자료구조 실습 보고서

[제 12 주] 리스트 반복자

제출일 : 2015 – 05 – 23

201402395 이승희

1. 프로그램 설명서

1) 주요 알고리즘 / 자료구조 / 기타

: 정렬된 ArrayList와 LinkedList에 원소를 삽입, 삭제 및 다른 행위들을 구현한다. 사용자는 리스트에 직접 접근하지 않고 반복자가 인스턴스 변수들에게 직접 접근한다. 이를 구현함에 있어서, 반복자를 List의 내부 클래스로 선언하면 List의 private 인스턴스 변수들을 직접 사용할 수 있으므로 효율적인 구현이 가능하다. 하나의 리스트에, 필요에 따라 동시에 여러 개의 반복자 객체를 생성할 수 있다.

2) 함수 설명서

(1) AppController

- Public void showSize() // this._appView의 outputSize 메소드를 사용하여 sortedList의 size를 출력한다
- Public void reset() // List를 초기화한다
- Public void showAll() // List의 모든 원소를 출력한다
- Public void add(int inputValue) // List에 원소를 추가한다
- Public void removeMin() // List의 가장 작은 값의 원소를 삭제한다
- Public void removeMax() // List의 가장 큰 값의 원소를 삭제한다.
- Public void removeFrom(int aPosition) // List의 aPosition번째의 원소를 삭제한다.
- Public void run() // 프로그램의 실질적인 main함수의 역할을 한다

(2) AppView

- Public String inputString() // Scanner을 이용하여 문자열을 입력받는다.
- Public char inputCharacter() // 명령어를 입력받는다
- Public int inputNumber() // 정수를 입력받는다
- Public void outputSize(int size) // size를 출력한다.
- Public void outputAdd(int anElement) // List에 추가된 원소 anElement를 출력

한다

- Public void outputRemove(int anElement) // List에서 삭제할 원소 anElement를 출력한다

(3) List

- Public void clear() // List를 초기화
- Public int size() // List의 원소의 개수를 반환
- Public Boolean isFull() // List가 가득 차 있는지 확인하여 가득 차 있으며 true를 반환
- Public Boolean isEmpty() // List가 비어 있는지 확인하여 비어 있으면 true를 반환
- Public Boolean contains(T anElement) // List의 내부에 있는 값인지 확인하는 함수. anElement 와 같은 값이 List에 존재하면 true를 반환
- Public Boolean add(T anElement) // List에 anElement를 삽입하여 삽입이 성공할 경우 true를 반환
- Public T removeMin() // List에서 가장 작은 원소를 삭제하고 삭제된 원소를 반환
- Public T removeMax() // List에서 가장 큰 원소를 삭제하고 삭제된 원소를 반환
- Public T removeFrom(int aPosition) // List에서 aPosition번째의 원소를 삭제하고 삭제된 원소를 반환

(4) Iterator

- Public Boolean hasNext() // 현재 원소의 다음 원소가 존재하는지 확인
- Public T next() // 현재 원소의 다음 원소를 반환

(5) SortedLinkedList (interface List를 상속받아 List의 메소드와 기능 동일)

- Public SortedLinkedList() // 생성자
- Public SortedLinkedList(int aMaxSize) // 생성자

- Public ListIterator<T> listIterator()
- Public class ListIterator<T> implements Iterator // interface Iterator를 상속받아 Iterator의 메소드와 기능 동일

(6) SortedArrayList // SortedLinkedList와 이하 동일

3) 종합 설명서

: 정렬된 리스트를 LinkedList, ArrayList로 각각 구현한다. 원소가 정수형인 리스트는 키보드의 입력에 따라 다양한 일을 수행한다.

- '%' : 정수 입력을 받는다. 받은 값은 리스트에 삽입하며, 삽입할 때 이미 존재하는지 미리 확인하여 존재하지 않으면 삽입한다. 만약 동일한 원소가 이미 존재할 경우 오류 메시지를 출력한다.
- '~' : 리스트를 초기화한다.
- '-' : 리스트의 최소값을 삭제한 후 삭제된 값을 출력한다. 삭제 전에는 리스트가 비어있는 상태인지 검사하여 만약 비어있으면 오류 메시지를 출력한다. 리스트는 정렬된 상태이므로 여기서 최소값은 리스트의 맨 앞의 값에 해당한다.
- '+' : 리스트에서 최대값을 삭제한 후 삭제된 값을 출력한다. 리스트는 정렬된 상태이므로 여기서 최대값은 리스트의 가장 마지막의 값에 해당한다.
- '#' : 리스트의 길이를 출력한다.
- '?' : 정수를 입력 받는다. 입력 받은 정수는 리스트의 크기 순서에 해당하며 리스트 상의 값을 찾아 삭제하고 그 원소를 출력한다. 만약 입력 받은 정수가 0보다 작거나 또는 리스트의 개수보다 크거나 같으면 찾을 수 없으므로, 적절한 오류 메시지를 내보낸 후 다시 입력 받는다.
- '/' : 리스트의 값을 차례대로 출력한다.
- '!' : 프로그램을 종료한다.

2. 프로그램 장단점 분석

: 반복자의 개념을 사용하면, 사용자는 리스트에 직접 접근하지 않고 리스트를 스캔한 반복자를 이용하여 간접적으로 접근하게 된다. 명령에 따른 기능의 수행은 반복자가 수행하게 되며, 이는 기능을 수행함에 있어서 리스트의 값이 변할 수 있는 오류를 범하지 않게

된다. 또한 반복자는 리스트의 인덱스와 같은 역할을 하여 현재의 위치에 해당하는 원소를 기억하게 된다. 이는 리스트의 값을 가져올 때 더 효율적이다.

3. 실행 결과 분석

1) 입력과 출력

```
<리스트를 시작합니다>
> 문자를 입력하시오 : %
- 숫자를 입력하시오 : 89
[Insert] 삽입된 원소는 89입니다.
> 문자를 입력하시오 : %
- 숫자를 입력하시오 : 41
[Insert] 삽입된 원소는 41입니다.
> 문자를 입력하시오 : %
- 숫자를 입력하시오 : 2
[Insert] 삽입된 원소는 2입니다.
> 문자를 입력하시오 : %
- 숫자를 입력하시오 : 49
[Insert] 삽입된 원소는 49입니다.
> 문자를 입력하시오 : /
[LIST] 2 41 49 89
> 문자를 입력하시오 : +
[Delete] 삭제된 원소는 89입니다.
> 문자를 입력하시오 : %
- 숫자를 입력하시오 : 34
[Insert] 삽입된 원소는 34입니다.
> 문자를 입력하시오 : %
- 숫자를 입력하시오 : 77
[Insert] 삽입된 원소는 77입니다.
> 문자를 입력하시오 : %
- 숫자를 입력하시오 : 1
[Insert] 삽입된 원소는 1입니다.
> 문자를 입력하시오 : -

[Delete] 삭제된 원소는 1입니다.
> 문자를 입력하시오 : /
[LIST] 2 34 41 49 77
> 문자를 입력하시오 : #
[Length] 리스트에는 현재 5개가 있습니다.
> 문자를 입력하시오 : ?
- 숫자를 입력하시오 : 4
[Delete] 삭제된 원소는 77입니다.
> 문자를 입력하시오 : ~
- 리스트를 비웠습니다
> 문자를 입력하시오 : /
[LIST]
> 문자를 입력하시오 : !
<리스트가 끝났습니다>
```

2) 결과 분석

: LinkedList를 사용하였고, 원하는 바와 같이 출력되었다.

4. 소스코드

1) AppController

```java
public class AppController {
    private AppView _appView;
    private SortedLinkedList<Integer> _sortedList;

    // private SortedArrayList<Integer> _sortedList;

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

    public void showSize() {
        this._appView.outputSize(this._sortedList.size());
    }

    public void reset() {
        this._sortedList.clear();
        this.showMessage(MessageID.Notice_Reset);
    }

    public void showAll() {
        this.showMessage(MessageID.Notice_ShowStartList);
        SortedLinkedList.ListIterator list = (SortedLinkedList.ListIterator) this._sortedList
                .listIterator();

        // SortedArrayList.ListIterator list = (SortedArrayList.ListIterator)
        // this._sortedList
        // .listIterator();

        while (list.hasNext()) {
            this._appView.outputElement((Comparable) list.next());

        }
        this.showMessage(MessageID.Notice_ShowEndList);

    }

    public void add(int inputValue) {
        Integer input = inputValue;
        if (this._sortedList.add(input))
            this._appView.outputAdd(input);
        else
            this.showMessage(MessageID.Error_WrongMenu);
    }


    public void removeMin() {
        this._appView.outputRemove((Integer) this._sortedList.removeMin());
    }

    public void removeMax() {
        this._appView.outputRemove((Integer) this._sortedList.removeMax());
    }

    public void removeFrom(int aPosition) {
        this._appView.outputRemove((Integer) this._sortedList.removeFrom(aPosition));
    }

    public void run() {
        this._sortedList = new SortedLinkedList<Integer>();
        // this._sortedList = new SortedArrayList<Integer>();
        char command = 0;
        int input;

        this.showMessage(MessageID.Notice_StartProgram);
        while (command != '!') {
            command = this._appView.inputCharacter();
            if (command == '%') {
                input = this._appView.inputNumber();
                this.add(input);
            } else if (command == '~')
                this.reset();
            else if (command == '-')
                this.removeMin();
            else if (command == '+')
                this.removeMax();
            else if (command == '#')
                this.showSize();
            else if (command == '?') {
                input = this._appView.inputNumber();
                this.removeFrom(input);
            } else if (command == '/')
                this.showAll();
            else if (command == '!')
                break;
            else
                this.showMessage(MessageID.Error_WrongMenu);

        }
        this.showMessage(MessageID.Notice_EndProgram);

    }
```

```
    private void showMessage(MessageID aMessageID) {
        switch (aMessageID) {
        case Notice_StartProgram:
            System.out.println("<리스트를 시작합니다>");
            break;
        case Error_WrongMenu:
            System.out.println("Error : 잘못된 입력입니다 ");
            break;
        case Notice_EndProgram:
            System.out.println("<리스트가 끝났습니다>");
            break;
        case Notice_ShowStartList:
            System.out.print("[LIST] ");
            break;
        case Notice_Reset:
            System.out.println("- 리스트를 비웠습니다");
            break;
        case Notice_ShowEndList:
            System.out.println("");
            break;
        case Error_InputFull:
            System.out.println("[Error] 리스트가 가득 차 있습니다.");
            break;
        case Error_Empty:
            System.out.println("[Error] 리스트가 비어있습니다.");
            break;
        default:
            break;
        }
    }
}
```

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._scanner.next().charAt(0);
    }

    public int inputNumber() {
        System.out.print("- 숫자를 입력하시오 : ");
        return this._scanner.nextInt();
    }

    public void outputSize(int size) {
        System.out.println("[Length] 리스트에는 현재 "+size+"개가 있습니다.");
    }

    public void outputAdd(Comparable anElement) {
        System.out.println("[Insert] 삽입된 원소는 "+anElement+"입니다. ");
    }

    public void outputRemove(Comparable anElement) {
        System.out.println("[Delete] 삭제된 원소는 "+anElement+"입니다. ");
    }

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

3) DS1_12_201402395_이승희

```
public class DS1_12_201402395_이승희 {
    public static void main(String[] args) {
        AppController appController = new AppController();
        appController.run();
    }

}
```

4) Iterator

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

5) List

```java
public interface List<T> {

    public void clear();
    public int size();

    public boolean contains(T anElement);
    public boolean isFull();
    public boolean isEmpty();

    public boolean add(T anElement);

    public T removeMin();
    public T removeMax();
    public T removeFrom(int aPosition);


}
```

6) MessageID

```java
public enum MessageID {
    Notice_StartProgram, Notice_EndProgram,
    Notice_Reset,
    Notice_ShowStartList,
    Notice_ShowEndList,

    Error_WrongMenu,
    Error_InputFull,
    Error_Empty,
}
```

7) Node

```java
public class Node<T extends Comparable>implements Comparable {
    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;
    }

    @Override
    public int compareTo(Object arg0){
        return (_element.compareTo(arg0));
    }


}
```

8) SortedArrayList

```
public class SortedArrayList<T extends Comparable> implements List {
    private static final int DEFAULT_MAX_SIZE = 20;
    private int _maxSize;
    private int _size;
    private T[] _element;

    public SortedArrayList() {
        this._maxSize = this.DEFAULT_MAX_SIZE;
        this._size = 0;
        this._element = (T[]) new Comparable[this._maxSize];
    }

    public SortedArrayList(int aMaxSize) {
        this._maxSize = aMaxSize;
        this._size = 0;
        this._element = (T[]) new Comparable[this._maxSize];

    }

    @Override
    public void clear() {
        this._size = 0;
        this._element = (T[]) new Comparable[this._maxSize];

    }

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

    @Override
    public boolean contains(Object anElement) {
        boolean found = false;
        for (int i = 0; i < this._size; i++)
            if (this._element[i].compareTo(anElement) == 0)
                found = true;

        return found;
    }

    @Override
    public boolean isFull() {
        return this._maxSize == this._size;
    }
```

```
    @Override
    public boolean isEmpty() {
        return this._size == 0;
    }

    @Override
    public boolean add(Object anElement) {
        if (!this.contains(anElement)) {
            if (!isFull()) {
                if (this._size == 0) {
                    this._element[0] = (T) anElement;
                } else {
                    int i = 0;
                    while (this._element[i] != null) {
                        if (this._element[i].compareTo(anElement) > 0)
                            break;
                        else
                            i++;
                    }
                    for (int j = this._size; j > i; j--)
                        this._element[j] = this._element[j - 1];
                    this._element[i] = (T) anElement;

                    this._size++;
                    return true;
                }
            }
        }
        return false;

    }

    @Override
    public Object removeMin() {
        if (this.isEmpty())
            return null;
        else {
            T removedElement = this._element[0];
            for (int i = 0; i < this._size; i++)
                this._element[i] = this._element[i + 1];
            this._size--;
            return removedElement;
        }
    }
```

```
    @Override
    public Object removeMax() {
        if (isEmpty())
            return null;
        else {
            T removedElement = this._element[this._size - 1];
            this._element[this._size - 1] = null;
            this._size--;
            return removedElement;
        }
    }

    @Override
    public Object removeFrom(int aPosition) {
        if (isEmpty())
            return null;
        else {
            T removedElement = this._element[aPosition];
            for (int i = aPosition; i < this._size; i++)
                this._element[i] = this._element[i + 1];
            this._element[this._size] = null;
            this._size--;
            return removedElement;
        }
    }

    public ListIterator<T> listIterator() {
        return new ListIterator();
    }


    public class ListIterator<T> implements Iterator {
        private int _nextPosition;

        private ListIterator() {
            this._nextPosition = 0;
        }

        public boolean hasNext() {
            return this._nextPosition < size();
        }

        public T next() {
            if (this._nextPosition == size())
                return null;
            else {
                T nextElement = (T) _element[this._nextPosition];
                this._nextPosition++;
                return nextElement;
            }
        }
    }

}
```

9) SortedLinkedList

```
public class SortedLinkedList<T extends Comparable> implements List {
    private static final int DEFAULT_MAX_SIZE = 20;
    private int _maxSize;
    private int _size;
    private Node _head;

    public SortedLinkedList() {
        this._maxSize = this.DEFAULT_MAX_SIZE;
        this._size = 0;
        this._head = null;
    }

    public SortedLinkedList(int aMaxSize) {
        this._maxSize = aMaxSize;
        this._size = 0;
        this._head = null;
    }

    @Override
    public void clear() {
        this._size = 0;
        this._head = null;

    }

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

    @Override
    public boolean contains(Object anElement) {
        boolean found = false;
        Node searchNode = this._head;
        while (searchNode != null && !found) {
            if (searchNode.compareTo(anElement) == 0)
                found = true;
            searchNode = searchNode.next();
        }
        return found;

    }


    @Override
    public boolean isFull() {
        return this._maxSize == this._size;
    }

    @Override
    public boolean isEmpty() {
        return this._size == 0;
    }



    @Override
    public T removeMin() {
        if (isEmpty())
            return null;
        else {
            Node headNode = this._head;
            this._head = this._head.next();
            this._size--;
            return (T) headNode.element();
        }
    }

    @Override
    public T removeMax() {
        Node previousNode = null;
        Node currentNode = this._head;

        if (isEmpty())
            return null;
        else {
            for (int i = 0; i < this._size - 1; i++) {
                previousNode = currentNode;
                currentNode = currentNode.next();
            }
            previousNode.setNext(currentNode.next());
        }
        this._size--;
        return (T) currentNode.element();
    }
```

```
    @Override
    public T removeFrom(int aPosition) {
        if (aPosition < 0 && aPosition >= this._size)
            return null;
        else {
            Node previousNode = null;
            Node currentNode = this._head;
            if (aPosition == 0)
                this._head = this._head.next();
            else {
                for (int i = 0; i < aPosition; i++) {
                    previousNode = currentNode;
                    currentNode = currentNode.next();
                }
                previousNode.setNext(currentNode.next());
            }
            this._size--;
            return (T) currentNode.element();
        }
    }

    public ListIterator<T> listIterator() {
        return new ListIterator();
    }

    public class ListIterator<T> implements Iterator {
        private Node _nextNode;

        private ListIterator() {
            this._nextNode = _head;
        }

        public boolean hasNext() {
            return (this._nextNode != null);
        }

        public T next() {
            if (this._nextNode == null) {
                return null;
            } else {
                T element = (T) this._nextNode.element();
                this._nextNode = this._nextNode.next();
                return element;
            }
        }
    }


    @Override
    public boolean add(Object anElement) {
        if (this.contains(anElement))
            return false;
        else {
            Node newNode = new Node((T) anElement);
            Node searchNode = this._head;
            Node previousNode = null;

            while (searchNode != null) {
                if (searchNode.compareTo(anElement) > 0) {
                    break;
                } else {
                    previousNode = searchNode;
                    searchNode = searchNode.next();
                }
            }

            if (searchNode == this._head) {
                newNode.setNext(searchNode);
                this._head = newNode;
            } else {
                newNode.setNext(searchNode);
                previousNode.setNext(newNode);
            }
            this._size++;
            return true;

        }
    }

}
```

5. 요약

1) List의 이해

(1) 기본 개념

: ArrayList와 LinkedList에 필요한 메소드들을 인터페이스로 정의한다. 이 List를 상

속받은 두 개의 서로 다른 리스트는 각각 다른 방법으로 구현된다.

(2) 구현

- Array List

  : 일반 배열로 구현한다.

- Linked List

  : 연결 체인으로 구현한다.

(3) 구현의 독립성

: List의 공개함수를 캡슐화하여 List 객체 사용자는 공개함수로만 접근한다. List는 사용자에게 공개함수만 제공하므로 독립적이라 할 수 있다.

2) Iterator의 이해

(1) 개념

- Friend Class

  : Iterator를 구현하기 위해서는 어쩔 수 없이 List의 private 속성에 접근해야 한다. 하지만 이는 private 정의에 어긋나게 된다. 그러나 Iterator를 List의 내부 클래스로 구현하면 private 속성에 직접 접근할 수 있게 된다.

(2) 사용법

: 공통된 메소드를 인터페이스로 정의한다.

(3) 구현

: List의 내부 클래스에서 구현한다.

3) 두 구현의 장단점을 각 행위 별로 비교

: 행위 별로 구현함에 있어서 LinkedList는 ArrayList보다 원소의 삽입, 삭제가 더 효율적이다. ArrayList는 원소의 삽입과 삭제가 일어날 때 마다 배열 전체의 값에 접근해야 하는 단점이 있기 때문이다. 하지만 removeFrom(int aPosition)과 같이 원소의 인덱스 값을 알고 있는 경우 ArrayList가 더 효율적이다. 배열의 인덱스 값을 알면 값에 바로 접근할 수 있는 반면 LinkedList는 head부터 차례대로 검색해야 하기 때문이다.

4) Iterator를 사용해 보고 느낀 점

: 사용이 익숙치 않고 개념이 어려워서 불편하지만, 구현한 프로그램과 같이 소규모의 프로그램은 효율이 떨어지는 것 같다. 하지만 행위를 구현하면서 리스트의 값이 변할 수 있는 오류는 범하지 않는 장점은 큰 것 같다.