**자료구조 실습 보고서**

[제10주] 큐(Queue)

**2021년 05월 10일**

**201702039 오명주**

1. **프로그램 설명서**
2. **프로그램의 전체 설계 구조**

* MVC (Model – View – Controller) 구조

Model : 프로그램이 “무엇”을 할 것인지 정의. 사용자의 요청에 맞는 알고리즘을 처리하고 DB와 상호작용하여 결과물을 산출하고 Controller에게 전달.

View : 화면에 무엇인가를 “보여주기 위한” 역할. 최종 사용자에게 “무엇”을 화면으로 보여줌.

Controller : 모델이 “어떻게” 처리할 지 알려주는 역할. 사용자로부터 입력을 받고 중개인 역할. Model과 View는 서로 직접 주고받을 수 없음. Controller을 통해 이야기함.

* ArrayQueue 프로그램에서의 각 클래스 별 MVC 구조 역할

**Model** :

* Interface Queue : Queue의 변수와 속성을 구성한다.
* CircularArrayQueue : Queue를 이용한 순환 배열 큐를 구성한다.
* Iterator : 반복자의 변수와 함수로 이루어져있다.

**View** :

* AppView : 프로그램의 입/출력을 담당한다.

**Controller** :

* AppController : Model을 통해 생성된 결과물을 AppView를 통해 출력한다.
* LinkedQueue 프로그램에서의 각 클래스 별 MVC 구조 역할

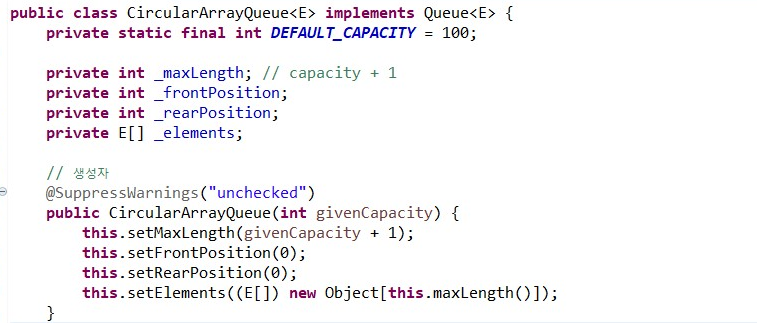
**Model** :

* Interface Queue : Queue의 변수와 속성을 구성한다.
* CircularlyLinkedQueue : Queue를 이용한 순환 연결리스트 큐를 구성한다.
* Iterator : 반복자의 변수와 함수로 이루어져 있다.
* LinkedNode<E> : 연결 노드에 대한 변수와 속성을 구성한다.

1. **함수 설명서**

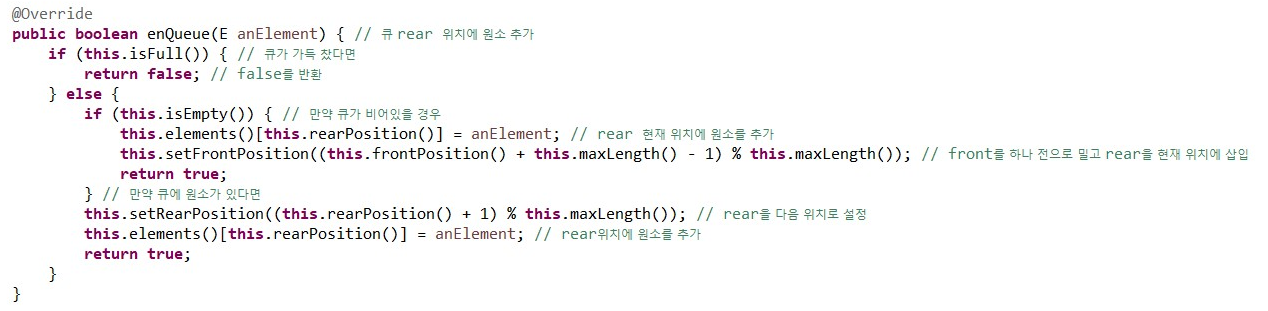
* 주요 알고리즘

1. CircularArrayQueue



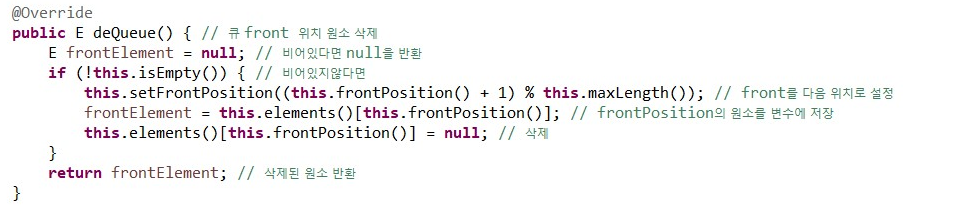
배열로 구현한 순환 원형 큐 클래스이다.

* 배열의 크기가 10이라면 maxlength를 10으로 설정, capacity는 9로 설정함으로써 9만큼 원소가 들어가면 배열이 가득 찼다고 판단한다. rear=0, front=0이면 empty인지 full인지 모르기 때문에 front=0, rear=9면 가득 찼다고 판단한다.
* 생성자 초기화는 maxlength로 진행한다.



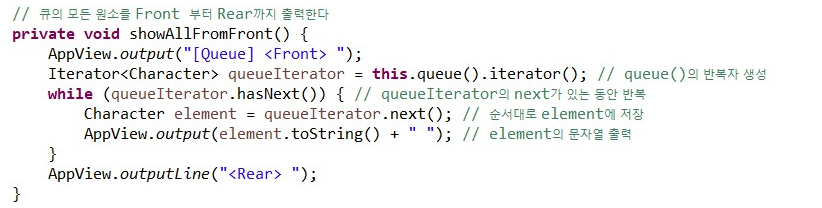
rear 위치에 원소를 추가하는 enQueue 함수.

* 만약 큐가 full 상태라면 false를 반환한다.
* 만약 큐가 비어 있는 상태라면, rear와 front 위치가 같기 때문에 front를 뒤로 밀고 현재 rear자리에 원소를 삽입한다. 이때 front 자리를 0인경우가 있기 때문에 maxlength를 더한 후 빼준다. 나머지 연산을 통해 0->10으로 가는 경우를 계산한다.
* 만약 큐에 원소가 있다면 setRearPosition을 이용하여 다음 rear위치를 잡는다. 이때 나머지 연산을 하는 이유는 10->0 인덱스를 계산해주기 위함이다. rearPosition에 원소를 추가하고 true를 반환한다.



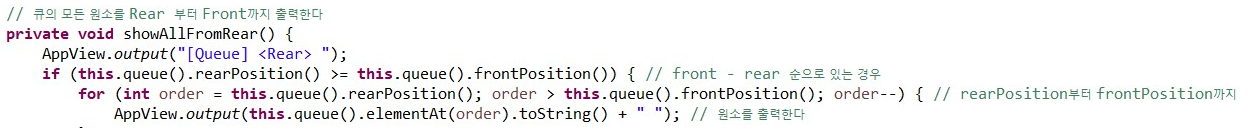
front 위치에 원소를 삭제하는 deQueue 함수.

* 삭제할 원소를 저장할 변수를 선언하고 null로 초기화한다.
* 큐가 empty 상태가 아니라면
* frontPosition을 다음 위치로 설정한다. front는 null을 가리키는 상태. 이때 나머지 연산을 하는 이유는 10->0 인덱스를 계산해주기 위함이다.
* frontPosition의 현재 위치를 변수에 저장하고 null로 삭제를 진행한다.
* 저장한 변수를 반환한다.



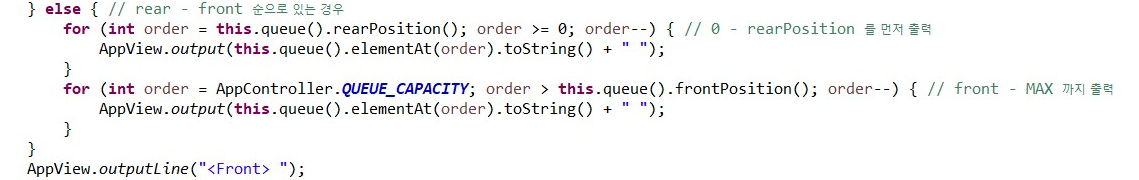
큐의 모든 원소를 front부터 rear까지 출력하는 showAllFromFront 함수.

* 반복자를 이용하여 큐의 next가 있는 동안 반복한다.
* element에 원소를 저장하여 차례로 출력한다.



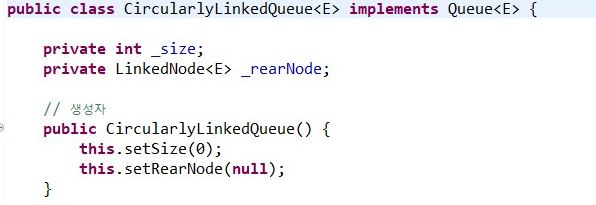
큐의 모든 원소를 rear부터 front까지 출력하는 showAllFromRear 함수.

* front가 rear보다 앞에있는 경우 ( 인덱스 숫자가 작은경우 )
* rearPosition부터 frontPosition까지 역순으로 출력



* rear가 front보다 앞에있는 경우,
* rearPosition부터 0까지 역순으로 출력 후, 배열의 MAX부터 frontPosition까지 역순으로 출력한다.

1. CircularlyLinkedQueue



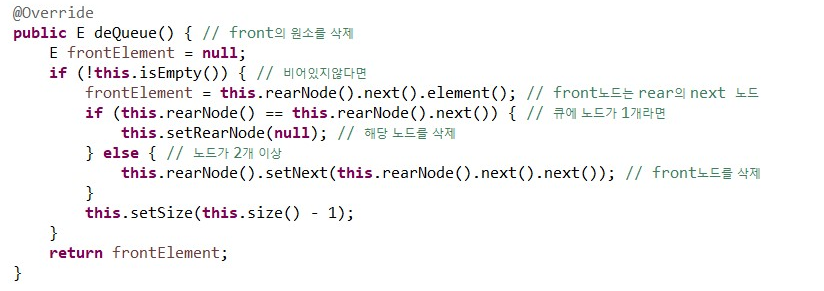
연결리스트로 구현한 순환 원형 큐 클래스이다.

* frontNode를 rearNode의 next로 정의하기 때문에 rearNode와 size만 선언한다.
* size는 null로, rearNode는 null로 초기화한다.



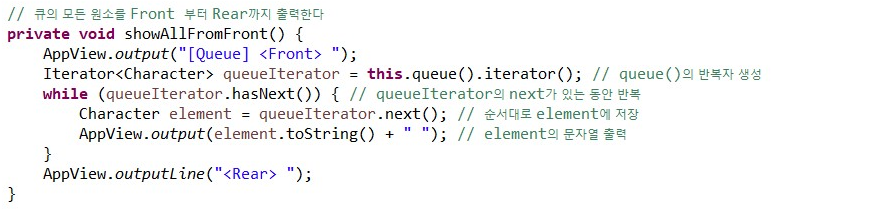
rear 위치에 원소를 추가하는 enQueue 함수.

* anElement를 element로 하는 newRearNode를 선언한다.
* 만약 큐가 비어있다면 하나의 원소만 순환하도록 자기자신을 setNext로 지정한다.
* 만약 비어있다면 newRearNode의 다음노드를 현재 rearNode로 지정하고 readNode의 다음 노드를 newRearNode로 지정한다.
* readNode를 newRearNode로 바꾸어 새로 지정해주고 사이즈를 +1 한다.



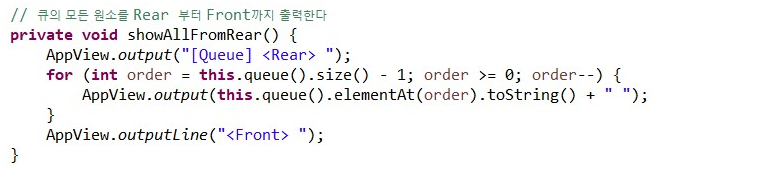
front 위치의 원소를 삭제하는 deQueue 함수

* 삭제할 원소를 저장할 변수를 선언하고 null로 초기화한다.
* 만약 큐가 비어있지 않다면 readNode의 next가 frontNode임으로 frontNode의 element를 변수에 저장한다.
* 만약 큐에 노드가 하나라면, 해당노드를 null처리하여 삭제한다.
* 만약 큐에 노드가 2개 이상이라면 frontNode를 삭제한다.
* 사이즈는 -1하여 설정한다.
* 삭제 원소를 반환



큐의 모든 원소를 front부터 rear까지 출력하는 showAllFromFront 함수.

* 반복자를 이용하여 큐의 next가 있는 동안 반복한다.
* element에 원소를 저장하여 차례로 출력한다.



큐의 모든 원소를 rear부터 front까지 출력하는 showAllFromRear 함수.

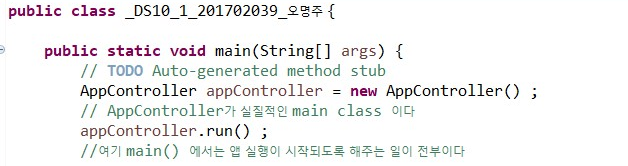
* 배열과는 다르게 연결리스트는 null이 없기 때문에 size-1만큼 걸어 나가 해당 원소를 출력한다.



* 주어진 anOrder만큼 next를 하여 걸어나가 해당 원소를 반환한다.

1. **종합 설명서**

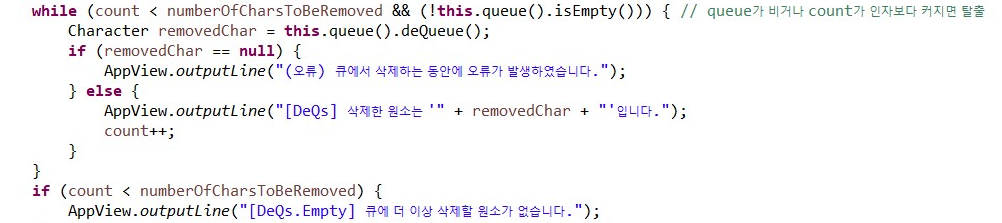
* 프로그램 실행 순서대로 설명해보자.



main에서 AppController 의 객체를 생성하여 run 한다. 프로그램을 실행한다.



* AppController의 run 함수에서는 프로그램을 실행한다.
* !가 입력되면 종료된다.
* 입력문자가 문자/알파벳이면 addToQueue 함수를 통해 큐에 삽입한다.
* 입력문자가 숫자면 removeN을 실행하여 입력문자만큼 삭제한다.
* 각각의 입력문자에 맞는 처리를 해준다.
* 그외 문자는 의미없는 문자로 count한다.
* ‘!’가 입력되면 통계를 출력하고 프로그램을 종료한다.



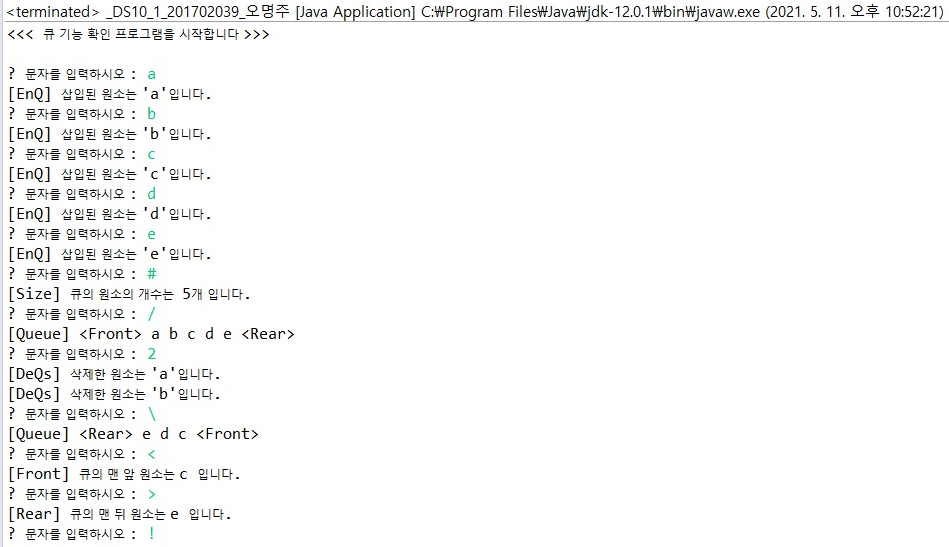
입력 받은 numberOfCharsToBeRemoved 정수만큼 && 큐가 empty가 아닌 동안 removeN을 반복한다.

만약 큐가 empty가 되어서 반복문을 탈출했을 때 입력받은 정수보다 count가 작으면 삭제할 원소가 없다는 출력문을 출력한다.

1. **프로그램 장단점 / 특이점 분석**

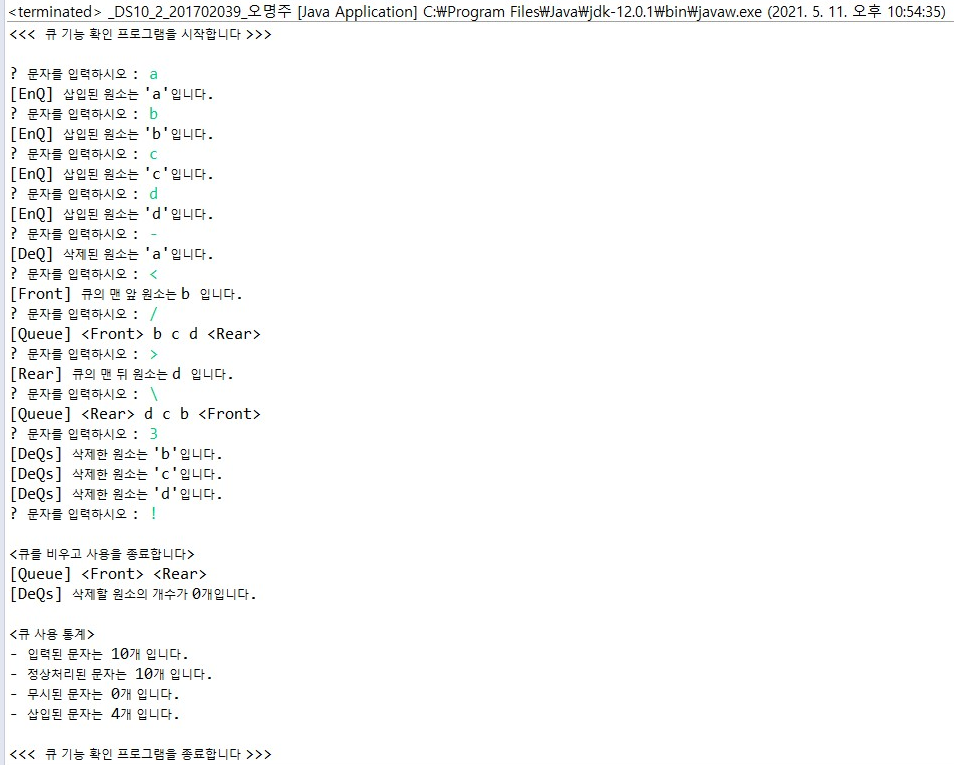
* **Array를 이용한 순환 원형 큐 장점**
* 배열로 구현하여 인덱스를 통한 원소 접근이 용이하다. 삭제 시, front 인덱스를 통하여 가능하고 삽입 시 rear 인덱스를 통해 쉽게 가능하다.
* Iterator을 통해 모든 원소를 차례로 출력할 수 있어 편리하였다.
* 일반 배열이라면 맨 앞 원소 삭제 시 뒤 모든 배열을 한칸씩 앞으로 옮겨야 하는 불편함이 있지만 순환 원형 배열이기 때문에 frontPosition 인덱스만 옮기면 되어서 편리하였다.
* **Array를 이용한 순환 원형 큐 단점**
* MAX\_CAPACITY를 미리 지정하여 최대 크기에 제한이 있다.
* 모든 원소 출력 시 front-rear 출력은 Iterator을 통해 쉽게 할 수 있었지만 rear-front 역순 출력은 배열에 null이 존재하기 때문에 frontPosition이 rearPosition보다 앞에 있는 경우와 뒤에 있는 경우를 나누어 생각하여야 하기 때문에 코드 구현이 길어졌다.
* **LinkedNode를 이용한 순환 원형 큐 장점**
* 연결 리스트를 이용하여 구현하였기 때문에 크기에 대한 제한이 없다.
* 배열에 비해 큐의 모든 원소를 출력하는게 쉽다. elementAt 함수를 이용하여 size-1 ~ 0까지 한번에 역순으로 출력이 가능하다.
* 순환 원형이기 때문에 rearNode의 next가 frontNode라고 정의할 수 있기때문에 따로 frontNode를 정의하지 않아도 되었다.
* **LinkedNode를 이용한 순환 원형 큐 단점**
* 배열에 비해 LinkedNode 클래스에 대한 정의가 필요하였다.

1. **실행 결과 분석**
2. **입력과 출력**
3. **ArrayCircularQueue**

****

****

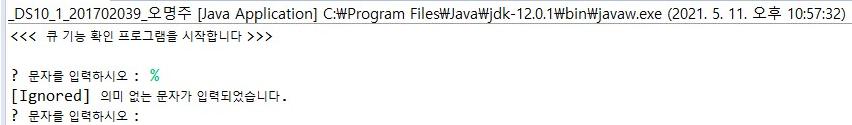
1. **LinkedCircularyQueue**

****

**[ 예외 처리 ]**



* ArrayCircularQueue에 MAX\_CAPACITY 이상 넣은 경우



* 의미 없는 문자를 입력한 경우

1. **결과 분석 (자신의 논리적 평가, 기타 느낀 점)**

* 지난 과제들에서 구현한 LinkedNode, Iterator, Interface 를 이용한 자료구조, 등을 이용하여 쉽게 구현할 수 있었다.
* Stack보다는 Queue가 더 효율적인 자료구조라는 생각이 들었다. 마트 계산대, 프로세스 관리 등 일상생활에서 쓰이는 경우가 더 많다.
* 생각해 볼 점

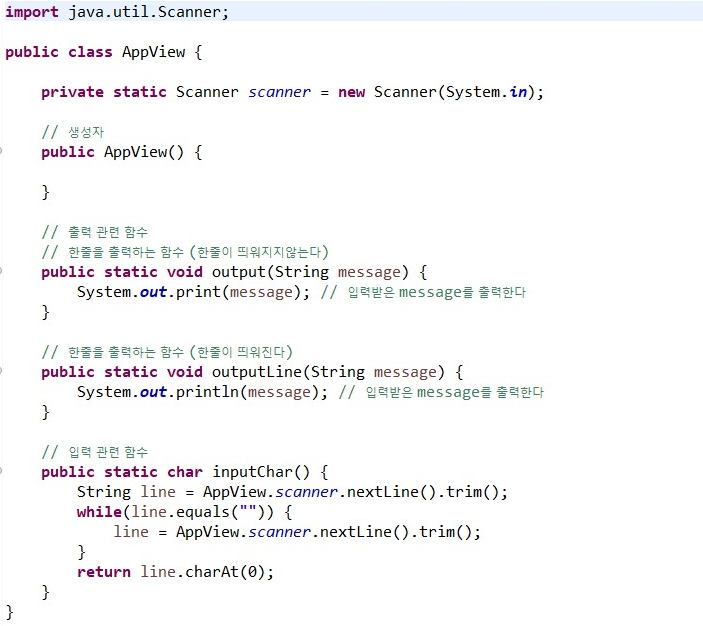
1. 구현에 있어서 리스트와 배열의 장단점

|  |  |  |
| --- | --- | --- |
|  | 장점 | 단점 |
| Array | - 인덱스를 통한 각각의 원소에 대한 접근이 편리하다. | - 최대 크기에 대한 제한이 존재한다.  - 순서대로 출력할 때 rear - front 순으로 있다면 출력이 길어진다.  - 인덱스 예외와 널 포인터 예외가 생길 수 있다.  - 배열이 삭제되면 공간이 남아 메모리가 낭비된다. |
| LinkedList | - 크기에 대한 제한이 없다.  - 메모리 재사용이 편리하다.  - rear의 next가 front 로 정의해둬서 front에 대한 접근이 쉬웠다. | - 각각의 원소에 대한 접근이 배열에 비해 반복적이다. (head부터 걸어 나감) |

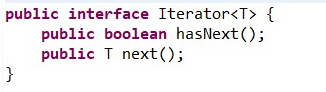
1. Queue를 Interface로 선언하는 이유는?

* 다른 방식으로 Queue를 구현하는데 용이하다. Array와 LinkedList를 통한 Queue를 구현할 때 Queue 인터페이스는 그대로 두고 자료구조만 바꾸어 implement를 이용하여 구현하였다.

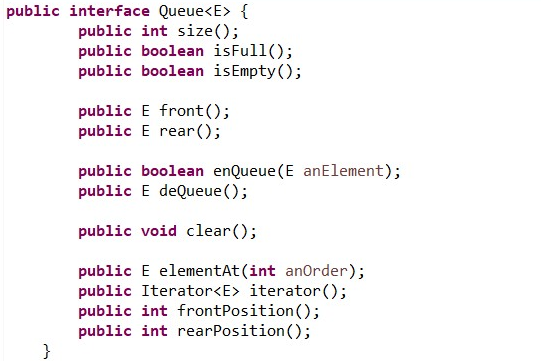
1. **소스코드**

****

**[AppView]**

****

**[Iterator]**

****

**[Queue]**

* **Array**

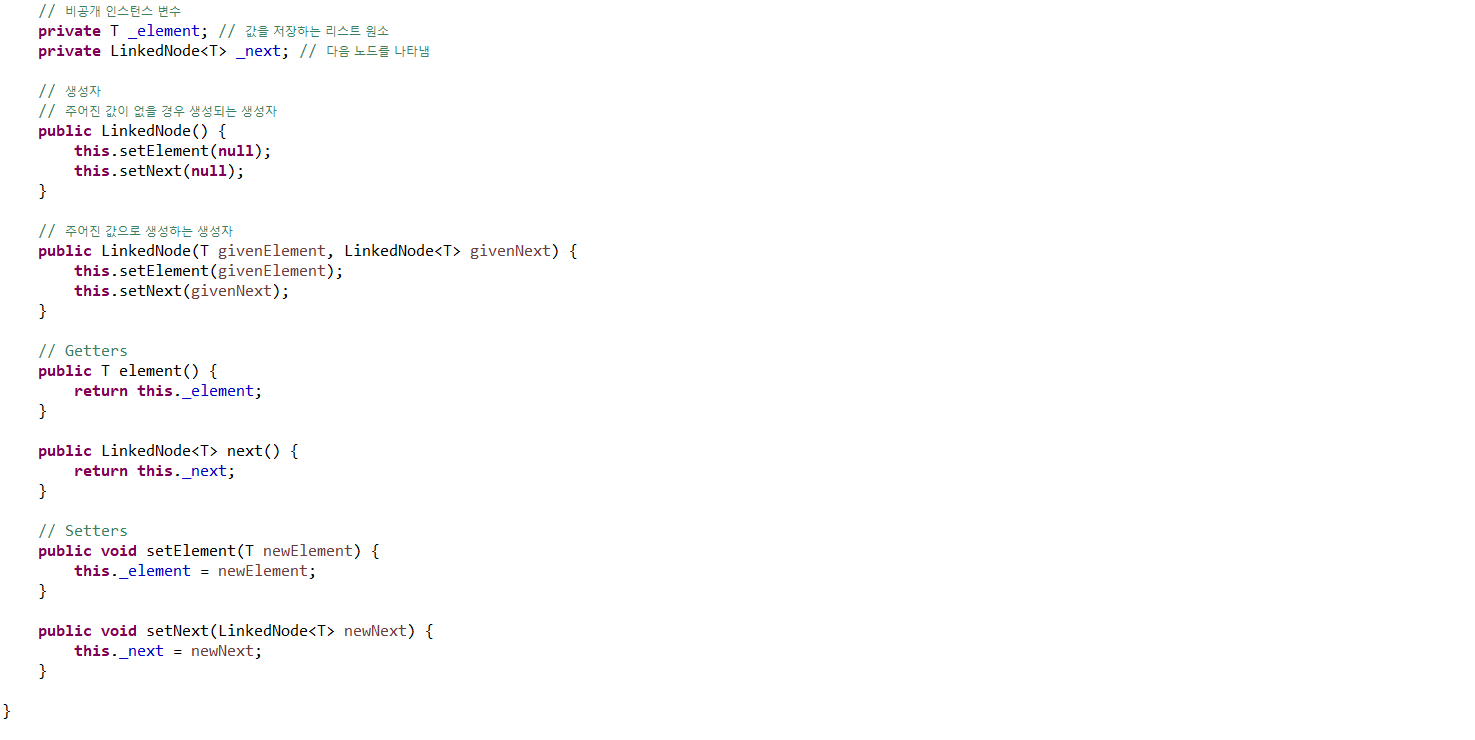




* **LinkedList**







**[LinkedNode]**