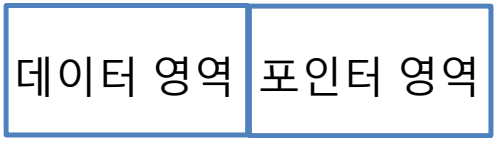

JAVA
자 료 구 조
Linked List

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

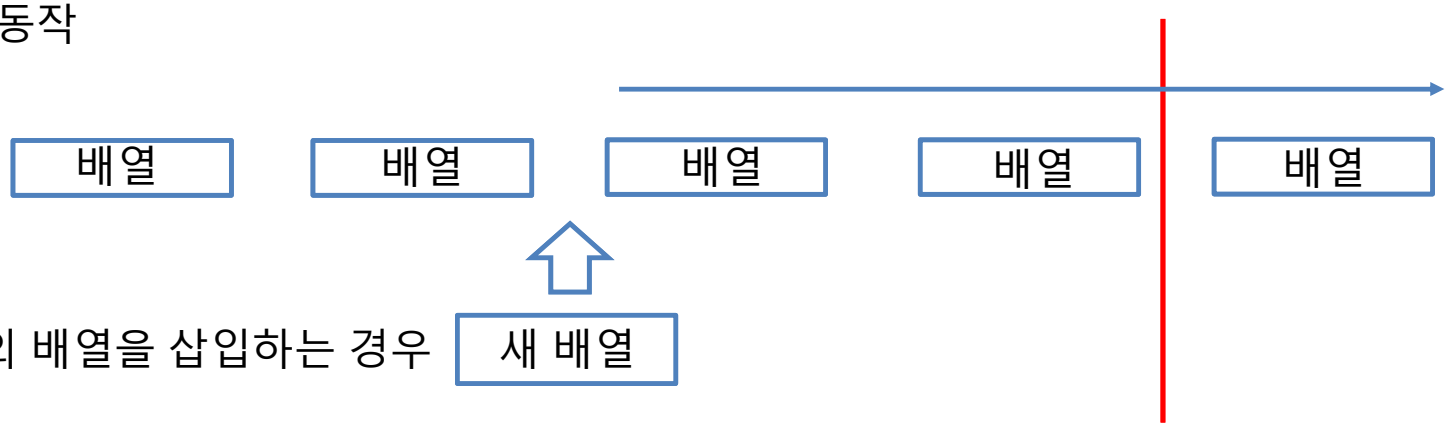
삽입, 삭제 연산이 자주 일어나는 환경에서 배열보다 유리한 자료구조

2. Linked List

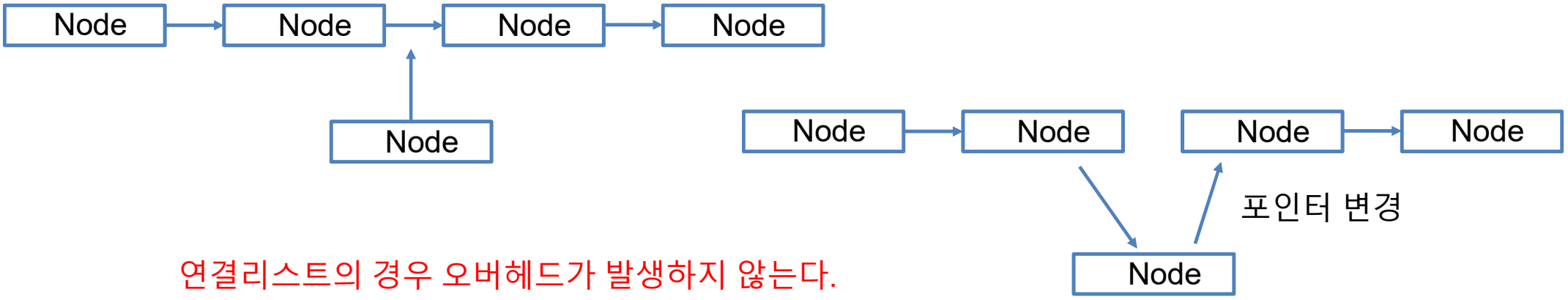
연결리스트의 3가지 기본 동작

- 삽입

배열의 경우



연결리스트의 경우

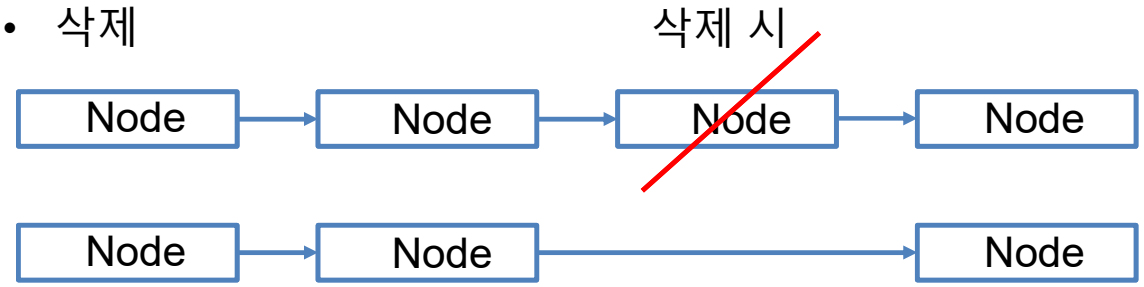


연결리스트의 경우 오버헤드가 발생하지 않는다.

2. Linked List

연결리스트의 3가지 기본 동작

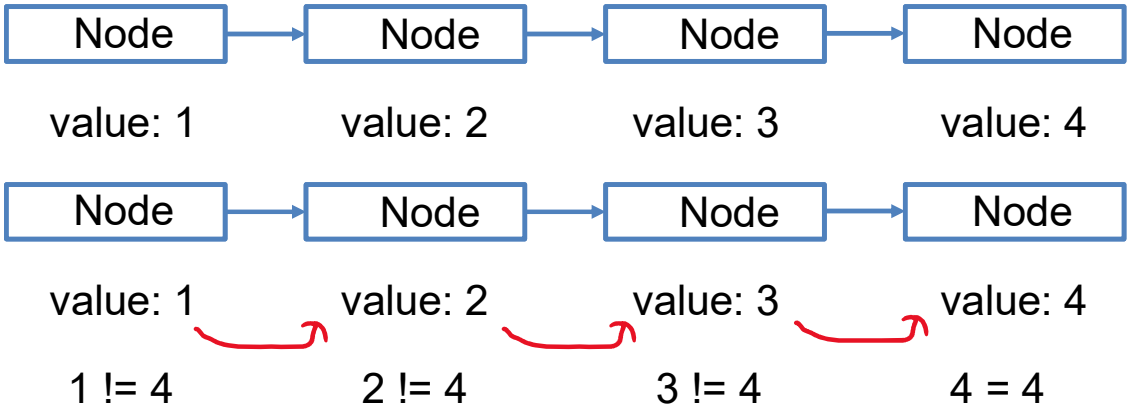
- 삭제



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

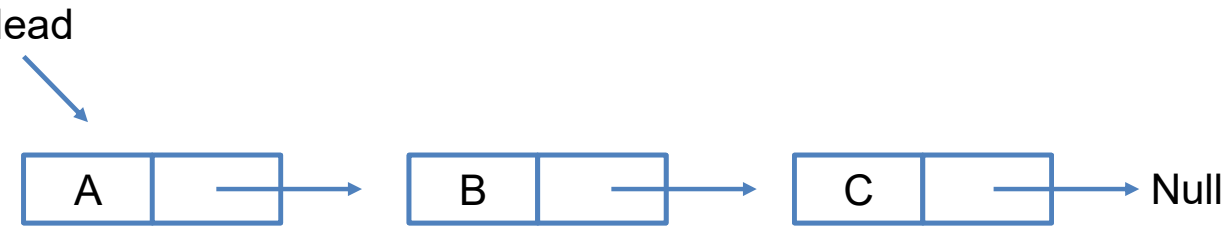
연결리스트의 경우 O(n)을 가진다.

다음 케이스의 경우 O(4)의 시간복잡도를 가짐.

2. Linked List

연결리스트의 3가지 종류

- **단일 연결 리스트**

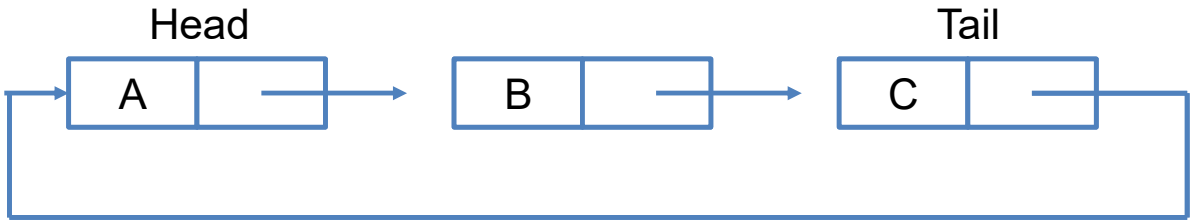


Head 존재, 각각의 노드는 다음 노드를 가리키고 마지막 노드의 포인터는 Null 값

- **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

```java
package linkedList;

public class SLLInsertionMain {
    Run | Debug
    public static void main(String[] args) {
        SinglyLinkedList singlyLinkedList = new SinglyLinkedList();
        singlyLinkedList.append(85);
        singlyLinkedList.append(59);
        singlyLinkedList.append(70);
        singlyLinkedList.append(23);
        singlyLinkedList.append(65);

        System.out.print("단일 연결 리스트: ");
        singlyLinkedList.printAll();
    }
}
```

85 – 59 – 70 – 23 – 65

2. Linked List

단일 연결 리스트 구현 : 삽입

```java
package linkedList;

/**
 * 단일 연결 리스트 클래스
 */
public class SinglyLinkedList {
    private Node head;

    SinglyLinkedList() {
        this.head = null;
    }

    SinglyLinkedList(int value) {
        this.head = new Node(value, null);
    }

    class Node {
        private int value; // 값
        private Node next; // 포인터

        Node(int value, Node next) {
            this.value = value;
            this.next = next;
        }

        public int getValue() {
            return this.value;
        }
    }

    public Node getHead() {
        return this.head;
    }
```

```java
    /**
     * 단일 연결 리스트 끝에 노드를 추가
     * @param value
     */
    public void append(int value) {
        if (this.head == null) {
            this.head = new Node(value, null);
            return;
        }

        Node pointer = this.head;
        while (pointer.next != null) {
            pointer = pointer.next;
        }

        pointer.next = new Node(value, null);
    }

    /**
     * 인자 값으로 같은 노드 삭제
     * @param value
     */
    public void delete(int value) {
        Node pointer = this.head;

        // 첫 노드의 값이 인자 값과 같으면
        if (pointer.getValue() == value) {
            Node removeNode = this.head;
            this.head = this.head.next;

            removeNode = null;
            return;
        }
```

```java
        Node temp = pointer;
        // 포인터가 null이 아니고 값이 다를 때까지 반복
        while (pointer != null && pointer.getValue() != value) {
            temp = pointer;
            pointer = pointer.next;
        }

        // 마지막 노드를 삭제하는 경우
        if (pointer.next == null) {
            temp.next = null;
        } else {
            temp.next = pointer.next;
        }
        pointer = null;
    }

    /**
     * 모든 노드 출력
     */
    public void printAll() {
        Node pointer = this.head;

        StringBuilder builder = new StringBuilder();
        while (pointer.next != null) {
            builder.append(pointer.getValue());
            builder.append(" -> ");
            pointer = pointer.next;
        }

        builder.append(pointer.getValue());
        System.out.println(builder.toString());
    }
}
```

2. Linked List

단일 연결 리스트 구현 : 삭제

```java
package linkedList;

public class SLLDeletionMain {
    Run | Debug
    public static void main(String[] args) {
        SinglyLinkedList singlyLinkedList = new SinglyLinkedList();
        singlyLinkedList.append(85);
        singlyLinkedList.append(59);
        singlyLinkedList.append(70);
        singlyLinkedList.append(23);
        singlyLinkedList.append(65);

        System.out.print("원본 단일 연결 리스트: ");
        singlyLinkedList.printAll();

        singlyLinkedList.delete(85);
        System.out.print("노드 85를 삭제한 단일 연결 리스트: ");
        singlyLinkedList.printAll();

        singlyLinkedList.delete(70);
        System.out.print("노드 70을 삭제한 단일 연결 리스트: ");
        singlyLinkedList.printAll();

        singlyLinkedList.delete(65);
        System.out.print("노드 65를 삭제한 단일 연결 리스트: ");
        singlyLinkedList.printAll();
    }
}
```

2. Linked List

단일 연결 리스트 구현 : 삭제

```java
package linkedList;

/**
 * 단일 연결 리스트 클래스
 */
public class SinglyLinkedList {
    private Node head;

    SinglyLinkedList() {
        this.head = null;
    }

    SinglyLinkedList(int value) {
        this.head = new Node(value, null);
    }

    class Node {
        private int value; // 값
        private Node next; // 포인터

        Node(int value, Node next) {
            this.value = value;
            this.next = next;
        }

        public int getValue() {
            return this.value;
        }
    }

    public Node getHead() {
        return this.head;
    }

    /**
     * 단일 연결 리스트 끝에 노드를 추가
     * @param value
     */
    public void append(int value) {
        if (this.head == null) {
            this.head = new Node(value, null);
            return;
        }

        Node pointer = this.head;
        while (pointer.next != null) {
            pointer = pointer.next;
        }

        pointer.next = new Node(value, null);
    }

    /**
     * 인자 값으로 같은 노드 삭제
     * @param value
     */
    public void delete(int value) {
        Node pointer = this.head;

        // 첫 노드의 값이 인자 값과 같으면
        if (pointer.getValue() == value) {
            Node removeNode = this.head;
            this.head = this.head.next;

            removeNode = null;
            return;
        }

        Node temp = pointer;
        // 포인터가 null이 아니고 값이 다를 때까지 반복
        while (pointer != null && pointer.getValue() != value) {
            temp = pointer;
            pointer = pointer.next;
        }

        // 마지막 노드를 삭제하는 경우
        if (pointer.next == null) {
            temp.next = null;
        } else {
            temp.next = pointer.next;
        }
        pointer = null;
    }

    /**
     * 모든 노드 출력
     */
    public void printAll() {
        Node pointer = this.head;

        StringBuilder builder = new StringBuilder();
        while (pointer.next != null) {
            builder.append(pointer.getValue());
            builder.append(" -> ");
            pointer = pointer.next;
        }

        builder.append(pointer.getValue());
        System.out.println(builder.toString());
    }
}
```

2. Linked List

이중 연결 리스트 구현 : 삽입

```java
package linkedList;

public class DLLInsertionMain {
    Run | Debug
    public static void main(String[] args) {
        DoubleLinkedList doubleLinkedList = new DoubleLinkedList();
        doubleLinkedList.append(1);
        doubleLinkedList.append(3);
        doubleLinkedList.append(2);
        doubleLinkedList.append(20);
        doubleLinkedList.append(19);

        System.out.print("이중 연결 리스트: ");
        doubleLinkedList.printAll();

        doubleLinkedList.printPrevNode(2);
        doubleLinkedList.printPrevNode(1);
        doubleLinkedList.printPrevNode(19);
    }
}
```

1 – 3 – 2 – 20 – 35

2. Linked List

이중 연결 리스트 구현 : 삽입

```java
package linkedList;

/**
 * 이중 연결 리스트 클래스
 */
public class DoubleLinkedList {
    private Node head;

    DoubleLinkedList() {
        this.head = null;
    }

    DoubleLinkedList(int value, Node head) {
        this.head = new Node(value, head, null);
    }

    class Node {
        private int value; // 값
        private Node prev; // 이전 포인터
        private Node next; // 다음 포인터

        Node(int value) {
            this.value = value;
            this.prev = null;
            this.next = null;
        }

        Node(int value, Node prev, Node next) {
            this.value = value;
            this.prev = prev;
            this.next = next;
        }

        public int getValue() {
            return this.value;
        }
    }

    public Node getHead() {
        return this.head;
    }

    /**
     * 이중 연결리스트 끝에 노드 추가
     * @param value
     */
    public void append(int value) {
        if (this.head == null) {
            this.head = new Node(value);
            return;
        }

        Node pointer = this.head;
        while(pointer.next != null) {
            pointer = pointer.next;
        }

        Node newNode = new Node(value);
        newNode.prev = pointer;
        pointer.next = newNode;
    }

    /**
     * 노드의 값으로 삭제
     * @param value
     */
    public void delete(int value) {
        Node pointer = this.head;

        if (pointer.getValue() == value) {
            Node removeNode = this.head;
            this.head = this.head.next;

            removeNode = null;
            return;
        }

        Node prevNode = pointer;
        while (pointer != null && pointer.getValue() != value) {
            prevNode = pointer;
            pointer = pointer.next;
        }

        Node temp = pointer.next;
        if (temp == null) {
            prevNode.next = null;
        } else {
            temp.prev = prevNode;
            prevNode.next = pointer.next;
        }

        pointer = null;
    }

    /**
     * 인자 값으로 앞 노드의 값을 알아내기
     * @param value
     * @return
     */
    public void printPrevNode(int value) {
        if (this.head == null) {
            System.out.println("이중 연결 리스트가 비어 있습니다.");
            return;
        }

        if (this.head.getValue() == value) {
            System.out.println(String.format("노드 %s 의 앞 노드는 존재 하지 않습니다.", value));
            return;
        }
```


다음 페이지까지 한 코드

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

```java
package linkedList;

public class DLLDeletionMain {
    Run | Debug
    public static void main(String[] args) {
        DoubleLinkedList doubleLinkedList = new DoubleLinkedList()
        doubleLinkedList.append(1);
        doubleLinkedList.append(3);
        doubleLinkedList.append(2);
        doubleLinkedList.append(20);
        doubleLinkedList.append(19);

        System.out.print("원본 이중 연결 리스트: ");
        doubleLinkedList.printAll();

        doubleLinkedList.delete(1);
        System.out.print("노드 1을 삭제한 이중 연결 리스트: ");
        doubleLinkedList.printAll();

        doubleLinkedList.delete(2);
        System.out.print("노드 2를 삭제한 이중 연결 리스트: ");
        doubleLinkedList.printAll();

        doubleLinkedList.delete(19);
        System.out.print("노드 19를 삭제한 이중 연결 리스트: ");
        doubleLinkedList.printAll();
    }
}
```

2. Linked List

이중 연결 리스트 구현 : 삭제

```java
package linkedList;

/**
 * 이중 연결 리스트 클래스
 */
public class DoubleLinkedList {
    private Node head;

    DoubleLinkedList() {
        this.head = null;
    }

    DoubleLinkedList(int value, Node head) {
        this.head = new Node(value, head, null);
    }

    class Node {
        private int value; // 값
        private Node prev; // 이전 포인터
        private Node next; // 다음 포인터

        Node(int value) {
            this.value = value;
            this.prev = null;
            this.next = null;
        }

        Node(int value, Node prev, Node next) {
            this.value = value;
            this.prev = prev;
            this.next = next;
        }

        public int getValue() {
            return this.value;
        }
    }

    public Node getHead() {
        return this.head;
    }

    /**
     * 이중 연결리스트 끝에 노드 추가
     * @param value
     */
    public void append(int value) {
        if (this.head == null) {
            this.head = new Node(value);
            return;
        }

        Node pointer = this.head;
        while(pointer.next != null) {
            pointer = pointer.next;
        }

        Node newNode = new Node(value);
        newNode.prev = pointer;
        pointer.next = newNode;
    }

    /**
     * 노드의 값으로 삭제
     * @param value
     */
    public void delete(int value) {
        Node pointer = this.head;

        if (pointer.getValue() == value) {
            Node removeNode = this.head;
            this.head = this.head.next;

            removeNode = null;
            return;
        }

        Node prevNode = pointer;
        while (pointer != null && pointer.getValue() != value) {
            prevNode = pointer;
            pointer = pointer.next;
        }

        Node temp = pointer.next;
        if (temp == null) {
            prevNode.next = null;
        } else {
            temp.prev = prevNode;
            prevNode.next = pointer.next;
        }

        pointer = null;
    }

    /**
     * 인자 값으로 앞 노드의 값을 알아내기
     * @param value
     * @return
     */
    public void printPrevNode(int value) {
        if (this.head == null) {
            System.out.println("이중 연결 리스트가 비어 있습니다.");
            return;
        }

        if (this.head.getValue() == value) {
            System.out.println(String.format("노드 %s 의 앞 노드는 존재 하지 않습니다.", value));
            return;
        }
```

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

```java
package linkedList;

public class CLLInsertionMain {
    Run | Debug
    public static void main(String[] args) {
        CircularLinkedList circularLinkedList = new CircularLinkedList();
        circularLinkedList.append(11);
        circularLinkedList.append(9);
        circularLinkedList.append(27);
        circularLinkedList.append(41);
        circularLinkedList.append(3);

        System.out.print("원형 단일 연결 리스트: ");
        circularLinkedList.printAll();
    }
}
```

11 – 9 – 27 – 41 – 3

2. Linked List

원형 단일 연결 리스트 구현: 삽입

```java
package linkedList;

/**
 * 원형 단일 연결 리스트 클래스
 */
public class CircularLinkedList {
    private Node head;
    private Node tail;

    CircularLinkedList() {
        this.head = null;
        this.tail = null;
    }

    class Node {
        private int value; // 값
        private Node next; // 다음 포인터

        Node(int value) {
            this.value = value;
            this.next = null;
        }

        public int getValue() {
            return this.value;
        }
    }

    /**
     * 원형 단일 연결 리스트 끝에 노드 추가
     * @param value
     */
    public void append(int value) {
        // 첫 노드 삽입 일 때
        if (this.head == null && this.tail == null) {
            Node node = new Node(value);
            this.head = node;
            this.tail = node;
            return;
        }

        Node pointer = this.tail;
        // 마지막 위치에 새 노드 삽입
        pointer.next = new Node(value);

        // 새로 삽입한 노드를 tail로 재할당
        this.tail = pointer.next;
        // tail의 next 포인터를 head로 연결하여 원형 연결 리스트 조건 만족시킴
        this.tail.next = head;
    }

    /**
     * 모든 노드 출력
     */
    public void printAll() {
        Node pointer = this.head;

        StringBuilder builder = new StringBuilder();
        while (pointer != this.tail) {
            builder.append(pointer.getValue());
            builder.append(" -> ");
            pointer = pointer.next;
        }
        builder.append(pointer.getValue());
        builder.append("(tail) -> ");

        builder.append(this.head.getValue());
        builder.append("(head)");

        System.out.println(builder.toString());
    }
}
```

2. Linked List

원형 단일 연결 리스트 구현: 삭제

```java
package linkedList;

public class CLLDeletionMain {
    Run | Debug
    public static void main(String[] args) {
        CircularLinkedList circularLinkedList = new CircularLinkedList
        circularLinkedList.append(11);
        circularLinkedList.append(9);
        circularLinkedList.append(27);
        circularLinkedList.append(41);
        circularLinkedList.append(3);

        System.out.println("원본 원형 단일 연결 리스트: ");
        circularLinkedList.printAll();

        System.out.println("노드 3을 삭제한 원형 단일 연결 리스트: ");
        circularLinkedList.delete(3);
        circularLinkedList.printAll();

        System.out.println("노드 11를 삭제한 원형 단일 연결 리스트: ");
        circularLinkedList.delete(11);
        circularLinkedList.printAll();

        System.out.println("노드 27를 삭제한 원형 단일 연결 리스트: ");
        circularLinkedList.delete(27);
        circularLinkedList.printAll();
    }
}
```

2. Linked List

원형 단일 연결 리스트 구현: 삭제

```java
package linkedList;

/**
 * 원형 단일 연결 리스트 클래스
 */
public class CircularLinkedList {
    private Node head;
    private Node tail;

    CircularLinkedList() {
        this.head = null;
        this.tail = null;
    }

    class Node {
        private int value; // 값
        private Node next; // 다음 포인터

        Node(int value) {
            this.value = value;
            this.next = null;
        }

        public int getValue() {
            return this.value;
        }
    }

    /**
     * 원형 단일 연결 리스트 끝에 노드 추가
     * @param value
     */
    /**
     * 원형 단일 연결 리스트 노드 삭제
     * @param value
     */
    public void delete(int value) {
        Node pointer = this.head;

        if (pointer.getValue() == value) {
            Node removeNode = this.head;
            this.head = this.head.next;

            removeNode = null;
            return;
        }

        // 포인터가 tail이 아니며 값이 다를 때까지 반복
        Node temp = null;
        while (pointer.next != this.tail && pointer.getValue() != value) {
            temp = pointer;
            pointer = pointer.next;
        }

        // 삭제 할 노드가 tail일 경우 앞 노드를 tail로 재할당하고 next를 head로 연결
        if (pointer.next.getValue() == value) {
            this.tail = pointer;
            this.tail.next = this.head;
        } else {
            temp.next = pointer.next;
        }
        pointer = null;
    }
    /**
     * 모든 노드 출력
     */
    public void printAll() {
        Node pointer = this.head;

        StringBuilder builder = new StringBuilder();
        while (pointer != this.tail) {
            builder.append(pointer.getValue());
            builder.append(" -> ");
            pointer = pointer.next;
        }
        builder.append(pointer.getValue());
        builder.append("(tail) -> ");

        builder.append(this.head.getValue());
        builder.append("(head)");

        System.out.println(builder.toString());
    }
}
```

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