자료구조: 2022년 1학기 [실습]

제 10 주:

큐 (Queue)

강 지 훈
jhkang@cnu.ac.kr
충남대학교 컴퓨터융합학부

[큐 실습 1]

배열로 구현된 큐
- 배열을 고리 형태로 -

실습 목표

실습 목표

- 이론적 관점
  - 큐의 개념을 이해한다
- 구현적 관점
  - 배열을 이용한 환형 큐의 구현
  - 큐를 Interface 로 정의

과제에서 해결할 문제

문제

- 키보드에서 문자를 반복 입력 받는다.
  - 한 번에 한 개의 문자를 입력 받는다.
- 입력의 종료 조건
  - '!' 키를 치면 더 이상 입력을 받지 않는다.
- 매번 한 개의 문자를 입력 받을 때 마다, 문자에 따라 정해진 일을 한다.

문자에 따라 해야 할 일 [1]

- 영문자 ('A' ~ 'Z', 'a' ~ 'z'): 큐에 삽입한다.
  - 다음과 같은 메시지를 내보낸다. (입력된 영문자가 'x' 라면)
    - "[EnQ] 삽입된 원소는 'x' 입니다."
  - 만일 큐가 full 이면 다음과 같은 메시지를 내보낸다.
    - (ERROR) 큐가 꽉 차서 삽입이 불가능합니다.

- '!': 큐의 원소를 모두 삭제하고, 프로그램을 종료한다.
  - 큐의 원소를 삭제하기 전에 먼저 다음 메시지를 내보낸다:
    - "<큐를 비우고, 사용을 종료합니다>"
  - 원소를 차례로 삭제하여 큐를 비운다.
    삭제할 때마다 다음과 같은 메시지를 내보낸다. (삭제된 원소가 'x' 라면)
    - "[DeQs] 삭제된 원소는 'x' 입니다."

  - 또한 큐를 사용한 통계 자료를 출력한다.
    - 입력된 문자의 개수
    - 정상 처리된 문자의 개수
    - 무시된 문자의 개수
    - 삽입된 문자의 개수

문자에 따라 해야 할 일 [2]

- 숫자문자('0' ~ '9'): 해당 수 만큼 큐에서 삭제한다.
  - 매번 삭제될 때마다 다음과 같은 메시지를 내보낸다. (삭제된 원소가 'x' 라면)
    - "[DeQs] 삭제된 원소는 'x' 입니다.
  - 삭제를 하려는데 큐가 empty 이면, 다음과 같은 메시지를 내보내고 삭제를 멈춘다.
    - "[DeQs.Empty] 큐에 더 이상 삭제할 원소가 없습니다."

- '-': 큐의 front 원소를 삭제한다.
  - 다음과 같은 메시지를 내보낸다. (front 원소가 'r' 라면)
    - "[DeQ] 삭제된 원소는 'r' 입니다.
  - 삭제를 하려는데 큐가 empty 이면, 다음과 같은 메시지를 내보내고 삭제를 멈춘다.
    - "[DeQ.Empty] 큐에 원소가 없습니다."

문자에 따라 해야 할 일 [3]

- '#': 큐의 길이를 다음과 같이 출력한다.
  (현재 큐에 5 개의 원소가 있다면)
  - "[Size] 큐에는 현재 5 개의 원소가 있습니다."

- '/': 큐의 내용을 front 부터 rear 까지 다음과 같이 차례로 출력한다.
  - "[Queue] <Front> A b c d E f <Rear>"
  - 반복자 (iterator()) 를 사용하여 구현한다.

- '\': 큐의 내용을 rear 부터 front 까지 다음과 같이 차례로 출력한다.
  - "[Queue] <Rear> f E d c b A <Front>"
  - elementAt() 을 사용하여 구현한다.

문자에 따라 해야 할 일 [4]

- '>': 큐의 rear 원소의 값을 출력한다. 큐는 변하지 않는다. (큐의 rear 원소가 'r' 이라면)
  - "[Front] 맨 뒤 원소는 'r' 입니다."
  - 큐가 empty 이면 다음과 같은 메시지를 내보낸다.
    - "[Front.Empty] 큐에 원소가 없습니다."
- '<': 큐의 front 원소의 값을 출력한다. 큐는 변하지 않는다. (큐의 front 원소가 'f' 라면)
  - "[Front] 맨 앞 원소는 'f' 입니다."
  - 큐가 empty 이면 다음과 같은 메시지를 내보낸다.
    - "[Front.Empty] 큐에 원소가 없습니다."
- 그 밖의 문자들: 다음과 같이 출력하고 무시한다.
  - "[Ignore] 의미 없는 문자가 입력되었습니다."

출력의 예 [1]

```
<<< 큐 기능 확인 프로그램을 시작합니다 >>>

? 문자를 입력하시오: -
[DeQ.Empty] 큐에 삭제할 원소가 없습니다.
? 문자를 입력하시오: A
[EnQ] 삽입된 원소는 'A' 입니다.
? 문자를 입력하시오: x
[EnQ] 삽입된 원소는 'x' 입니다.
? 문자를 입력하시오: h
[EnQ] 삽입된 원소는 'h' 입니다.
? 문자를 입력하시오: d
[EnQ] 삽입된 원소는 'd' 입니다.
? 문자를 입력하시오: #
[Size] 큐에는 현재 4 개의 원소가 있습니다.
? 문자를 입력하시오: /
[Queue] <Front> A x h d <Rear>
? 문자를 입력하시오: \
[Queue] <Rear> d h x A <Front>
? 문자를 입력하시오: >
[Rear] 큐의 맨 뒤 원소는 'd' 입니다.
? 문자를 입력하시오: <
[Front] 큐의 맨 앞 원소는 'A' 입니다.
? 문자를 입력하시오: -
[DeQ] 삭제된 원소는 'A' 입니다.
? 문자를 입력하시오: 2
[DeQs] 삭제된 원소는 'x' 입니다.
[DeQs] 삭제된 원소는 'h' 입니다.
? 문자를 입력하시오: /
[Queue] <Front> d <Rear>
? 문자를 입력하시오: 2
[DeQs] 삭제된 원소는 'd' 입니다.
[DeQs.Empty] 큐에 더이상 삭제할 원소가 없습니다.
? 문자를 입력하시오: >
[Rear.Empty] 큐가 비어서 맨 뒤 원소가 존재하지 않습니다.
? 문자를 입력하시오: <
[Front.Empty] 큐가 비어서 맨 앞 원소가 존재하지 않습니다.
```

```
? 문자를 입력하시오: B
[EnQ] 삽입된 원소는 'B' 입니다.
? 문자를 입력하시오: w
[EnQ] 삽입된 원소는 'w' 입니다.
? 문자를 입력하시오: E
[EnQ] 삽입된 원소는 'E' 입니다.
? 문자를 입력하시오: T
[EnQ] 삽입된 원소는 'T' 입니다.
? 문자를 입력하시오: m
[EnQ] 삽입된 원소는 'm' 입니다.
? 문자를 입력하시오: 6
[DeQs] 삭제된 원소는 'B' 입니다.
[DeQs] 삭제된 원소는 'w' 입니다.
[DeQs] 삭제된 원소는 'E' 입니다.
[DeQs] 삭제된 원소는 'T' 입니다.
[DeQs] 삭제된 원소는 'm' 입니다.
[DeQs.Empty] 큐에 더이상 삭제할 원소가 없습니다.
? 문자를 입력하시오: /
[Queue] <Front> <Rear>
? 문자를 입력하시오: !

<큐를 비우고 사용을 종료합니다>
[Queue] <Front> <Rear>
[DeQs] 삭제할 원소의 개수가 0 개 입니다.

<큐 사용 통계>
- 입력된 문자는 23 개 입니다.
- 정상 처리된 문자는 23 개 입니다.
- 무시된 문자는 0 개 입니다.
- 삽입된 문자는 9 개 입니다.

<<< 큐 기능 확인 프로그램을 종료합니다 >>>
```

출력의 예 [2]

```
<<< 큐 기능 확인 프로그램을 시작합니다 >>>

? 문자를 입력하시오: A
[EnQ] 삽입된 원소는 'A' 입니다.
? 문자를 입력하시오: x
[EnQ] 삽입된 원소는 'x' 입니다.
? 문자를 입력하시오: D
[EnQ] 삽입된 원소는 'D' 입니다.
? 문자를 입력하시오: H
[EnQ] 삽입된 원소는 'H' 입니다.
? 문자를 입력하시오: w
[EnQ] 삽입된 원소는 'w' 입니다.
? 문자를 입력하시오: K
(오류) 큐가 꽉 차서, 더이상 넣을 수 없습니다.
? 문자를 입력하시오: $
[Ignore] 의미 없는 문자가 입력되었습니다.
? 문자를 입력하시오: /
[Queue] <Front> A x D H w <Rear>
? 문자를 입력하시오:

5
[DeQs] 삭제된 원소는 'A' 입니다.
[DeQs] 삭제된 원소는 'x' 입니다.
[DeQs] 삭제된 원소는 'D' 입니다.
[DeQs] 삭제된 원소는 'H' 입니다.
[DeQs] 삭제된 원소는 'w' 입니다.
```

```
? 문자를 입력하시오: >
[Rear.Empty] 큐가 비어서 맨 뒤 원소가 존재하지 않습니다.
? 문자를 입력하시오: <
[Front.Empty] 큐가 비어서 맨 앞 원소가 존재하지 않습니다.
? 문자를 입력하시오:     -
[DeQ.Empty] 큐에 삭제할 원소가 없습니다.
? 문자를 입력하시오: !

<큐를 비우고 사용을 종료합니다>
[Queue] <Front> <Rear>
[DeQs] 삭제할 원소의 개수가 0 개 입니다.

<큐 사용 통계>
- 입력된 문자는 12 개 입니다.
- 정상 처리된 문자는 11 개 입니다.
- 무시된 문자는 1 개 입니다.
- 삽입된 문자는 6 개 입니다.

<<< 큐 기능 확인 프로그램을 종료합니다 >>>
```

문자열의 앞뒤 공백 문자를 제거하고 입력 받는다.

의미 없는 문자의 처리

"Enter" 키를 쳤을 때에도, 다음 줄에서 입력을 받게 한다.

구현할 내용

과제에서 필요한 Class / Interface

이 과제에서 필요한 Class 는?

- AppController
- AppView
- Model
  - Interface "Queue<T>"
  - Class “CircularArrayQueue<T>” (implements "Queue<T>")

main()

main()을 위한 class

```
public class _DS10_S1_학번_이름 {
    public static void main (String[] args)
    {
        AppController appController = new AppController() ;
            // AppController 가 실질적인 main class 이다.
        appController.run() ;
            // 여기 main()에서는 앱 실행이 시작되도록 해주는 일이 전부이다.
    }
}
```

Class "AppController"

AppController: 상수, 인스턴스 변수, Getter/Setter

```
public class AppController {
        // 상수
        private static final int QUEUE_CAPACITY = 10;

        // 비공개 변수들
        private Queue<Character>   _queue ;

        private int  _inputChars ;        // 입력된 문자의 개수
        private int  _addedChars ;        // 삽입된 문자의 개수
        private int  _ignoredChars ;      // 무시된 문자의 개수

        // Getter/Setter
        private Queue<Character> queue() {...}
        private void setQueue (Queue<Character> newQueue) {...}
        private int inputChars() {...}
        private void setInputChars(int newInputChars) {...}
        private int addedChars() {...}
        private void setAddedChars(int newAddedChars) {...}
        private int ignoredChars() {...}
        private void setIgnoredChars(int newIgnoredChars) {...}

        // 생성자
        ……

        // 비공개 함수의 구현
        ……

        // 공개 함수의 구현
        ……
} // End of class "AppController"
```

AppController: 생성자

```
public class AppController {
        // 비공개 변수들
        ……

        // 생성자
        public  AppController() {
              this.setQueue (
                    new CircularArrayQueue<Character>(AppController.QUEUE_CAPACITY));
              this.setInputChars (0) ;
              this.setAddedChars (0) ;
              this.setIgnoredChars (0) ;
        }

        // 비공개 함수의 구현
        ……

        // 공개 함수의 구현
        ……
} // End of class "AppController"
```

AppController: 비공개 함수

```
public class AppController {
    ...
    // 비공개 함수의 구현
    // 횟수 계산
    private void  countInputChar () {...}
    private void  countIgnoredChar () {...}
    private void  countAddedChar () {...}

    // 큐 수행 관련
    private void  addToQueue (char aCharForAdd) {...}
    private void  removeOne () {...}
    private void  removeN (int numberOfCharsToBeRemoved) {...}
    private void  quitQueueProcessing() {...}

    // 출력 관련
    private void showAllFromFront () {...}
    private void showAllFromRear () {...}
    private void showFrontElement() {...}
    private void showRearElement() {...}
    private void showQueueSize() {...}
    private void showStatistics() {...}

    // 입력 관련
    private char inputChar() {...}
```

AppController: showAllFromFront()

```
private void showAllFromFront() {
    // 큐의 모든 원소를 Front 부터 Rear 까지 출력한다.
    // Iterator 를 사용한다.
    AppView.output ("[Queue] <Front> ") ;
    Iterator<Character> queueIterator = this.queue().iterator();
    while ( queueIterator.hasNext() ) {
        Character element = queueIterator.next();
        AppView.output (element.toString() + " " ) ;
    }
    AppView.outputLine ("<Rear>") ;
}
```

AppController: showAllFromRear()

```
private void showAllFromRear() {
    // 큐의 모든 원소를 Rear 부터 Front 까지 출력한다.
    // elementAt() 을 사용한다.
    AppView.output ("[Queue] <Read> ") ;
    for ( int order = this.queue().size() -1 ; order >= 0 ; order-- ) {
        AppView.output (this.queue().elementAt(order).toString() + " " ) ;
    }
    AppView.outputLine ("<Front>") ;
}
```

AppController: showFrontElement(), showRearElement(), showQueueSize()

- private void showFrontElement()
  - Queue 객체의 front() 을 이용하여 Front 원소를 출력
  - 큐가 비어 있으면, front 원소 대신 비어 있다는 메시지 출력
- private void showRearElement()
  - Queue 객체의 rear() 를 이용하여 Rear 원소를 출력
  - 큐가 비어 있으면, Rear 원소 대신 비어 있다는 메시지 출력
- private void showQueueSize()
  - Queue 객체의 size() 를 이용하여 원소의 개수를 출력

AppController: countInputChars(),…

- private void countInputChar()
  - this.setInputChars (this.inputChars()+1) ;
- private void countIgnoredChar()
  - this.setIgnoredChars (this.ignoredChars()+1) ;
- private void countAddedChar()
  - this.setAddedChars (this.addedChars()+1) ;

AppController: 비공개 함수 [4]

- private void addToQueue (Character anElement)
  - 큐가 가득 찬 경우
    - "[EnQ.Empty] 큐가 꽉 차서 더 이상 넣을 수가 없습니다." 를 출력
  - 정상적으로 enQueue가 되었을 경우
    - 삽입된 원소를 출력한다.
  - 유의:
    - 삽입하는 원소의 자료형은,
      기본 자료형인 "char" 가 아니라 class "Character" 이다.
    - this.countAdded() 를 호출하여 삽입된 원소의 숫자를 증가시켜야 하는데,
      이 행위는 addToQueue() 를 call 하는 곳에서 실행한다.

AppController: removeOne()

- private void removeOne ()
  - 큐가 empty 인 경우:
    - "[DeQ.Empty] 큐에 삭제할 원소가 없습니다."
  - 큐가 empty 가 아닌 경우에는 "deQueue()" 를 실행하여 삭제된 원소를 얻어, 다음과 같이 출력한다: (삭제된 원소가 'X' 라면)
    - "[DeQ] 삭제된 원소는 'X' 입니다."
  - 삭제 오류가 발생하면 다음과 같이 출력:
    - "(오류) 큐에서 삭제하는 동안에 오류가 발생하였습니다."

```
private void removeOne() {
    if (this.queue().isEmpty()) {
        AppView.outputLine("[DeQ.Empty] 큐에 삭제할 원소가 없습니다.");
    }
    else {
        Character removedChar = this.queue().deQueue() ;
        if (removedChar == null) {
            AppView.outputLine("(오류) 큐에서 삭제하는 동안에 오류가 발생하였습니다.");
        }
        else {
            AppView.outputLine("[DeQ] 삭제된 원소는 '" + removedChar + "' 입니다.");
        }
    }
}
```

AppController: 비공개 함수 [4]

- private void removeN (int numberOfCharsToBeRemoved)
  - 삭제할 개수가 0 개이면 다음과 같이 출력:
    - "[DeQs] 삭제할 원소의 개수가 0 개입니다."
  - "deQueue()" 를 삭제하는 횟수만큼 실행하여, 매번 다음과 같이 출력한다: (삭제된 원소가 'X' 라면)
    - "[DeQs] 삭제된 원소는 'X' 입니다."
  - 삭제 오류가 발생하면 다음과 같이 출력:
    - "(오류) 큐에서 삭제하는 동안에 오류가 발생하였습니다."
  - 삭제를 반복하는 동안에, 큐가 비게 되어 더 이상 삭제가 불가능하게 되면 다음과 같은 메시지를 출력한다
    - "[DeQs.Empty] 큐에 더 이상 삭제할 원소가 없습니다."

AppController: 비공개 함수 [5]

- private void quitQueueProcessing()
  - 큐의 모습을 출력한다:
    - showAllFromFront() 사용
  - 큐에 들어있는 모든 원소를 deQueue 한다:
    - removeN(this.queue().size()) 사용

- private void showStatistics()
  - 입력된 문자의 개수를 출력한다.
  - 정상 처리된 문자의 개수를 출력한다.
  - 무시된 문자의 개수를 출력한다.
  - 삽입된 문자의 개수를 출력한다.

```
private void showStatistics() {
    AppView.outputLine("");
    AppView.outputLine("<큐 사용 통계>");
    AppView.outputLine("- 입력된 문자는 " + this.inputChars() + " 개 입니다.");
    AppView.outputLine
        ("- 정상 처리된 문자는 " + (this.inputChars()-this.ignoredChars()) + " 개 입니다.");
    AppView.outputLine("- 무시된 문자는 " + this.ignoredChars() + " 개 입니다.");
    AppView.outputLine("- 삽입된 문자는 " + this.addedChars() + " 개 입니다.");
}
```

AppController: 유일한 공개 함수 "run()" [1]

```
// 공개 함수의 구현
public void  run() {
      AppView.outputLine ("<<<큐 기능 확인 프로그램을 시작합니다 >>>) ;
      AppView.outputLine() ;

      char  inputElement = this.inputChar() ;
      while ( inputElement != '!' ) {
            this.countInputChar () ;
            if ( (Character.isAlphabet(input) ) {
                  this.addToQueue ( Character.valueOf (inputElement) ) ;
                  this.countAddedChar() ;
            }
            else if ( Character.isDigit(inputElement) ) {
                  this.removeN ( Character.getNumericValue (inputElement)) ) ;
            }
            else if ( inputElement == '-' ) {
                  this.removeOne() ;
            }
            else if ( inputElement == '#' ) {
                  this.showQueueSize() ;
            }
```

AppController: 유일한 공개 함수 "run()" [2]

```
// 공개 함수의 구현
public void  run() {
      ..... // 앞 쪽의 내용 생략
      while ( inputElement ! = '!' ) {
            ..... // 앞 쪽의 내용 생략
            else if ( inputElement == '/' ) {
                  this.showAllFromFront () ;
            }
            else if  (inputElement == '\\' ) { // Back Slash (\)는 반드시 이렇게 '\\'
                  this.showAllFromRear () ;
            }
            else if ( inputElement == '<' ) {
                  this.showFrontElement () ;
            }
            else if  (inputElement == '>' ) {
                  this.showRearElement () ;
            }
            else {
                  AppView.outputLine ("[Ignore] 의미 없는 문자가 입력되었습니다.") ;
                  this.countIgnoredChar () ;
            }
            inputElement = this.inputChar () ;
      }
      this.quitQueueProcessing () ;

      this.showStatistics () ;
      AppView.outputLine("") ;
      AppView.outputLine ("<<<큐 기능 확인 프로그램을 종료합니다 >>>) ;
}
```

Class "AppView"

AppView

- 이전 실습의 것을 그대로 사용한다.

Interface "Queue<E>"

interface Queue⟨E⟩

```
public interface Queue<E>
{
    // 반드시 필요한 method
    public int size() ;
    public boolean isFull();
    public boolean isEmpty();

    public E front() ;
    public E rear() ;

    public boolean enQueue(E anElement);
    public E deQueue();

    public void clear();

    // 편의성을 위한 method
    public E elementAt(int anOrder);
    public Iterator<E> iterator();

}
```

Interface “Iterator<E>”

Interface Iterator⟨E⟩

```
public interface Iterator<E>
{
        public boolean   hasNext() ;
        public E   next() ;
}
```

Class “CircularArrayQueue”

implements "Queue<E>"

CircularArrayQueue: 상수, 인스턴스 변수

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

    // Constants
    private static final int  DEFAULT_CAPACITY = 100 ;

    // Instance Variables
    private int       _maxLength ; // capacity+1
    private int       _frontPosition ;
    private int       _rearPosition ;
    private E[]       _elements ;

    // Getter/Setter
    private int       maxLength() {...}
    private void      setMaxLength (int newMaxlength) {...}
    public int        capacity() {
          return  (this.maxLength() - 1) ;
    }
    private int       frontPosition() {...}
    private void      setFrontPosition (int newFrontPosition) {...}
    private int       rearPosition() {...}
    private void      setRearPosition() (int newRearPosition) {...}
    private void      elements () {...}
    private E[]       setElements (E[] newElements) {...}

    @Override
    public int  size() {
          if ( this.rearPosition() >= this.frontPosition() ) {
                return  (this.rearPosition() – this.frontPosition()) ;
          }
          else {
                return  (this.rearPosition() + this.maxLength() – this.frontPosition()) ;
          }
    }
```

Circular Array Queue 로 구현할 경우, 저장 가능한 원소의 최대 개수는, 최대 길이보다 하나 작다.

강의 자료를 참고할 것

CircularArrayQueue: Getter/Setter

- public int capacity()
  - (this.maxLength() +1) 값을 반환
  - 인스턴스 변수가 존재하지 않는 getter로, 계산을 하여 얻는다.
- public boolean isEmpty ()
  - this.frontPosition() 과 this.rearPosition() 이 같으면 true 를, 아니면 false 를 반환
- public boolean isFull ()
  - 다음 삽입될 위치가 this.frontPosition() 과 같으면 true 를, 아니면 false 를 반환.
  - 다음 삽입될 위치는 어떻게 확인 할까?
- public int size ()
  - 크기를 유지하는 인스턴스 변수가 존재하지 않는 getter 로, 계산을 하여 얻는다.

CircularArrayQueue: 생성자

- public CircularArrayQueue (int givenCapacity)
  - this._maxLength 를 (givenCapacity+1) 로 초기화
  - this._frontPosition 과 this._rearPosition 는 0 으로 초기화
  - this._elements 배열의 원소의 자료형이 generic type 인 E 로 선언되어 있다. 그러므로, 원소의 자료형으로 Object 를 사용하여 배열을 생성해야 한다.
    - this.setElements ((E[ ]) new Object[this.maxLength()]) ;

- public CircularArrayQueue ()
  - 기본 생성자.
  - 위에 주어진 생성자와 동일한 초기화 기능을 한다.
    - 단, capacity 의 값은 DEFAULT_CAPACITY 로 초기화한다.

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

CircularArrayQueue: front(), rear()

- public E front ()
  - 비어 있는 경우 null 을 반환
  - 비어 있지 않은 경우 큐의 맨 앞에 있는 원소를 반환

- public E rear ()
  - 비어 있는 경우 null 을 반환
  - 비어 있지 않은 경우 큐의 맨 뒤에 있는 원소를 반환

CircularArrayQueue: enQueue()

- public boolean enQueue (E anElement)
  - 가득 차 있을 경우 false 를 반환
  - 가득 차 있지 않을 경우
    - this.rearPosition() 을 새로운 위치로 변경한다.
    - this._elements[] 의 this.rearPosition() 위치에 anElement 를 삽입.

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

유의: 배열의 마지막 위치의 다음 위치는 0

CircularArrayQueue: deQueue()

- public E deQueue()
  - 비어있는 경우 null 을 반환
  - 만약 비어 있지 않으면
    - this.frontPosition() 의 값을 (+1 하여) 다음 위치로 조정
      - 유의: this.frontPosition() 은 항상 front 원소의 직전 위치를 가리킨다.
    - this._elements[this.frontPosition()] 의 원소, 즉 front 원소를 빼내어 보관해 놓는다.
    - front 원소를 배열로부터 제거한다:
      - 배열의 해당 칸 this._elements[this._frontPosition()] 을 null 값으로 설정한다.
  - 보관해 놓은 front 원소를 결과값으로 돌려준다.

CircularArrayQueue: clear(), elementAt()

- public void clear ()
  - this._frontPosition 과 this._rearPosition 을 0 으로 초기화한다.
  - this._elements[] 의 모든 칸을 null 로 초기화한다.

- public E elementAt (int anOrder)
  - 리스트에서 anOrder 위치의 원소를 반환한다.
  - 유의: 개념적인 순서 (order) 와 큐의 배열에서의 원소의 position 은 다르다.
    - front 원소의 위치가 리스트의 맨 앞 위치, 즉 0 번째 위치이다.
      - 유의: 큐의 front / rear 중 어느 쪽을, 리스트의 맨 앞 위치로 볼 것인가는 정해야 한다.
    - ((this.frontPosition() + 1) % this.maxLength()) 위치가 anOrder 가 0 인 위치이다.
    - 그러므로 순서 anOrder 의 원소는, 배열에서
      ((this.frontPosition() + 1 + anOrder) % this.maxLength())
      의 위치에 있게 된다.

CircularArrayQueue: 반복자

```
public Iterator<E> iterator() {
    return (new CircularArrayQueueIterator()) ;
}

private class  CircularArrayQueueIterator implements Iterator<E>
{
    private int _nextOrder;

    private int nextOrder() {...
    private void setNextOrder (int newNextOrder) {...
    private CircularArrayQueueIterator() {
        this.setNextOrder(0);
    }

    @Override
    public boolean hasNext() {
        return (this.nextOrder() < CircularArrayQueue.this.size());
    }
    @Override
    public E next() {
        E nextElement = null ;
        if (this.hasNext()) {
            nextElement = CircularArrayQueue.this.elementAt(this.nextOrder());
            this.setNextOrder(this.nextOrder()+1);
        }
        return nextElement;
    }
}
```

요약

확인하자

- 환형 배열 큐
  - 큐의 개념
  - 직선형 배열 그대로가 아닌, 환형 배열 큐로 구현하는 이유
  - 구현 방법

- 환형 배열 큐는, 원소를 삽입 (enQueue) 하거나 삭제 (deQueue) 할 때에 front 와 rear 의 위치를 계산하여야 한다.
  - 보고서에 본인이 구현한 위치 계산 법을 설명하시오.

생각해 볼 점

- 큐를 포함하여 일반적으로 리스트를 배열로 구현할 경우, 사용자의 관점인 원소의 순서 (order) 와 실제 구현된 배열에서의 위치 (position) 개념은 동일한 개념이 아니다. 이 둘, 즉 사용자의 order 와 구현적 관점인 position 을 구분하는 이유는?

⇒ 생각해 볼 점에 대해, 자신의 의견을 보고서에 작성하시오.

[큐 실습 2]

환형 연결 체인으로 구현된 큐

실습 목표

실습 목표

- 이론적 관점
  - 큐의 개념을 이해한다
- 구현적 관점
  - 연결체인을 이용하여, 환형 큐로 구현하는 방법을 이해한다

과제에서 해결할 문제

문제

- [실습 1] 의 문제와 동일

이 과제에서 필요한 Class 는?

- AppController
- AppView
- Model
  - Interface "Queue<T>"
  - Class “CircularlyLinkedQueue<T> implements Queue<T>”
  - Class “LinkedNode<T>”
  - Interface Iterator<T>

구현할 내용

main()

main()을 위한 class

```
public class _DS10_S2_학번_이름 {
    public static void main (String[] args)
    {
        AppController  appController = new AppController() ;
            // AppController 가 실질적인 main class 이다.
        appController.run() ;
            // 여기 main()에서는 앱 실행이 시작되도록 해주는 일이 전부이다.
    }
}
```

Class “AppController”

Class "AppController" 의 구현 [1]

```
public class AppController
{
        // 비공개 변수들
        private Queue<Character>  _queue ; // 지난 주와 동일한 선언
        private int  _inputChars ;   // 입력된 문자의 개수
        private int  _ignoredChars ;     // 무시된 문자의 개수
        private int  _addedChars ; // 삽입된 문자의 개수

        // Getter/Setter
        ......

        // 생성자
        public AppController() {
              this.setQueue ( new CircularlyLinkedQueue<Character>() );
              this.setInputChars (0) ;
              this.setAddedChars (0) ;
              this.setIgnoredChars (0) ;
        }

        // 비공개함수의 구현
        ......

        // 공개함수의 구현
        ......
} // End of class "AppController"
```

AppController: 달라진 점

- [큐 실습 1]의 AppController 의 생성자에서,
  큐 객체 생성을 class "CircularArrayQueue<E>" 대신에
  class "CircularlyLinkedQueue<E>" 를 사용하는 것을 제외하고는,
  모든 코드는 동일하다.

- 그러므로 [큐 실습 1]의 class "AppController"를 복사하여 수정하여 사용한다.

Class “AppView”

Class "AppView" 의 공개함수

- [큐 실습 1] 의 것을 복사하여 그대로 사용한다.

Class “LinkedNode<E>”

LinkedNode⟨E⟩: 인스턴스 변수, Getter/Setter, 생성자

```
public class  LinkedNode<E>
{
        // Private Instance Variable
        private E                    _element ;
        private LinkedNode<E>      _next ;

        // Getter/Setter
        public E  element () {...}
        public void  setElement (E newElement) {...}
        public LinkedNode<E>  next () {...}
        public void  setNext (LinkedNode<E> newNext) {...}

        // 생성자
        public  LinkedNode () {...}
        public  LinkedNode (E givenElement, LinkedNode<E> givenNext) {...}
```

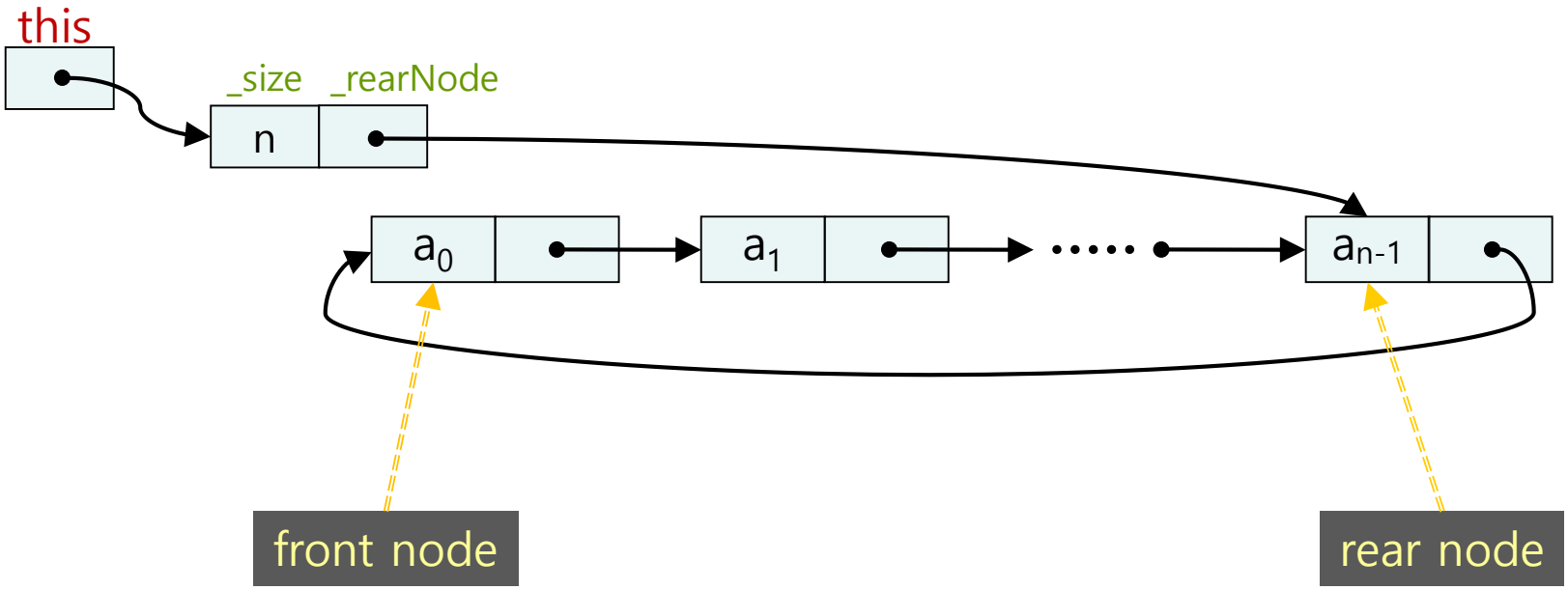
Interface “Queue”

Interface Queue⟨E⟩:

- [큐 실습 1]의 것을 복사하여 그대로 사용한다

Class "CircularlyLinkedQueue<E>"

객체의 모습



CircularlyLinkedQueue: 인스턴스 변수, Getter/Setter

```
public class CircularlyLinkedQueue<E>
        implements Queue<E>
{
        // 인스턴스 변수
        private int                 _size ;
        private LinkedNode<E>  _rearNode ;

        // Getter/Setter
        public int  size () {...}
        private void  setSize (int newSize) {...}
        private LinkedNode<E> rearNode() {...}
        private void  setRearNode (LinkedNode<E> newRearNode) {...}
```

CircularlyLinkedQueue: 생성자, getter

- 생성자
  - public CircularlyLinkedQueue ()
    - this._size 와 this._rearNode 의 초기화

- public int capacity ()
  - 큐가 수용할 수 있는 원소의 최대 개수를 돌려 주어야 한다. CircularlyLinkedQueue 에서는 이 값을 어떻게 해야 할까?
  - 연결체인이므로, capacity 는 무한대로 가정할 수 있다.
  - 무한대 값은 어떻게 표현?
    - 인스턴스 변수 _size 의 type 이 int 이다. 그러므로, capacity 의 값은 int 의 최대 값 (Integer.MAX_VALUE) 을 넘을 수는 없다.
    - 그렇다면, 적절한 방안은?

- public int size ()
  - 현재 큐가 가지고 있는 원소의 개수, 즉 this._size 의 값을 돌려준다.

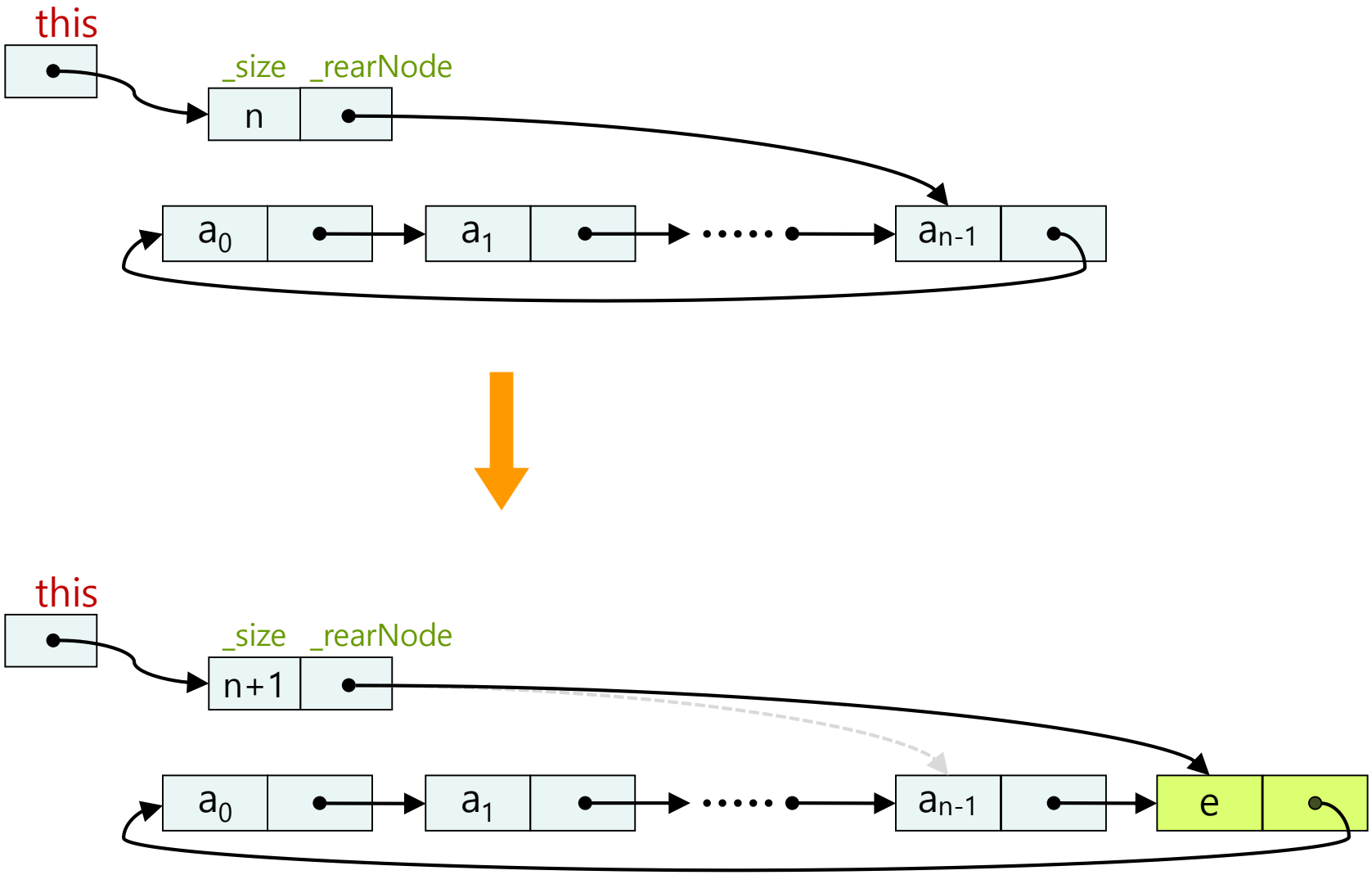
CircularlyLinkedQueue: 공개함수

- 상태 알아보기
  - public boolean isEmpty ()
    - this._rear 의 값이 null 이면 true 를 반환
  - public boolean isFull ()
    - 연결 체인이 가득 차게 되는 경우는 없다고 가정한다.
  - public E front ()
    - 큐가 비어 있으면 null 을 돌려준다.
    - 비어 있지 않으면, _rearNode 의 다음 node 가 front node 이다. 이 front node 의 element 를 돌려준다.

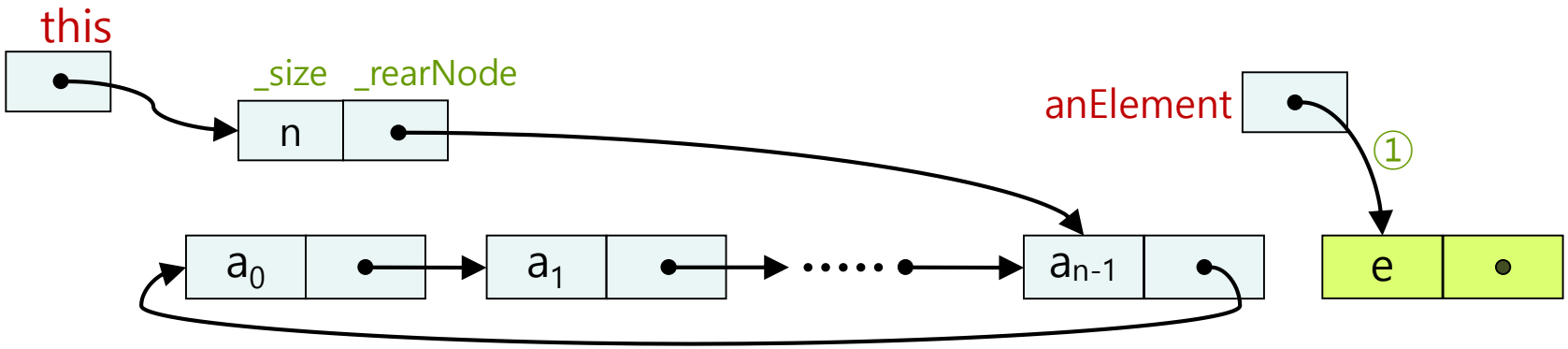
CircularlyLinkedQueue: 삽입 함수 enQueue() 의 구현

- public boolean enQueue (E anElement)
  - 가득 차 있을 경우:
    - false 를 반환하여야 하나, 연결체인으로 구현되므로 이 경우는 발생하지 않는다.
  - 가득 차 있지 않을 경우
    - empty 라면?
    - empty 가 아니라면?

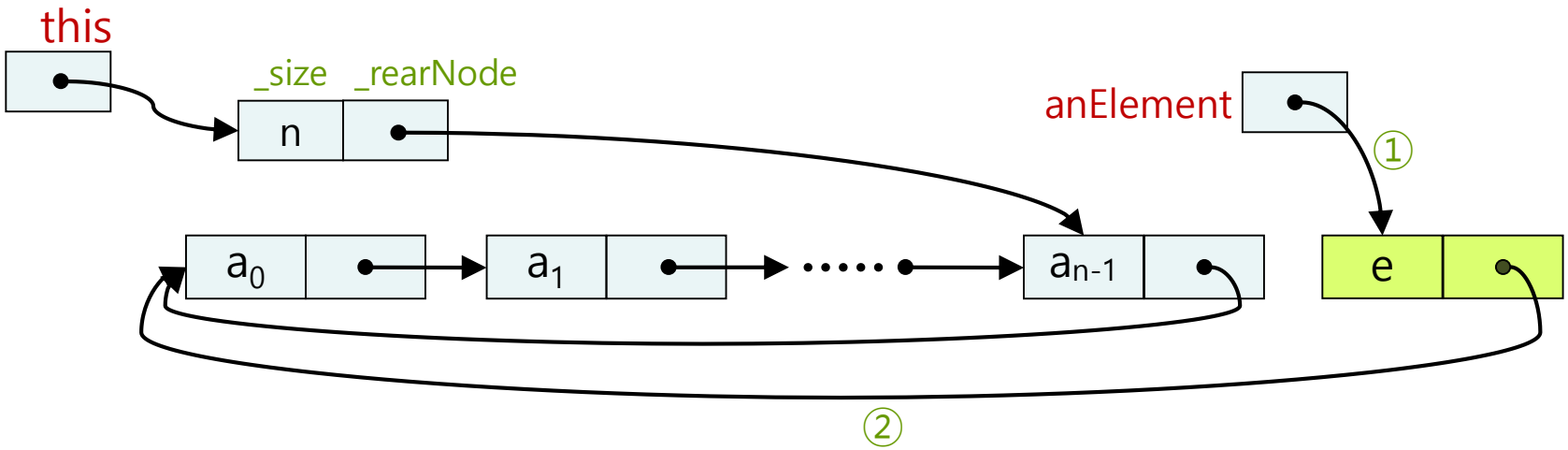
Rear에 삽입: 전과 후의 모습



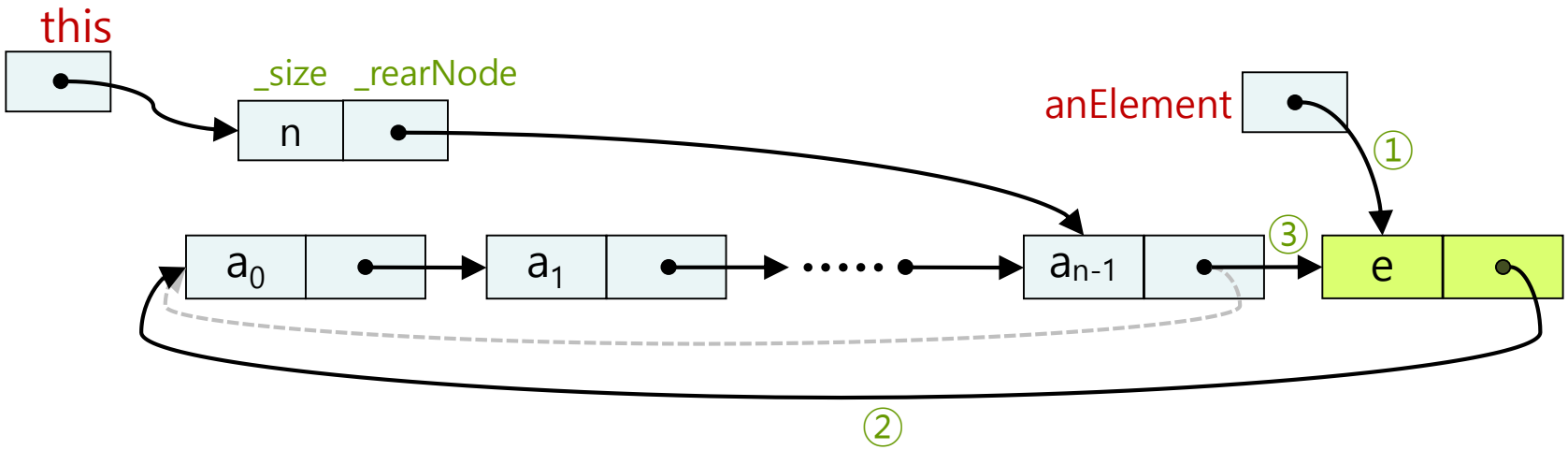
Rear에 삽입: 단계 1



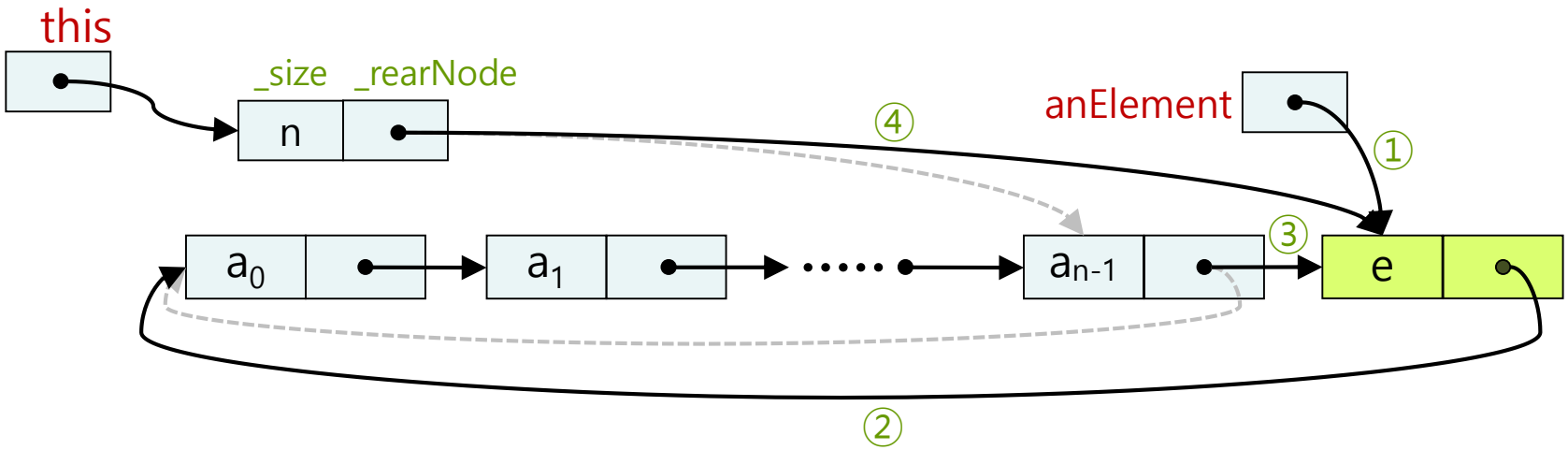
Rear에 삽입: 단계 2



Rear에 삽입: 단계 3

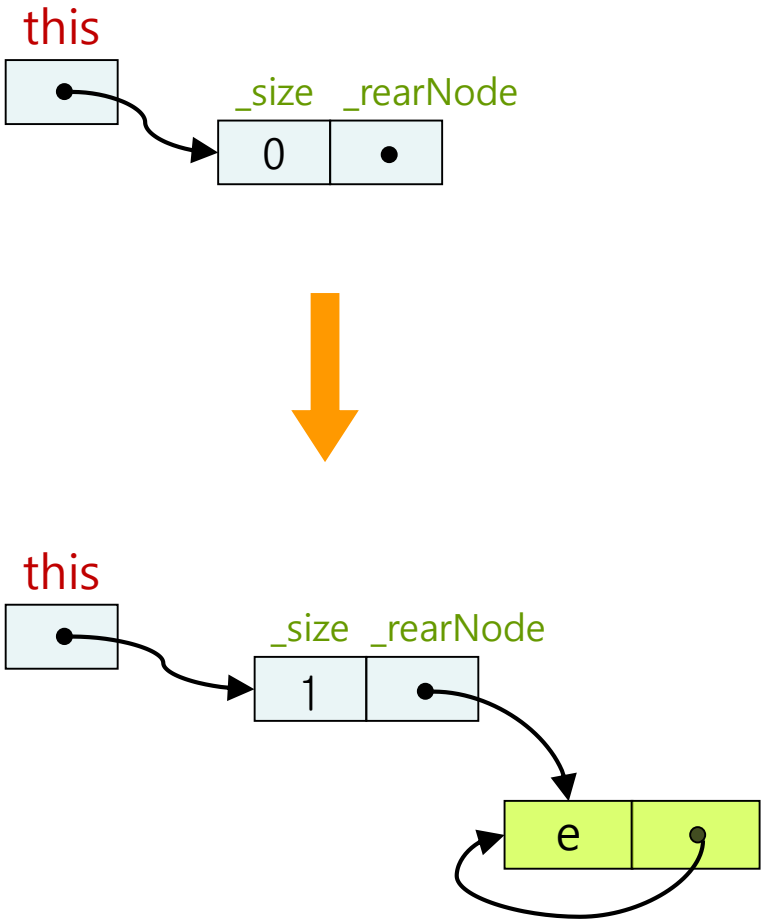


Rear에 삽입: 단계 4



삽입: 리스트가 비어 있다면 ?

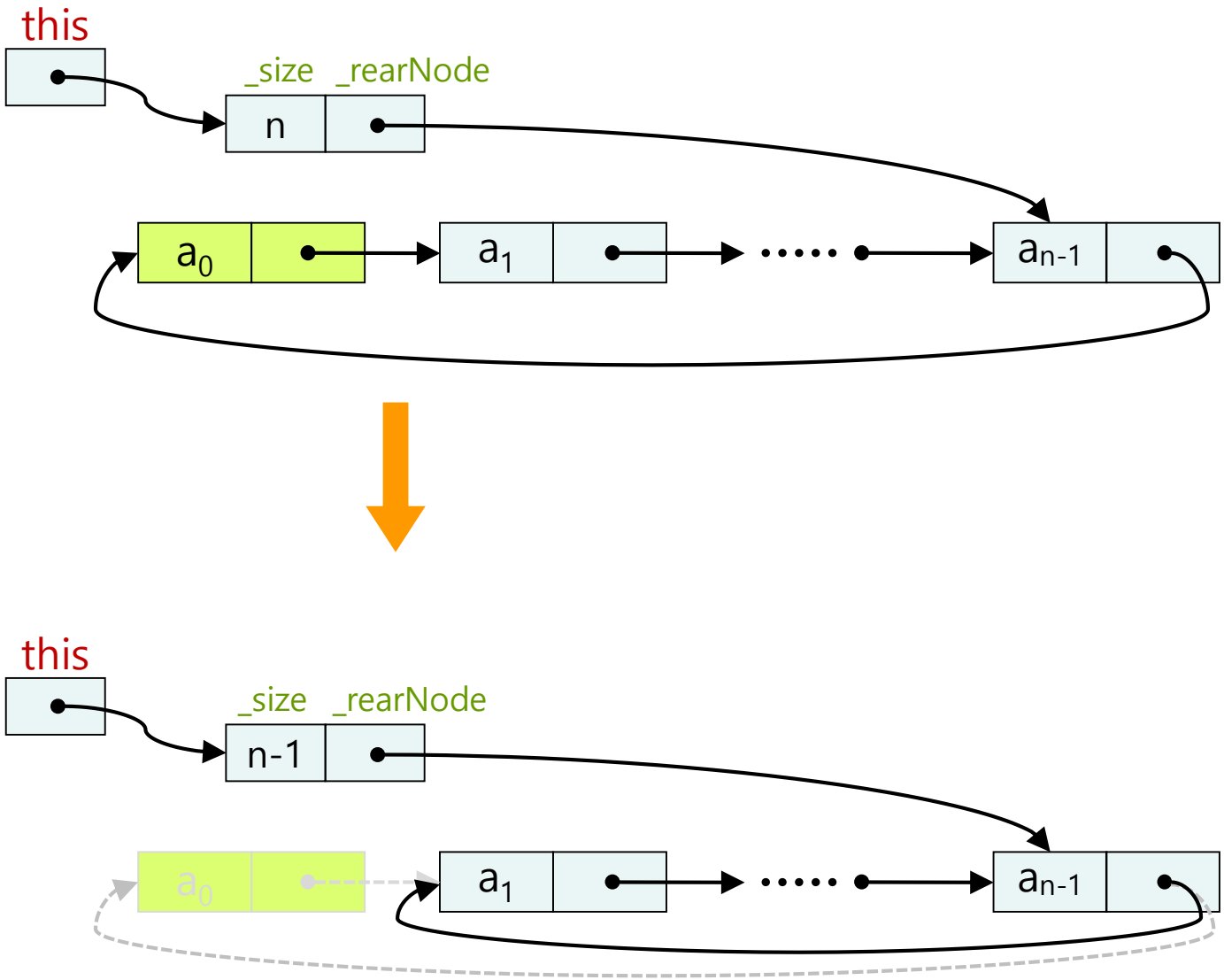
- 비어 있지 않은 경우와 동일한 코드로 가능할까?



CircularlyLinkedQueue: deQueue()

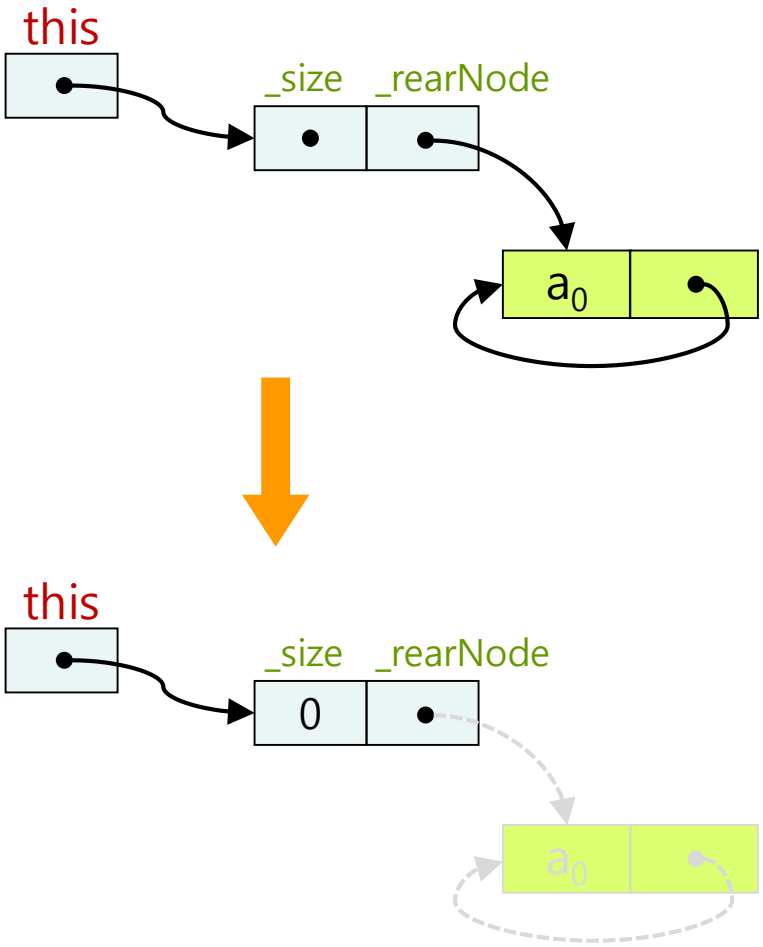
- public E deQueue ()
  - 비어 있으면: null 을 돌려준다.
  - 만약 비어 있지 않으면: front element 를 삭제하여 돌려준다.

Front 에서 삭제



삭제: 리스트에 노드가 한 개만 있다면?

- 노드가 두 개 이상인 경우에 삭제하는 코드를 그대로 사용할 수 있을까?



CircularlyLinkedQueue:

- public void clear ()
  - 처음 생성했을 때의 empty queue 상태와 동일하게 만든다.

- public E elementAt (int anOrder)
  - 순서 (order):
    - 주어진 anOrder 의 값은 0 부터 (this._size – 1) 까지 사이의 값이다.
    - front 노드의 순서 (order) 값을 0 으로 한다.
  - front 노드부터 세어서 anOrder 위치의 노드의 원소를 돌려준다.
  - front 노드는 rear 노드의 다음 노드이다. 그러므로 front 노드는 다음과 같이 얻을 수 있다.
    - LinkedNode<E> frontNode = this.rearNode().next() ;

CircularlyLinkedQueue: 반복자 [1]

- Inner class 로 구현한다.
- 환형 연결 체인 큐의 끝을 인식하는 방법: count 변수를 사용한다.
- count: 아직 반복에 사용되지 않고 남아 있는 원소의 개수
  - 초기화 : 현재 큐에 있는 원소의 개수
  - 다음 원소를 얻는 "next()" 가 실행될 때 마다, count 의 값을 하나 줄인다.
  - count 가 0 이 되면, 모든 원소가 반복에 의해 소진 되었으므로 "hasNext()" 는 false 를 얻게 된다.

```
public class CircularlyLinkedQueue<E> {
    ......
    @Override
    public Iterator<E> iterator() {
        return new CircularlyLinkedQueueIterator<E>() ;
    }

    private class CircularlyLinkedQueueIterator implements Iterator<E> {
        private LinkedNode<E> _nextNode ;
        private int _count ;
```

반복자를 얻는 공개함수

반복자를 구현하는 Inner class

아직 반복에 사용되지 않고 남아 있는 원소의 개수

CircularlyLinkedQueue: 반복자 [2]

```
public class CircularlyLinkedQueue<E> {
    ......
    private class CircularlyLinkedQueueIterator implements Iterator<E> {
        ......
        private LinkedNode<E>  nextNode() {...}
        private void setNextNode (LinkedNode<E> newNextNode) {...}
        private int count() {...}
        private void setCount (int newCount) {...}

        private CircularlyLinkedQueueIterator() {
            this.setNextNode ( CircularlyLinkedQueue.this.rearNode() ) ;
            this.setCount ( CircularlyLinkedQueue.this.size() ) ;
        }

        @Override
        public boolean hasNext() {
            return ( this.count() > 0 ) ;
        }

        @Override
        public E  next() {
            if ( this.hasNext() ) {
                this.setNextNode (this.nextNode().next() ) ;
                E  nextElement = this.nextNode().element() ;
                this.setCount (this.count() -1) ;
                return  nextElement ;
            }
            else {
                return  null ;
            }
        }
    }
```

큐의 rearNode 로 초기화 한다.

큐의 원소의 개수로 초기화 한다.

다음 원소가 있는지 알려준다

다음 원소를 얻는다

요 약

확인하자

- 환형 연결 체인 Circularly Linked Queue)으로 큐를 구현하는 방법

생각해 볼 점

- 환형 배열 (circular array) 로 구현한 큐와 연결체인 (linked chain) 으로 구현한 큐의 차이점과 장단점은?

- Queue 를 interface 로 선언하면 좋은 점은?
  - 큐를 다른 방식으로 구현할 경우를 생각해 보자.

⇒ 생각해 볼 점에 대해, 자신의 의견을 보고서에 작성하시오.

[실습 끝]