15장. 컬렉션 프레임워크

Contents

1절. 컬렉션 프레임워크 소개

❖ 컬렉션 프레임워크(Collection Framework)

- **컬렉션(Collection)**
  - **사전적 의미로 요소(객체)를 수집해서 저장하는 것.**
- 응용프로그램을 개발하다 보면 다수의 객체를 저장해 두고 필요할 때마다 꺼내서 사용하는 경우가 많음.
  - (예) 10개의 Product 객체를 저장해 두고, 필요할 때마다 하나씩 꺼내서 이용하면서 추가, 검색, 삭제하는 경우

❖ 가장 간단한 방법은 **배열**을 이용하는 것~!

1절. 컬렉션 프레임워크 소개

❖ 컬렉션 프레임워크(Collection Framework)

```
//길이 10인 배열 생성
Product[]    array  =  new  Product[10] ;
//객체 추가
array[0] = new  Product(“Model One”);
array[1] = new  Product(“Model Two”);
//객체 검색
Product  modelOne = array[0];
Product  modelTwo = array[1];
//객체 삭제
array[0] = null ;
array[1] = null ;
```

1절. 컬렉션 프레임워크 소개

❖ 컬렉션 프레임워크(Collection Framework)

▪ 배열의 문제점

- 저장할 수 있는 객체 수가 배열을 생성할 때 결정

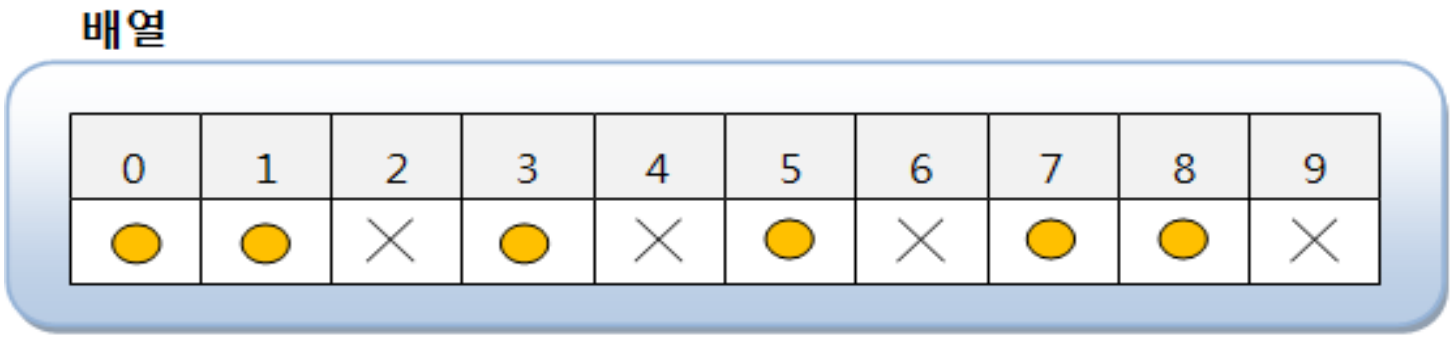
  → 불특정 다수의 객체를 저장하기에는 문제

  ➢ 물론 배열의 길이를 크게 생성하면 되지만, 메모리 공간의 낭비 초래

- 객체 삭제했을 때 해당 인덱스가 비게 됨

  → 낱알 빠진 옥수수 같은 배열

  → 새로운 객체를 저장하려면 어디가 비어있는지 확인하는 코드도 필요함.



1절. 컬렉션 프레임워크 소개

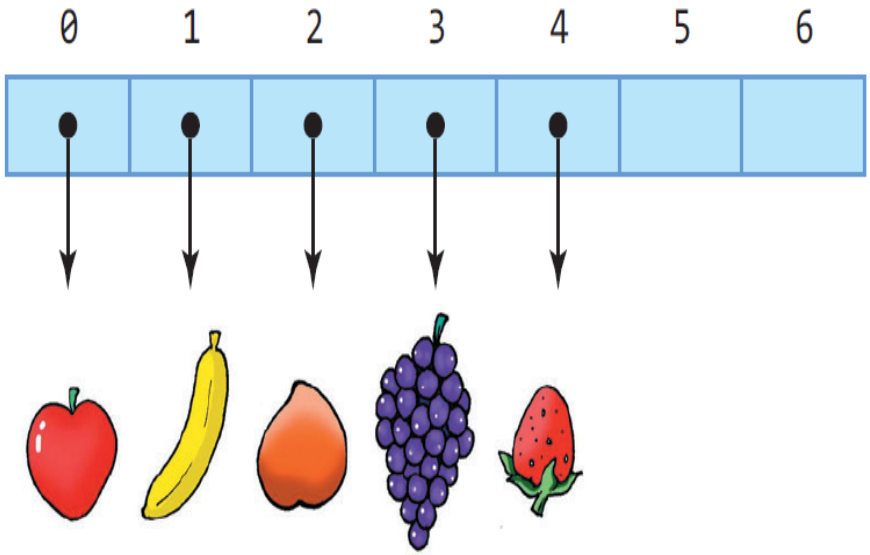
❖ 컬렉션 프레임워크(Collection Framework)

- 요소(element) 객체들의 저장소
  - 객체들의 컨테이너라고도 불림
  - 요소의 개수에 따라 크기 자동 조절
  - 요소의 삽입, 삭제에 따른 요소의 위치 자동 이동
- 고정 크기의 배열을 다루는 어려움 해소
- 다양한 객체들의 삽입, 삭제, 검색 등의 관리 용이

1절. 컬렉션 프레임워크 소개

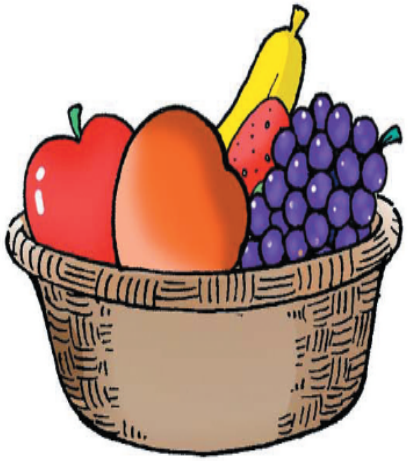
❖ 컬렉션 프레임워크(Collection Framework)

배열(array)



- 고정 크기 이상의 객체를 관리할 수 없다.
- 배열의 중간에 객체가 삭제되면 응용프로그램에서 자리를 옮겨야 한다.

컬렉션(collection)



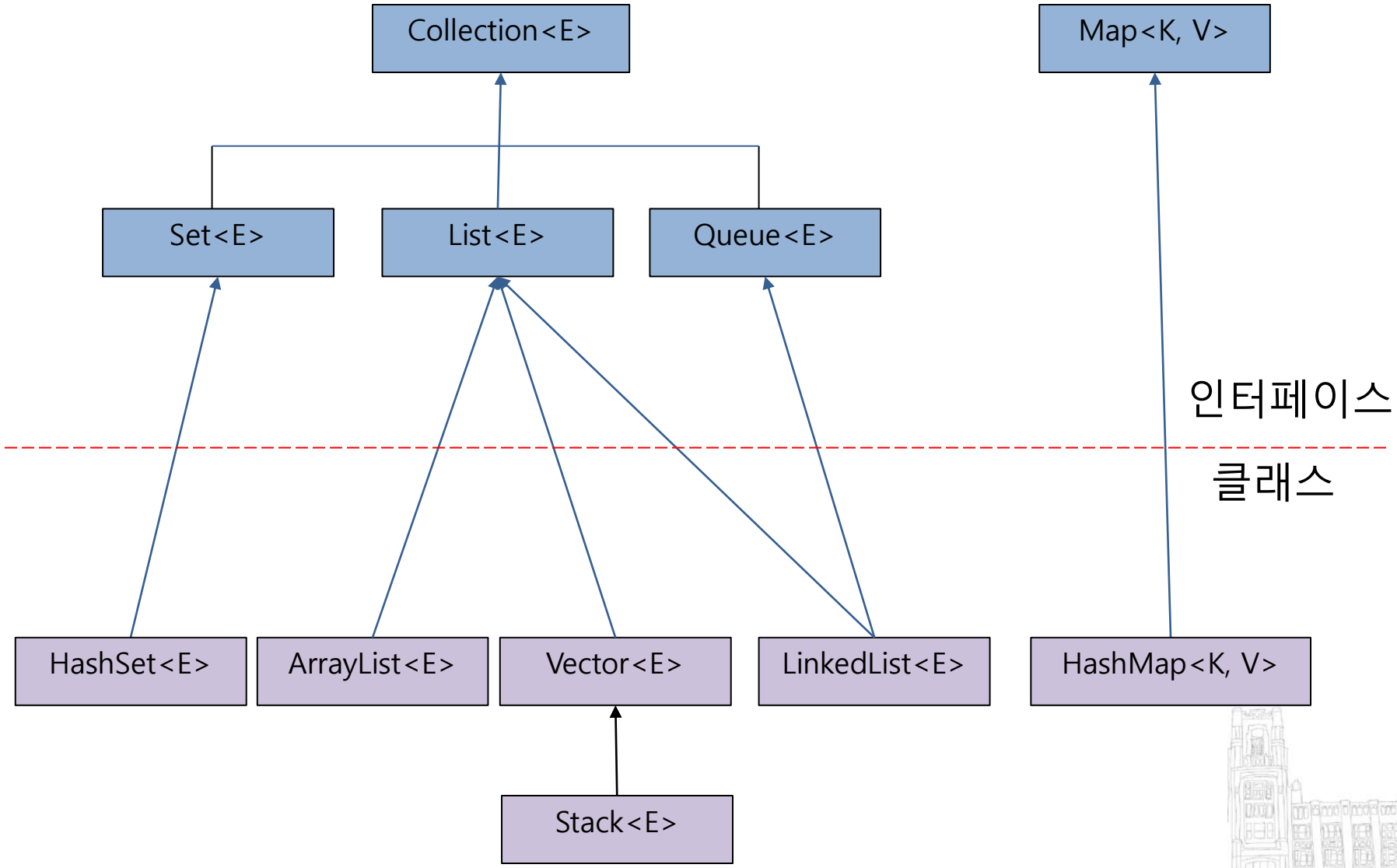
- 가변 크기로서 객체의 개수를 염려할 필요 없다.
- 컬렉션 내의 한 객체가 삭제되면 컬렉션이 자동으로 자리를 옮겨준다.

1절. 컬렉션 프레임워크 소개

❖ 컬렉션 프레임워크(Collection Framework)

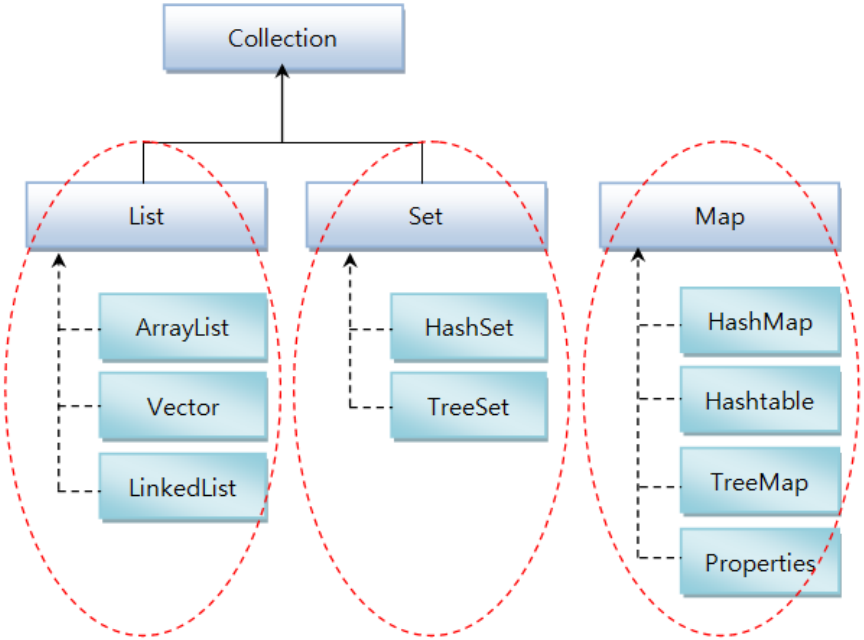
- 객체들을 효율적으로 추가, 삭제, 검색할 수 있도록 제공되는 컬렉션 라이브러리
- 자바는 위에서 언급한 배열의 이러한 문제점을 해결하고, 널리 알려져 있는 자료구조(Data Structure)를 바탕으로 객체들을 효율적으로 추가, 삭제, 검색할 수 있도록 java.util 패키지에 컬렉션과 관련된 인터페이스와 클래스들을 포함시킴.
  - ✓ 이들을 총칭해서 컬렉션 프레임워크(Collection Framework)라고 부름.
- Collection 은 객체를 수집해서 저장하는 역할을 수행.
- Framwork 란 사용방법을 미리 정해놓은 라이브러리를 의미함
- 인터페이스를 통해서 정형화된 방법으로 다양한 컬렉션 클래스 이용

컬렉션을 위한 자바 인터페이스와 클래스



1절. 컬렉션 프레임워크 소개

❖ 컬렉션 프레임워크의 주요 인터페이스



인터페이스 분류		특징	구현 클래스
Collection	List 계열	- 순서를 유지하고 저장 - 중복 저장 가능	ArrayList, Vector, LinkedList
	Set 계열	- 순서를 유지하지 않고 저장 - 중복 저장 안됨	HashSet, TreeSet
Map 계열		- 키와 값의 쌍으로 저장 - 키는 중복 저장 안됨	HashMap, Hashtable, TreeMap, Properties

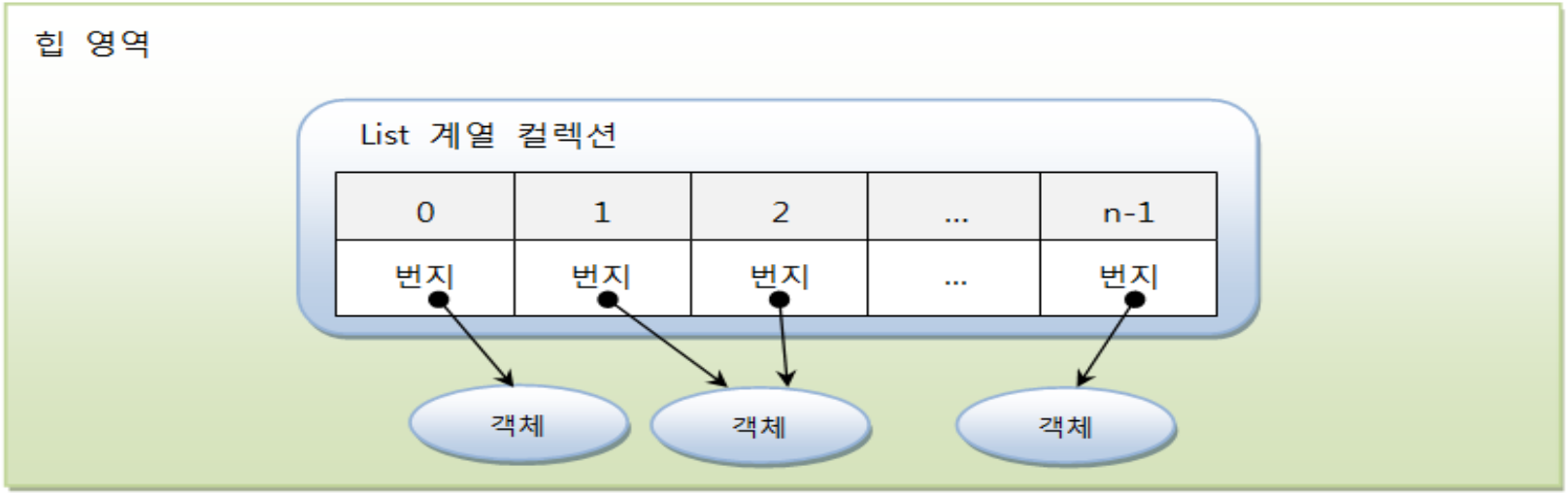
컬렉션과 제네릭

- 컬렉션은 제네릭(generics) 기법으로 구현됨
- 컬렉션의 요소는 객체만 가능
  - 기본적으로 int, char, double 등의 기본 타입 사용 불가
    - JDK 1.5부터 자동 박싱/언박싱으로 기본 타입 값을 객체로 자동 변환
- 제네릭
  - 특정 타입만 다루지 않고, 여러 종류의 타입으로 변신할 수 있도록 클래스나 메소드를 일반화 시키는 기법
    - <E>, <K>, <V> : 타입 매개 변수
      - 요소 타입을 일반화한 타입
  - 제네릭 클래스 사례
    - 제네릭 스택 : Stack<E>
      - 타입 파라미터 E에 특정 타입으로 구체화
      - 정수만 다루는 스택 : Stack<Integer>
      - 문자열만 다루는 스택 : Stack<String>

2절. List 컬렉션

❖ List 컬렉션의 특징 및 주요 메소드

- 특징
  - 인덱스로 관리
  - 중복해서 객체 저장 가능
- 구현 클래스
  - ArrayList
  - Vector
  - LinkedList



2절. List 컬렉션

❖ List 컬렉션의 특징 및 주요 메소드

▪ 주요 메소드

기능	메소드	설명
객체 추가	boolean add(E e)	주어진 객체를 맨끝에 추가
	void add(int index, E element)	주어진 인덱스에 객체를 추가
	set(int index, E element)	주어진 인덱스에 저장된 객체를 주어진 객체로 바꿈
객체 검색	boolean contains(Object o)	주어진 객체가 저장되어 있는지 여부
	E get(int index)	주어진 인덱스에 저장된 객체를 리턴
	isEmpty()	컬렉션이 비어 있는지 조사
	int size()	저장되어있는 전체 객체수를 리턴
객체 삭제	void clear()	저장된 모든 객체를 삭제
	E remove(int index)	주어진 인덱스에 저장된 객체를 삭제
	boolean remove(Object o)	주어진 객체를 삭제

ArrayList<E>

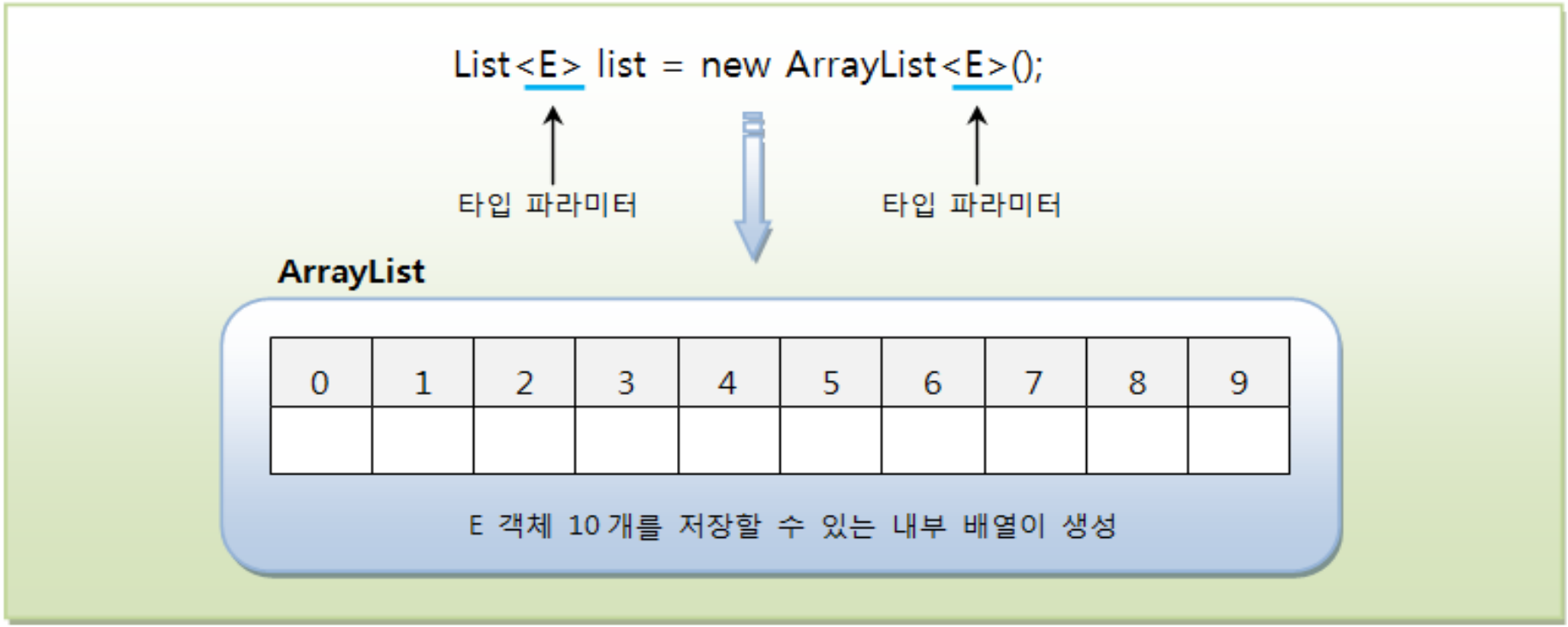
ArrayList<E>의 특성

- java.util.ArrayList, 가변 크기 배열을 구현한 클래스
  - <E>에서 E 대신 요소로 사용할 특정 타입으로 구체화
- ArrayList에 삽입 가능한 것
  - 객체, null
  - 기본 타입은 박싱/언박싱으로 Wrapper 객체로 만들어 저장
- ArrayList에 객체 삽입/삭제
  - 리스트의 맨 뒤에 객체 추가
  - 리스트의 중간에 객체 삽입
  - 임의의 위치에 있는 객체 삭제 가능
- 벡터와 달리 스레드 동기화 기능 없음
  - 다수 스레드가 동시에 ArrayList에 접근할 때 동기화되지 않음
  - 개발자가 스레드 동기화 코드를 직접 작성해야 함.

2절. List 컬렉션

❖ ArrayList (p.725~729)

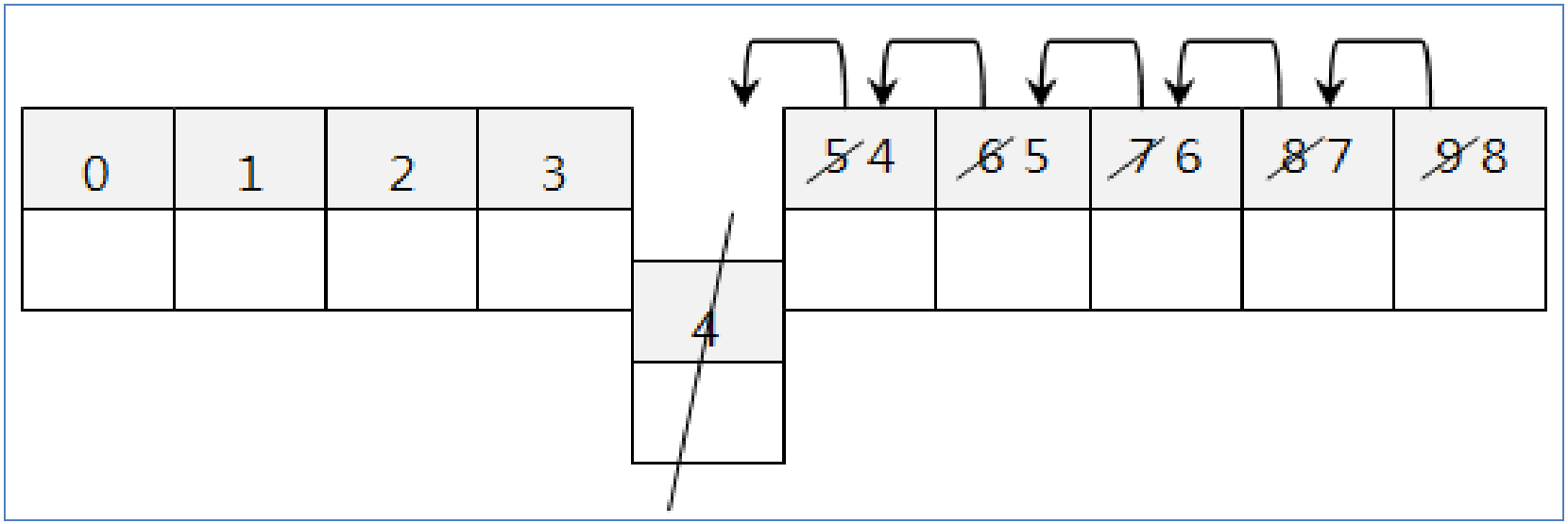
- 저장 용량(capacity)
  - 초기 용량 : 10 (따로 지정 가능)
  - 저장 용량을 초과한 객체들이 들어오면 자동적으로 늘어남. 고정도 가능



2절. List 컬렉션

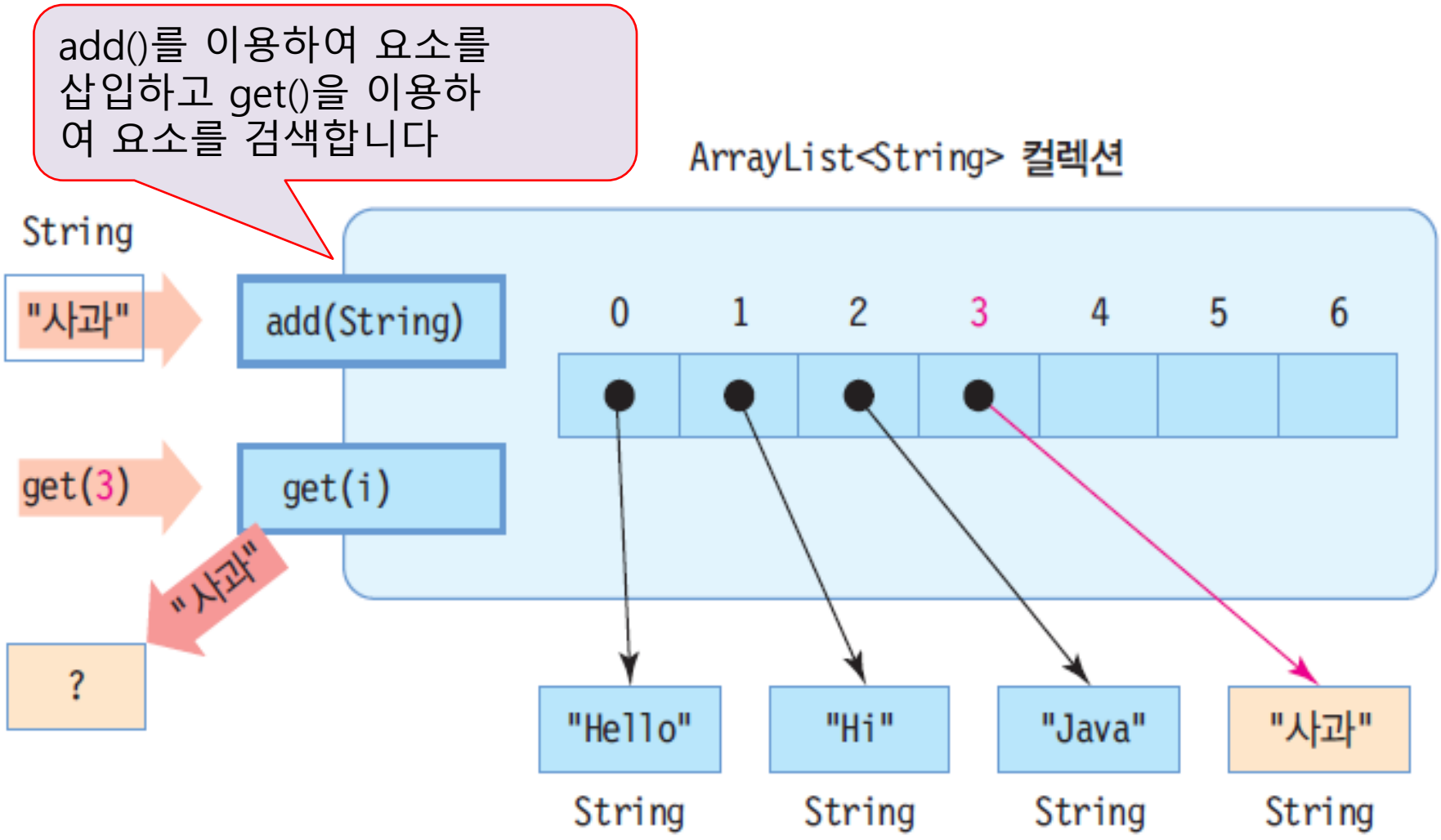
❖ ArrayList (p.725~729)

- 객체 제거
  - 바로 뒤 인덱스부터 마지막 인덱스까지 모두 앞으로 1씩 당겨짐



ArrayList<String> 컬렉션의 내부 구성

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

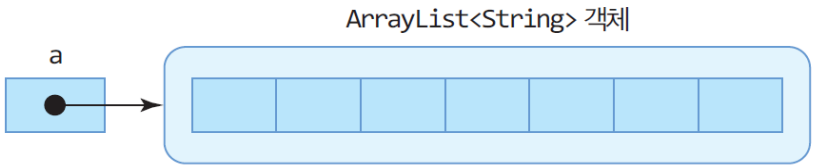


ArrayList<E> 클래스의 주요 메소드

메소드	설명
boolean add(E element)	ArrayList의 맨 뒤에 element 추가
void add(int index, E element)	인덱스 index 위치에 element 삽입
boolean addAll(Collection<? extends E> c)	컬렉션 c의 모든 요소를 ArrayList의 맨 뒤에 추가
void clear()	ArrayList의 모든 요소 삭제
boolean contains(Object o)	ArrayList가 지정된 객체를 포함하고 있으면 true 리턴
E elementAt(int index)	index 인덱스의 요소 리턴
E get(int index)	index 인덱스의 요소 리턴
int indexOf(Object o)	o와 같은 첫 번째 요소의 인덱스 리턴, 없으면 -1 리턴
boolean isEmpty()	ArrayList가 비어있으면 true 리턴
E remove(int index)	index 인덱스의 요소 삭제
boolean remove(Object o)	o와 같은 첫 번째 요소를 ArrayList에서 삭제
int size()	ArrayList가 포함하는 요소의 개수 리턴
Object[] toArray()	ArrayList의 모든 요소를 포함하는 배열 리턴

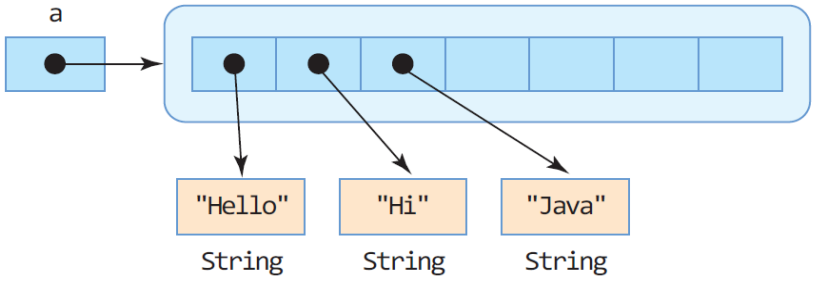
ArrayList 생성

```
ArrayList<String> a = new ArrayList<String>(7);
```



요소 삽입

```
a.add("Hello");
a.add("Hi");
a.add("Java");
```



요소 개수 n

```
int n = a.size(); // n은 3
```

용량

~~`int c = a.capacity();`~~ `// capacity() 메소드 없음`

오류

n = 3

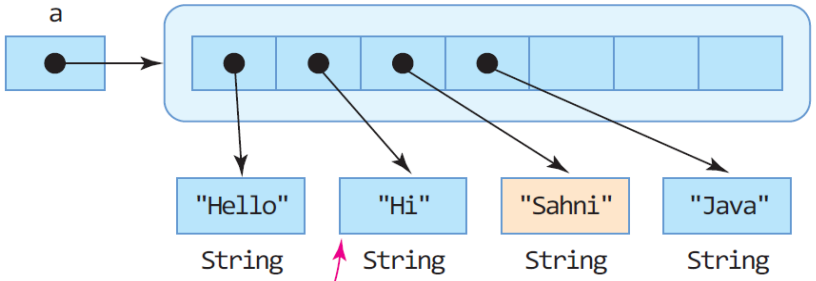
요소 중간 삽입

```
a.add(2, "Sahni");
```

오류

~~`a.add(5, "Sahni");`~~

```
// a.size()보다 큰 위치에 삽입 불가능, 오류
```



요소 알아내기

```
String str = a.get(1);
```

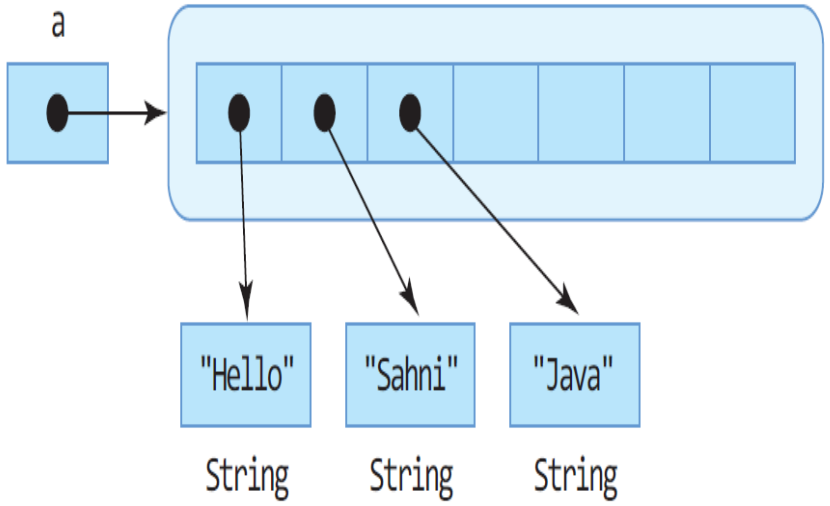
"Hi"

요소 삭제

```
a.remove(1);
```

오류

```
a.remove(4); // 오류
```



모든 요소 삭제

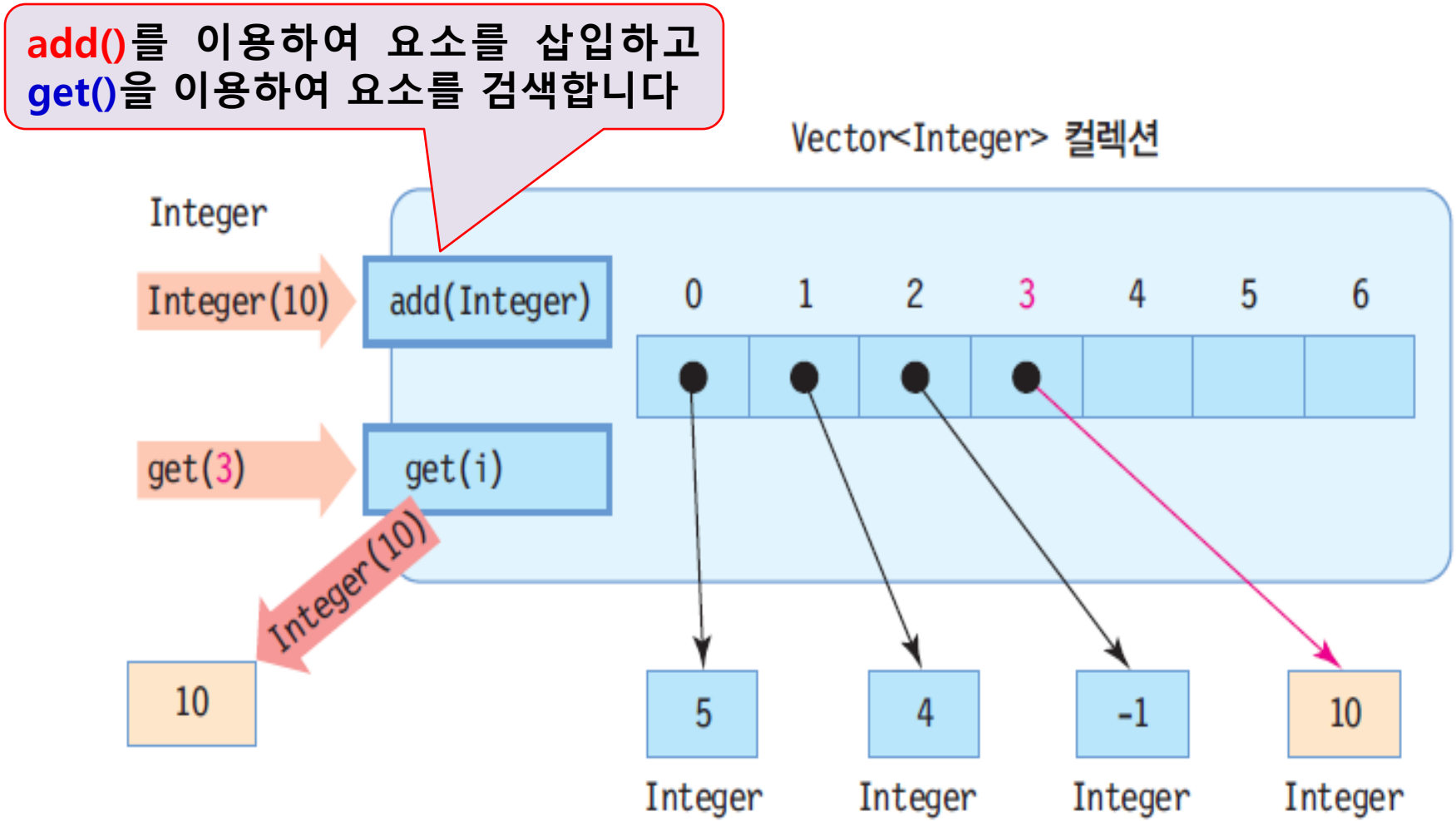
```
a.clear();
```

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

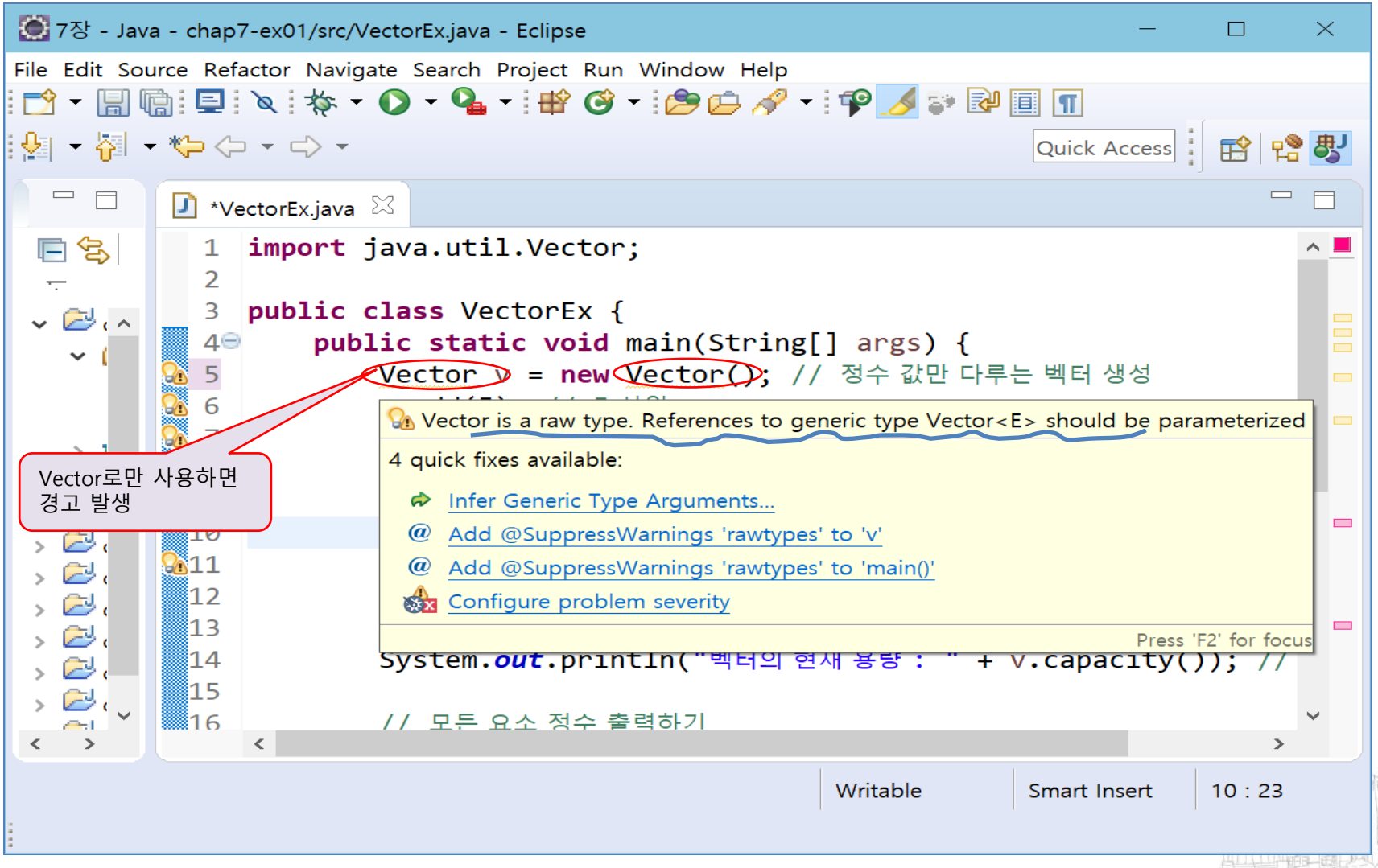
Vector<Integer> 컬렉션 내부 구성

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



타입 매개 변수 사용하지 않는 경우 경고 발생

Vector<Integer>로 타입 매개 변수를 사용하여야 함



Vector<E> 클래스의 주요 메소드

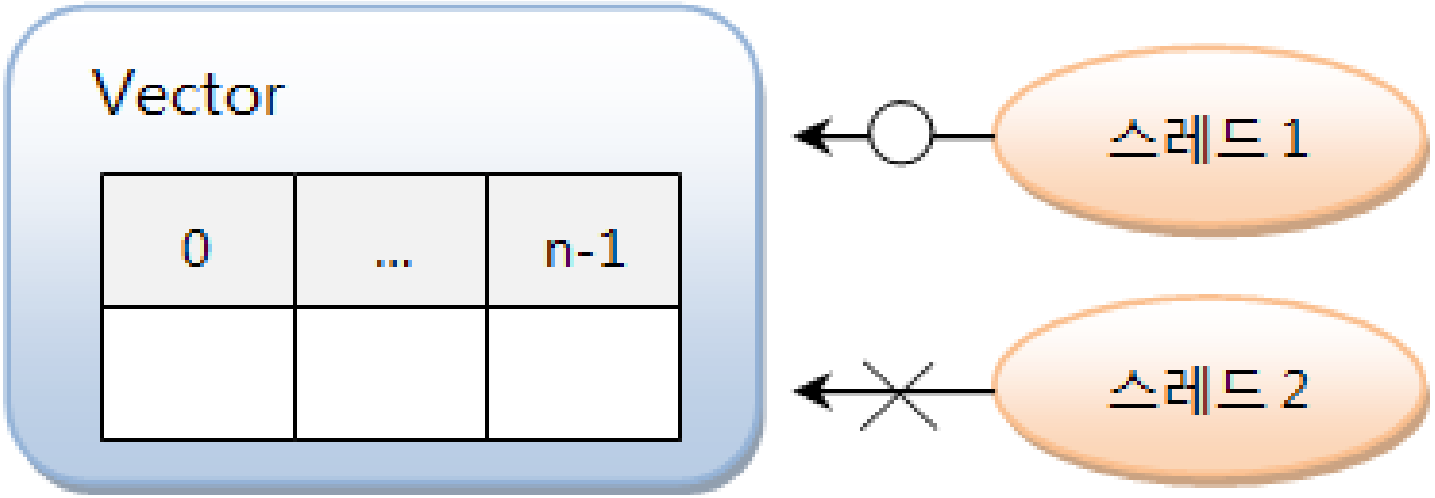
메소드	설명
boolean add(E element)	벡터의 맨 뒤에 element 추가
void add(int index, E element)	인덱스 index에 element를 삽입
int capacity()	벡터의 현재 용량 리턴
boolean addAll(Collection<? extends E> c)	컬렉션 c의 모든 요소를 벡터의 맨 뒤에 추가
void clear()	벡터의 모든 요소 삭제
boolean contains(Object o)	벡터가 지정된 객체 o를 포함하고 있으면 true 리턴
E elementAt(int index)	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

2절. List 컬렉션

❖ Vector

```
List<E> list = new Vector<E>();
```

- 특징
  - Vector는 스레드 동기화(synchronization)
    - 복수의 스레드가 동시에 Vector에 접근해 객체를 추가, 삭제하더라도 스레드에 안전(thread safe)



컬렉션과 자동 박싱/언박싱

JDK 1.5 이전

- 기본 타입 데이터를 Wrapper 클래스를 이용하여 객체로 만들어 사용

```
Vector<Integer> v = new Vector<Integer>();
v.add(new Integer(4));
```

- 컬렉션으로부터 요소를 얻어올 때, Wrapper 클래스로 캐스팅 필요

```
Integer n = (Integer)v.get(0);
int k = n.intValue(); // k = 4
```

JDK 1.5부터

- 자동 박싱/언박싱이 작동하여 기본 타입 값 사용 가능

```
Vector<Integer> v = new Vector<Integer> ();
v.add(4); // 4 → new Integer(4)로 자동 박싱
int k = v.get(0); // Integer 타입이 int 타입으로 자동 언박싱, k = 4
```

  - 제네릭의 타입 매개 변수를 기본 타입으로 구체화할 수는 없음


```
Vector<int> v = new Vector<int> (); // 오류
```

2절. List 컬렉션

❖ LinkedList<E>의 특성

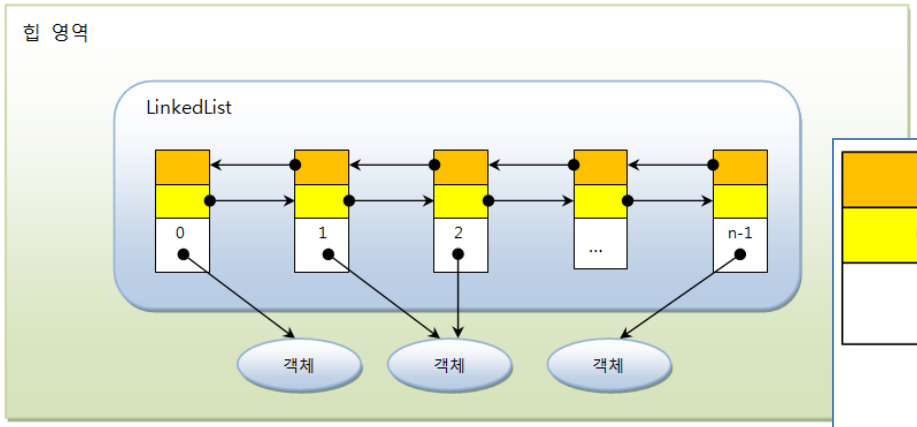
- java.util.LinkedList
  - E에 요소로 사용할 타입 지정하여 구체화
- List 인터페이스를 구현한 컬렉션 클래스
- Vector, ArrayList 클래스와 매우 유사하게 작동
- 요소 객체들은 양방향으로 연결되어 관리됨
- 요소 객체는 맨 앞, 맨 뒤에 추가 가능
- 요소 객체는 인덱스를 이용하여 중간에 삽입 가능
- 맨 앞이나 맨 뒤에 요소를 추가하거나 삭제할 수 있어 스택이나 큐로 사용 가능

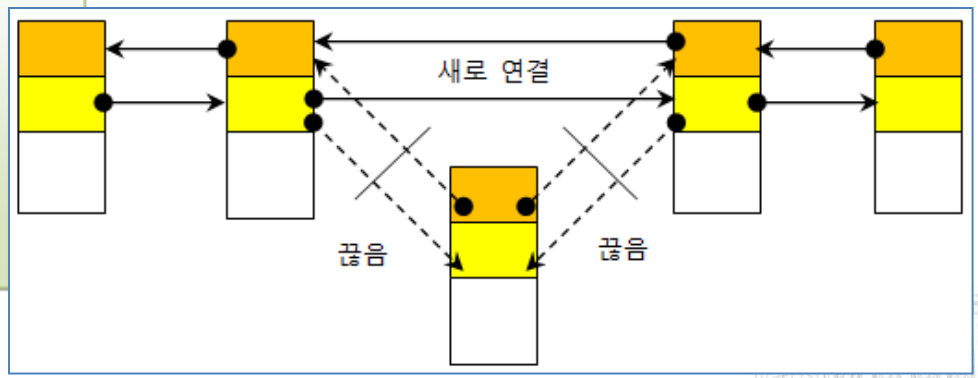
2절. List 컬렉션

❖ LinkedList

```
List<E> list = new LinkedList<E>();
```

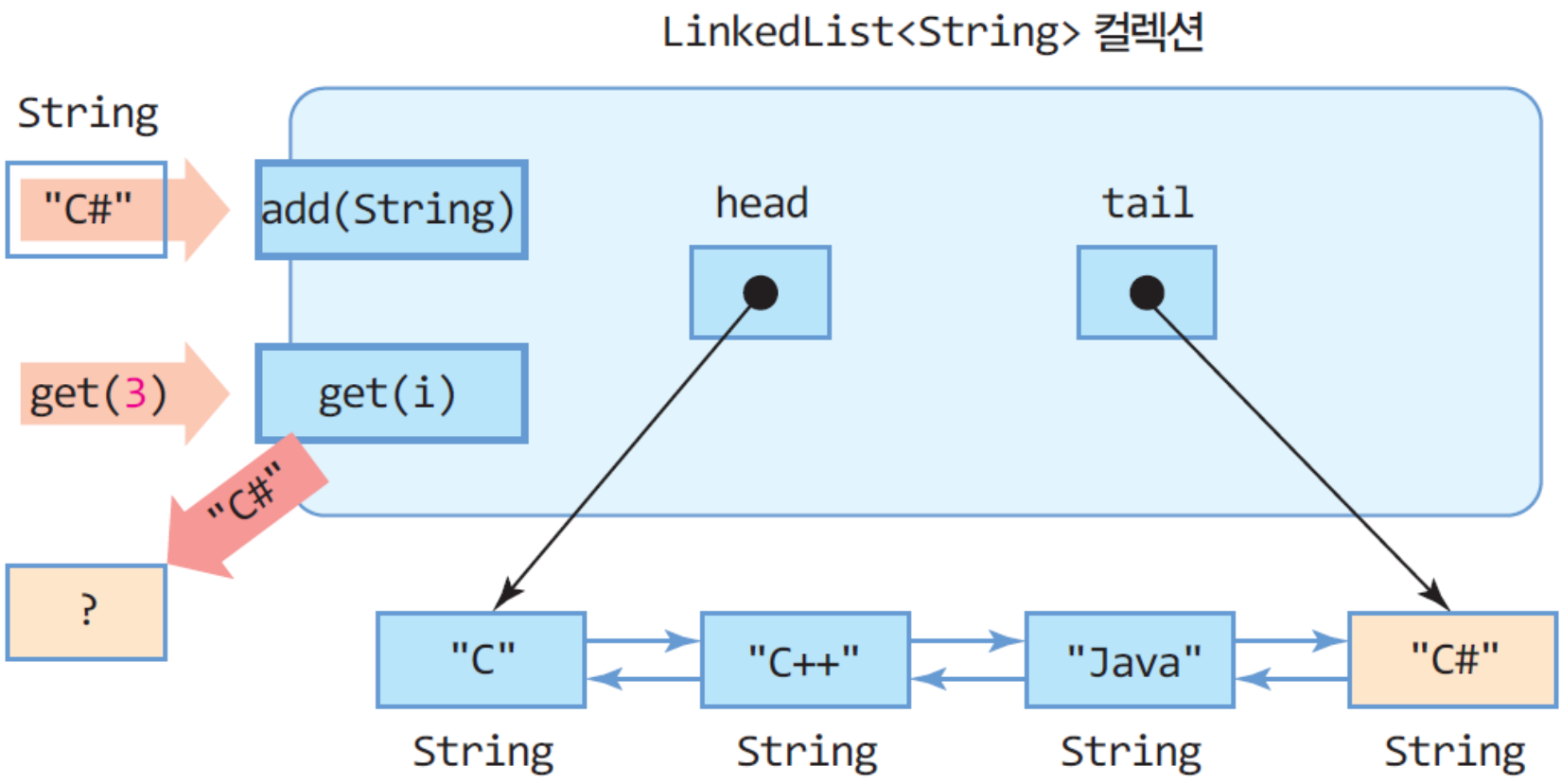
- 특징
  - 인접 참조를 링크해서 체인처럼 관리
  - 특정 인덱스에서 객체를 제거하거나 추가하게 되면 바로 앞뒤 링크만 변경
  - 빈번한 객체 삭제와 삽입이 일어나는 곳에서는 ArrayList보다 좋은 성능





LinkedList<String>의 내부 구성과 put(), get() 메소드

```
LinkedList<String>  l = new LinkedList<String>();
```



Collections 클래스 활용

❖ Collections 클래스

- java.util 패키지에 포함
- 컬렉션에 대해 연산을 수행하고 결과로 컬렉션 리턴
- 모든 메소드는 static 타입
- 주요 메소드
  - 컬렉션에 포함된 요소들을 정렬하는 sort() 메소드
  - 요소의 순서를 반대 순으로 정렬하는 reverse() 메소드
  - 요소들의 최대, 최솟값을 찾아내는 max(), min() 메소드
  - 특정 값을 검색하는 binarySearch() 메소드

컬렉션의 순차 검색을 위한 Iterator

❖ Iterator<E> 인터페이스(1)

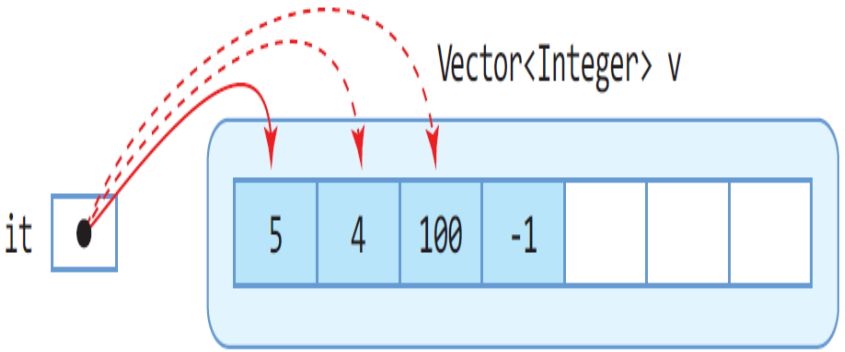
- Vector<E>, ArrayList<E>, LinkedList<E> 가 상속받는 인터페이스
  - 리스트 구조의 컬렉션에서 요소의 순차 검색을 위한 메소드 포함
- Iterator<E> 인터페이스 메소드

메소드	설명
boolean hasNext()	방문할 요소가 남아 있으면 true 리턴
E next()	다음 요소 리턴
void remove()	마지막으로 리턴된 요소 제거

컬렉션의 순차 검색을 위한 Iterator

- Iterator<E> 인터페이스(2)
  - iterator() 메소드 : Iterator 객체 반환
    - Iterator 객체를 이용하여 인덱스 없이 순차적 검색 가능

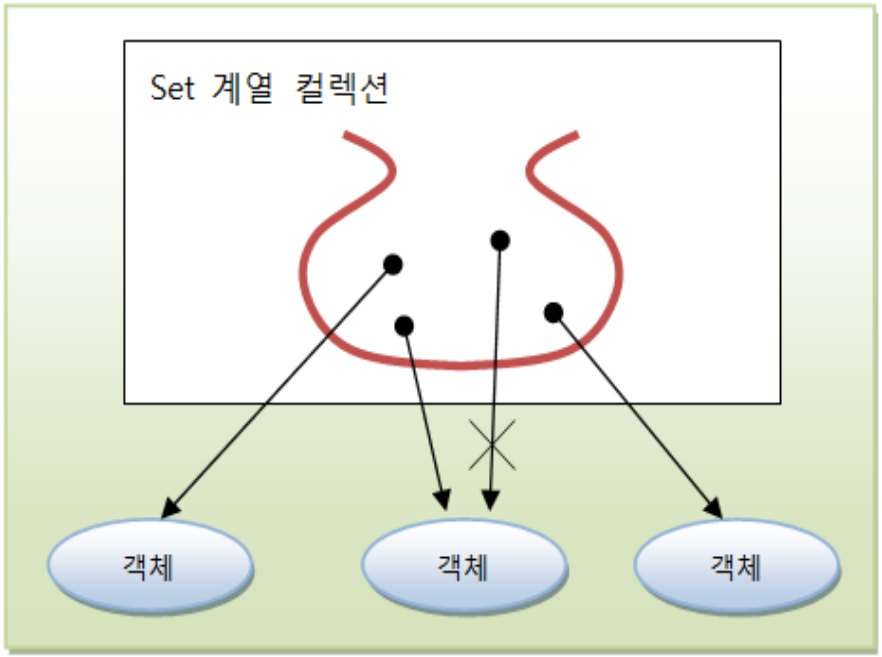
```
Vector<Integer> v = new Vector<Integer>();
Iterator<Integer> it = v.iterator();
while(it.hasNext()) { // 모든 요소 방문
        int n = it.next(); // 다음 요소 리턴
        …
}
```



3절. Set 컬렉션

❖ Set 컬렉션의 특징 및 주요 메소드

- 특징
  - 수학의 집합에 비유
  - 저장 순서가 유지되지 않음
  - 객체를 중복 저장 불가
  - 하나의 null만 저장 가능
- 구현 클래스
  - HashSet, LinkedHashSet, TreeSet



3절. Set 컬렉션

❖ Set 컬렉션의 특징 및 주요 메소드

- 주요 메소드

기능	메소드	설명
객체 추가	boolean add(E e)	주어진 객체를 저장, 객체가 성공적으로 저장되면 true 를 리턴하고 중복 객체면 false 를 리턴
객체 검색	boolean contains(Object o)	주어진 객체가 저장되어 있는지 여부
	isEmpty()	컬렉션이 비어 있는지 조사
	Iterator<E> iterator()	저장된 객체를 한번씩 가져오는 반복자 리턴
	int size()	저장되어있는 전체 객체수 리턴
객체 삭제	void clear()	저장된 모든 객체를 삭제
	boolean remove(Object o)	주어진 객체를 삭제

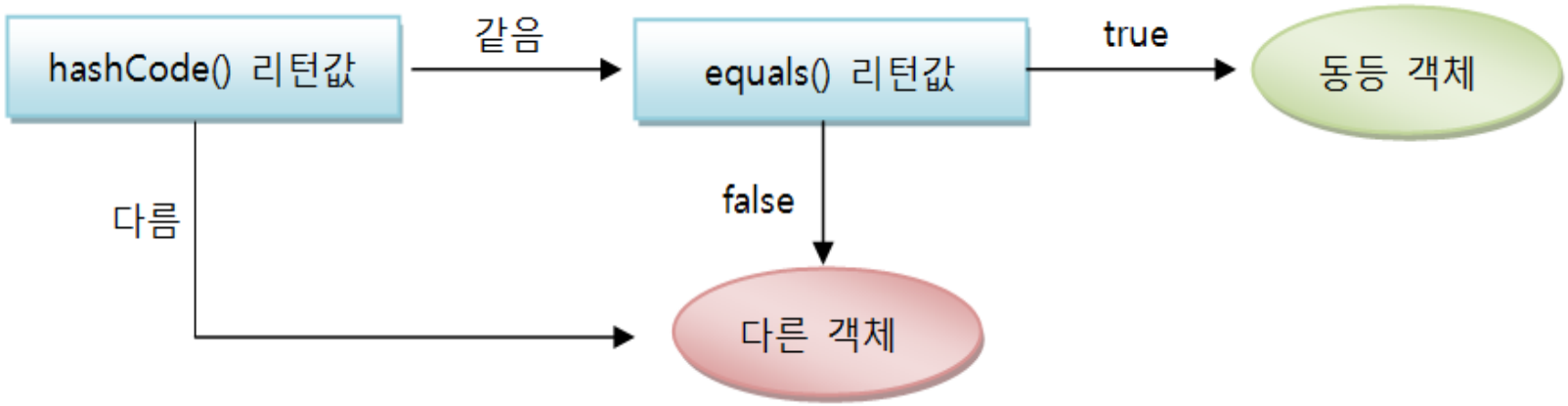
- **전체 객체 대상으로 한 번씩 반복해 가져오는 반복자(Iterator) 제공**
  - **인덱스로 객체를 검색해서 가져오는 메소드 없음**

3절. Set 컬렉션

❖ HashSet (p.736~739)

```
Set<E> set = new HashSet<E>();
```

- HashSet은 객체들을 순서없이 저장하고 동일한 객체는 중복 저장하지 않음.
- 특징
  - 동일 객체 및 동등 객체는 중복 저장하지 않음
  - 동등 객체 판단 방법



3절. Set 컬렉션

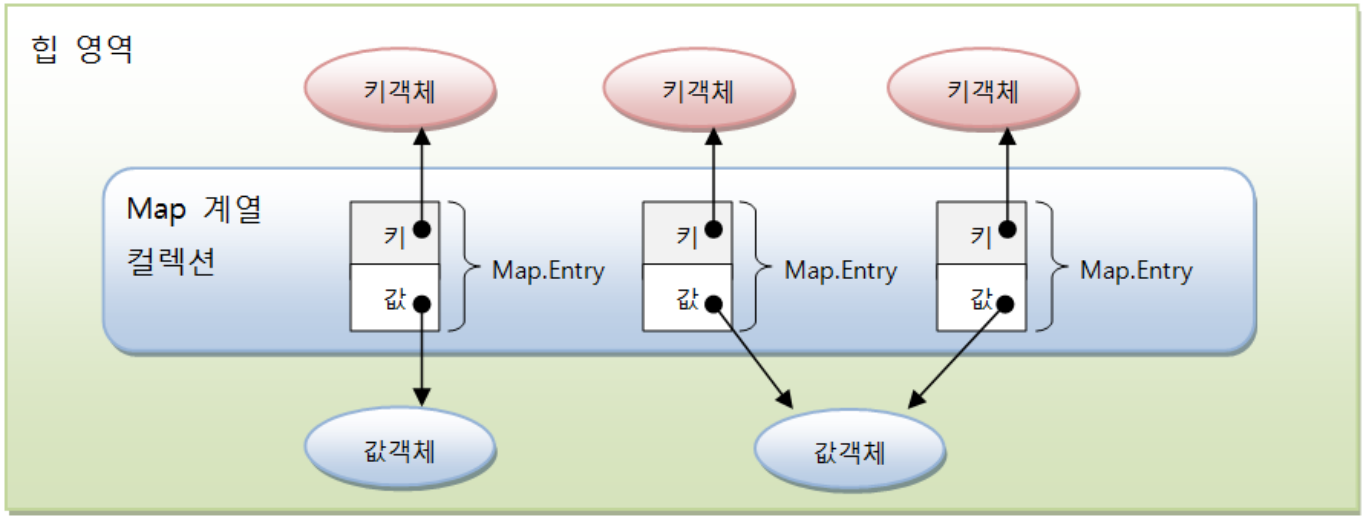
❖ HashSet (p.736~739)

- 문자열을 HashSet 에 저장할 경우
  - 같은 문자열을 갖는 String 객체는 동등한 객체로 간주되고 다른 문자열을 갖는 String 객체는 다른 객체로 간주됨.
    - String 클래스가 hashCode() 메서드와 equals() 메서드를 재정의해서 같은 문자열일 경우 hashCode() 의 리턴값을 같게, equals() 의 리턴값은 true가 나오도록 했기 때문.

4절. Map 컬렉션

❖ Map 컬렉션의 특징 및 주요 메소드

- 특징
  - 키(key)와 값(value)으로 구성된 Map.Entry 객체를 저장하는 구조
  - 키와 값은 모두 객체
  - 키는 중복될 수 없지만 값은 중복 저장 가능
- 구현 클래스
  - HashMap, Hashtable, LinkedHashMap, Properties, TreeMap



4절. Map 컬렉션

- 주로 키 타입은 String 을 많이 사용함
  - String 은 문자열이 같을 경우 동등 객체가 될 수 있도록 hasCode()와 equals() 메서드가 재정의 되어 있기 때문.
  - HashMap 을 생성하기 위해서는 키 타입과 값 타입을 파라미터로 주고 기본 생성자를 호출하면 됨.

```
Map<K, V> map = new HashMap< K, V> ( ) ;
```

키 타입 값 타입 키 타입 값 타입

4절. Map 컬렉션

❖ Map 컬렉션의 특징 및 주요 메소드

▪ 주요 메소드

기능	메소드	설명
객체 추가	V put(K key, V value)	주어진 키와 값을 추가, 저장이 되면 값을 리턴
객체 검색	boolean containsKey(Object key)	주어진 키가 있는지 여부
	boolean containsValue(Object value)	주어진 값이 있는지 여부
	Set<Map.Entry<K,V>> entrySet()	키와 값의 쌍으로 구성된 모든 Map.Entry 객체를 Set 에 담아서 리턴
	V get(Object key)	주어진 키의 값을 리턴
	boolean isEmpty()	컬렉션이 비어있는지 여부
	Set<K> keySet()	모든 키를 Set 객체에 담아서 리턴
	int size()	저장된 키의 총 수를 리턴
	Collection<V> values()	저장된 모든 값 Collection 에 담아서 리턴
객체 삭제	void clear()	모든 Map.Entry(키와 값)를 삭제
	V remove(Object key)	주어진 키와 일치하는 Map.Entry 삭제, 삭제가 되면 값을 리턴

4절. Map 컬렉션

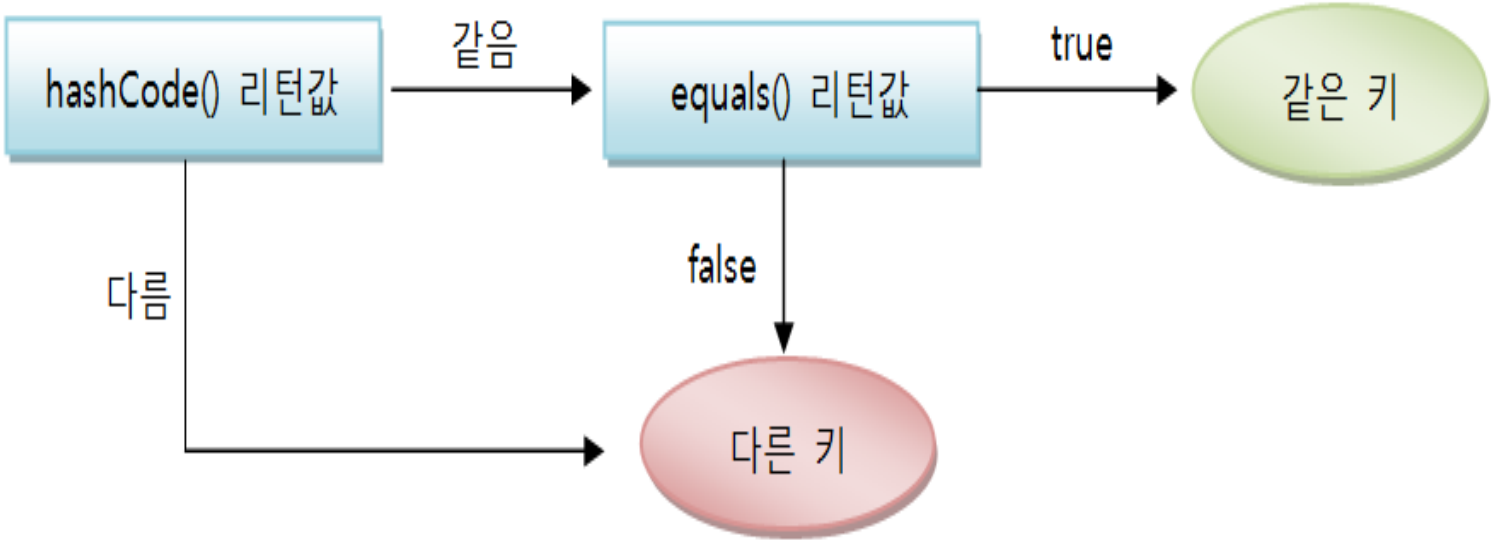
❖ HashMap (p.742~745)

- 특징

```
Map<K, V> map = new HashMap<K, V>();
```

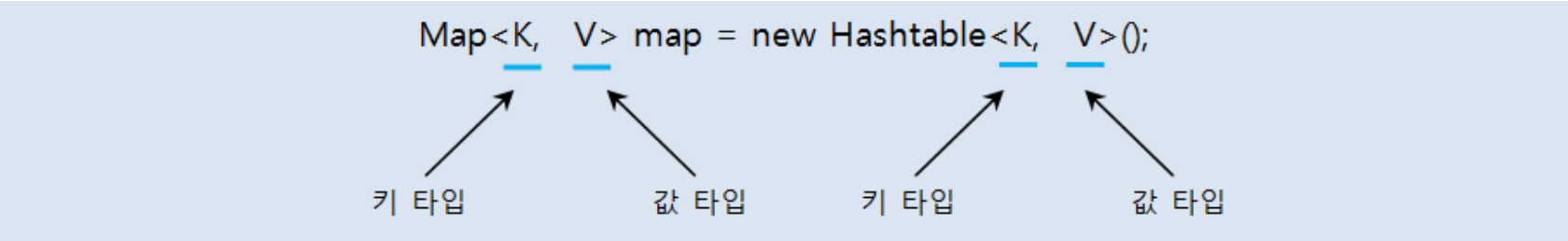
키 타입 — 값 타입 — 키 타입 — 값 타입

  - 키 객체는 hashCode()와 equals() 를 재정의해 동등 객체가 될 조건을 정해야 함.



4절. Map 컬렉션

❖ Hashtable (1)

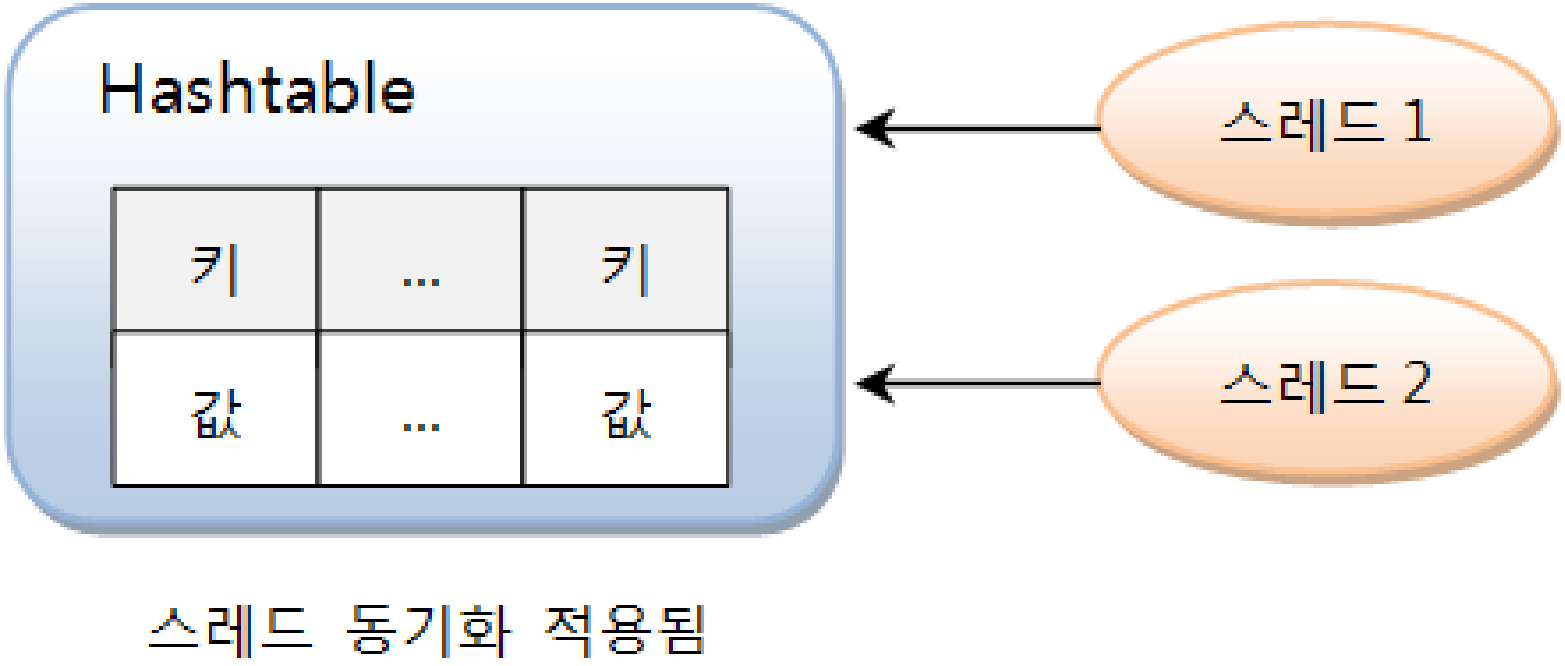


- 특징
  - 키 객체 만드는 법은 HashMap과 동일
  - Hashtable은 스레드 동기화(synchronization)가 된 상태
    - Hashtable은 동기화 된 메서드로 구성되어 있기 때문에 멀티스레드가 동시에 이 메서드들을 실행 할 수는 없음.
    - 하나의 스레드가 실행 완료되어야만 다른 스레드들 실행할 수 있음.
    - 복수의 스레드가 동시에 Hashtable에 접근해서 객체를 추가, 삭제하더라도 스레드에 안전(thread safe)

4절. Map 컬렉션

❖ Hashtable(2)

- 특징



스레드 동기화 적용됨

4절. Map 컬렉션

❖ Properties (p.748~750)

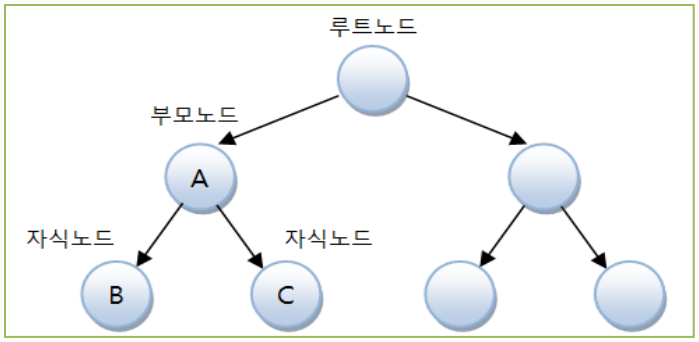
- 특징
  - 키와 값을 String 타입으로 제한한 Map 컬렉션
  - Properties는 프로퍼티(~.properties) 파일을 읽어 들일 때 주로 사용

- 프로퍼티(~.properties) 파일
  - 옵션 정보, 데이터베이스 연결 정보, 국제화(다국어) 정보를 기록
    - 텍스트 파일로 활용
  - 애플리케이션에서 주로 변경이 잦은 문자열을 저장
    - 유지 보수를 편리하게 만들어 줌
  - 키와 값이 = 기호로 연결되어 있는 텍스트 파일
    - ISO 8859-1 문자셋으로 저장
    - 한글은 유니코드(Unicode)로 변환되어 저장

5절. 검색 기능을 강화시킨 컬렉션

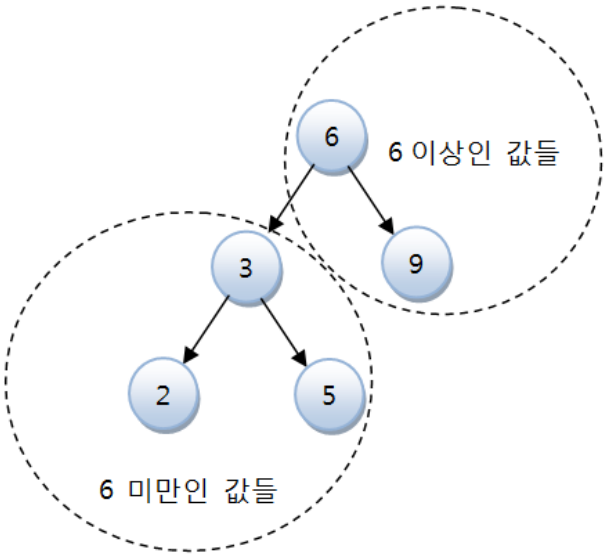
❖ 검색 기능을 강화시킨 컬렉션 (계층 구조 활용)

- **TreeSet, TreeMap** ➡ 이진트리(binary tree) 사용하기 때문에 검색 속도 향상



❖ 이진 트리 구조

- **부모 노드와 자식 노드로 구성**
  - **왼쪽 자식 노드: 부모 보다 적은 값**
  - **오른쪽 자식 노드: 부모 보다 큰 값**



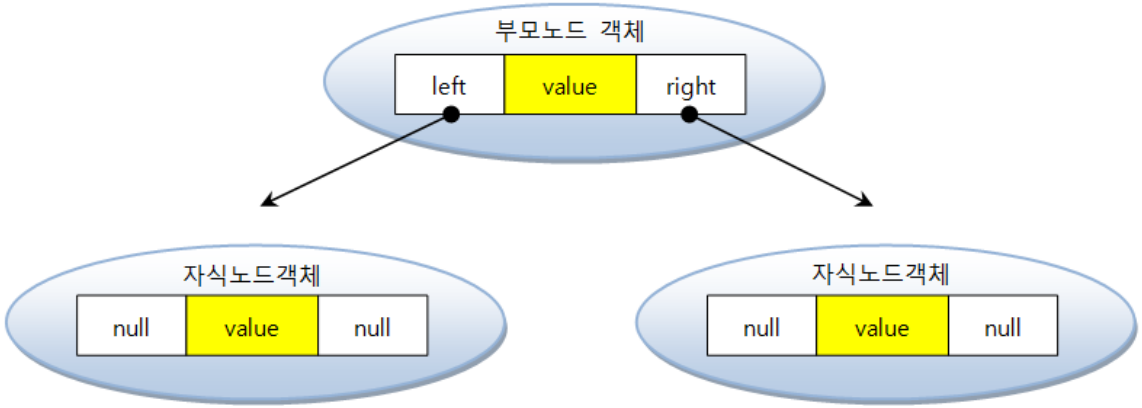
- **정렬 쉬움**
  - **올림 차순: [왼쪽노드→부모노드→오른쪽노드]**
  - **내림 차순: [오른쪽노드→부모노드→왼쪽노드]**

5절. 검색 기능을 강화시킨 컬렉션

❖ TreeSet

▪ 특징

- 이진 트리(binary tree)를 기반으로 한 Set 컬렉션
- 왼쪽과 오른쪽 자식 노드를 참조하기 위한 두 개의 변수로 구성



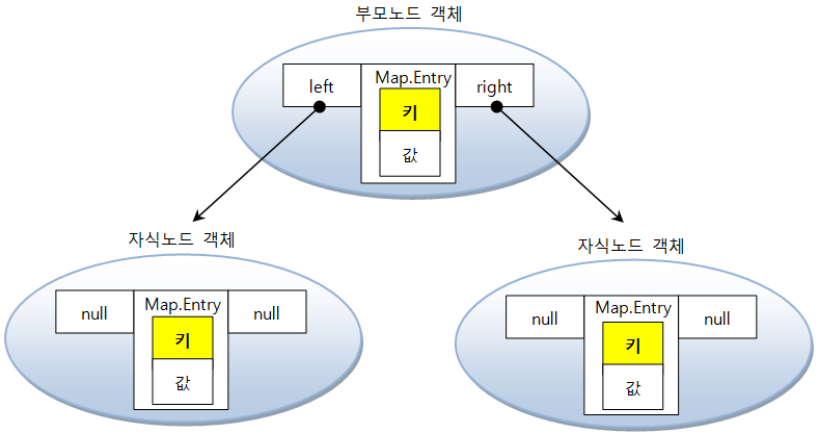
▪ 주요 메소드

- 특정 객체를 찾는 메소드: first(), last(), lower(), higher(), …
- 정렬 메소드: descendingIterator(), descendingSet()
- 범위 검색 메소드: headSet(), tailSet, subSet()

5절. 검색 기능을 강화시킨 컬렉션

❖ TreeMap

- 특징
  - 이진 트리(binary tree) 를 기반으로 한 Map 컬렉션
  - 키와 값이 저장된 Map.Entry를 저장
  - 왼쪽과 오른쪽 자식 노드를 참조하기 위한 두 개의 변수로 구성



- 주요 메소드
  - 단일 노드 객체를 찾는 메소드: firstEntry(), lastEntry(), lowerEntry(), higherEntry(), …
  - 정렬 메소드: descendingKeySet(), descendingMap()
  - 범위 검색 메소드: headMap(), tailMap, subMap()

5절. 검색 기능을 강화시킨 컬렉션

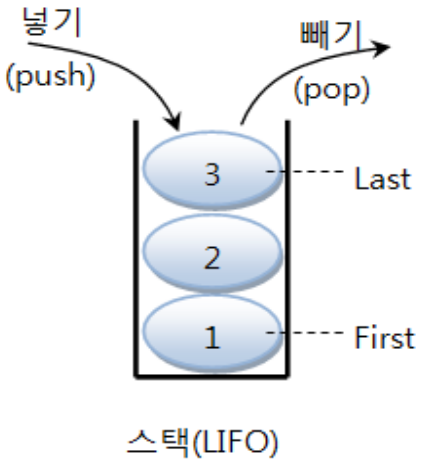
❖ Comparable과 Comparator

- TreeSet과 TreeMap의 자동 정렬
  - TreeSet의 객체와 TreeMap의 키는 저장과 동시에 자동 오름차순 정렬
  - 숫자(Integer, Double)타입일 경우에는 값으로 정렬
  - 문자열(String) 타입일 경우에는 유니코드로 정렬
  - TreeSet과 TreeMap은 정렬 위해 java.lang.Comparable을 구현 객체를 요구
    - Integer, Double, String은 모두 Comparable 인터페이스 구현
    - Comparable을 구현하고 있지 않을 경우에는 저장하는 순간 ClassCastException 발생

6절. LIFO와 FIFO 컬렉션

❖ Stack 클래스

```
Stack<E> stack = new Stack<E>();
```



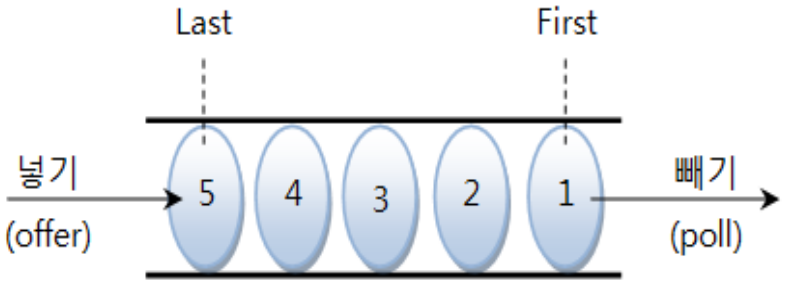
- 특징
  - 후입선출(LIFO: Last In First Out) 구조
  - 응용 예: JVM 스택 메모리
- 주요 메소드

리턴타입	메소드	설명
E	push(E item)	주어진 객체를 스택에 넣는다.
E	peek()	스택의 맨위 객체를 가져온다. 객체를 스택에서 제거하지는 않는다.
E	pop()	스택의 맨위 객체를 가져온다. 객체를 스택에서 제거한다.

6절. LIFO와 FIFO 컬렉션

❖ Queue 인터페이스

```
Queue queue = new LinkedList();
```



큐(FIFO)

- 특징
  - 선입선출(FIFO: First In First O
  - 응용 예: 작업 큐, 메시지 큐, …
  - 구현 클래스: LinkedList
- 주요 메소드

리턴타입	메소드	설명
boolean	offer(E e)	주어진 객체를 넣는다.
E	peek()	객체 하나를 가져온다. 객체를 큐에서 제거하지 않는다.
E	poll()	객체 하나를 가져온다. 객체를 큐에서 제거한다.

7절. 동기화된(synchronized) 컬렉션

❖ 비 동기화된 컬렉션을 동기화된 컬렉션으로 래핑

- Collections의 synchoronizedXXX() 메소드 제공
  - 772 페이지의 예제 그림으로 쉽게 이해하도록!!!

리턴 타입	메소드(매개 변수)	설명
List<T>	synchronizedList(List<T> list)	List를 동기화된 List로 리턴
Map<K,V>	synchronizedMap(Map<K,V> m)	Map을 동기화된 Map으로 리턴
Set<T>	synchronizedSet(Set<T> s)	Set을 동기화된 Set으로 리턴

8절. 병렬 처리를 위한 컬렉션

❖ 동기화(Synchronized) 컬렉션의 단점

- 하나의 스레드가 요소 처리할 때 전체 잠금 발생
  - 다른 스레드는 대기 상태
  - 멀티 스레드가 병렬적으로 컬렉션의 요소들을 빠르게 처리할 수 없음

8절. 병렬 처리를 위한 컬렉션

- 컬렉션 요소를 병렬처리하기 위해 제공되는 컬렉션
  - ConcurrentHashMap
    - 부분(segment) 잠금 사용
      - 처리하는 요소가 포함된 부분만 잠금
      - 나머지 부분은 다른 스레드가 변경 가능하게 → 부분 잠금
  - ConcurrentLinkedQueue
    - 락-프리(lock-free) 알고리즘을 구현한 컬렉션
      - 잠금 사용하지 않음
      - 여러 개의 스레드가 동시에 접근하더라도 최소한 하나의 스레드가 성공하도록(안전하게 요소를 저장하거나 얻도록) 처리