



JAVA 자료 구조

Linked List

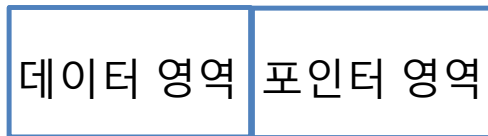
2. Linked List

연결리스트



노드

자료구조에 데이터를 담거나 표현하는 기초적인 단위



*Appendix 참조
자바에 포인터가 없는 이유

1. 크기를 미리 정하지 않고도 동적 할당이 가능하다
2. 삽입과 삭제 연산에서 오버헤드가 배열보다 적다
3. 특정 인덱스로 접근이 불가능하므로 데이터를 검색하려면 순차적으로 첫 노드부터 끝까지 방문해야한다.

기본 동작 : 삽입, 삭제, 검색

삽입, 삭제 연산이 자주 일어나는 환경에서 배열보다 유리한 자료구조

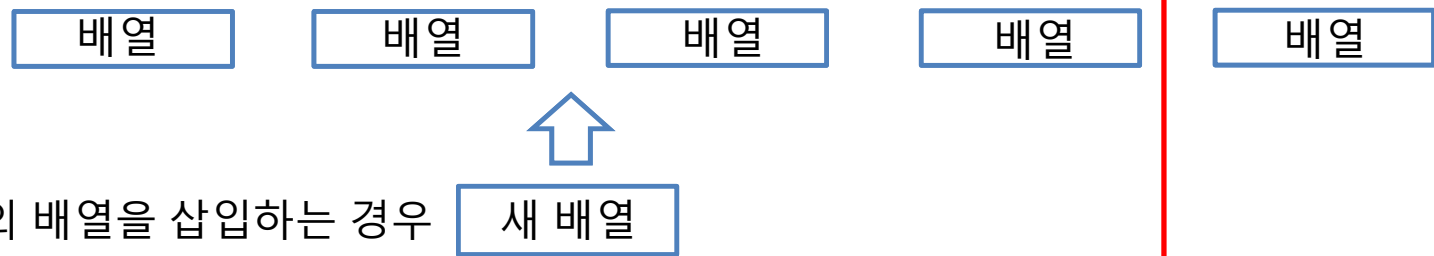
2. Linked List

연결리스트의 3가지 기본 동작

- 삽입

배열의 경우

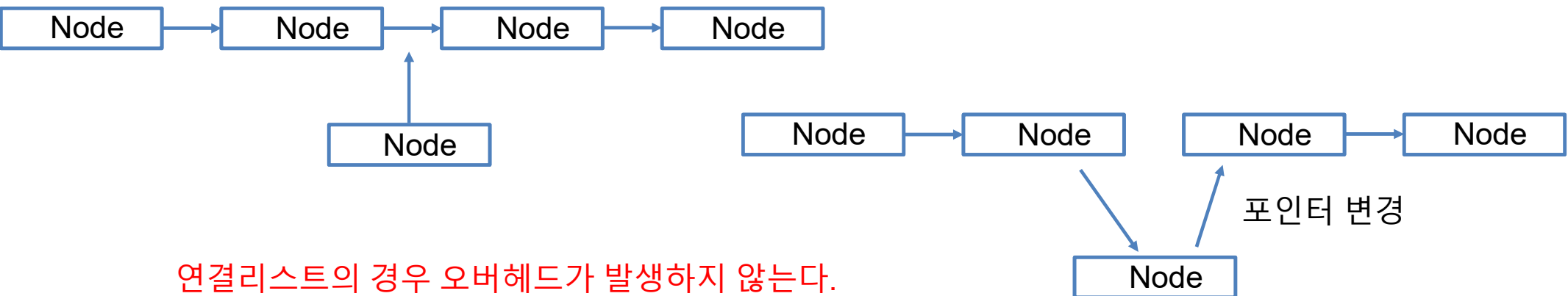
1. 하나의 배열을 삽입하는 경우



2. 한 칸씩 뒤로 밀리면서 **오버헤드** 발생

기존에 잡아놓은 메모리 영역

연결리스트의 경우

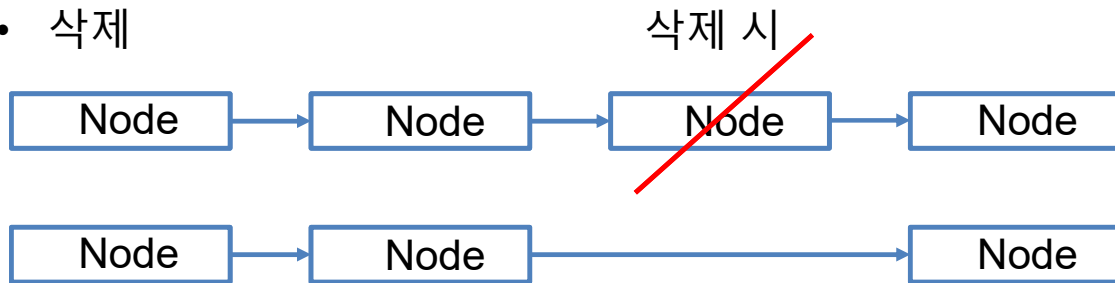


연결리스트의 경우 오버헤드가 발생하지 않는다.

2. Linked List

연결리스트의 3가지 기본 동작

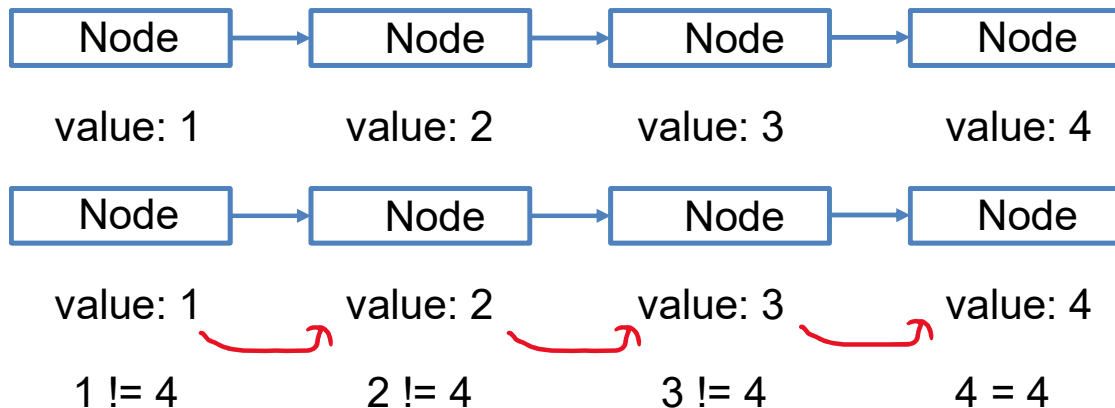
- 삭제



- 1) 포인터를 회수한다
- 2) 노드를 null로 초기화 한다.

Gabage Collection이 자동으로 메모리 회수

- 검색



value : 4 검색 시 찾을 때까지 모두 순회

배열의 경우 Time Complexity가 $O(1)$ 이나

연결리스트의 경우 $O(n)$ 을 가진다.

다음 케이스의 경우 $O(4)$ 의 시간복잡도를 가짐.

2. Linked List

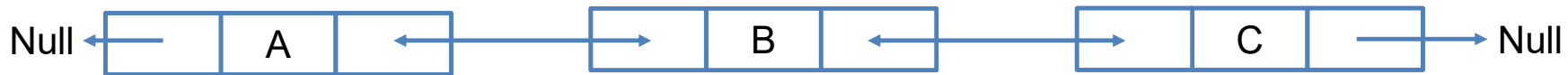
연결리스트의 3가지 종류

- 단일 연결 리스트



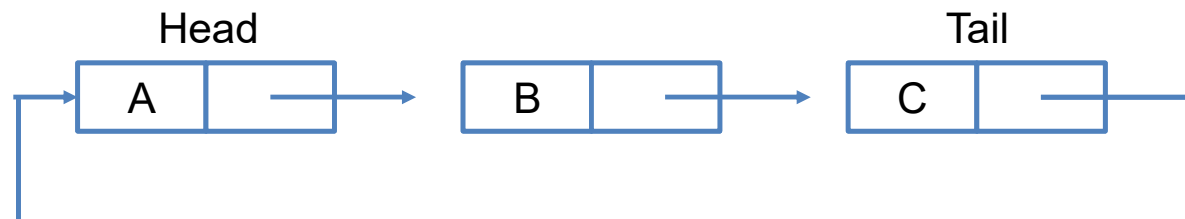
Head 존재, 각각의 노드는 다음 노드를 가리키고 마지막 노드의 포인터는 Null 값

- 이중 연결 리스트



인접한 노드는 모두 가리키며 포인터 영역이 하나의 노드당 두 개

- 원형 단일 연결 리스트



Head와 Tail 존재 마지막 노드의 포인터는 Head를 가리킨다.

2. Linked List

단일 연결 리스트 구현 : 삽입

```
1  package linkedList;  
2  
3  public class SLLInsertionMain {  
    Run | Debug  
4      public static void main(String[] args) {  
5          SinglyLinkedList singlyLinkedList = new SinglyLinkedList();  
6          singlyLinkedList.append(85);  
7          singlyLinkedList.append(59);  
8          singlyLinkedList.append(70);  
9          singlyLinkedList.append(23);  
10         singlyLinkedList.append(65);  
11  
12         System.out.print("단일 연결 리스트: ");  
13         singlyLinkedList.printAll();  
14     }  
15 }
```

85 – 59 – 70 – 23 – 65

2. Linked List

단일 연결 리스트 구현 : 삽입

```

1 package linkedList;
2
3 /**
4  * 단일 연결 리스트 클래스
5  */
6 public class SinglyLinkedList {
7     private Node head;
8
9     SinglyLinkedList() {
10         this.head = null;
11     }
12
13     SinglyLinkedList(int value) {
14         this.head = new Node(value, null);
15     }
16
17     class Node {
18         private int value; // 값
19         private Node next; // 포인터
20
21         Node(int value, Node next) {
22             this.value = value;
23             this.next = next;
24         }
25
26         public int getValue() {
27             return this.value;
28         }
29     }
30
31     public Node getHead() {
32         return this.head;
33     }
34
35     /**
36      * 단일 연결 리스트 끝에 노드를 추가
37      * @param value
38      */
39     public void append(int value) {
40         if (this.head == null) {
41             this.head = new Node(value, null);
42             return;
43         }
44
45         Node pointer = this.head;
46         while (pointer.next != null) {
47             pointer = pointer.next;
48         }
49
50         pointer.next = new Node(value, null);
51     }
52
53     /**
54      * 인자 값으로 같은 노드 삭제
55      * @param value
56      */
57     public void delete(int value) {
58         Node pointer = this.head;
59
60         // 첫 노드의 값이 인자 값과 같으면
61         if (pointer.getValue() == value) {
62             Node removeNode = this.head;
63             this.head = this.head.next;
64
65             removeNode = null;
66             return;
67         }
68
69         Node temp = pointer;
70         // 포인터가 null이 아니고 값이 다를 때까지 반복
71         while (pointer != null && pointer.getValue() != value) {
72             temp = pointer;
73             pointer = pointer.next;
74         }
75
76         // 마지막 노드를 삭제하는 경우
77         if (pointer.next == null) {
78             temp.next = null;
79         } else {
80             temp.next = pointer.next;
81         }
82         pointer = null;
83     }
84
85     /**
86      * 모든 노드 출력
87      */
88     public void printAll() {
89         Node pointer = this.head;
90
91         StringBuilder builder = new StringBuilder();
92         while (pointer.next != null) {
93             builder.append(pointer.getValue());
94             builder.append(" -> ");
95             pointer = pointer.next;
96         }
97
98         builder.append(pointer.getValue());
99         System.out.println(builder.toString());
100     }
101 }

```

2. Linked List

단일 연결 리스트 구현 : 삭제

```
1 package linkedList;
2
3 public class SLLDeletionMain {
4     public static void main(String[] args) {
5         SinglyLinkedList singlyLinkedList = new SinglyLinkedList();
6         singlyLinkedList.append(85);
7         singlyLinkedList.append(59);
8         singlyLinkedList.append(70);
9         singlyLinkedList.append(23);
10        singlyLinkedList.append(65);
11
12        System.out.print("원본 단일 연결 리스트: ");
13        singlyLinkedList.printAll();
14
15        singlyLinkedList.delete(85);
16        System.out.print("노드 85를 삭제한 단일 연결 리스트: ");
17        singlyLinkedList.printAll();
18
19        singlyLinkedList.delete(70);
20        System.out.print("노드 70을 삭제한 단일 연결 리스트: ");
21        singlyLinkedList.printAll();
22
23        singlyLinkedList.delete(65);
24        System.out.print("노드 65를 삭제한 단일 연결 리스트: ");
25        singlyLinkedList.printAll();
26    }
27 }
```


2. Linked List

단일 연결 리스트 구현 : 삭제

```

1 package linkedList;
2
3 /**
4  * 단일 연결 리스트 클래스
5  */
6 public class SinglyLinkedList {
7     private Node head;
8
9     SinglyLinkedList() {
10         this.head = null;
11     }
12
13     SinglyLinkedList(int value) {
14         this.head = new Node(value, null);
15     }
16
17     class Node {
18         private int value; // 값
19         private Node next; // 포인터
20
21         Node(int value, Node next) {
22             this.value = value;
23             this.next = next;
24         }
25
26         public int getValue() {
27             return this.value;
28         }
29     }
30
31     public Node getHead() {
32         return this.head;
33     }
34
35     /**
36      * 단일 연결 리스트 끝에 노드를 추가
37      * @param value
38      */
39     public void append(int value) {
40         if (this.head == null) {
41             this.head = new Node(value, null);
42             return;
43         }
44
45         Node pointer = this.head;
46         while (pointer.next != null) {
47             pointer = pointer.next;
48         }
49
50         pointer.next = new Node(value, null);
51     }
52
53     /**
54      * 인자 값으로 같은 노드 삭제
55      * @param value
56      */
57     public void delete(int value) {
58         Node pointer = this.head;
59
60         // 첫 노드의 값이 인자 값과 같으면
61         if (pointer.getValue() == value) {
62             Node removeNode = this.head;
63             this.head = this.head.next;
64
65             removeNode = null;
66             return;
67         }
68
69         Node temp = pointer;
70         // 포인터가 null이 아니고 값이 다를 때까지 반복
71         while (pointer != null && pointer.getValue() != value) {
72             temp = pointer;
73             pointer = pointer.next;
74         }
75
76         // 마지막 노드를 삭제하는 경우
77         if (pointer.next == null) {
78             temp.next = null;
79         } else {
80             temp.next = pointer.next;
81         }
82         pointer = null;
83     }
84
85     /**
86      * 모든 노드 출력
87      */
88     public void printAll() {
89         Node pointer = this.head;
90
91         StringBuilder builder = new StringBuilder();
92         while (pointer.next != null) {
93             builder.append(pointer.getValue());
94             builder.append(" -> ");
95             pointer = pointer.next;
96         }
97
98         builder.append(pointer.getValue());
99         System.out.println(builder.toString());
100     }
101 }

```

2. Linked List

이중 연결 리스트 구현 : 삽입

```
1 package linkedList;  
2  
3 public class DLLInsertionMain {  
    Run | Debug  
4     public static void main(String[] args) {  
5         DoubleLinkedList doubleLinkedList = new DoubleLinkedList();  
6         doubleLinkedList.append(1);  
7         doubleLinkedList.append(3);  
8         doubleLinkedList.append(2);  
9         doubleLinkedList.append(20);  
10        doubleLinkedList.append(19);  
11  
12        System.out.print("이중 연결 리스트: ");  
13        doubleLinkedList.printAll();  
14  
15        doubleLinkedList.printPrevNode(2);  
16        doubleLinkedList.printPrevNode(1);  
17        doubleLinkedList.printPrevNode(19);  
18    }  
19 }
```

1 - 3 - 2 - 20 - 35

2. Linked List

이중 연결 리스트 구현 : 삽입

```

1  package linkedList;
2
3  /**
4   * 이중 연결 리스트 클래스
5   */
6  public class DoubleLinkedList {
7      private Node head;
8
9      DoubleLinkedList() {
10         this.head = null;
11     }
12
13     DoubleLinkedList(int value, Node head) {
14         this.head = new Node(value, head, null);
15     }
16
17     class Node {
18         private int value; // 값
19         private Node prev; // 이전 포인터
20         private Node next; // 다음 포인터
21
22         Node(int value) {
23             this.value = value;
24             this.prev = null;
25             this.next = null;
26         }
27
28         Node(int value, Node prev, Node next) {
29             this.value = value;
30             this.prev = prev;
31             this.next = next;
32         }
33
34         public int getValue() {
35             return this.value;
36         }
37     }
38
39     public Node getHead() {
40         return this.head;
41     }
42
43     /**
44      * 이중 연결리스트 끝에 노드 추가
45      * @param value
46      */
47     public void append(int value) {
48         if (this.head == null) {
49             this.head = new Node(value);
50             return;
51         }
52
53         Node pointer = this.head;
54         while(pointer.next != null) {
55             pointer = pointer.next;
56         }
57
58         Node newNode = new Node(value);
59         newNode.prev = pointer;
60         pointer.next = newNode;
61     }
62
63     /**
64      * 노드의 값으로 삭제
65      * @param value
66      */
67     public void delete(int value) {
68         Node pointer = this.head;
69
70         if (pointer.getValue() == value) {
71             Node removeNode = this.head;
72             this.head = this.head.next;
73
74             removeNode = null;
75         }
76     }
77
78     Node prevNode = pointer;
79     while (pointer != null && pointer.getValue() != value) {
80         prevNode = pointer;
81         pointer = pointer.next;
82     }
83
84     Node temp = pointer.next;
85     if (temp == null) {
86         prevNode.next = null;
87     } else {
88         temp.prev = prevNode;
89         prevNode.next = pointer.next;
90     }
91
92     pointer = null;
93 }
94
95 /**
96  * 인자 값으로 앞 노드의 값을 알아내기
97  * @param value
98  * @return
99  */
100 public void printPrevNode(int value) {
101     if (this.head == null) {
102         System.out.println("이중 연결 리스트가 비어 있습니다.");
103         return;
104     }
105
106     if (this.head.getValue() == value) {
107         System.out.println(String.format("노드 %s 의 앞 노드는 존재 하지 않습니다.", value));
108         return;
109     }
110 }

```

2. Linked List

이중 연결 리스트 구현 : 삽입

```
111 Node pointer = this.head;
112 while (pointer != null && pointer.getValue() != value) {
113     pointer = pointer.next;
114 }
115
116 if (pointer == null) {
117     System.out.println(String.format("노드 %s 은 존재하지 않습니다.", value));
118 } else {
119     System.out.println(String.format("노드 %s 의 앞 노드의 값은 %s 입니다.", value, pointer.prev.getValue()));
120 }
121 }
122
123 /**
124  * 모든 노드 출력
125  */
126 public void printAll() {
127     Node pointer = this.head;
128
129     StringBuilder builder = new StringBuilder();
130     while (pointer.next != null) {
131         builder.append(pointer.getValue());
132         builder.append(" <-> ");
133         pointer = pointer.next;
134     }
135
136     builder.append(pointer.getValue());
137     System.out.println(builder.toString());
138 }
139 }
```


2. Linked List

이중 연결 리스트 구현 : 삭제

```
1 package linkedList;
2
3 public class DLLDeletionMain {
4     Run | Debug
5     public static void main(String[] args) {
6         DoubleLinkedList doubleLinkedList = new DoubleLinkedList()
7         doubleLinkedList.append(1);
8         doubleLinkedList.append(3);
9         doubleLinkedList.append(2);
10        doubleLinkedList.append(20);
11        doubleLinkedList.append(19);
12
13        System.out.print("원본 이중 연결 리스트: ");
14        doubleLinkedList.printAll();
15
16        doubleLinkedList.delete(1);
17        System.out.print("노드 1을 삭제한 이중 연결 리스트: ");
18        doubleLinkedList.printAll();
19
20        doubleLinkedList.delete(2);
21        System.out.print("노드 2를 삭제한 이중 연결 리스트: ");
22        doubleLinkedList.printAll();
23
24        doubleLinkedList.delete(19);
25        System.out.print("노드 19를 삭제한 이중 연결 리스트: ");
26        doubleLinkedList.printAll();
27    }
28 }
```


2. Linked List

이중 연결 리스트 구현 : 삭제

```

1 package linkedList;
2
3 /**
4  * 이중 연결 리스트 클래스
5  */
6 public class DoubleLinkedList {
7     private Node head;
8
9     DoubleLinkedList() {
10         this.head = null;
11     }
12
13     DoubleLinkedList(int value, Node head) {
14         this.head = new Node(value, head, null);
15     }
16
17     class Node {
18         private int value; // 값
19         private Node prev; // 이전 포인터
20         private Node next; // 다음 포인터
21
22         Node(int value) {
23             this.value = value;
24             this.prev = null;
25             this.next = null;
26         }
27
28         Node(int value, Node prev, Node next) {
29             this.value = value;
30             this.prev = prev;
31             this.next = next;
32         }
33
34         public int getValue() {
35             return this.value;
36         }
37     }
38
39     public Node getHead() {
40         return this.head;
41     }
42
43     /**
44      * 이중 연결리스트 끝에 노드 추가
45      * @param value
46      */
47     public void append(int value) {
48         if (this.head == null) {
49             this.head = new Node(value);
50             return;
51         }
52
53         Node pointer = this.head;
54         while(pointer.next != null) {
55             pointer = pointer.next;
56         }
57
58         Node newNode = new Node(value);
59         newNode.prev = pointer;
60         pointer.next = newNode;
61     }
62
63     /**
64      * 노드의 값으로 삭제
65      * @param value
66      */
67     public void delete(int value) {
68         Node pointer = this.head;
69
70         if (pointer.getValue() == value) {
71             Node removeNode = this.head;
72             this.head = this.head.next;
73
74             removeNode = null;
75         }
76
77         Node prevNode = pointer;
78         while (pointer != null && pointer.getValue() != value) {
79             prevNode = pointer;
80             pointer = pointer.next;
81         }
82
83         Node temp = pointer.next;
84         if (temp == null) {
85             prevNode.next = null;
86         } else {
87             temp.prev = prevNode;
88             prevNode.next = pointer.next;
89         }
90
91         pointer = null;
92     }
93
94     /**
95      * 인자 값으로 앞 노드의 값을 알아내기
96      * @param value
97      * @return
98      */
99
100     public void printPrevNode(int value) {
101         if (this.head == null) {
102             System.out.println("이중 연결 리스트가 비어 있습니다.");
103             return;
104         }
105
106         if (this.head.getValue() == value) {
107             System.out.println(String.format("노드 %s 의 앞 노드는 존재 하지 않습니다.", value));
108             return;
109         }
110     }

```

2. Linked List

이중 연결 리스트 구현 : 삭제

```
111     Node pointer = this.head;
112     while (pointer != null && pointer.getValue() != value) {
113         pointer = pointer.next;
114     }
115
116     if (pointer == null) {
117         System.out.println(String.format("노드 %s 은 존재하지 않습니다.", value));
118     } else {
119         System.out.println(String.format("노드 %s 의 앞 노드의 값은 %s 입니다.", value, pointer.prev.getValue()));
120     }
121 }
122
123 /**
124  * 모든 노드 출력
125  */
126 public void printAll() {
127     Node pointer = this.head;
128
129     StringBuilder builder = new StringBuilder();
130     while (pointer.next != null) {
131         builder.append(pointer.getValue());
132         builder.append(" <-> ");
133         pointer = pointer.next;
134     }
135
136     builder.append(pointer.getValue());
137     System.out.println(builder.toString());
138 }
139 }
```

2. Linked List

원형 단일 연결 리스트 구현: 삽입

```
1  package linkedlist;  
2  
3  public class CLLInsertionMain {  
    Run | Debug  
4      public static void main(String[] args) {  
5          CircularLinkedList circularLinkedList = new CircularLinkedList();  
6          circularLinkedList.append(11);  
7          circularLinkedList.append(9);  
8          circularLinkedList.append(27);  
9          circularLinkedList.append(41);  
10         circularLinkedList.append(3);  
11  
12         System.out.print("원형 단일 연결 리스트: ");  
13         circularLinkedList.printAll();  
14     }  
15 }
```

11 – 9 – 27 – 41 – 3

2. Linked List

원형 단일 연결 리스트 구현: 삽입

```

1  package linkedList;
2
3  /**
4   * 원형 단일 연결 리스트 클래스
5   */
6  public class CircularLinkedList {
7      private Node head;
8      private Node tail;
9
10     CircularLinkedList() {
11         this.head = null;
12         this.tail = null;
13     }
14
15     class Node {
16         private int value; // 값
17         private Node next; // 다음 포인터
18
19         Node(int value) {
20             this.value = value;
21             this.next = null;
22         }
23
24         public int getValue() {
25             return this.value;
26         }
27     }
28
29     /**
30      * 원형 단일 연결 리스트 끝에 노드 추가
31      * @param value
32      */
33     public void append(int value) {
34         // 첫 노드 삽입 일 때
35         if (this.head == null && this.tail == null) {
36             Node node = new Node(value);
37             this.head = node;
38             this.tail = node;
39             return;
40         }
41
42         Node pointer = this.tail;
43         // 마지막 위치에 새 노드 삽입
44         pointer.next = new Node(value);
45
46         // 새로 삽입한 노드를 tail로 재할당
47         this.tail = pointer.next;
48         // tail의 next 포인터를 head로 연결하여 원형 연결 리스트 조건 만족시킴
49         this.tail.next = head;
50     }
51
52     /**
53      * 모든 노드 출력
54      */
55     public void printAll() {
56         Node pointer = this.head;
57
58         StringBuilder builder = new StringBuilder();
59         while (pointer != this.tail) {
60             builder.append(pointer.getValue());
61             builder.append(" -> ");
62             pointer = pointer.next;
63         }
64         builder.append(pointer.getValue());
65         builder.append("(tail) -> ");
66
67         builder.append(this.head.getValue());
68         builder.append("(head)");
69
70         System.out.println(builder.toString());
71     }
72 }

```


2. Linked List

원형 단일 연결 리스트 구현: 삭제

```
1 package linkedList;  
2  
3 public class CLLDeletionMain {  
4     Run | Debug  
5     public static void main(String[] args) {  
6         CircularLinkedList circularLinkedList = new CircularLinkedList  
7         circularLinkedList.append(11);  
8         circularLinkedList.append(9);  
9         circularLinkedList.append(27);  
10        circularLinkedList.append(41);  
11        circularLinkedList.append(3);  
12  
13        System.out.println("원본 원형 단일 연결 리스트: ");  
14        circularLinkedList.printAll();  
15  
16        System.out.println("노드 3을 삭제한 원형 단일 연결 리스트: ");  
17        circularLinkedList.delete(3);  
18        circularLinkedList.printAll();  
19  
20        System.out.println("노드 11를 삭제한 원형 단일 연결 리스트: ");  
21        circularLinkedList.delete(11);  
22        circularLinkedList.printAll();  
23  
24        System.out.println("노드 27를 삭제한 원형 단일 연결 리스트: ");  
25        circularLinkedList.delete(27);  
26        circularLinkedList.printAll();  
27    }
```


2. Linked List

원형 단일 연결 리스트 구현: 삭제

```

1  package linkedList;
2
3  /**
4   * 원형 단일 연결 리스트 클래스
5   */
6  public class CircularLinkedList {
7      private Node head;
8      private Node tail;
9
10     CircularLinkedList() {
11         this.head = null;
12         this.tail = null;
13     }
14
15     class Node {
16         private int value; // 값
17         private Node next; // 다음 포인터
18
19         Node(int value) {
20             this.value = value;
21             this.next = null;
22         }
23
24         public int getValue() {
25             return this.value;
26         }
27     }
28
29     /**
30     * 원형 단일 연결 리스트 끝에 노드 추가
31     * @param value
32     */
33
34     /**
35     * 원형 단일 연결 리스트 노드 삭제
36     * @param value
37     */
38     public void delete(int value) {
39         Node pointer = this.head;
40
41         if (pointer.getValue() == value) {
42             Node removeNode = this.head;
43             this.head = this.head.next;
44
45             removeNode = null;
46             return;
47         }
48
49         // 포인터가 tail이 아니며 값이 다를 때까지 반복
50         Node temp = null;
51         while (pointer.next != this.tail && pointer.getValue() != value) {
52             temp = pointer;
53             pointer = pointer.next;
54         }
55
56         // 삭제 할 노드가 tail일 경우 앞 노드를 tail로 재할당하고 next를 head로 연결
57         if (pointer.next.getValue() == value) {
58             this.tail = pointer;
59             this.tail.next = this.head;
60         } else {
61             temp.next = pointer.next;
62         }
63         pointer = null;
64     }
65
66     /**
67     * 모든 노드 출력
68     */
69     public void printAll() {
70         Node pointer = this.head;
71
72         StringBuilder builder = new StringBuilder();
73         while (pointer != this.tail) {
74             builder.append(pointer.getValue());
75             builder.append(" -> ");
76             pointer = pointer.next;
77         }
78         builder.append(pointer.getValue());
79         builder.append("(tail -> ");
80
81         builder.append(this.head.getValue());
82         builder.append("(head)");
83
84         System.out.println(builder.toString());
85     }
86 }

```

2. Linked List

ex) 3_1

단일 연결 리스트 삽입 메서드에서 중복을 허용하지 않는 삽입 메서드로 수정하라

2. Linked List

ex) 3_2

앞의 예제를 참고하여 원형 이중 연결리스트 및 아래 세 가지 동작을 구현하라

- ✓ 삽입
- ✓ 삭제
- ✓ 검색

2. Linked List

Homework

백준 사이트 3045, 17872 풀기.

3045. 이중 연결리스트

17872. 달팽이 리스트