자료구조 실습 보고서

[제 9-1주]

제출일: 19.05.22
학번/이름: 201603867/조성환

1. 프로그램 설명서

1) 주요 알고리즘/ 자료구조/ 기타

Queue라는 자료구조를 사용함. queue는 선입선출로 입력을 위한 마지막위치, 출력을 위한 처음 위치를 기억하는 식으로 자료구조를 이용함.

2) 함수 설명서

.main()
AppController의 객체를 생성함.
appcontroller.run()을 통해 프로그램을 실행함.

AppController
생성자
queue를 set하고, input, added, ignore char를 0으로 셋함.

run()
프로그램 시작의 출력문과 함께 시작함.
inputChar()로 char을 입력받고 이를 input에 넣음
입력받은 input이 '!'이 될 때까지 while을 통해 계속 입력받음.
input에 따라 다른 함수를 실행함.
문자면 addToQueue() 숫자면 removeN() -면 removeOne() # 이면 showQueueSize()
/ 면 showAllFromFront \면 show AllFromRear < 면 showFrontElement > 면 showRearElement. 만약 이외의 문자면 의미없는 문자가 입력되었음을 출력 후 countIgnoredChar()을 실행함
마지막으로 input에 inputChar()을 통해 입력을 받음
만약 !를 입력받아 while문을 나오면 quitQueueProcessing()을 실행
showStatistics()을 실행한 후 종료한다는 출력문과 함께 종료함.

count Input Ignore Added Char()
입력받은, 무시한, 추가한 문자를 세줌. queue가 정상적으로 동작했는지를 알아보기 위함.

inputChar()
AppView.intputChar()을 통해 입력 받음

addToQueue()
queue가 꽉 차있는지 확인 후 꽉 차있으면 못넣는다는 출력문. 공간이 있다면 queue().enQueue로 입력을 한 후 추가된 원소를 출력한 후 countAddedChar()을 해줌.

removeN()
입력받은 숫자가 0 이하인지 확인함. 먼저 removeN에 숫자를 넣는 방법은 Character.getNumericValue(input)을 통해 숫자인지 확인이 가능함.
만약 0보다 크면 queue가 비어있는지를 확인함. 비어있으면 더 이상 삭제할 원소가없다고 출

력함. 비어있지 않다면

CircularlyLinkedQueue

CircularlyLinkedQueue()
size를 0으로 set하고 , rearNode를 null로 셋함.

rearNode()
맨 끝의 노드를 설정함.

front()
frontElement를 null로 초기화함
frontElement에 rearNode.next.element로 맨 마지막의 다음을 가리킴으로써 맨 앞을 볼 수 잇음
해당 element를 리턴함.

rear()
rearNode.element()로 해당 값을 가져옴
이 값을 리턴함

enQueue()
LinkedNode 객체를 생서함.
만약 리스트가 비어있으면 본인을 rearNode로 next를 set함
비어있지 않으면 newRearNode의 next를 rearNode의 next로 둠
rearNode의 next를 newRearNode로 둠.
rearNode로 newRearNOde를 셋하고 size를 1키움

3) 종합 설명서

queue를 Linked로 구현함. Linked기 때문에 필요가 있음.
queue는 선입선출이기 때문에 입력과 출력을 위해 앞과 뒤만 알아둘 필요가 있음. 입력시에는 rearposition에 영향, 출력시에는 frontpositio에 영향을 줌. 또한 size와 empty, full을 front와 rear로 표시해야함.

2. 프로그램 장단점/ 특이점 분석

queue는 선입선출의 특징이있음. 이 때문에 계쏙 앞과 뒤의 위치를 알아야함. 하지만 Linked의 경우는 array와 달리 capacity의 한계가 정해져있지 않음. 이 때문에 rear 과 front의 계산이 복잡하지 않으며 추가, 삭제 등 모든 일을 할 때 딱히 구현에 어려운 점이 없었음. 단점은 아무래도 linked이기 때문에 구현에 어려운점이 약간 있었던 것 같음.

3. 실행 결과 분석

1) 입력과 출력

입력은 주로 영어문자와 숫자, 사전에 입력된 특수문자로 입력했음. 상황에맞게 추가 후 나열, 삭제 후 나열, 오류 처리가 되도록 삭제로 입력함.

```
<<< 큐 기능 확인 프로그램을 시작합니다 >>>

? 문자를 입력하시오: a
[EnQ] 추가된 원소는'a' 입니다.
? 문자를 입력하시오: s
[EnQ] 추가된 원소는's' 입니다.
? 문자를 입력하시오: #
[개수]2개 입니다.
? 문자를 입력하시오: /
[Queue] <Front>a s <Rear>
? 문자를 입력하시오: -
[DeQ] 삭제된 원소는'a' 입니다.
? 문자를 입력하시오: W
[Queue] <Rear>s <Front>
? 문자를 입력하시오: f
[EnQ] 추가된 원소는'f' 입니다.
? 문자를 입력하시오: e
[EnQ] 추가된 원소는'e' 입니다.
? 문자를 입력하시오: W
[Queue] <Rear>e f s <Front>
? 문자를 입력하시오: 5
[DeQs] 삭제된 원소는 's' 입니다.
[DeQs] 삭제된 원소는 'f' 입니다.
[DeQs] 삭제된 원소는 'e' 입니다.
[DeQs.Empty] 큐에 더 이상 삭제할 원소가 없습니다.
[DeQs.Empty] 큐에 더 이상 삭제할 원소가 없습니다.
? 문자를 입력하시오: a
[EnQ] 추가된 원소는'a' 입니다.
? 문자를 입력하시오: !
[Queue] <Front>a <Rear>
[DeQs] 삭제된 원소는 'a' 입니다.

<큐 사용 통계>
```

```
<큐 사용 통계>
- 입력된 문자는 11 개 입니다.
- 정상 처리된 문자는 11 개 입니다.
- 무시된 문자는 0 개 입니다.
- 삽입된 문자는 5 개 입니다.

<<< 큐 기능 확인 프로그램을 종료합니다 >>>
```

2) 결과 분석

입력에 맞게 기대한대로 결과가 나타남

4. 소스 코드

AppController

```java
package controller;

import model.CircularlyLinkedQueue;
import model.Iterator;
import model.Queue;
import view.AppView;

public class AppController {
    //private var
    private Queue<Character> _queue;
    private int _inputChars;
    private int _addedChars;
    private int _ignoredChars;

    //getter setter
    public Queue<Character> queue() { return _queue; }
    public void setQueue(Queue<Character> newQueue) { this._queue = newQueue; }

    public int inputChars() { return _inputChars; }
    public void setInputChars(int newInputChars) { this._inputChars = newInputChars; }

    public int addedChars() { return _addedChars; }
    public void setAddedChars(int newAddedChars) { this._addedChars = newAddedChars; }

    public int ignoredChars() { return _ignoredChars; }
    public void setIgnoredChars(int newIgnoredChars) { this._ignoredChars = newIgnoredChars; }

    //constructors
    public AppController(){
        this.setQueue(
                new CircularlyLinkedQueue<Character>());
        this.setInputChars(0);
        this.setAddedChars(0);
```

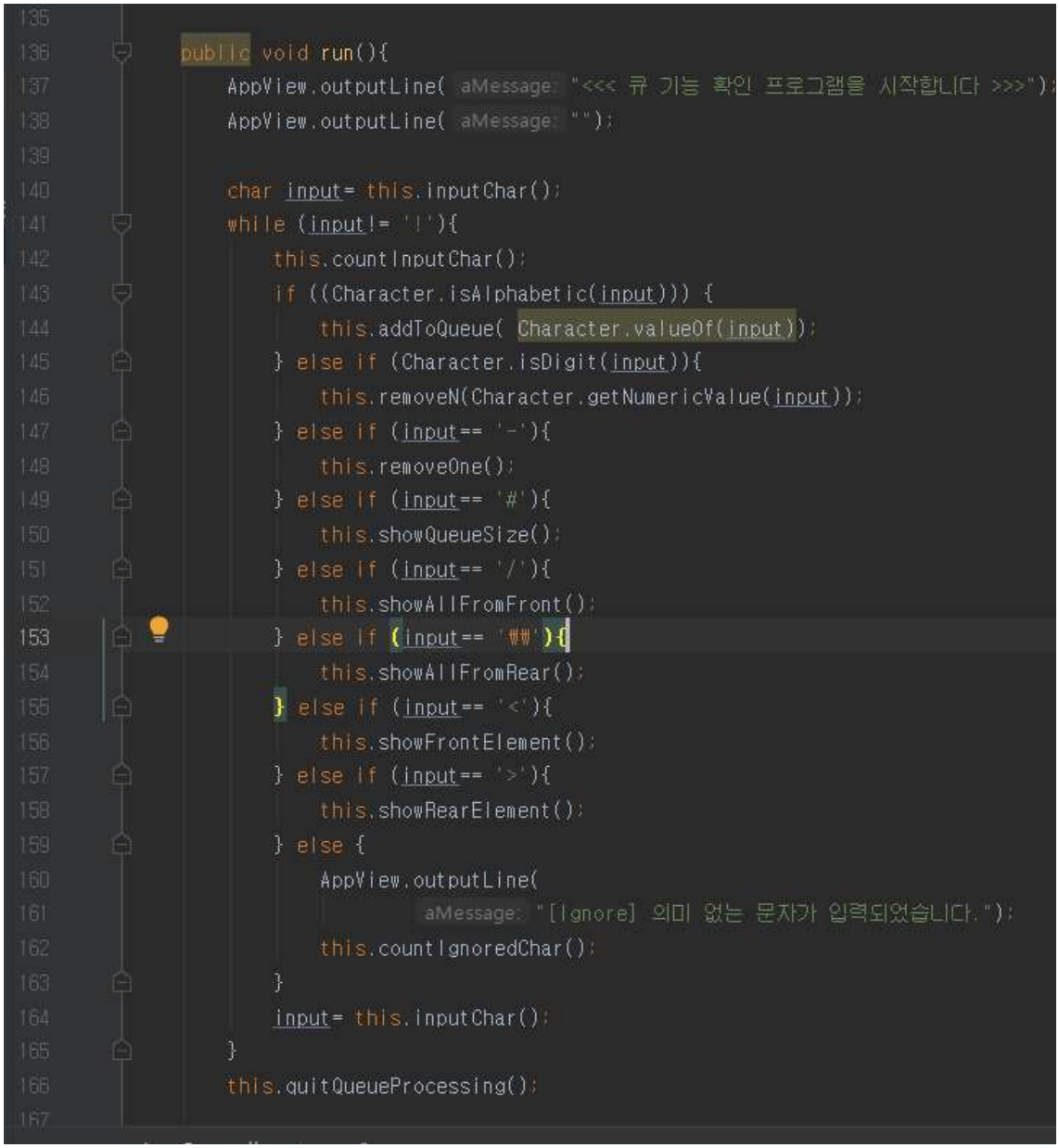
```
34          this.setIgnoredChars(0);
35      }
36
37      //private methods
38      //회수 계산
39      private void countInputChar(){ this.setInputChars(this.inputChars()+1);}
40      private void countIgnoredChar(){ this.setIgnoredChars(this.ignoredChars()+1);}
41      private void countAddedChar(){ this .setAddedChars(this.addedChars()+1);}
42
43      //큐 수행 관련
44      private void addToQueue(char aCharForAdd){
45          if (this.queue().isFull()){
46              AppView.outputLine( aMessage: "[EnQ.Full] 큐가 꽉 차서 더 이상 넣을 수가 없습니다.");
47          } else {
48              this.queue().enQueue((Character)aCharForAdd);
49              AppView.outputLine( aMessage: "[EnQ] 추가된 원소는'"+ aCharForAdd+ "' 입니다.");
50              this.countAddedChar();
51          }
52      }
53      private void removeOne(){
54          if (this.queue().isEmpty()){
55              AppView.outputLine( aMessage: "[DeQ.Empty] 큐에 삭제할 원소가 없습니다.");
56          } else {
57              Character removedChar= this.queue().deQueue();
58              if (removedChar== null)
59                  AppView.outputLine( aMessage: "(오류) 큐에서 삭제하는 동안에 오류가 발생하였습니다.");
60              else
61                  AppView.outputLine( aMessage: "[DeQ] 삭제된 원소는'"+removedChar+ "' 입니다.");
62          }
63      }
64      private void removeN(int numberOfCharsToBeRemoved){
65          if (numberOfCharsToBeRemoved<= 0) AppView.outputLine(
66                  aMessage: "[DeQs] 삭제할 원소의 개수가 0개 이하 입니다.");
```

```
        else {
            for(int i= 0; i< numberOfCharsToBeRemoved; i++){
                if (this.queue().isEmpty()) AppView.outputLine( aMessage: "" +
                        "[DeQs.Empty] 큐에 더 이상 삭제할 원소가 없습니다.");
                else {
                    Character removedeCharacter= null;
                    removedeCharacter= this.queue().deQueue();
                    if (removedeCharacter== null) AppView.outputLine
                            ( aMessage: " (오류) 큐에서 삭제하는 동안에 오류가 발생하였습니다.");
                    else AppView.outputLine( aMessage: "[DeQs] 삭제된 원소는 '"+removedeCharacter+ "' 입니다.");
                }
            }
        }
    }
    private void quitQueueProcessing(){
        this.showAllFromFront();
        this.removeN(this.queue().size());
    }

    //출력 관련
    private void showAllFromFront(){
        AppView.output( aMessage: "[Queue] <Front>");
        Iterator<Character> queueIterator= this.queue().iterator();
        while (queueIterator.hasNext()){
            Character element= queueIterator.next();
            AppView.output( aMessage: element.toString()+ " ");
        }
        AppView.outputLine( aMessage: "<Rear>");
    }
    private void showAllFromRear(){
        AppView.output( aMessage: "[Queue] <Rear>");
        for (int order= this.queue().size()-1; order>= 0; order--){
            AppView.output( aMessage: this.queue().elementAt(order).toString()+ " ");
```

```
        }
        AppView.outputLine( aMessage: "<Front>");
    }
    private void showFrontElement(){
        if (this.queue().isEmpty()){
            AppView.outputLine( aMessage: "비어있음");
        } else {
            AppView.outputLine( aMessage: "[Queue]"+this.queue().front());
        }
    }
    private void showRearElement(){
        if (this.queue().isEmpty()){
            AppView.outputLine( aMessage: "비어있음");
        } else {
            AppView.outputLine( aMessage: "[Queue]"+ this.queue().rear());
        }
    }
    private void showQueueSize() { AppView.outputLine( aMessage: "[개수]"+ this.queue().size()+ "개 입니다."); }
    private void showStatistics(){
        AppView.outputLine( aMessage: "");
        AppView.outputLine( aMessage: "<큐 사용 통계>");
        AppView.outputLine( aMessage: "- 입력된 문자는 "+ this.inputChars()+ " 개 입니다.");
        AppView.outputLine
                ( aMessage: "- 정상 처리된 문자는 "+ (this.inputChars()- this.ignoredChars())+ " 개 입니다.");
        AppView.outputLine( aMessage: "- 무시된 문자는 "+ this.ignoredChars()+ " 개 입니다.");
        AppView.outputLine( aMessage: "- 삽입된 문자는 "+ this.addedChars()+ " 개 입니다.");
    }

    //입력 관련
    private char inputChar(){
        AppView.output( aMessage: "? 문자를 입력하시오: ");
        return AppView.inputChar();
    }
```



```
135
136     public void run(){
137         AppView.outputLine( aMessage: "<<< 큐 기능 확인 프로그램을 시작합니다 >>>");
138         AppView.outputLine( aMessage: "");
139
140         char input= this.inputChar();
141         while (input!= '!'){
142             this.countInputChar();
143             if ((Character.isAlphabetic(input))) {
144                 this.addToQueue( Character.valueOf(input));
145             } else if (Character.isDigit(input)){
146                 this.removeN(Character.getNumericValue(input));
147             } else if (input== '-'){
148                 this.removeOne();
149             } else if (input== '#'){
150                 this.showQueueSize();
151             } else if (input== '/'){
152                 this.showAllFromFront();
153             } else if (input== '##'){
154                 this.showAllFromRear();
155             } else if (input== '<'){
156                 this.showFrontElement();
157             } else if (input== '>'){
158                 this.showRearElement();
159             } else {
160                 AppView.outputLine(
161                         aMessage: "[Ignore] 의미 없는 문자가 입력되었습니다.");
162                 this.countIgnoredChar();
163             }
164             input= this.inputChar();
165         }
166         this.quitQueueProcessing();
167
```

```
168         this.showStatistics();
169         AppView.outputLine( aMessage: "");
170         AppView.outputLine( aMessage: "<<< 큐 기능 확인 프로그램을 종료합니다 >>>");
171     }
172
173 }
174
```

CircularlyQueue

```java
package model;

public class CircularlyLinkedQueue<E> implements Queue<E> {

    //private var
    private int _size;
    private LinkedNode<E> _rearNode;

    //getter setter
    @Override
    public int size() { return this._size; }
    public void setSize(int newSize) { this._size = newSize; }

    public LinkedNode<E> rearNode() { return _rearNode; }
    public void setRearNode(LinkedNode<E> newRearNode) { this._rearNode = newRearNode; }

    //constructor
    public CircularlyLinkedQueue() {
        this.setSize(0);
        this.setRearNode(null);
    }

    //public methods
    @Override
    public boolean isFull() { return false; }
    @Override
    public boolean isEmpty() {
        if (this.rearNode() == null) {
            return true;
        } else {
            return false;
        }
    }
```

```java
    @Override
    public E front() {
        E frontElement = null;
        if(this.isEmpty()){
        } else {
            frontElement = this.rearNode().next().element();
        }
        return frontElement;
    }
    @Override
    public E rear() {
        return this.rearNode().element();
    }

    @Override
    public boolean enQueue(E anelement) {
        LinkedNode<E> newRearNode = new LinkedNode(anelement, givenNext: null);
        if(this.isEmpty()){
            newRearNode.setNext(newRearNode);
        } else{
            newRearNode.setNext(this.rearNode().next());
            this.rearNode().setNext(newRearNode);
        }
        this.setRearNode(newRearNode);
        this.setSize(this.size()+1);
        return true;
    }

    @Override
    public E deQueue() {
        E frontElement = null;
        if(this.isEmpty()){
        } else {
```

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

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

        public LinkedNode<E> nextNode() { return _nextNode; }
        public void setNextNode(LinkedNode<E> newNextNode) { this._nextNode = newNextNode; }
        public int count() {
            return _count;
        }
        public void setCount(int newCount) { this._count = newCount; }

        private CircularlyLinkedQueueIterator(){
            this.setNextNode((LinkedNode<E>) CircularlyLinkedQueue.this.rearNode());
            this.setCount(CircularlyLinkedQueue.this.size());
        }
        public boolean hasNext(){
            return (this.count() >0);
        }

        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;
            }
```