

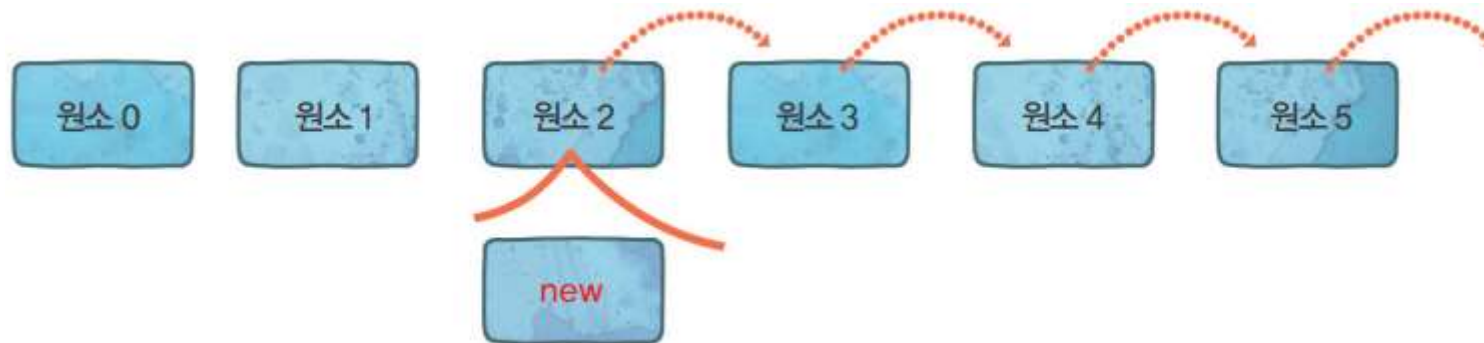
# 컬렉션 프레임워크

예제 소스 코드는 파일과 연결되어 있습니다.  
editplus(유료), notepad++(무료)와 같은 편집 도구를  
미리 설치하여 PPT를 슬라이드 쇼로 진행할 때 소스  
파일과 연결하여 보면 강의하실 때 편리합니다.

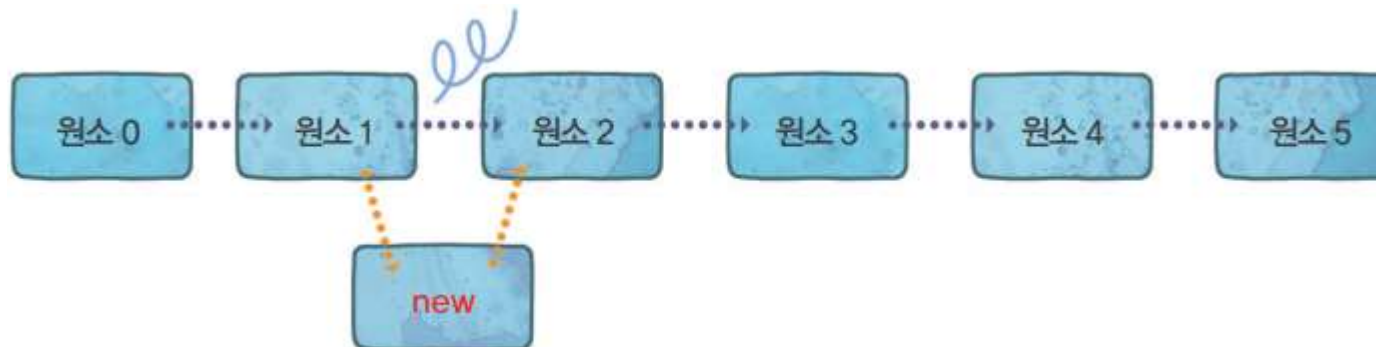
# 컬렉션 프레임워크 기초

## ■ 필요성

- 유사한 객체를 여러 개 저장하고 조작해야 할 때가 빈번
- 고정된 크기의 배열의 불편함



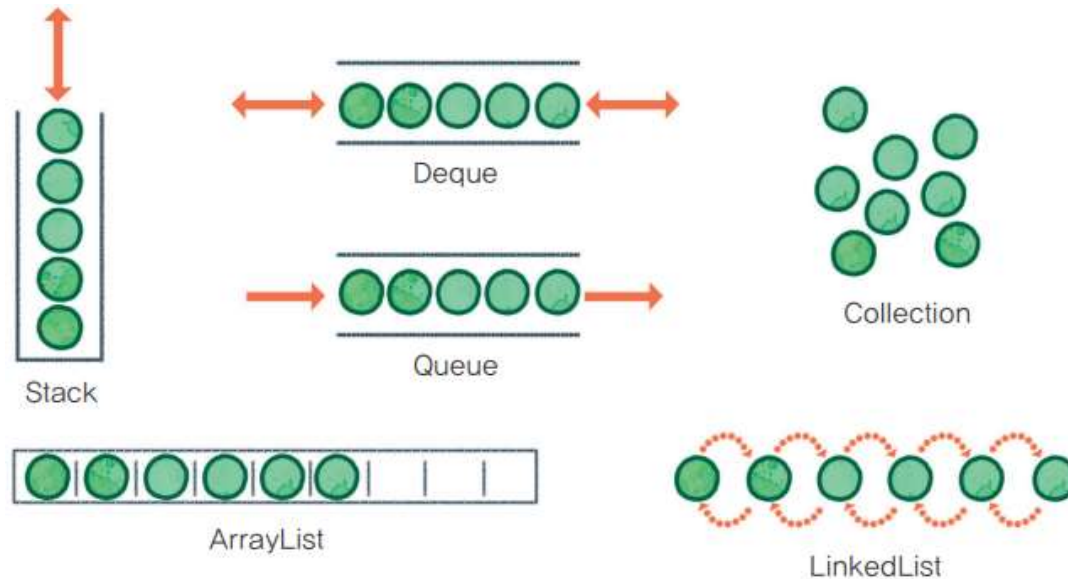
- 연결 리스트를 사용하면



# 컬렉션 프레임워크 기초

## ■ 의미

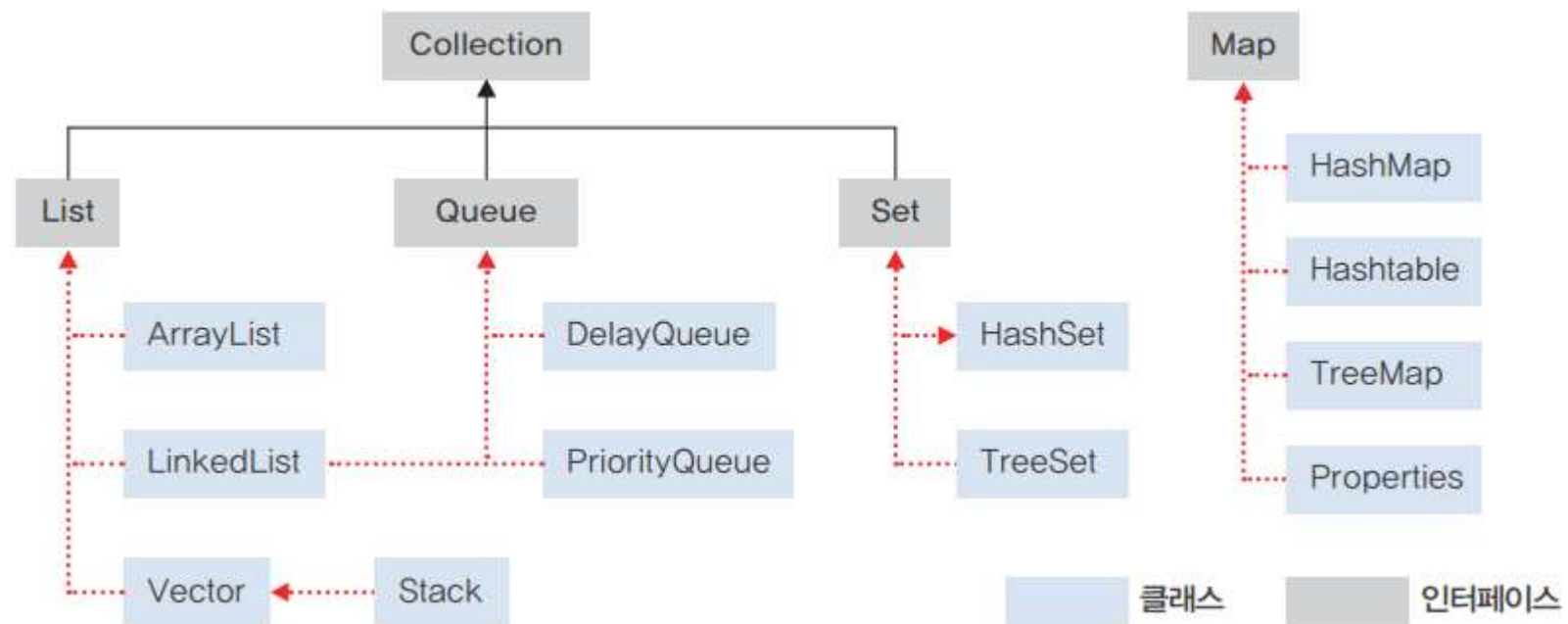
- 유사한 객체의 집단을 효율적으로 관리할 수 있도록 컬렉션 프레임워크를 제공
- 컬렉션 : 데이터를 한곳에 모아 편리하게 저장 및 관리하는 가변 크기의 객체 컨테이너
- 컬렉션 프레임워크 : 객체를 한곳에 모아 효율적으로 관리하고 편리하게 사용할 수 있도록 제공하는 환경



# 컬렉션 프레임워크 기초

## ■ 구조

- 컬렉션 프레임워크는 인터페이스와 클래스로 구성
- 인터페이스는 컬렉션에서 수행할 수 있는 각종 연산을 제네릭 타입으로 정의해 유사한 클래스에 일관성 있게 접근하게 함
- 클래스는 컬렉션 프레임워크 인터페이스를 구현한 클래스
- java.util 패키지에 포함 (DelayQueue는 java.util.concurrent 패키지에 포함)



# Collection 인터페이스

## ■ Collection 인터페이스와 구현 클래스

인터페이스		특징	구현 클래스
Collection	List	객체의 순서가 있고, 원소가 중복될 수 있다.	ArrayList, Stack, Vector, LinkedList
	Queue	객체를 입력한 순서대로 저장하며, 원소가 중복될 수 있다.	DelayQueue, PriorityQueue, LinkedList
	Set	객체의 순서가 없으며, 동일한 원소를 중복할 수 없다.	HashSet, TreeSet, EnumSet

# Collection 인터페이스

## ■ Collection 인터페이스

### ● 주요 메서드

메서드	설명
<code>boolean add(E e)</code>	객체를 맨 끝에 추가한다.
<code>void clear()</code>	저장된 모든 객체를 제거한다.
<code>boolean contains(Object o)</code>	명시한 객체의 저장 여부를 조사한다.
<code>boolean isEmpty()</code>	리스트가 비어 있는지 조사한다.
<code>Iterator&lt;E&gt; iterator()</code>	Iterator를 반환한다.
<code>boolean remove(Object o)</code>	명시한 첫 번째 객체를 제거하고, 제거 여부를 반환한다.
<code>int size()</code>	저장된 전체 객체의 개수를 반환한다.
<code>T[] toArray(T[] a)</code>	리스트를 배열로 반환한다.

### ● 이외에도 Collection 인터페이스는 다음과 같은 유용한 디폴트 메서드를 제공

```
default void forEach(Consumer<? super T> action)
default boolean removeIf(Predicate <? super E> filter)
default <T> T[] toArray(IntFunction<T[]> generator)
```

→ 자바 11부터

# Collection 인터페이스

## ■ 컬렉션의 반복 처리

- Collection 인터페이스는 `iterator()` 메서드를 통하여 반복자를 제공
- 키-값 구조의 Map 컬렉션은 반복자를 제공하지 않는다.
- Iterator 인터페이스가 제공하는 주요 메서드

메서드	설명
<code>boolean hasNext()</code>	다음 원소의 존재 여부를 반환한다.
<code>E next()</code>	다음 원소를 반환한다.
<code>default void remove()</code>	마지막에 순회한 컬렉션의 원소를 삭제한다.

- 예제 : [sec02/IteratorDemo](#)



# Collection 인터페이스

## ■ List 컬렉션

- 순서가 있는 객체를 중복 여부와 상관없이 저장하는 리스트 자료구조를 지원. 배열과 매우 유사하지만 크기가 가변적. 원소의 순서가 있으므로 원소를 저장하거나 읽어올 때 인덱스를 사용

메서드	설명
<code>void add(int index, E element)</code>	객체를 인덱스 위치에 추가한다.
<code>E get(int index)</code>	인덱스에 있는 객체를 반환한다.
<code>int indexOf(Object o)</code>	명시한 객체가 있는 첫 번째 인덱스를 반환한다.
<code>E remove(int index)</code>	인덱스에 있는 객체를 제거한다.
<code>E set(int index, E element)</code>	인덱스에 있는 객체와 주어진 객체를 교체한다.
<code>List&lt;E&gt; subList(int from, int to)</code>	범위에 해당하는 객체를 리스트로 반환한다.

- 디폴트 메서드

```
default void replaceAll(UnaryOperator<E> operator)
default void sort(Comparator<? super E> c)
```

- 팩토리 메서드(자바 9부터)

```
static <E> List<E> of(E... elements)
```



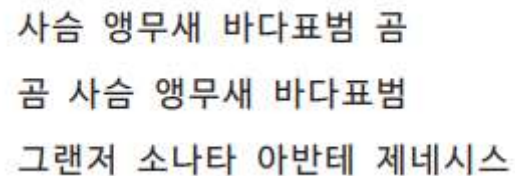
# Collection 인터페이스

## ■ List 컬렉션

- List 타입과 배열 사이에는 다음 메서드를 사용하여 상호 변환

```
public static <T> List<T> asList(T... a) // java.util.Arrays 클래스의 정적 메서드
<T> T[] toArray(T[] a)                // java.util.List 클래스의 메서드
```

- 예제 : [sec02/ListDemo](#)



```
사슴 앵무새 바다표범 곰
곰 사슴 앵무새 바다표범
그랜저 소나타 아반테 제네시스
```

- 대표적인 List 구현 클래스 : ArrayList, Vector

# Collection 인터페이스

## ■ List 컬렉션

- ArrayList 클래스 : Vector 클래스 ArrayList와 동일한 기능을 제공하지만, ArrayList와 달리 동기화된 메서드로 구현해서 스레드에 안전
- 예제 : [sec02/ArrayListDemo](#)

```
1  
false  
true  
뉴그랜저 뉴아반테 뉴제네시스 뉴싼타페  
true
```

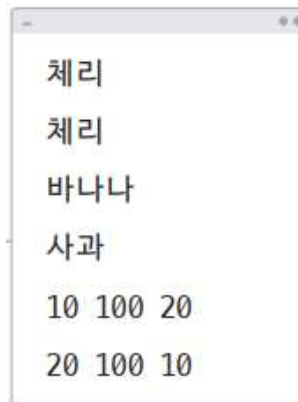
# Collection 인터페이스

## ■ List 컬렉션

- Stack 클래스 : 후입선출 방식으로 객체를 관리하며, Vector의 자식 클래스이다. 대부분의 인덱스가 0부터 시작하지만 Stack 클래스는 1부터 시작한다.

메서드	설명
boolean empty( )	스택이 비어 있는지 조사한다.
E peek( )	스택의 최상위 원소를 제거하지 않고 엿본다.
E pop( )	스택의 최상위 원소를 반환하며, 스택에서 제거한다.
E push(E item)	스택의 최상위에 원소를 추가한다.
int search(Object o)	주어진 원소의 인덱스 값(1부터 시작)을 반환한다.

- 예제 : [sec02/StackDemo](#)



```
체리
체리
바나나
사과
10 100 20
20 100 10
```

# Collection 인터페이스

## ■ List 컬렉션

### ● LinkedList 클래스

구분	ArrayList 클래스	LinkedList 클래스
구현	가변 크기 배열	이중 연결 리스트
초기 용량	10	0
get( ) 연산	빠름	느림
add( ), remove( ) 연산	느림	빠름
메모리 부담	적음	많음
Iterator	순방향	순방향, 역방향

### ● 예제 : [sec02/PerformanceDemo](#)

```
ArrayList로 처리한 시간 : 979047448  
LinkedList로 처리한 시간 : 2805415
```

# Collection 인터페이스

## ■ Queue 컬렉션

- 선입선출 방식을 지원
- 큐의 입구와 출구를 각각 후단(tail)과 전단(head)이라고 하며, 후단에서 원소를 추가하고 전단에서 원소를 제거
- 큐의 중간에 원소를 추가하거나 중간에 있는 원소를 제거할 수는 없다.
- Queue 인터페이스에 추가된 메서드

기능	예외를 던짐	null 또는 false를 반환
삽입	boolean add(E e)	boolean offer(E e)
삭제	E remove()	E poll()
검색	E element()	E peek()

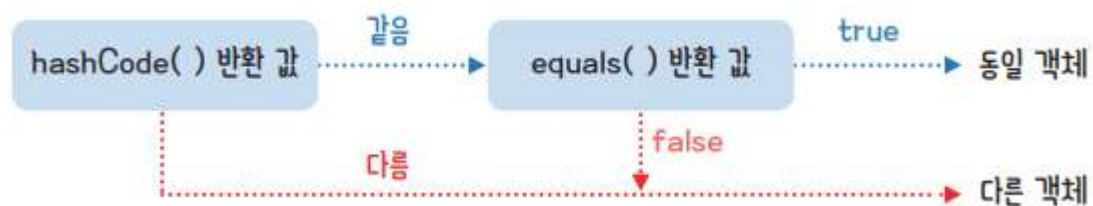
- 예제 : [sec02/QueueDemo](#)

```
null
바나나를 추가했나요? true
헤드 엿보기 : 사과
사과 제거하기
새로운 헤드 : 바나나
바나나 제거하기
새로운 헤드 : 체리
체리를 포함하고 있나요? true
사과를 포함하고 있나요? false
```

# Collection 인터페이스

## ■ Set 컬렉션

- 순서가 없으며, 중복되지 않는 객체를 저장하는 자료구조 지원
- 추가된 메서드는 없지만 중복 원소를 배제
- 인덱스가 없어 저장 순서 무시
- Set 컬렉션에 같은 객체란?



# Collection 인터페이스

## ■ Set 컬렉션

- Set 팩토리 메서드(자바 9부터)

```
static <E> Set<E> of(E... elements)
```

- 예제 : [sec02/HashSet1Demo](#), [sec02/HashSet2Demo](#)

```
1단계 : [포도, 수박, 사과, 바나나]
2단계 : [null, 수박, 사과, 바나나]
4
true
3단계 : [포도, 수박, 사과, 바나나]
true
```

- 예제 : [sec02/TreeSetDemo](#)

```
[포도, 망고, 수박, 사과, 키위]
[망고, 사과, 수박, 키위, 포도]
망고
포도
망고
수박
```

- 오버라이딩하지 않을 때 실행 결과

```
2
[Fruit(사과), Fruit(사과)]
```

- 오버라이딩할 때 실행 결과

```
1
[Fruit(사과)]
```

# Map 인터페이스

---

## ■ 특징

- 키와 값, 이렇게 쌍으로 구성된 객체를 저장하는 자료구조
- 맵이 사용하는 키와 값도 모두 객체
- 키는 중복되지 않고 하나의 값에만 매핑되어 있으므로 키가 있다면 대응하는 값을 얻을 수 있다.
- Map 객체에 같은 키로 중복 저장되지 않도록 하려면 Set 객체처럼 키로 사용할 클래스에 대한 hashCode( )와 equals( ) 메서드를 오버로딩해야 한다.
  
- 구현 클래스 : HashMap, Hashtable , TreeMap, Properties



# Map 인터페이스

## ■ 주요 메서드

- Map 인터페이스가 제공

메서드	설명
<code>void clear()</code>	모든 매핑을 삭제한다.
<code>boolean containsKey(Object key)</code>	주어진 키의 존재 여부를 반환한다.
<code>boolean containsValue(Object value)</code>	주어진 값의 존재 여부를 반환한다.
<code>Set&lt;Map.Entry&lt;K, V&gt;&gt; entrySet()</code>	모든 매핑을 Set 타입으로 반환한다.
<code>V get(Object key)</code>	주어진 키에 해당하는 값을 반환한다.
<code>boolean isEmpty()</code>	컬렉션이 비어 있는지 여부를 반환한다.
<code>Set&lt;K&gt; keySet()</code>	모든 키를 Set 타입으로 반환한다.
<code>V put(K key, V value)</code>	주어진 키-값을 저장하고 값을 반환한다.
<code>V remove(Object key)</code>	키와 일치하는 원소를 삭제하고 값을 반환한다.
<code>int size()</code>	컬렉션의 크기를 반환한다.
<code>Collection&lt;V&gt; values()</code>	모든 값을 Collection 타입으로 반환한다.

- Map.Entry<K, V> 인터페이스가 제공

메서드	설명
<code>K getKey()</code>	원소에 해당하는 키를 반환한다.
<code>V getValue()</code>	원소에 해당하는 값을 반환한다.
<code>V setValue()</code>	원소의 값을 교체한다.

# Map 인터페이스

## ■ 주요 메서드

- 팩토리 메서드(자바 9부터)

```
static <K, V> Map<K,V> of(K k1, V v1)
```

- 디폴트 메서드

```
default void forEach(BiConsumer action)
default void replaceAll(BiFunction function)
```

- 예제 : [sec03/MapDemo](#)

4종류의 과일이 있습니다.

```
{바나나=3, 사과=5, 포도=10, 딸기=1}
```

바나나가 3개 있습니다.  
사과가 5개 있습니다.  
포도가 10개 있습니다.  
딸기가 1개 있습니다.  
바나나가 3개 있습니다.  
바나나(3) 사과(5) 포도(10) 딸기(1)

Map 객체의 원소는 입력된  
순서와 관계없다. 따라서 실행할  
때마다 순서가 다를 수 있다.

# Map 인터페이스

## ■ HashMap과 Hashtable

- Hashtable은 HashMap과 달리 동기화된 메서드로 구현되어 스레드에 안전
- HashMap에서는 키와 값으로 null을 사용할 수 있지만 Hashtable에서는 사용할 수 없다.
- 예제 : [sec03/Fruit](#) , [sec03/HashMap1Demo](#)

현재 5종류의 과일이 있습니다.

바나나를 없앤 후 4종류의 과일이 있습니다.

망고를 추가한 후 현재 {사과=5, 포도=10, 귤=2, 망고=2, 딸기=1}가 있습니다.

모두 없앤 후 0종류의 과일이 있습니다.

- 예제 : [sec03/HashMap1Demo](#)

- 오버라이딩할 때

2

{null=3, Fruit[사과]=2}

- 오버라이딩하지 않을 때

3

{null=3, Fruit[사과]=5, Fruit[사과]=2}

HashMap에서 null도  
키로 사용할 수 있다.

동일한 키-값이라도  
다른 원소로 취급한다.

# Collections 클래스

## ■ 특징

- 컬렉션을 다루는 다양한 메서드를 제공하는 java.util 패키지의 클래스
- 컬렉션 원소 정렬, 섞기, 탐색 등 문제를 쉽게 해결

## ■ 정렬하기

```
static void reverse(List list)
static void sort(List list)
static void sort(List list, Comparator c)

static Comparator reverseOrder()
static Comparator reverseOrder(Comparator c)
```

- 예제 : [sec03/SortDemo](#)



```
[포도, 키위, 수박, 사과, 망고]
[망고, 사과, 수박, 키위, 포도]
```

# Collections 클래스

## ■ 돌리기 및 섞기

```
static void rotate(List<?> list, int distance)
```

```
static void shuffle(List<?> list)
```

```
static void shuffle(List<?> list, Random r)
```

- 예제 : [sec03/ShuffleDemo](#)

최초 리스트 : [A, B, C, D, E, F, G]

돌린 리스트 : [E, F, G, A, B, C, D]

섞은 리스트 : [G, A, B, E, D, C, F]

## ■ 탐색하기

```
static <T> int binarySearch(List<T> list, T key)
```

```
static <T> int binarySearch(List<T> list, T key, Comparator<T> c)
```

- 예제 : [sec03/SearchDemo](#)

[갈이, 돌, 보라, 황금을]

1

# Collections 클래스

## ■ 기타 메서드

메서드	설명
<code>addAll()</code>	명시된 원소들을 컬렉션에 삽입한다.
<code>copy()</code>	리스트를 다른 리스트로 복사한다.
<code>disjoint()</code>	2개의 컬렉션에서 공통된 원소가 있는지 조사한다.
<code>fill()</code>	리스트의 모든 원소를 특정 값으로 덮어쓴다.
<code>frequency()</code>	컬렉션에 주어진 원소의 빈도수를 반환한다.
<code>max()</code>	리스트에서 최댓값을 반환한다.
<code>min()</code>	리스트에서 최솟값을 반환한다.
<code>nCopies()</code>	매개변수 값으로 주어진 객체를 주어진 횟수만큼 복사해 List 객체를 반환한다.
<code>reverse()</code>	리스트의 원소들을 역순으로 정렬한다.
<code>swap()</code>	리스트에서 주어진 위치에 있는 두 원소를 교체한다.

- 예제 : [sec04/EtcDemo](#)

```
망고 : 1
사과 : 3
수박 : 1
키위 : 1
포도 : 2
```

# Vector<E>

---

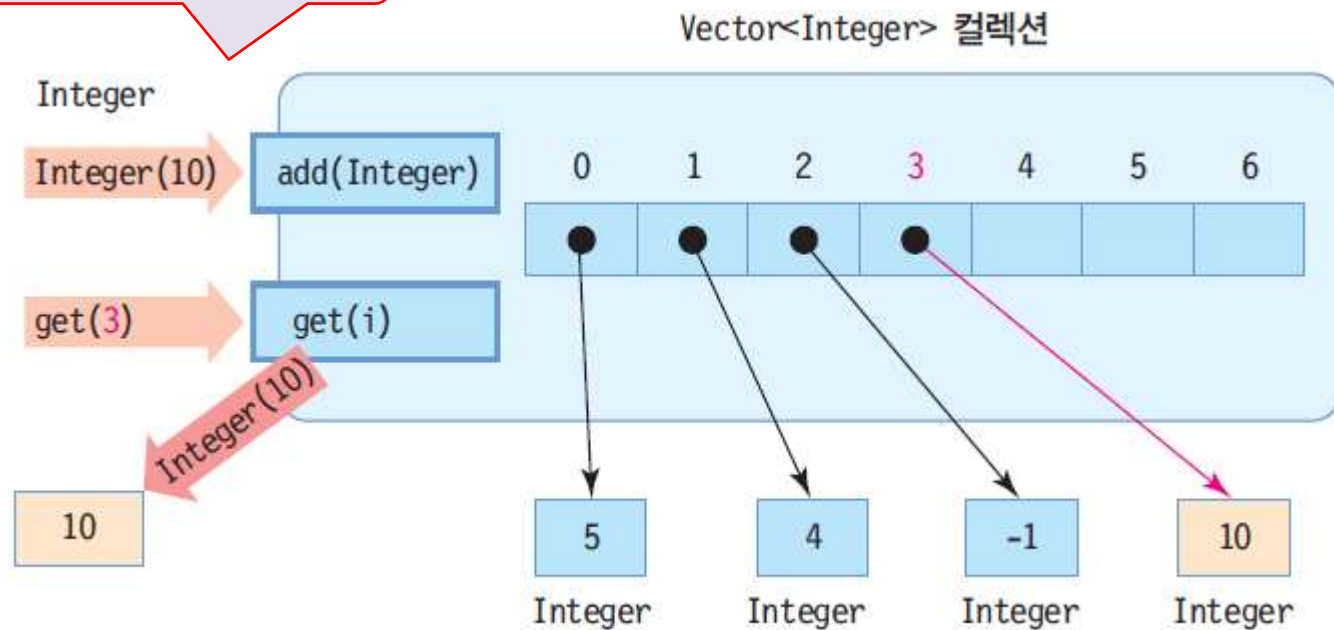
## ■ Vector<E>의 특성

- java.util.Vector
  - <E>에서 E 대신 요소로 사용할 특정 타입으로 구체화
- 여러 객체들을 삽입, 삭제, 검색하는 컨테이너 클래스
  - 배열의 길이 제한 극복
  - 원소의 개수가 넘쳐나면 자동으로 길이 조절
- Vector에 삽입 가능한 것
  - 객체, null
  - 기본 타입은 Wrapper 객체로 만들어 저장
- Vector에 객체 삽입
  - 벡터의 맨 뒤에 객체 추가
  - 벡터 중간에 객체 삽입
- Vector에서 객체 삭제
  - 임의의 위치에 있는 객체 삭제 가능 : 객체 삭제 후 자동 자리 이동

# Vector<Integer> 컬렉션 내부 구성

```
Vector<Integer> v = new Vector<Integer>();
```

add()를 이용하여 요소를 삽입하고  
get()을 이용하여 요소를 검색합니다



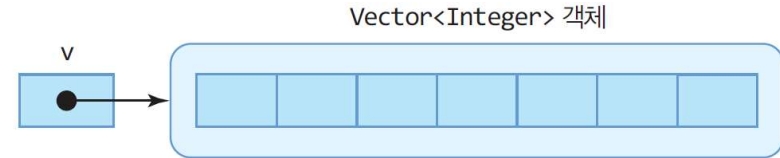


# Vector<E> 클래스의 주요 메소드

메소드	설명
boolean add(E element)	벡터의 맨 뒤에 element 추가
void add(int index, E element)	인덱스 index에 element를 삽입
int capacity()	벡터의 현재 용량 리턴
boolean addAll(Collection<? extends E> c)	컬렉션 c의 모든 요소를 벡터의 맨 뒤에 추가
void clear()	벡터의 모든 요소 삭제
boolean contains(Object o)	벡터가 지정된 객체 o를 포함하고 있으면 true 리턴
E elementAt(int index)	인덱스 index의 요소 리턴
E get(int index)	인덱스 index의 요소 리턴
int indexOf(Object o)	o와 같은 첫 번째 요소의 인덱스 리턴, 없으면 -1 리턴
boolean isEmpty()	벡터가 비어 있으면 true 리턴
E remove(int index)	인덱스 index의 요소 삭제
boolean remove(Object o)	객체 o와 같은 첫 번째 요소를 벡터에서 삭제
void removeAllElements()	벡터의 모든 요소를 삭제하고 크기를 0으로 만듦
int size()	벡터가 포함하는 요소의 개수 리턴
Object[] toArray()	벡터의 모든 요소를 포함하는 배열 리턴

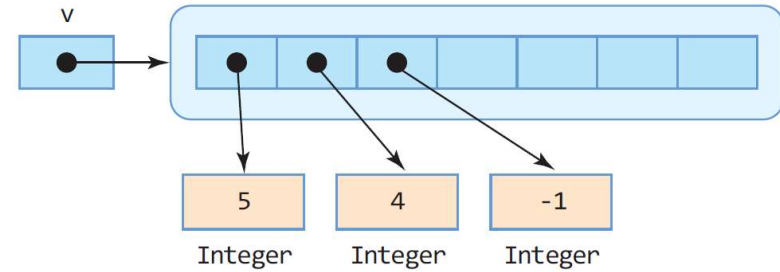
벡터 생성

```
Vector<Integer> v = new Vector<Integer>(7);
```



요소 삽입

```
v.add(5);  
v.add(4);  
v.add(-1);
```



요소 개수 n  
벡터의 용량 c

```
int n = v.size(); // n은 3  
int c = v.capacity(); // c는 7
```

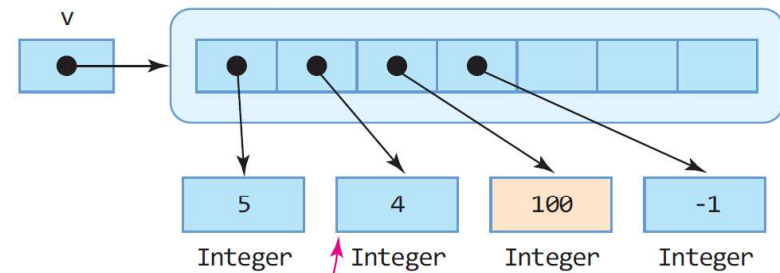
n = 3  
c = 7

요소 중간 삽입

```
v.add(2, 100);
```

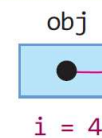
오류

```
v.add(5, 100);  
// v.size()보다 큰 곳에 삽입 불가능, 오류
```



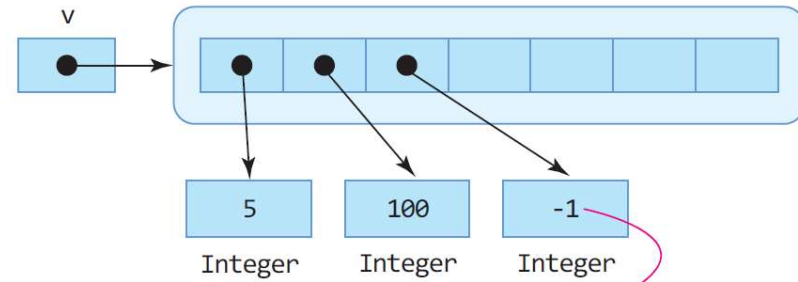
요소 얻어내기

```
Integer obj = v.get(1);  
int i = obj.intValue();
```



요소 삭제 `v.remove(1);`

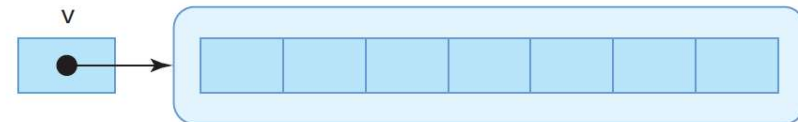
**오류** `v.remove(4);`  
// 인덱스 4에 요소 객체가 없으므로 오류



마지막 요소 `int last = v.lastElement();`

`last = -1`

모든 요소 삭제 `v.removeAllElements();`



정수만 다루는 벡터를 생성하고, 활용하는 기본 사례를 보인다.

```
import java.util.Vector;

public class VectorEx {
    public static void main(String[] args) {
        // 정수 값만 다루는 제네릭 벡터 생성
        Vector<Integer> v = new Vector<Integer>();

        v.add(5); // 5 삽입
        v.add(4); // 4 삽입
        v.add(-1); // -1 삽입

        // 벡터 중간에 삽입하기
        v.add(2, 100); // 4와 -1 사이에 정수 100 삽입

        System.out.println("벡터 내의 요소 객체 수 : " + v.size());
        System.out.println("벡터의 현재 용량 : " + v.capacity());

        // 모든 요소 정수 출력하기
        for(int i=0; i<v.size(); i++) {
            int n = v.get(i);
            System.out.println(n);
        }
    }
}
```

```
// 벡터 속의 모든 정수 더하기
int sum = 0;
for(int i=0; i<v.size(); i++) {
    int n = v.elementAt(i);
    sum += n;
}
System.out.println("벡터에 있는 정수 합 : "
                    + sum);
}
```

벡터 내의 요소 객체 수 : 4  
벡터의 현재 용량 : 10  
5  
4  
100  
-1  
벡터에 있는 정수 합 : 108

점 (x, y)를 표현하는 Point 클래스를 만들고, Point의 객체만 다루는 벡터를 작성하라.

```
import java.util.Vector;

class Point {
    private int x, y;
    public Point(int x, int y) {
        this.x = x;
        this.y = y;
    }

    public String toString() {
        return "(" + x + "," + y + ")";
    }
}
```

```
public class PointVectorEx {
    public static void main(String[] args) {
        // Point 객체를 요소로만 가지는 벡터 생성
        Vector<Point> v = new Vector<Point>();

        // 3 개의 Point 객체 삽입
        v.add(new Point(2, 3));
        v.add(new Point(-5, 20));
        v.add(new Point(30, -8));

        v.remove(1); // 인덱스 1의 Point(-5, 20) 객체 삭제

        // 벡터에 있는 Point 객체 모두 검색하여 출력
        for(int i=0; i<v.size(); i++) {
            Point p = v.get(i); // 벡터에서 i 번째 Point 객체 얻어내기
            System.out.println(p); // p.toString()을 이용하여 객체 p 출력
        }
    }
}
```

(2,3)  
(30,-8)

## ■ 컬렉션을 매개변수로 받는 메소드의 원형

- 예) public void printVector(Vector<Integer> v)

// Integer 벡터를 매개변수로 받아 원소를 모두 출력하는 printVector() 메소드

```
public void printVector(Vector<Integer> v) {  
    for(int i=0; i<v.size(); i++) {  
        int n = v.get(i); // 벡터의 i 번째 정수  
        System.out.println(n);  
    }  
}
```

```
Vector<Integer> v = new Vector<Integer>(); // Integer 벡터 생성  
printVector(v); // 메소드 호출
```