

< ArrayStack 구현 >

1. ArrayStack 클래스

```
3  class ArrayStack<E> implements Stack<E> {
4      5 usages
5      private int size;
6      5 usages
7      private int top;
8      4 usages
9      private E[] listArray;
10
11     //초기화 & 생성자
12     1 usage
13     public ArrayStack(int size) {
14         listArray = (E[]) new Object[size];
15         top = -1; //-1로 해줘야 첫 원소가 들어왔을 때 1 증가시키
16     }
```

2. 메서드 정의

```
14     @Override
15     public void clear() {
16         size = 0;
17         top = -1;
18     }
19
20     6 usages
21     @Override
22     public void push(E item) {
23         size++;
24         listArray[++top] = item;
25     }
26
27     2 usages
28     @Override
29     public E pop() {
30         if (size == 0) {
31             System.out.println("Stack is empty");
32         }
33         size--;
34         return listArray[top--];
35     }
```

```

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

```
3  class LinkedStack<E> implements Stack<E> {  
    8 usages  
4      private Link<E> top;  
    5 usages  
5      private int size;  
6  
    1 usage  
7      public LinkedStack() {  
8          top = null;  
9          size = 0;  
10     }  
11 }
```

2. 메서드 정의

```
12     @Override  
13     public void clear() {  
14         size = 0;  
15         top = null;  
16     }  
17  
    6 usages  
18     @Override  
19     public void push(E item) {  
20         top = new Link<E>(item, top);  
21         size++;  
22     }  
23  
    2 usages  
24     @Override  
25     public E pop() {  
26         E item = top.item; // top 요소의 값 저장  
27         top = top.getNext(); // top 요소를 다음 요소로 변경  
28         size--; // 스택 크기 1 감소  
29         return item; // 삭제한 값 반환  
30     }  
31  
    4 usages  
32     @Override  
33     public E topValue() {  
34         return top.item;  
35     }  
36 }
```

```

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

```
3  class ArrayQueue<E> implements Queue<E> {  
    7 usages  
4      private int size; // 큐에 들어가있는 요소 개수  
    6 usages  
5      private int front, rear; // 큐의 맨 앞, 맨 뒤 요소  
    10 usages  
6      private E[] listArray; // 큐 배열  
7  
    1 usage  
8      public ArrayQueue(int size) { // 매개변수로 받음  
9          listArray = (E[]) new Object[size]; // 주어진 크기로 새 배열 생성  
10         front = 0; // 맨 앞 요소의 인덱스 0으로 초기화  
11         rear = -1;  
12         this.size = 0;  
13     }
```

2. 메서드 정의

```
15      @Override  
16      public void clear() {  
17          listArray = (E[]) new Object[listArray.length];  
18          front = 0;  
19          rear = -1;  
20          size = 0;  
21      }
```

```

23     @Override
24     public void enqueue(E item) {
25         if (isFull()) {
26             System.out.println("Queue is full");
27         }
28         rear = (rear + 1) % listArray.length;
29         listArray[rear] = item;
30         size++;
31     }
32
33     4 usages
34     @Override
35     public E dequeue() {
36         if (isEmpty()) {
37             System.out.println("Queue is empty");
38         }
39         E queueFront = listArray[front];
40         front = (front + 1) % listArray.length;
41         size--;
42         return queueFront;

```

```

44     @Override
45     public E frontValue() {
46         if (isEmpty()) {
47             System.out.println("Queue is empty");
48         }
49         return listArray[front];
50     }
51
52     4 usages
53     @Override
54     public int length() {
55         return size;
56     }
57
58     9 usages
59     @Override
60     public boolean isEmpty() {
61         return size == 0;
62     }
63
64     2 usages
65     @Override
66     public boolean isFull() {
67         return size == listArray.length;

```


3. Test

```
68 ▶ public class Array_Queue {
69 ▶     public static void main(String[] args) throws Exception {
70         Queue<Integer> queue = new ArrayQueue<Integer>(size: 5);
71
72         queue.enqueue(item: 1);
73         queue.enqueue(item: 2);
74         queue.enqueue(item: 3);
75
76         System.out.println("Front of the queue: " + queue.frontValue());
77         System.out.println("Queue size: " + queue.length());
78
79         int element = queue.dequeue();
80         System.out.println("Dequeued element: " + element);
81         System.out.println("Queue size after dequeue: " + queue.length());
82
83         queue.enqueue(item: 4);
84         queue.enqueue(item: 5);
85
86         System.out.println("Queue is full: " + queue.isFull());
87
88         while (!queue.isEmpty()) { // 큐가 비어있지 않은 경우 반복
89             element = queue.dequeue(); // 앞에 있는 요소 삭제
90             System.out.println("Dequeued element: " + element);
91         }
92
93         System.out.println("Queue is empty: " + queue.isEmpty());
94     }
95 }
```

```
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
```

< LinkedList 구현 >

1. LinkedList 클래스

```
3  class LinkedList<E> implements Queue<E> {  
    7 usages  
4      private Link<E> front;  
    5 usages  
5      private Link<E> rear;  
    6 usages  
6      private int size;  
7  
    1 usage  
8      public LinkedList() {  
9          //처음 생성 시 아무런 데이터가 없으므로 front와 rear은 가리킬 노드가 없는 상태임  
10         front = rear = new Link<E>(item: null, next: null);  
11         size = 0;  
12     }  
}
```

+) Link

```
3      public class Link<E> { // Node 라고 표현해도 됨  
    5 usages  
4          public E item;  
    6 usages  
5          public Link<E> next; // ref  
6  
    2 usages  
7      public Link(E item, Link<E> next) {  
8          this.item = item;  
9          this.next = next;  
10     }  
11 }  
12 }
```

2. 메서드 정의

```

14      @Override
15      public void clear() {
16          while (front != null) {
17              Link<E> next = front.next;
18              front.item = null;
19              front.next = null;
20              front = next;
21          }
22          front = rear = null;
23          size = 0;
24      }

```

```

26      @Override
27      public void enqueue (E item) {
28          Link<E> newLink = new Link<E> (item, next: null);
29
30          if (isEmpty()) {
31              // 비어있는 경우, 새 요소가 front 이자 rear 됨
32              front = newLink;
33          }
34          else {
35              rear.next = newLink;
36          }
37          rear = newLink;
38          size++;
39      }

```

```

41      @Override
42      public E dequeue() {
43          if (isEmpty()) {
44              System.out.println("Queue is empty");
45              return null;
46          }
47
48          // 삭제할 요소를 반환하기 위한 임시 변수
49          E element = front.item;
50
51          // nextLink 는 front 노드의 다음 노드를 가리키게 됨
52          Link<E> nextLink = front.next;
53
54          //front 의 모든 요소들 삭제
55          front.item = null;
56          front.next = null;
57
58          // front 가 가리키는 노드를 삭제된 front 의 다음 노드를 가리키
59          front = nextLink;
60          size--;
61
62          return element;
63      }

```

```

65         @Override
66         public E frontValue() {
67             if (isEmpty()) {
68                 System.out.println("Queue is empty");
69                 return null;
70             }
71             return front.item;
72         }
73
74         4 usages
75         @Override
76         public int length() {
77             return size;
78         }
79
80         8 usages
81         @Override
82         public boolean isEmpty() {
83             return size == 0;
84         }
85
86         2 usages
87         @Override
88         public boolean isFull() {
89             return false;
90         }
91     }

```

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

```
4      class InternalNode<E> implements BinNode<E> {  
5          3 usages  
6          private E element;  
7          2 usages  
8          private BinNode<E> leftChild;  
9          2 usages  
10         private BinNode<E> rightChild;  
11  
12         6 usages  
13         public InternalNode(E element, BinNode<E> leftChild, BinNode<E> rightChild) {  
14             this.element = element;  
15             this.leftChild = leftChild;  
16             this.rightChild = rightChild;  
17         }  
18     }
```

2. LeafNode 클래스

```
5      class LeafNode<E> implements BinNode<E> {  
6          3 usages  
7          private E element;  
8      }
```

3. InternalNode 메서드 정의

```
15         @Override  
16         public E element() {  
17             return element;  
18         }  
19  
20         no usages  
21         @Override  
22         public void setElement(E element) {  
23             this.element = element;  
24         }  
25  
26         4 usages  
27         @Override  
28         public BinNode<E> left() {  
29             return leftChild;  
30         }  
31  
32         4 usages  
33         @Override  
34         public BinNode<E> right() {  
35             return rightChild;  
36         }  
37  
38         no usages  
39         @Override  
40         public boolean isLeaf() {  
41             return false;  
42         }  
43     }
```

4. LeafNode 메서드 정의

```
12      @Override
13      public E element() {
14          return element;
15      }
16
17      no usages
18      @Override
19      public void setElement(E element) {
20          this.element = element;
21      }
22
23      4 usages
24      @Override
25      public BinNode<E> left() {
26          return null;
27      }
28
29      4 usages
30      @Override
31      public BinNode<E> right() {
32          return null;
33      }
34
35      no usages
36      @Override
37      public boolean isLeaf() {
38          return true;
39      }
```

5. TreeTest

```
3 ▶ public class TreeTest {
4 ▶     public static void main(String[] args){
5         System.out.println("Test your trees here!");
6         BinNode<String> node9 = new LeafNode<>(element: "I");
7         BinNode<String> node8 = new LeafNode<>(element: "H");
8         BinNode<String> node7 = new LeafNode<>(element: "G");
9         BinNode<String> node6 = new InternalNode<>(element: "F", node8, node9);
10        BinNode<String> node5 = new InternalNode<>(element: "E", leftChild: null, node7);
11        BinNode<String> node4 = new LeafNode<>(element: "D");
12        BinNode<String> node3 = new InternalNode<>(element: "C", leftChild: null, node6);
13        BinNode<String> node2 = new InternalNode<>(element: "B", node4, node5);
14        BinNode<String> node1 = new InternalNode<>(element: "A", node2, node3);
15
16        InternalNode<String> tree = new InternalNode(node1.element(), node1.left(), node1.right());
17
18        System.out.println("Preorder:");
19        TreeTest.preorder(node1);
20
21        System.out.println("\nInorder:");
22        TreeTest.inorder(node1);
23
24        System.out.println("\nPostorder:");
25        TreeTest.postorder(node1);
26    }
```

```
30        // preorder
31        3 usages
32        public static <E> void preorder(BinNode<E> node) {
33            if (node == null) {
34                return;
35            }
36            System.out.print(node.element() + " ");
37            preorder(node.left());
38            preorder(node.right());
39        }
40        // inorder
41        3 usages
42        public static <E> void inorder(BinNode<E> node) {
43            if (node == null) {
44                return;
45            }
46            inorder(node.left());
47            System.out.print(node.element() + " ");
48            inorder(node.right());
49        }
50        // postorder
51        3 usages
52        public static <E> void postorder(BinNode<E> node) {
53            if (node == null) {
54                return;
55            }
56            postorder(node.left());
57            postorder(node.right());
58            System.out.print(node.element() + " ");
59        }
```