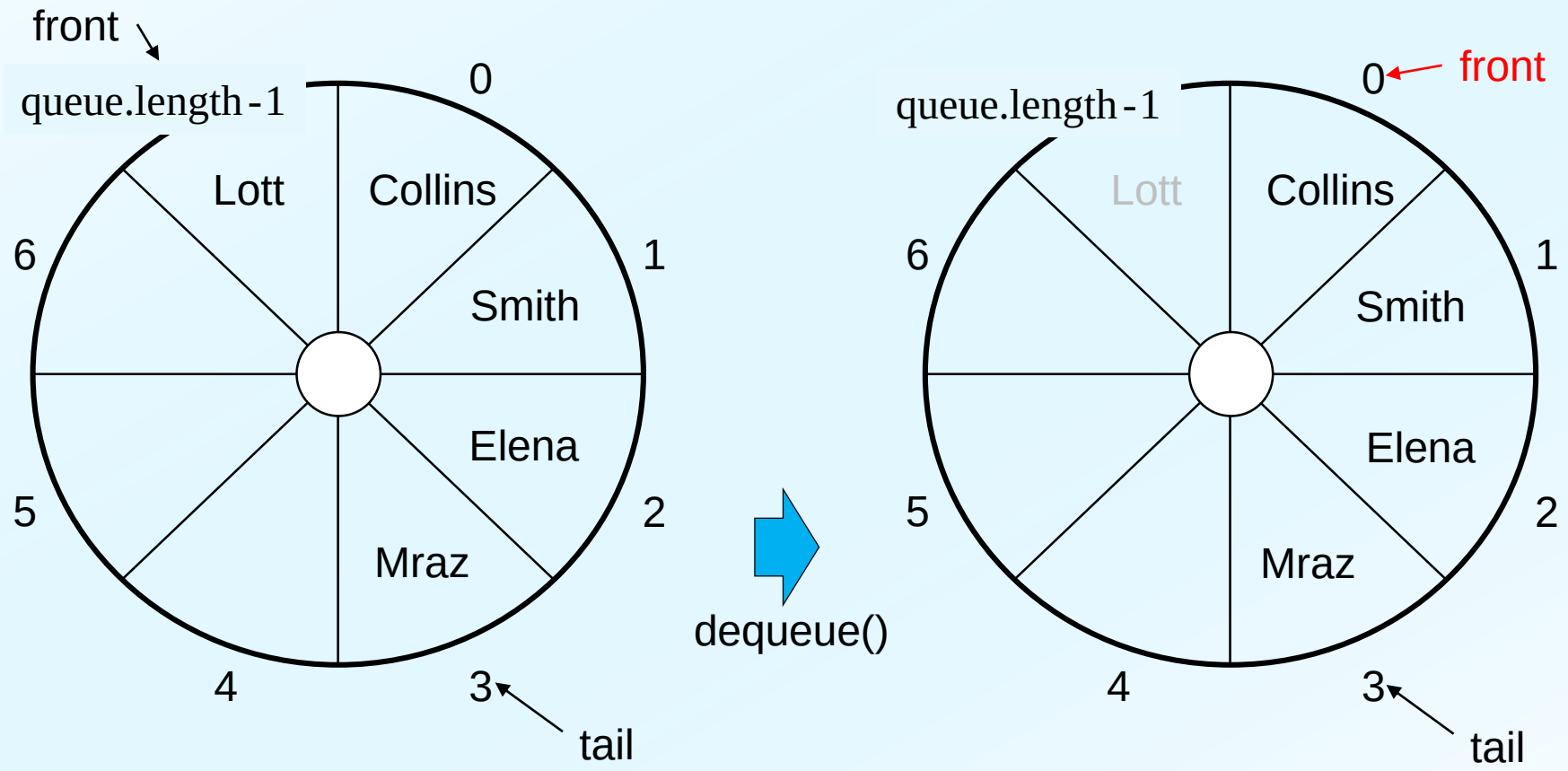
```
queueFront ← queue[front]
front ← (front+1) % queue.length
numItems--
return queueFront
```

```
dequeue(): ◀ Remove the front item
        if (isEmpty())
                /*에러 처리*/
        else
                queueFront ← queue[front]
                front ← (front+1) % queue.length
                numItems--
                return queueFront
```

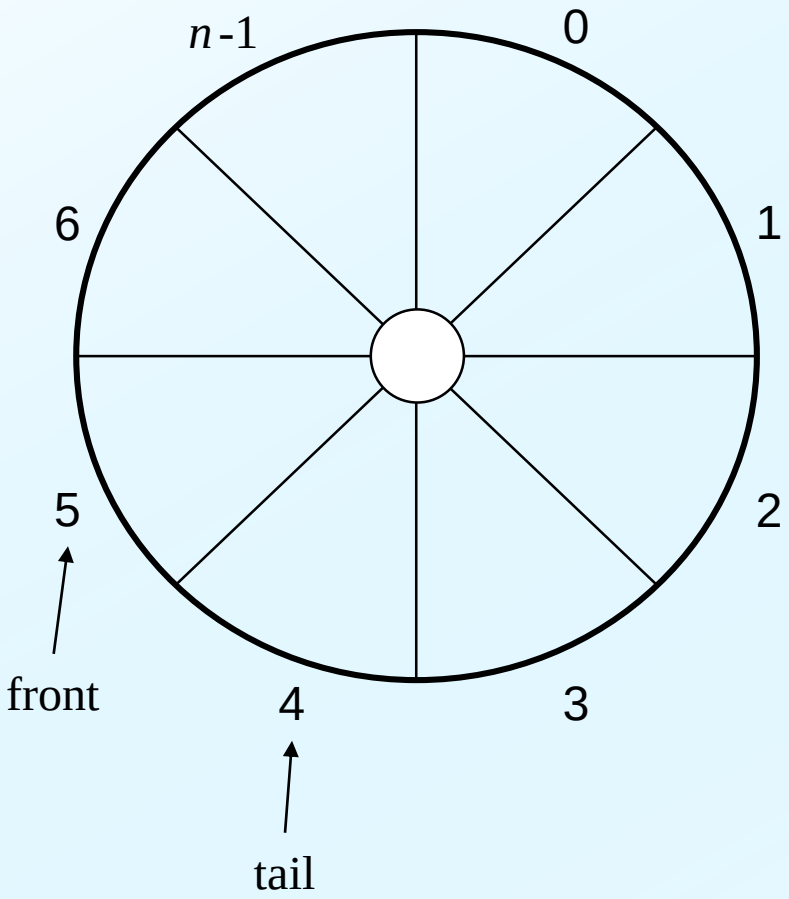


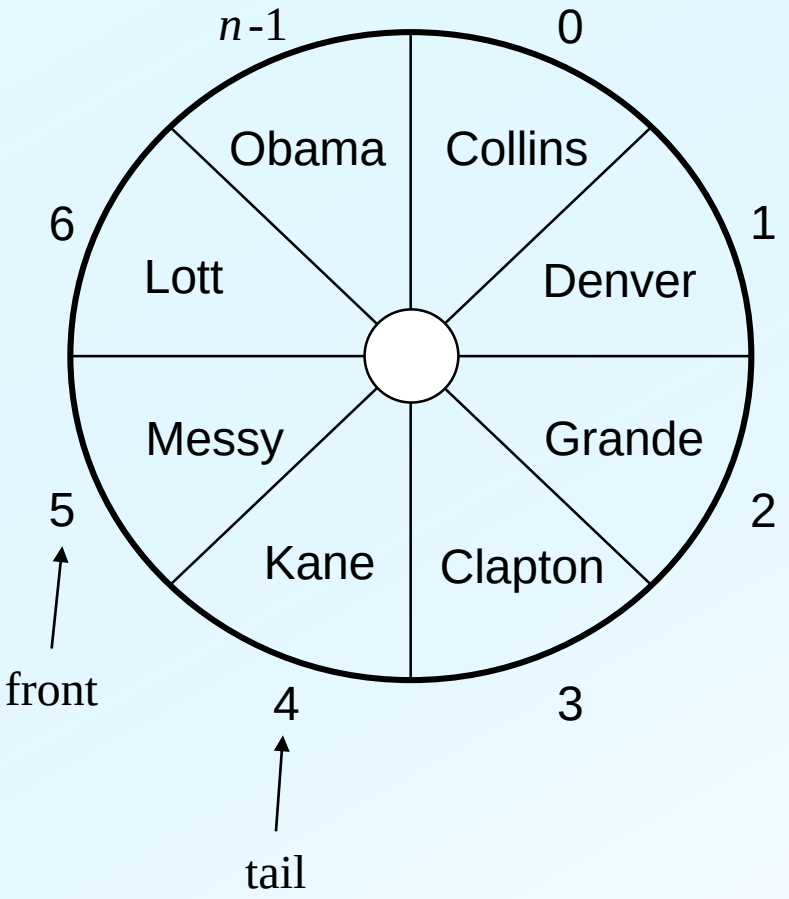
경계를 넘는 예

front ← (front+1) % queue.length



front와 tail의 상대 위치로
Empty와 Full을 구분할 수 없다





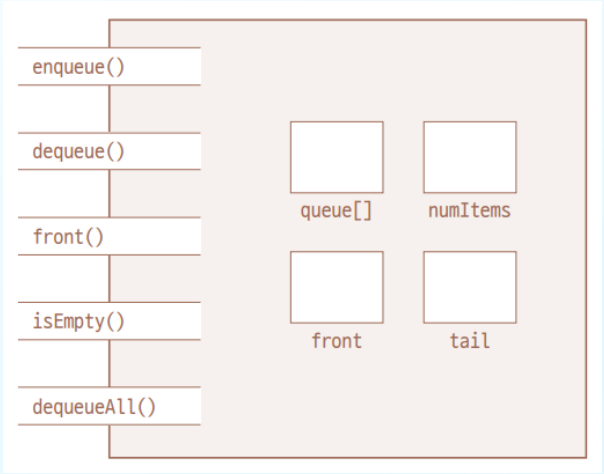
기타 작업

```
front():
        if (isEmpty())  /* 에러 처리 */
        else return queue[front]
```

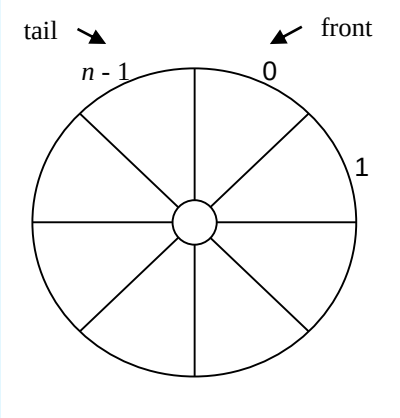
```
isEmpty():
        if (numItems = 0)
                return true
        else
                return false
```

```
dequeueAll():
        front ← 0
        tail ← queue.length - 1
        numItems ← 0
```

Java 구현




```
public interface QueueInterface<E> {
        public void enqueue(E newItem);
        public E dequeue();
        public E front( );
        public boolean isEmpty();
        public void dequeueAll();
}
```




```
public class ArrayQueue<E> implements QueueInterface<E> {
        private E queue[];
        private int front, tail, numItems;
        private static final int DEFAULT_CAPACITY = 64;
        public ArrayQueue(int n) { // 생성자 1
                queue = (E[]) new Object[n];
                front = 0;
                tail = n-1;
                numItems = 0;
        }
        public ArrayQueue() { // 생성자 2
                queue = (E[]) new Object[DEFAULT_CAPACITY];
                front = 0;
                tail = queue.length -1;
                numItems = 0;
        }
…
```

1/2 of class ArrayQueue

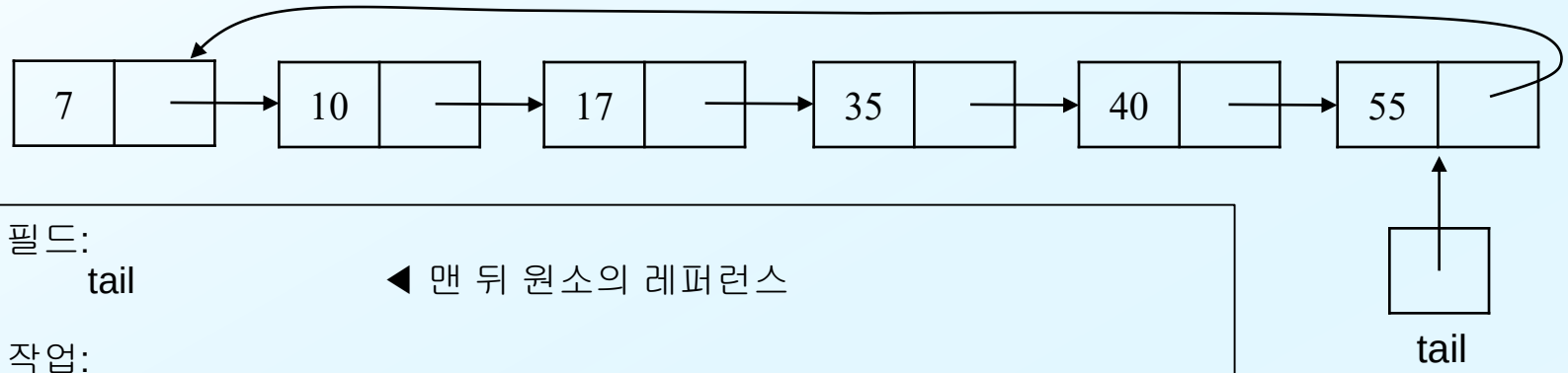
```
…
        private boolean isFull() {
                return (numItems == queue.length);
        }
        public void enqueue(E newItem) {
                if (isFull()) { /*에러 처리*/ }
                else {
                        tail = (tail+1) % queue.length;
                        queue[tail] = newItem;
                        numItems++;
                }
        }
        public E dequeue() {
                if (isEmpty()) { /*에러 처리*/ }
                else {
                        E queueFront = queue[front];
                        front = (front+1) % queue.length;
                        numItems--;
                        return queueFront;
                }
        }
        public E front() {
                if (isEmpty()) { /*에러 처리*/ }
                else return queue[front];
        }
        public boolean isEmpty() {
                return (numItems == 0);
        }
        public void dequeueAll() {
                queue = (E[]) new Object[queue.length];
                front = 0;
                tail = queue.length - 1;
                numItems = 0;
        }
} // class ArrayQueue<>
```

이 클래스의 객체를 생성해서 사용할 때는

```
// 타입 E를 String으로 결정하면서 수행하는 예
ArrayQueue<String> q = new ArrayQueue<>();
q.enqueue("test 1");
q.dequeue();
if (q.isEmpty()) …
```

3. Linked Queue

Linked Queue 객체 구조

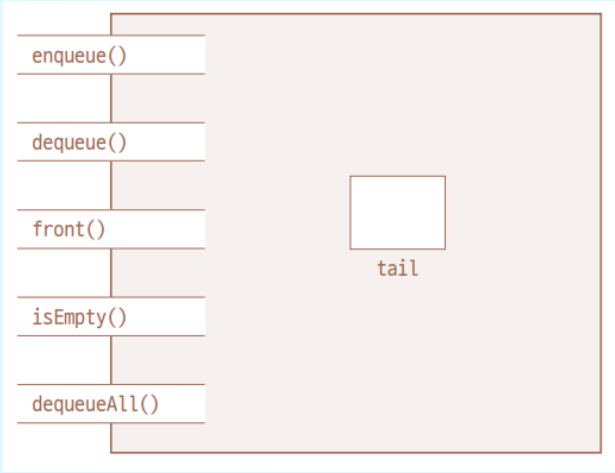


필드:

tail ◀ 맨 뒤 원소의 레퍼런스

작업:

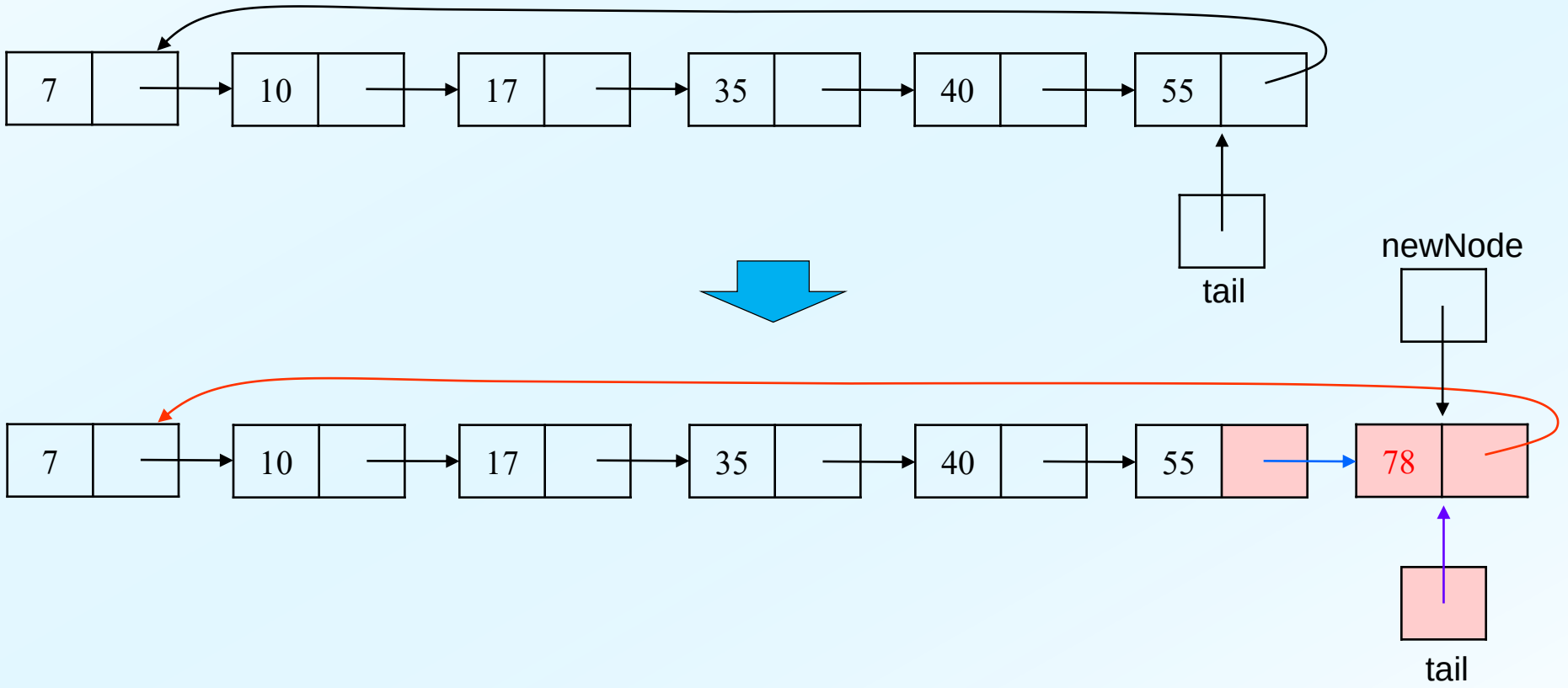
enqueue() ◀ 큐의 맨 뒤에 원소를 삽입한다
dequeue() ◀ 큐의 맨 앞에 있는 원소를 알려주고 삭제한다
front() ◀ 큐의 맨 앞에 있는 원소를 알려준다
isEmpty() ◀ 큐가 빈 큐인지를 알려준다
dequeueAll() ◀ 큐를 깨끗이 청소한다



삽입 Insertion

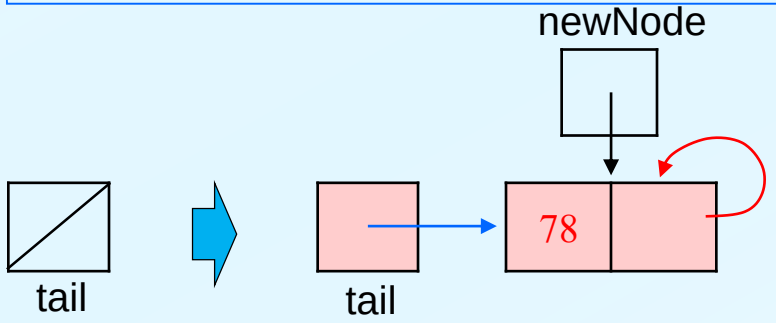
보통의 경우

newNode.item ← *x*
newNode.next ← tail.next
tail.next ← newNode
tail ← newNode



첫 삽입의 경우

```
newNode.item ← x
newNode.next ← newNode
tail ← newNode
```



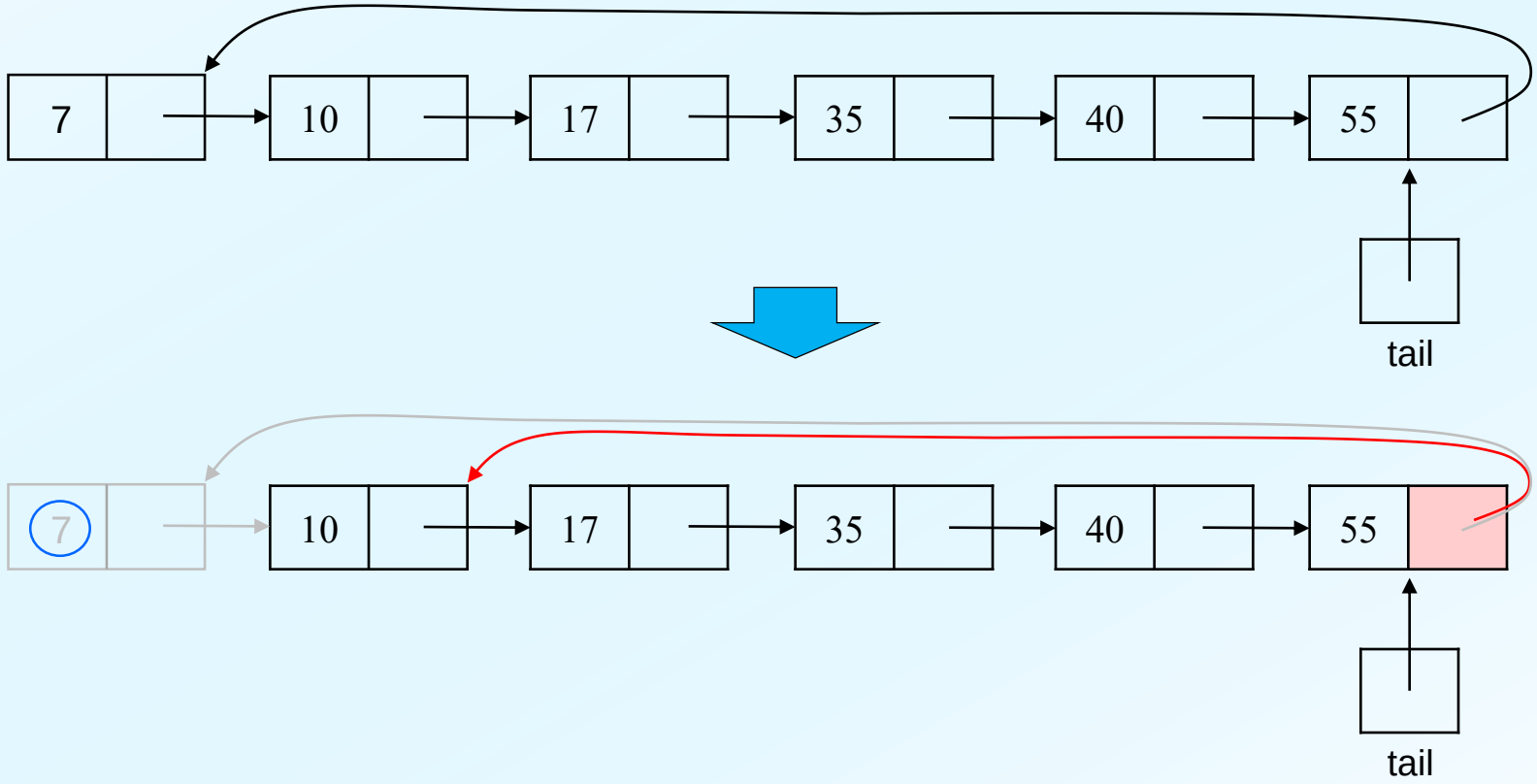
다 합치면..

```
enqueue(x):
        if (isEmpty())
                newNode.item ← x
                newNode.next ← newNode
                tail ← newNode
        else
                newNode.item ← x
                newNode.next ← tail.next
                tail.next ← newNode
                tail ← newNode
```

삭제 Deletion

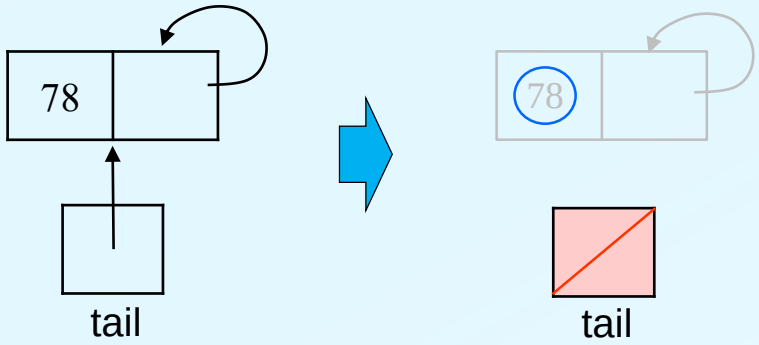
보통의 경우

```
tmp ← tail.next.item
tail.next ← tail.next.next
return tmp
```



원소가 1개인 경우

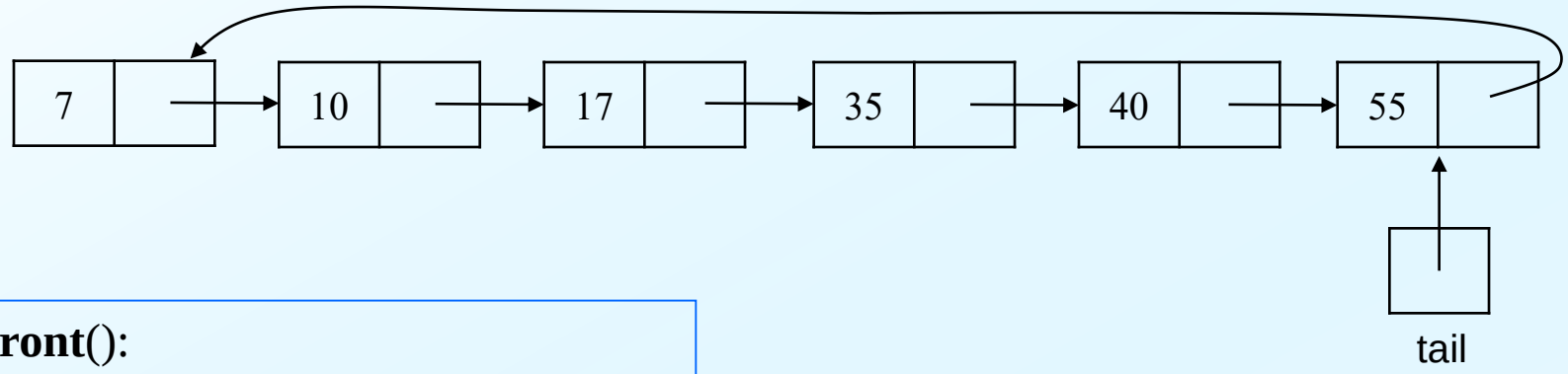
```
tmp ← tail.next.item  // 또는 tmp ← tail.item
tail ← null
return tmp
```



다 합치면..

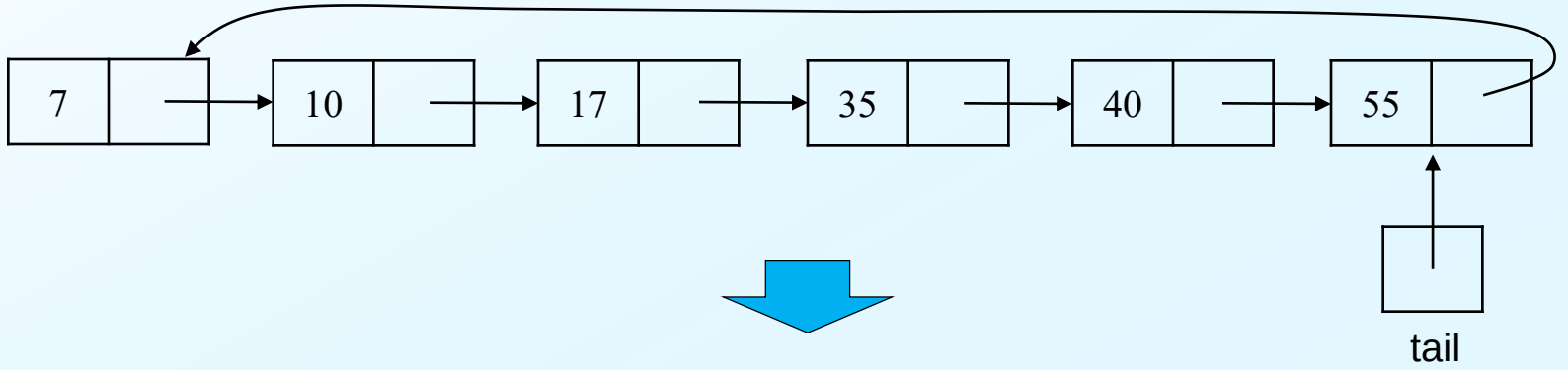
```
dequeue():
        if (isEmpty())
                /*에러 처리*/
        else
                tmp ← tail.next.item
                if (tail.next = tail) // 원소가 하나뿐
                        tail ← null
                else                  // 2개 이상의 원소
                        tail.next ← tail.next.next
                return tmp
```

기타 작업

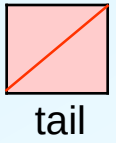



```
front():
        if (isEmpty())  /* 에러 처리 */
        else return tail.next.item
```

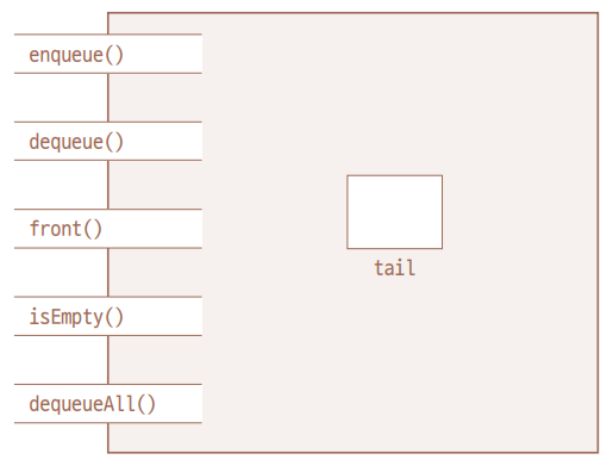
```
isEmpty():
        if (tail = null)
                return true
        else
                return false
```



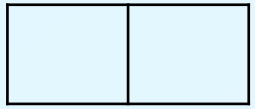

```
dequeueAll():
        tail ← null
```



Java 구현




```
public interface QueueInterface<E> {
        public void enqueue(E newItem);
        public E dequeue();
        public E front();
        public boolean isEmpty();
        public void dequeueAll();
}
```




```
public class Node<E> {
    public E item;
    public Node<E> next;
    public Node(E newItem) {
        item = newItem;
        next = null;
    }
    public Node(E newItem, Node<E> nextNode) {
        item = newItem;
        next = nextNode;
    }
}
```

```
public class LinkedQueue<E> implements QueueInterface<E> {
        private Node<E> tail;
        public LinkedQueue() {
                tail = null;
        }
        public void enqueue(E newItem) {
                Node<E> newNode = new Node<>(newItem);
                if (isEmpty()) {
                        newNode.next = newNode;
                        tail = newNode;
                } else {
                        newNode.next = tail.next;
                        tail.next = newNode;
                        tail = newNode;
                }

        }
…
```

1/2 of class LinkedQueue

```
        public E dequeue() {
            if (isEmpty()) { /* 에러 처리 */ }
            else {
                E tmp = tail.next.item;
                if (tail.next == tail) { // 원소가 하나뿐
                    tail = null;
                } else { // 2개 이상의 원소
                    tail.next = tail.next.next;
                }
                return tmp;
            }
        }
        public E front() {
            if (isEmpty()) { /* 에러 처리 */ }
            else return tail.next.item;  // tail.next: front
        }
        public boolean isEmpty() {
            return (tail == null);
        }
        public void dequeueAll() {
            tail = null;
        }
} // End LinkedQueue<>
```

이 클래스의 객체를 생성해서 사용할 때는

```
// 타입 E를 String으로 결정하면서 수행하는 예
LinkedQueue<String> q = new LinkedQueue<>();
q.enqueue(“test 1”);
q.dequeue();
if (q.isEmpty()) …
```

4. Queue by Reusing

상속

```
public class InheritedQueue<E> extends LinkedList<E>
                                        implements QueueInterface<E> {
        public InheritedQueue() {
                super();
        }
        public void enqueue(E newItem) {
                append(newItem);
        }
        public E dequeue() {
                return remove(0);
        }
        public E front() {
                return get(0);
        }
        public void dequeueAll() {
                clear();
        }
} // End InheritedQueue<>
```

또는.. super를 명시해도 된다

super: parent class

```
public class InheritedQueue<E> extends LinkedList<E>
                                    implements QueueInterface<E> {
        public InheritedQueue() {
                super();
        }
        public void enqueue(E newItem) {
                super.append(newItem);
        }
        public E dequeue() {
                return super.remove(0);
        }
        public E front() {
                return super.get(0);
        }
        public void dequeueAll() {
                super.clear();
        }
} // End InheritedQueue<>
```

비교

```
public class LinkedQueue<E> implements QueueInterface<E> {
        private Node<E> tail;
        public LinkedQueue() {
                tail = null;
        }
        public void enqueue(E newItem) {
                Node<E> newNode = new Node<>(newItem);
                if (isEmpty()) {
                        newNode.next = newNode;
                        tail = newNode;
                } else {
                        newNode.next = tail.next;
                        tail.next = newNode;
                        tail = newNode;
                }
        }
        public E dequeue() {
                if (isEmpty()) { /* 에러 처리 */ }
                else {
                        Node<E> front = tail.next;
                        if (front == tail)  // 원소가 하나뿐
                                tail = null;
                        else // 2개 이상의 원소
                                tail.next = front.next;
                        return front.item;
                }
        }
        public E front() {
                if (isEmpty()) { /* 에러 처리 */ }
                else return tail.next.item;  // tail.next: front
        }
        public boolean isEmpty() {
                return (tail == null);
        }
        public void dequeueAll() {
                tail = null;
        }
} // End LinkedQueue<>
```

```
public class InheritedQueue<E> extends LinkedList<E>
                                implements QueueInterface<E> {
        public InheritedQueue() {
                super();
        }
        public void enqueue(E newItem) {
                append(newItem);
        }
        public E dequeue() {
                return remove(0);
        }
        public E front() {
                return get(0);
        }
        public void dequeueAll() {
                clear();
        }
} // End InheritedQueue<>
```

재사용

```
public class ListQueue<E> implements QueueInterface<E> {
        private ListInterface<E> list;
        public ListQueue() {
                list = new LinkedList<>(); // 또는 ArrayList<>()
        }
        public void enqueue(E newItem) {
                list.append(newItem);;
        }
        public E dequeue() {
                return list.remove(0);
        }
        public E front() {
                return list.get(0);
        }
        public boolean isEmpty() {
                return list.isEmpty();
        }
        public void dequeueAll() {
                list.clear();
        }
} // End ListQueue<>
```

비교

```java
public class LinkedQueue<E> implements QueueInterface<E> {
    private Node<E> tail;
    public LinkedQueue() {
        tail = null;
    }
    public void enqueue(E newItem) {
        Node<E> newNode = new Node<>(newItem);
        if (isEmpty()) {
            newNode.next = newNode;
            tail = newNode;
        } else {
            newNode.next = tail.next;
            tail.next = newNode;
            tail = newNode;
        }
    }
    public E dequeue() {
        if (isEmpty()) { /* 에러 처리 */ }
        else {
            Node<E> front = tail.next;
            if (front == tail)  // 원소가 하나뿐
                tail = null;
            else // 2개 이상의 원소
                tail.next = front.next;
            return front.item;
        }
    }
    public E front() {
        if (isEmpty()) { /* 에러 처리 */ }
        else return tail.next.item;  // tail.next: front
    }
    public boolean isEmpty() {
        return (tail == null);
    }
    public void dequeueAll() {
        tail = null;
    }
} // End LinkedQueue<>
```

```java
public class ListQueue<E> implements QueueInterface<E> {
    private ListInterface<E> list;
    public ListQueue() {
        list = new LinkedList<>(); // 또는 ArrayList<>()
    }
    public void enqueue(E newItem) {
        list.append(newItem);;
    }
    public E dequeue() {
        return list.remove(0);
    }
    public E front() {
        return list.get(0);
    }
    public boolean isEmpty() {
        return list.isEmpty();
    }
    public void dequeueAll() {
        list.clear();
    }
} // End ListQueue<>
```

5. Queue 활용 예

Palindrome 체크

```
public class Palindrome {
        public static void main(String[] args) { // 테스트용 main
                System.out.println("Palindrome Check!");
                String str = "lioninoil"; // 테스트 문자열
                boolean t = isPalindrome(str);
                System.out.println("Is " + str + " Palindrome?: " + t);
        }
        private static boolean isPalindrome(String A) {
                ArrayStack<Character> s = new ArrayStack<>(A.length);
                ArrayQueue<Character> q = new ArrayQueue<>(A.length);
                for (int i = 0; i < A.length(); i++) {
                        s.push(A.charAt(i)); // 문자열 A의 i번 문자
                        q.enqueue(A.charAt(i));
                }
                while ( !q.isEmpty() && s.pop() == q.dequeue() ) {
                }
                if (q.isEmpty()) return true;
                else return false;
        }
}
```

출력

```
Palindrome Check!
Is lioninoil Palindrome?: true
```