Section 01

컬렉션

1. 컬렉션

■ 컬렉션의 필요성

- 컬렉션(collection)은 배열처럼 데이터를 저장하는 데 사용

(a) 배열 (b) 컬렉션

[그림 13-1] 배열과 컬렉션

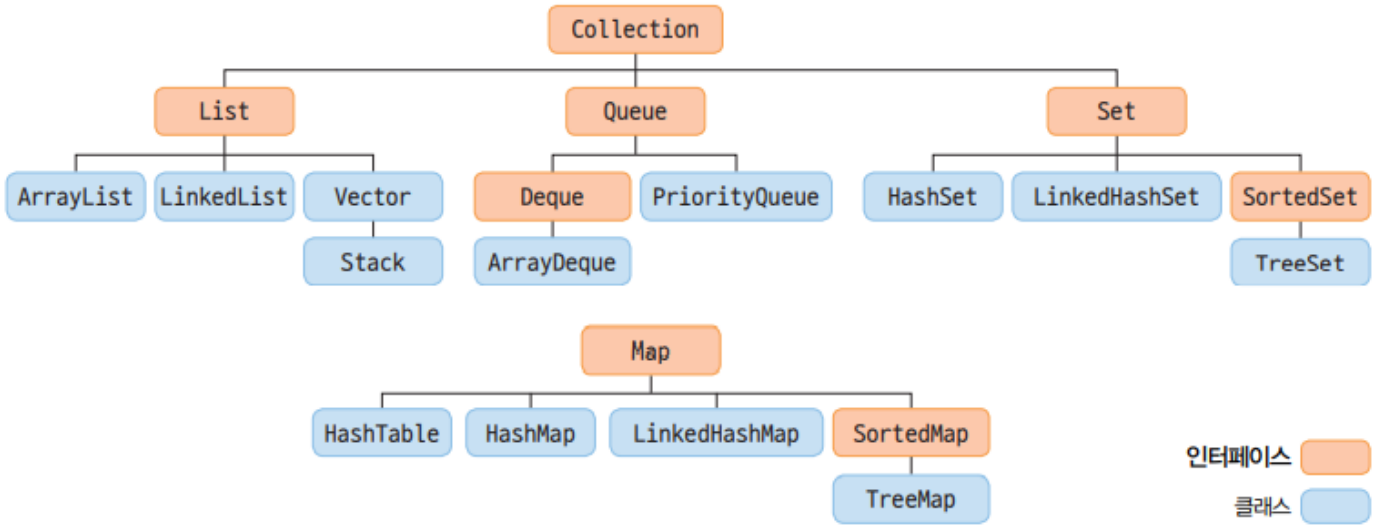
[표 13-1] 배열과 컬렉션의 차이점

배열	컬렉션
크기가 고정되어 있다. 배열을 생성한 후에는 크기를 늘리거나 줄일 수 없다.	크기가 유동적이다. 요구 사항에 따라 크기를 늘리거나 줄일 수 있다.
배열은 기본값(예 int, str형의 값)과 객체를 모두 저장하는 데 사용할 수 있다.	개체를 저장하는 데 사용할 수 있지만 기본값은 사용할 수 없다.
유사한 유형의 데이터를 저장한다.	일부 컬렉션은 유사한 데이터를 저장하는 데 사용할 수 있고, 일부 컬렉션은 유사하거나 다른 유형의 데이터를 저장하는 데 사용할 수 있다.
메모리 효율성이 떨어진다.	메모리 효율성이 높다.
성능 면에서 더 좋다.	성능에 따라 컬렉션을 권장하지 않는다.

1. 컬렉션

컬렉션 프레임워크의 개념

- 많은 데이터를 쉽고 효과적으로 처리할 수 있는 표준화된 방법을 제공하는 클래스의 집합을 말함
- 데이터를 저장하는 자료 구조와 데이터를 처리하는 알고리즘을 구조화하여 클래스로 구현해 놓은 인터페이스



[그림 13-2] 컬렉션 프레임워크의 계층 구조

1. 컬렉션

컬렉션 프레임워크의 개념

- 컬렉션 프레임워크는 객체 그룹을 저장하고 조작하는 데 도움되는 다양한 인터페이스와 클래스를 제공(기본적으로 객체를 저장하기 위해 만들어짐)

[표 13-2] 컬렉션 프레임워크의 주요 인터페이스와 클래스

인터페이스	설명	컬렉션 클래스
List<E>	순서가 있는 데이터의 집합으로, 데이터의 중복을 허용한다.	ArrayList LinkedList Vector
Set<E>	순서가 없는 데이터의 집합으로, 데이터의 중복을 허용하지 않는다.	HashSet TreeSet LinkedHashSet
Queue<E>	요소가 헤드에서만 제거되는 특수한 종류의 목록을 처리하도록 컬렉션을 확장한다.	ArrayDeque PriorityQueue
Map<K, V>	순서가 없는 키와 값 한 쌍으로 이루어진 데이터의 집합으로, 키는 중복될 수 없지만 값은 중복될 수 있다.	HashMap TreeMap HashTable LinkedHashMap

1. 컬렉션

컬렉션 프레임워크의 개념

[표 13-3] 컬렉션 프레임워크의 주요 공통 메서드

메서드	설명
`boolean add(E e)`	컬렉션에 요소를 삽입한다.
`boolean addAll(Collection c)`	호출 컬렉션에 지정된 컬렉션 요소를 삽입한다.
`boolean remove(Object element)`	컬렉션을 확장하여 개체의 시퀀스 목록을 처리한다.
`boolean removeAll(Collection c)`	호출 컬렉션에서 지정된 컬렉션의 모든 요소를 삭제한다.
`boolean removeIf(Predicate filter)`	지정된 술어를 충족하는 컬렉션의 모든 요소를 삭제한다.
`boolean retainAll(Collection c)`	지정된 컬렉션을 제외하고 컬렉션을 호출하는 모든 요소를 삭제한다.
`int size()`	컬렉션에 포함된 모든 요소 개수를 반환한다.
`void clear()`	컬렉션의 모든 요소를 삭제한다.
`boolean contains(Object element)`	요소를 검색한다.
`boolean containsAll(Collection c)`	지정된 컬렉션을 검색한다.

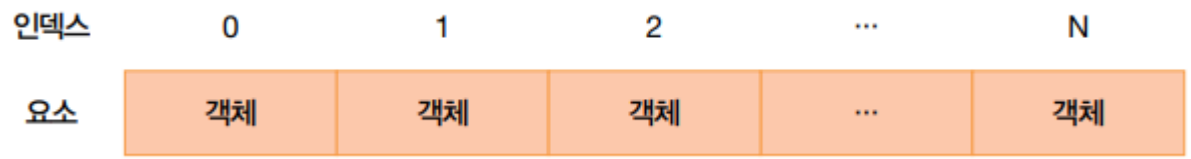
Section 02

List 컬렉션

2. List 컬렉션

List 컬렉션

- Collection 인터페이스의 자식 인터페이스
- 선형 자료 구조의 형태로 요소를 저장함
- 요소를 인덱스로 관리하기 때문에 요소를 저장하면 자동으로 저장 순서의 인덱스가 부여되고, 인덱스로 요소를 검색하거나 삭제할 수 있음



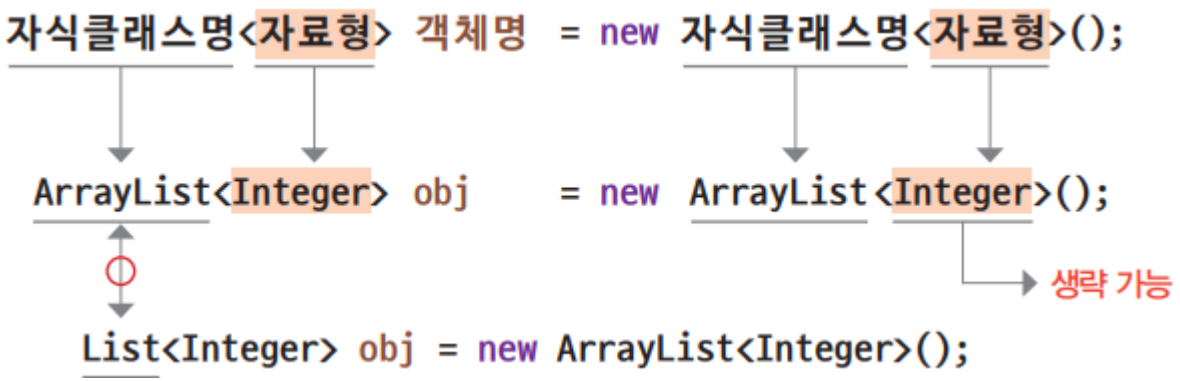
[그림 13-3] List 컬렉션의 구조

2. List 컬렉션

List 컬렉션

- List 컬렉션을 구현하는 대표적인 클래스: ArrayList, LinkedList, Vector

```
자식클래스명<E> 객체명 = new 자식클래스명<E>();
List<E> 객체명 = new 자식클래스명<E>();
```



[그림 13-4] List 컬렉션의 자식 클래스인 ArrayList 선언 형식

2. List 컬렉션

List 컬렉션

- [List 컬렉션의 주요 자식 클래스를 선언하는 예]

```
List<Integer> obj1 = new ArrayList();
   // ArrayList<Integer> obj1 = new ArrayList<Integer>();과 같음

List<Integer> obj2 = new LinkedList();
   // LinkedList<Integer> obj2 = new LinkedList<Integer>();과 같음

List<Integer> obj3 = new Vector();
   // Vector<Integer> obj3 = new Vector<Integer>();과 같음
```

2. List 컬렉션

List 컬렉션

[표 13-4] List 인터페이스의 주요 메서드

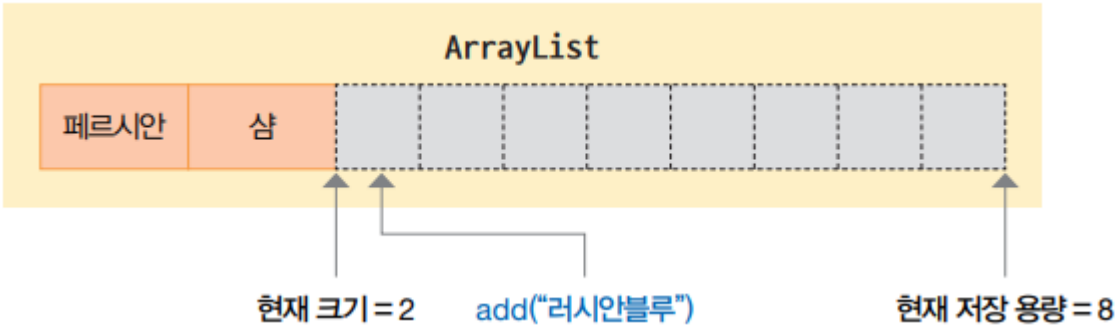
메서드	설명
`boolean add(E e)`	요소를 삽입한다.
`boolean add(int index, E obj)`	주어진 인덱스에 요소를 추가한다.
`Object set(int index, E element)`	지정된 인덱스에 요소를 저장한다.
`boolean contains(Object o)`	지정된 요소를 검색한다.
`Object get(int index)`	지정된 인덱스에 저장된 요소를 반환한다.
`boolean isEmpty()`	컬렉션이 비어 있는지 여부를 확인한다.
`int size()`	모든 요소의 개수를 반환한다.
`void clear()`	모든 요소를 삭제한다.
`boolean remove(Object o)`	지정된 요소를 삭제한다.
`int indexOf(Object o)`	지정된 요소가 처음 나타나는 인덱스를 반환한다.
`int lastIndexOf(Object o)`	지정된 요소가 마지막으로 나타나는 인덱스를 반환한다.
`Object[] toArray()`	모든 요소를 Object형의 배열로 반환한다.

2. List 컬렉션

ArrayList

- List 인터페이스를 구현한 클래스
- 선형 순서로 저장하는 데 사용하는 배열 기반 데이터 구조의 구현을 제공함
- 한 번 생성되면 크기가 변하지 않는 배열과 달리 ArrayList는 크기가 가변적임

→저장 용량을 초과하면 부족한 만큼 자동으로 저장 용량이 늘어나기 때문에 일반적인 배열보다 기능성과 유연성이 뛰어나 널리 사용됨



[그림 13-5] ArrayList 클래스의 구조

2. List 컬렉션

ArrayList

- ArrayList 선언

→ArrayList 클래스에는 비어 있거나 다른 컬렉션의 요소에서 ArrayList를 만드는 데 사용할 수 있는 생성자가 있음

[표 13-5] ArrayList 클래스의 생성자

생성자	설명
`ArrayList()`	빈 ArrayList를 생성한다.
`ArrayList(Collection C)`	Collection C의 요소로 초기화된 ArrayList를 생성한다.
`ArrayList(int initialCapacity)`	지정된 초기 용량을 가진 ArrayList를 생성한다.

→[ArrayList 클래스를 생성하는 예]

```
ArrayList list = new ArrayList();                          // 클래스 유형 미설정 Object로 선언
ArrayList<Integer> num = new ArrayList<Integer>();         // int형의 객체 요소만 사용 가능

ArrayList<Integer> num2 = new ArrayList<Integer>(10);      // 초기 용량을 10으로 지정
```

new에서 클래스 유형 파라미터 생략 가능
ArrayList<Integer> num = new ArrayList<>();

2. List 컬렉션

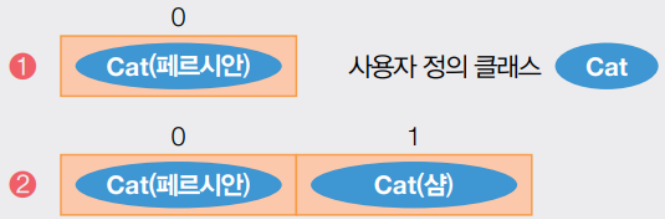
ArrayList

- ArrayList 요소 삽입

→add() 메서드: ArrayList에 요소 삽입, 삽입 순서대로 목록에 요소를 추가함

```
ArrayList<String> cats = new ArrayList<String>();
cats.add("페르시안");          // 값 추가 ❶
cats.add(null);               // null 값 추가 ❷
cats.add(1, "샴");            // 인덱스 1에 값("샴") 추가 ❸
```


```
ArrayList<Cat> cats = new ArrayList<Cat>();
Cat cat = new Cat("페르시안");
cats.add(cat);                // cat 객체 생성 후 추가 ❶
cats.add(new Cat("샴"));      // cat 객체 생성과 동시에 추가 ❷
```



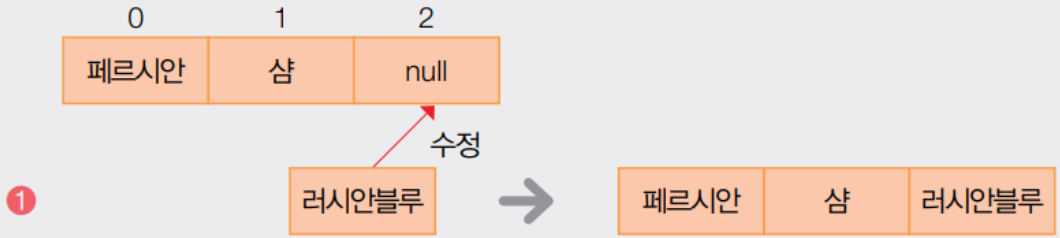
2. List 컬렉션

ArrayList

- ArrayList 요소 수정

→set() 메서드: 요소 수정, 인덱스와 새 요소를 제공하면 해당 인덱스의 요소가 주어진 새 요소로 업데이트됨

```
ArrayList<String> cats = new ArrayList<String>();
cats.set(2, "러시안블루");    // 인덱스가 2인 요소를 "러시안블루"로 변경 ●——❶
```



2. List 컬렉션

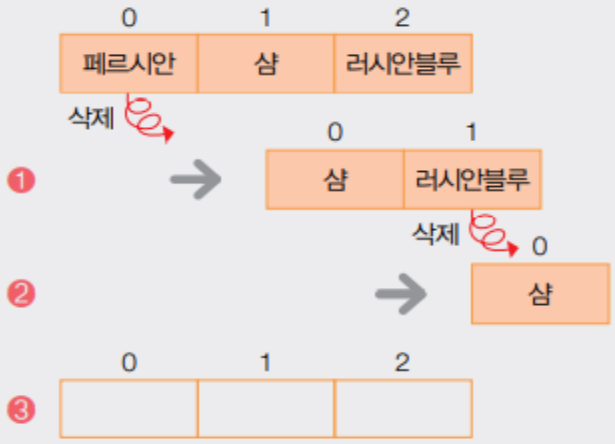
ArrayList

- ArrayList 요소 삭제

→remove() 메서드: 요소 삭제, 인덱스 값을 전달하여 요소를 삭제할 수도 있음

→clear() 메서드: 모든 요소를 삭제

```
ArrayList<String> cats = new ArrayList<String>();
cats.remove("페르시안");      // 요소 삭제 ❶
cats.remove(1);               // 인덱스가 1인 요소 삭제 ❷
cats.clear();                 // 모든 요소 삭제 ❸
```



2. List 컬렉션

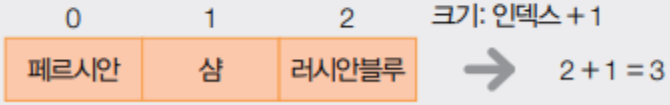
ArrayList

- ArrayList 크기 구하기

→size() 메서드: ArrayList의 크기를 구함

→ArrayList의 크기는 마지막 인덱스에 1을 더한 값임

```
ArrayList<String> cats = new ArrayList<String>();
System.out.println(cats.size());     // cats 크기: 3
```



2. List 컬렉션

ArrayList

- ArrayList 요소 값 출력

→get(index) 메서드: ArrayList의 요소 값을 출력

→ArrayList에서 원하는 인덱스의 요소 값이 반환됨

→전체 요소 값을 출력할 때는 대개 for문을 사용하지만 Iterator문을 사용할 수도 있음

2. List 컬렉션

ArrayList

- ArrayList 요소 값 출력

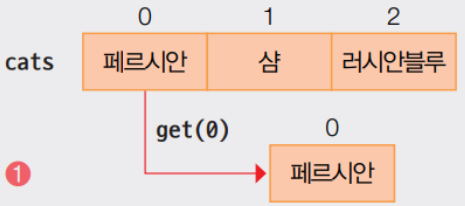
```
ArrayList<String> cats = new ArrayList<String>();

System.out.println(cats.get(0));     // 0번째 인덱스 출력 ●——❶

for (String cat : cats) {            // for문으로 전체 출력
    System.out.println(cat);
}

Iterator iter = cats.iterator();     // Iterator 선언

while (iter.hasNext()) {             // 다음 요소 값이 있는지 확인
    System.out.println(iter.next()); // 요소 값 출력
}                                    ——— Iterator문으로 전체 출력
```



2. List 컬렉션

ArrayList

- ArrayList 요소 값 검색

→contains(value) 메서드: ArrayList에서 원하는 요소 값을 검색, 이때 검색한 값이 있으면 true가 반환됨

→indexOf(value) 메서드: 요소 값이 있는 인덱스를 찾음, 이때 값이 없으면 -1이 반환됨

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

System.out.println(cats.contains("샴"));  // cats에 "샴"이 있으면 true 반환
System.out.println(cats.indexOf("샴"));   // "샴"이 있는 인덱스 반환, 없으면 -1 반환
```

2. List 컬렉션

ArrayList

ArrayList를 이용한 요소 삽입· 삭제· 수정· 검색· 출력 예시

```
import java.util.ArrayList;
public class Example01 {
    public static void main(String[] args) {
        ArrayList<String> cats = new ArrayList<String>();
        cats.add("페르시안");
        cats.add("null");
        System.out.println(cats);
        cats.add(1, "샴");
        System.out.println(cats);
        cats.set(2, "러시안블루");
        System.out.println(cats);
        cats.remove("페르시안");
        System.out.println(cats);
        cats.remove(1);
        System.out.println(cats);
        System.out.println(cats.size());
        System.out.println(cats.get(0));
        System.out.println(cats.contains("샴"));
        System.out.println(cats.indexOf("샴"));
  }
}
```

실행 결과

```
[페르시안, null]
[페르시안, 샴, null]
[페르시안, 샴, 러시안블루]
[샴, 러시안블루]
[샴]
1
샴
true
0
```

2. List 컬렉션

예제 13-1 ArrayList를 이용하여 숫자 정렬하기

```
01 import java.util.ArrayList;
02 import java.util.Comparator;
03
04 public class Collection01 {
05     public static void main(String[] args) {
06         ArrayList<Integer> num = new ArrayList<Integer>();
07
08         for (int i = 10; i >= 1; i--)
09             num.add(i);
10
11         System.out.println(num);
12
13        System.out.print("정렬 전 : ");
14         for (int i = 0; i < num.size(); i++)
15                 System.out.print(num.get(i) + " ");
16
```

2. List 컬렉션

```
17  num.sort(Comparator.naturalOrder());      // Comparator.reveseOrder()
18  //자연 순서를 정의하는 Comparator 객체를 반환
19        System.out.println();
20        System.out.print("정렬 후 : ");
21         for (int i = 0; i < num.size(); i++)
22               System.out.print(num.get(i) + " ");
23     }
24 }
```

실행 결과

```
[10, 9, 8, 7, 6, 5, 4, 3, 2, 1]
정렬 전 : 10 9 8 7 6 5 4 3 2 1
정렬 후 : 1 2 3 4 5 6 7 8 9 10
```

2. List 컬렉션

LinkedList

- List 인터페이스를 구현한 클래스
- 데이터와 포인터를 가진 요소가 선형 순서로 저장됨

  →데이터를 담고 있는 요소들이 연결되어 있고, 요소의 포인터가 다음 요소를 정하는 구조
- 단점: ArrayList에 비해 데이터 삽입이나 삭제가 용이하지만, 인덱스가 없기 때문에 특정 요소에 접근하려면 순차 탐색을 함으로써 검색 속도가 떨어짐

[그림 13-6] LinkedList 클래스의 구조

2. List 컬렉션

LinkedList

- LinkedList의 생성자

[표 13-6] LinkedList의 생성자

생성자	설명
LinkedList()	빈 LinkedList를 생성한다.
LinkedList(Collection C)	Collection C의 요소로 초기화된 LinkedList를 생성한다.

- [LinkedList 클래스를 생성하는 예]

```
LinkedList list = new LinkedList();                          // 클래스 유형 미설정 Object로 선언
LinkedList<Integer> num = new LinkedList<Integer>();         // int형의 객체 요소만 사용 가능
```

new에서 클래스 유형 파라미터 생략 가능
LinkedList<Integer> num = new LinkedList<>();

2. List 컬렉션

LinkedList

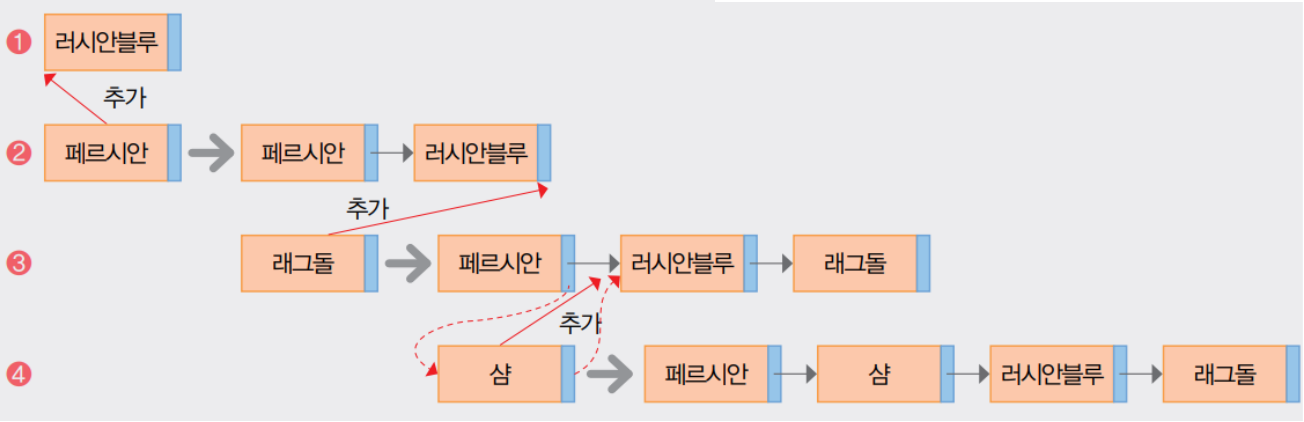
- LinkedList 요소 삽입

→add(index, value) 메서드: LinkedList에 요소 삽입, index를 생략하면 맨 뒤에 데이터가 추가됨

✓addFirst(value): 맨 앞에 데이터 삽입

✓addLast(vale): 맨 뒤에 데이터 삽입

```
LinkedList<String> cats = new LinkedList<String>();
cats.add("러시안블루");      ❶
cats.addFirst("페르시안");   ❷
cats.addLast("래그돌");      ❸
cats.add(1, "샴");           ❹
```

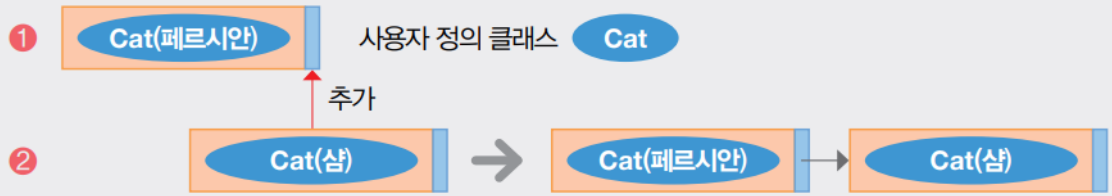


2. List 컬렉션

LinkedList

- LinkedList 요소 삽입

```
LinkedList<Cat> cats = new LinkedList<Cat>();
Cat cat = new Cat("페르시안");
cats.add(cat);                  // cat 객체 생성 후 추가
cats.add(new Cat("샴"));        // cat 객체 생성과 동시에 추가
```



2. List 컬렉션

LinkedList

- LinkedList 요소 삭제

→remove(index, value) 메서드: LinkedList의 요소 삭제

✓특정 인덱스의 값을 삭제할 수 있음

✓index를 생략하면 지정된 요소 값을 삭제함

✓removeFirst(): 맨 앞의 데이터 삭제

✓removeLast(): 맨 뒤의 데이터 삭제

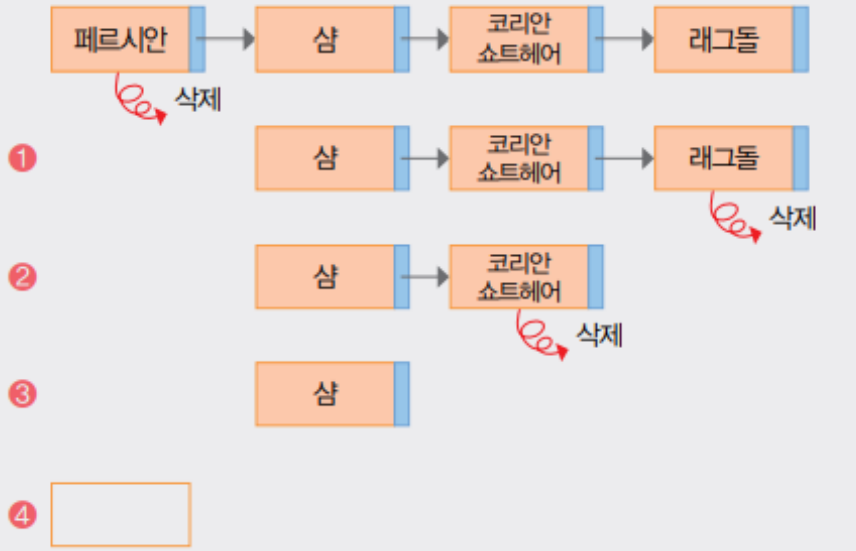
→clear() 메서드: 모든 요소를 삭제

2. List 컬렉션

LinkedList

- LinkedList 요소 삭제

```
LinkedList<String> cats = new LinkedList<String>();
cats.removeFirst();     // 맨 앞의 요소 삭제 ❶
cats.removeLast();      // 맨 뒤의 요소 삭제 ❷
cats.remove(1);         // 인덱스가 1인 요소 삭제 ❸
cats.clear();           // 모든 요소 삭제 ❹
```



2. List 컬렉션

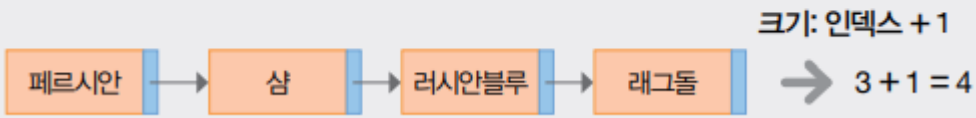
LinkedList

- LinkedList 크기 구하기

→size() 메서드: LinkedList의 크기를 구함

→ArrayList와 마찬가지로 LinkedList의 크기는 마지막 인덱스에 1을 더한 값임

```
LinkedList<String> cats = new LinkedList<String>();
System.out.println(cats.size());     // cats 크기: 4
```



2. List 컬렉션

LinkedList

- LinkedList 요소 값 출력

→get(index) 메서드: LinkedList의 요소 값을 출력,LinkedList에서 원하는 인덱스의 요소 값이 반환됨

→전체 요소 값을 출력할 때는 대개 for문을 사용하지만 Iterator문을 사용할 수도 있음

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

System.out.println(cats.get(0));        // 0번째 인덱스 출력

for (String cat : cats) {               // for문으로 전체 출력
    System.out.println(cat);
}

Iterator iter = cats.iterator();        // Iterator 선언

while (iter.hasNext()) {                // 다음 요소 값이 있는지 확인
    System.out.println(iter.next());    // 요소 값 출력
}
```

2. List 컬렉션

LinkedList

- LinkedList 요소 값 검색

→contains(value) 메서드: LinkedList에서 원하는 요소 값을 검색, 이때 검색 한 값이 있으면 true가 반환됨

→indexOf(value) 메서드: 요소 값이 있는 인덱스를 찾음, 찾는 값이 없으면 -1 반환

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

System.out.println(cats.contains("샴"));  // cats에 "샴"이 있으면 true 반환
System.out.println(cats.indexOf("샴"));   // "샴"이 있는 인덱스 반환, 없으면 -1 반환
```

2. List 컬렉션

LinkedList

LinkedList를 이용한 요소 삽입· 삭제· 수정· 검색· 출력 예시

```
import java.util.LinkedList;
public class Example02 {
    public static void main(String[] args) {
        LinkedList<String> cats = new LinkedList<String>();
        cats.add("러시안블루");
        cats.addFirst("페르시안");
        cats.addLast("래그돌");
        System.out.println(cats);
       cats.add(1, "샴");
        System.out.println(cats);

        cats.set(2, "코리안쇼트헤어");
        System.out.println(cats);

        cats.removeFirst();
        cats.removeLast();
        System.out.println(cats);
        cats.remove(1);
        System.out.println(cats);
        System.out.println(cats.size());
        System.out.println(cats.get(0));
        System.out.println(cats.contains("샴"));
        System.out.println(cats.indexOf("샴"));
    }
}
```

실행 결과

```
[페르시안, 러시안블루, 래그돌]
[페르시안, 샴, 러시안블루, 래그돌]
[페르시안, 샴, 코리안쇼트헤어, 래그돌]
[샴, 코리안쇼트헤어]
[샴]
1
샴
true
0
```

2. List 컬렉션

예제 13-2 LinkedList를 이용하여 숫자 정렬하기

```
01 import java.util.Collections;
02 import java.util.LinkedList;
03
04 public class Collection02 {
05       public static void main(String[] args) {
06           LinkedList<Integer> num = new LinkedList<Integer>();
07
08           for (int i = 10; i >= 1; i--)
09               num.add(i);
10
11           System.out.println(num);
12
13           System.out.print("정렬 전 : ");
14           for (int i = 0; i < num.size(); i++)
15               System.out.print(num.get(i) + " ");
16
```

2. List 컬렉션

```
17        num.sort(Comparator.naturalOrder());
18
19         System.out.println();
20         System.out.print("정렬 후 : ");
21         for (int i = 0; i < num.size(); i++)
22           System.out.print(num.get(i) + " ");
23     }
24 }
```

실행 결과

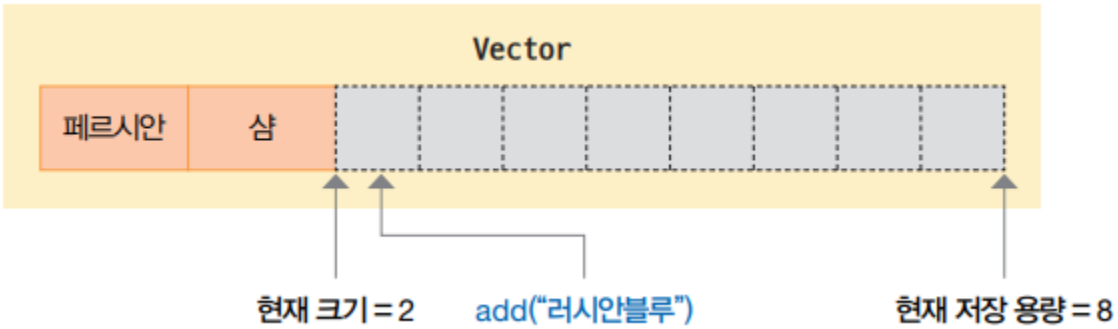
```
[10, 9, 8, 7, 6, 5, 4, 3, 2, 1]
정렬 전 : 10 9 8 7 6 5 4 3 2 1
정렬 후 : 1 2 3 4 5 6 7 8 9 10
```

2. List 컬렉션

Vector

- List 인터페이스를 구현한 클래스
- ArrayList와 마찬가지로 선형 순서로 저장하지만 자동 동기화를 보장하기 때문에 멀티스레드 환경에서 안정적으로 사용할 수 있음
- 스레드가 아닌 환경에서는 Vector를 거의 사용하지 않음

  →따라서 요소 검색, 추가, 삭제, 수정 등의 성능이 저하됨



[그림 13-7] Vector 클래스의 구조

2. List 컬렉션

Vector

- Vector 선언

[표 13-7] Vector 클래스의 생성자

생성자	설명
`Vector()`	빈 Vector를 생성한다.
`Vector(Collection C)`	Collection C의 요소로 초기화된 Vector를 생성한다.
`Vector(int initialCapacity)`	지정된 초기 용량을 가진 Vector를 생성한다.
`Vector(int initialCapacity, int capacityIncrement)`	지정된 초기 용량 및 용량 증분을 가진 빈 Vector를 생성한다.

- [Vector 클래스를 생성하는 예]

```
Vector list = new Vector();                          // 클래스 유형 미설정 Object로 선언
Vector<Integer> num = new Vector<Integer>();         // int형의 객체 요소만 사용 가능
Vector<Integer> num3 = new Vector<Integer>(10);      // 초기 용량을 10으로 지정
```

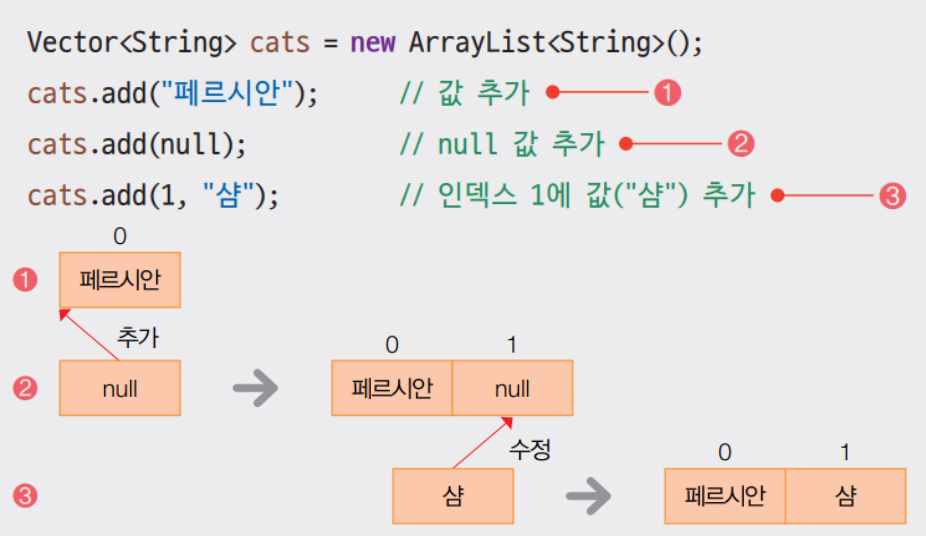
new에서 클래스 유형 파라미터 생략 가능
Vector<Integer> num = new Vector<>();

2. List 컬렉션

Vector

- Vector 요소 삽입

→add() 메서드: Vector에 요소를 삽입, 삽입 순서대로 목록에 요소를 추가함



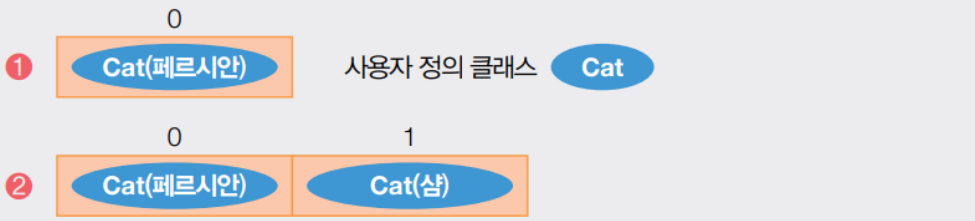
2. List 컬렉션

Vector

- Vector 요소 삽입

→add() 메서드: Vector에 요소를 삽입, 삽입 순서대로 목록에 요소를 추가함

```
Vector<Cat> cats = new Vector<Cat>();
Cat cat = new Cat("페르시안");
cats.add(cat);                  // cat 객체 생성 후 추가
cats.add(new Cat("샴"));        // cat 객체 생성과 동시에 추가
```



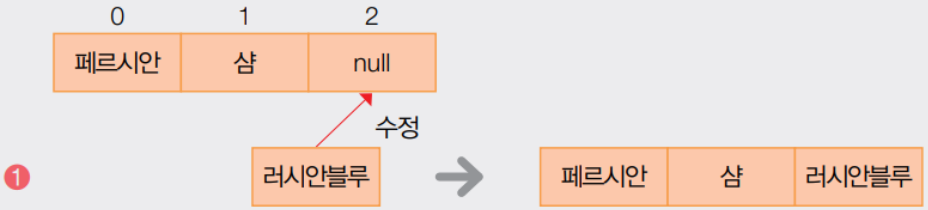
2. List 컬렉션

Vector

- Vector 요소 수정

→set() 메서드: 요소 수정, 인덱스와 새 요소를 제공하면 해당 인덱스의 요소가 새 요소로 업데이트됨

```
Vector<String> cats = new Vectort<String>();
cats.set(2, "러시안블루");    // 인덱스가 2인 요소를 "러시안블루"로 변경 ●——❶
```



2. List 컬렉션

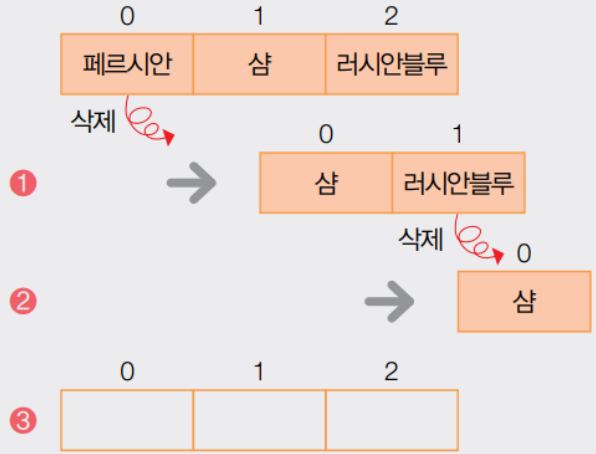
Vector

- Vector 요소 삭제

→remove() 메서드: 요소 삭제, 인덱스 값을 전달하여 요소를 삭제할 수도 있음

→clear() 또는 removeAllElements() 메서드: 모든 요소 삭제

```
Vector<String> cats = new Vectort<String>();
cats.remove("페르시안");      // 요소 삭제 ●——❶
cats.remove(1);              // 인덱스가 1인 요소 삭제 ●——❷
cats.clear();                // 모든 요소 삭제 ┐
cats.removeAllElements();    // 모든 요소 삭제 ┘——❸
```



2. List 컬렉션

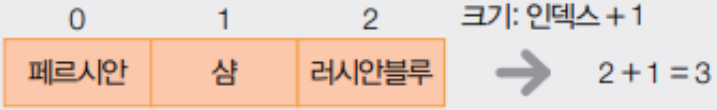
Vector

- Vector 크기 구하기

→size() 메서드: Vector의 크기를 구함

→Vector의 크기: 마지막 인덱스에 1을 더한 값

```
Vector<String> cats = new Vectort<String>();
System.out.println(cats.size());     // cats 크기: 3
```



2. List 컬렉션

Vector

- Vector 요소 값 출력

→get(index) 메서드: 요소 값을 출력하기 위해 사용, Vector에서 원하는 인덱스의 요소 값이 반환됨

→전체 요소 값을 출력할 때 대개 for문을 사용하지만 Iterator문을 사용할 수 있음

```
Vector<String> cats = new Vectort<String>();

System.out.println(cats.get(0));     // 0번째 인덱스 출력

for (String cat : cats) {            // for문으로 전체 출력
    System.out.println(cat);
}

Iterator iter = cats.iterator();     // Iterator 선언

while (iter.hasNext()) {             // 다음 요소 값이 있는지 확인
   System.out.println(iter.next()); // 요소 값 출력
}
```

2. List 컬렉션

Vector

- Vector 요소 값 검색

→contains(value) 메서드: 원하는 요소 값 검색, 찾는 값이 있으면 true 반환

→indexOf(value) 메서드: 요소 값이 있는 인덱스를 찾음, 찾는 값이 없으면 -1이 반환됨

```
Vector<String> cats = new Vectort<String>();

System.out.println(cats.contains("샴"));   // cats에 "샴"이 있으면 true 반환
System.out.println(cats.indexOf("샴"));    // "샴"이 있는 인덱스 반환, 없으면 -1 반환
```

2. List 컬렉션

Vector

BufferedReader 클래스를 이용한 파일 읽기 예시

```
import java.util.Vector;
public class Example03 {
    public static void main(String[] args) {
        Vector<String> cats = new Vector<String>();
        cats.add("페르시안");
        cats.add(null);
        System.out.println(cats);
        cats.add(1, "샴");
        System.out.println(cats);
        cats.set(2, "러시안블루");
        System.out.println(cats);
       cats.remove("페르시안");
        System.out.println(cats);
        cats.remove(1);
        System.out.println(cats);
        System.out.println(cats.size());
        System.out.println(cats.get(0));
        System.out.println(cats.contains("샴"));
        System.out.println(cats.indexOf("샴"));
    }
}
```

실행 결과

```
[페르시안, null]
[페르시안, 샴, null]
[페르시안, 샴, 러시안블루]
[샴, 러시안블루]
[샴]
1
샴
true
0
1
```

2. List 컬렉션

예제 13-3 Vector를 이용하여 숫자 정렬하기

```
01 import java.util.Collections;
02 import java.util.Vector;
03
04 public class Collection03 {
05     public static void main(String[] args) {
06         Vector<Integer> num = new Vector<Integer>();
07
08         for (int i = 10; i >= 1; i--)
09             num.add(i);
10
11         System.out.println(num);
12
13         System.out.print("정렬 전 : ");
14         for (int i = 0; i < num.size(); i++)
15             System.out.print(num.get(i) + " ");
16
```

2. List 컬렉션

```
17      num.sort(Comparator.naturalOrder());
18
19       System.out.println();
20       System.out.print("정렬 후 : ");
21       for (int i = 0; i < num.size(); i++)
22           System.out.print(num.get(i) + " ");
23     }
24 }
```

실행 결과

```
[10, 9, 8, 7, 6, 5, 4, 3, 2, 1]
정렬 전 : 10 9 8 7 6 5 4 3 2 1
정렬 후 : 1 2 3 4 5 6 7 8 9 10
```

Section 03

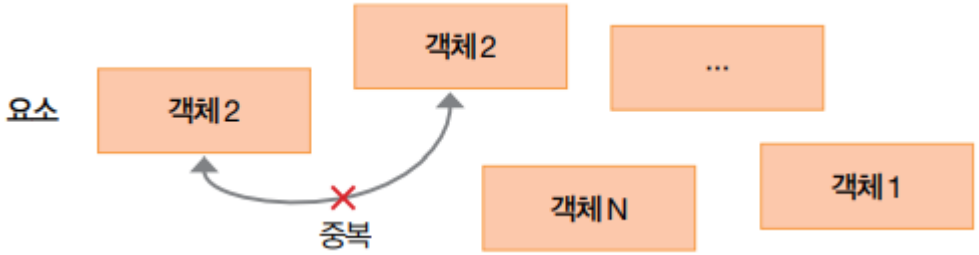
Set 컬렉션

3. Set 컬렉션

Set 컬렉션

- Collection 인터페이스의 자식 인터페이스
- 비선형 자료 구조의 형태로 요소를 저장함
- Set 컬렉션은 요소를 인덱스로 관리하지 않기 때문에 저장 순서가 보장되지 않음
- 순서 자체가 없으므로 요소를 검색하여 가져오는 get() 메서드가 존재하지 않음

  →iterator() 메서드로 Iterator 클래스의 객체를 생성하고 데이터를 가져와야 함
- Set 컬렉션에는 같은 요소를 중복 저장할 수 없고 null도 하나만 저장할 수 있음



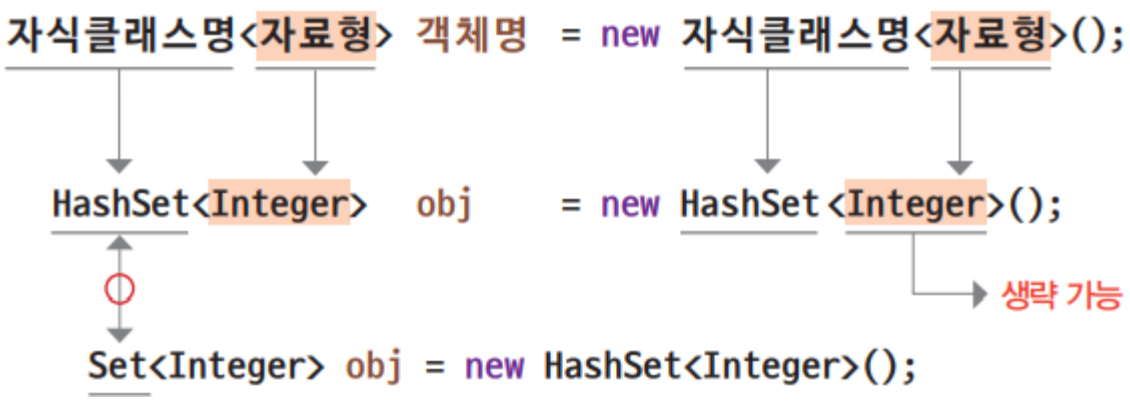
[그림 13-8] Set 컬렉션의 구조

3. Set 컬렉션

Set 컬렉션

- Set 컬렉션을 구현하는 대표적인 클래스: HashSet, TreeSet, LinkeHashSet

```
자식클래스명<자료형> 객체명 = new 자식클래스명<자료형>();
Set<자료형> 객체명 = new 자식클래스명<자료형>();
```



[그림 13-9] Set 컬렉션의 자식 클래스인 HashSet 선언 형식

3. Set 컬렉션

Set 컬렉션

- [Set 컬렉션의 주요 자식 클래스를 선언하는 예]

```
Set<Integer> obj1 = new HashSet<Integer>();
   // HashSet<Integer> obj1 = new HashSet<Integer>();과 같음

Set<Integer> obj2 = new TreeSet<Integere>();
   // TreeSet<Integer> obj2 = new TreeSet<Integer>();과 같음
```

[표 13-8] Set 인터페이스의 주요 메서드

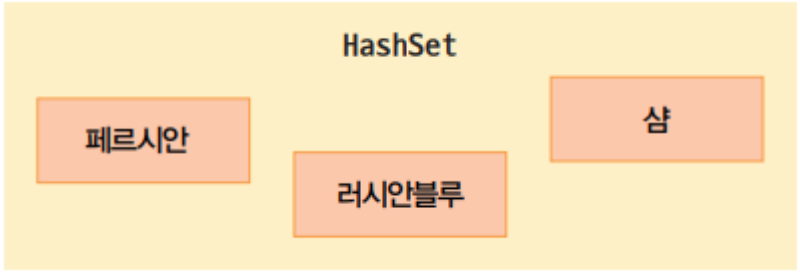
메서드	설명
boolean add(E e)	요소를 삽입한다.
boolean contains(Object o)	지정된 요소를 검색한다.
boolean isEmpty()	컬렉션이 비어 있는지 여부를 확인한다.
int size()	모든 요소의 개수를 반환한다.
void clear()	모든 요소를 삭제한다.
boolean remove(Object o)	지정된 요소를 삭제한다.

3. Set 컬렉션

HashSet

- Set 컬렉션을 구현하는 대표적인 클래스
- 요소를 저장하기 위해 해싱(hashing) 방법을 활용

  →정보를 저장하는 해시 테이블 사용
- 요소 목록은 비선형 순서로 저장 되고 요소를 중복해서 저장할 수 없음

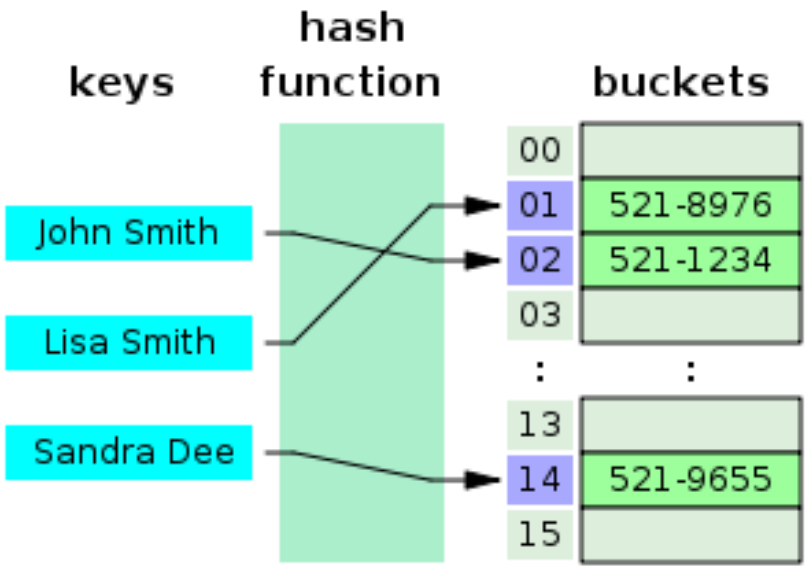


[그림 13-10] HashSet 클래스의 구조

3. Set 컬렉션

Hash Table

- (Key, Value)로 데이터를 저장하는 자료구조 중 하나로 빠르게 데이터를 검색할 수 있는 자료구조
- 내부적으로 배열(버킷)을 사용하여 데이터를 저장
- 해시 테이블은 각각의 Key값에 해시함수를 적용해 배열의 고유한 index를 생성하고, 이 index를 활용해 값을 저장하거나 검색



3. Set 컬렉션

HashSet

- HashSet 클래스의 생성자

[표 13-9] HashSet 클래스의 생성자

생성자	설명
HashSet()	빈 HashSet을 생성한다.
HashSet(Collection C)	Collection C의 요소로 초기화된 HashSet을 생성한다.
HashSet(int initialCapacity)	지정된 초기 용량을 가진 HashSet을 생성한다.
HashSet(int initialCapacity, float loadFactor)	지정된 초기 용량, 로드 계수를 가진 HashSet을 생성한다.

loadFactor : 해시셋의 용량이 어느 정도 차면 새로운 해시 버킷을 추가로 생성할지 결정하는 비율

- [HashSet 클래스를 생성하는 예]

```
HashSet<Integer> set1 = new HashSet<Integer>();          // int형의 객체 요소만 사용 가능
```

new에서 클래스 유형 파라미터 생략 가능
HashSet<Integer> set1 = new HashSet<>();

```
HashSet<Integer> set2 = new HashSet<Integer>(10);        // 초기 용량을 10으로 지정
HashSet<Integer> set2 = new HashSet<Integer>(10, 0.7f);
   // 초기 용량을 10, 로드 계수를 0.7f로 지정
```

3. Set 컬렉션

HashSet

- HashSet 요소 삽입

→add() 메서드: HashSet에 요소 추가

→입력된 값이 HashSet 내부에 존재하지 않으면 그 값을 HashSet에 추가하고 true를 반환하며, 내부에 그 값이 존재하면 false를 반환함

```
HashSet<String> cats = new HashSet<String>();
cats.add("페르시안");      // 값 추가 ❶
cats.add(null);           // null 값 추가 ❷
cats.add("샴");           // 값 추가 ❸
```


```
HashSet<Cat> cats = new HashSet<Cat>();
Cat cat = new Cat("페르시안");
cats.add(cat);              // cat 객체 생성 후 추가 ❶
cats.add(new Cat("샴"));    // cat 객체 생성과 동시에 추가 ❷
```

3. Set 컬렉션

HashSet

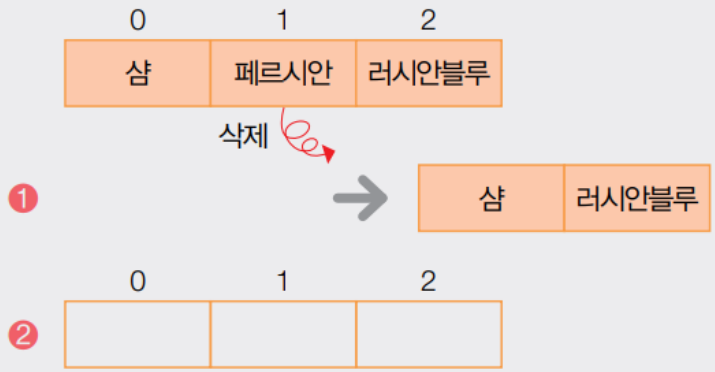
- HashSet 요소 삭제

→remove() 메서드: 요소 삭제

→삭제할 특성 요소 값이 HashSet에 존재하면 그 값을 삭제하고 true를 반환하며, 그 값이 존재하지 않으면 false를 반환함

→clear() 메서드: 모든 요소 삭제

```
HashSet<String> cats = new HashSet<String>();
cats.remove("페르시안");      // 요소 삭제 ●——❶
cats.clear();                 // 모든 요소 삭제 ●——❷
```



3. Set 컬렉션

HashSet

HashSet을 이용한 요소 삽입· 삭제· 수정· 검색· 출력 예시

```
import java.util.HashSet;
public class Example04 {
    public static void main(String[] args) {
        HashSet<String> cats = new HashSet<String>();
        cats.add("페르시안");
        cats.add("샴");
       System.out.println(cats);
        cats.add("러시안블루");
        System.out.println(cats);
        cats.remove("페르시안");
        System.out.println(cats);
        System.out.println(cats.size());
        System.out.println(cats.contains("샴"));
       }
}
```

실행 결과

```
[샴, 페르시안]
[샴, 페르시안, 러시안블루]
[샴, 러시안블루]
2
true
```

3. Set 컬렉션

예제 13-4 HashSet을 배열로 변경하기

```
01 import java.util.HashSet;
02
03 public class Collection04 {
04     public static void main(String[] args) {
05         HashSet<String> str = new HashSet<String>();
06
07         str.add("A");
08         str.add("B");
09         str.add("C");
10         System.out.println(str);
11
12         System.out.print("HashSet 요소 : ");
13         for (String elements : str)
14             System.out.print(elements + " ");
15
```

3. Set 컬렉션

```
16        String[] array = new String[str.size()];
17        str.toArray(array);
18
19       System.out.println();
20       System.out.print("Array 요소 : ");
21       for (int i = 0; i < array.length; i++)
22           System.out.print(array[i] + " ");
23     }
24 }
```

실행 결과

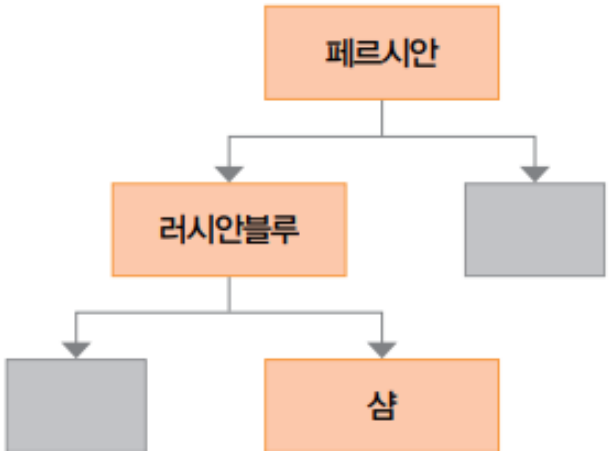
```
[A, B, C]
HashSet 요소 : A B C
Array 요소 : A B C
```

3. Set 컬렉션

TreeSet

- TreeSet은 요소를 중복해서 저장할 수 없고 저장 순서가 유지되지 않는 비선형 트리 기반 데이터 구조의 구현을 제공
- TreeSet은 HashSet과 달리 이진 탐색 트리(binary search tree) 구조로 데이터를 저장함

→데이터를 추가하거나 삭제할 때 HashSet보다 시간이 더 걸리지만, 데이터를 검색하거나 정렬할 때는 효율적임



[그림 13-11] TreeSet 클래스의 구조

3. Set 컬렉션

TreeSet

- TreeSet 선언

→TreeSet 클래스에는 비어 있거나 다른 컬렉션의 요소에서 TreeSet을 만드는 데 사용할 수 있는 생성자가 있음

[표 13-10] TreeSet 클래스의 생성자

생성자	설명
TreeSet()	빈 TreeSet을 생성한다.
TreeSet(Collection C)	Collection C의 요소로 초기화된 TreeSet을 생성한다.
TreeSet(Comparator comparator)	지정된 매개변수에 따라 정렬된 TreeSet을 생성한다.
TreeSet(SortedSet s)	동일한 요소를 포함하고 동일한 순서를 사용하는 지정된 매개변수(정렬된 세트)로 TreeSet을 생성한다.

→[TreeSet 클래스를 생성하는 예]

```
TreeSet<Integer> set1 = new TreeSet<Integer>();      // int형의 객체 요소만 사용 가능
```

new에서 클래스 유형 파라미터 생략 가능
TreeSet<Integer> set1 = new TreeSet<>();

```
TreeSet<Integer> set2 = new TreeSet<Integer>(set1);     // set1의 모든 값을 TreeSet set2로 생성
TreeSet<Integer> set2 = new TreeSet<Integer>(10, 0.7f);
   // 초기 용량을 10, 로드 계수를 0.7f로 지정
```

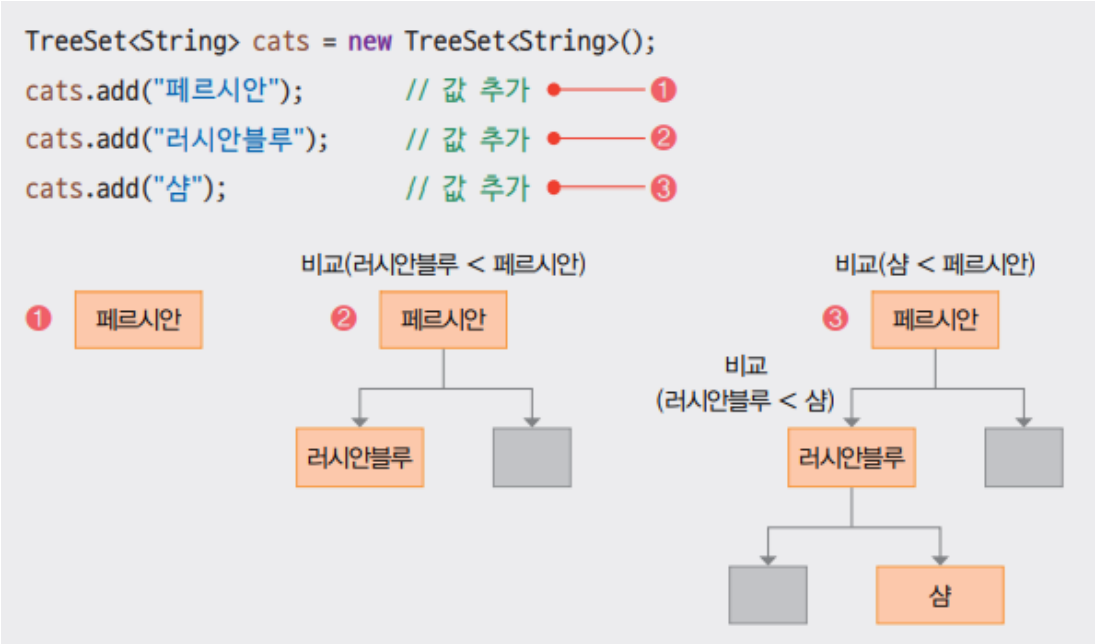
3. Set 컬렉션

TreeSet

- TreeSet 요소 삽입

→add() 메서드: TreeSet에 요소 삽입

→입력된 값이 TreeSet 내부에 존재하지 않으면 그 값을 TreeSet에 추가하고 true를 반환하며, 내부에 그 값이 존재하면 false를 반환함



3. Set 컬렉션

TreeSet

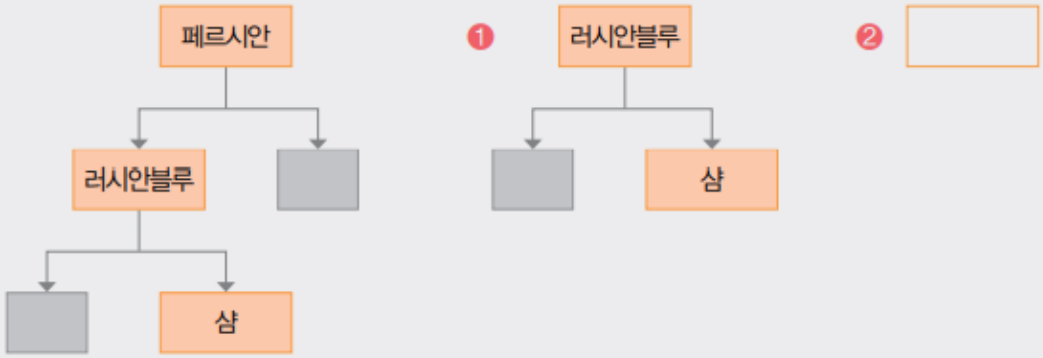
- TreeSet 요소 삭제

→remove() 메서드: 요소 삭제

→삭제할 특성 요소 값이 TreeSet에 존재하면 그 값을 삭제하고 true를 반환하며, 삭제하려는 값이 존재하지 않으면 false를 반환함

→clear() 메서:드모든 요소 삭제

```
TreeSet<String> cats = new TreeSet<String>();
cats.remove("페르시안");      // 요소 삭제 ●——❶
cats.clear();                 // 모든 요소 삭제 ●——❷
```



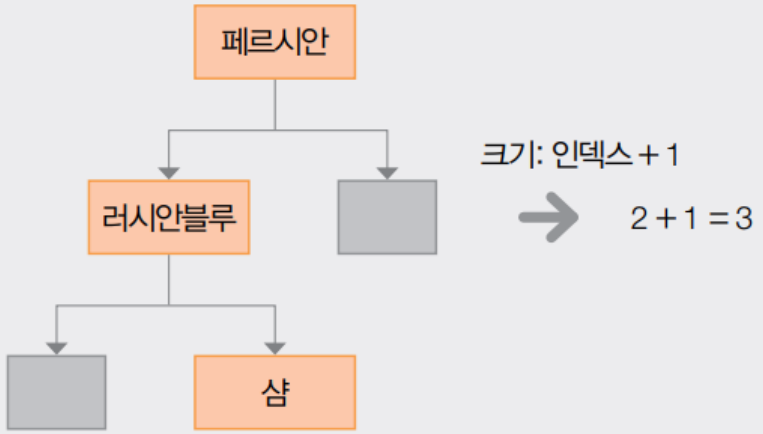
3. Set 컬렉션

TreeSet

- TreeSet 크기 구하기

→size() 메서드: TreeSet의 크기를 구함

```
TreeSet<String> cats = new TreeSet<String>();
System.out.println(cats.size());     // cats 크기: 3
```



3. Set 컬렉션

TreeSet

- TreeSet 요소 값 출력

→first() 메서드: TreeSet의 첫 번째 요소 값 출력 (왼쪽 마지막 노드 :최소값)

→Last() 메서드: 마지막 요소 값 출력(오른쪽 마지막 노드:최대값)

→전체 요소 값을 출력할 때는 대개 for문을 사용하지만 Iterator문을 사용할 수도 있음

```
TreeSet<String> cats = new TreeSet<String>();

System.out.println(cats);              // 전체 출력
System.out.println(cats.first());      // 첫 번째 요소 값 출력
System.out.println(cats.last());       // 마지막 요소 값 출력

for (String cat : cats) {              // for문으로 전체 출력
   System.out.println(cat);
}

Iterator iter = cats.iterator();       // Iterator 선언

while (iter.hasNext()) {               // 다음 요소 값이 있는지 확인
   System.out.println(iter.next());    // 요소 값 출력
}
```

3. Set 컬렉션

TreeSet

- TreeSet 요소 값 검색

→contains(value) 메서드: 원하는 요소 값 검색, 검색한 값이 있으면 true 반환

```
TreeSet<String> cats = new TreeSet<String>();
System.out.println(cats.contains("샴"));  // cats에 "샴"이 있으면 true 반환
```

3. Set 컬렉션

TreeSet

TreeSet을 이용한 요소 삽입· 삭제· 수정· 검색· 출력 예시

```
import java.util.TreeSet;
public class Example05 {
   public static void main(String[] args) {
        TreeSet<String> cats = new TreeSet<String>();
        cats.add("페르시안");
        cats.add("□□□□□");
        System.out.println(cats);
        cats.add("□");
        System.out.println(cats);
        cats.remove("페르시안");
        System.out.println(cats);
        System.out.println(cats.size());
        System.out.println(cats.contains("샴"));
        System.out.println(cats.first());
        System.out.println(cats.last());
   }
}
```

실행 결과

```
[러시안블루, 페르시안]
[러시안블루, 샴, 페르시안]
[러시안블루, 샴]
2
true
러시안블루
샴
```

3. Set 컬렉션

예제 13-5 TreeSet을 배열로 변경하기

```
01 import java.util.TreeSet;
02
03 public class Collection05 {
04     public static void main(String[] args) {
05         TreeSet<String> str = new TreeSet<String>();
06
07         str.add("A");
08         str.add("B");
09         str.add("C");
10         System.out.println(str);
11
12         System.out.print("TreeSet 요소 : ");
13         for (String elements : str)
14             System.out.print(elements + " ");
15
```

3. Set 컬렉션

```
16      String[] array = new String[str.size()];
17      str.toArray(array);
18
19       System.out.println();
20      System.out.print("Array 요소 : ");
21      for (int i = 0; i < array.length; i++)
22         System.out.print(array[i] + " ");
23     }
24 }
```

실행 결과

```
[A, B, C]
TreeSet 요소 : A B C
Array 요소 : A B C
```

Section 04
Map 컬렉션

4. Map 컬렉션

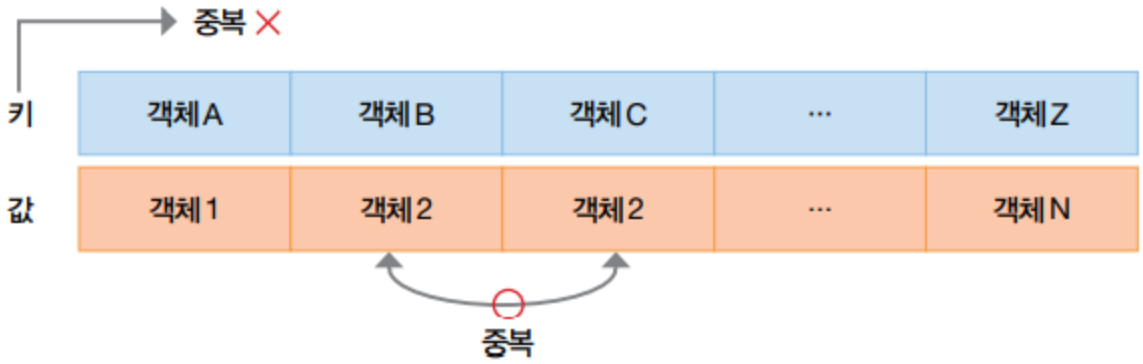
Map 컬렉션

- 키(key)와 값(value)으로 구성된 자료 구조의 형태로 데이터를 저장

→키는 중복해서 저장할 수 없지만 값은 중복해서 저장할 수 있음

→ 만약 기존에 있던 키와 같은 키가 입력되면 해당 키의 값이 새로운 값으로 변경

→Map 컬렉션은 키로 데이터를 관리함

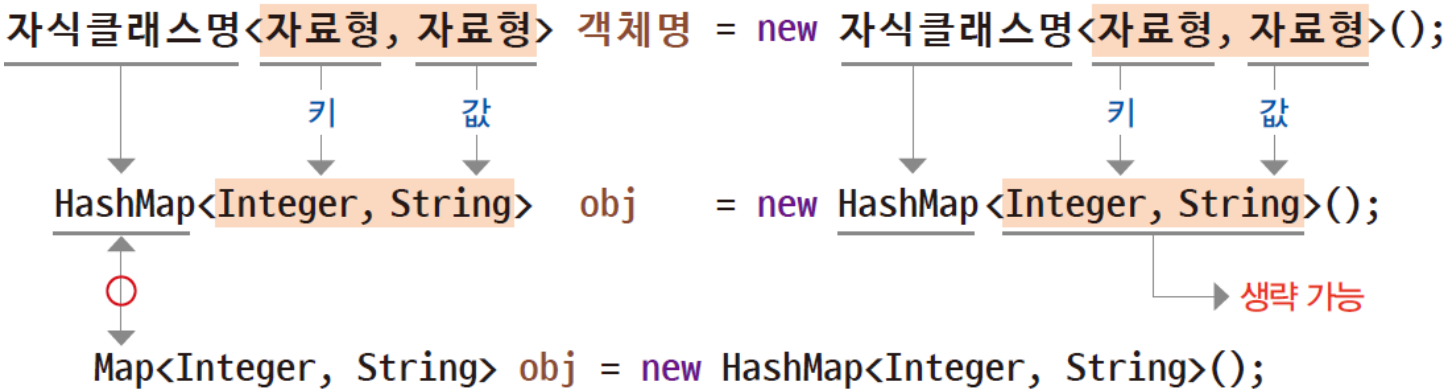


[그림 13-12] Map 컬렉션의 구조

Map 컬렉션

- Map 컬렉션을 구현하는 대표적인 클래스: HashMap, HashTable, LinkedHashMap, TreeMap

```
자식클래스명<자료형, 자료형> 객체명 = new 자식클래스명<자료형, 자료형>();
Map<자료형, 자료형> 객체명 = new 자식클래스명<자료형, 자료형>();
```



[그림 13-13] Map 컬렉션의 자식 클래스인 HashMap 선언 형식

4. Map 컬렉션

Map 컬렉션

- [Map 컬렉션의 주요 자식 클래스를 선언하는 예]

```
Map<Integer, String> obj1 = new HashMap();
   // HashMap<Integer, String> obj1 = new HashMap<Integer, String>();과 같음

Map<Integer, String> obj2 = new TreeMap();
   // TreeMap<Integer, String> obj2 = new TreeMap<Integer, String>();과 같음
```

4. Map 컬렉션

Map 컬렉션

[표 13-11] Map 인터페이스의 메서드

메서드	설명
V put(K Key, V value)	지정된 키와 값을 추가하여 저장되면 값을 반환한다.
boolean containsKey(Object Key)	지정된 키가 있는지 확인한다.
boolean containsValue(Object value)	지정된 값이 있는지 확인한다.
Set<Map.Entry<K,V>> entrySet()	모든 Map.Entry 객체를 Set에 담아 반환한다.
Set<K> keySet()	모든 키를 Set 객체에 담아 반환한다.
V get(Object key)	지정된 키에 있는 값을 반환한다.
boolean isEmpty()	컬렉션이 비어 있는지 여부를 확인한다.
int Size()	저장되어 있는 모든 객체의 개수를 반환한다.
Collection<V> values()	저장된 모든 값을 Collection에 담아 반환한다.
void clear()	저장된 모든 Map.Entry를 삭제한다.
V remove(Object Key)	지정된 키와 일치하는 Map.Entry를 삭제하고 값을 반환한다.

4. Map 컬렉션

HashMap

- Map 컬렉션의 대표적인 구현 클래스
- 저장 순서를 보장하지 않음
- 키와 값을 쌍으로 저장하기 위한 Map 기반 컬렉션이며 HashMap<Key, Value> 또는 HashMap<K, V>로 표시함
- 해싱을 사용하기 때문에 많은 양의 데이터를 검색 하는 데 유용함

4. Map 컬렉션

HashMap

- HashMap 선언

[표 13-12] HashMap 클래스의 생성자

생성자	설명
`HashMap()`	빈 HashMap을 생성한다.
`HashMap(int initialCapacity)`	지정된 초기 용량을 가진 HashMap을 생성한다.
`HashMap(int initialCapacity, float loadFactor)`	지정된 초기 용량, 로드 계수를 가진 HashMap을 생성한다.
`HashMap(Map m)`	지정된 Map과 같은 매핑으로 새 HashMap을 생성한다.

- [HashMap 클래스를 생성하는 예]

```
HashMap<String, String> map1 = new HashMap<String, String>();    // HashMap 생성
```

new에서 클래스 유형 파라미터 생략 가능
HashMap⟨String, String⟩ map1 = new HashMap⟨⟩();

```
HashMap<String, String> map3 = new HashMap<>(map1);   // map1의 모든 값을 가진 HashMap 생성
HashMap<String, String> map4 = new HashMap<>(10);     // 초기 용량을 10으로 지정
HashMap<String, String> map5 = new HashMap<>(10, 0.7f);
   // 초기 용량을 10, 로드 계수를 0.7f로 지정
```

4. Map 컬렉션

HashMap

- HashMap 요소 삽입

→put(key, value) 메서드: HashMap에 요소 삽입

→입력된 키가 HashMap 내부에 존재하면 새로운 키로 변경

4. Map 컬렉션

HashMap

- HashMap 요소 수정

→replace(key, value) 메서드: HashMap의 요소 수정

→입력된 키가 HashMap 내부에 존재하면 새로운 키로 변경

```
Map<String, Integer> map = new HashMap<>();
map.put("apple", 1);

// 기존 값을 새 값으로 대체
map.replace("apple", 10);
System.out.println(map.get("apple")); // 출력: 10

// 존재하지 않는 키에 대해 replace를 호출하면 아무 일도 일어나지 않음
map.replace("banana", 20);
System.out.println(map.get("banana")); // 출력: null
```

4. Map 컬렉션

HashMap

- HashMap 요소 삭제

  →remove(key) 메서드: 요소 삭제

  ✓즉, 키로만 Map의 요소를 삭제할 수 있음

  →clear() 메서드: 모든 요소 삭제

- HashMap 크기 구하기

  →size() 메서드: HashMap의 크기를 구함

4. Map 컬렉션

HashMap

- HashMap 요소 값 출력

→get(key) 메서드: 요소 값 출력, HashMap에서 원하는 키의 요소 값이 반환됨

→entrySet() 또는 KeySet() 메서드: 전체 요소 값 출력

```
HashMap<Integer, String> cats = new HashMap<Integer, String>();

System.out.println(cats);           // 전체 출력
System.out.println(cats.get(2));    // 키 2의 값 출력

// entrySet() 활용
for (Entry<Integer, String> entry : cats.entrySet()) {
   System.out.print(" (" + entry.getKey() + ", " + entry.getValue() + ") ");
}

Iterator<Entry<Integer, String>> entries = cats.entrySet().iterator();
while (entries.hasNext()) {
   Map.Entry<Integer, String> entry = entries.next();
   System.out.print(" (" + entry.getKey() + ", " + entry.getValue() + ") ");
}
```

```
// KeySet() 활용
for (Integer i : cats.keySet()) {
   System.out.print(" (" + i + ", " + cats.get(i) + ") ");
}

Iterator<Integer> keys = cats.keySet().iterator();      // Iterator 선언
while (keys.hasNext()) {    // 다음 요소 값이 있는지 확인
   int key = keys.next();    // 키 얻어오기
   System.out.print(" (" + key + ", " + cats.get(key) + ") ");      // 요소 값 출력
}
```

4. Map 컬렉션

HashMap

HashMap을 이용한 요소 삽입· 삭제· 수정· 검색· 출력 예시

```
import java.util.HashMap;
import java.util.Iterator;
public class Example06 {
    public static void main(String[] args) {
        HashMap<Integer, String> cats = new HashMap<Integer,
       String>();
        cats.put(1, "페르시안");
        cats.put(2, "샴");
        System.out.println(cats);
        cats.put(3, "러시안블루");
        System.out.println(cats);
        cats.put(3, "래그돌");
        System.out.println(cats);
        cats.remove(1);
        System.out.println(cats);
        System.out.println(cats.size());
        System.out.println(cats.get(2));
        System.out.print("(Key, Value) =");
        Iterator<Integer> keys = cats.keySet().iterator();
        while (keys.hasNext()) {
            int key = keys.next();
            System.out.print(" (" + key + ", " + cats.get(key) + ") ");
        }
    }
}
```

실행 결과

```
{1=페르시안, 2=샴}
{1=페르시안, 2=샴, 3=러시안블루}
{1=페르시안, 2=샴, 3=래그돌}
{2=샴, 3=래그돌}
2
샴
(Key, Value) = (2, 샴)  (3, 래그돌)
```

4. Map 컬렉션

예제 13-6 HashMap의 키와 값 출력하기

```
01 import java.util.HashMap;
02 import java.util.Map.Entry;
03
04 public class Collection06 {
05     public static void main(String[] args) {
06         HashMap<Integer, String> str = new HashMap<Integer, String>();
07
08         str.put(1, "A");
09         str.put(2, "B");
10         str.put(3, "C");
11
12         System.out.println(str);
13
14        System.out.print("Keys : ");
15        for (Integer key : str.keySet()) {
16             System.out.print(key);
17             System.out.print(", ");
18      }
```

4. Map 컬렉션

```
19
20        System.out.println();
21        System.out.print("Values : ");
22        for (String value : str.values()) {
23            System.out.print(value);
24            System.out.print(", ");
25        }
26
27        System.out.println();
28        System.out.print("Keys=Values : ");
29        for (Entry<Integer, String> entry : str.entrySet()) {
30                System.out.print(entry);
31                System.out.print(", ");
32        }
33    }
34 }
```

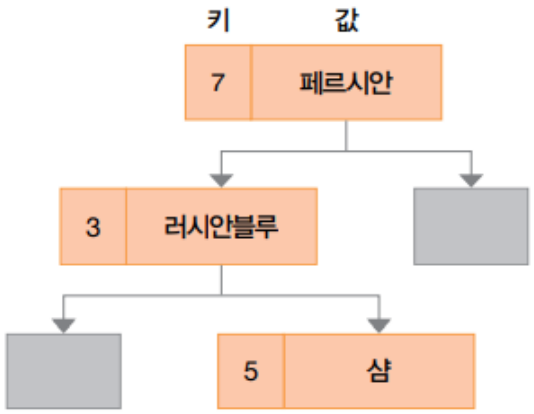
실행 결과

```
{1=A, 2=B, 3=C}
Keys : 1, 2, 3,
Values : A, B, C,
Keys=Values : 1=A, 2=B, 3=C,
```

4. Map 컬렉션

TreeMap

- TreeMap은 키와 값을 쌍으로 저장하기 위한 Map 기반 컬렉션
- 요소를 중복해서 저장할 수 없고 저장 순서가 유지되지 않는 비선형 트리 기반 데이터 구조의 구현을 제공
- 일반적으로 TreeMap은 Map으로서의 성능이 HashMap보다 떨어짐
- TreeMap은 데이터를 저장할 때 즉시 정렬하기 때문에 데이터를 추가하거나 삭제하는 데 HashMap보다 시간이 더 걸림
- 하지만 데이터가 정렬 된 상태로 Map을 유지해야 하거나 정렬된 데이터를 검색할 때는 효율적임



[그림 13-15] TreeMap 클래스의 구조

4. Map 컬렉션

TreeMap

- TreeMap 선언

→TreeMap 클래스에는 비어 있거나 다른 컬렉션의 요소에서 TreeMap을 만드는데 사용할 수 있는 생성자가 있음

[표 13-13] TreeMap 클래스의 생성자

생성자	설명
`TreeMap()`	빈 TreeMap을 생성한다.
`TreeMap(Comparator comparator)`	comparator에 따라 정렬된 TreeMap을 생성한다.
`TreeMap(Map m)`	지정된 Map 요소로 초기화된 TreeMap을 생성한다.
`TreeMap(SortedMap m)`	정렬된 Map 요소로 초기화된 TreeMap을 생성한다.

- [TreeMap 클래스를 생성하는 예]

```
TreeMap<Integer, String> map1 = new TreeMap<Integer, String>();
   // int형의 객체 요소와 문자열만 사용 가능

TreeMap<Integer, String> map2 = new TreeMap<Integer, String>(map1);
   // map1의 모든 값을 TreeMap map2로 생성
```

new에서 클래스 유형 파라미터 생략 가능
TreeMap<Integer, String> set1 = new TreeSMap<>();

4. Map 컬렉션

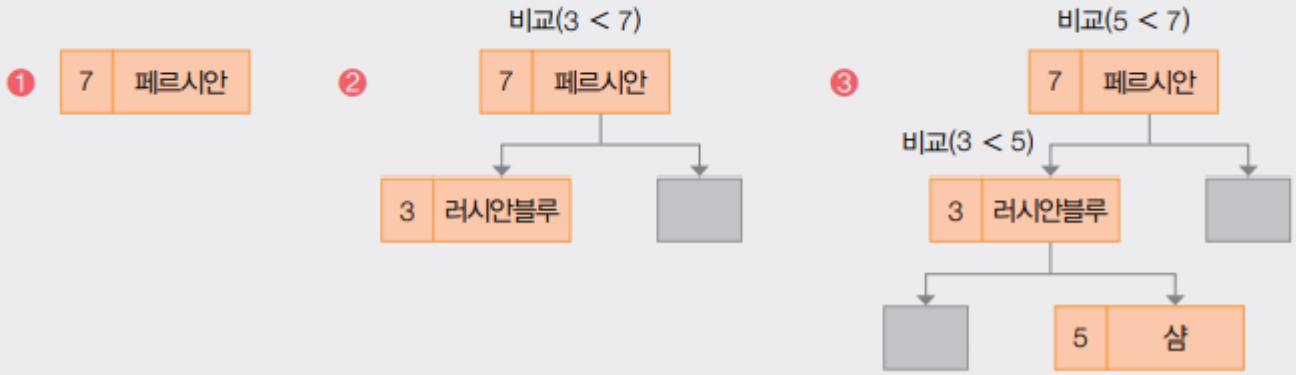
TreeMap

- TreeMap 요소 삽입

→put(key, value) 메서드: TreeMap에 요소 삽입

→입력된 키가 TreeMap 내부에 존재하지 않으면 그 키를 TreeMap에 추가하고, 내부에 그 키가 존재하면 새로운 키로 변경함

```
TreeMap<Integer, String> cats = new TreeMap<Integer, String>();
cats.put(7, "페르시안");      // 키 1에 값 추가 ●——❶
cats.put(3, "러시안블루");    // 키 2에 값 추가 ●——❷
cats.put(5, "샴");            // 키 3에 값 추가 ●——❸
```



4. Map 컬렉션

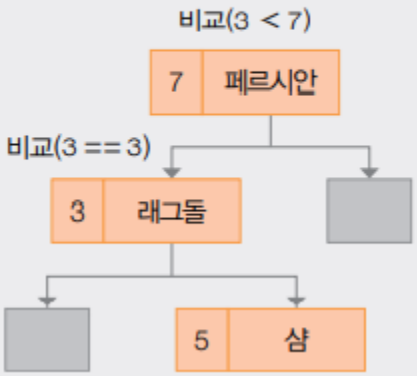
TreeMap

- TreeMap 요소 수정

→replace(key, value) 메서드: 요소 수정

→입력된 키가 TreeMap 내부에 존재하면 새로운 키로 변경

```
TreeMap<Integer, String> cats = new TreeMap<Integer, String>();
cats.replace(3, "러시안블루");      // 키가 3인 요소를 "러시안블루"로 변경
```



4. Map 컬렉션

TreeMap

- TreeMap 요소 삭제

→remove(key) 메서드: 요소 삭제

✓키로만 Map의 요소를 삭제할 수 있음

→clear() 메서드: 모든 요소 삭제

```
TreeMap<Integer, String> cats = new TreeMap<Integer, String>();
cats.remove(5);     // 키 5의 값 삭제 ❶
cats.clear();       // 모든 요소 삭제 ❷
```

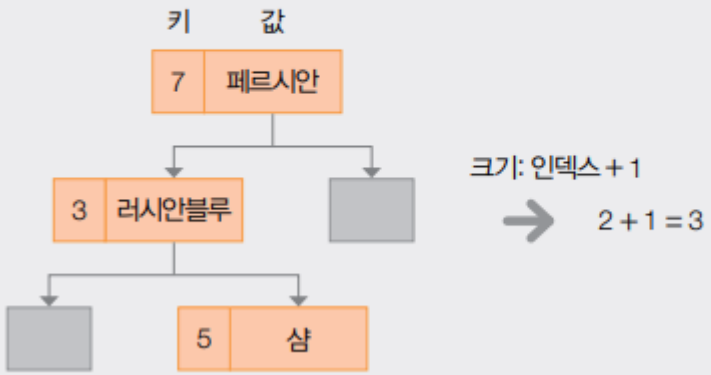
4. Map 컬렉션

TreeMap

- TreeMap 크기 구하기

→size() 메서드: TreeMap의 크기를 구함

```
TreeMap<Integer, String> cats = new TreeMap<Integer, String>();
System.out.println(cats.size());     // cats 크기: 3
```



4. Map 컬렉션

TreeMap

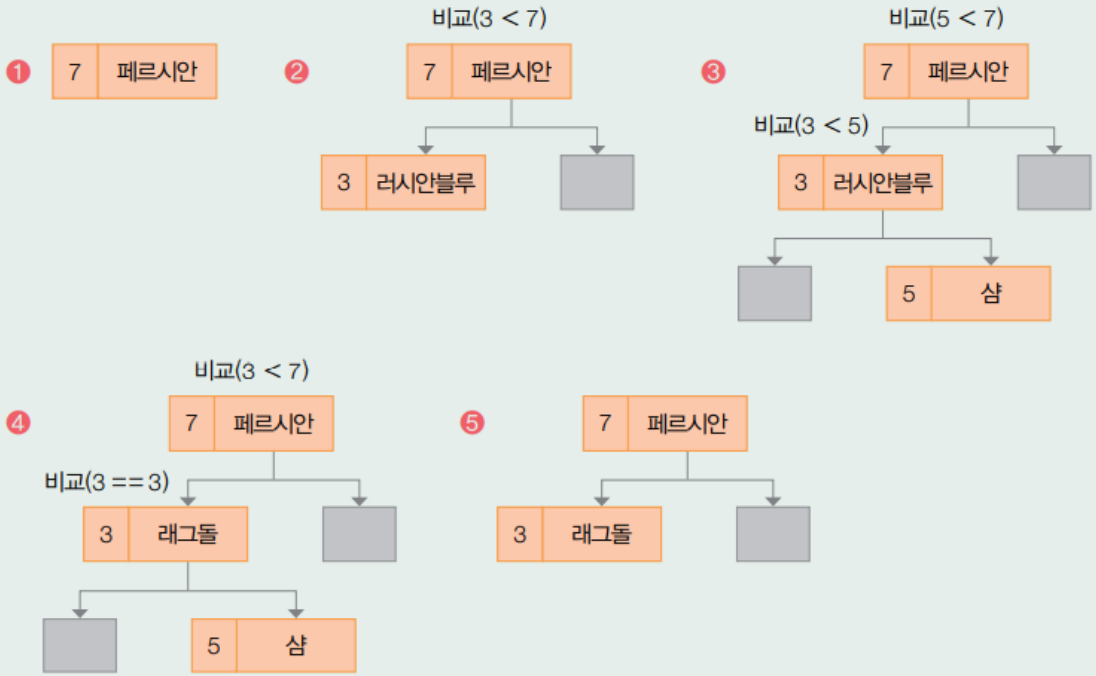
TreeMap을 이용한 요소 삽입· 삭제· 수정· 검색· 출력 예시

```
import java.util.TreeMap;
import java.util.Iterator;
public class Example07 {
    public static void main(String[] args) {
        TreeMap<Integer, String> cats = new TreeMap<Integer, String>();
        cats.put(7, "페르시안");
        cats.put(3, "러시안블루");
        System.out.println(cats);
        cats.put(5, "샴");
        System.out.println(cats);
       cats.replace(3, "래그돌");
        System.out.println(cats);
        cats.remove(5);
        System.out.println(cats);
        System.out.println(cats.size());
        System.out.println(cats.get(7));
        System.out.print("(Key, Value) =");
        Iterator<Integer> keys = cats.keySet().iterator();
        while (keys.hasNext()) {
           int key = keys.next();
            System.out.print(" (" + key + ", " + cats.get(key) + ") ");
        }
  }
}
```

4. Map 컬렉션

실행 결과

```
{3=러시안블루, 7=페르시안}
{3=러시안블루, 5=샴, 7=페르시안}
{3=래그돌, 5=샴, 7=페르시안}
{3=래그돌, 7=페르시안}
2
페르시안
(Key, Value) = (3, 래그돌)  (7, 페르시안)
```



4. Map 컬렉션

예제 13-7 TreeMap의 키와 값 출력하기

```
01 import java.util.TreeMap;
02 import java.util.Map.Entry;
03
04 public class Collection07 {
05     public static void main(String[] args) {
06         TreeMap<Integer, String> str = new TreeMap<Integer, String>();
07
08         str.put(1, "A");
09         str.put(2, "B");
10         str.put(3, "C");
11
12         System.out.println(str);
13
14         System.out.print("Keys : ");
15         for (Integer key : str.keySet()) {
16             System.out.print(key);
17             System.out.print(", ");
18         }
```

4. Map 컬렉션

```
19
20      System.out.println();
21      System.out.print("Values : ");
22      for (String value : str.values()) {
23          System.out.print(value);
24          System.out.print(", ");
25      }
26
27      System.out.println();
28      System.out.print("Keys=Values : ");
29      for (Entry<Integer, String> entry : str.entrySet()) {
30        System.out.print(entry);
31        System.out.print(", ");
32      }
33   }
34 }
```

실행 결과

```
{1=A, 2=B, 3=C}
Keys : 1, 2, 3,
Values : A, B, C,
Keys=Values : 1=A, 2=B, 3=C,
```