< ArrayStack 구현 >

1. ArrayStack 클래스

```
class ArrayStack<E> implements Stack<E> {
    5 usages
    private int size;
    5 usages
    private int top;
    4 usages
    private E[] listArray;

    //초기화 & 생성자
    1 usage
    public ArrayStack(int size) {
        listArray = (E[]) new Object[size];
        top = -1; //-1로 해줘야 첫 원소가 들어왔을 때 1 증가시
    }
```

2. 메서드 정의

```
    @Override
    public void clear() {
        size = 0;
        top = -1;
    }

    6 usages
    @Override
    public void push(E item) {
        size++;
        listArray[++top] = item;
    }

    2 usages
    @Override
    public E pop() {
        if (size == 0) {
            System.out.println("Stack is empty");
        }
        size--;
        return listArray[top--];
    }
```

```
35      @Override
36      public E topValue() {
37          return listArray[top];
38      }
39
        5 usages
40      @Override
41      public int length() {
42          return size;
43      }
44  }
```

3. **Test**

```
46  public class Array_Stack {
47      public static void main(String[] args) throws Exception {
48          Stack<String> myList = new ArrayStack<String>( size: 20);
49
50          myList.push( item: "first push");
51          myList.push( item: "second push");
52          myList.push( item: "third push");
53
54          System.out.println("length before pop: ");
55          System.out.println(myList.length());
56
57          System.out.println("top value: ");
58          System.out.println(myList.topValue());
59
60          System.out.println("pop!: ");
61          System.out.println(myList.pop());
62
63          System.out.println("top value: ");
64          System.out.println(myList.topValue());
65
66          System.out.println("length after pop: ");
67          System.out.println(myList.length());
68      }
```

```
length before pop:
3
top value:
third push
pop!:
third push
top value:
second push
length after pop:
2

Process finished with exit code 0
```

< LinkedStack 구현 >

1. LinkedStack 클래스

```
class LinkedStack<E> implements Stack<E> {
    8 usages
    private Link<E> top;
    5 usages
    private int size;

    1 usage
    public LinkedStack() {
        top = null;
        size = 0;
    }
```

2. 메서드 정의

```
    @Override
    public void clear() {
        size = 0;
        top = null;
    }

    6 usages
    @Override
    public void push(E item) {
        top = new Link<E>(item, top);
        size++;
    }

    2 usages
    @Override
    public E pop() {
        E item = top.item; // top 요소의 값 저장
        top = top.getNext(); // top 요소를 다음 요소로 변경
        size--; // 스택 크기 1 감소
        return item; // 삭제한 값 반환
    }

    4 usages
    @Override
    public E topValue() {
        return top.item;
    }
```

```
37      @Override
38      public int length() {
39          return size;
40      }
41  }
```

3. **Test**

```
50  public class Linked_Stack {
51      public static void main(String[] args) throws Exception {
52          Stack<String> myList = new LinkedStack<~>();
53
54          myList.push( item: "first push");
55          myList.push( item: "second push");
56          myList.push( item: "third push");
57
58          System.out.println("length before pop: ");
59          System.out.println(myList.length());
60
61          System.out.println("top value: ");
62          System.out.println(myList.topValue());
63
64          System.out.println("pop!: ");
65          System.out.println(myList.pop());
66
67          System.out.println("top value: ");
68          System.out.println(myList.topValue());
69
70          System.out.println("length after pop: ");
71          System.out.println(myList.length());
72
73          myList.clear();
74          System.out.println(myList.length());
75      }
76  }
```

```
length before pop:
3
top value:
third push
pop!:
third push
top value:
second push
length after pop:
2
0

Process finished with exit code 0
```

< ArrayQueue 구현 >

1. ArrayQueue 클래스

```
class ArrayQueue<E> implements Queue<E> {
    7 usages
    private int size; // 큐에 들어가있는 요소 개수
    6 usages
    private int front, rear; // 큐의 맨 앞, 맨 뒤 요소
    10 usages
    private E[] listArray; // 큐 배열

    1 usage
    public ArrayQueue(int size) { // 매개변수로 받음
        listArray = (E[]) new Object[size]; // 주어진 크기로 새 배열 생
        front = 0; // 맨 앞 요소의 인덱스 0으로 초기화
        rear = -1;
        this.size = 0;
    }
```

2. 메서드 정의

```
    @Override
    public void clear() {
        listArray = (E[]) new Object[listArray.length];
        front = 0;
        rear = -1;
        size = 0;
    }
```

```java
    @Override
    public void enqueue(E item) {
        if (isFull()) {
            System.out.println("Queue is full");
        }
        rear = (rear + 1) % listArray.length;
        listArray[rear] = item;
        size++;
    }

    4 usages
    @Override
    public E dequeue() {
        if (isEmpty()) {
            System.out.println("Queue is empty");
        }
        E queueFront = listArray[front];
        front = (front + 1) % listArray.length;
        size--;
        return queueFront;
    }
```

```java
    @Override
    public E frontValue() {
        if (isEmpty()) {
            System.out.println("Queue is empty");
        }
        return listArray[front];
    }

    4 usages
    @Override
    public int length() {
        return size;
    }

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

    2 usages
    @Override
    public boolean isFull() {
        return size == listArray.length;
    }
}
```

3. Test

```java
public class Array_Queue {
    public static void main(String[] args) throws Exception {
        Queue<Integer> queue = new ArrayQueue<Integer>( size: 5);

        queue.enqueue( item: 1);
        queue.enqueue( item: 2);
        queue.enqueue( item: 3);

        System.out.println("Front of the queue: " + queue.frontValue());
        System.out.println("Queue size: " + queue.length());

        int element = queue.dequeue();
        System.out.println("Dequeued element: " + element);
        System.out.println("Queue size after dequeue: " + queue.length());

        queue.enqueue( item: 4);
        queue.enqueue( item: 5);

        System.out.println("Queue is full: " + queue.isFull());

        while (!queue.isEmpty()) { // 큐가 비어있지 않은 경우 반복
            element = queue.dequeue(); // 앞에 있는 요소 삭제
            System.out.println("Dequeued element: " + element);
        }

        System.out.println("Queue is empty: " + queue.isEmpty());
    }
}
```

```
ListArray.length: 5
size: 1
ListArray.length: 5
size: 2
ListArray.length: 5
size: 3
Front of the queue: 1
Queue size: 3
Dequeued element: 1
Queue size after dequeue: 2
ListArray.length: 5
size: 3
ListArray.length: 5
size: 4
Queue is full: false
Dequeued element: 2
Dequeued element: 3
Dequeued element: 4
Dequeued element: 5
Queue is empty: true

Process finished with exit code 0
```

< LinkedQueue 구현 >

1. LinkedQueue 클래스

```
class LinkedQueue<E> implements Queue<E> {
    7 usages
    private Link<E> front;
    5 usages
    private Link<E> rear;
    6 usages
    private int size;

    1 usage
    public LinkedQueue() {
        //처음 생성 시 아무런 데이터가 없으므로 front와 rear은 가리킬 노드가 없는 상태임
        front = rear = new Link<E>( item: null,  next: null);
        size = 0;
    }
```

+) Link

```
public class Link<E> { // Node 라고 표현해도 됨
    5 usages
    public E item;
    6 usages
    public Link<E> next; // ref

    2 usages
    public Link(E item, Link<E> next) {
        this.item = item;
        this.next = next;
    }
}
```

2. 메서드 정의

```java
@Override
public void clear() {
    while (front != null) {
        Link<E> next = front.next;
        front.item = null;
        front.next = null;
        front = next;
    }
    front = rear = null;
    size = 0;
}
```

```java
@Override
public void enqueue (E item) {
    Link<E> newLink = new Link<E> (item,  next: null);

    if (isEmpty()) {
        // 비어있는 경우, 새 요소가 front 이자 rear 됨
        front = newLink;
    }
    else {
        rear.next = newLink;
    }
    rear = newLink;
    size++;
}
```

```java
@Override
public E dequeue() {
    if (isEmpty()) {
        System.out.println("Queue is empty");
        return null;
    }

    // 삭제할 요소를 반환하기 위한 임시 변수
    E element = front.item;

    // nextLink 는 front 노드의 다음 노드를 가리키게 됨
    Link<E> nextLink = front.next;

    //front 의 모든 요소들 삭제
    front.item = null;
    front.next = null;

    // front 가 가리키는 노드를 삭제된 front 의 다음 노드를 가리키
    front = nextLink;
    size--;

    return element;
}
```

```java
    @Override
    public E frontValue() {
        if (isEmpty()) {
            System.out.println("Queue is empty");
            return null;
        }
        return front.item;
    }

    4 usages
    @Override
    public int length() {
        return size;
    }

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

    2 usages
    @Override
    public boolean isFull() {
        return false;
    }
}
```

3. **Test**

```
Front of the queue: 1
Queue size: 3
Dequeue element: 1
Queue size after dequeue: 2
Dequeued element: 2
Dequeued element: 3
Dequeued element: 4
Dequeued element: 5
Queue is empty: true

Process finished with exit code 0
```

< InternalNode & LeafNode 구현 >

1. InternalNode 클래스

```
class InternalNode<E> implements BinNode<E> {
    3 usages
    private E element;
    2 usages
    private BinNode<E> leftChild;
    2 usages
    private BinNode<E> rightChild;

    6 usages
    public InternalNode(E element, BinNode<E> leftChild, BinNode<E> rightChild) {
        this.element = element;
        this.leftChild = leftChild;
        this.rightChild = rightChild;
    }
```

2. LeafNode 클래스

```
class LeafNode<E> implements BinNode<E> {
    3 usages
    private E element;
```

3. InternalNode 메서드 정의

```
    @Override
    public E element() {
        return element;
    }

    no usages
    @Override
    public void setElement(E element) {
        this.element = element;
    }

    4 usages
    @Override
    public BinNode<E> left() {
        return leftChild;
    }

    4 usages
    @Override
    public BinNode<E> right() {
        return rightChild;
    }

    no usages
    @Override
    public boolean isLeaf() {
        return false;
    }
}
```

4. LeafNode 메서드 정의

```java
    @Override
    public E element() {
        return element;
    }

    no usages
    @Override
    public void setElement(E element) {
        this.element = element;
    }

    4 usages
    @Override
    public BinNode<E> left() {
        return null;
    }

    4 usages
    @Override
    public BinNode<E> right() {
        return null;
    }

    no usages
    @Override
    public boolean isLeaf() {
        return true;
    }
```

5. TreeTest

```java
public class TreeTest {
    public static void main(String[] args){
        System.out.println("Test your trees here!");
        BinNode<String> node9 = new LeafNode<>( element: "I");
        BinNode<String> node8 = new LeafNode<>( element: "H");
        BinNode<String> node7 = new LeafNode<>( element: "G");
        BinNode<String> node6 = new InternalNode<>( element: "F", node8, node9);
        BinNode<String> node5 = new InternalNode<>( element: "E",  leftChild: null, node7);
        BinNode<String> node4 = new LeafNode<>( element: "D");
        BinNode<String> node3 = new InternalNode<>( element: "C",  leftChild: null, node6);
        BinNode<String> node2 = new InternalNode<>( element: "B", node4, node5);
        BinNode<String> node1 = new InternalNode<>( element: "A", node2, node3);

        InternalNode<String> tree = new InternalNode(node1.element(), node1.left(), node1.right());

        System.out.println("Preorder:");
        TreeTest.preorder(node1);

        System.out.println("\nInorder:");
        TreeTest.inorder(node1);

        System.out.println("\nPostorder:");
        TreeTest.postorder(node1);
    }
```

```java
    // preorder
    3 usages
    public static <E> void preorder(BinNode<E> node) {
        if (node == null) {
            return;
        }
        System.out.print(node.element() + " ");
        preorder(node.left());
        preorder(node.right());
    }
    // inorder
    3 usages
    public static <E> void inorder(BinNode<E> node) {
        if (node == null) {
            return;
        }
        inorder(node.left());
        System.out.print(node.element() + " ");
        inorder(node.right());
    }
    // postorder
    3 usages
    public static <E> void postorder(BinNode<E> node) {
        if (node == null) {
            return;
        }
        postorder(node.left());
        postorder(node.right());
        System.out.print(node.element() + " ");
    }
```