제 7 장 제네릭과 컬렉션

자바의 Collections 자료구조

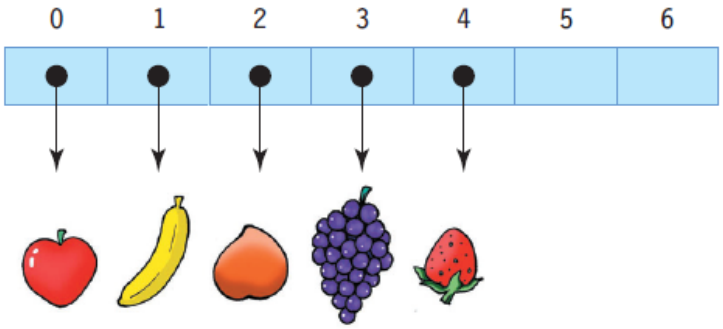
컬렉션(collection)의 개념

- 여러 객체를 보관할 수 있게 만들어진 클래스
- java.util 패키지에 있는 이런 자료구조
- Container 클래스
- 컬렉션 프레임워크(Framework)

컬렉션(collection)의 개념

- 컬렉션
  - 요소(element)라고 불리는 가변 개수의 객체들의 모음
    - 객체들의 컨테이너라고도 불림
    - 요소의 개수에 따라 컬렉션은 자동 크기 조절
    - 컬렉션은 요소의 삽입, 삭제에 따른 요소의 이동 자동 관리
  - 고정 크기의 배열을 다루는 어려움 해소
  - 다양한 객체들의 삽입, 삭제, 검색 등을 관리하기 용이

배열(array)



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

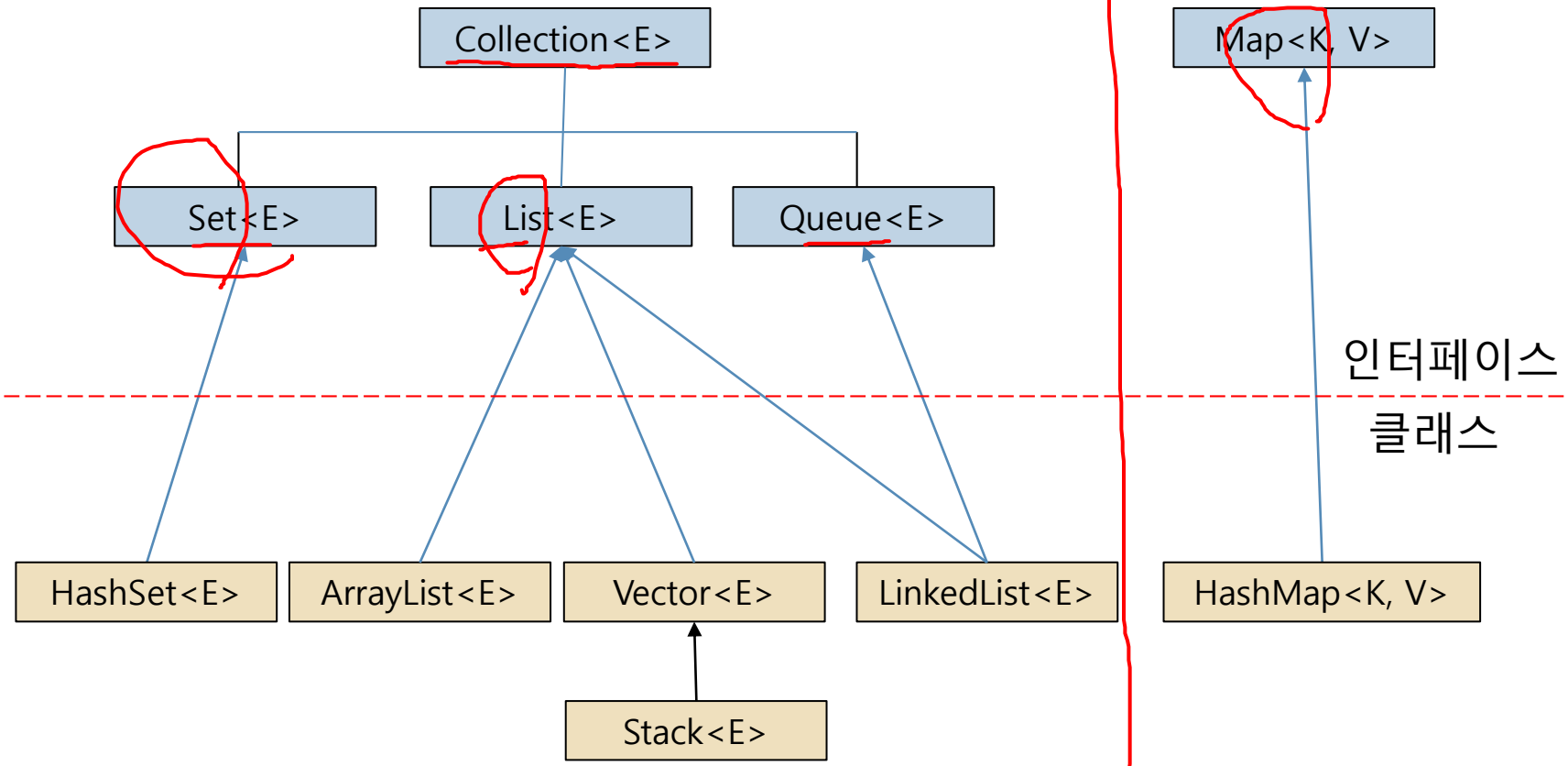
컬렉션(collection)

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

Collections Framework에서 중요한 사항들

- 자료구조 : 여러 개의 데이터가 어떤 형대로 결합되어 있는 가에 대한 얘기
- 자료구조들의 종류는 결국은 어떤 구조에서 얼마나 빨리 원하는 데이터를 찾는가에 따라 결정된다.
  - 순서를 유지할 것인가?
  - 중복을 허용할 것인가?
  - 몇 번 만에 데이터를 찾아낼 수 있는가?
  - 다른 자료구조들에 비해서 어떤 단점과 장점을 가지고 있는가?

컬렉션을 위한 인터페이스와 클래스



데이터의 저장 형태에 따른 자료구조 용어

- List
  - 순서를 가지고 있으며, 중복을 허용하는 보관구조(인덱스 번호가 중요한 역할을 함)
- Set
  - 순서를 가지지 않고, 데이터의 중복을 허용하지 않는 구조
- Map
  - 키와 값을 가지며, 키를 가지고 원하는 데이터를 검색하는 구조

API에서 자료구조를 지원하는 방식

- 자료구조마다 특징을 가지고 있는데 이것을 인터페이스로 정의
- 특정 클래스는 이런 자료구조의 인터페이스를 각자 구현하는 형태로 지원
- 따라서 하나의 자료구조를 여러 개의 클래스가 지원하기도 하고 , 여러 가지 자료구조를 하나의 클래스가 다 구현하기도 한다.

Element 요소

- 자료구조에서 공통으로 사용되는 용어 객체
  - Element라는 용어는 자료구조 안에 들어가는 데이터를 의미
  - Java에서는 모든 자료구조가 객체 자료형을 처리하기 때문에 Element라고 하면 실제 객체를 지칭

equals( )와 hashCode( )

- 자료구조 안에 들어가는 객체들은 단순히 메모리상의 위치 비교를 하는 것이 아니다.
- 때로는 메모리가 달라도 검색이 가능해야 하는데 이를 위해서 java.lang.Object의 equals( )와 hashCode( )메소드가 사용된다.
- equals( ): 메모리상의 위치가 다르다고 해도 검색이 가능하도록 설계하려면 override해 준다.
- hashCode( ): 해싱 알고리즘을 사용할 때 객체들을 분류하고 보관할때 기준이 되는 hash value를 만들어 내는 메소드

Set 계열 인터페이스

- 특징: 순서가 없고, 중복을 허용하지 않음
- 장점: 빠른 속도
- 단점: 단순 집합의 개념으로 정렬하려면 별도의 처리가 필요하다.
- 구현 클래스
  - HashSet
  - TreeSet

```
import java.util.HashSet;

    public class HashSetEx1 {
    public static void main(String[] args) {
    HashSet set = new HashSet();

    set.add("Kim");
    set.add("Lee");
    set.add("Kim");
    set.add("Park");
    set.add("Choi");

    System.out.println(set);
  }
}
```

[Lee, Kim, Park, Choi]

List 계열

- 특징: 순서가 있고, 중복을 허용 (배열과 유사)
- 장점: 가변적인 배열
- 단점: 원하는 데이터가 뒤쪽에 위치하는 경우 속도의 문제
- 방식: equals( )를 이용한 데이터 검색
- 구현 클래스
  - ArrayList
  - Vector
  - LinkedList
  - Stack

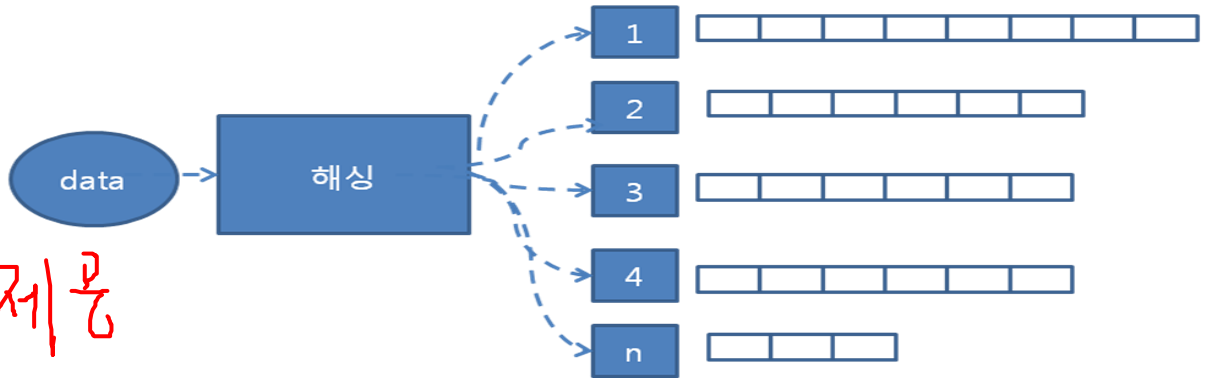
```
import java.util.ArrayList;

public class ArrayListEx1 {
	public static void main(String[] args) {
	ArrayList list = new ArrayList();

	list.add("Kim");
	list.add("Lee");
	list.add("Kim");
	list.add("Park");
	list.add("Choi");

	System.out.println(list);
  }
}
```

[Kim, Lee, Kim, Park, Choi]

Map계열

- 특징: Key(키)와 Value(값)으로 나누어 데이터 관리, 순서는 없으며, 키에 대한 중복은 없음
- 장점: 빠른 속도
- 단점: Key의 검색 속도가 검색 속도를 좌우
- 구현 클래스
  - HashMap
  - TreeMap

data의 특정 값을 hashing 해서 분배하는 구조



```
import java.util.HashMap;

public class HashMapEx1 {
	public static void main(String[] args) {
	HashMap map = new HashMap();

	map.put("1", "apple");
	map.put("2", "banana");
	map.put("3", "peach");
	map.put("2", "mango");
	map.put("4", "apple");

	System.out.println(map);

	}
}
```

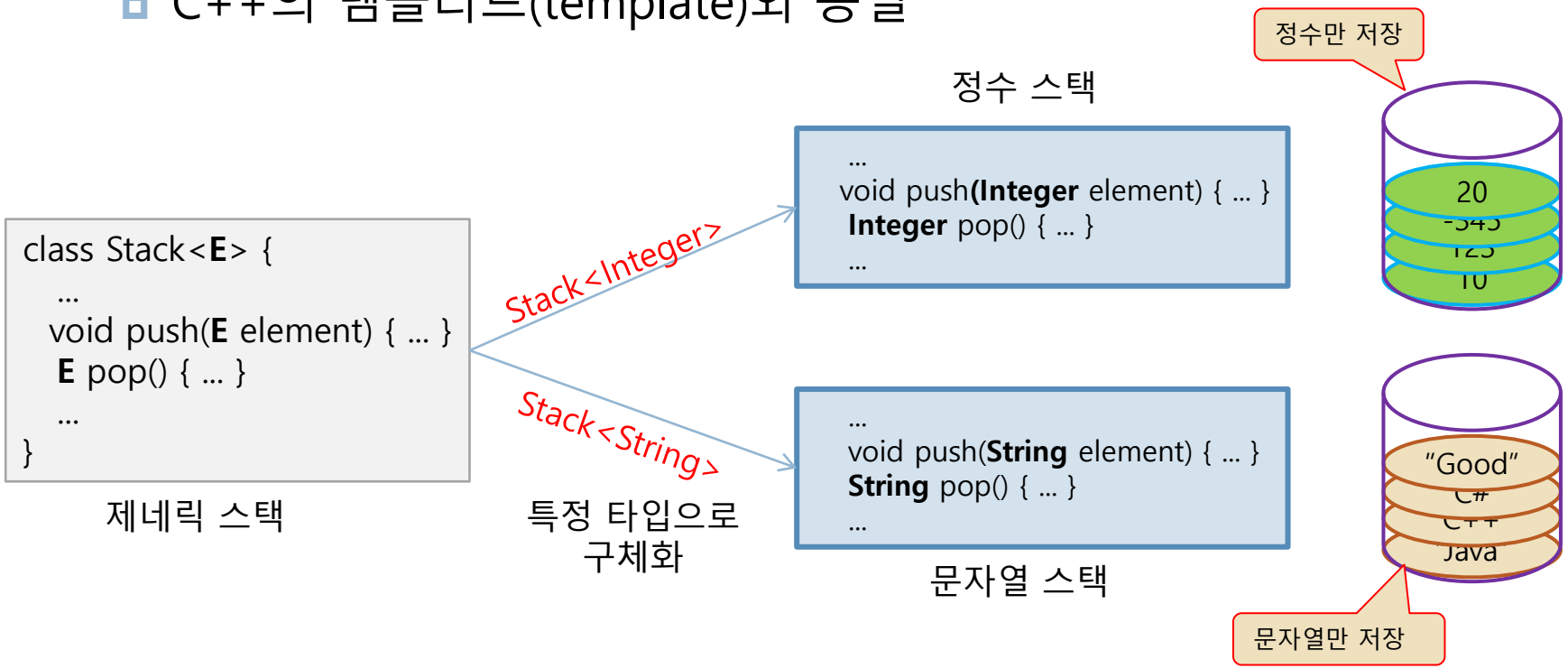
```
{1=apple, 2=mango, 3=peach, 4=apple}
```

컬렉션과 제네릭

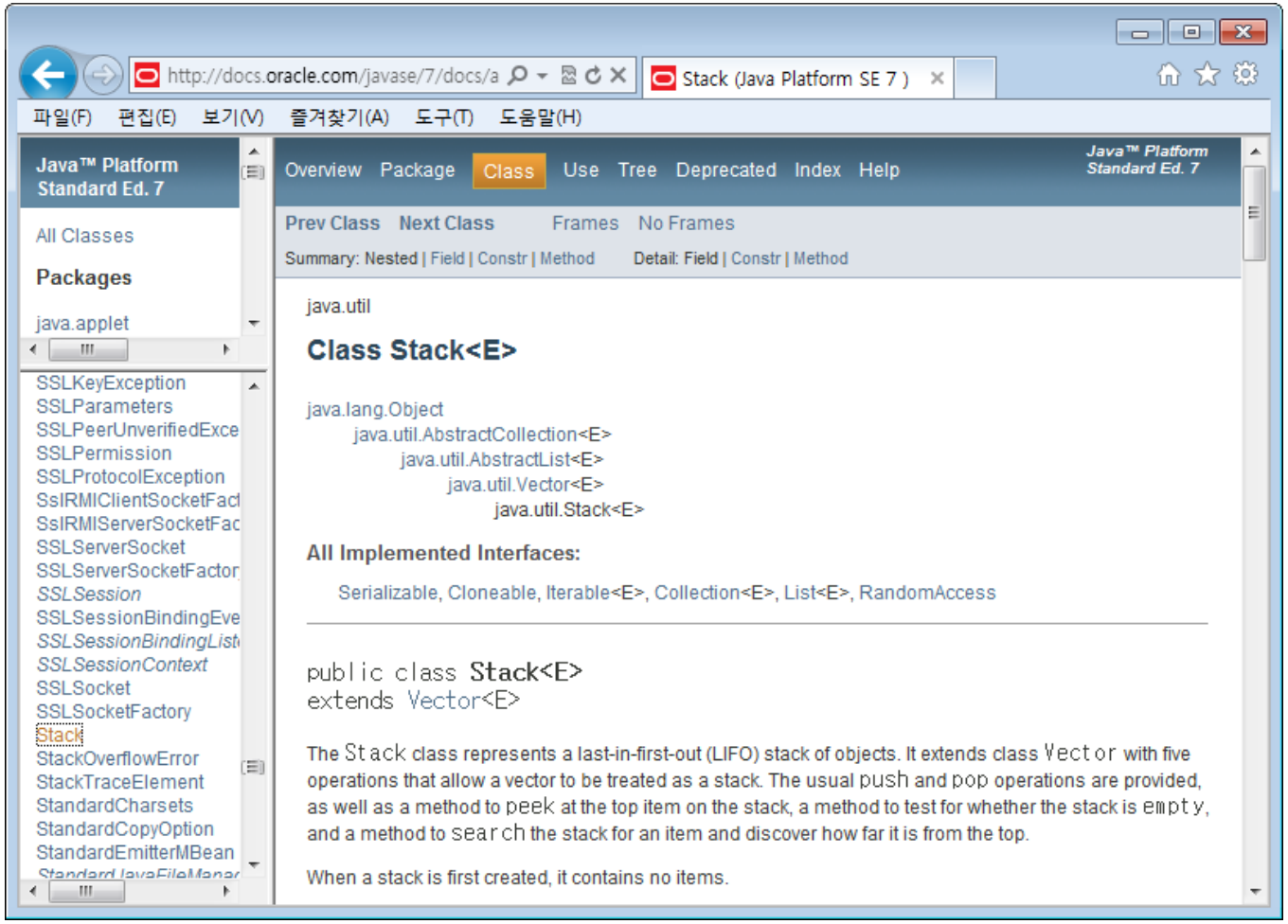
- 컬렉션은 제네릭(generics) 기법으로 구현됨
- 컬렉션의 요소는 객체만 사용 가능
  - 기본적으로 int, char, double 등의 기본 타입 사용 불가
    - JDK 1.5부터 자동 박싱/언박싱 기능으로 기본 타입 사용 가능
- 제네릭
  - 특정 타입만 다루지 않고, 여러 종류의 타입으로 변신할 수 있도록 클래스나 메소드를 일반화시키는 기법
    - <E>, <K>, <V> : 타입 매개 변수
      - 요소 타입을 일반화한 타입
  - 제네릭 클래스 사례
    - 제네릭 벡터 : Vector<E>
  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

제네릭의 기본 개념

- JDK 1.5에서 도입(2004년 기점)
- 모든 종류의 데이터 타입을 다룰 수 있도록 일반화된 타입 매개 변수로 클래스나 메소드를 작성하는 기법
  - C++의 템플리트(template)와 동일



제네릭 Stack<E> 클래스의 JDK 매뉴얼



Vector<E>

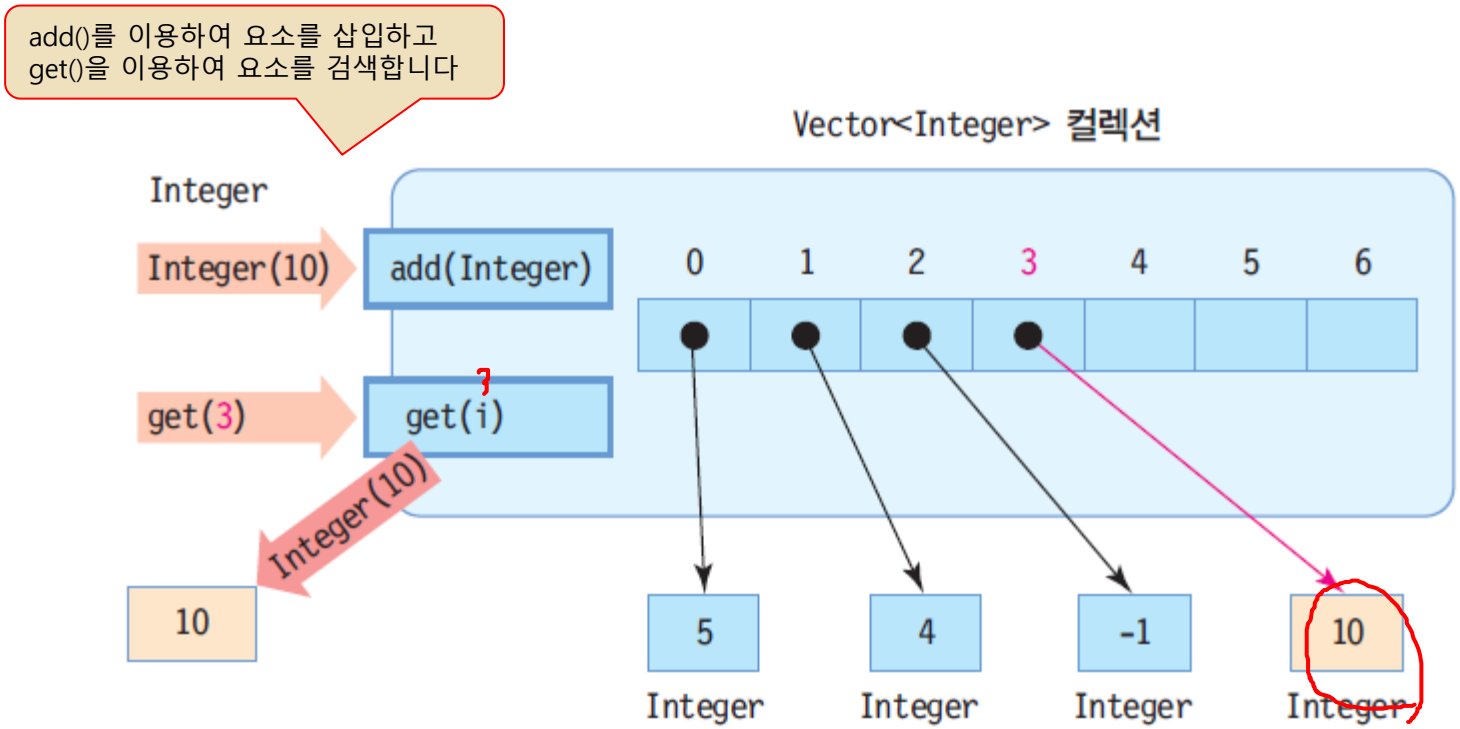
List 중복 O 순서 O

- 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 타입(Wrapper 객체로 만들든지, 자동박싱/언박싱 사용하든지)
  - Vector에 객체 삽입
    - 벡터의 맨 뒤에 객체 추가 : 공간이 모자라면 자동 늘림
    - 벡터 중간에 객체 삽입 : 삽입된 뒤의 객체는 뒤로 하나씩 이동
  - Vector에서 객체 삭제
    - 임의의 위치에 있는 객체 삭제 가능 : 객체 삭제 후 자동 자리 이동

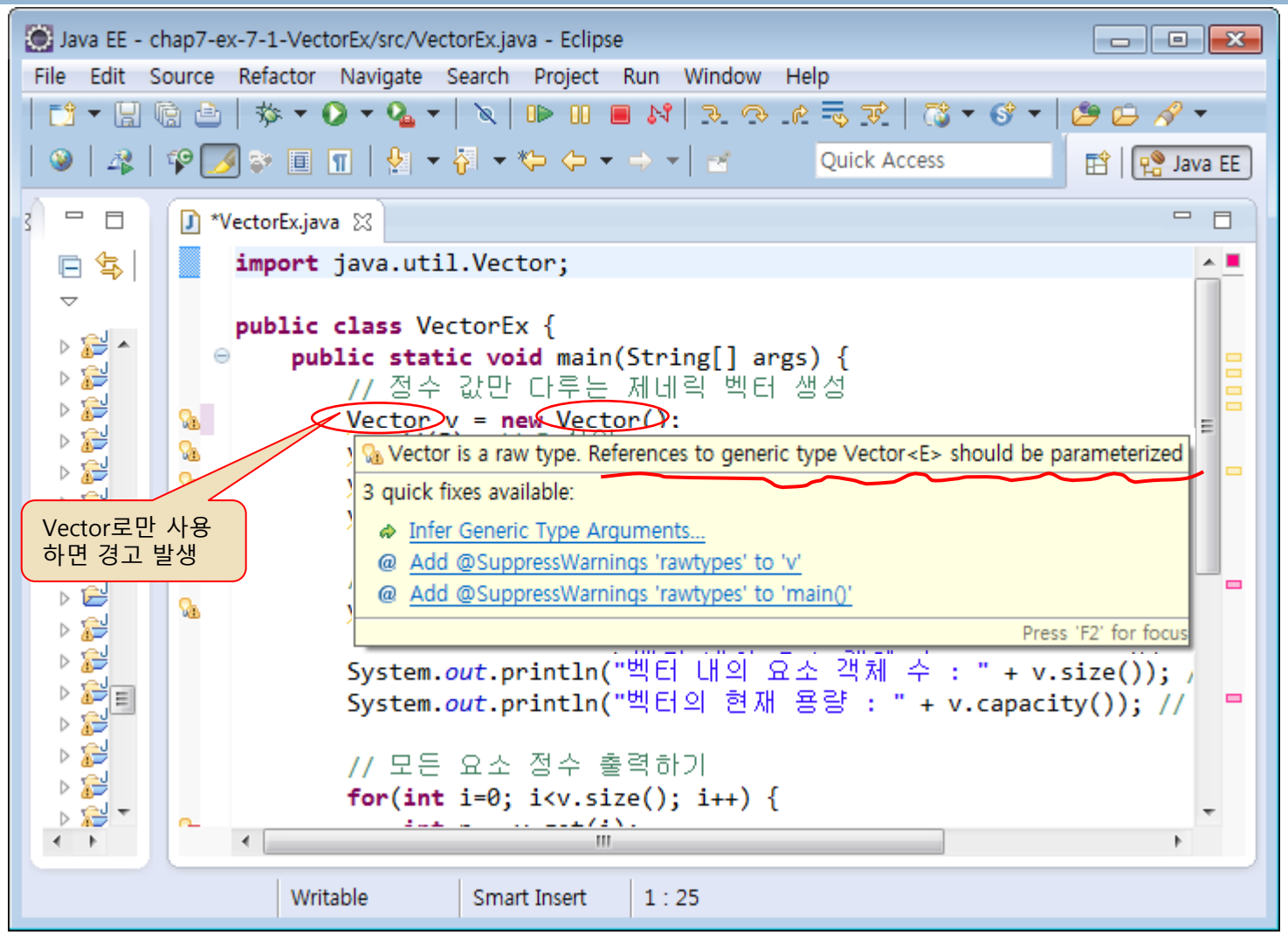
10 -20 -40

Vector<Integer> 컬렉션 내부 구성

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



타입 매개 변수 사용하지 않는 경우 경고 발생



Vector<Integer>나 Vector<String> 등 타입 매개 변수를 사용하여야 함

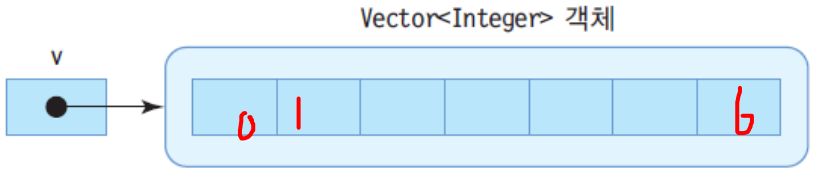
Vector<E> 클래스의 주요 메소드

ArrayList

메소드	설명
boolean add(E e)	벡터의 맨 뒤에 요소 추가
void add(int index, E element)	지정된 인덱스에 지정된 객체를 삽입
int capacity()	벡터의 현재 용량 반환
boolean addAll(Collection<? extends E> c)	c가 지정하는 컬렉션의 모든 요소를 벡터의 맨 뒤에 추가
void clear()	벡터의 모든 요소 삭제
boolean contains(Object o)	벡터가 지정된 객체를 포함하고 있으면 true 반환
E elementAt(int index)	지정된 인덱스의 요소 반환
E get(int index)	지정된 인덱스의 요소 반환
int indexOf(Object o)	지정된 객체와 같은 첫 번째 요소의 인덱스 반환. 없으면 -1 반환
boolean isEmpty()	벡터가 비어있으면 true 반환
E remove(int index)	지정된 인덱스의 요소 삭제
boolean remove(Object o)	지정된 객체와 같은 첫 번째 요소를 벡터에서 삭제
void removeAllElements()	벡터의 모든 요소를 삭제하고 크기를 0으로 만듬
int size()	벡터가 포함하는 요소의 개수 반환
Object[] toArray()	벡터의 모든 요소를 포함하는 배열을 반환

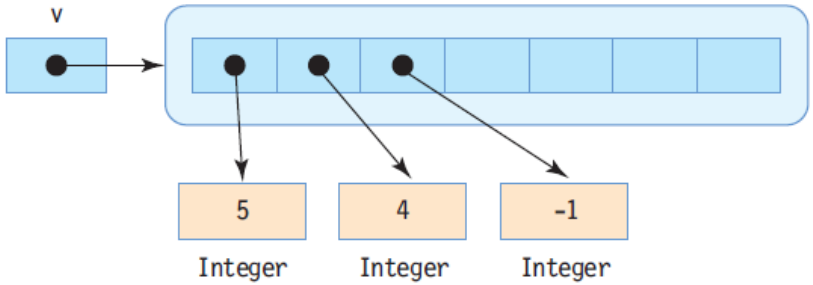
벡터 생성

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



요소 삽입

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



요소 개수 n
벡터의 용량 c

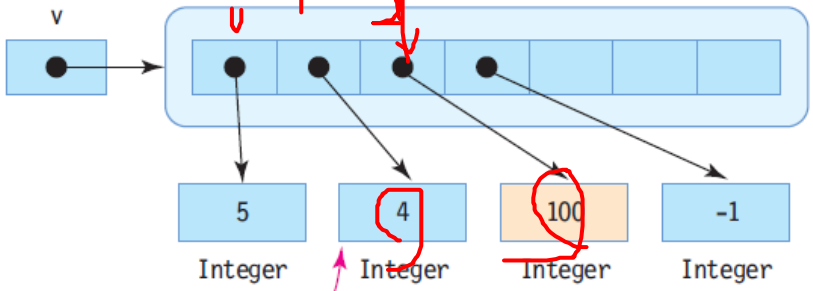
```
int n = v.size(); // n은 3
int c = v.capacity(); // c는 7
```

n = 3
c = 7

요소 중간 삽입

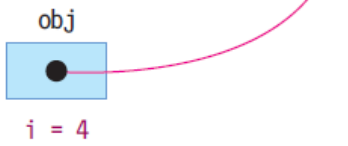
```
v.add(2, 100);
```

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



요소 얻어내기

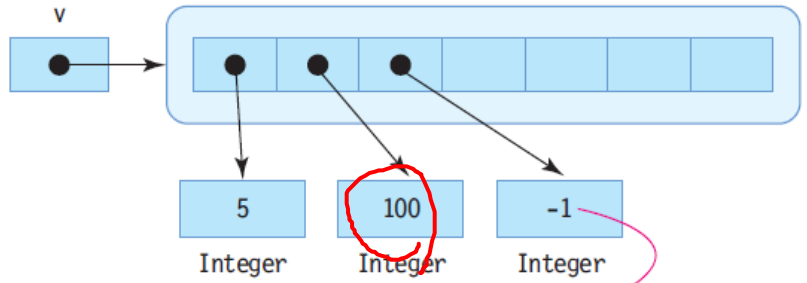
```
Integer obj = v.get(1);
int i = obj.intValue();
```



요소 삭제

```
v.remove(1);
```

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



마지막 요소

```
int last = v.lastElement();
```

last = -1

모든 요소 삭제

```
v.removeAllElements();
```

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));
v.add(new Character('r'));
v.add(new Double(3.14));
```

  - 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/언박싱의 기능 추가

```
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
```

예제 7-1 : 정수 값만 다루는 Vector<Integer>

정수 값만 다루는 제네릭 벡터를 생성하고 활용하는 사례를 보인다.
다음 코드에 대한 결과는 무엇인가?

```
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);
   }
}
```

```
벡터내의 요소 객체 수 :4
벡터의 현재 용량 :10
5
4
100
-1
108
```

예제 7-2 Point 클래스의 객체들만 저장하는 벡터 만들기

(x, y) 한 점을 추상화한 Point 클래스를 만들고
Point 클래스의 객체만 저장하는 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));

        // 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
        }
    }
}
```

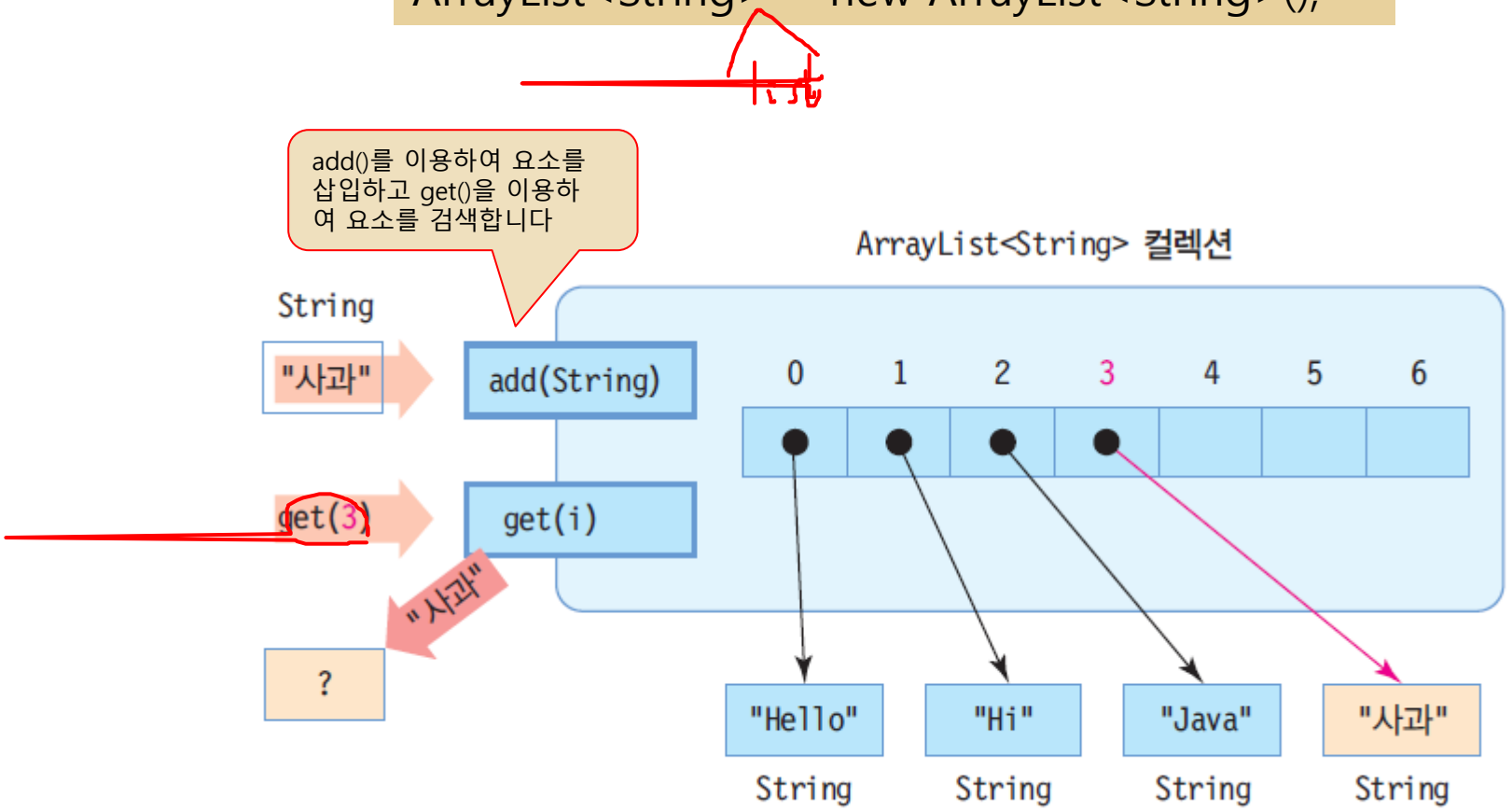
```
(2,3)
(-5,20)
(30,-8)
```

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 타입(Wrapper 객체로 만들든지, 자동박싱/언박싱 사용하든지)
  - ArrayList에 객체 삽입/삭제
    - 리스트의 맨 뒤에 객체 추가 : 공간이 모자라면 자동 늘림
    - 리스트의 중간에 객체 삽입 : 삽입된 뒤의 객체는 뒤로 하나씩 이동
    - 임의의 위치에 있는 객체 삭제 가능 : 객체 삭제 후 자동 자리 이동
  - 벡터와 달리 자동으로 스레드 동기화 지원 않음
    - 다수 스레드가 동시에 ArrayList에 접근할 때 동기화시키지 않음
    - 개발자가 스레드 동기화 코드 작성

ArrayList<String> 컬렉션의 내부 구성

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

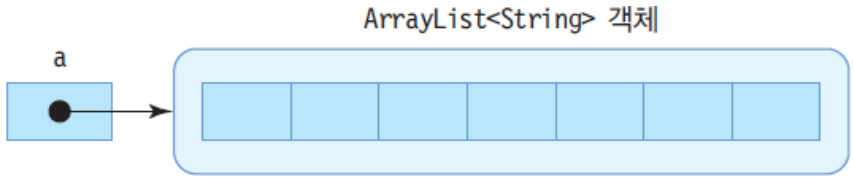


ArrayList<E> 클래스의 주요 메소드

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

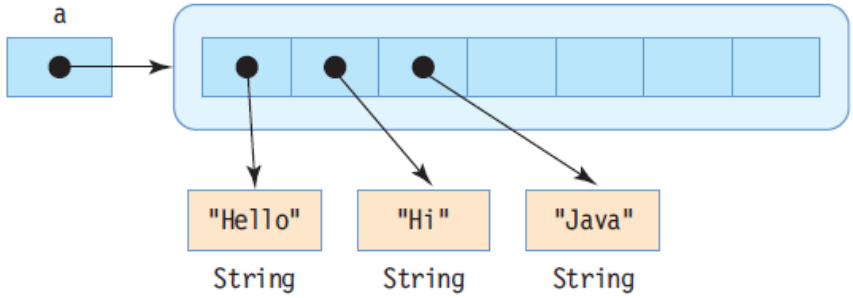
ArrayList 생성

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



요소 삽입

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



요소 개수 n
용량

```
int n = a.size(); // n은 3
int c = a.capacity(); // capacity() 메소드 없음
```

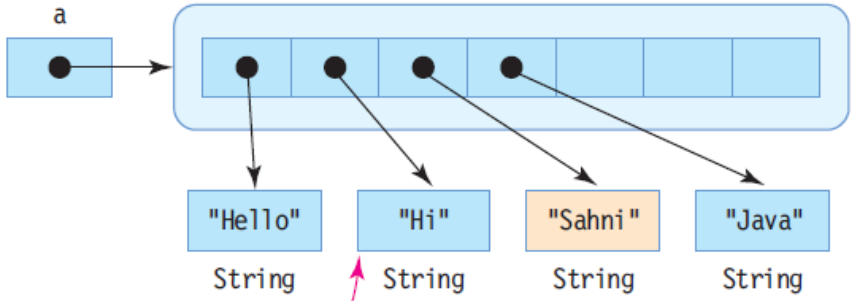
Vector

n = 3

요소 중간 삽입

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

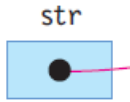
```
a.add(5, "Sahni");
// a.size()보다 큰 위치에 삽입 불가능, 오류
```



요소 알아내기

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

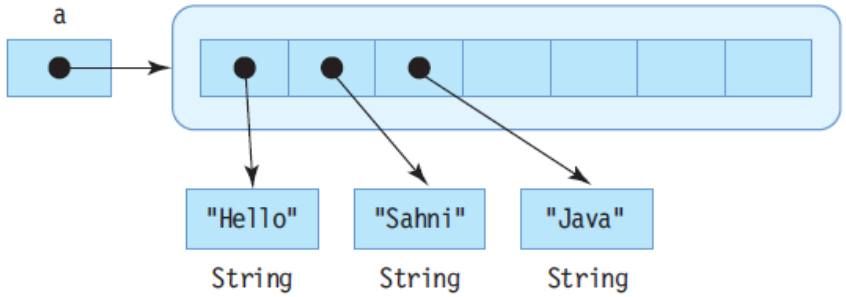
"Hi"



요소 삭제

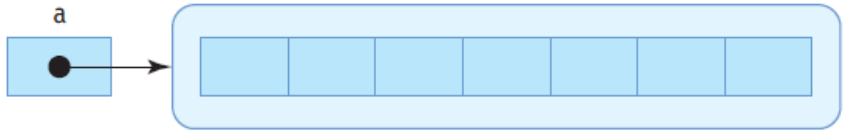
```
a.remove(1);
```

```
a.remove(4); // 오류
```



모든 요소 삭제

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



예제 7-3 : ArrayList에 문자열을 달기

키보드로 문자열을 입력 받아 ArrayList에 삽입하고 가장 긴 이름을 출력하라.

```
import java.util.*;

public class ArrayListEx {
    public static void main(String[] args) {
        // 문자열만 삽입가능한 ArrayList 컬렉션 생성
        ArrayList<String> a = new ArrayList<String>();

        // 키보드로부터 4개의 이름 입력받아 ArrayList에 삽입
        Scanner scanner = new Scanner(System.in);
        for(int i=0; i<4; i++) {
            System.out.print("이름을 입력하세요>>");
            String s = scanner.next(); // 키보드로부터 이름 입력
            a.add(s); // ArrayList 컬렉션에 삽입
        }

        // ArrayList에 들어 있는 모든 이름 출력
        for(int i=0; i<a.size(); i++) {
            // ArrayList의 i 번째 문자열 얻어오기
            String name = a.get(i);
            System.out.print(name + " ");
        }

        // 가장 긴 이름 출력
        int longestIndex = 0;
        for(int i=1; i<a.size(); i++) {
            if(a.get(longestIndex).length() < a.get(i).length())
                longestIndex = i;
        }
        System.out.println("₩n가장 긴 이름은 : " +
                a.get(longestIndex));
    }
}
```

```
이름을 입력하세요>>Mike
이름을 입력하세요>>Jane
이름을 입력하세요>>Ashley
이름을 입력하세요>>Helen
Mike Jane Ashley Helen
가장 긴 이름은 : Ashley
```

컬렉션의 순차 검색을 위한 Iterator

- Iterator<E> 인터페이스
  - 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()	다음 반복에서 사용될 요소가 있으면 true 반환
E next()	다음 요소 반환
void remove()	마지막으로 반환된 요소 제거

  - iterator() 메소드
    - iterator()를 호출하면 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
        ...
}
```

예제 7-4 : Iterator를 이용하여 Vector의 모든 요소 출력하고 합 구하기

Vector<Integer>로부터 Iterator를 얻어내고
벡터의 모든 정수를 출력하고 합을 구하라.

```
import java.util.*;

public class IteratorEx {
    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
        v.add(2, 100); // 4와 -1 사이에 정수 100 삽입

        // Iterator를 이용한 모든 정수 출력하기
        Iterator<Integer> it = v.iterator(); // Iterator 객체 얻기
        while(it.hasNext()) {
            int n = it.next();
            System.out.println(n);
        }
        // Iterator를 이용하여 모든 정수 더하기
        int sum = 0;
        it = v.iterator(); // Iterator 객체 얻기
        while(it.hasNext()) {
            int n = it.next();
            sum += n;
        }
        System.out.println("벡터에 있는 정수 합 : " + sum);
    }
}
}
```

```
5
4
100
-1
벡터에 있는 정수 합 : 108
```

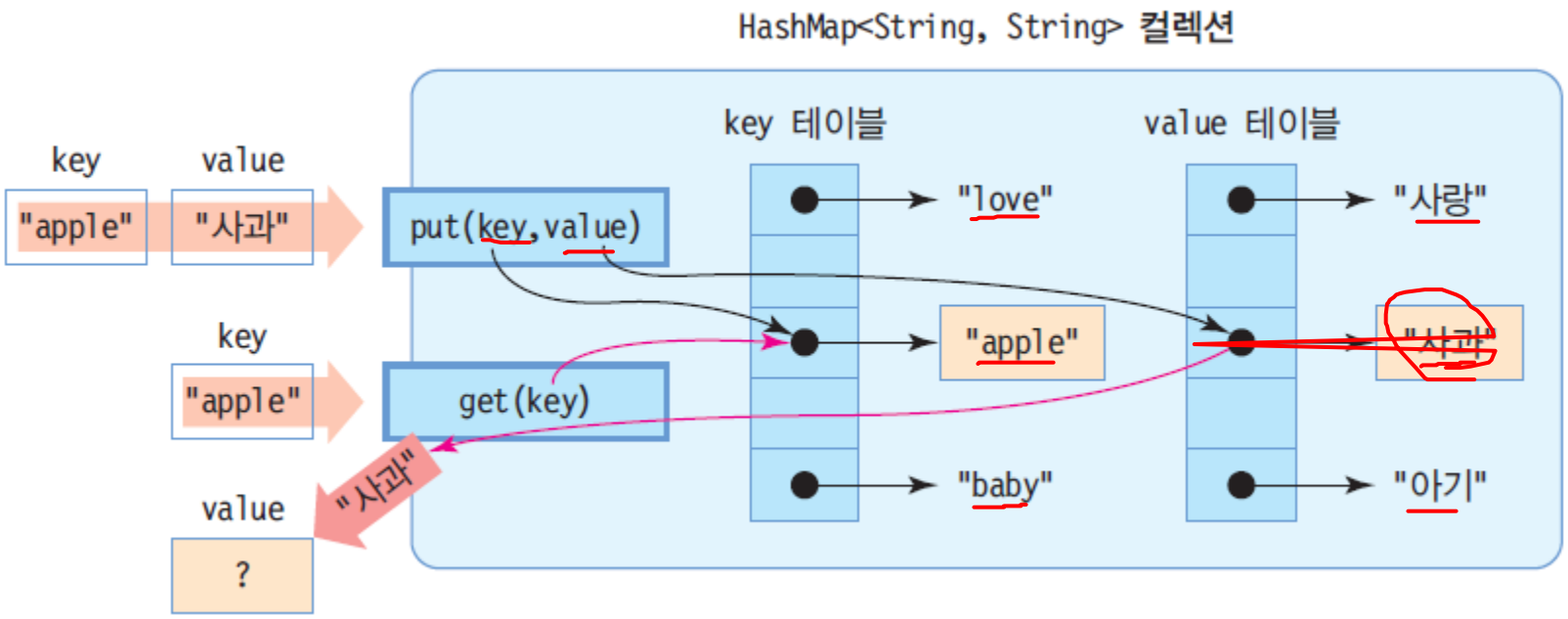
HashMap<K,V>

- HashMap<K,V>의 특성
  - java.util.HashMap
    - K는 키로 사용할 요소의 타입을, V는 값을 사용할 요소의 타입 지정
  - 키(key)와 값(value)의 쌍으로 구성되는 요소를 다루는 컬렉션
    - 키와 값이 한 쌍으로 삽입됨
    - 키는 내부적으로 해시맵에 삽입되는 위치 결정에 사용
    - 값을 검색하기 위해서는 반드시 키 이용
  - 삽입 및 검색이 빠른 특징
  - 요소 삽입 : get() 메소드
  - 요소 검색 : put() 메소드
- HashMap<String, String> 생성, 요소 삽입, 요소 검색

```
HashMap<String, String> h = new HashMap<String, String>();
h.put("apple", "사과"); // "apple" 키와 "사과" 값의 쌍을 해시맵에 삽입
String kor = h.get("apple"); // "apple" 키로 값 검색. kor는 "사과"
```

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

```
HashMap<String, String> map = new HashMap<String, String>();
```

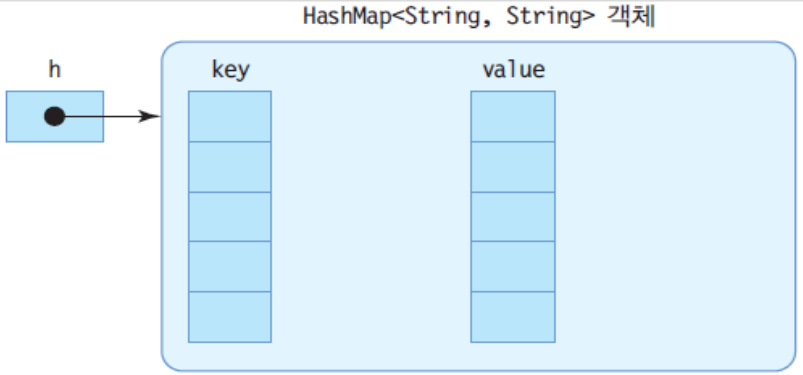


HashMap<K,V>의 주요 메소드

메소드	설명
void clear()	HashMap의 모든 키 삭제
boolean containsKey(Object key)	키를 포함하고 있으면 true 리턴
boolean containsValue(Object value)	하나 이상의 키를 지정된 값에 매핑시킬 수 있으면 true 리턴
V get(Object key)	지정된 키에 맵핑되는 값을 리턴하거나 맵핑되는 값이 없으면 null 리턴
boolean isEmpty()	HashMap이 비어 있으면 true 리턴
Set<K> keySet()	HashMap에 있는 모든 키를 담은 Set<k> 컬렉션 리턴
V put(K key, V value)	key와 value를 매핑하여 HashMap에 저장
V remove(Object key)	지정된 키와 이에 매핑된 모든 값들을 HashMap에서 삭제
int size()	HashMap에 포함된 요소의 개수 리턴

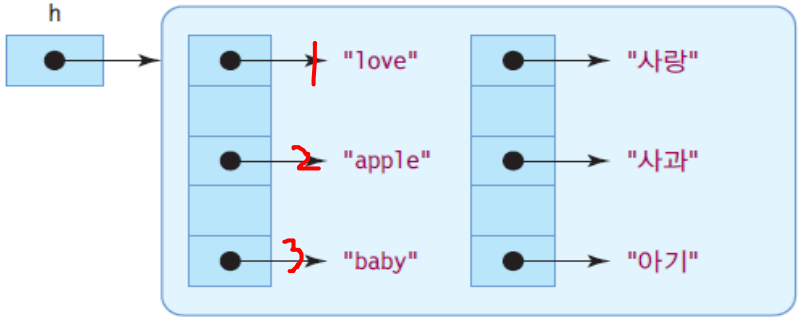
해시맵 생성

```
HashMap<String, String> h =
new HashMap<String, String>();
```



(키, 값) 삽입

```
h.put("baby", "아기");
h.put("love", "사랑");
h.put("apple", "사과");
```



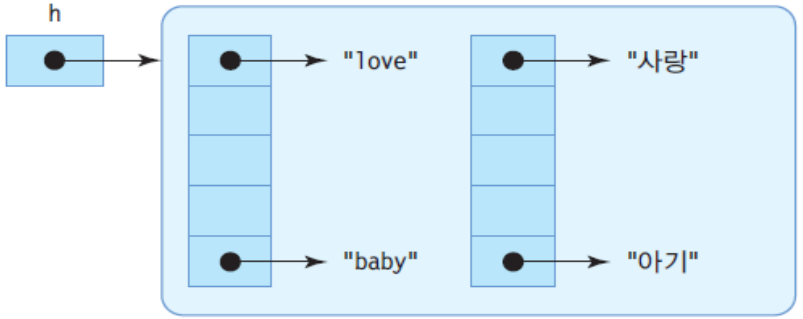
키로 값 읽기

```
String kor = h.get("love");
```

kor = "사랑"

키로 요소 삭제

```
h.remove("apple");
```



요소 개수

```
int n = h.size();
```

n = 2

예제 7-5 : HashMap을 이용하여 영어 단어와 한글 단어를 쌍으로 저장하는 검색하는 사례

영어 단어와 한글 단어를 쌍으로 HashMap에 저장하고
영어 단어로 한글 단어를 검색하는 프로그램을 작성하라.

```
import java.util.*;

public class HashMapDicEx {
    public static void main(String[] args) {
        // 영어 단어와 한글 단어의 쌍을 저장하는 HashMap 컬렉션 생성
        HashMap<String, String> dic = new HashMap<String, String>();

        // 3 개의 (key, value) 쌍을 dic에 저장
        dic.put("baby", "아기"); // "baby"는 key, "아기"은 value
        dic.put("love", "사랑");
        dic.put("apple", "사과");

        // dic 컬렉션에 들어 있는 모든 (key, value) 쌍 출력
        Set<String> keys = dic.keySet(); // key 문자열을 가진 Set 리턴
        Iterator<String> it = keys.iterator();
        while(it.hasNext()) {
            String key = it.next();
            String value = dic.get(key);
            System.out.println("(" + key + "," + value + ")");
        }
        // 영어 단어를 입력 받고 한글 단어 검색
        Scanner scanner = new Scanner(System.in);
        for(int i=0; i<3; i++) {
            System.out.print("찾고 싶은 단어는?");
            String eng = scanner.next();
            System.out.println(dic.get(eng));
        }
    }
}
```

Set

```
(love,사랑)
(apple,사과)
(baby,아기)
찾고 싶은 단어는?apple
사과
찾고 싶은 단어는?babo
null
찾고 싶은 단어는?love
사랑
```

"babo"를 해시맵에서 찾을 수 없기 때문에 null 리턴

예제 7-6 HashMap을 이용하여 자바 과목의 점수를 기록 관리하는 코드 작성

HashMap을 이용하여 학생의 이름과 자바 점수를 기록 관리해보자.

```
import java.util.*;

public class HashMapScoreEx {
    public static void main(String[] args) {
        // 사용자 이름과 점수를 기록하는 HashMap 컬렉션 생성
        HashMap<String, Integer> javaScore =
                new HashMap<String, Integer>();

        // 5 개의 점수 저장
        javaScore.put("한홍진", 97);
        javaScore.put("황기태", 34);
        javaScore.put("이영희", 98);
        javaScore.put("정원석", 70);
        javaScore.put("한원선", 99);

        System.out.println("HashMap의 요소 개수 :" + javaScore.size());

        // 모든 사람의 점수 출력.
        // javaScore에 들어 있는 모든 (key, value) 쌍 출력
         // key 문자열을 가진 집합 Set 컬렉션 리턴
        Set<String> keys = javaScore.keySet();

        // key 문자열을 순서대로 접근할 수 있는 Iterator 리턴
        Iterator<String> it = keys.iterator();
        while(it.hasNext()) {
            String name = it.next();
            int score = javaScore.get(name);
            System.out.println(name + " : " + score);
        }
    }
}
```

```
HashMap의 요소 개수 :5
한원선 : 99
한홍진 : 97
황기태 : 34
이영희 : 98
정원석 : 70
```

예제 7-7 HashMap을 이용한 학생 정보 저장

id와 전화번호로 구성되는 Student 클래스를 만들고, 이름을 '키'로 하고 Student 객체를 '값'으로 하는 해시맵을 작성하라.

```
import java.util.*;

class Student { // 학생을 표현하는 클래스
    int id;
    String tel;
    public Student(int id, String tel) {
        this.id = id; this.tel = tel;
    }
}
```

```
HashMap의 요소 개수 :3
한원선 : 2 010-222-2222
황기태 : 1 010-111-1111
이영희 : 3 010-333-3333
```

출력된 결과는 삽입된 결과와 다르다는 점을 기억하기 바람

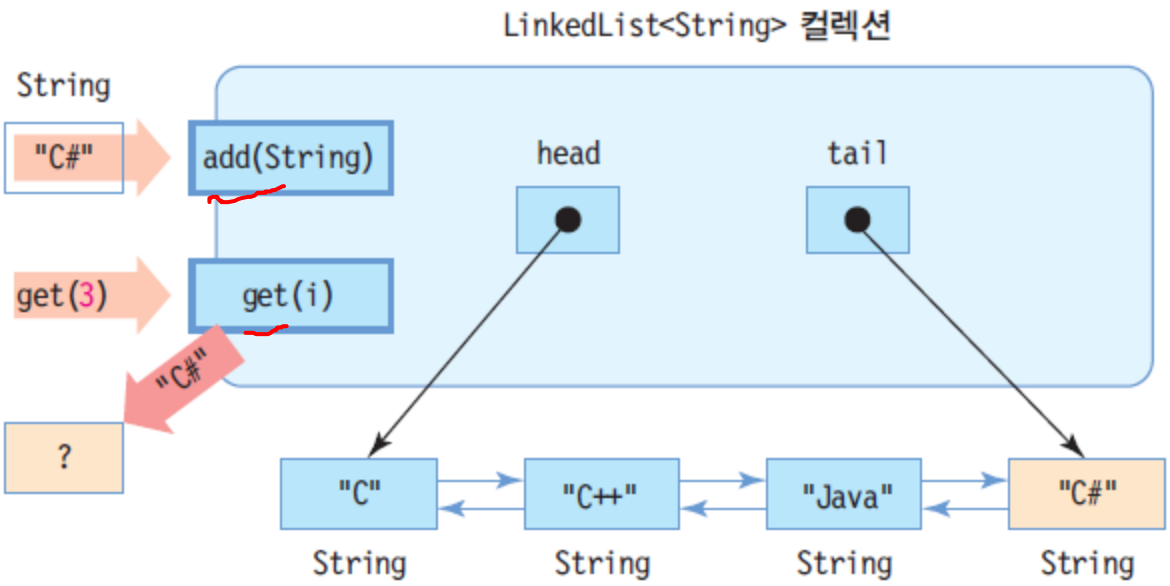
```
public class HashMapStudentEx {
    public static void main(String[] args) {
        // 학생 이름과 Student 객체를 쌍으로 저장하는 HashMap 컬렉션 생성
        HashMap<String, Student> map = new HashMap<String, Student>();

        // 3 명의 학생 저장
        map.put("황기태", new Student(1, "010-111-1111"));
        map.put("한원선", new Student(2, "010-222-2222"));
        map.put("이영희", new Student(3, "010-333-3333"));

        System.out.println("HashMap의 요소 개수 :" + map.size());

        // 모든 학생 출력. map에 들어 있는 모든 (key, value) 쌍 출력
        // key 문자열을 가진 집합 Set 컬렉션 리턴
        Set<String> names = map.keySet();

        // key 문자열을 순서대로 접근할 수 있는 Iterator 리턴
        Iterator<String> it = names.iterator();
        while(it.hasNext()) {
            String name = it.next(); // 다음 키. 학생 이름
            Student student = map.get(name);
            System.out.println(name + " : " + student.id + " " + student.tel);
        }
    }
}
```

LinkedList<E>

- LinkedList<E>의 특성
  - java.util.LinkedList
    - E에 요소로 사용할 타입 지정하여 구체와
  - 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

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 소팅하는 sort() 메소드
    - 요소의 순서를 반대로 하는 reverse() 메소드
    - 요소들의 최대, 최솟값을 찾아내는 max(), min() 메소드
    - 특정 값을 검색하는 binarySearch() 메소드

예제 7-8 : Collections 클래스의 활용

Collections 클래스를 활용하여 문자열 정렬, 반대로 정렬, 이진 검색 등을 실행하는 사례를 살펴보자.

```
import java.util.*;

public class CollectionsEx {
	static void printList(LinkedList<String> l) {
		Iterator<String> iterator = l.iterator();
		while (iterator.hasNext()) {
			String e = iterator.next();
			String separator;
			if (iterator.hasNext())
				separator = "->";
			else
				separator = "₩n";
			System.out.print(e+separator);
		}
	}
```

```
	public static void main(String[] args) {
		LinkedList<String> myList = new LinkedList<String>();
		myList.add("트랜스포머");
		myList.add("스타워즈");
		myList.add("매트릭스");
		myList.add(0,"터미네이터");
		myList.add(2,"아바타");

		Collections.sort(myList); // 요소 정렬
		printList(myList); // 정렬된 요소 출력

		Collections.reverse(myList); // 요소의 순서를 반대로
		printList(myList); // 요소 출력

		int index = Collections.binarySearch(myList, "아바타") + 1;
		System.out.println("아바타는 " + index + "번째 요소입니다.");
	}
}
```

static 메소드이므로 클래스 이름으로 바로 호출

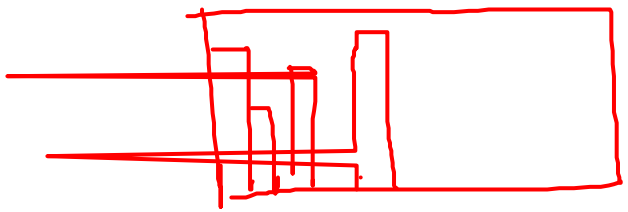
소팅된 순서대로 출력

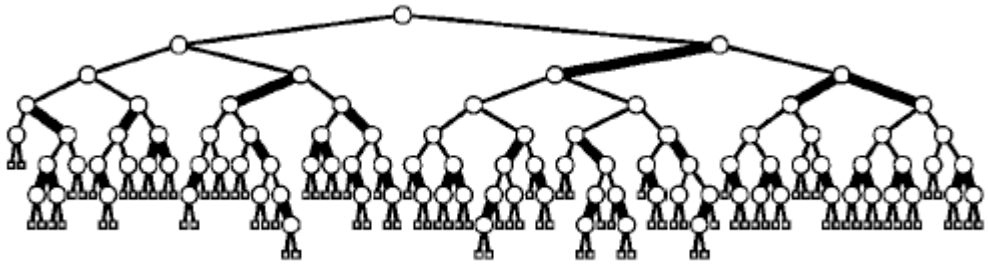
거꾸로 출력

```
매트릭스->스타워즈->아바타->터미네이터->트랜스포머
트랜스포머->터미네이터->아바타->스타워즈->매트릭스
아바타는 3번째 요소입니다.
```

Tree 계열

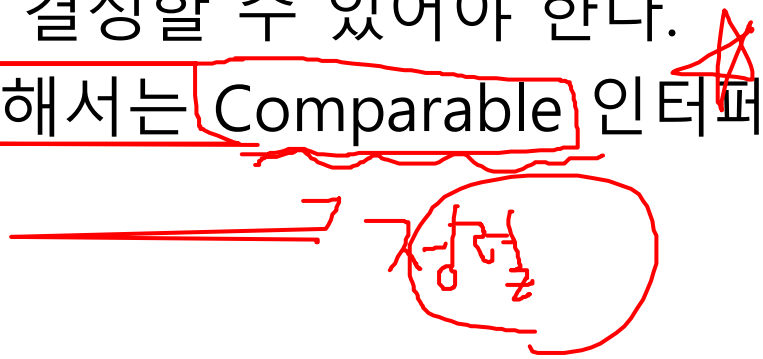
- 각 Element(Node)들의 균형된 분포가 관건
- Balanced Tree - Node들의 분포가 고른 Tree구조
  - 어떤 Node를 찾아도 검색 depth가 일정하게 구성
- 구현 클래스
  - TreeSet
  - TreeMap





Tree 계열

- 정리를 잘하는 Tree로 시작하는 클래스
- Tree로 시작하는 클래스는 데이터를 넣을 때 위치를 찾아간다.
- 이 때문에 Tree로 시작하는 자료구조에 어떤 객체를 넣을 때는 반드시 순서를 결정할 수 있어야 한다.
- 이 순서를 결정하기 위해서는 Comparable 인터페이스를 사용



Tree 계열

- TreeSet
- TreeMap
- Tree가 붙는 클래스는 내부적으로 데이터가 들어가는 순간에 이미 위치를 결정
- Tree가 붙는 클래스를 이용할 때에는 정렬할 수 있는 데이터 만들어서 넣는 것이 핵심

내가 만든 클래스가 정렬기능을 가질려면?

- Comparable 인터페이스 구현
  - public int compareTo(T o) 메소드 오버라이딩
  - 정렬시 앞으로 가야 하는 데이터는 음수 반환
  - 어떤 데이터와 자신이 같은 데이터이면 0
  - 자신이 뒤로 가야 하는 데이터면 양수를 반환

- Comparator 인터페이스 구현
  - Compare(T o1, T o2) 오버라이딩

정렬되는 모든 객체들의 Interface

- Java에서의 정렬은 각 객체가 다른 객체와의 비교를 객체 스스로가 판단해서 동작하는 방식
- 순서가 있는 자료구조 안에 들어가는 모든 객체들은 직접 Comparable 인터페이스를 구현
- Comparator를 인터페이스를 구현한 객체를 활용하면 자료구조가 원하는 방식으로 정렬 방식을 선택 가능

```
public class PlayerVO  {
	private String name;
	private String position;
	private int regYear;

	public PlayerVO(String name, String position, int regYear) {
		this.name = name;
		this.position = position;
		this.regYear = regYear;
	}

	public String toString() {
		return name+":"+position+":"+regYear;
	}

}
```

```
public class PlayerSortTest {
	public static void main(String[] args) {
		ArrayList<PlayerVO> list = new ArrayList<PlayerVO>();

		list.add(new PlayerVO("홍길동", "투수", 1999));
		list.add(new PlayerVO("강감찬", "포수", 2005));
		list.add(new PlayerVO("임꺽정", "1루수", 2003));
		list.add(new PlayerVO("을지문득", "2루수", 2010));

		System.out.println(list);
		Collections.sort(list);
		System.out.println(list);

	//
		//특정 데이터에 대해 정렬하려면 Comparator인터페이스를 구현
		Comparator comparator = new YearComparator();
		Collections.sort(list, comparator);
		System.out.println(list);
	}
}
```

Comparable

//[홍길동:투수:1999, 강감찬:포수:2005, 임꺽정:1루수:2003, 을지문득:2루수:2010]
//정렬시도
//정렬시도
//정렬시도
//정렬시도
//정렬시도
//정렬시도
//[강감찬:포수:2005, 을지문득:2루수:2010, 임꺽정:1루수:2003, 홍길동:투수:1999]
//[홍길동:투수:1999, 임꺽정:1루수:2003, 강감찬:포수:2005, 을지문득:2루수:2010]

```
//다른 기준으로 정렬
//데이터를 단순히 하나의 기준이 아니라 다른 기준으로 정렬할려면 Comparator 인터페이스 구현
public class PlayerVO implements Comparable<PlayerVO> {
	private String name;
	private String position;
	private int regYear;

	public PlayerVO(String name, String position, int regYear) {
		this.name = name;
		this.position = position;
		this.regYear = regYear;
	}

	public String toString() {
		return name+":"+position+":"+regYear;
	}

	@Override
	public int compareTo(PlayerVO otherPlayer) {
		System.out.println("정렬시도");
		return this.name.compareTo(otherPlayer.name);
	}

	public int getRegYear() {
		return this.regYear;
	}
}
```

```
import java.util.Comparator;

//특정 데이터에 대해 정렬하려면 Comparator인터페이스 구현
public class YearComparator implements Comparator<PlayerVO> {

	@Override
	public int compare(PlayerVO p1, PlayerVO p2) {
		//년도가 앞서면 음수가 나오게...
		return p1.getRegYear() - p2.getRegYear();
	}

}
```

```
public class HashMapUI {
	public static void main(String[] args) {
		int nMenu = 0;
		boolean bFlag = true;
		String strName = "";
		int nScore = 0;

		Scanner scan = new Scanner(System.in);
		HashMap<String, Integer> map = new HashMap<String, Integer>();

		while(bFlag) {
		System.out.println("--------------------------------");
		System.out.println("1. 성적 입력");
		System.out.println("2. 성적 삭제");
		System.out.println("3. 성적 검색");
		System.out.println("4. 전체성적 출력");
		//System.out.println("5. 전체성적 출력(이름순)");
		//System.out.println("6. 전체성적 출력(성적순)");
		System.out.println("0. 종료");
		System.out.println("--------------------------------");

		nMenu = Integer.parseInt(scan.nextLine());
		System.out.println("0 ~ 4사이의 숫자를 입력해주세요.");

		switch(nMenu) {
		case 0://종료
		System.out.println("종료");
		bFlag = false;
		break;
```

```
case 1:
        System.out.print("이름 : >> ");
        strName = scan.nextLine();

        System.out.print("성적 : >> ");
        nScore = Integer.parseInt(scan.nextLine());

        //HashMap에 이름과 성적을 입력한다.
        map.put(strName, nScore);
         break;

case 2: //성적 삭제
        System.out.print("이름 : >> ");
        strName = scan.nextLine();
        map.remove(strName);
        break;

case 3://성적 검색
        System.out.print("이름 : >>");
        strName = scan.nextLine();
        nScore = map.get(strName);
        System.out.println("이름 : " + strName + ", 점수 : " + nScore);
        break;

case 4://전체성적 조회
        //HashMap에 있는 모든 키들을 가져와 배열에 저장한다.
        Set<String> keys = map.keySet();
        Iterator it = keys.iterator();

        while(it.hasNext()) {
        String name = (String) it.next();
        Integer score = map.get(name);
        System.out.println("이름 : " + name + ", 성적 : " + score);
        }
        break;

default:
        System.out.println("0 ~ 4사이의 숫자를 입력해주세요.");
}
}//end while
}
```

Report

- 이름과 성적에 대해 정렬하는 메뉴를 추가
- 작성한 소스를 pdf 파일로 변환하여 과제방에 올리세요.
- 파일명 : 학번_이름.pdf