제 38강

큐와 스택

교재:p232~233

복습

컬렉션 프레임웍(Collection Framework)이란?

: 데이터를 효율적으로 다루기 위한 클래스들의 집합

- **자료구조(Data Structure)**

: 자료들을 저장할 때, 효율적인 구조로 저장하는 것(데이터의 구조)

- **컬렉션프레임워크**

: 자바에서 제공하는 자료구조

복습

<컬렉션 프레임워크의 종류>

- **Set 인터페이스**: 집합, 중복된 데이터를 갖지 않으며 저장 순서를 유지하지 않는 자료구조

→ 대표 클래스: HashSet, TreeSet

- **List 인터페이스**: 데이터를 일렬로 늘어놓은 구조, 중복을 허용하고 저장 순서를 유지

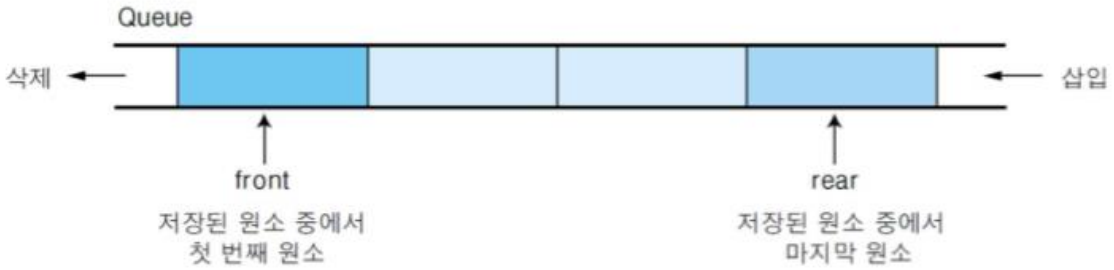
→ 대표 클래스: ArrayList, LinkedList

목차

1. Queue

Queue(큐)?

: 한 쪽 끝에서는 삽입이 다른 쪽 끝에서는 삭제가 일어나는 구조



큐의 특징

- 선입선출(FIFO) 의 구조
- front: 삭제, rear: 삽입

1. Queue

[큐와 관련된 메서드]

메서드	설명
boolean add(E e)	Queue에 요소 e 추가
E element( )	Queue의 제일 상단 요소 반환
E remove( )	Queue의 최상단 요소를 반환 후 제거
boolean offer(E e)	Queue에 요소 e를 추가
E peek( )	Queue의 제일 상단 요소 반환
E poll( )	Queue의 최상단 요소를 반환 후 제거

→ Queue 는 인터페이스 이므로, 사용하려면 업 캐스팅 필요!

1. Queue

[큐 사용법]

1) Queue<Element> q = new LinkedList<Element>( ); → 주로 사용하는 방법
2) Queue <Element> q= new Array <Element>( );

1. Queue

<실습> QueueEx.java

```
public class QueueEx {
    public static void main(String[] args) {

            Queue<String> q = new LinkedList<String>();
            q.add("apple");
            q.add("banana");
            q.add("cherry");

            System.out.println(q);

            System.out.println("peek:"+q.peek());
            System.out.println("remove:"+q.remove());
            q.offer("tomato");
            System.out.println(q);
            System.out.println("poll:"+q.poll());
            System.out.println(q);
    }
}
```

<실행 결과>

```
[apple, banana, cherry]
peek:apple
remove:apple
[banana, cherry, tomato]
poll:banana
[cherry, tomato]
```

1. Queue

<실습> QueueEx.java

```
public class QueueEx {
    public static void main(String[] args) {

            Queue<String> q = new LinkedList<String>();
            q.add("apple");
            q.add("banana");
            q.add("cherry");

            System.out.println(q);

            System.out.println("peek:"+q.peek());
            System.out.println("remove:"+q.remove());
            q.offer("tomato");
            System.out.println(q);
            System.out.println("poll:"+q.poll());
            System.out.println(q);
    }
}
```

<명령에 따른 큐의 구조 변화>

```
q.add("apple");
q.add("banana");
q.add("cherry");
```

peek: 최 상단 요소(front) 반환

```
System.out.println("peek:"+q.peek());
```

```
peek:apple
```

1. Queue

<실습> QueueEx.java

```
public class QueueEx {
    public static void main(String[] args) {

            Queue<String> q = new LinkedList<String>();
            q.add("apple");
            q.add("banana");
            q.add("cherry");

            System.out.println(q);

            System.out.println("peek:"+q.peek());
            System.out.println("remove:"+q.remove());
            q.offer("tomato");
            System.out.println(q);
            System.out.println("poll:"+q.poll());
            System.out.println(q);
    }
}
```

<명령에 따른 큐의 구조 변화>

```
System.out.println("remove:"+q.remove());
```

```
remove:apple
```

offer: queue에 요소 추가

```
q.offer("tomato");
```

1. Queue

<실습> QueueEx.java

```
public class QueueEx {
    public static void main(String[] args) {

            Queue<String> q = new LinkedList<String>();
            q.add("apple");
            q.add("banana");
            q.add("cherry");

            System.out.println(q);

            System.out.println("peek:"+q.peek());
            System.out.println("remove:"+q.remove());
            q.offer("tomato");
            System.out.println(q);
            System.out.println("poll:"+q.poll());
            System.out.println(q);
    }
}
```

<명령에 따른 큐의 구조 변화>

pool : Queue의 최상단 요소 반환 후 제거

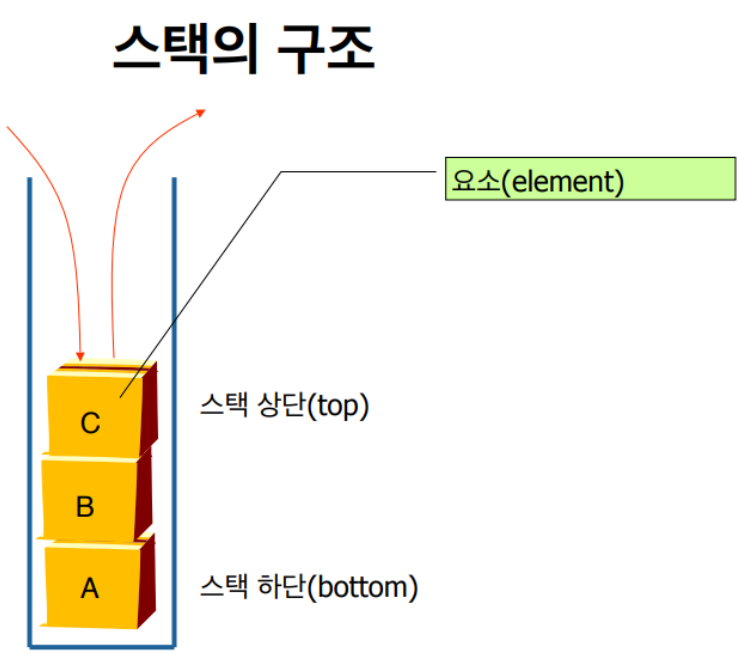
```
System.out.println("poll:"+q.poll());
```


```
poll:banana
```

2. Stack

Stack(스택)?

: 한 쪽 끝에서 삽입과 삭제가 모두 일어나는 구조



스택의 특징

- 후입선출(LIFO) 의 구조
- top: 삽입, 삭제
- Vector의 자식 클래스

```
public class Stack<E> extends Vector<E> {
    /**
     * Creates an empty Stack.
     */
    public Stack() {
    }
```

2. Stack

[스택과 관련된 메서드]

메서드	설명
E peek( )	top 요소를 반환
E pop( )	top 요소 제거 후 반환
E push(E item)	top에 요소 추가
int search(Object O)	요소 O가 있는지 검색해서 위치 반환

→ Stack 은 클래스이므로, 바로 사용 가능

2. Stack

<실습> StackEx.java

```
public class StackEx {
    public static void main(String[] args) {
            Stack<String> s = new Stack<String>();

            s.push("apple");
            s.push("banana");
            s.push("cherry");
            System.out.println(s);

            System.out.println("peek: "+s.peek());
            System.out.println("pop: "+s.pop());
            System.out.println(s);
            s.push("tomato");
            System.out.println(s);
            System.out.println("search(apple):"+s.search("apple"));
            System.out.println("search(banana):"+s.search("banana"));
            System.out.println("search(tomato):"+s.search("tomato"));
    }
}
```

<실행 결과>

```
[apple, banana, cherry]
peek: cherry
pop: cherry
[apple, banana]
[apple, banana, tomato]
search(apple):3
search(banana):2
search(tomato):1
```

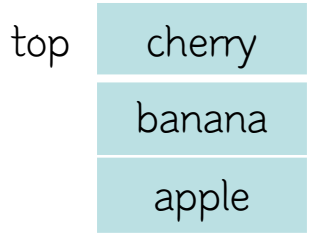
2. Stack

<실습> StackEx.java

```
public class StackEx {
    public static void main(String[] args) {
            Stack<String> s = new Stack<String>();

            s.push("apple");
            s.push("banana");
            s.push("cherry");
            System.out.println(s);

            System.out.println("peek: "+s.peek());
            System.out.println("pop: "+s.pop());
            System.out.println(s);
            s.push("tomato");
            System.out.println(s);
            System.out.println("search(apple):"+s.search("apple"));
            System.out.println("search(banana):"+s.search("banana"));
            System.out.println("search(tomato):"+s.search("tomato"));
    }
}
```

<명령에 따른 스택의 구조 변화>

```
s.push("apple");
s.push("banana");
s.push("cherry");
```



peek: top 요소 반환

```
peek: cherry
```

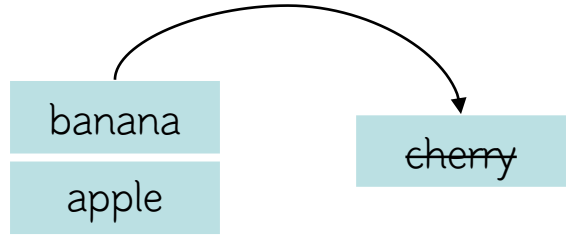
2. Stack

<실습> StackEx.java

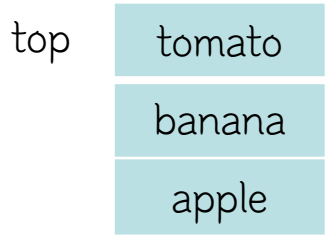
```
public class StackEx {
    public static void main(String[] args) {
            Stack<String> s = new Stack<String>();

            s.push("apple");
            s.push("banana");
            s.push("cherry");
            System.out.println(s);

            System.out.println("peek: "+s.peek());
            System.out.println("pop: "+s.pop());
            System.out.println(s);
            s.push("tomato");
            System.out.println(s);
            System.out.println("search(apple):"+s.search("apple"));
            System.out.println("search(banana):"+s.search("banana"));
            System.out.println("search(tomato):"+s.search("tomato"));
    }
}
```

<명령에 따른 스택의 구조 변화>

```
System.out.println("pop: "+s.pop());
```




```
s.push("tomato");
```



2. Stack

<실습> StackEx.java

```
public class StackEx {
    public static void main(String[] args) {
            Stack<String> s = new Stack<String>();

            s.push("apple");
            s.push("banana");
            s.push("cherry");
            System.out.println(s);

            System.out.println("peek: "+s.peek());
            System.out.println("pop: "+s.pop());
            System.out.println(s);
            s.push("tomato");
            System.out.println(s);
            System.out.println("search(apple):"+s.search("apple"));
            System.out.println("search(banana):"+s.search("banana"));
            System.out.println("search(tomato):"+s.search("tomato"));
    }
}
```

<명령에 따른 스택의 구조 변화>

```
System.out.println("search(apple):"+s.search("apple"));
System.out.println("search(banana):"+s.search("banana"));
System.out.println("search(tomato):"+s.search("tomato"));
```

```
search(apple):3
search(banana):2
search(tomato):1
```