Java의 정석

Chapter 11

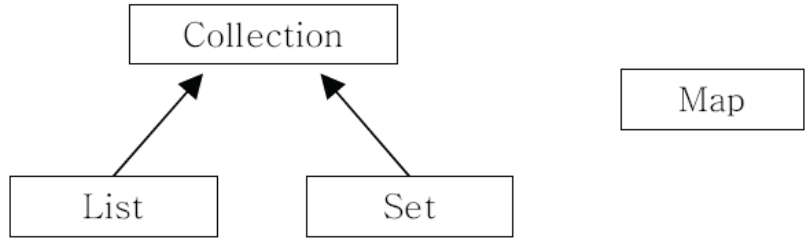
1.컬렉션 프레임워크

컬렉션 프레임워크

컬렉션 프레임워크란?

- 데이터 군(群)을 저장하는 클래스들을 표준화한 설계
- 다수의 데이터를 쉽게 처리할 수 있는 방법을 제공하는 클래스들로 구성
- 여기서 컬렉션이란 다수의 데이터, 즉 데이터 그룹을 의미
- 여기서 프레임웍이란 표준화, 정형화된 체계적인 프로그래밍 방식
- 컬렉션 클래스란 다수의 데이터를 저장할 수 있는 클래스(예, Vector, ArrayList, HashSet)

컬렉션 프레임웍의 핵심 인터페이스



[그림11-1] 컬렉션 프레임웍의 핵심 인터페이스간의 상속계층도

인터페이스	특 징
List	순서가 있는 데이터의 집합. 데이터의 중복을 허용한다. 예) 대기자 명단
	구현클래스 : ArrayList, LinkedList, Stack, Vector 등
Set	순서를 유지하지 않는 데이터의 집합. 데이터의 중복을 허용하지 않는다. 예) 양의 정수집합, 소수의 집합
	구현클래스 : HashSet, TreeSet 등
Map	키(key)와 값(value)의 쌍(pair)으로 이루어진 데이터의 집합 순서는 유지되지 않으며, 키는 중복을 허용하지 않고, 값은 중복을 허용한다. 예) 우편번호, 지역번호(전화번호)
	구현클래스 : HashMap, TreeMap, Hashtable, Properties 등

[표11-1] 컬렉션 프레임웍의 핵심 인터페이스와 특징

ArrayList

ArrayList 란?

- 컬렉션 프레임웍에서 가장 많이 사용되는 컬렉션 클래스
- List인터페이스를 구현하기 때문에 데이터의 저장순서가 유지되고 중복을 허용
- 배열에 더 이상 저장할 공간이 없으면 보다 큰 새로운 배열을 생성해서 기존의 배열에 저장된 내용을 새로운 배열로 복사한 다음에 저장됨

ArrayList

ArrayList 사용 예시

```
ArrayList list1 = new ArrayList(10);
list1.add(new Integer(5));
list1.add(new Integer(4));
list1.add(new Integer(2));
list1.add(new Integer(0));
list1.add(new Integer(1));
list1.add(new Integer(3));

ArrayList list2 = new ArrayList(list1.subList(1,4));
print(list1, list2);

Collections.sort(list1);    // list1과 list2를 정렬한다.
Collections.sort(list2);    // Collections.sort(List l)
print(list1, list2);

System.out.println("list1.containsAll(list2):"
                           + list1.containsAll(list2));

list2.add("B");
list2.add("C");
list2.add(3, "A");
print(list1, list2);

list2.set(3, "AA");
print(list1, list2);

// list1에서 list2와 겹치는 부분만 남기고 나머지는 삭제한다.
System.out.println("list1.retainAll(list2):"
                           + list1.retainAll(list2));
print(list1, list2);

//  list2에서 list1에 포함된 객체들을 삭제한다.
for(int i= list2.size()-1; i >= 0; i--) {
```

```
---------- java ----------
list1:[5, 4, 2, 0, 1, 3]
list2:[4, 2, 0]

list1:[0, 1, 2, 3, 4, 5]
list2:[0, 2, 4]

list1.containsAll(list2):true
list1:[0, 1, 2, 3, 4, 5]
list2:[0, 2, 4, A, B, C]

list1:[0, 1, 2, 3, 4, 5]
list2:[0, 2, 4, AA, B, C]

list1.retainAll(list2):true
list1:[0, 2, 4]
list2:[0, 2, 4, AA, B, C]

list1:[0, 2, 4]
list2:[AA, B, C]
```

ArrayList

ArrayList 장점

- 구조가 간단하고 데이터를 읽어오는 데 걸리는 시간(접근시간, access time)이 가장 빠르다
- 인덱스 값을 기준으로 어디든 한 번에 참조가 가능하다

ArrayList

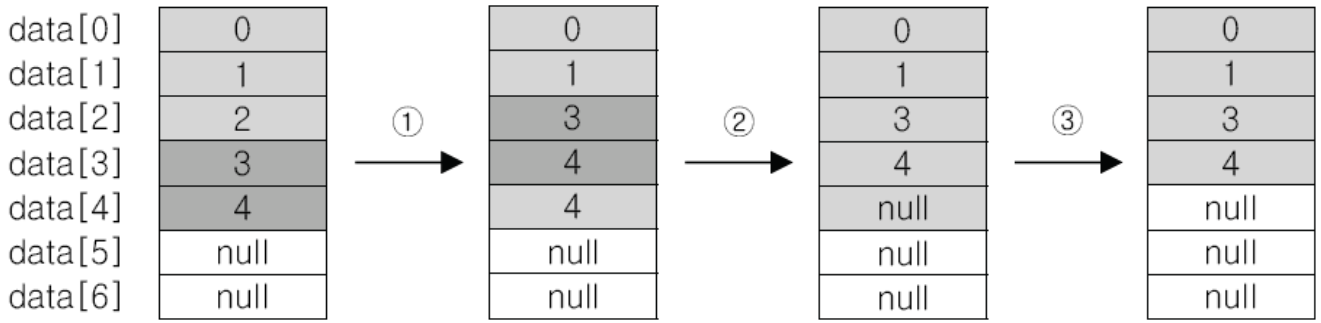
ArrayList 단점

- ArrayList나 Vector 같이 배열을 이용한 자료구조는 데이터를 읽어오고 저장하는데는 효율이 좋지만, 용량을 변경해야 할 때는 새로운 배열을 생성한 후 기존의 배열로부터 새로 생성된 배열로 데이터를 복사해야하기 때문에 효율이 떨어진다.
- 그러므로 처음에 인스턴스를 생성할 때 저장할 데이터의 개수를 잘 고려하여 충분한 용량의 인스턴스를 생성해야한다.

Vector

Vector에 저장된 객체 삭제과정

- Vector에 저장된 세 번째 데이터(data[2])를 삭제하는 과정. v.remove(2);를 호출



① 삭제할 데이터 아래의 데이터를 한 칸씩 위로 복사해서 삭제할 데이터를 덮어쓴다.

```
System.arraycopy(data, 3, data, 2, 2)
```

data[3]에서 data[2]로 2개의 데이터를 복사하라는 의미이다.

② 데이터가 모두 한 칸씩 이동했으므로 마지막 데이터는 null로 변경한다.

```
data[size-1] = null;
```

③ 데이터가 삭제되어 데이터의 개수가 줄었으므로 size의 값을 감소시킨다.

```
size--;
```

※ 삭제할 데이터가 마지막 데이터인 경우, ①의 과정은 필요없다.

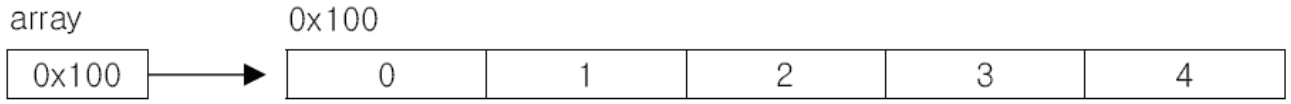
배열 -> LinkedList

배열의 단점

- 크기를 변경할 수 없다. => 새로운 배열 생성하여 데이터 복사해야한다. => 메모리 낭비
- 비순차적인 데이터의 추가 또는 삭제에 시간이 많이 걸린다.
- 차례대로 데이터를 추가하고 마지막에서부터 데이터를 삭제하는 것은 빠르지만,
- 배열의 중간에 데이터를 추가하려면, 빈자리를 만들기 위해 다른 데이터를 복사해서 이동해야함.

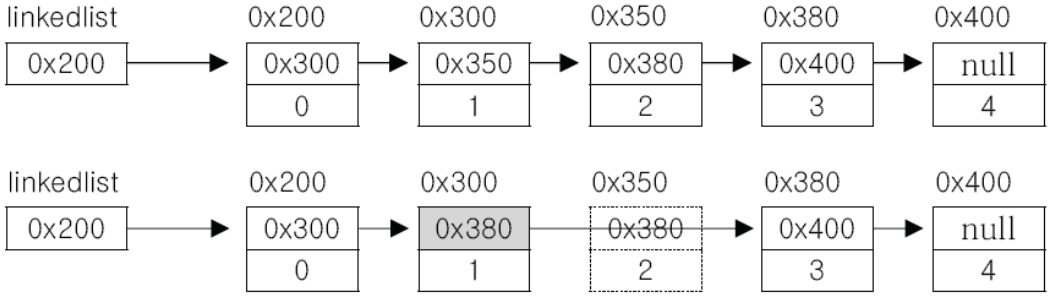
LinkedList

- 배열

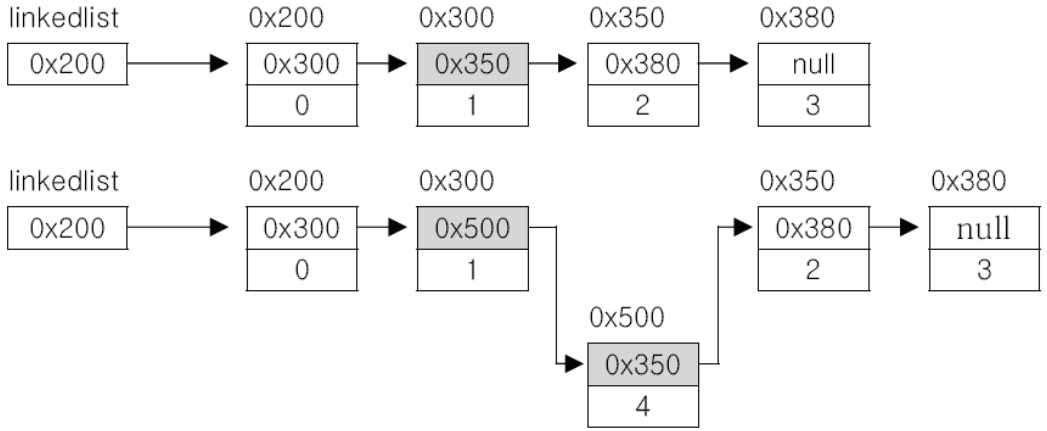


- LinkedList

▶ 데이터의 삭제 : 단 한 번의 참조변경만으로 가능



▶ 데이터의 추가 : 하나의 Node객체생성과 한 번의 참조변경만으로 가능



LinkedList

LinkedList 장점

- 구조가 간단하고 데이터를 읽어오는 데 걸리는 시간(접근시간, access time)이 가장 빠르다.
- 리스트 내에서 자료의 이동이 필요하지 않다.
- 사용 후 기억 장소의 재사용이 가능하다.
- 데이터 삭제시 삭제하고자 하는 요소의 이전요소가 삭제하고자 하는 요소의 다음요소를 참조하도록 변경하기만 하면 됨.
  =>처리속도 빠름

LinkedList

LinkedList 단점

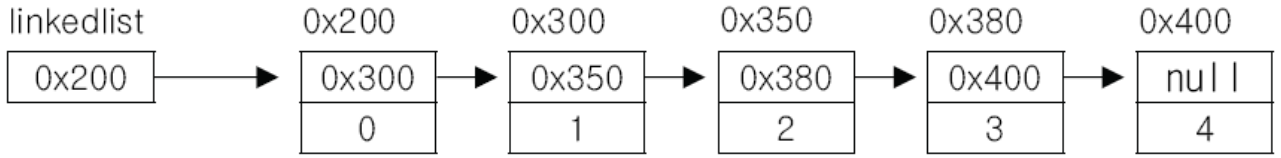
- 이동방향이 단방향이기때문에 다음요소에 대한 접근은 쉽지만 이전요소에 대한 접근이 어려움
- 이 점을 보완한 것이 더블 링크드 리스트(이중 연결 리스트)

Doubly circular LinkedList

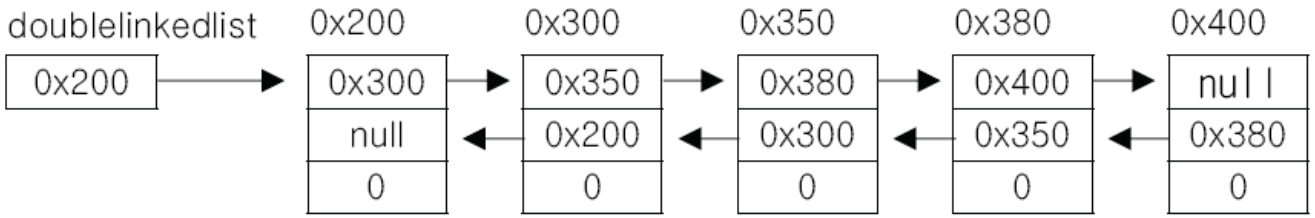
- 단순히 더블 링크드 리스트의 첫번째 요소와 마지막 요소를 서로 연결시킨 것. => 링크드 리스트의 단점인 접근성을 높이기 위한 것

LinkedList

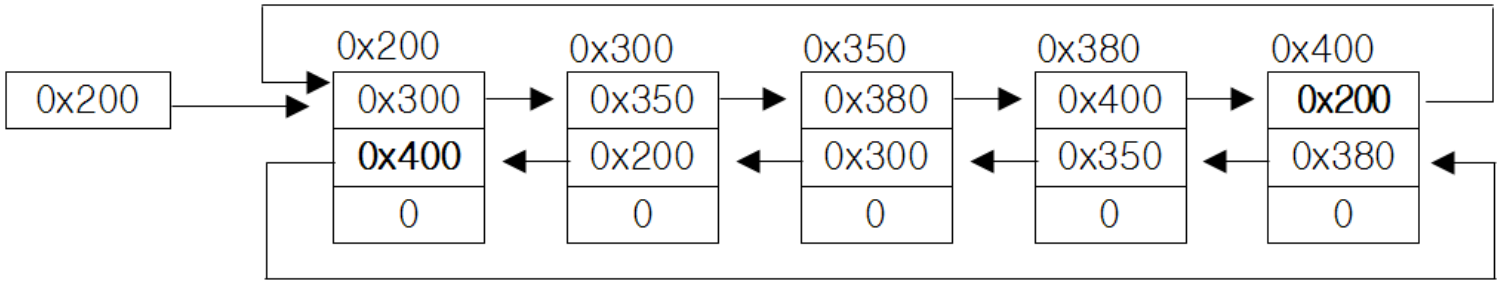
▶ 링크드 리스트(linked list) – 연결리스트. 데이터 접근성이 나쁨



▶ 더블리 링크드 리스트(doubly linked list) – 이중 연결리스트, 접근성 향상



▶ 더블리 써큘러 링크드 리스트(doubly circular linked list) – 이중 원형 연결리스트



ArrayList LinkedList 성능차이

- 순차적으로 데이터를 추가/삭제하는 경우(마지막 데이터부터 역순으로 삭제), ArrayList가 빠르다.
- 중간 데이터를 추가/삭제하는 경우, LinkedList가 빠르다.
- 접근시간(access time)은 ArrayList가 빠르다.

```
n번째 데이터의 주소 = 배열의 주소 + n * 데이터 타입의 크기
```

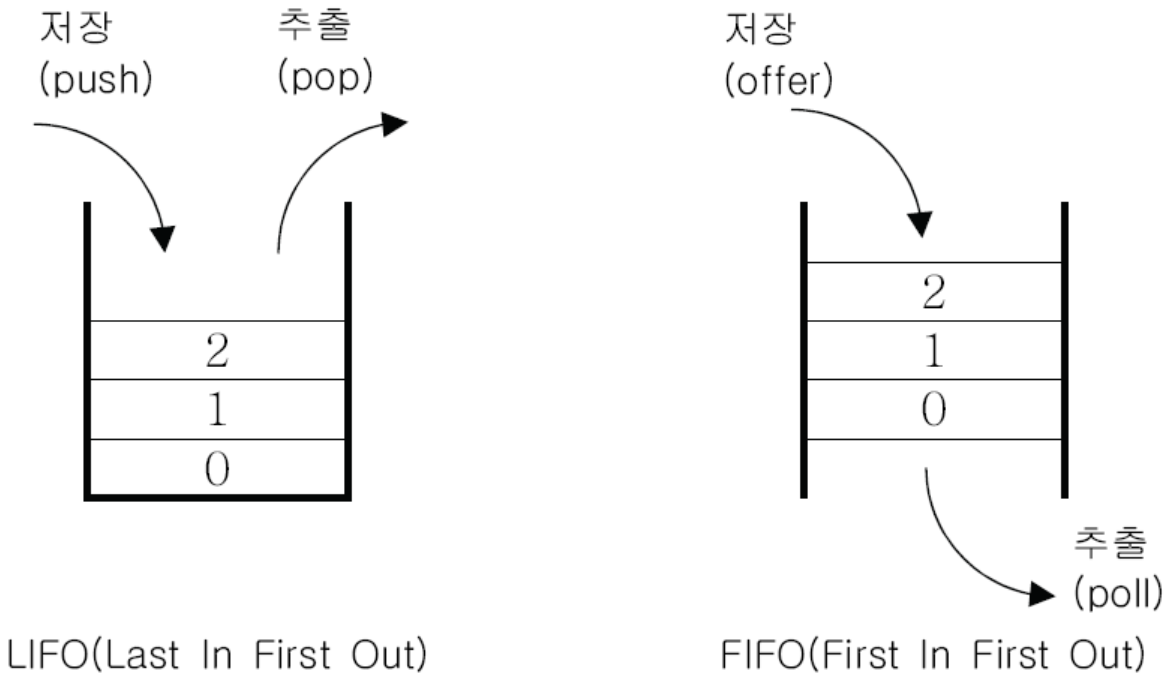
- => LinkedList는 저장해야하는 데이터의 개수가 많아질수록 데이터를 읽어오는 시간(접근 시간)이 길어진다는 단점이 있음.

Stack과 Queue

- 스택(Stack) : LIFO구조. 마지막에 저장된 것을 제일 먼저 꺼내게 된다.
- 수식계산, 수식괄호검사, undo/redo, 뒤로/앞으로(웹브라우져)등에 사용됨.

- 큐(Queue) : FIFO구조. 제일 먼저 저장한 것을 제일 먼저 꺼내게 된다.
- 최근 사용문서, 인쇄작업대기목록, 버퍼(buffer)등에 사용됨.

Stack과 Queue



- 순차적으로 데이터를 추가하고 삭제하는 stack에는 arrayList와 같은 배열기반의 컬렉션 클래스가 적합.

- Queue는 데이터를 꺼낼 때 항상 첫번째 저장된 데이터를 삭제하므로 arrayList보다 데이터의 추가/삭제가 쉬운 LinkedList로 구현하는 것이 더 적합.

Stack과 Queue

PriorityQueue

- Queue인터페이스의 구현체중 하나.
- 저장한 순서에 관계없이 우선순위가 높은것부터 꺼내게 되는 특징.
- Null 저장 X
- 저장공간으로 배열을 사용하며 각 요소를 힙 이라는 자료구조의 형태로 저장.(가장 큰 값이나 가장 작은 값을 빠르게 찾을 수 있다는 특징이 있음)

Stack과 Queue

DeQue

- Queue의 변형으로, 한쪽끝으로만 추가/삭제 할 수있는 큐와 달리 디큐는 양쪽 끝에 추가/삭제가 가능하다.

Iterator, ListIterator

Iterator

- 컬렉션에 저장된 요소들을 읽어오는 표준화된 방법.
- 컬렉션 클래스에 저장된 요소들을 나열하는 방법을 제공.
- 컬렉션 클래스의 iterator()를 호출해서 Iterator를 구현한 객체를 얻는다.

메서드	설 명
boolean hasNext()	읽어 올 요소가 남아있는지 확인한다. 있으면 true, 없으면 false를 반환한다.
Object next()	다음 요소를 읽어 온다. next()를 호출하기 전에 hasNext()를 호출해서 읽어 올 요소가 있는지 확인하는 것이 안전하다.
void remove()	next()로 읽어 온 요소를 삭제한다. next()를 호출한 다음에 remove()를 호출해야한다.(선택적 기능)

[표11-12] Iterator인터페이스의 메서드

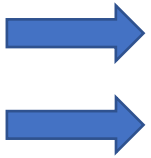
Iterator, ListIterator

ListIterator

- Iterator의 기능을 확장(상속)
- Iterator의 접근성을 향상시킨 것이 ListIterator이다.(단방향 →양방향)
- - listIterator()를 통해서 얻을 수 있다.(List를 구현한 컬렉션 클래스에 존재)
- 양방향 이동하기 전에 반드시 hasNext()나 hasPrevious()호출해서 이동할 수 있는지 확인해야함.

Iterator, ListIterator

ListIterator



메서드	설 명
void add(Object o)	컬렉션에 새로운 객체(o)를 추가한다.(선택적 기능)
boolean hasNext()	읽어 올 다음 요소가 남아있는지 확인한다. 있으면 true, 없으면 false를 반환한다.
boolean hasPrevious()	읽어 올 이전 요소가 남아있는지 확인한다. 있으면 true, 없으면 false를 반환한다.
Object next()	다음 요소를 읽어 온다. next()를 호출하기 전에 hasNext()를 호출해서 읽어 올 요소가 있는지 확인하는 것이 안전하다.
Object previous()	이전 요소를 읽어 온다. previous()를 호출하기 전에 hasPrevious()를 호출해서 읽어 올 요소가 있는지 확인하는 것이 안전하다.
int nextIndex()	다음 요소의 index를 반환한다.
int previousIndex()	이전 요소의 index를 반환한다.
void remove()	next() 또는 previous()로 읽어 온 요소를 삭제한다. 반드시 next()나 previous()를 먼저 호출한 다음에 이 메서드를 호출해야한다.(선택적 기능)
void set(Object o)	next() 또는 previous()로 읽어 온 요소를 지정된 객체(o)로 변경한다. 반드시 next()나 previous()를 먼저 호출한 다음에 이 메서드를 호출해야한다.(선택적 기능)

Arrays 주요 메소드

copyOf()	• **배열 전체**를 복사해서 새로운 배열을 생성한 후 반환합니다.
copyOfRange()	• **배열 일부**를 복사해서 새로운 배열을 생성한 후 반환합니다. • 지정된 범위의 끝은 포함되지 않습니다.
fill()	• 배열의 모든 요소를 지정된 값으로 채웁니다.
setAll()	• 배열을 채우는데 사용할 함수형 인터페이스를 매개변수로 받습니다. • 메소드 호출시 함수형 인터페이스를 구현한 객체를 매개변수나 람다식을 지정해야 합니다.
sort(Object [] o)	• 배열을 정렬할 때 사용합니다. • 객체 배열에 저장된 객체(Comparable을 구현한 클래스의 객체)에 구현된 내용에 따라 정렬됩니다.
sort(Object [] o, Comparator c)	• 지정한 Comparator에 의해 정렬합니다. • Comparator를 구현해서 정렬기준을 제공할 수 있습니다.
binarySearch()	• 배열에 저장된 요소를 검색할 때 사용합니다. • 배열에서 지정된 값이 저장된 위치(index)를 찾아서 반환합니다. 이때 배열이 정렬된 상태여야 정확한 결과를 얻을 수 없습니다. • 찾는 값과 일치하는 요소가 여러 개 존재하는 경우 이들 중 어떤 위치의 값이 반환될지는 알 수 없습니다.
toString()	• 배열의 모든 요소를 문자열로 출력할 수 있습니다. • toString()는 일차원 배열에서만 사용할 수 있으며 다차원 배열에서는 deepToString()메소드를 사용해야 합니다.
equals()	• 두 배열에 저장된 모든 요소를 비교해서 같으면 true를 그렇지않으면 false를 반환합니다. • equals()는 일차원 배열에서만 사용 가능하며 다차원 배열에서는 deepEquals()메소드를 사용해야 합니다.

HashSet

HashSet

- 중복된 요소를 저장하지 않으며, 순서를 유지하지 않는다.
- 저장순서를 유지하고자 한다면 => LinkedHashSet사용
- 새로운 요소를 추가할땐 add(), addAll() 사용
- 동일 객체에 대해 hashCode()를 여러 번 호출해도 동일한 값을 반환해야 한다.

- String인스턴스와 intege인스턴스는 서로다른 객체이므로 숫자가 같게 보여도 중복으로 간주하지 않음.

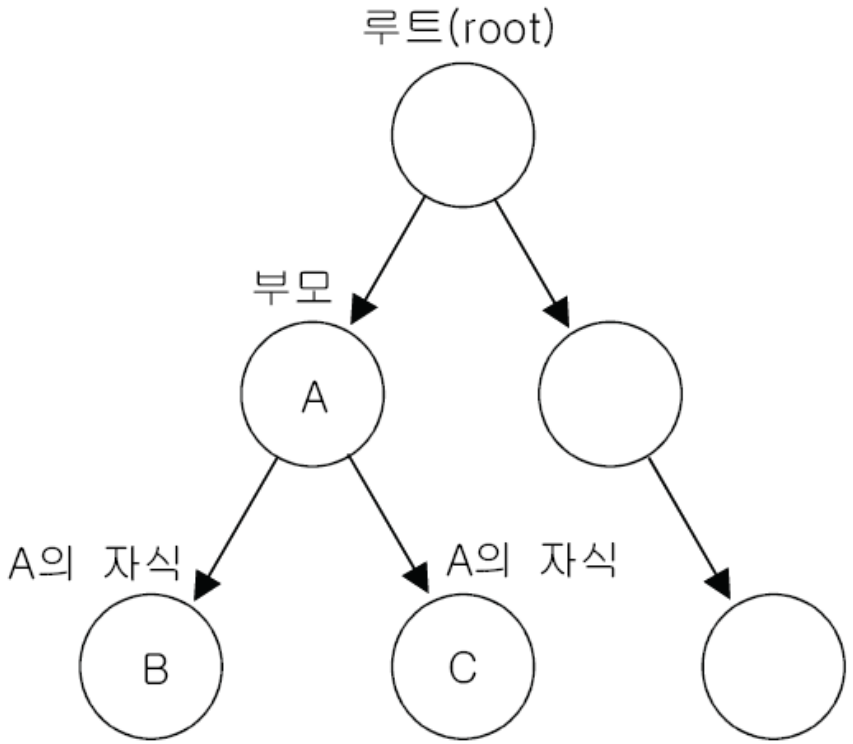
TreeSet

TreeSet

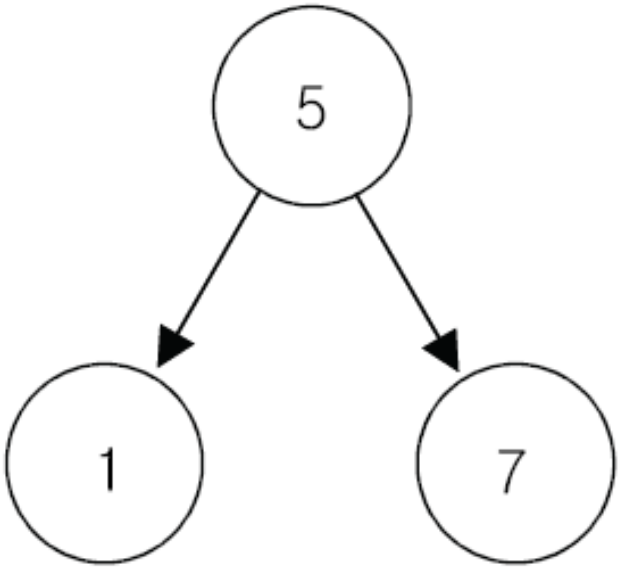
- 검색과 정렬에 유리하다.
- Set인터페이스를 구현한 컬렉션 클래스.(중복허용×, 순서유지×, 정렬저장○)
- 이진검색트리(binary search tree - 정렬, 검색에 유리)의 구조로 되어있다.
- 모든 트리는 하나의 루트(root node)를 가지며, 서로 연결된 두 요소를 부모자식관계에 있다 하고, 하나의 부모에 최대 두 개의 자식을 갖는다.
- 왼쪽 자식의 값은 부모의 값보다 작은 값을, 오른쪽 자식의 값은 부모보다 큰 값을 저장한다.
- 검색과 정렬에 유리하지만, HashSet보다 데이터 추가, 삭제시간이 더 걸린다.

TreeSet

TreeSet



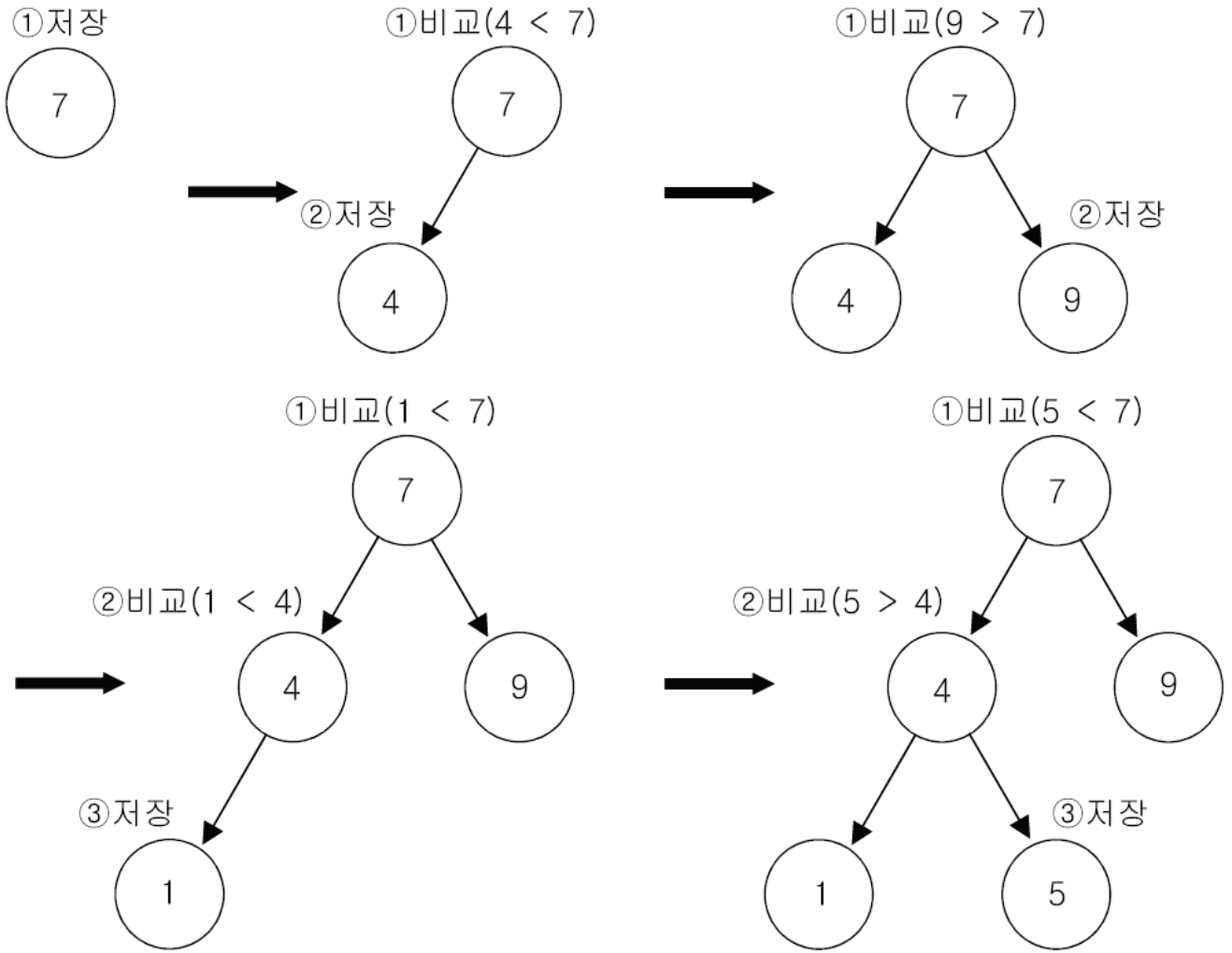
5,1,7 의 순서로 저장한 이진트리



TreeSet

TreeSet

7,4,9,1,5 순서로 데이터 저장



Hashtable, HashMap

Hashtable, HashMap

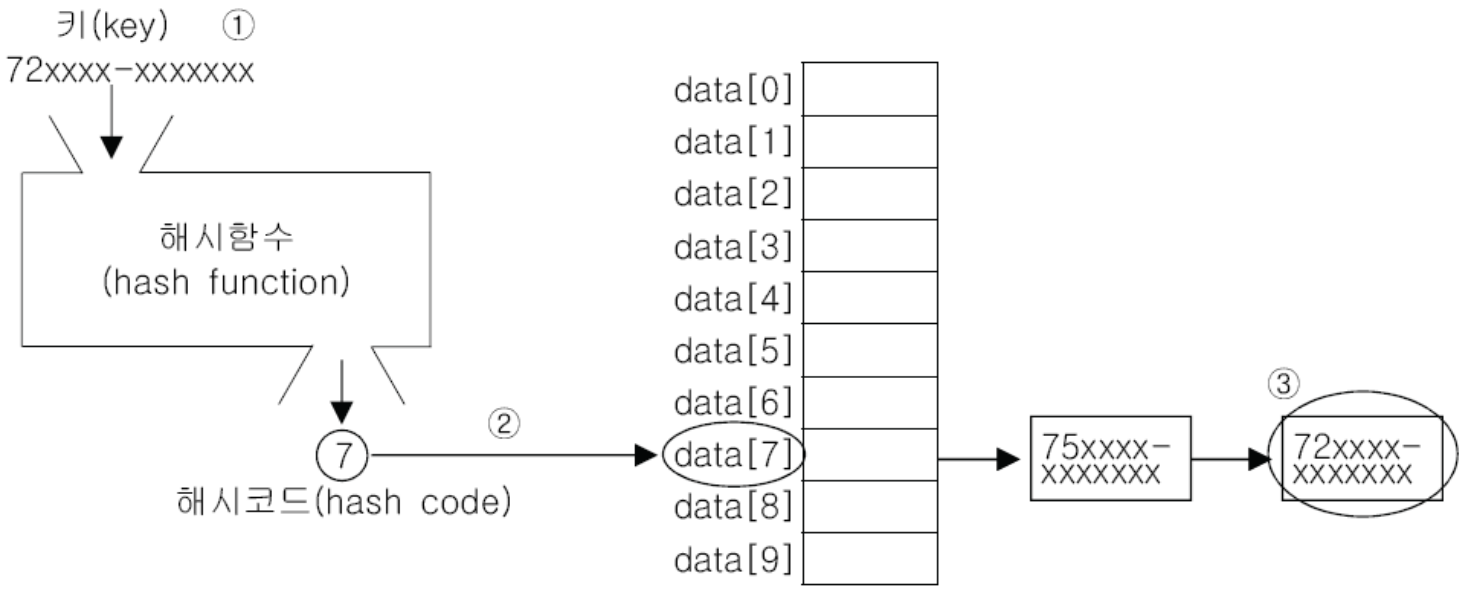
- HashMap은 Hashtable의 신버젼이며, Hashtable과 달리 HashMap은 동기화처리가 되어 있지 않다. 가능하면 Hashtable보다는 HashMap을 사용.
- HashMap은 해싱(hashing)기법을 사용해서 데이터를 저장하기 때문에 많은 양의 데이터를 검색할 때 성능이 뛰어나다.
- HashMap은 Map인터페이스를 구현하였으며, 데이터를 키와 값의 쌍으로 저장한다.
- Ex)

```
HashMap map = new HashMap();
map.put("castello", "1234");
map.put("asdf", "1111");
map.put("asdf", "1234");
```

- (“key”, ”value”)
- Key값은 유일해야함, value는 중복 허용.

해싱

- 해시함수를 이용해서 해시테이블에 저장하고 검색하는 기법



- 해시함수는 같은 키값에 대해 항상 같은 해시코드를 반환해야한다.
- 서로 다른 키값일지라도 같은 값의 해시코드를 반환할 수 있다.

TreeMap

TreeMap

- 이진검색트리의 형태로 키와 값의 쌍으로 이루어진 데이터를 저장
- Map의 장점인 빠른 검색과 Tree의 장점인 정렬과 범위검색의 장점을 모두 갖고 있다.
- 이진검색트리처럼, 데이터를 저장할 때 정렬하기 때문에 저장시간이 길다는 단점을 가지고 있다.
- 정렬된 상태로 데이터를 조회하는 경우가 빈번하다면, 데이터를 조회할 때 정렬해야 하는 HashMap보다는 이미 정렬된 상태로 저장되어 있는 TreeMap이 빠른 조회결과를 얻을 수 있다.
- 주로 HashMap을 사용하고, 정렬이나 범위검색이 필요한 경우에만 TreeMap을 사용하는 것이 좋다.

참고

1. 자바의 정석 참고