Lab Exercise #8

연결 리스트를 이용한 리스트 구현 실습

2018년도 2학기

컴퓨터프로그래밍2

김 영 국

충남대학교 컴퓨터공학과

목차

- 실습
  - LinkedList 구현 실습
- 과제
  - indexOf(), set() 메소드 구현
  - 다항식의 덧셈

실습8-1. LinkedList 구현 실습(1)

- List 인터페이스를 구현하는 LinkedList를 만들어보자.
  - LinkedList<T>는 실습7-1에서 만들었던 List<T>를 구현한다.

```
public class LinkedList<T> implements List<T>
```

  - LinkedList<T>에서 내부적으로 사용할 Node<E> 클래스를 다음과 같이 만들어보자.

```
private class Node<E> {
    private E data; // 데이터가 저장될 필드
    private Node<E> next; // 다음 노드를 가리키는 필드

    public Node(E input) {
        this.data = input; //데이터 저장
        this.next = null;
    }

    @Override
    public String toString() {
        // 노드의 데이터를 확인
        return String.valueOf(this.data);
    }
}
```

실습8-1. LinkedList 구현 실습(2)

- LinkedList<T>의 필드를 다음과 같이 만들어보자.

```
private Node<T> head; // 첫 번째 노드를 가리키는 필드
private Node<T> tail; // 마지막 노드를 가리키는 필드
private int size = 0; // 리스트의 원소의 개수
```

- LinkedList<T>의 toString() 메소드를 다음과 같이 만들어보자.

```
@Override
public String toString() {
    // 리스트에 원소가 없으면 head가 null이다.
    if(head == null)
        return "[]";
    // head 노드를 temp로 참조
    Node<T> temp = head;
    StringBuilder str = new StringBuilder("[");
    while(temp.next != null) {
        // temp의 다음 연결이 null이 아니면 while문 실행
        str.append(temp.data + ", ");
        temp = temp.next;
    }
    // 다음 연결이 null인 노드지만 노드에 데이터는
    // 있기에 str에 append해준다.
    str.append(temp.data);
    return new String(str + "]");
}
```

실습8-1. LinkedList 구현 실습(3)

- LinkedList<T>의 isEmpty(), size() 메소드를 다음과 같이 만들어보자.

```
@Override
public int size() {
    return size;
}

@Override
public boolean isEmpty() {
    return size == 0;
}
```

- size(), isEmpty(), toString 메소드의 테스트 코드와 결과

```
public class LinkedListTest {
    public static void main(String[] args) {
        List<Integer> list = new LinkedList<>();
        System.out.println("list size: " + list.size());
        System.out.println("list isEmpty?: "+ list.isEmpty());
        System.out.println("list: " + list);
    }
}
```

```
list size: 0
list isEmpty?: true
list: []
```

실습8-1. LinkedList 구현 실습(4)

- LinkedList에 데이터를 추가하는 addFirst(), addLast(), add() 메소드와 LinkedList 내부에서만 사용하고 해당 인덱스의 Node를 반환하는 node() 메소드를 만들어보자.
  - addFirst() 메소드

```
@Override
public boolean addFirst(T element) {
    // 새로운 노드 생성
    Node<T> newNode = new Node<>(element);
    // 첫 번째 자리에 위치하므로 가장 첫번째 노드인 헤드로
    // newNode의 next로 연결해준다.
    newNode.next = head;
    // 헤드를 newNode로 옮겨준다.
    head = newNode;
    size++;
    // 만약 head.next == null인경우, 즉 리스트 원소가 하나일경우
    // 헤드와 tail은 동일하다.
    if(head.next == null)
        tail = head;
    return true;
}
```

실습8-1. LinkedList 구현 실습(5)

- addLast() 메소드

```
@Override
public boolean addLast(T element) {
    // size가 0인 경우에는 앞서 구현했던 addFirst()를 호출한다.
    if(isEmpty())
        addFirst(element);
    else {
        Node<T> newNode = new Node<>(element);
        tail.next = newNode;
        tail = newNode;
        size++;
    }
    return true;
}
```

- node() 메소드

```
// LinkedList 내부에서만 사용하기에 private으로 선언
private Node<T> node(int index) {
    Node<T> x = head;
    // head노드에서 index번 next를 호출하면
    // LinkedList의 index node를 알 수 있다.
    for(int i = 0; i < index; i++)
        x = x.next;
    return x;
}
```

실습8-1. LinkedList 구현 실습(6)

- add() 메소드

```
@Override
public boolean add(int index, T element) {
    // 잘못된 index가 오면 Exception 발생
    if(index < 0 || index > size)
        throw new IndexOutOfBoundsException();
    if(index == 0) {
        // index가 0이면 addFirst로 원소 삽입
        addFirst(element);
    } else {
        // 인자로 온 index 바로 전의 노드를 temp1에 지정
        Node<T> temp1 = node(index - 1);
        // 인자로 온 index 노드를 temp2에 지정
        Node<T> temp2 = temp1.next;
        // 새로운 노드를 만들고...
        Node<T> newNode = new Node<>(element);
        // 해당 index 전 노드인 temp1의 다음을 newNode로 지정
        temp1.next = newNode;
        // newNode 다음을 temp2로 지정
        newNode.next = temp2;
        size++;
        // 만약 이전에 temp1이 마지막 노드여서 index에 해당하는 노드가 null이면
        // temp2는 null이되고 newNode.next는 null을 가리키게된다.
        // 이런 경우에는 아래 if문을 통해 newNode가 tail 노드가 된다.
        if(newNode.next == null) {
            tail = newNode;
        }
    }
    return true;
}
```

실습8-1. LinkedList 구현 실습(7)

- addFirst(), addLast(), add() 메소드의 테스트 코드와 결과

```
public class LinkedListTest {
    public static void main(String[] args) {
        List<Integer> list = new LinkedList<>();
        for(int i = 5; i >= 0; i--)
            list.addFirst(i);
        for(int i = 11; i <= 15; i++)
            list.addLast(i);
        for(int i = 6; i <= 10; i++)
            list.add(i, i);
        System.out.println(list);
    }
}
```

```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15]
```

- 인덱스에 해당하는 값을 반환하고 제거하는 remove() 메소드와 리스트의 가장 첫 번째 원소를 반환하고 제거하는 removeFirst() 메소드, 가장 마지막 원소를 반환하고 제거하는 removeLast() 메소드를 작성해보자. 그후 원소를 제거하지 않고 반환하는 get() 메소드를 작성해보자.

실습8-1. LinkedList 구현 실습(8)

- removeFirst() 메소드

```
@Override
public T removeFirst() {
    Node<T> temp = head; // 헤드노드를 참조
    head = temp.next; //
    T returnData = temp.data;
    temp = null; // 자바에서는 굳이 필요 없는 문장, 하지만 학습을 위해 사용해보자.
    size--;
    return returnData;
}
```

- removeLast 메소드

```
@Override
public T removeLast() {
    return remove(size - 1);
}
```

실습8-1. LinkedList 구현 실습(9)

- remove() 메소드

```
@Override
public T remove(int index) {
    if(index < 0 || index >= size)
        throw new IndexOutOfBoundsException();
    if(index == 0)
        return removeFirst();

    Node<T> temp = node(index - 1); // 삭제될 노드의 전 노드
    Node<T> nodeToDelete = temp.next; // 삭제될 노드
    temp.next = temp.next.next; // temp의 링크를 삭제될 노드의 다음으로 바꾼다.
    T returnData = nodeToDelete.data; // 리턴될 데이터를 저장
    if(nodeToDelete == tail) // 만약 삭제할 노드가 tail이라면 tail을 바꿔준다.
        tail = temp;
    nodeToDelete = null; // 명시적인 삭제
    size--;
    return returnData;
}
```

- get() 메소드

```
@Override
public T get(int index) {
    if(index < 0 || index >= size)
        throw new IndexOutOfBoundsException();
    return node(index).data;
}
```

실습8-1. LinkedList 구현 실습(10)

- removeFirst(), removeLast(), remove(), get() 테스트 코드와 결과

```
public class LinkedListTest {
    public static void main(String[] args) {
        List<Integer> list = new LinkedList<>();
        for(int i = 0; i < 20; i++)
            list.addLast(i);
        System.out.println("리스트: " + list);
        System.out.println("removeFirst() 호출: "+ list.removeFirst());
        System.out.println("removeLast() 호출: " + list.removeLast());
        System.out.println("remove(4) 호출: " + list.remove(4));
        System.out.println("변경된 리스트: " + list);
        System.out.println("get(5) 호출: " + list.get(5));
    }
}
```

```
리스트: [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19]
removeFirst() 호출: 0
removeLast() 호출: 19
remove(4) 호출: 5
변경된 리스트: [1, 2, 3, 4, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18]
get(5) 호출: 7
```

과제8-1. indexOf(), set() 구현

- 실습에서 만들었던 LinkedList에는 indexOf()와 set()이 구현을 하지 않은 상태이다.
- indexOf()와 set()을 구현해보자.
  - LinkedList가 구현하고 있는 List 인터페이스의 indexOf와 Set은 아래 설명과 같다.

```
/**
 * element에 해당하는 값이 있으면
 * 그 값이 발견되는 첫번째 인덱스 값을 리턴
 * 없으면 -1을 리턴
 * @param element
 * @return element가 발견되는 첫번째 인덱스(없으면 -1을 리턴)
 */
public int indexOf(E element);
```

```
/**
 * 리스트의 index의 원소를 element로 변경.
 * @param index
 * @param element
 * @return 변경되기 전의 리스트의 원소
 */
public E set(int index, E element);
```

과제8-2. 다항식의 덧셈(1)

- 페이스북에 올라온 Term 클래스는 내부 필드로 지수(coef) 차수(exp)를 가지고 있으며 0 이상의 차수를 가진다.
- static final로 선언된 Term ZERO는 private 생성자로 객체를 생성하고 이는 뒤에 설명할 다항식의 끝부분을 나타낸다.

```
public class Term {
    private double coef;
    private int exp;
    public static final Term ZERO = new Term();
    private Term() {
        this.exp = -1;
    }

    public Term(double coef, int exp) {
        if(coef == 0.0 || exp < 0)
            throw new IllegalArgumentException();
        this.coef = coef;
        this.exp = exp;
    }
```

과제8-2. 다항식의 덧셈(2)

- Term 클래스는 같은 차수에 대하여 plus() 메소드 호출을 통해 덧셈 연산을 지원한다.

```
public static void main(String[] arg) {
    Term term1 = new Term(3,5);
    Term term2 = new Term(2,5);
    System.out.println("term1: " + term1);
    System.out.println("term2: " + term2);

    Term term3 = term1.plus(term2);
    System.out.println("term1 + term2: " + term3);
}
```

```
term1: 3.0x^5
term2: 2.0x^5
term1 + term2: 5.0x^5
```

과제8-2. 다항식의 덧셈(3)

- 다항식(Polynomial)은 Term들의 연결리스트로 표현할 수 있다.
  - 페이스북에 올라온 Polynomial 클래스는 내부적으로 실습 시간에 만들었던 LinkedList를 사용하고 plus() 메소드를 통해 지수의 내림차순으로 Term들을 연결하도록 만들어져 있다.

```
public static void main(String[] arg) {
    Polynomial f = new Polynomial(new Term(1, 1)); // 다항식 f를 생성
    f.plus(new Term(3,3)); // x Term에 3x^3 Term을 연결
    f.plus(new Term(2,2)); // 3x^3 + x에 2x^2 Term을 연결
    System.out.println("f(x): " + f);
}

f(x): 3.0x^3 +2.0x^2 +x
```

  - 만약 다항식에 이미 있는 차수의 Term을 추가하면 더해져서 다항식을 구성한다.

```
public static void main(String[] arg) {
    Polynomial f = new Polynomial(new Term(1, 1)); // 다항식 f를 생성
    f.plus(new Term(3,3)); // x Term에 3x^3 Term을 연결
    f.plus(new Term(2,2)); // 3x^3 + x에 2x^2 Term을 연결
    f.plus(new Term(-2,1)); // 다항식에 1차항 연결
    System.out.println("f(x): " + f);
}

f(x): 3.0x^3 +2.0x^2-x
```

과제8-2. 다항식의 덧셈(4)

- 다항식(Polynomial)은 Term들의 연결리스트로 표현할 수 있다.
  - 페이스북에 올라온 Polynomial 클래스는 내부적으로 실습 시간에 만들었던 LinkedList를 사용하고 plus() 메소드를 통해 지수의 내림차순으로 Term들을 연결하도록 만들어져 있다.

```
public static void main(String[] arg) {
    Polynomial f = new Polynomial(new Term(1, 1)); // 다항식 f를 생성
    f.plus(new Term(3,3)); // x Term에 3x^3 Term을 연결
    f.plus(new Term(2,2)); // 3x^3 + x에 2x^2 Term을 연결
    System.out.println("f(x): " + f);
}

f(x): 3.0x^3 +2.0x^2 +x
```

  - 만약 다항식에 이미 있는 차수의 Term을 추가하면 더해져서 다항식을 구성한다.

```
public static void main(String[] arg) {
    Polynomial f = new Polynomial(new Term(1, 1)); // 다항식 f를 생성
    f.plus(new Term(3,3)); // x Term에 3x^3 Term을 연결
    f.plus(new Term(2,2)); // 3x^3 + x에 2x^2 Term을 연결
    f.plus(new Term(-2,1)); // 다항식에 1차항 연결
    System.out.println("f(x): " + f);
}

f(x): 3.0x^3 +2.0x^2-x
```

과제8-2. 다항식의 덧셈(5)

- Polynomial 클래스의 polyPlus() 메소드를 구현해보자.

```
public Polynomial polyPlus(Polynomial otherPoly) {
    // polyPlus를 호출하고 있는 다항식과 인자로온 다른 다항식을 더한 새로운 다항식을 리턴
    return null;
}
```

- 테스트 코드와 결과는 다음과 같다.

```
public static void main(String[] arg) {
    Polynomial f = new Polynomial(new Term(1, 1));
    f.plus(new Term(3,3));
    f.plus(new Term(2,2));
    f.plus(new Term(7,0));

    Polynomial g = new Polynomial();
    g.plus(new Term(4,4));
    g.plus(new Term(5,5));
    g.plus(new Term(1,2));
    g.plus(new Term(2,3));

    System.out.println("f(x): " + f);
    System.out.println("g(x): " + g);

    Polynomial z = g.polyPlus(f);
    System.out.println("f(x) + g(x): " + z);
}
```

```
f(x): 3.0x^3 +2.0x^2 +x +7.0
g(x): 5.0x^5 +4.0x^4 +2.0x^3 +x^2
f(x) + g(x): 5.0x^5 +4.0x^4 +5.0x^3 +3.0x^2 +x +7.0
```

과제 제출 및 기한

- 제출 방법
  - 사이버캠퍼스를 통하여 제출
  - 소스코드를 제출
- 제출 기한
  - 이번 주 토요일(11/11) 자정