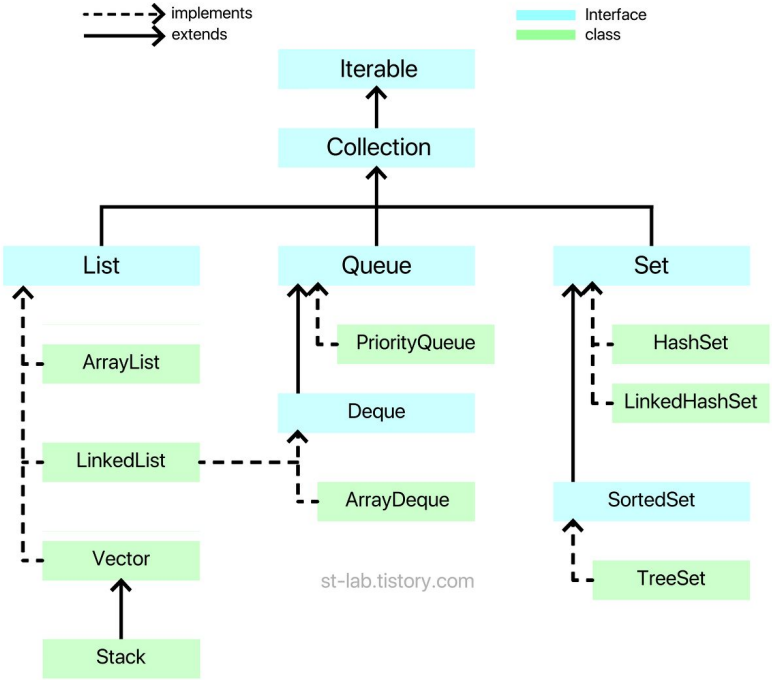


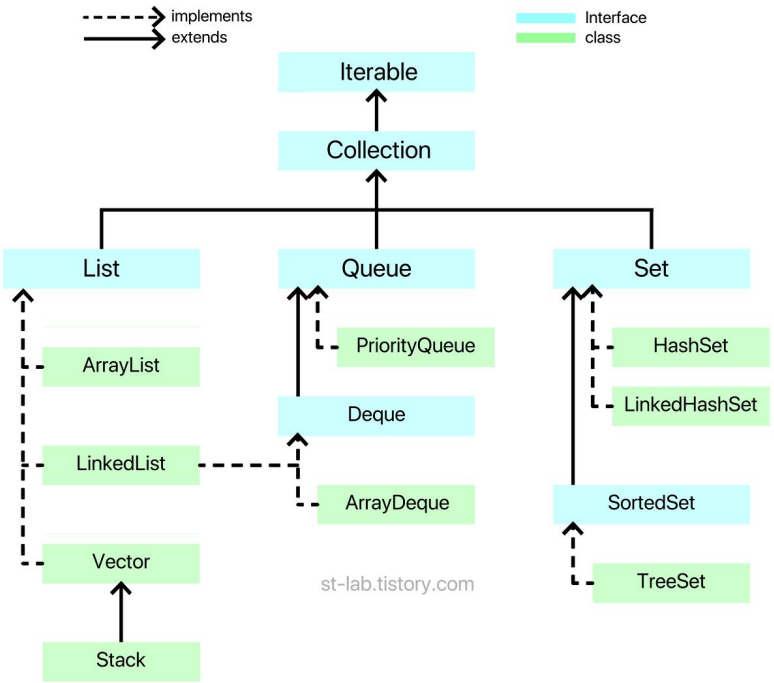
1. ‘class나 Interface 이름’ = new Class

```
List<String> a = new ArrayList<>();
```

오른쪽은 인터페이스가 올 수 없음.

```
List<String> a = new List<>(); => x
```

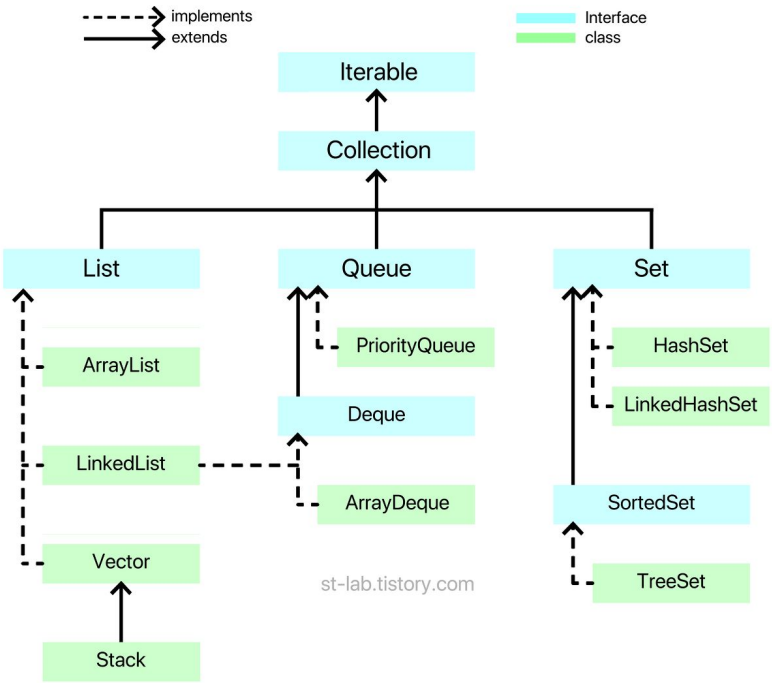
```
Queue<String> b = new ArrayDeque<>();
Queue<String> c = new LinkedList<>();
```



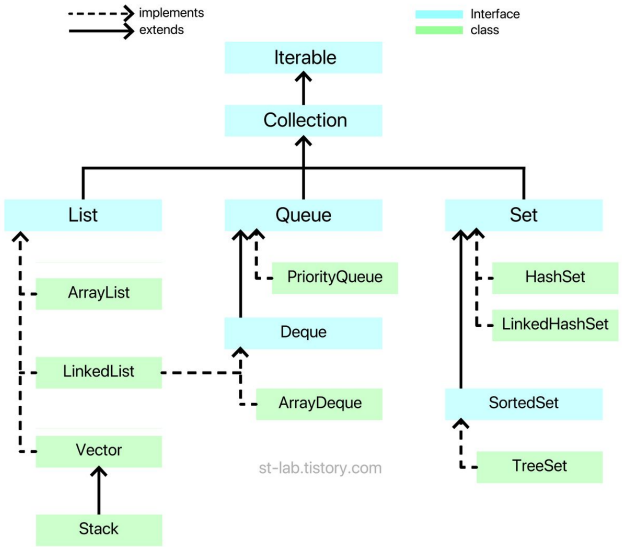

```java
List<String> a = new ArrayList<>();
a.add("sfd");
a.add(0, "afd");


Collection<String> d = new ArrayList();
d.add("sfd");
d.add(0, "afd");
```

둘다 **ArrayList**지만, **d**는 **Collection** 인터페이스에 들어있다.

Collection에는 **add(String)**이라는 메소드는 있지만,
add(int, String)는 없다.




```java
List<String> a = new ArrayList<>();
a.add("sfd");
a.add(0, "afd");


Collection<String> d = new ArrayList();
d.add("sfd");
d.add(0, "afd");


이렇게 하면 작동되긴 함.
ArrayList<String> e = (ArrayList)d;
e.add(0, "SFe");
```



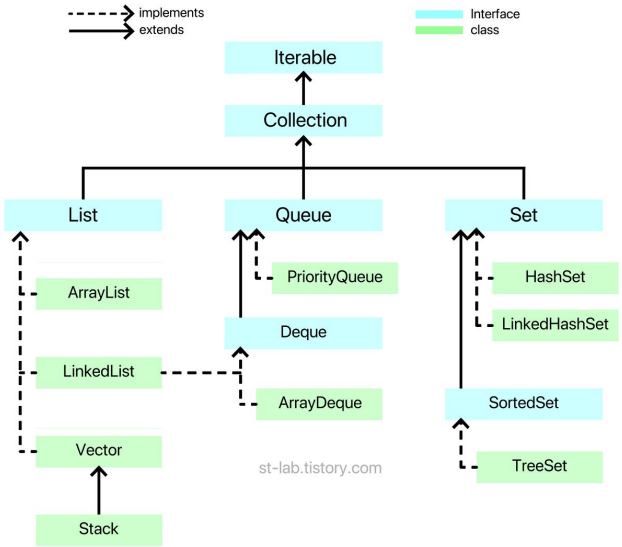

```java
List<String> a = new ArrayList<>();
a.add("sfd");
a.add(0, "afd");
```

↓

```java
List<String> a = new LinkedList<>();
a.add("sfd");
a.add(0, "afd");
```

`ArrayList`를 `LinkedList`로 바꿔 내부적으로 다르게 동작하지만

`a`를 다른 곳에 쓰고 있었어도 `List`의 메소드만 사용했기 때문에 컴파일 문제가 생기지 않음.




```java
ArrayList<String> a = new ArrayList<>();
a.add("sfd");
a.add(0, "afd");
```

↓

```java
LinkedList<String> b = new LinkedList<>();
a.add("sfd");
a.add(0, "afd");
```

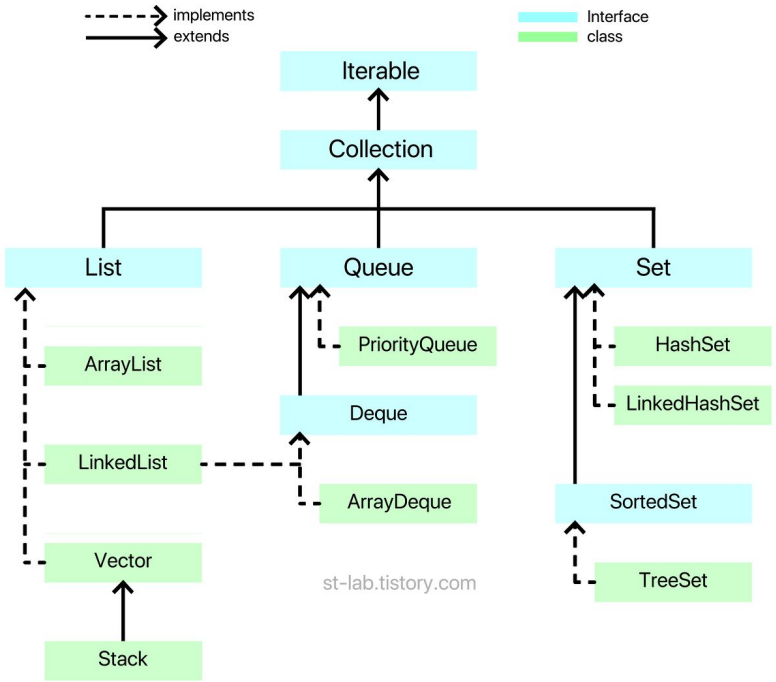
만약 다른곳에서 a를 쓰고 있었고,
a를 통해 ArrayList만 있는 메소드를 쓰고 있었으면?

그 부분도 다시 고쳐야 된다.

List라는 인터페이스에는 존재하지 않는 ArrayList만의 메소드를 쓸일은 거의 없으니

List<String> a = new ArrayList<>(); 이렇게 하자는게 교수님 의견?

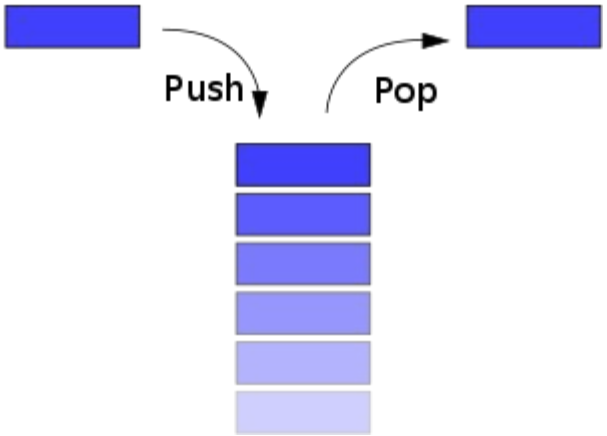
물론 다른 자료형까지 '인터페이스 = new 자료형' 고집하자는 소리는 아닐듯




```
import java.util.ArrayDeque;
import java.util.ArrayList;
import java.util.Collection;
import java.util.List;
```

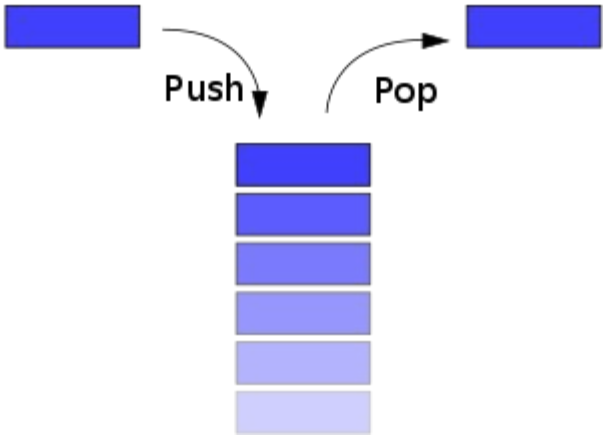
=> 귀찮으면 `import java.util.*;`

1. Stack

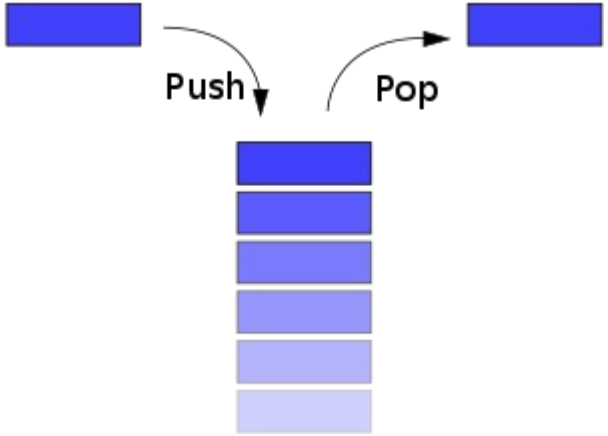


- 나중에 들어온 데이터가, 먼저 나간다.
  LIFO(Last In First Out) 구조
- DFS 알고리즘에서 재귀나 Stack을 씀
- 알고리즘 문제에서 재귀는 Stack으로 100% 대체 가능.

1. Stack



- 나중에 들어온 데이터가, 먼저 나간다.
  LIFO(Last In First Out) 구조
- DFS 알고리즘에서 재귀나 Stack을 씀
- 재귀랑 Stack은 서로 대체 가능
  (재귀 자체가 Stack임).

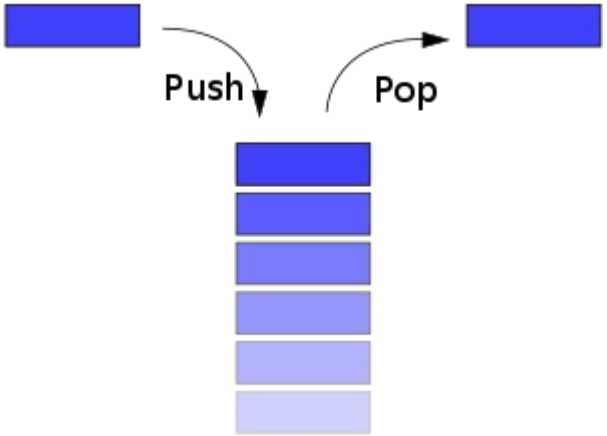


{3, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0}

{3, 5, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0}

{3, 5, 7, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0}

```
int size = 0
int stack = new int[20];

void push(data){
      stack[size++] = data;
}
int pop(){
      stack[size] = 0;
      return stack[size--]
}


add(3); // stack[0] = 3, size : 0 -> 1

add(5); // stack[1] = 5, size : 1 -> 2

add(7); // stack[2] = 7, size : 2 -> 3
```

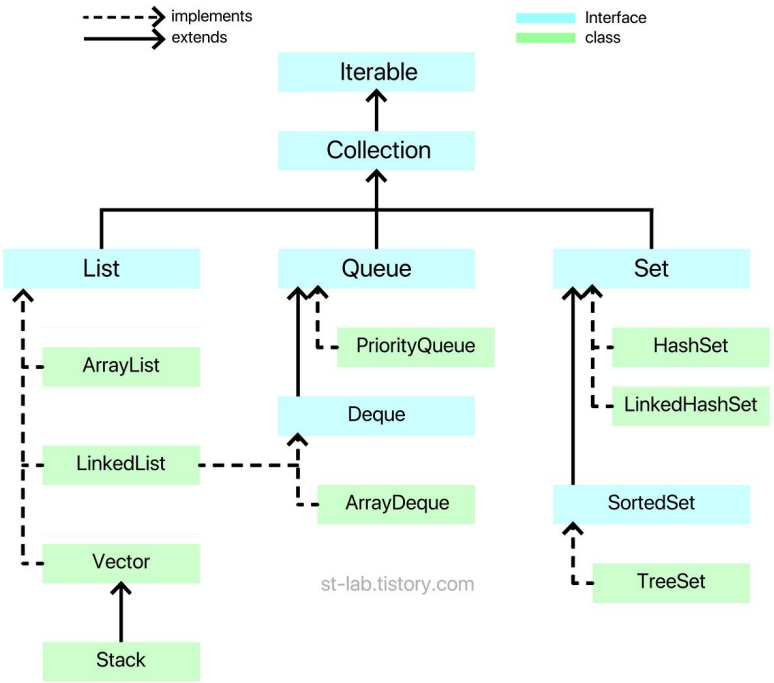


{3, 5, 7, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0}

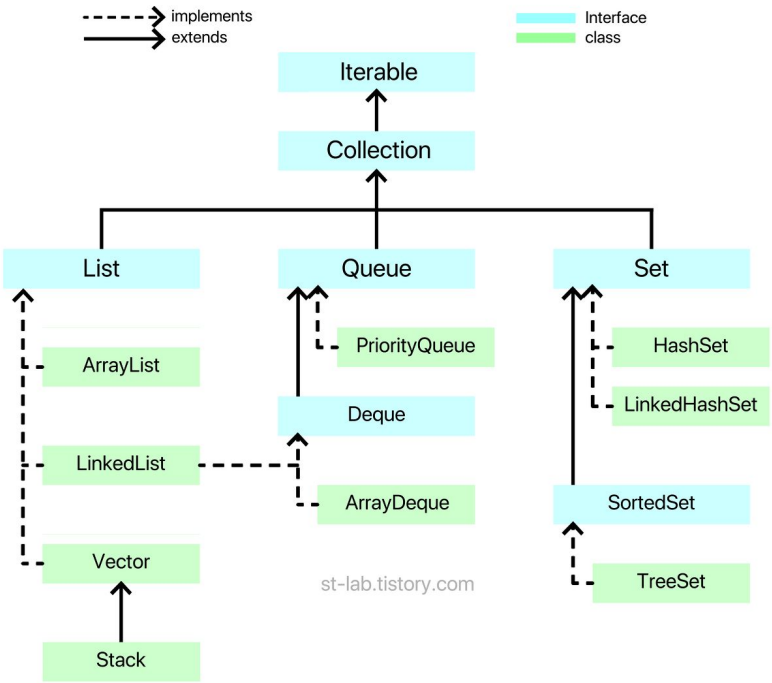
{3, 5, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0}

{3, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0}

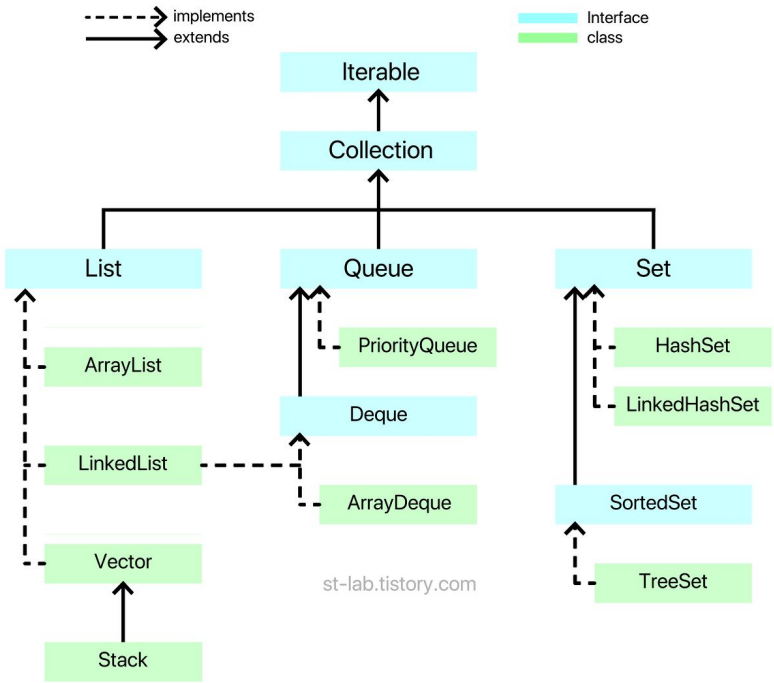
```
int size = 0
int stack = new int[20];

void push(data){
      stack[size++] = data;
}
int pop(){
      stack[size] = 0;
      return stack[size--]
}


int b = pop(); // b= 7

pop();

pop();
```



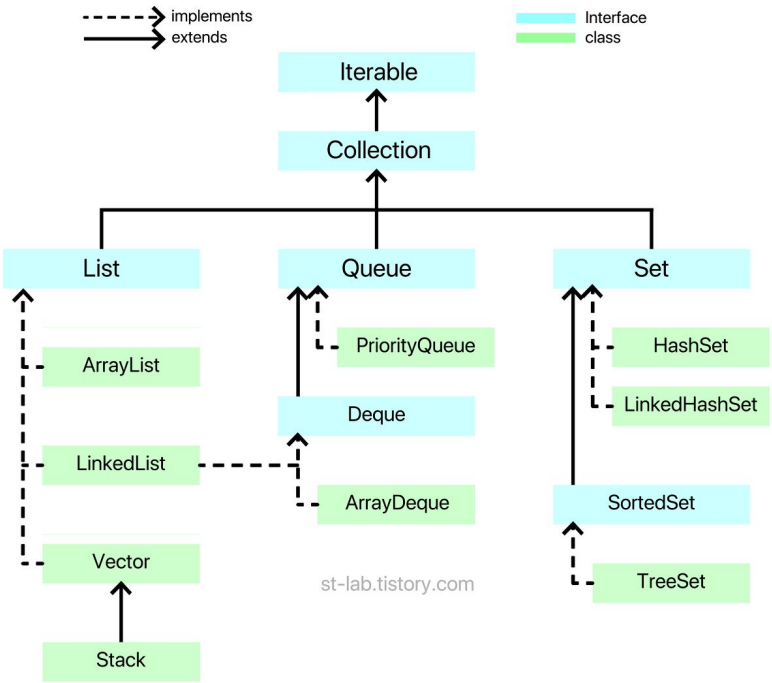

```java
import java.util.Stack;

Stack<Integer> stack = new Stack<>(); //int형 스택 선언
Stack<String> stack = new Stack<>(); //char형 스택 선언

Stack<Integer> stack = new Stack<Integer>();
Stack<Integer> stack = new Stack<Double>(); // => x

Stack<int> stack = new Stack<>(); // => class만 가능


int, double, char

Integer, Double, Character
```



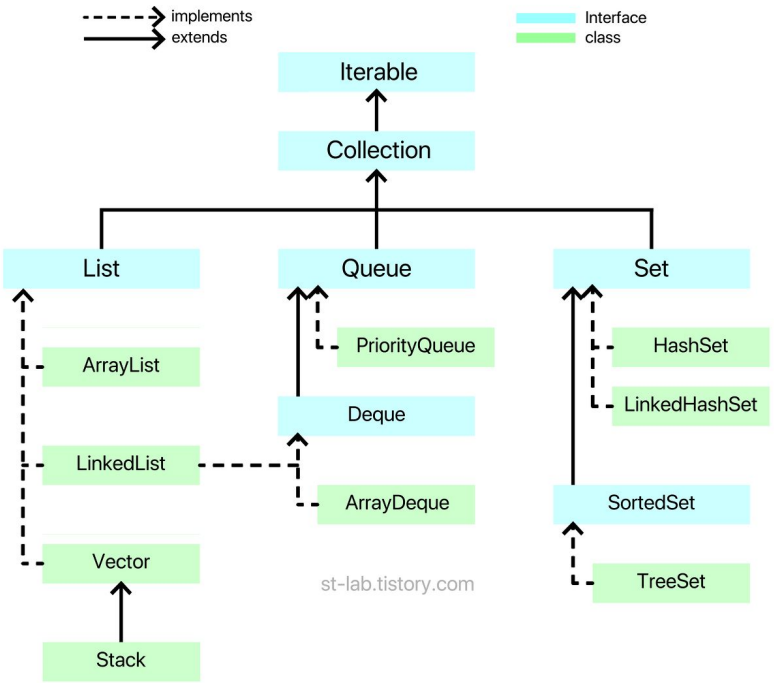

```java
Stack<int> stack = new Stack<>();
stack.push(3);      // stack에 값 1 추가
stack.push(5);      // stack에 값 2 추가
stack.push(7);
stack.push(9);

// stack 제일 위의 값 제거, result = 9
int result = stack.pop();

// stack의 가장 상단의 값 출력,result = 7
result = stack.peek();


// stack안에 해당 개체가 있으면 해당 index를 반환
// index는 제일 위가 1
// 없으면 -1 반환
int result = stack.search(3);
```



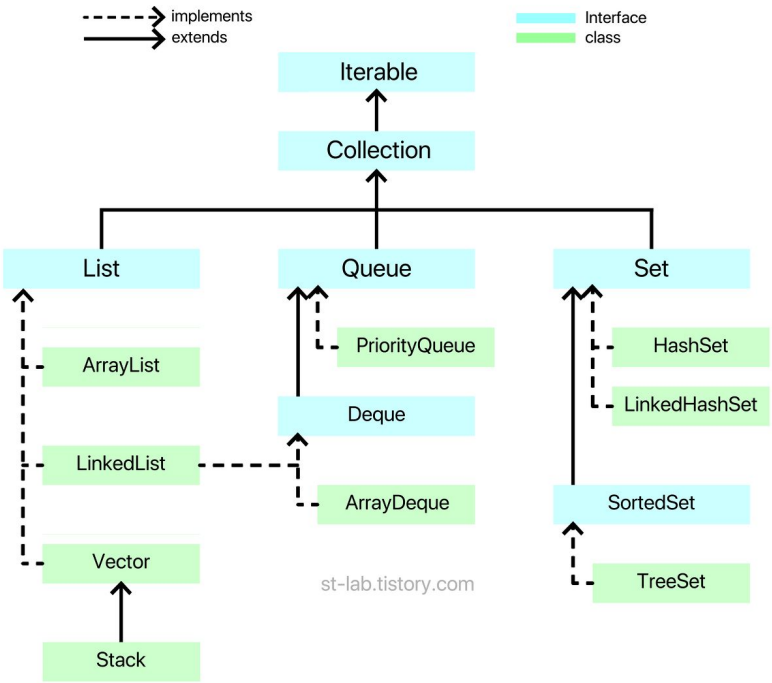

```java
Stack<int> stack = new Stack<>();


stack.size();      // stack의 크기 출력 : 0
stack.empty();     // stack이 비어있는제 check (비어있다면 true)
stack.contains(1) // stack에 1이 있는지 check (있다면 true)
stack.clear();
Integer[] stackArray = stack.toArray(new Integer[0]);
```



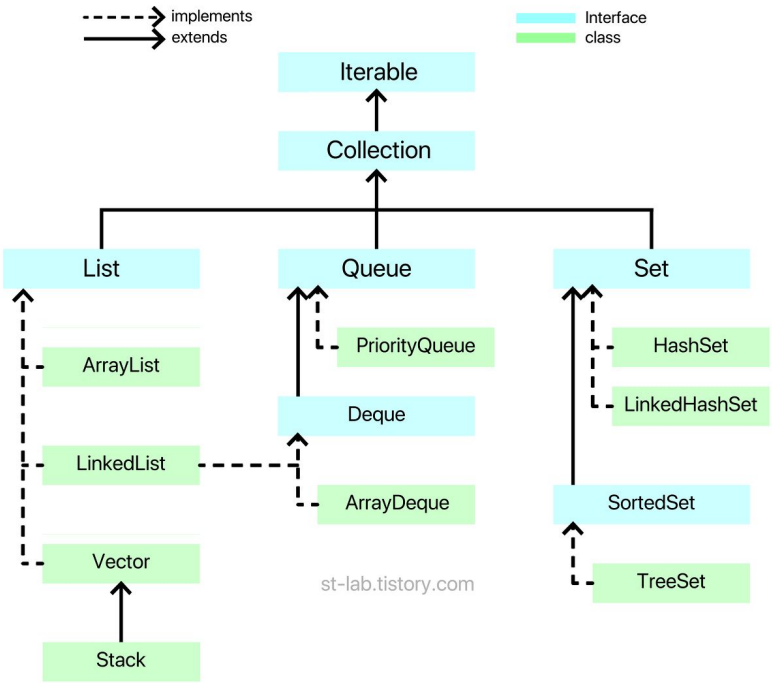

```
Collection

add(Object)
remove(Object)
removeAll(Collection)
retainAll(Collection)
contains(Object) => true, false
containsAll(Collection)
size()
isEmpty()
toArray(new Array[])
equals(Object)
clear()


List
get(int)
set(int, Object)
```




```java
List<Integer> a = new ArrayList<>();
a.add(3); // 3
a.add(5); // 3, 5
a.add(6); // 3, 5, 6
a.add(7); // 3, 5, 6, 7

int size = a.size(); // 4
boolean t1 = a.contains(3);     //true
boolean t2 = a.containsAll(Arrays.asList(3,6,5,7)); // true
a.clear();
boolean t3 = a.isEmpty(); // true

a.add(9);
a.add(6);
a.add(3);
a.add(2);

// a.remove(3)은 다른 결과
a.remove(new Integer(3));
a.removeAll(Arrays.asList(1,3));

// 6,9만 남김
a.retainAll(Arrays.asList(6,9));
Integer[] arr = a.toArray(new Integer[0]);

ArrayList<Integer> b = new ArrayList<>();
boolean t4 = a.equals(b); // false
```



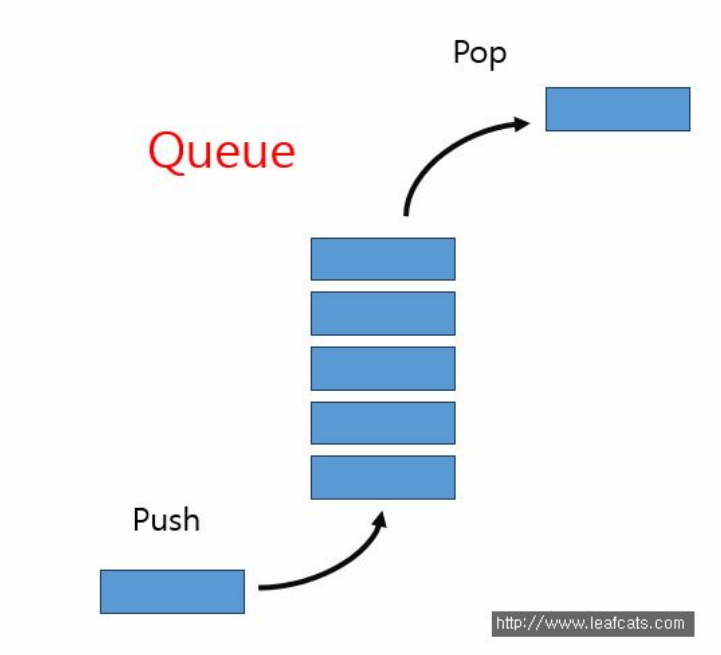

```java
Stack<int> stack = new Stack<>();
stack.push(3);
stack.add(3);

int result1 = stack.remove(stack.size()-1);
int result2 = stack.pop();

// stack의 가장 상단의 값 출력,result = 7
result1 = stack.peek();
result2 = stack.get(stack.size()-1);
```




```java
List<int> stack = new Stack<>();
stack.push(3);
stack.add(3);

int result1 = stack.remove(stack.size()-1);
int result2 = stack.pop();

// stack의 가장 상단의 값 출력,result = 7
result1 = stack.peek();
result2 = stack.get(stack.size()-1);
```

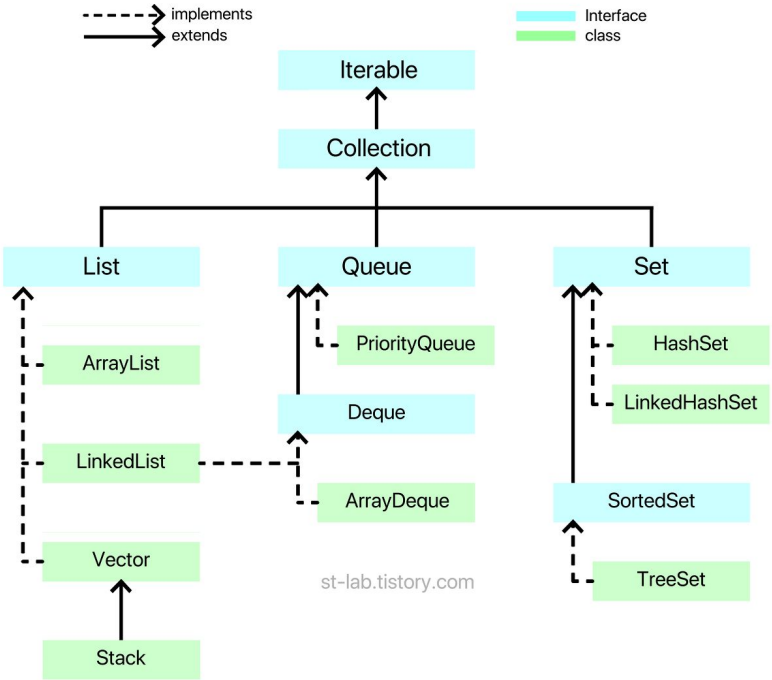
push,pop,peek는 Stack에만 있는 method
나머지는 Collection interface에 정의 되어 있고,
implements하는 방식.

2. Queue



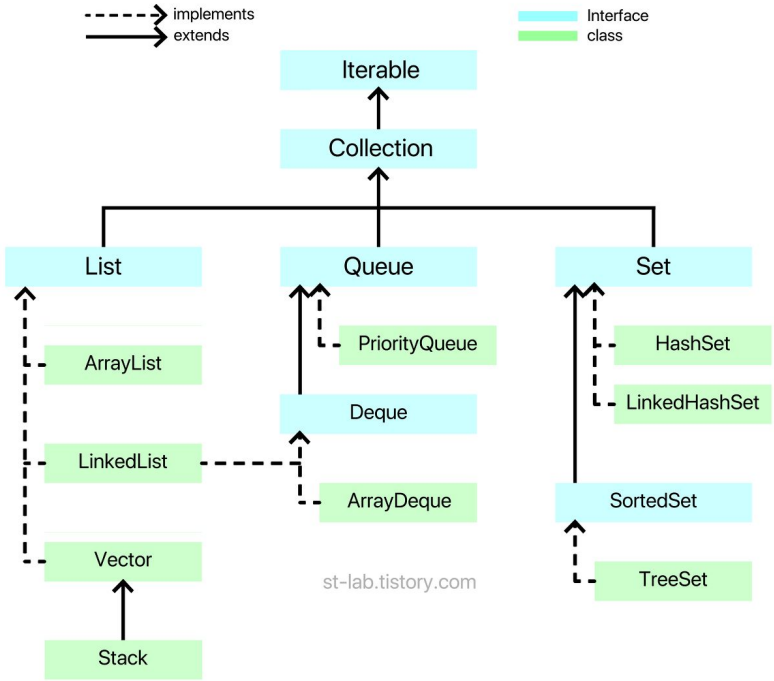
- 먼저 들어온 데이터가, 먼저 나간다.
  FIFO(First In First Out) 구조
- 심플하게 말하면 대기열.
- BFS 알고리즘 쓸 때 자주 사용.

2. Queue



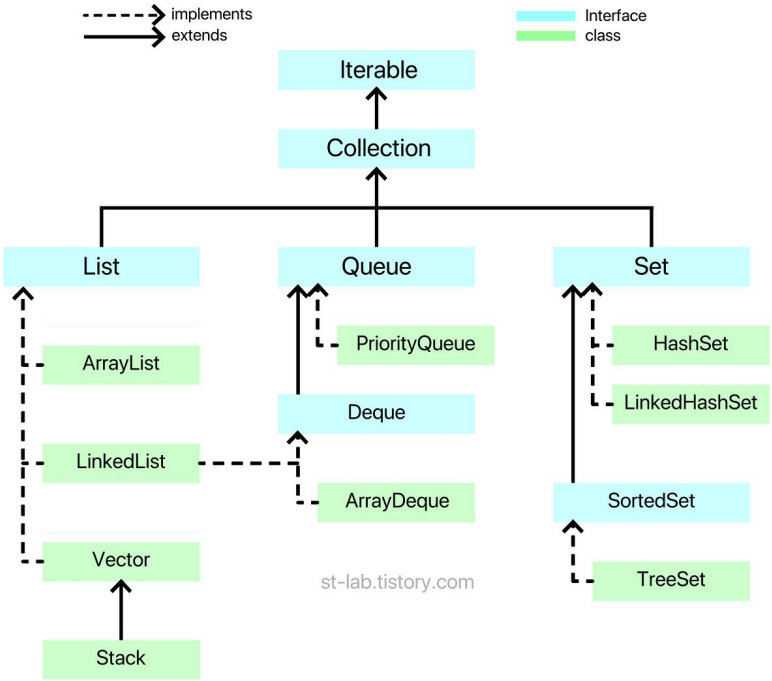

```java
import java.util.*;
// Queue 선언해보기.
```

2. Queue

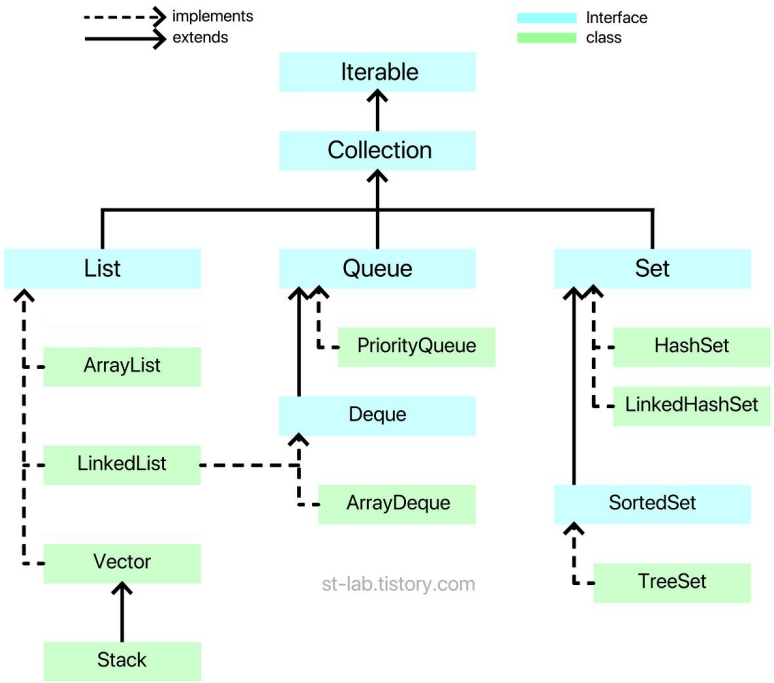



```java
import java.util.*;
// Queue 선언해보기.
Queue<Integer> queue = new PriorityQueue<>();
Queue<Integer> queue = new LinkedList<>();
Queue<Integer> queue = new ArrayDeque<>();


LinkedList<Integer> queue = new LinkedList<>();
ArrayDeque<Integer> queue = new ArrayDeque<>();



// 일반적으로 이거 2개 사용합니다!
Queue<Integer> queue = new LinkedList<>();
Queue<Integer> queue = new ArrayDeque<>();
```

2. Queue



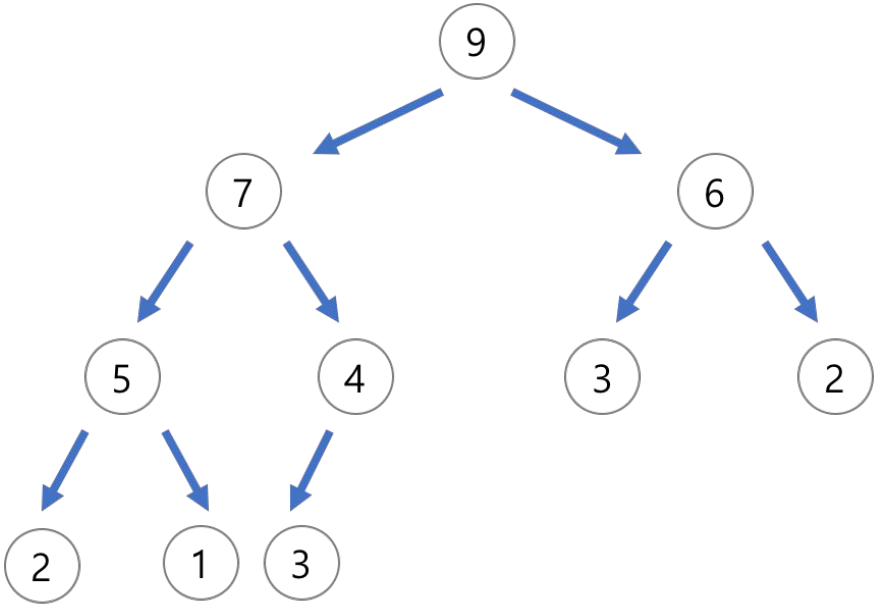

```java
import java.util.*;
Queue<String> queue = new LinkedList<>(); //int형 queue 선언

queue.offer("a");      // queue에 값 1 추가 add 대신 사용
queue.offer("b");      // queue에 값 2 추가
queue.offer("c");      // queue에 값 3 추가

// queue에 첫번째 값을 반환하고 제거 비어있다면 null
// result = a
String result = queue.poll();


// queue의 첫번째 값 참조 (element 대신 사용)
// result = b
result = queue.peek();

queue.clear();       // queue 초기화
```



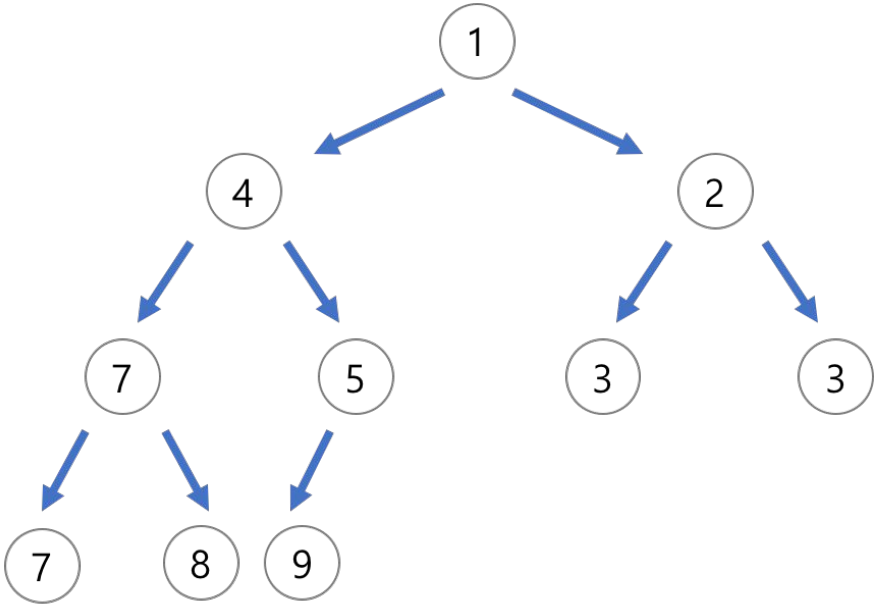

```
Collection

add(Object)
remove(Object)
removeAll(Collection)
retainAll(Collection)
contains(Object)
containsAll(Collection)
size()
isEmpty()
toArray(new Array[])
equals(Object)
clear()


Queue

offer(Object)
element()
peek()
poll()
remove() // 위에 remove랑 뭐가 다를까요?
```

3. Heap

삽입, 삭제가 O(logn)으로 일정한 이진트리

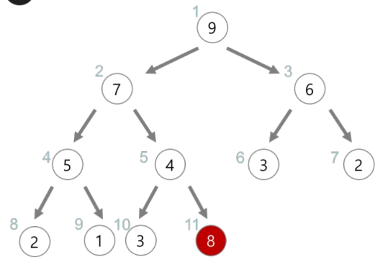


-최대 힙(max heap)-

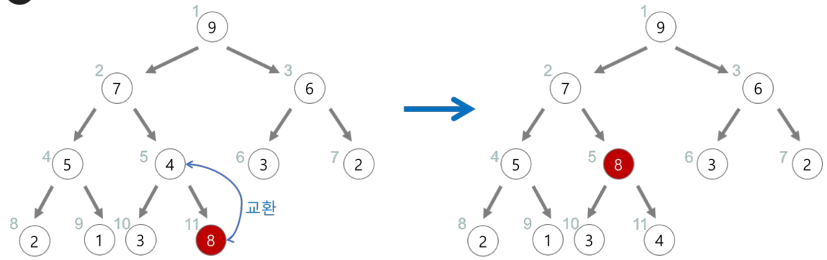


-최소 힙(min heap)-

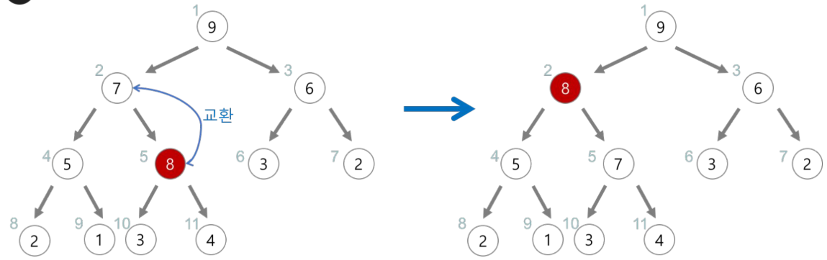
1 인덱스순으로 가장 마지막 위치에 이어서 새로운 요소 8을 삽입



2 부모 노드 4 < 삽입 노드 8 이므로 서로 교환

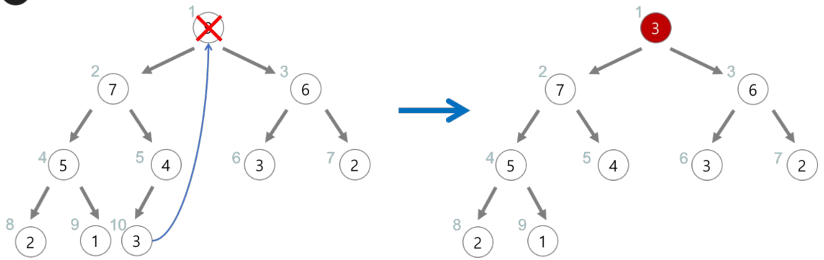


3 부모 노드 7 < 삽입 노드 8 이므로 서로 교환

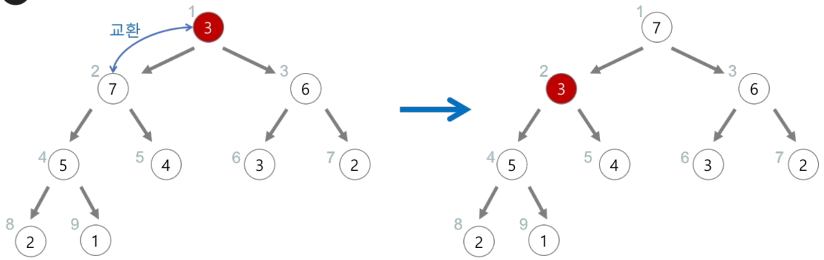


4 부모 노드 9 > 삽입 노드 8 이므로 더 이상 교환하지 않는다.

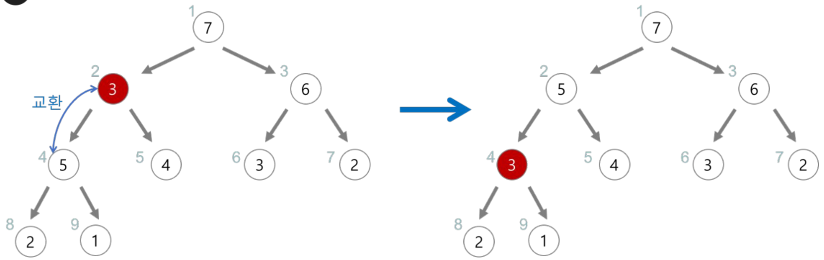
1 최댓값인 루트 노드 9를 삭제. (빈자리에는 최대 힙의 마지막 노드를 가져온다.)



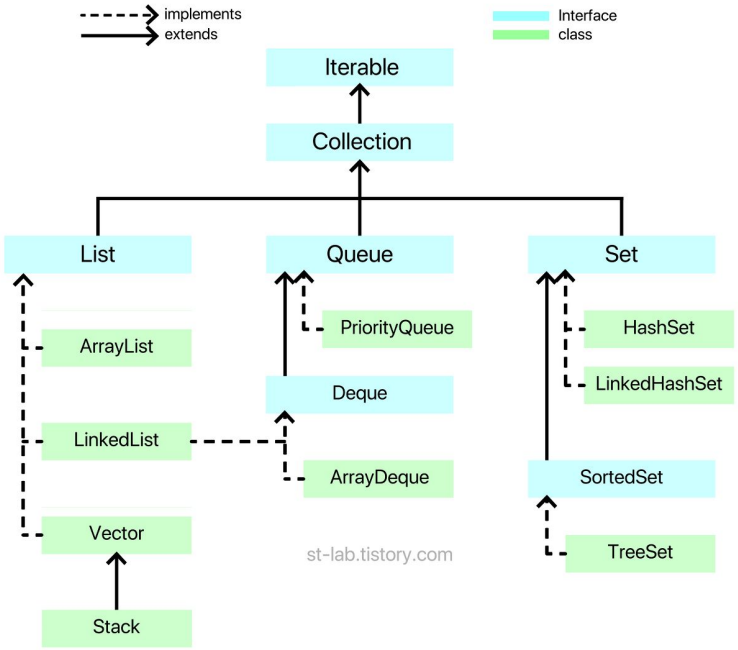
2 삽입 노드와 자식 노드를 비교. 자식 노드 중 더 큰 값과 교환. (자식 노드 7 > 삽입 노드 3 이므로 서로 교환)



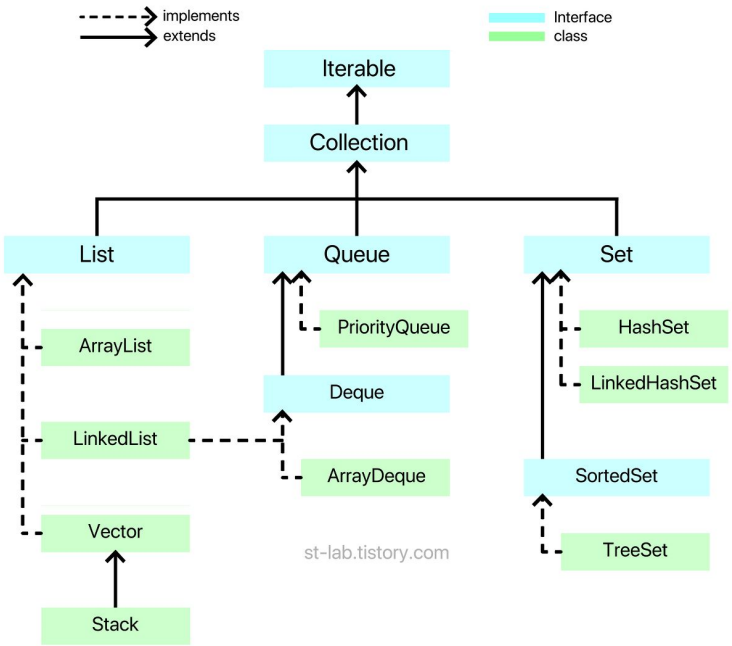
3 삽입 노드와 더 큰 값의 자식 노드를 비교. 자식 노드 5 > 삽입 노드 3 이므로 서로 교환



4 자식 노드 1, 2 < 삽입 노드 3 이므로 더 이상 교환하지 않는다.

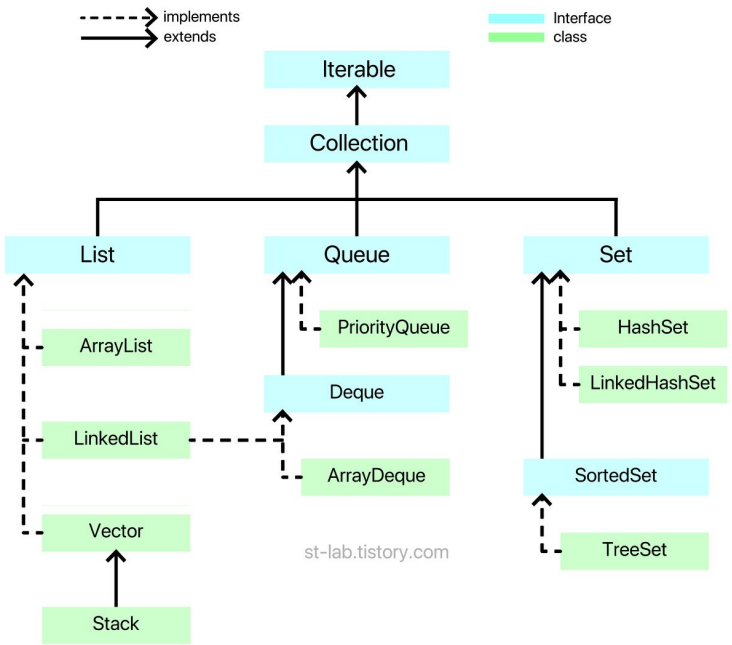



```java
import java.util.PriorityQueue;

//int형 priorityQueue 선언
PriorityQueue<Integer> priorityQueue = new PriorityQueue<>();


//int형 priorityQueue 선언 (우선순위가 높은 숫자 순)
PriorityQueue<Integer> priorityQueue2
          = new PriorityQueue<>(Collections.reverseOrder());

priorityQueue.offer(2);     // priorityQueue에 값 2 추가
priorityQueue.offer(1);     // priorityQueue에 값 1 추가
priorityQueue.offer(3);     // priorityQueue에 값 3 추가

// result = 1
int result = priorityQueue.poll();
// priorityQueue에 첫번째 값 참조 = 1

// result = 2
result = priorityQueue.peek();
```



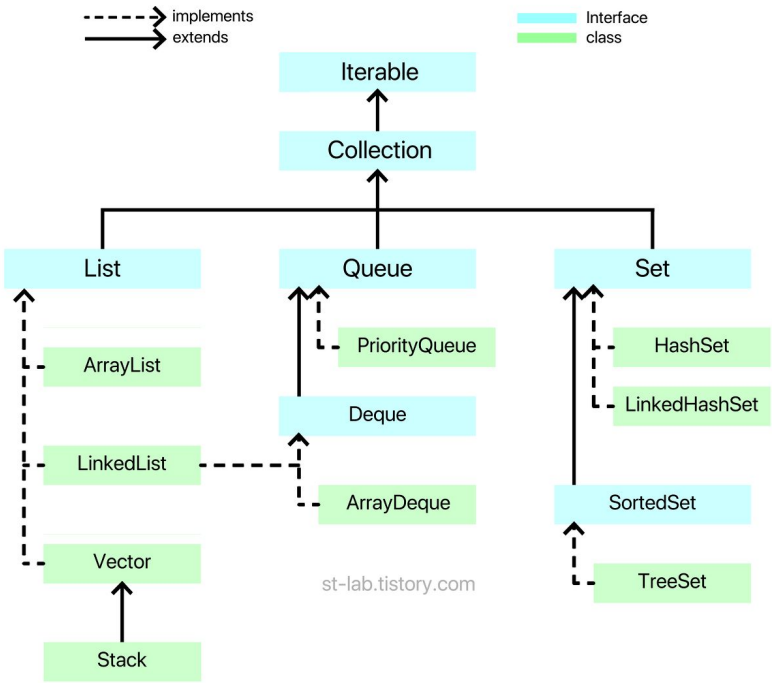
PriorityQueue인데 Queue Interface에 붙이면?

```java
import java.util.PriorityQueue;

//int형 priorityQueue 선언
Queue<Integer> priorityQueue = new PriorityQueue<>();



priorityQueue.offer(2);     // priorityQueue에 값 2 추가
priorityQueue.offer(1);     // priorityQueue에 값 1 추가
priorityQueue.offer(3);     // priorityQueue에 값 3 추가

// result = 1
int result = priorityQueue.poll();


// priorityQueue의 첫번째 값 참조 = 1
// result = 2
result = priorityQueue.peek();
```




```java
class Student {
	int num;
	String name;

	public Student(int num, String name) {
		this.num = num;
		this.name = name;
	}

	@Override
	public String toString() {
		return "Student [num=" + num + ", name=" + name + "]";
	}
}
class MyComparator implements Comparator<Student> {
	@Override
	public int compare(Student o1, Student o2) {
		return o1.num - o2.num;
	}
}
PriorityQueue<Student> pq
		= new PriorityQueue<Student>(new MyComparator());

pq.offer(new Student(4, "강동원"));
pq.offer(new Student(1, "송중기"));
pq.offer(new Student(3, "현빈"));
pq.offer(new Student(2, "유연석"));
System.out.println(pq.poll()); // num = 1
System.out.println(pq.poll()); // num = 2
```

```
PriorityQueue<Student> pq
= new PriorityQueue<Student>((Student o1, Student o2)-> {
     return o1.num - o2.num;
});

PriorityQueue<Student> pq2 = new PriorityQueue<Student>(
     new Comparator<Student>() {
          @Override
          public int compare(Student o1, Student o2) {
               return o1.num - o2.num;
          }
     });
```



Queue

offer(Object)
element()
peek()
poll()
remove()

PriorityQueue

comparator()

스택

1. 제로 : 실버4
https://www.acmicpc.net/problem/10773
2. 단어 뒤집기 2 : 실버3
https://www.acmicpc.net/problem/17413

큐

1. 큐 : 실4
https://www.acmicpc.net/problem/10845
2. 요세푸스 문제 : 실버 5
https://www.acmicpc.net/problem/1158

우선순위 큐

1. 국회의원 선거 : 실버5
https://www.acmicpc.net/problem/1417

고난이도
트럭 : 실버1, 큐
https://www.acmicpc.net/problem/13335
이중 우선순위 큐 : 골드5, 우선순위 큐
https://www.acmicpc.net/problem/7662