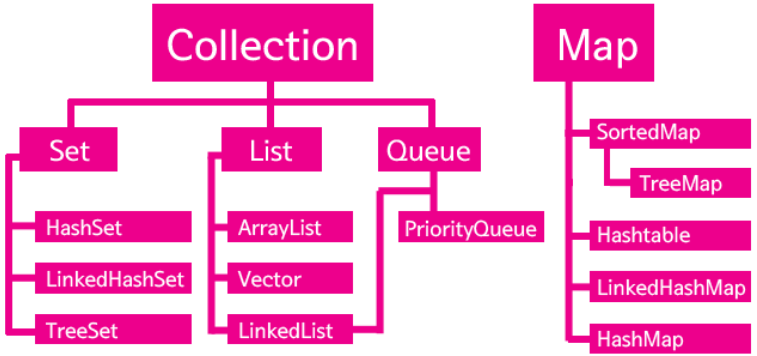

Collection
Map
Set
List
Queue
SortedMap
TreeMap
HashSet
ArrayList
PriorityQueue
Hashtable
LinkedHashSet
Vector
LinkedHashMap
TreeSet
LinkedList
HashMap

큐 (Queue)

큐 코딩테스트 대표 문제

- 큐 자료구조 구현 문제
  - 직접 나오지는 않지만, STL을 금지해서 구현해야하는 경우가 있음(삼성SW역량 B유형, 오프라인 면접시)
- 기본 큐 활용
  - 프린터, 대기 문제
- 큐 응용
  - Deque, 우선순위큐(heap) 문제
  - 탐색(BFS가 이용하는 것이 Queue)
    - cf 탐색(DFS를 구현하기 위해서 재귀 또는 스택이 필요)

큐 (QUEUE): FIFO 추상 데이터 타입

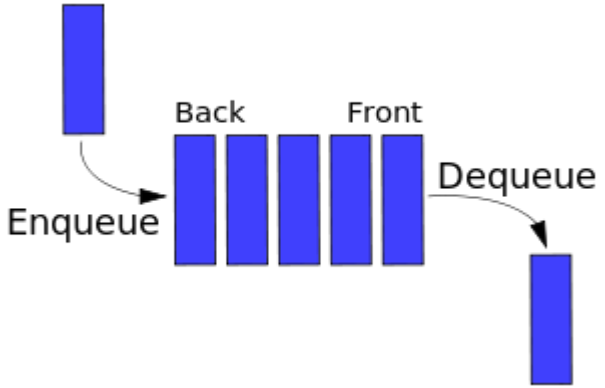
- First In, First Out
- JAVA 에서는 보통 Linked List 사용
- 탐색, 저장, 순서 쌍 맞추기 등에 활용
- Interface: Enqueue (add), Dequeue (remove)

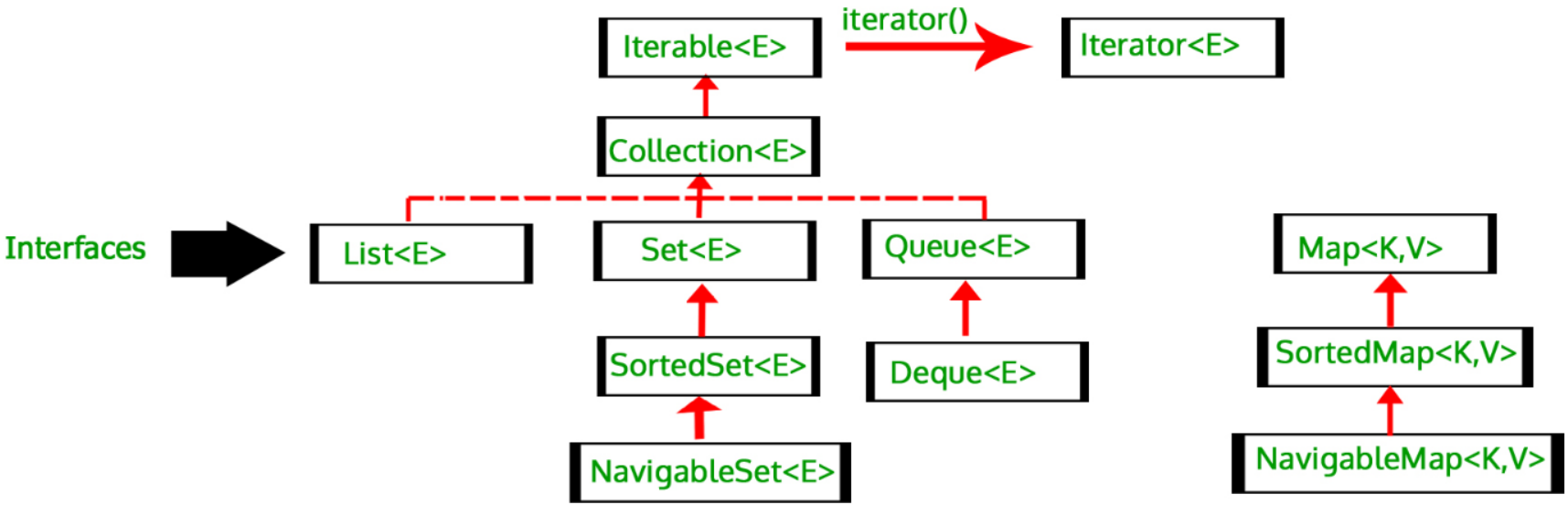
```java
public class QueueExample
{
  public static void main(String[] args)
  {
    Queue<Integer> q = new LinkedList<>();

    // Adds elements {0, 1, 2, 3, 4} to queue
    for (int i=0; i<5; i++)
     q.add(i);

    // Display contents of the queue.
    System.out.println("Elements of queue-"+q);

    // To remove the head of queue.
    int removedele = q.remove();
    System.out.println("removed element-" + removedele);
```





Implementation classes

ArrayList
LinkedList
Stack
Vector

HashSet
LinkedHashSet
TreeSet

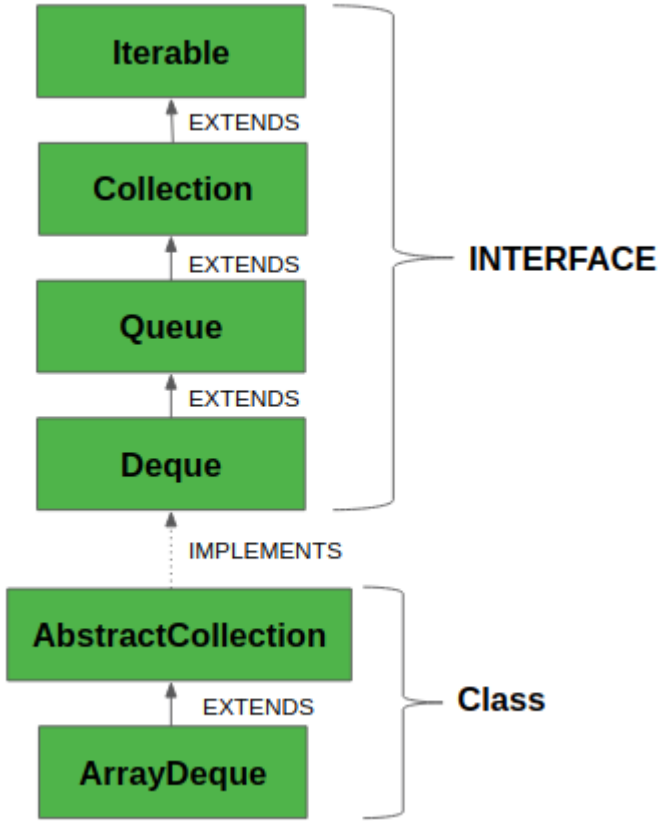
PriorityQueue
ArrayDeque
LinkedList(Deque)

HashMap
HashLinkedMap
HashTable
TreeMap

Java Deque Interface

- java.util.Deque
  - double ended queue: stack or queue
- Deque 는 인터페이스
  - 상속받은 클래스 사용

```
Deque<String> deque = new LinkedList<String>();
```



java.util

Interface Deque<E>

Type Parameters:

`E - the type of elements held in this collection`

All Superinterfaces:

`Collection<E>, Iterable<E>, Queue<E>`

All Known Subinterfaces:

`BlockingDeque<E>`

All Known Implementing Classes:

`ArrayDeque, ConcurrentLinkedDeque, LinkedBlockingDeque, LinkedList`

Summary of Deque methods

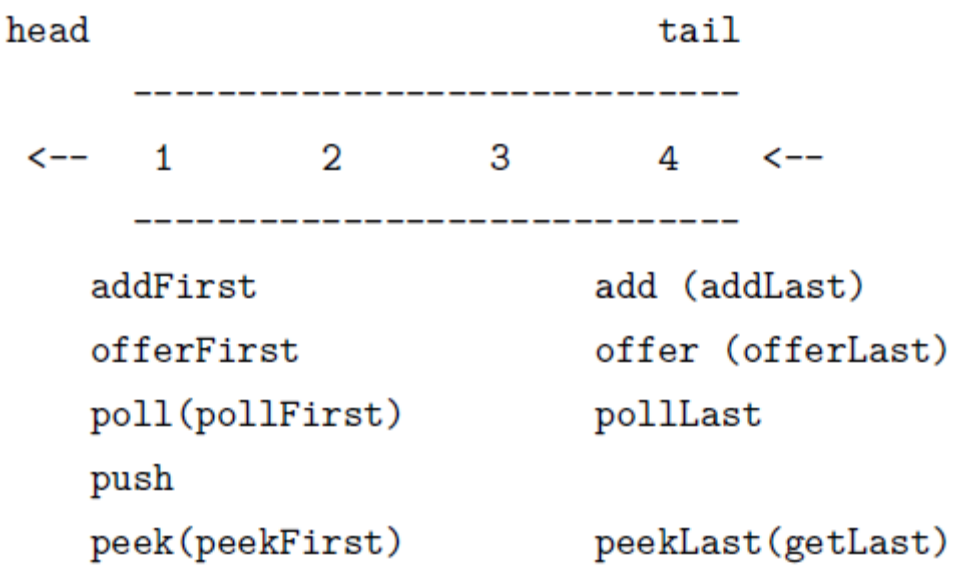
	First Element (Head)		Last Element (Tail)	
	Throws exception	*Special value*	*Throws exception*	*Special value*
Insert	`addFirst(e)`	`offerFirst(e)`	`addLast(e)`	`offerLast(e)`
Remove	`removeFirst()`	`pollFirst()`	`removeLast()`	`pollLast()`
Examine	`getFirst()`	`peekFirst()`	`getLast()`	`peekLast()`

Comparison of Queue and Deque methods

Queue Method	Equivalent Deque Method
`add(e)`	`addLast(e)`
`offer(e)`	`offerLast(e)`
`remove()`	`removeFirst()`
`poll()`	`pollFirst()`
`element()`	`getFirst()`
`peek()`	`peekFirst()`

Comparison of Stack and Deque methods

Stack Method	Equivalent Deque Method
`push(e)`	`addFirst(e)`
`pop()`	`removeFirst()`
`peek()`	`getFirst()`


head
tail
<--
1
2
3
4
<--
addFirst
offerFirst
poll(pollFirst)
push
peek(peekFirst)
add (addLast)
offer (offerLast)
pollLast
peekLast(getLast)

Queue vs. Deque

```
import java.util.ArrayDeque;
import java.util.LinkedList;
import java.util.PriorityQueue;
import java.util.Queue;

public class Test {
    public static void main(String[] args) {
        Queue<Integer> queue1 =new ArrayDeque<>();
        Queue<Double> queue2 =new LinkedList<>();
        Queue<String> queue3 =new PriorityQueue<>();
    }
}
```

```
import java.util.Deque;
import java.util.LinkedList;

public class Test {
    public static void main(String[] args) {
        Deque<Integer> deque1 =new ArrayDeque<>();
        Deque<Double> deque2 =new LinkedList<>();
    }
}
```

```java
// Java program to demonstrate working of
// Deque in Java
import java.util.*;

public class DequeExample
{
    public static void main(String[] args)
    {
        Deque<String> deque = new LinkedList<String>();

        // We can add elements to the queue in various ways
        deque.add("Element 1 (Tail)"); // add to tail
        deque.addFirst("Element 2 (Head)");
        deque.addLast("Element 3 (Tail)");
        deque.push("Element 4 (Head)"); //add to heac
        deque.offer("Element 5 (Tail)");
        deque.offerFirst("Element 6 (Head)");
        deque.offerLast("Element 7 (Tail)");

        System.out.println(deque + "\n");

        // Iterate through the queue elements.
        System.out.println("Standard Iterator");
        Iterator iterator = deque.iterator();
        while (iterator.hasNext())
            System.out.println("\t" + iterator.next()


        // Reverse order iterator
        Iterator reverse = deque.descendingIterator()
        System.out.println("Reverse Iterator");
        while (reverse.hasNext())
            System.out.println("\t" + reverse.next())
```

```java
// Peek returns the head, without deleting
// it from the deque
System.out.println("Peek " + deque.peek());
System.out.println("After peek: " + deque);

// Pop returns the head, and removes it from
// the deque
System.out.println("Pop " + deque.pop());
System.out.println("After pop: " + deque);

// We can check if a specific element exists
// in the deque
System.out.println("Contains element 3: " +
                   deque.contains("Element 3 (Tail)"));

// We can remove the first / last element.
deque.removeFirst();
deque.removeLast();
System.out.println("Deque after removing " +
```

스크린샷

Queue 직접 만들어보기

```
package com.cscnu.Queue;

public interface Queue {
		public void Enqueue (Object object);
		public Object Dequeue();
		public Object getFront();
		public int size ();
}
```

```
public class ArrayQueue implements Queue {
        private static final int MAXQUEUE = 1000;
        private Object[] obj = new Object[MAXQUEUE];
        private int size = 0;
        private int front = 0, rear = 0;

        public void Enqueue (Object object) {
                if (size >= MAXQUEUE) {
                        System.out.println("The queue is full.");
                } else {
                        obj[rear] = object;
                        rear ++;
                        size ++;
                }
        }
        public Object getFront () {
                if (size == 0) {
                        throw new IllegalStateException("The queue is empty.");
                } else {
                        return obj[front];
                }
        }
```

```
		public Object Dequeue() {
			if (size == 0) {
				throw new IllegalStateException ("The
queue is empty.");
			} else {
				Object temp = obj[front];
				size --;
				System.arraycopy(obj, 1, obj, 0, size);
				rear --;
				return temp;
			}
		}
		public int size () {
			return size;
		}
}
```

Queue 만들기 확장

- 배열 큐
- 링크드 리스트 큐
- 순환 큐(Circular queue)
    - Double linked list 클래스에서 dummy head 노드를 만들어 순환 큐 만들기!

Double Linked List Node Class

```
public class Node {
        public Node next = null;
        public Node prev = null;
        public Object data = null;
        Node (Object data) {
                this.data = data;
        }
        Node (Object data, Node next) {
                this.data = data;
                this.next = next;
        }
        Node (Object data, Node prev, Node next) {
                this.data = data;
                this.prev = prev;
                this.next = next;
        }
}
```

```
public class CircularDoubleLinkedList implements DoubleLinked {
        private Node head;
        private int size;

        public CircularDoubleLinkedList () {
                this.head = new Node (null, head, head);
                this.size = 0;
        }


    public boolean insertFirst (Object data) {
            if (isEmpty()) {
                    head.next = new Node (data, head, head);
                    head.prev = head.next;
            } else {
                    Node tmp = head.next;
                    head.next = new Node (data, head, tmp);
                    tmp.prev = head.next;
            }
            size ++;
            return true;
```

연습문제: 큐 기본 학습

- 세가지 명령을 수행하는 장치가 있다. 주어진 명령어 셋을 수행한 결과를 출력하시오.
  1 - Enqueue, 2 - Dequeue, 3 - print head number

[입력]
10
1 97
2
1 20
2
1 26
1 20
2
3
1 91
3

[출력]
20
20

연습문제: 큐를 이용한 탐색(토마토, KOI 기출)

- M x N 칸이 있는 상자에 토마토가 들어있다. 익은 토마토와 인접한(좌우상하) 토마토는 하루가 지나면 익는다. 자연적으로 익는 경우가 없다고 가정할 때 모든 토마토가 익으려면 몇 일이 걸리는가? 0 안 익은 토마토, 1 익은 토마토, -1 빈 칸

[입력]
6 4
0 0 0 0 0 0
0 0 0 0 0 0
0 0 0 0 0 0
0 0 0 0 0 1

[출력]
8

[입력]
6 4
0 -1 0 0 0 0
-1 0 0 0 0 0
0 0 0 0 0 0
0 0 0 0 0 1

[출력]
-1