큐를 배열로 구현한 ArrayQueue 클래스

```
01 import java.util.NoSuchElementException;
02 public class ArrayQueue <E>{
03     private E[] q;     // 큐을 위한 배열
04     private int front, rear, size;
05     public ArrayQueue() {          // 큐 생성자
06         q = (E[]) new Object[2]; // 초기에 크기가 2인 배열 생성
07         front = rear = size = 0;
08     }
09     public int     size()    { return size;}          // 큐에 있는 항목의 수를 리턴
10     public boolean isEmpty() { return (size == 0);} // 큐가 empty이면 true를 리턴

       // add(), remove(), resize() 메소드 선언
   }
```

- Line 01: 큐에서 underflow가 발생했을 때 java 라이브러리에 선언된 NoSuchElementException을 이용하여 프로그램 정지
  - 참고로 EmptyQueueException은 자바 라이브러리에 선언되어 있지 않음
- Line 05~08: ArrayQueue 객체 생성자로, 객체는 1차원 배열 q와 3개의 필드인 front, rear, size를 가짐
  - 초기의 배열 크기는 2이고, 각 필드를 0으로 초기화
- Line 09: 큐의 item 수를 리턴하는 메소드
- Line 10: 큐가 empty이면 true를 리턴하는 메소드

```
44      public static void main(String[] args) {
45          ListQueue<String> q = new ListQueue<String>();
46              q.remove();
47              q.add("apple");      q.add("orange");
48              q.add("cherry");     q.add("pear");
49              q.print();
50
51              q.remove(); q.print();
52              q.remove(); q.print();
53
54              q.add("grape");      q.print();
55      }
56  }
```

Problems @ Javadoc Console

```
<terminated> ListQueue [Java Application] C:₩Program Files₩Java₩jdk1.8.0_40₩bin₩javaw.exe
Exception in thread "main" java.util.NoSuchElementException
        at ListQueue.remove(ListQueue.java:23)
        at ListQueue.main(ListQueue.java:46)
```

```
01 public void add(E newItem) {    // 큐 삽입 연산
02     if ((rear+1)%q.length == front) // 비어있는 원소가 1개뿐인 경우(즉, 큐가 full인 경우)
03         resize(2*q.length);     // 큐의 크기를 2배로 확장
04     rear = (rear+1) % q.length;
05     q[rear] = newItem;          // 새 항목을 add
06     size++;
07 }
```

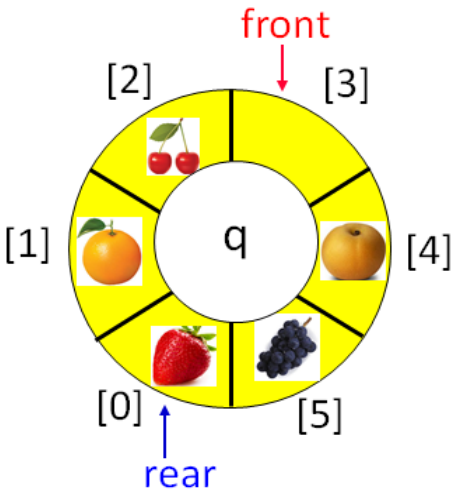
- **add() 메소드: 큐에 새 item을 삽입**
  - Line 02: 삽입할 빈 자리가 있는지 확인한다. 만일 (rear+1)%q.length (즉, rear 다음 원소의 인덱스)가 front와 같으면 overflow 가 발생한 것이므로, resize() 메소드를 호출하여 배열을 2배 크기로 확장
  - Line 04: rear를 (rear+1)%q.length로 갱신한 후, line 05에서 newItem을 q[rear]에 저장
  - Line 06: size 1 증가

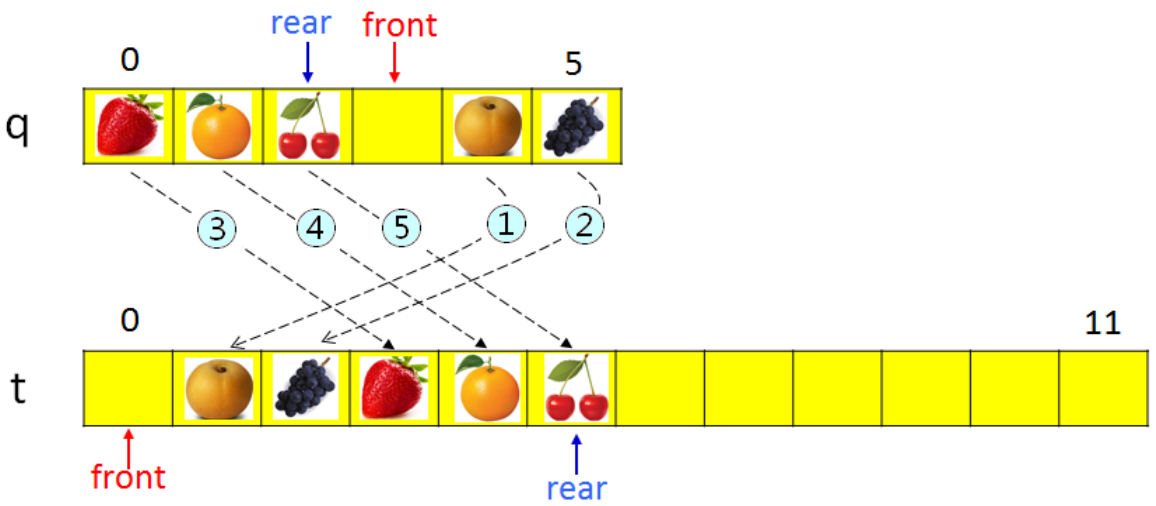
```
01 public E remove() {  //큐 삭제 연산
02     if (isEmpty()) throw new NoSuchElementException(); // underflow 경우 프로그램 정지
03     front  = (front+1) % q.length;
04     E item = q[front];
05     q[front] = null;     // null로 만들어 가비지 컬렉션되도록
06     size--;
07     if (size > 0 && size == q.length/4) // 큐의 항목수가 배열 크기의 1/4가 되면
08         resize(q.length/2);             // 큐를 1/2 크기로 축소
09     return item;
10 }
```

- **remove() 메소드: 큐에서 item을 삭제**
  - Line 02: underflow를 체크하고 underflow 발생시 NoSuchElementException을 throw하여 프로그램 정지
  - front를 (front+1)%q.length로 갱신한 후, q[front]를 변수 item에 저장하여 line 09에서 리턴
  - Line 05~06: q[front]를 null로 만들어 가비지 컬렉션이 되도록 하고, size를 1 감소
  - Line 07~08: 삭제 후 배열에 1/4만큼만 item들로 채워져 있으면 배열 크기를 1/2로 감소시키기 위해 resize() 메소드 호출

```
01  private void resize(int newSize) {     // 큐의 배열 크기 조절
02      Object[] t = new Object[newSize]; // newSize 크기의 새로운 배열 t 생성
03      for(int i = 1, j=front+1; i <size+1; i ++, j++){
04          t[i] = q[j%q.length];          // 배열q의 항목들을 배열 t[1]로부터 복사
05      }
06      front = 0;
07      rear  = size;
08      q = (E[]) t;        // 배열 t를 배열 q로
09  }
```

- **resize() 메소드 2.1절의 resize()와 거의 동일**
  - Line 06~07: front를 0으로 rear는 큐에 있는 item의 수인 size로 갱신하고,
  - Line 08: q가 새 배열 t를 참조





```java
public class main {
    public static void main(String[] args) {
        ArrayQueue<String> queue = new ArrayQueue<String>();
        queue.add("apple");    queue.add("orange");
        queue.add("cherry");   queue.add("pear");     queue.print();
        queue.remove();        queue.print();
        queue.add("grape");    queue.print();
        queue.remove();        queue.print();
        queue.add("lemon");    queue.print();
        queue.add("mango");    queue.print();
        queue.add("lime");     queue.print();
        queue.add("kiwi");     queue.print();
        queue.remove();        queue.print();
    }
}
```

Problems @ Javadoc Console Console

<terminated> ArrayQueue [Java Application] C:₩Program Files₩Java₩jdk1.8.0_40₩bin₩javaw.exe

null	apple	orange	cherry	pear	null	null	null
null	null	orange	cherry	pear	null	null	null
null	null	orange	cherry	pear	grape	null	null
null	null	null	cherry	pear	grape	null	null
null	null	null	cherry	pear	grape	lemon	null
null	null	null	cherry	pear	grape	lemon	mango
lime	null	null	cherry	pear	grape	lemon	mango
lime	kiwi	null	cherry	pear	grape	lemon	mango
lime	kiwi	null	null	pear	grape	lemon	mango

front

rear

큐를 연결리스트로 구현한 ListQueue 클래스

```
01 import java.util.NoSuchElementException;
02 public class ListQueue <E> {
03     private Node<E> front, rear;
04     private int size;
05     public ListQueue() {    // 생성자
06         front = rear = null;
07         size  = 0;
08     }
09     public int     size()    { return size; }         // 큐의 항목의 수를 리턴
10     public boolean isEmpty() { return size() == 0; } // 큐가 empty이면 true리턴

       // add(), remove() 메소드 선언
   }
```

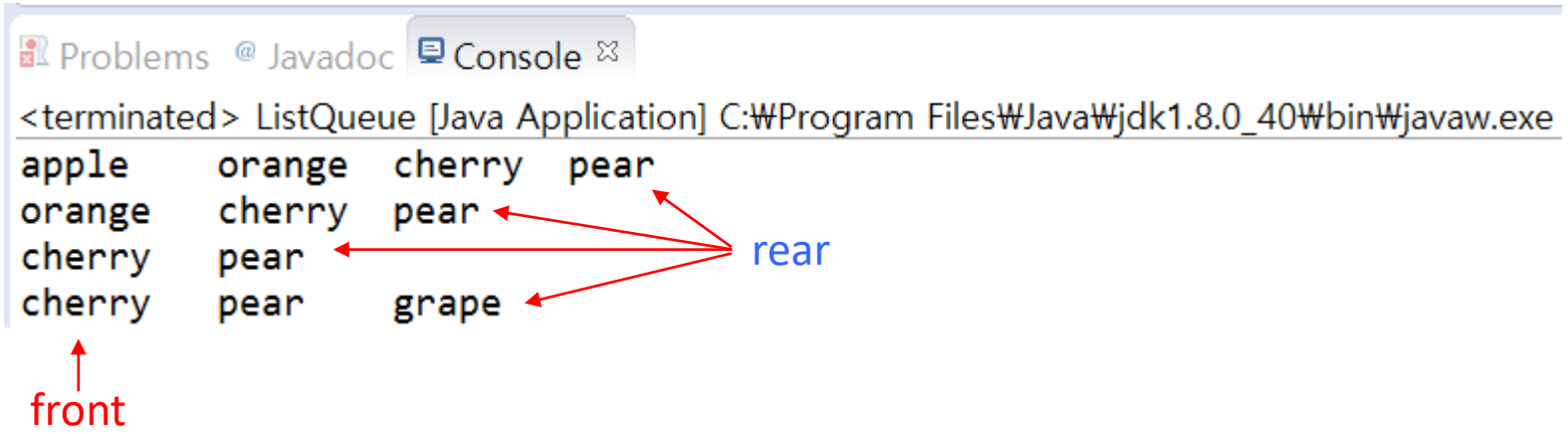
- Node 클래스:2.2절의 Node 클래스와 동일
- Line 01: java.util 라이브러리에 선언되어 있는 NoSuchElementException 클래스이고, underflow 발생 시 프로그램 종료
- Line 05~08: ListQueue 객체의 생성자, ListQueue객체는 큐의 첫 item을 가진 Node 레퍼런스를 저장하는 front와 마지막 Node를 가리키는 rear, 큐의 item 수를 저장하는 size를 가진다.
- Line 09~10: 각각 스택의 item 수를 리턴하고, 스택이 empty이면 true를 리턴하는 메소드

```
01 public void add(E newItem){
02     Node newNode = new Node(newItem, null);       // 새 노드 생성
03     if (isEmpty()) front = newNode; // 큐가 empty이었으면 front도 newNode를 가리키게 한다
04     else rear.setNext(newNode);     // 그렇지않으면 rear의 next를 newNode를 가리키게 한다
05     rear = newNode;                 // 마지막으로 rear가 newNode를 가리키게 한다
06     size++;                         // 큐 항목 수 1 증가
07 }
```

- **add() 메소드: 새 item을 큐의 뒤(rear)에 삽입**
  - Line 02: 노드를 생성
  - Line 03: 연결리스트가 empty인 경우 front가 새 노드를 가리키도록
  - 연결 리스트가 empty가 아닌 경우 line 04에서 rear가 참조하는 노드의 next가 새 노드(newNode)를 가리키도록 하여 새 노드를 연결리스트의 마지막 노드로 연결
  - Line 05: rear가 새 노드를 가리키게 하고, line 06에서 size 1 증가

```
1    public E remove() {
2        if (isEmpty()) throw new NoSuchElementException(); // underflow 경우에 프로그램 정지
3        E frontItem = front.getItem();  // front가 가리키는 노드의 항목을 frontItem에 저장
4        front = front.getNext();        // front가 front 다음 노드를 가리키게 한다.
5        if (front == null) rear = null;     // 큐가 empty이면 rear = null
6        size--;                         // 큐 항목 수 1 감소
7        return frontItem;
8    }
```

- **remove() 메소드: item을 큐의 앞(front)에서 삭제**
  - Line 2: underflow를 검사
  - Line 3: front가 가리키는 노드의 item을 line 07에서 리턴
  - Line 5: 삭제 후 연결리스트가 empty가 된 경우 rear를 null로 갱신
  - Line 6: size 1 감소



큐 자료구조의 응용

- CPU의 태스크 스케줄링(Task Scheduling)
- 네트워크 프린터
- 실시간(Real-time) 시스템의 인터럽트(Interrupt) 처리
- 다양한 이벤트 구동 방식(Event-driven) 컴퓨터 시뮬레이션
- 콜 센터의 전화 서비스 처리 등
- 4장의 이진트리의 레벨순서 순회(Level-order Traversal)
- 9장의 그래프에서 너비우선탐색(Breath-First Search) 등

수행시간

- 배열로 구현한 큐의 add와 remove 연산은 각각 O(1) 시간이 소요
- 배열 크기를 확대 또는 축소시키는 경우에 큐의 모든 item들을 새 배열로 복사해야 하므로 O(N) 시간이 소요. 상각분석: 각 연산의 평균 수행시간은 O(1)
- 단순연결리스트로 구현한 큐의add와 remove 연산은 각각 O(1) 시간, 삽입 또는 삭제 연산이 rear 와 front로 인해 연결리스트의 다른 노드들을 일일이 방문할 필요 없기 때문
- 배열과 단순연결리스트로 구현한 큐의 장단점은 2장의 리스트를 배열과 단순연결리스트로 구현하였을 때의 장단점과 동일

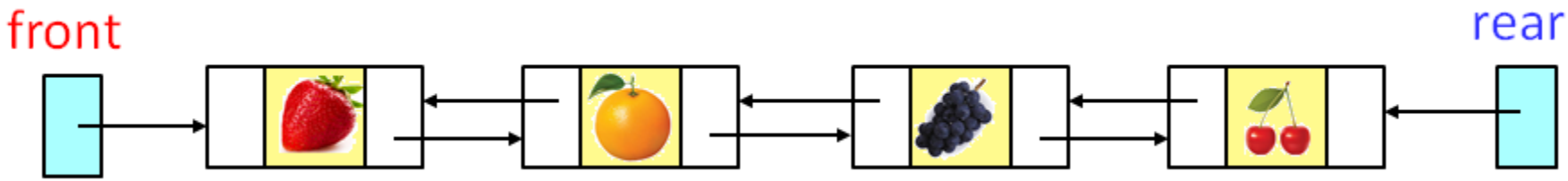
3.3 데크

- **데크(Double-ended Queue, Deque): 양쪽 끝에서 삽입과 삭제를 허용하는 자료구조**
- **데크는 스택과 큐 자료구조를 혼합한 자료구조**
- **따라서 데크는 스택과 큐를 동시에 구현하는데 사용**

- **응용**
  - 스크롤(Scroll)
  - 웹 브라우저의 방문 기록 등
    - 웹 브라우저 방문 기록의 경우, 최근 방문한 웹 페이지 주소는 앞에 삽입하고, 일정 수의 새 주소들이 앞쪽에서 삽입되면 뒤에서 삭제가 수행
  - 개수 제한이 있는 문서편집기의 undo

데크의 구현

- 데크를 이중연결리스트로 구현하는 것이 편리
- 단순연결리스트는 rear가 가리키는 노드의 이전 노드의 레퍼런스를 알아야 삭제가 가능하기 때문



수행시간

- 데크를 배열이나 이중연결리스트로 구현한 경우, 스택과 큐의 수행시간과 동일
- 양 끝에서 삽입과 삭제가 가능하므로 프로그램이 다소 복잡
- 이중연결리스트로 구현한 경우는 더 복잡함
- 자바 SE 7은 java.util 패키지에서 Deque 인터페이스를 제공, Queue 클래스에서 상속됨

요약

- 스택은 한 쪽 끝에서만 item을 삭제하거나 새로운 item을 저장하는 후입선출(LIFO) 자료구조
- 스택은 컴파일러의 괄호 짝 맞추기, 회문 검사하기, 후위표기법수식 계산하기, 중위표기법 수식을 후위표기법으로 변환하기, 미로 찾기, 트리의 노드 방문, 그래프의 깊이우선탐색에 사용. 또한 프로그래밍에서 매우 중요한 메소드 호출 및 재귀호출도 스택 자료구조를 바탕으로 구현
- 큐는 삽입과 삭제가 양 끝에서 각각 수행되는 선입선출(FIFO) 자료구조

요약

- 배열로 구현된 큐에서 삽입과 삭제를 거듭하게 되면 큐의 item들이 오른쪽으로 편중되는 문제가 발생한다. 이를 해결하기 위한 방법은 배열을 원형으로, 즉, 배열의 마지막 원소가 첫 원소와 맞닿아 있다고 생각하는 것
- 큐는 CPU의 태스크 스케줄링, 네트워크 프린터, 실시간 시스템의 인터럽트 처리, 다양한 이벤트 구동 방식 컴퓨터 시뮬레이션, 콜 센터의 전화 서비스 처리 등에 사용되며, 이진트리의 레벨순회와 그래프의 너비우선 탐색에 사용
- 데크는 양쪽 끝에서 삽입과 삭제를 허용하는 자료구조로서 스택과 큐 자료구조를 혼합한 자료구조
- 데크는 스크롤, 문서 편집기의 undo 연산, 웹 브라우저의 방문 기록 등에 사용

스택, 큐, 데크 자료구조의 연산 수행시간 비교

자료구조	구현	삽입	삭제	비고
스택 큐 데크	배열	O(1)*	O(1)*	* 상각분석 평균시간(부록 I)
	연결리스트†	O(1)	O(1)	†데크는 이중연결리스트로 구현