

컴프2 4주차 : 수신전환기능 구현

과제 목표 : 리스트를 활용하여 수신전환기능을 구현한다.

학번 : 201902722

이름 : 유형곤

큰 과제 : 수신전환 기능 구현하기

해결과정의 흐름 :

저는 수신전환 기능을 구현하기 위해서 1234->5678처럼 번호 쌍을 저장할 수 있는

Redirect 클래스를 작성했고, 이를 리스트에 저장해서 관리했습니다.

작은 과제 : Redirect 클래스 구현하기

```
private String from;
private String to;

public Redirect(String from, String to) {
    this.from = from;
    this.to = to;
}

public boolean isFrom(String from) {
    return this.from.equals(from);
}

public String getTo() {
    return to;
}

public String toString() {
    return String.format("%s->%s", from, to);
}
```

```

public static boolean isFrom(Object data, String from) {
    if(data instanceof Redirect) {
        Redirect redir = (Redirect) data;
        return redir.from.equals(from);
    } else {
        return false;
    }
}

public static String getTo(Object data) {
    if(data instanceof Redirect) {
        Redirect redir = (Redirect) data;
        return redir.to;
    } else {
        return null;
    }
}

```

Redirect 클래스는 단순히 수신전환을 위해 필요한 클래스로 출발지(from), 도착지(to)를 멤버변수로 합니다.

기본적인 메서드로는 생성자, toString 메서드를 구현해주었습니다.

Boolean isFrom(Object data, from) : 해당 Redirect 객체의 from이 매개변수의 from과 동일한지 확인합니다.

String getTo(Object data) : 해당 Redirect 객체의 출발지를 리턴합니다.

작은 과제 : MyList 클래스 구현하기

MyList에서는 노드를 표현하기 위해서 클래스 내부에 Node 클래스를 만들었습니다.

```

private class Node {
    private Object data;
    private Node next;
    public Node(Object data) {
        this.data = data;
        this.next = null;
    }
    public String toString() {
        return data.toString();
    }
}

```

Node는 자신의 데이터와, 다음 노드를 가리키는 변수인 data, next가 존재합니다.

여기서는 Single Linked List를 구현할 것이므로 prev는 없습니다.

```

public int size() {
    return size;
}

public void addFirst(Object data) {
    Node node = new Node(data);
    node.next = head;
    head = node;
    if(head.next == null) {
        tail = head;
    }
    size++;
}

```

Int size() : 리스트의 사이즈를 반환합니다.

addFirst(Object data) : 리스트의 맨 앞에 Node를 추가합니다. 새로운 노드의 next로 head를 가리키고, 새로운 노드를 head로 교체합니다.

```

public void addLast(Object data) {
    Node node = new Node(data);
    if(size == 0) {
        addFirst(data);
        return;
    }
    tail.next = node;
    tail = node;
    size++;
}

```

addLast(Object data) : 리스트의 맨 뒤에 노드를 추가합니다. tail의 next에 새로운 노드를 넣고 tail을 새로운 노드로 교체하면 됩니다.

```

public void add(int idx, Object data) {
    Node node = node(idx-1);
    Node newNode = new Node(data);
    newNode.next = node.next;
    node.next = newNode;
    size++;
}

```

add(int idx, Object data) : idx-1번째 요소를 불러와서, idx-1번째 요소의 next를 새로운 노드로, 새로운 노드의 next를 기존의 idx번째 요소와 연결하면 새로운 노드의 인덱스가 idx가 됩니다.

```

public void add(Object data) {
    addLast(data);
}

public Object removeFirst() {
    if(size == 0) {
        return null;
    }
    Object ret = head.data;
    head = head.next;
    size--;
    return ret;
}

```

add(Object data) : add함수에 인덱스가 주어지지 않은 경우 addLast를 호출하도록 함수를 오버라이딩합니다.

removeFirst() : 리스트의 첫 번째 요소를 제거합니다. Head의 데이터를 저장하고, head를 head.next로 교체한 후 저장한 데이터를 리턴합니다.

```
public Object remove(int idx) {
    if(size == 0) {
        return null;
    } else if(idx == 0) {
        return removeFirst();
    }
    Node node = node(idx-1);
    Node toRemove = node.next;
    node.next = toRemove.next;
    Object ret = toRemove.data;
    if(toRemove == tail) {
        tail = toRemove.next;
    }
    size--;
    return ret;
}
```

```
public Object removeLast() {
    return remove(size-1);
}
```

remove(int idx) : 인덱스가 idx인 노드를 삭제하는 함수로, 인덱스가 idx-1인 원소를 가져온 후, idx번째의 데이터를 저장하고 idx-1의 next를 idx의 next로 교체하면 됩니다.

removeLast() : remove(size-1)을 호출합니다.

```
public Node node(int k) {
    Node node = head;
    for(int i = 0; i < k; i++) {
        if(node.next == null) {
            throw new ListIndexException("리스트의 인덱스가 잘못되었습니다.");
        }
        node = node.next;
    }
    return node;
}
```

node(int k) : k번째 node를 반환합니다.

```

public String search(String from) {
    Node node = head;
    for(int i = 0; i < size; i++) {

        if(Redirect.isFrom(node.data, from)) {
            return Redirect.getTo(node.data);
        }
        node = node.next;
    }
    return null;
}

```

이번 과제를 위해 만든 함수로, 리스트의 node 중에서 from을 기준으로 Redirect 객체를 검색합니다.

```

public String toString() {
    StringBuilder sb = new StringBuilder();
    sb.append("[");
    Node node = head;
    for(int i = 0; i < size; i++) {
        sb.append(node);
        if(i != size-1) {
            sb.append(", ");
            node = node.next;
        }
    }
    sb.append("]");
    return sb.toString();
}

```

toString() : list를 문자열로 변환합니다. 형태 : [1, 2, 3, 4], 문자열 덧셈의 비효율성을 고려하여 StringBuilder를 사용하였습니다.

작은 과제 : Redirect 클래스와 리스트 (MyList) 클래스를 활용하여 문제해결하기

```
public static void main(String[] args) throws IOException {
    BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
    BufferedWriter bw = new BufferedWriter(new OutputStreamWriter(System.out));
    StringTokenizer st = new StringTokenizer(br.readLine());
    int n = Integer.parseInt(st.nextToken());
    String start = st.nextToken();
    MyList list = new MyList();

    for(int i = 0; i < n; i++) {
        st = new StringTokenizer(br.readLine());
        String from = st.nextToken();
        String to = st.nextToken();
        Redirect redir = new Redirect(from, to); //수신전화 리다이렉트
        list.add((Object)redir);
    }
}
```

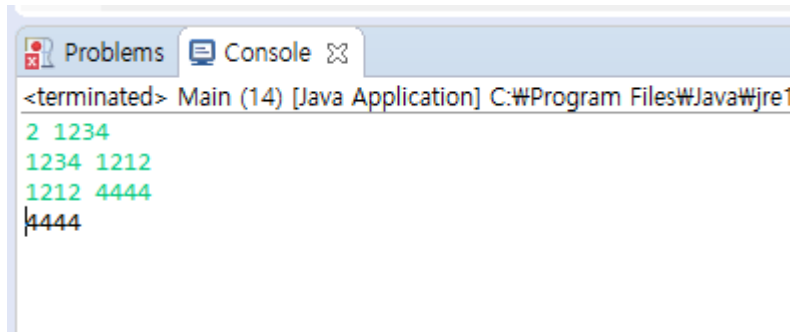
Main 함수의 초반 부분에서는, 입출력에 대한 변수를 초기화하고, 리스트와 전화가 시작되는 start (맨 처음 전화가 걸리는 번호) 변수가 선언 및 초기화됩니다. 그 후 반복문을 통해서 각각의 수신전환을 Redirect 클래스의 형태로 저장하여, 리스트에 담습니다.

```
String tmp = list.search(start); //start에 매칭되는 수신전화 목록을 찾음
String end;
boolean loop = false;
if(tmp == null) { //tmp가 null이라면 리다이렉트할 번호가 없다는 뜻
    end = start;
} else {
    end = tmp;
    tmp = list.search(end);
    while(!end.equals(start) && tmp != null) { //자기 자신이 나오거나 더이상 번호가 없을때까지 반복
        end = tmp;
        tmp = list.search(end);
    }
    if(end.equals(start)) { //자기 자신이 나온 경우 : 무한루프
        loop = true;
    }
}

bw.write((loop) ? "0000" : end + "\n");
br.close();
bw.close();
}
```

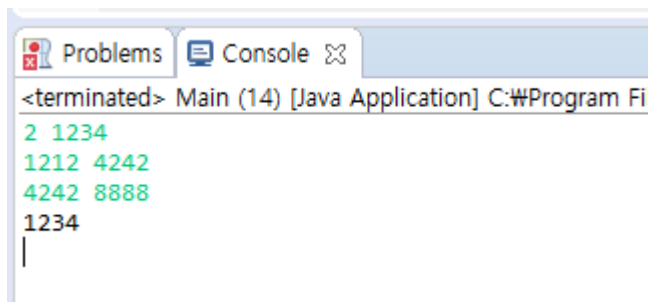
그 후, 리스트의 다음 요소가 존재하지 않거나 처음 걸린 전화번호가 나올 때까지 수신전환을 합니다. 무한루프가 걸린 경우에는 0000을 출력하고, 그게 아니라면 최종적으로 전화가 걸릴 번호를 출력합니다.

#실행결과



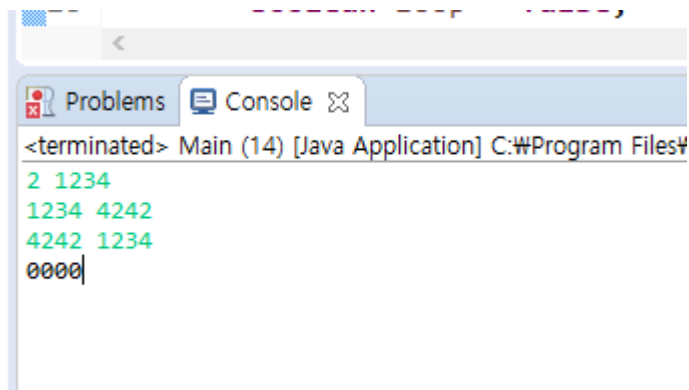
```
<terminated> Main (14) [Java Application] C:\Program Files\Java\jre1
2 1234
1234 1212
1212 4444
4444
```

<일반적인 경우>



```
<terminated> Main (14) [Java Application] C:\Program Fi
2 1234
1212 4242
4242 8888
1234
```

<처음 걸린 번호가 수신전환이 안 되어있는 경우>



```
<terminated> Main (14) [Java Application] C:\Program Files\
2 1234
1234 4242
4242 1234
0000
```

<사이클이 생긴 경우>