자료구조 실습 보고서
[제 10 주] 연결 리스트 큐 - 대기 큐 처리

제출일 : 2015 - 05 - 07
201402395 이승희

1. 프로그램 설명서
   1) 주요 알고리즘 / 자료구조 / 기타
      : 지난 주와 동일한 프로그램을 연결 리스트로 구현하였다.
   2) 함수 설명서
      (1) AppController
         - public void showAll() // this._linkedQueue 의 size 만큼 elementAt 를 호출하여 출력한다.

      (2) Node
         - public Node() // 생성자
         - public Node(T anElement) // 생성자
         - public Node(T anElement, Node aNode) // 생성자, 각 인자에 맞는 생성자에 따라 this_element 와 this._next 를 초기화
         - public T element() // this._element 를 반환
         - public Node next() // this._next 를 반환
         - public void setElement(T anElement) // this._element 에 anElement 를 저장
         - pubic void setNext(Node aNode) // this._node 에 aNode 를 저장

      (3) LinkedQueue
         - public LinkedQueue() // this._rear 와 this,_size 를 초기화
         - public int size() // this._size 를 반환
         - public Boolean isEmpty() // Queue 가 비어있는지 확인
         - public Boolean isFull() // Queue 가 가득 차 있는지 확인
         - public T frontElement() // this._front 의 값을 반환
         - public Boolean enQueue(T anElement) // Queue 에 anElement 를 추가
         - public T deQueue() // Queue 의 _front 의 원소를 제거하고 반환
         - public void clear() // Queue 를 초기화
         - public T elemnetAt(int givenPosition) // givenPosition 에 있는 원소를 반환

         // 이하 지난주와 동일

   3) 종합 설명서
      : 연결 리스트의 특성상 배열이 아닌 Node 의 개념을 적용하여 maxSize 가 없고 , Queue 가 가득 차는 경우는 없다.
2. 프로그램 장단점 분석
   : 지난 주에 했던 circularQueue 와 다르게 간단하고 Queue 의 크기가 정해져있지 않아서 무한히 원소를 저장할 수 있다.

3. 실행 결과 분석

1) 입력과 출력

```
> 프로그램을 시작합니다.
[ 큐 입력을 시작합니다. ]
- 문자를 입력하시오 : s
[EnQueue] 삽입된 원소는 s 입니다.
- 문자를 입력하시오 : t
[EnQueue] 삽입된 원소는 t 입니다.
- 문자를 입력하시오 : a
[EnQueue] 삽입된 원소는 a 입니다.
- 문자를 입력하시오 : r
[EnQueue] 삽입된 원소는 r 입니다.
- 문자를 입력하시오 : /
[Queue]  <Front>  s  t  a  r  <Rear>
- 문자를 입력하시오 : 3
[RemoveN] 3개의 원소를 삭제하려고 합니다.
[DeQueue] 삭제된 원소는 s입니다.
[DeQueue] 삭제된 원소는 t입니다.
[DeQueue] 삭제된 원소는 a입니다.
- 문자를 입력하시오 : q
[EnQueue] 삽입된 원소는 q 입니다.
- 문자를 입력하시오 : u
[EnQueue] 삽입된 원소는 u 입니다.
- 문자를 입력하시오 : e
[EnQueue] 삽입된 원소는 e 입니다.
- 문자를 입력하시오 : ^
[Front] 맨 앞 원소는 r입니다.
- 문자를 입력하시오 : /
[Queue]  <Front>  r  q  u  e  <Rear>
- 문자를 입력하시오 : 5
[RemoveN] 5개의 원소를 삭제하려고 합니다.
[DeQueue] 삭제된 원소는 r입니다.
[DeQueue] 삭제된 원소는 q입니다.
[DeQueue] 삭제된 원소는 u입니다.
[DeQueue] 삭제된 원소는 e입니다.
[Empty] 큐에 원소가 없습니다.
- 문자를 입력하시오 : ]
Error : 의미 없는 문자가 입력되었습니다.
- 문자를 입력하시오 : B
[EnQueue] 삽입된 원소는 B 입니다.
- 문자를 입력하시오 : a
[EnQueue] 삽입된 원소는 a 입니다.
- 문자를 입력하시오 : g
[EnQueue] 삽입된 원소는 g 입니다.
- 문자를 입력하시오 : /
[Queue]  <Front>  B  a  g  <Rear>
- 문자를 입력하시오 : !
[ 큐 입력을 종료합니다. ]
[DeQueue] 삭제된 원소는 B입니다.
[DeQueue] 삭제된 원소는 a입니다.
[DeQueue] 삭제된 원소는 g입니다.
... . .입력된 문자는 모두 17 개 입니다.
... . .정상 처리된 문자는 16 개 입니다.
... . .무시된 문자는 1 개 입니다.
... . .삽입된 문자는 10 개 입니다.

> 프로그램을 종료합니다.
```

2) 결과 분석

: 결과를 보아 CircularQueue 와 다르게 삽입은 rear 에서 , 삭제는 front 에서 일어난다. 만약 Queue 가 비어있다면 삽입된 원소는 front 에 저장된다.

4. 소스코드

(1) AppController

```
public class AppController<T> {
    private AppView _appView;
    private LinkedQueue _linkedQueue;
    private int _inputChars;
    private int _ignoredChars;
    private int _addedChars;

   public void showAll() {
       this.showMessage(MessageID.Show_QueueStart);
       for (int i = 0; i < this._linkedQueue.size(); i++)
           this._appView.outputElement((Character) this._linkedQueue
                   .elementAt(i));
       this.showMessage(MessageID.Show_QueueEnd);
   }
```

(2) Node

```
public class Node<T> {
    private T _element;
    private Node<T> _next;

    public Node() {
        this._element = null;
        this._next = null;
    }

    public Node(T anElement) {
        this._element = anElement;
    }

    public Node(T anElement, Node<T> aNode) {
        this._element = anElement;
        this._next = aNode;
    }

    public T element() {
        return this._element;
    }

    public Node<T> next() {
        return this._next;
    }

    public void setElement(T anElement) {
        this._element = anElement;
    }

    public void setNext(Node<T> aNode) {
        this._next = aNode;
    }
}
```

(3) LinkedQueue

```
public class LinkedQueue<T> {
    private int _size;
    private Node<T> _rear;
    private Node<T> _front;

    public LinkedQueue() {
        this._rear = null;
        this._size = 0;
        this._front = null;
    }

    public int size() {
        return this._size;
    }

    public boolean isEmpty() {
        return this._front == this._rear && this._front == null;

    }

    public int maxsize() {
        return this._size;
    }

    public T frontElement() {
        if (isEmpty())
            return null;
        else
            return this._front.element();
    }

    public boolean enQueue(T anElement) {
        if (isFull())
            return false;
        else {
            Node<T> newNode = new Node<T>(anElement, null);
            if (isEmpty()) {
                this._front = newNode;
```

```
            } else {
                this._rear.setNext(newNode);
            }
            this._rear = newNode;
            this._size++;
            return true;
        }

    }

    public boolean isFull() {
        return false;
    }

    public T deQueue() {
        T frontElement = null;
        if (!isEmpty()) {
            frontElement = this._front.element();
            this._front = this._front.next();
            if (this._front == null)
                this._rear = null;
            this._size--;
        }
        return frontElement;
    }

    public void clear() {
        this._front = null;
        this._rear = null;
        this._size = 0;
    }

    public T elementAt(int givenPosition) {
        Node frontElement = this._front;

        if (givenPosition == 0)
            return this._front.element();
        else {
            for (int i = 0; i < givenPosition; i++) {
                if (frontElement == this._rear)
                    break;
                else
                    frontElement = frontElement.next();
            }
            return (T) frontElement.element();
        }
    }

}
```

//나머지는 지난주와 동일하다.

5. 문제 정리
    1) 연결 체인
        - 삽입과 삭제
          : 연결 체인에서 원소의 삽입은 rear 에서 일어나며, 삽입 후에는 추가된 원소가 rear 가 되고 기존의 rear 의 next 가 된다. 삭제는 front 에서 일어나며, 기존의 front 는 삭제되고 삭제 후 front 는 기존의 front.next 의 elemet 가 된다.
        - 리스트 구현에 있어서 배열과의 차이점과 장단점
          : 배열은 크기가 정해져있기 때문에 원소의 수가 제한적인 단점이 있지만 원소의 정보가 알고싶을 때 접근이 용이하다. 연결 체인은 원소의 수가 무한하고 삽입과 삭제가 용이하다는 장점이 있지만 Node 라는 클래스가 필요하며 각 Node 에 접근하기가 힘들다. LinkedQueue 클래스의 public T elementAt(int givenPosition)을 구현할 때, 배열의 인덱스값에 해당하는 원소를 반환하는 Array 와는 달리 front 부터 출력하는 대신에 각 기능을 수행할 때 마다 front 의 값이 바뀌기 때문에 수행 후 다시 front 를 초기화해주어야 하는 단점이 있었다.

2) 배열과 연결체인으로 구현한 큐의 차이점

: 배열로 구현한 큐와 달리 연결체인은 원소를 무한히 저장할 수 있다.

6. 디버깅 capture

: Queue 에 원소 a 를 input 한 결과를 디버깅한 모습이다.