자료구조 실습 보고서

[제 9 주] 배열 큐

제출일 : 2015 - 04 - 29

201402395 이승희

1. 프로그램 설명서
   1) 주요 알고리즘 / 자료구조 / 기타
      : 배열 큐를 환형 큐로 구현한 프로그램이다. 원소의 추가, 삭제와 같은 기능을 수행하게 되며 프로그램의 전체 구조는 MVC 디자인 패턴을 따른다.
   2) 함수 설명서
      (1) AppController
         - public void showFrontElement() // this._arrayQueue 의 frontElement 메소드를 이용하여 top 원소를 받는다.
         - public void showQueueSize() // this._arrayQueue 의 size 메소드를 이용하여 원소의 개수를 얻는다.
         - public void showAll() // this._arrayQueue 의 elementAt 메소드를 이용하여 front 부터 rear 까지 출력한다.
         - public void initCharCounts() // this._inputChars, this._addedChars, this._ignoredChars 를 각각 초기화한다.
         - public void countAdded() // 문자가 삽입되면 this._addedChars 를 1 증가시킨다.
         - public void countIgnored() // 무시할 문자가 입력되면 this._ignoredChars 를 1 증가시킨다.
         - public void countInputChar() // 문자가 입력될 때 마다 this._inputChars 를 1 증가시킨다.
         - public void add(char c) // 원소를 this._arrayQueue 에 삽입한다. 큐가 가득차면 error message 를 출력한다.
         - public void removeOne() // this._arrayQueue 의 deQueue 를 실행하여 삭제된 원소를 얻어온다.
         - public void removeN(int numOfCharsToBeDeleted) // this._appView 의 outputRemoveN 메소드를 이용하여 삭제하는 개수를 출력하며, removeOne 을 원하는 숫자만큼 실행한다.
         - public void conclusion() // 큐 내부에 있는 모든 원소를 deQueue 한다.
         - public void showMessage(MessageID aMessageID) // 각 MessageID 에 맞는 message 를 출력한다.
         - public void run() // 프로그램의 실질적인 실행을 위한 함수이다.
      (2) AppView
         - public String inputString() // this._scanner 를 이용하여 문자열을 입력받는다.
         - Public char inputCharacter() // inputString 을 이용하여 실질적으로 문자를 입력받는다.

- Public void outputAdd(char anElement) // 삽입된 원소를 출력한다.
- Public void outputElement(char anElement) // Queue 에 존재하는 원소를 출력한다.
- Public void outputFrontelement(char anElement) // Queue 의 FrontElement 를 출력한다.
- Public void outputQueueSize(int aQueueSize) // Queue 의 크기를 출력한다.
- Public ovid outputRemove(char anElement) // 제거된 원소를 출력한다.
- Public void outputRemoveN(int aNumOfCharsToBeRemoved) // 원하는 숫자만큼의 remove 실행 횟수를 출력한다.
- Public void outputResult(int aNumOfInputChars, int aNumOfIgnoredChars, int aNumOfAddedChars) // 입력된 문자 개수, 정상 처리된 문자 개수, 무시된 문자 개수, 삽입된 문자 개수를 출력한다.

(3) CircularArrayQueue

- public CirvularArrayQueue() // 생성자
- public CircularArrayQueue(int initialCapacity) // this._maxSize 를 원하는 최대값으로 초기화한다.
- public int front() // this._front+1 을 반환한다.
- public int rear() // this._rear +1 을 반환한다.
- public int maxSize() // this._maxSize 값을 반환한다.
- public boolean isEmpty() // this._front 와 this._rear 가 같은지 확인하여 같으면 true 값을 반환한다.
- public boolean isFull() // 다음 삽입될 위치가 this._front 와 같은지 확인하여 같으면 true 를 반환한다.
- public int size() // this._front 가 this._rear 보다 작거나 같으면 this._rear 에서 this._front 를 뺀 값을 반환하고, 클 경우 (this._rear + this._maxSize - this._front)의 값을 반환한다.
- public T frontElement() // Queue 가 비어있는 경우 null 을 반환한다. 비어있지 않는 경우 front element 를 반환한다.
- public Boolean enQueue(T anElement) // Queue 가 가득 차 있을 경우 false 를 반환하고 가득 차 있지 않을 경우 저장할 this._rear 를 적합한 위치로 조정하여 this._element 의 this._rear 위치에 anElement 를 삽입한다.
- public T deQueue() // Queue 가 비어있는 경우 null 을 반환하고 비어있지 않으면 front 에 있는 element 를 저장하기 위해

frontelement 를 선언하여 위치가 조정된
this._element[this._front]의 값을 저장하고
this._element[this._front]의 값은 초기화한다.

- public void clear() // this._front, this._rear, this._elements 를 초기화한다.
- public T elementAt(int givenPosition) // 원하는 위치의 원소를 반환한다.

3) 종합 설명서

: 키보드에서 문자를 반복 입력(한 번에 한 개의 문자)받으면 입력받은 문자에 따라 다음과 같이정해진 일을 하게 된다.

- 영문자 : 큐에 삽입하게 되며 5 개 이상 입력되면 큐가 full 이 되어 삽입이 불가능하다.
- ' ! ' : 큐의 원소를 모두 삭제하고 문자, 정상 처리된 문자, 무시된 문자, 삽입된 문자를 출력함과 동시에 프로그램을 종료한다.
- 숫자 문자 : 해당 수 만큼 큐에서 삭제하며 매번 삭제될 때 마다 삭제된 원소를 출력한다. 만약 삭제를 하려는데 큐가 Empty 상태라면 error 메시지를 출력한다.
- ' - ' : 큐가 비어있지 않다면 큐의 front 원소를 삭제한다
- ' # ' : 큐의 길이를 출력한다.
- ' / ' : 큐의 내용을 front 부터 rear 까지 차례로 출력한다.
- ' ^ ' : 큐의 front 원소의 값을 출력하고 큐는 변하지 않는다.
- 그 외 : Error message 를 출력하고 무시한다.

2. 프로그램 장단점 분석

: 리스트 큐의 단점을 보완하여 환영큐로 작성하였다. 단점이 있다면 위치 계산이 복잡하는 점과, 원소를 maxSize 보다 1 만큼 작은 개수의 원소를 저장할 수 있다는 것이다.

3. 실행 결과분석

1) 입력과 출력

```
> 프로그램을 시작합니다.
[ 큐 입력을 시작합니다. ]
- 문자를 입력하시오 : A
[EnQueue] 삽입된 원소는 A 입니다.
- 문자를 입력하시오 : B
[EnQueue] 삽입된 원소는 B 입니다.
- 문자를 입력하시오 : C
[EnQueue] 삽입된 원소는 C 입니다.
- 문자를 입력하시오 : D
[EnQueue] 삽입된 원소는 D 입니다.
- 문자를 입력하시오 : E
Error : 큐가 꽉 차서 삽입이 불가능 합니다.
- 문자를 입력하시오 : -
[DeQueue] 삭제된 원소는 A입니다.
- 문자를 입력하시오 : h
[EnQueue] 삽입된 원소는 h 입니다.
- 문자를 입력하시오 : /
[Queue]  <Front>  B  C  D  h  <Rear>
- 문자를 입력하시오 : -
[DeQueue] 삭제된 원소는 B입니다.
- 문자를 입력하시오 : z
[EnQueue] 삽입된 원소는 z 입니다.
- 문자를 입력하시오 : ^
[Front] 맨 앞 원소는 C입니다.
- 문자를 입력하시오 : /
[Queue]  <Front>  C  D  h  z  <Rear>
- 문자를 입력하시오 : 3
[RemoveN] 3개의 원소를 삭제하려고 합니다.
[DeQueue] 삭제된 원소는 C입니다.
[DeQueue] 삭제된 원소는 D입니다.
[DeQueue] 삭제된 원소는 h입니다.
- 문자를 입력하시오 : 3
[RemoveN] 3개의 원소를 삭제하려고 합니다.
[DeQueue] 삭제된 원소는 z입니다.
[Empty] 큐에 원소가 없습니다.
[Empty] 큐에 원소가 없습니다.
- 문자를 입력하시오 : B
[EnQueue] 삽입된 원소는 B 입니다.
```

```
- 문자를 입력하시오 : ]
Error : 의미 없는 문자가 입력되었습니다.
- 문자를 입력하시오 : k
[EnQueue] 삽입된 원소는 k 입니다.
- 문자를 입력하시오 : /
[Queue]  <Front>  B  k  <Rear>
- 문자를 입력하시오 : !
[ 큐 입력을 종료합니다. ]
[DeQueue] 삭제된 원소는 B입니다.
[DeQueue] 삭제된 원소는 k입니다.
... . .입력된 문자는 모두 18 개 입니다.
... . .정상 처리된 문자는 17 개 입니다.
... . .무시된 문자는 1 개 입니다.
... . .삽입된 문자는 8 개 입니다.

> 프로그램을 종료합니다.
```

2) 결과 분석

: 원하는 바와 같이 정상적으로 실행된다.

4. 소스코드

1) AppController

```
public class AppController {
    private AppView _appView;
    private CircularArrayQueue _arrayQueue;
    private int _inputChars;
    private int _ignoredChars;
    private int _addedChars;

    public AppController() {
        this._appView = new AppView();
        this.initCharCounts();
    }

    public void showFrontElement() {
        this._appView.outputFrontElement((Character) this._arrayQueue
                .frontElement());
    }

    public void showQueueSize() {
        this._appView.outputQueueSize(this._arrayQueue.size());

    }

    public void showAll() {
        this.showMessage(MessageID.Show_QueueStart);
        for (int i = this._arrayQueue.front(); i != this._arrayQueue.rear(); i++) {
            if (i == this._arrayQueue.maxSize())
                i = 0;
            this._appView.outputElement((Character) this._arrayQueue
                    .elementAt(i));
        }
        this.showMessage(MessageID.Show_QueueEnd);

    }

    public void initCharCounts() {
        this._inputChars = 0;
        this._addedChars = 0;
        this._ignoredChars = 0;
    }

    public void countAdded() {
        this._addedChars++;
    }
```

```
public void countIgnored() {
    this._ignoredChars++;
}

public void countInputChar() {
    this._inputChars++;
}

@SuppressWarnings("unchecked")
public void add(char c) {
    if (this._arrayQueue.isFull())
        this.showMessage(MessageID.Error_InputFull);
    else {
        if (this._arrayQueue.enQueue(c)) {
            this._appView.outputAdd(c);
            this.countAdded();
        }
    }

}

public void removeOne() {
    if (!this._arrayQueue.isEmpty()) {
        char removed = (char) (Character) this._arrayQueue.deQueue();
        this._appView.outputRemove(removed);
    }

    else
        this.showMessage(MessageID.Error_Empty);
}

public void removeN(int numOfCharsToBeDeleted) {
    this._appView.outputRemoveN(numOfCharsToBeDeleted);
    for (int i = 0; i < numOfCharsToBeDeleted; i++)
        this.removeOne();

}

public void conclusion() {
    for (int i = this._arrayQueue.size() - 1; i > -1; i--)
        this.removeOne();
    this._appView.outputResult(this._inputChars, this._ignoredChars,
            this._addedChars);

}
public void showMessage(MessageID aMessageID) {
    switch (aMessageID) {
    case Notice_StartProgram:
        System.out.println("> 프로그램을 시작합니다.");
        break;
    case Notice_StartMenu:
        System.out.println("[ 큐 입력을 시작합니다. ]");
        break;
    case Notice_EndProgram:
        System.out.println("> 프로그램을 종료합니다.");
        break;
    case Notice_EndMenu:
        System.out.println("[ 큐 입력을 종료합니다. ]");
        break;
    case Error_InputFull:
        System.out.println("Error : 큐가 꽉 차서 삽입이 불가능 합니다.");
        break;
    case Error_Empty:
        System.out.println("[Empty] 큐에 원소가 없습니다. ");
        break;
    case Show_QueueStart:
        System.out.print("[Queue]  <Front> ");
        break;
    case Show_QueueEnd:
        System.out.println(" <Rear>");
        break;
    default:
        System.out.println("Error : 의미 없는 문자가 입력되었습니다.");
    }
}
```

```
    public void run() {
        this._arrayQueue = new CircularArrayQueue<Character>();
        char input;

        this.showMessage(MessageID.Notice_StartProgram);
        this.showMessage(MessageID.Notice_StartMenu);

        input = this._appView.inputCharacter();
        while (input != '!') {
            this.countInputChar();
            if ((input >= 'A' && input <= 'Z')
                    || (input >= 'a' && input <= 'z'))
                this.add(input);
            else if (input >= '0' && input <= '9')
                this.removeN(Integer.parseInt(String.valueOf(input)));
            else if (input == '-')
                this.removeOne();
            else if (input == '#')
                this.showQueueSize();
            else if (input == '/')
                this.showAll();
            else if (input == '^')
                this.showFrontElement();
            else {
                this.showMessage(MessageID.Error_WrongMenu);
                this.countIgnored();
            }
            input = this._appView.inputCharacter();

        }
        this.showMessage(MessageID.Notice_EndMenu);
        this.conclusion();
        this.showMessage(MessageID.Notice_EndProgram);
    }
}
```

2) AppView

```
import java.util.*;

public class AppView {
    private Scanner _scanner;

    public AppView() {
        this._scanner = new Scanner(System.in);

    }

    public String inputString() {
        return this._scanner.next();
    }

    public char inputCharacter() {
        System.out.print("- 문자를 입력하시오 : ");
        return this.inputString().charAt(0);
    }

    public void outputAdd(char anElement) {
        System.out.println("[EnQueue] 삽입된 원소는 " + anElement + " 입니다. ");

    }

    public void outputElement(char anElement) {
        System.out.print(" " + anElement + " ");
    }

    public void outputFrontElement(char anElement) {
        System.out.println("[Front] 맨 앞 원소는 " + anElement + "입니다. ");
    }

    public void outputQueueSize(int aQueueSize) {
        System.out.println("[Size] 큐에는 현재 " + aQueueSize + "개의 원소가 있습니다.");
    }

    public void outputRemove(char anElement) {
        System.out.println("[DeQueue] 삭제된 원소는 " + anElement + "입니다.");

    }
```

```
    public void outputRemoveN(int aNumOfCharsToBeRemoved) {
        System.out.println("[RemoveN] " + aNumOfCharsToBeRemoved
                + "개의 원소를 삭제하려고 합니다.");
    }

    public void outputResult(int aNumOfInputChars, int aNumOfIgnoredChars,
            int aNumOfAddedChars) {
        System.out.println("... . .입력된 문자는 모두 " + aNumOfInputChars + " 개 입니다.");
        System.out.println("... . .정상 처리된 문자는 "
                + (aNumOfInputChars - aNumOfIgnoredChars) + " 개 입니다.");
        System.out.println("... . .무시된 문자는 " + aNumOfIgnoredChars + " 개 입니다.");
        System.out.println("... . .삽입된 문자는 " + aNumOfAddedChars + " 개 입니다.");
        System.out.println();

    }
}
```

3) CircularArrayQueue

```
public class CircularArrayQueue<T> {
    private static final int DEFAULT_INITIAL_CAPACITY = 5;
    private int _maxSize;
    private int _front;
    private int _rear;
    private T[] _elements;

    public CircularArrayQueue() {
        this(CircularArrayQueue.DEFAULT_INITIAL_CAPACITY);
    }

    public CircularArrayQueue(int initialCapacity) {
        this._maxSize = initialCapacity;
        this._front = 0;
        this._rear = 0;
        this._elements = (T[]) new Object[this._maxSize];
    }

    public int front() {
        return this._front + 1;
    }

    public int rear() {
        return this._rear + 1;
    }

    public int maxSize() {
        return this._maxSize;
    }

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

    public boolean isFull() {
        return ((this._rear + 1) % this._maxSize == this._front);
    }

    public int size() {
        if (this._front <= this._rear)
            return (this._rear - this._front);
        else
            return ((this._rear + this._maxSize) - this._front);
    }
```

```
    public T frontElement() {
        if (this.isEmpty())
            return null;
        else
            return this._elements[this.front()];
    }

    public boolean enQueue(T anElement) {
        if (this.isFull()) {
            return false;
        } else {
            this._rear = (this._rear + 1) % this._maxSize;
            this._elements[this._rear] = anElement;
            return true;
        }
    }

    public T deQueue() {
        T frontElement;
        this._front = this.front();
        if (this._front == _maxSize)
            this._front = 0;
        frontElement = this._elements[this._front];
        this._elements[this._front] = null;
        return frontElement;
    }

    public void clear() {
        for (int i = 1; i <= this.size(); i++)
            this._elements[(this._front + i) % this._maxSize] = null;
        this._front = 0;
        this._rear = 0;

    }

    public T elementAt(int givenPosition) {
        return this._elements[givenPosition];
    }
}
```

4) MessageID

```
public enum MessageID {
    // Message IDs for Notices:
    Notice_StartProgram, Notice_EndProgram,
    Notice_StartMenu,
    Notice_EndMenu,
    Show_QueueStart,
    Show_QueueEnd,
    // MessageIDs for Errors:
    Error_WrongMenu,
    Error_InputFull,
    Error_Empty,
}
```

5) DS1_09_201402395_이승희

```
public class DS1_09_201402395_이승희 {

    public static void main(String[] args) {
        AppController appController = new AppController();
        appController.run();
    }

}
```

5. 문제 9

1) 큐

(1) 개념

: 리스트의 한쪽 끝에서만 삽입과 삭제가 일어나는 (후입선출) 스택과는 달리 리스트의 한쪽 끝에서는 원소들이 삭제되고 반대쪽 끝에서는 원소들의 삽입만 가능하게 만든 순서화된 리스트이다. 가장 먼저 리스트에 삽입된 원소가 가장 먼저 삭제되는 선입선출의 개념이다.

(2) 사용법

: A, B, C, D, E 의 원소를 각각 삽입하게 되면 큐에는 [A B C D E]의 형태로 존재하며 여기서 front 는 A, rear 는 E 이다. 원소의 삭제는 front 에서 일어나며 원소의 추가는 rear 에서 일어난다. 만약 원소를 삭제하고 'K'라는 원소를 추가한다면 큐는 [B C D E K]의 형태를 가진다.

2) 환형큐

- 이해 및 구조

: 원형 큐는 배열을 가상으로 가정하고 원형 배열에 데이터를 저장함으로써 배열의 마지막에 도달하여 저장공간이 없으면 배열의 처음에 다음 원소를 저장하여 원형 배열로 다루게된다.

3) 큐를 환형 큐 형태로 사용하는 이유는?

: 리스트 형태의 큐에 대한 삽입과 삭제 작업이 반복됨에 따라 점차 오른쪽으로 front 와 rear 의 위치가 이동한다. 결국 rear 가 큐의 사이즈보다 1 만큼 작은 곳에 존재하여 큐가 포화상태가 된다. 그러나 front 의 왼쪽에는 빈 공간이 있다. 이를 해결하기 위해서는 front 위치부터 rear 의 위치까지의 자료를 다시 '0'번째부터 위치하도록 옮기는 작업을 해야 한다. 이는 상당한 성능 손실을 가져오게 된다.

4) 큐는 실제 구현할 때 front 와 rear 의 위치를 계산하여야 한다. 보고서에 본인이 구현한 위치 계산법을 설명여라.

: 큐에 원소를 모두 채우면 큐가 empty 인지 full 인지 구분이 불가능하기 때문에 maxSize 보다 하나 작은만큼 원소를 채울 수 있다. 즉, (this._rear + 1)%this._maxSize 의 값이 this._front 와 같으면 큐가 full 이라고 할 수 있다. 따라서, 원소를 추가하면 this._rear 의 위치는 (this._rear + 1)%this._maxSize 와 같으며 이 위치에 해당하는 this._element[this._rear]에 값을 저장하면 된다. Front 의 위치를 변하지 않는다.