chapter11 컬렉션 프레임워크

ch11-1 컬렉션 프레임워크

컬렉션-여러 객체를 모아놓은 것을 의미

프레임워크-표준화, 정형화된 프로그래밍 방식작업

컬렉션 프레임워크

-컬렉션(다수의 객체)를 다루기 위한 표준화된 프로그래 밍 방식

-컬렉션을 쉽고 편리하게 다룰 수 있는 다양한 클래스를 제공

-java.util패키지에 포함. JDK1.2부터 제공

컬렉션 클래스

-다수의 데이터를 저장 할 수 있는 클래스

(예 Vector, ArrayList, HashSet)

ch11-2 컬렉션 프레임워크의 핵심 인터페이스

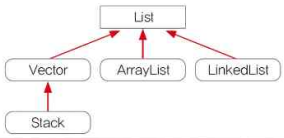
List / Set / Map

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

ch11-3 Collection 인터페이스의 메서드

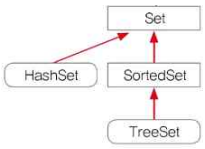
메서드	설 명
boolean add(Object o) boolean addAll(Collection c)	지정된 객체(o) 또는 Collection(c) 의 객체들을 Collection에 추가한다.
void clear( )	Collection의 모든 객체를 삭제한다.
boolean contains(Object o) boolean containsAll(Collection c)	지정된 객체(o) 또는 Collection의 객체들이 Collection에 포함되어 있는지 확인한다.
boolean equals(Object o)	동일한 Collection인지 비교한다.
int hashCode( )	Collection의 hash code를 반환한다.
boolean isEmpty( )	Collection이 비어있는지 확인한다.
Iterator iterator( )	Collection의 Iterator를 얻어서 반환한다.
boolean remove(Object o)	지정된 객체를 삭제한다.
boolean removeAll(Collection c)	지정된 Collection에 포함된 객체들을 삭제한다.
boolean retainAll(Collection c)	지정된 Collection에 포함된 객체만을 남기고 다른 객체들은 Collection 에서 삭제한다. 이 작업으로 인해 Collection에 변화가 있으면 true를 그렇지 않으면 false를 반환한다.
int size( )	Collection에 저장된 객체의 개수를 반환한다.
Object[ ] toArray( )	Collection에 저장된 객체를 객체배열(Object[ ])로 반환한다.
Object[ ] toArray(Object[ ] a)	지정된 배열에 Collection의 객체를 저장해서 반환한다.

ch11-4 List인터페이스 - 순서O, 중복O



메서드	설 명
void add(int index, Object element) boolean addAll(int index, Collection c)	지정된 위치(index)에 객체(element) 또는 컬렉션에 포함된 객체들을 추가한다.
Object get(int index)	지정된 위치(index)에 있는 객체를 반환한다.
int indexOf(Object o)	지정된 객체의 위치(index)를 반환한다. (List의 첫 번째 요소부터 순방향으로 찾는다.)
int lastIndexOf(Object o)	지정된 객체의 위치(index)를 반환한다. (List의 마지막 요소부터 역방향으로 찾는다.)
ListIterator listIterator( ) ListIterator listIterator(int index)	List의 객체에 접근할 수 있는 ListIterator를 반환한다.
Object remove(int index)	지정된 위치(index)에 있는 객체를 삭제하고 삭제된 객체를 반환한다.
Object set(int index, Object element)	지정된 위치(index)에 객체(element)를 저장한다
void sort(Comparator c)	지정된 비교자(comparator)로 List를 정렬한다.
List subList(int fromIndex, int toIndex)	지정된 범위(fromIndex부터 toIndex)에 있는 객체를 반환한다.

ch11-5 Set인터페이스 - 순서X, 중복X



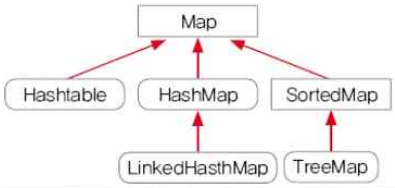
* Set인터페이스의 메서드 - Collection인터페이스와 동일

메서드	설 명
boolean add(Object o)	지정된 객체(o)를 Collection에 추가한다.
void clear( )	Collection의 모든 객체를 삭제한다.
boolean contains(Object o)	지정된 객체(o)가 Collection에 포함되어 있는지 확인한다.
boolean equals(Object o)	동일한 Collection인지 비교한다.
int hashCode( )	Collection의 hash code를 반환한다.
boolean isEmpty( )	Collection이 비어있는지 확인한다.
Iterator iterator( )	Collection의 Iterator를 얻어서 반환한다.
boolean remove(Object o)	지정된 객체를 삭제한다.
int size( )	Collection에 저장된 객체의 개수를 반환한다.
Object[ ] toArray( )	Collection에 저장된 객체를 객체배열(Object[ ])로 반환한다.
Object[ ] toArray(Object[ ] a)	지정된 배열에 Collection의 객체를 저장해서 반환한다.

* 집합과 관련된 메서드(Collection에 변화가 있으면 true, 아니면 false를 반환.

메서드	설 명
boolean addAll(Collection c)	지정된 Collection(c)의 객체들을 Collection에 추가한다.(합집합)
boolean containsAll(Collection c)	지정된 Collection의 객체들이 Collection에 포함되어 있는지 확인한다.(부분집합)
boolean removeAll(Collection c)	지정된 Collection에 포함된 객체들을 삭제한다.(차집합)
boolean retainAll(Collection c)	지정된 Collection에 포함된 객체만을 남기고 나머지는 Collection에서 삭제한다.(교집합)

ch11-6 Map인터페이스-순서X, 중복(키X, 값O)



메서드	설 명
void clear( )	Map의 모든 객체를 삭제한다.
boolean containsKey(Object key)	지정된 key객체와 일치하는 Map의 key객체가 있는지 확인한다.
boolean containsValue(Object value)	지정된 value객체와 일치하는 Map의 value객체가 있는지 확인한다.
Set entrySet( )	Map에 저장되어 있는 key-value쌍을 Map.Entry타입의 객체로 저장한 Set으로 반환한다.
boolean equals(Object o)	동일한 Map인지 비교한다.
Object get(Object key)	지정한 key객체에 대응하는 value객체를 찾아서 반환한다.
int hashCode( )	해시코드를 반환한다.
boolean isEmpty( )	Map이 비어있는지 확인한다.
Set keySet( )	Map에 저장된 모든 key객체를 반환한다.
Object put(Object key, Object value)	Map에 value객체를 key객체에 연결(mapping)하여 저장한다.
void putAll(Map t)	지정된 Map의 모든 key-value쌍을 추가한다.
Object remove(Object key)	지정한 key객체와 일치하는 key-value객체를 삭제한다.
int size( )	Map에 저장된 key-value쌍의 개수를 반환한다.
Collection values( )	Map에 저장된 모든 value객체를 반환한다.

ch11-7 ArrayList

-ArrayList는 기존의 Vector를 개선한 것으로 구현원리와 기능적으로 동일. ArrayList와 달리 Vector는 자체적으로 동기화처리가 되어있다.

-List인터페이스를 구현하으모, 저장순서 유지, 중복 허용

-데이터의 저장공간으로 배열을 사용한다. (배열기반)

ch11-8 ArrayList의 메서드

ArrayList( )
ArrayList(Collection c)
ArrayList(int initialCapacity)

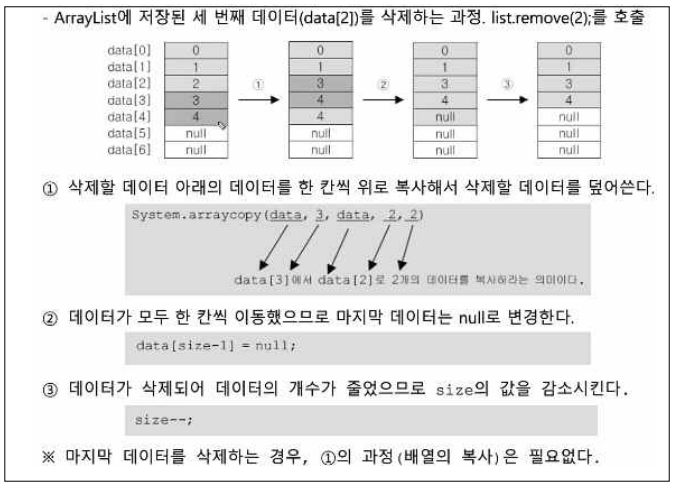
boolean add(Object o)
void add(int index, Object element)
boolean addAll(Collection c)
boolean addAll(int index, Collection c)

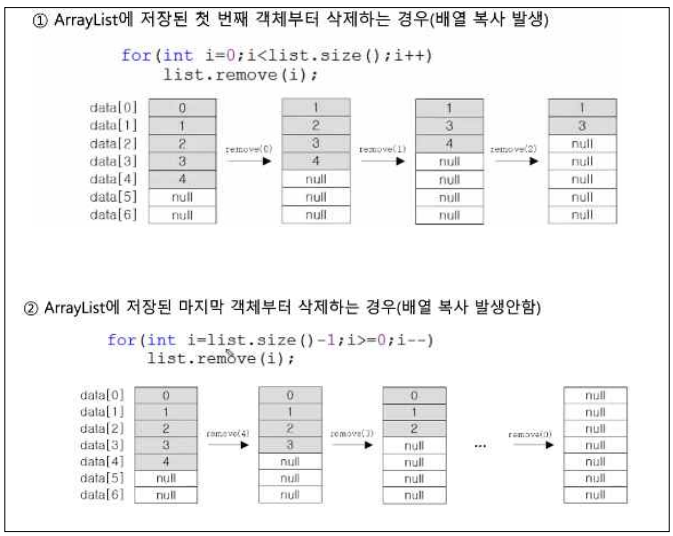
boolean remove(Object o)
Object remove(int index)
boolean removeAll(Collection c)
void clear( )

int indexOf(Object o)
int lastIndexOf(Object o)
boolean contains(Object o)
Object get(int index)
Object set(int index, Object element)

List subList(int fromIndex, int toIndex)
Object[ ] toArray( )
Object[ ] toArray(Object[ ] a)
boolean isEmpty( )
void trimToSize( )
int size( )

ch11-10ArrayList에 저장된 객체의 삭제과정





*java API 소스보기 - /jdk설치경로/src.zip

ch11-12 LinkedList - 배열의 장단점

장점 : 배열은 구조가 간단하고 데이터를 읽는 데 걸리는 시간(접근시간, access time)이 짧다.

단점 1.크기를 변경할 수 없다.
-크기를 변경해야 하는 경우 새로운 배열을 생성 후 데이터를 복사해야함.
2.비순차적인 데이터의 추가,삭제 시간이 많이 걸림
-데이터를 추가하거나 삭제 하기위해, 다른 데이터를 옮겨야함
-그러나 순차적인 데이터 추가와 삭제는 빠르다.

(끝에 추가) (끝부터 삭제)

ch11-12 LinkedList - 배열의 단점을 보완

-배열과 달리 링크드리스트는 불연속적으로 존재하는 데이터를 연결한다.
데이터의 삭제 : 한번의 참조 변경만으로 가능(변경유리)
데이터의 추가 : 한번의 Node객체생성과 두 번의 참조변경만으로 가능

LinkedList의 단점도 있다!
불연속 적이기 때문에 데이터의 접근성이 나쁨 ->
이중연결리스트로 접근성 향상 (더블리 링크드 리스트)->

```
class Node {
    Node next;
    Object obj;
}
```

이중 원형 연결리스트(더블리 써큘러 링크드 리스트)

```
class Node {
    Node next;
    Node previous;
    Object obj;
}
```

ArrayList vs LinkedList - 성능비교
1.순차적으로 데이터를 추가, 삭제-ArrayList가 빠름
2.비순차적으로 데이터를 추가, 삭제-ListLinked가 빠름
3.접근시간 - ArrayList가 빠름

컬렉션	읽기(접근시간)	추가 / 삭제	비 고
ArrayList	빠르다	느리다	순차적인 추가삭제는 더 빠름. 비효율적인 메모리사용
LinkedList	느리다	빠르다	데이터가 많을수록 접근성이 떨어짐

ch11-15 스택과 큐

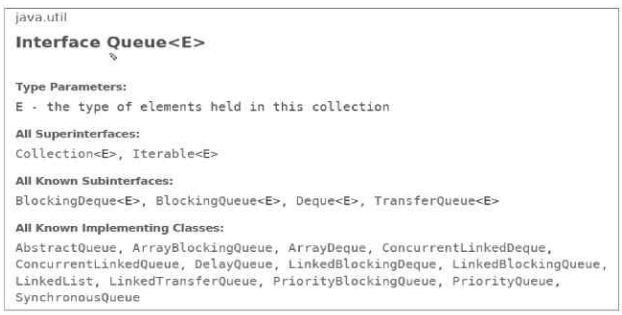
스택 - 마지막에 저장된 것을 제일 먼저 꺼낸다.
큐 - 제일 먼저 저장한 것을 제일 먼저 꺼낸다.

ch11-16 스택과 큐의 메서드

메서드	설 명
boolean empty( )	Stack이 비어있는지 알려준다.
Object peek( )	Stack의 맨 위에 저장된 객체를 반환. pop( )과 달리 Stack에서 객체를 꺼내지는 않음.(비었을 때는 EmptyStackException발생)
Object pop( )	Stack의 맨 위에 저장된 객체를 꺼낸다. (비었을 때는 EmptyStackException발생)
Object push(Object item)	Stack에 객체(item)를 저장한다.
int search(Object o)	Stack에서 주어진 객체(o)를 찾아서 그 위치를 반환. 못찾으면 -1을 반환. (배열과 달리 위치는 0이 아닌 1부터 시작)

▲ 표 11-1 Stack의 메서드

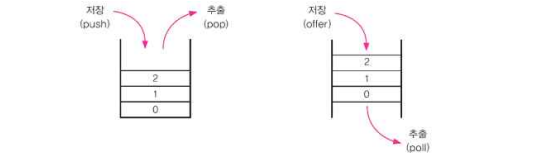
ch11-18 인터페이스를 구현한 클래스 찾기



▲ 그림 11-11 Java API문서에서 찾은 Queue

- > Queue q = new LinkedList(); 로 사용

ch11-19 스택과 큐의 활용



스택의 활용 예 – 수식계산, 수식괄호검사, 워드프로세서의 undo/redo, 웹브라우저의 뒤로/앞으로

큐의 활용 예 – 최근사용문서, 인쇄작업 대기목록, 버퍼(buffer)

ch11-22 Iterator, ListIterator, Enumeration

-컬렉션에 저장된 데이터를 접근(읽기)하는데 사용되는 인터페이스

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)
void forEachRemaining(Consumer<? super E> action)	컬렉션에 남아있는 요소들에 대해 지정된 작업(action)을 수행한다. 람다식을 사용하는 디폴트 메서드.(JDK1.8부터 추가)

▲ 표11-12 Iterator인터페이스의 메서드

-Enumeration은 Iterator의 구버전

메서드	설 명
boolean hasMoreElements( )	읽어 올 요소가 남아있는지 확인한다. 있으면 true, 없으면 false를 반환한다. Iterator의 hasNext( )와 같다.
Object nextElement( )	다음 요소를 읽어 온다. nextElement( )를 호출하기 전에 hasMoreElements( )를 호출해서 읽어올 요소가 남아있는지 확인하는 것이 안전하다. Iterator의 next( )와 같다.

▲ 표11-13 Enumeration인터페이스의 메서드

-ListIterator는 Iterator의 접근성을 향상 (단방향->양방향)
*Iterator만 잘 알면됨...

Iterator - 컬렉션마다 저장구조가 다른데 그 컬렉션에 저장된 요소들을 읽어오는 방법을 표준화한 것
(컬렉션이 바뀌어도 읽어오는 코드를 바꾸지 않아도 됨)

<컬렉션에 iterator()를 호출해서 객체 얻어 사용>

```
List list = new ArrayList(); // 다른 컬렉션으로 변경할 때는 이 부분만 고치면 된다.
Iterator it = list.iterator();

while(it.hasNext()) {  // boolean hasNext() 읽어올 요소가 있는지 확인
    System.out.println(it.next());  // Object next() 다음 요소를 읽어옴
}
```

ch11-24 Map과 Iterator

-Map에는 iterator()가 없다. (컬렉션 자손 아님)
-keySet(), entrySet(), values()를 호출하여 Set이나 컬렉션을 얻고 그 다음 iterator를 사용한다.

```
Map map = new HashMap();
        ...
Iterator it = map.entrySet().iterator();
```

```
Set eSet = map.entrySet();
Iterator it = eSet.iterator();
```

③

②

①

```
Iterator it = map.entrySet().iterator();
```

ch11-25 Arrays-배열을 다루기 편리한 메서드(static)제공

1. 배열의 출력 - toString()
   오버로딩된...
2. 배열의 복사 - copyOf(), copyOfRange()

```
int[] arr =  {0,1,2,3,4};
int[] arr2 = Arrays.copyOf(arr, arr.length);  // arr2=[0,1,2,3,4]
int[] arr3 = Arrays.copyOf(arr, 3);           // arr3=[0,1,2]
int[] arr4 = Arrays.copyOf(arr, 7);           // arr4=[0,1,2,3,4,0,0]
int[] arr5 = Arrays.copyOfRange(arr, 2, 4);   // arr5=[2,3] ← 4는 불포함
int[] arr6 = Arrays.copyOfRange(arr, 0, 7);   // arr6=[0,1,2,3,4,0,0]
```

3. 배열 채우기 - fill(), setAll()

```
int[] arr =  new int[5];
Arrays.fill(arr, 9);      // arr=[9,9,9,9,9]
Arrays.setAll(arr, (i) -> (int)(Math.random()*5)+1); //arr=[1,5,2,1,1]
```

4. 배열의 정렬과 검색 - sort(), binarySearch()

```
int[] arr = { 3, 2, 0, 1, 4};
int idx = Arrays.binarySearch(arr, 2);      // idx=-5 ← 잘못된 결과

Arrays.sort(arr);  // 배열 arr을 정렬한다.
System.out.println(Arrays.toString(arr));    // [0, 1, 2, 3, 4]
int idx = Arrays.binarySearch(arr, 2);       // idx=2  ← 올바른 결과
```

5. 다차원 배열의 출력 - deepToString

```
int[]   arr   = {0,1,2,3,4};
int[][] arr2D = {{11,12}, {21,22}};

System.out.println(Arrays.toString(arr)); // [0, 1, 2, 3, 4]
System.out.println(Arrays.deepToString(arr2D)); // [[11, 12], [21, 22]]
```

6. 다차원 배열의 비교 - deepEquals()

```
String[][] str2D  = new String[][]{{"aaa","bbb"},{"AAA","BBB"}};
String[][] str2D2 = new String[][]{{"aaa","bbb"},{"AAA","BBB"}};

System.out.println(Arrays.equals(str2D, str2D2));     // false
System.out.println(Arrays.deepEquals(str2D, str2D2)); // true
```

7. 배열을 List로 변환 - asList(Object... a)

```
List list = Arrays.asList(new Integer[]{1,2,3,4,5}); // list =[1, 2, 3, 4, 5]
List list = Arrays.asList(1,2,3,4,5);               // list =[1, 2, 3, 4, 5]
list.add(6);  // UnsupportedOperationException 예외 발생

List list = new ArrayList(Arrays.asList(1,2,3,4,5));
```

8.람다와 스트림관련 -parallelXXX(), spliterator(),stream()

ch11-30 Comparator와 Comparable

-객체 정렬에 필요한 메서드를 정의한 인터페이스
정렬기준을 제공한다.

Comparable	기본 정렬기준을 구현하는데 사용.
Comparator	기본 정렬기준 외에 다른 기준으로 정렬하고자할 때 사용

```
public interface Comparator {
    int compare(Object o1, Object o2); // o1, o2 두 객체를 비교
    boolean equals(Object obj); // equals를 오버라이딩하라는 뜻
}
public interface Comparable {
    int compareTo(Object o); // 주어진 객체(o)를 자신과 비교
}
```

-compare()와 compareTo()는 두 객체의 비교 결과를 반환 하도록 작성

```
public final class Integer extends Number implements Comparable {
    ...
    public int compareTo(Integer anotherInteger) {
        int v1 = this.value;
        int v2 = anotherInteger.value;
        // 값으면 0, 오른쪽 값이 크면 -1, 작으면 1을 반환
        return (v1 < v2 ? -1 : (v1==v2? 0 : 1));
    }
    ...
}
```

ch11-31 Comparator와 Comparable의 예제

예제 11-7

```
import java.util.*;

class Ex11_7 {
   public static void main(String[] args) {
      String[] strArr = {"cat", "Dog", "lion", "tiger"};

      Arrays.sort(strArr); // String의 Comparable구현에 의한 정렬
      System.out.println("strArr=" + Arrays.toString(strArr));

      Arrays.sort(strArr, String.CASE_INSENSITIVE_ORDER); // 대소문자 구분안함
      System.out.println("strArr=" + Arrays.toString(strArr));

      Arrays.sort(strArr, new Descending()); // 역순 정렬
      System.out.println("strArr=" + Arrays.toString(strArr));
   }
}

class Descending implements Comparator {
   public int compare(Object o1, Object o2){
      if( o1 instanceof Comparable && o2 instanceof Comparable) {
         Comparable c1 = (Comparable)o1;
         Comparable c2 = (Comparable)o2;
         return c1.compareTo(c2) * -1 ; // -1을 곱해서 기본 정렬방식의 역으로 변경한다.
                                        // 또는 c2.compareTo(c1)와 같이 순서를 바꿔도 된다.
      }
      return -1;
   }
}
```

결과

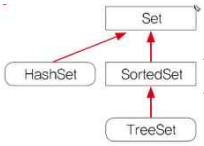
```
strArr=[Dog, cat, lion, tiger]
strArr=[cat, Dog, lion, tiger]
strArr=[tiger, lion, cat, Dog]
```

ch11-32 Integer 와 Comparable

```
public final class Integer extends Number implements Comparable {
    ...
  public int compareTo(Object o) {
     return compareTo((Integer)o);
  }

  public int compareTo(Integer anotherInteger) {
     int thisVal = this.value;
     int anotherVal = anotherInteger.value;

     // 비교하는 값이 크면 -1, 같으면 0, 작으면 1을 반환한다.
     return (thisVal<anotherVal ? -1 : (thisVal==anotherVal ? 0 : 1));
  }
    ...
}
```

```
// Arrays.sort()와 같은 메서드가 정렬을 수행하는 과정에서, compareTo()을 호출한다.
public int compareTo(Integer anotherInteger) {
   int thisVal = this.value;
   int anotherVal = anotherInteger.value;

   // 왼쪽 값이 크면 음수를, 두 값이 같으면 0, 왼쪽 값이 크면 양수를 반환한다.
   return thisVal - anotherVal; // 내림 차순의 경우 반대로 뺄셈하면 된다
}
```

ch11-34 HashSet - 순서X, 중복X



HashSet

-Set인터페이스를 구현한 대표적인 컬렉션 클래스

-순서를 유지하려면, LinkedHashSet클래스를 사용

TreeSet

-범위 검색과 정렬에 유리한 컬렉션 클래스

-HashSet보다 데이터 추가, 삭제에 시간이 더 걸림

ch11-34 HashSet 주요 메서드

HashSet( )
HashSet(Collection c)
HashSet(int initialCapacity)
HashSet(int initialCapacity, float loadFactor)

boolean add(Object o)
boolean addAll(Collection c)
boolean remove(Object o)
boolean removeAll(Collection c)
boolean retainAll(Collection c)
void clear( )

boolean isEmpty( )
int size( )
Object[ ] toArray( )
Object[ ] toArray(Object[ ] a)

boolean contains(Object o)
boolean containsAll(Collection c)
Iterator iterator( )

ch11-37 HashSet - 예제

-HashSet은 객체를 저장 하기전에 기존에 같은 객체가 있는지 확인

같은 객체가 없으면 저장하고, 있으면 저장하지 않는다.

-boolean add(Object o)는 저장객체의 equals()와 hashCode()를 호출

```
class Person {
    String name;
    int age;

    Person(String name, int age) {
        this.name = name;
        this.age = age;
    }

    public String toString() {
        return name +":"+ age;
    }
}
```

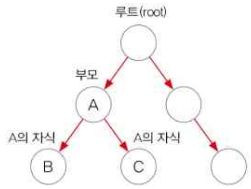
```
public boolean equals(Object obj) {
    if(!(obj instanceof Person)) return false;

    Person tmp = (Person)obj;

    return name.equals(tmp.name) && age==tmp.age;
}

public int hashCode() {
    return (name+age).hashCode();
}
```

ch11-39 TreeSet - 범위 탐색, 정렬

-이진 탐색 트리로 구현. 범위 탐색과 정렬에 유리

-이진 트리는 모든 노드가 최대 2개의 하위 노드를 갖음

-각 요소가 나무형태로 연결(LinkedList의 변형)



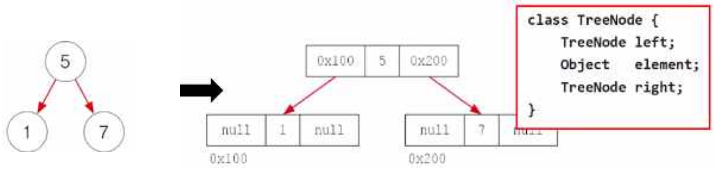

```
class Node {
    Node   next;    // 다음 요소의 주소를 저장
    Object obj;     // 데이터를 저장
}
```

```
class TreeNode {
    TreeNode left;    // 왼쪽 자식노드
    Object   element; // 저장할 객체
    TreeNode right;   // 오른쪽 자식노드
}
```

ch11-40 이진 탐색 트리

-부모보다 작은 값은 왼쪽 큰 값은 오른쪽에 저장

-데이터가 많아 질수록 추가, 삭제에 시간이 더 걸린다.



ch11-41 TreeSet-데이터 저장과정 boolean add(Object o)

*TreeSet에 7,4,9,1,5 순서로 저장시, 아래와 같은 과정

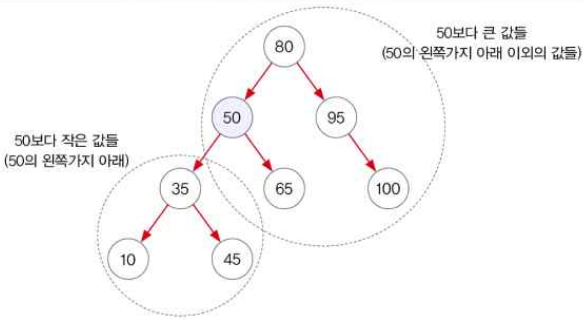
(루트부터 트리를 따라 내려가며 값을 비교, 작으면 왼쪽 크면 오른쪽에 저장)

ch11-42 TreeSet - 주요 생성자와 메서드

생성자 또는 메서드	설 명
TreeSet( )	기본 생성자
TreeSet(Collection c)	주어진 컬렉션을 저장하는 TreeSet을 생성
TreeSet(Comparator comp)	주어진 정렬기준으로 정렬하는 TreeSet을 생성
Object first( )	정렬된 순서에서 첫 번째 객체를 반환한다.
Object last( )	정렬된 순서에서 마지막 객체를 반환한다.
Object ceiling(Object o)	지정된 객체와 같은 객체를 반환. 없으면 큰 값을 가진 객체 중 제일 가까운 값의 객체를 반환. 없으면 null
Object floor(Object o)	지정된 객체와 같은 객체를 반환. 없으면 작은 값을 가진 객체 중 제일 가까운 값의 객체를 반환. 없으면 null
Object higher(Object o)	지정된 객체보다 큰 값을 가진 객체 중 제일 가까운 값의 객체를 반환. 없으면 null
Object lower(Object o)	지정된 객체보다 작은 값을 가진 객체 중 제일 가까운 값의 객체를 반환. 없으면 null
SortedSet subSet(Object fromElement, Object toElement)	범위 검색(fromElement와 toElement사이)의 결과를 반환한다.(끝 범위인 toElement는 범위에 포함되지 않음)
SortedSet headSet(Object toElement)	지정된 객체보다 작은 값의 객체들을 반환한다.
SortedSet tailSet(Object fromElement)	지정된 객체보다 큰 값의 객체들을 반환한다.

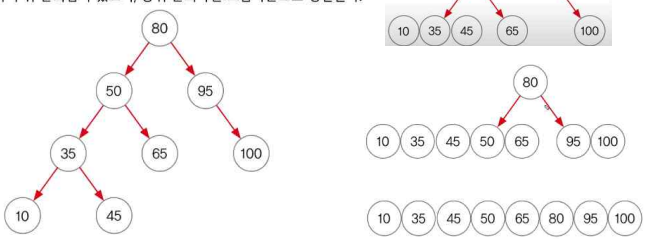
ch11-45 TreeSet - 예제 subSet(), headSet(), tailSet()

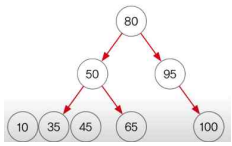
메서드	설 명
SortedSet subSet(Object fromElement, Object toElement)	범위 검색(fromElement와 toElement사이)의 결과를 반환한다.(끝 범위인 toElement는 범위에 포함되지 않음)
SortedSet headSet(Object toElement)	지정된 객체보다 작은 값의 객체들을 반환한다.
SortedSet tailSet(Object fromElement)	지정된 객체보다 큰 값의 객체들을 반환한다.



[알아두면 좋아요] 트리 순회(tree traversal)

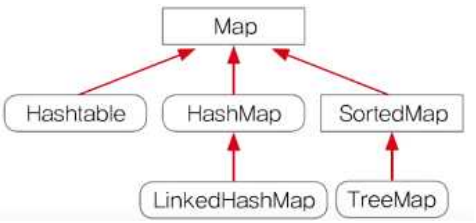
- 이진 트리의 모든 노드를 한번씩 읽는 것을 트리 순회라고 한다.
- 전위, 중위 후위 순회법이 있으며, 중위 순회하면 오름차순으로 정렬된다.





ch11-43 HashMap과 Hashtable 순서X,중복(키X,값O)

map인터페이스를 구현. 데이터를 키와 값의 쌍으로 저장
HashMap(동기화X)은 Hashtable(동기화O)의 신버젼



-HashMap
map인터페이스를 구현한 대표적인 클래스
순서를 유지하려면 LinkedHashMap클래스를 사용
-TreeMap
범위 검색과 정렬에 유리한 컬렉션 클래스
HashMap보다 데이터 추가, 삭제에 시간이 더걸림

ch11-47 HashMap의 키와 값

해싱기법으로 데이터를 저장. 데이터가 많아도 검색 빠름
map인터페이스를 구현. 데이터를 키와 값의 쌍으로 저장

키(key) 컬렉션 내의 키(key) 중에서 유일해야 한다.
값(value) 키(key)와 달리 데이터의 중복을 허용한다.

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

키(key)	값(value)
myId	1234
asdf	1234

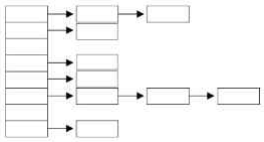
```
public class HashMap extends AbstractMap
                     implements Map, Cloneable,Serializable {
    transient Entry[] table;
    ...
    static class Entry implements Map.Entry {
        final Object key;
        Object value;
        ...
    }
}
```

비객체지향적인 코드	객체지향적인 코드

Object[] key;
Object[] value;
	Entry[] table;
	class Entry {
	Object key;
	Object value;
	}

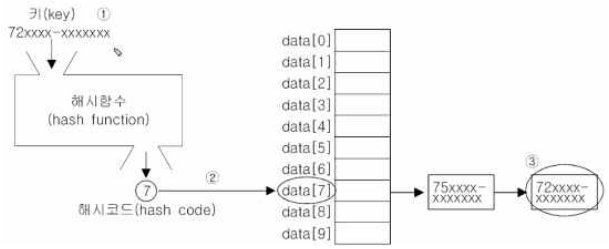
-해싱이란? 해시함수를 이용하여 해시테이블에 데이터를 저장하고 어떤 키 값에 대한 저장위치(해쉬코드)를 읽어(index값 반환)오는 것

-해시테이블은 배열과 링크드 리스트가 조합된 형태



여러 링크드리스트를 배열로 묶음
배열의 장점: 접근성
링크드리스트 장점 : 변경유리

-해시테이블에 저장된 데이터를 가져오는 과정



① 키로 해시함수를 호출해서 해시코드를 얻는다.
② 해시코드(해시함수의 반환값)에 대응하는 링크드리스트를 배열에서 찾는다.
③ 링크드리스트에서 키와 일치하는 데이터를 찾는다.
※ 해시함수는 같은 키에 대해 항상 같은 해시코드를 반환해야 한다.
서로 다른 키일지라도 같은 값의 해시코드를 반환할 수도 있다.

ch11-48 HashMap - 주요 메서드

HashMap( )
HashMap(int initialCapacity)
HashMap(int initialCapacity, float loadFactor)
HashMap(Map m)

Set entrySet( )
Set keySet( )
Collection values( )

키(key)	값(value)
myId	1234
asdf	1234

Object put(Object key, Object value)
void putAll(Map m)
Object remove(Object key)
Object replace(Object key, Object value)
boolean replace(Object key, Object oldValue, Object newValue)

Object get(Object key)
Object getOrDefault(Object key, Object defaultValue)
boolean containsKey(Object key)
boolean containsValue(Object value)

int size( )
boolean isEmpty( )
void clear( )
Object clone( )

ch11-52 Collections 컬렉션을 위한 메서드(static)를 제공

1.컬렉션 채우기,복사,정렬,검색 - fill(), copy(), binarySearch()등

2.컬렉션의 동기화 - synchronizedXXX()

```
static Collection synchronizedCollection(Collection c)
static List       synchronizedList(List list)
static Set        synchronizedSet(Set s)
static Map        synchronizedMap(Map m)
static SortedSet  synchronizedSortedSet(SortedSet s)
static SortedMap  synchronizedSortedMap(SortedMap m)
```

```
List syncList = Collections.synchronizedList(new ArrayList(...));
```

3.변경불가(readOnly) 컬렉션 만들기 unmodifiableXXX()

```
static  Collection    unmodifiableCollection(Collection c)
static  List          unmodifiableList(List list)
static  Set           unmodifiableSet(Set s)
static  Map           unmodifiableMap(Map m)
static  NavigableSet  unmodifiableNavigableSet(NavigableSet s)
static  SortedSet     unmodifiableSortedSet(SortedSet s)
static  NavigableMap  unmodifiableNavigableMap(NavigableMap m)
static  SortedMap     unmodifiableSortedMap(SortedMap m)
```

4. 싱글톤 컬렉션 만들기 - singletonXXX()

```
static  List singletonList(Object o)
static  Set  singleton(Object o)       // singletonSet이 아님에 주의
static  Map  singletonMap(Object key, Object value)
```

5. 한 종류의 객체만 저장하는 컬렉션 만들기 - checkdeXXX()

```
static  Collection    checkedCollection(Collection c, Class type)
static  List          checkedList(List list, Class type)
static  Set           checkedSet(Set s, Class type)
static  Map           checkedMap(Map m, Class keyType, Class valueType)
static  Queue         checkedQueue(Queue queue, Class type)
static  NavigableSet  checkedNavigableSet(NavigableSet s, Class type)
static  SortedSet     checkedSortedSet(SortedSet s, Class type)
static  NavigableMap  checkedNavigableMap(NavigableMap m, Class keyType, Class valueType)
static  SortedMap     checkedSortedMap(SortedMap m, Class keyType, Class valueType)
```

```
List list = new ArrayList();
List checkedList = checkedList(list, String.class); // String만 저장가능
checkedList.add("abc");              // OK.
checkedList.add(new Integer(3));     // 에러. ClassCastException발생
```

ch11-56 컬렉션 클래스 정리 & 요약