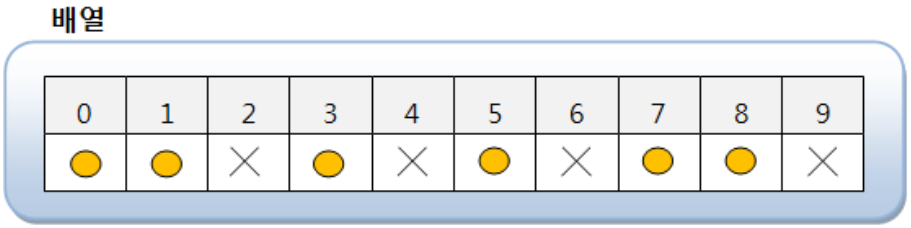
❖ 컬렉션 프레임워크(Collection Framework)

- 컬렉션
  - 사전적 의미로 요소(객체)를 수집해 저장하는 것

- 배열의 문제점
  - 저장할 수 있는 객체 수가 배열을 생성할 때 결정
    → 불특정 다수의 객체를 저장하기에는 문제

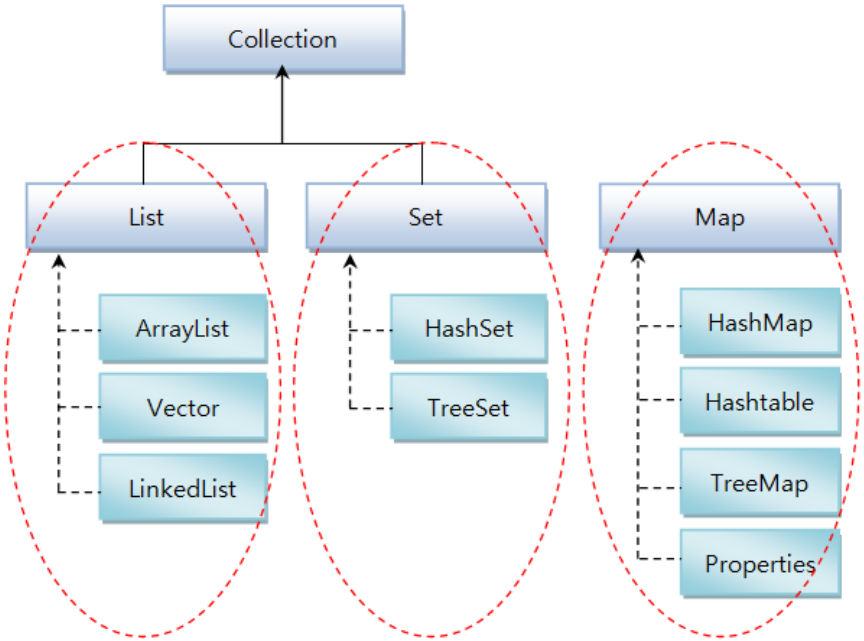
  - 객체 삭제했을 때 해당 인덱스가 비게 됨
    → 낱알 빠진 옥수수 같은 배열
    → 객체를 저장하려면 어디가 비어있는지 확인해야



❖ 컬렉션 프레임워크(Collection Framework)

- 객체들을 효율적으로 추가, 삭제, 검색할 수 있도록 제공되는 컬렉션 라이브러리
- java.util 패키지에 포함
- 인터페이스를 통해서 정형화된 방법으로 다양한 컬렉션 클래스 이용

❖ 컬렉션 프레임워크의 주요 인터페이스



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

■ Collection 인터페이스

메소드	설명
int size()	**원소의 개수 반환**
boolean isEmpty()	공백 상태이면 true 반환
boolean contains(Object obj)	obj를 포함하고 있으면 true 반환
boolean add(E element)	**원소 추가**
E get(int index)	**원소 조회**
boolean remove(Object Obj)	**원소 삭제**
Iterator<E> iterator()	원소 확인
boolean addAll(Collection<? Extends E> c)	c에 있는 모든 원소 추가
boolean containsAll(Collection<?> c)	c에 있는 모든 원소가 포함되어 있으면 true
boolean removeAll(Collection<?> c)	c에 있는 모든 원소 삭제
void clear()	**모든 원소 삭제**
Object[] toArray()	컬렉션을 배열로 변환
<T> T[] toArray(T[] a)	컬렉션을 배열로 변환

■ ArrayList<E>

✓ List<E> 인터페이스를 구현하는 대표적인 제네릭 클래스

✈ ArrayList<E>, LinkedList<E>

✓ List<E> 인터페이스를 구현 클래스의 인스턴스 저장 특징

✈ 동일한 인스턴스의 중복 저장을 허용한다.

✈ 인스턴스의 저장 순서가 유지된다.

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

    /* 데이터의 저장 */
    list.add(new Integer(11));
    list.add(new Integer(22));
    list.add(new Integer(33));

    /* 데이터의 참조 */
    System.out.println("1차 참조");
    for(int i=0; i<list.size(); i++)
        System.out.println(list.get(i));   // 0이 첫 번째

    /* 데이터의 삭제 */
    list.remove(0);     // 0이 전달되었으므로 첫 번째 데이터 삭제
    System.out.println("2차 참조");
    for(int i=0; i<list.size(); i++)
        System.out.println(list.get(i));
}
```

ArrayList<E>는 이름이 의미하듯이 배열 기반으로 데이터를 저장한다.

실행 결과

```
1차 참조
11
22
33
2차 참조
22
33
```

■ ArrayList 사용

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

        list.add("1번");
        list.add("2번");

        list.add("=> 3번");
        list.add("=> 4번");

        System.out.println("ArrayList elements");
        for (int idx = 0; idx < list.size(); idx++) {
            System.out.println(list.get(idx));
        }
        list.remove(3);
        list.remove(2);

        list.add(0, "=> 3번");
        list.add(1, "=> 4번");

        System.out.println("ArrayList elements");
        for (int idx = 0; idx < list.size(); idx++) {
            System.out.println(list.get(idx));
        }
    }
}
```

■ LinkedList<E>

✓ 데이터의 저장방식

✈ 이름이 의미하듯이 '리스트'라는 자료구조를 기반으로 데이터를 저장한다.

✓ 사용방법

✈ ArrayList<E>의 사용방법과 거의 동일하다! 다만, 데이터를 저장하는 방식에서 큰 차이가 있을 뿐이다.

✈ 대부분의 경우 ArrayList<E>를 대체할 수 있다.

```
public static void main(String[] args)
{
    LinkedList<Integer> list=new LinkedList<Integer>();

    /* 데이터의 저장 */
    list.add(new Integer(11));
    list.add(new Integer(22));
    list.add(new Integer(33));

    /* 데이터의 참조 */
    System.out.println("1차 참조");
    for(int i=0; i<list.size(); i++)
        System.out.println(list.get(i));

    /* 데이터의 삭제 */
    list.remove(0);
    System.out.println("2차 참조");
    for(int i=0; i<list.size(); i++)
        System.out.println(list.get(i));
}
```

실행결과

```
1차 참조
11
22
33
2차 참조
22
33
```

■ LinkedList 사용

```
public class LinkedListExam {
    public static void main(String args[]) {
        LinkedList<String> list =
                new LinkedList<String>();

        list.add("B");
        list.add("D");
        list.add("C");
        list.addLast("Z");
        list.addFirst("A");
        list.add(1, "A2");
        System.out.println(
                "LinkedList elements : " + list);
        list.remove("C");
        list.remove(2);
        System.out.println("after delete : " + list);

        list.removeFirst();
        list.removeLast();
        System.out.println(
                "after delete first and last : " + list);

        String val = list.get(1);
        list.set(1, val + " Changed");
        System.out.println("after change : " + list);

    }
}
```

■ ArrayList<E> 와 LinkedList<E> 의 차이점

ArrayList<E>의 특징, 배열의 특징과 일치한다.

- 저장소의 용량을 늘리는 과정에서 많은 시간이 소요된다. — ArrayList<E>의 **단점**
- 데이터의 삭제에 필요한 연산과정이 매우 길다. — ArrayList<E>의 **단점**
- 데이터의 참조가 용이해서 빠른 참조가 가능하다. — ArrayList<E>의 **장점**

LinkedList<E>의 특징, 리스트 자료구조의 특징과 일치한다.

- 저장소의 용량을 늘리는 과정이 간단하다. — LinkedList<E>의 **장점**
- 데이터의 삭제가 매우 간단하다. — LinkedList<E>의 **장점**
- 데이터의 참조가 다소 불편하다. — LinkedList<E>의 **단점**

■ 컬렉션 클래스를 이용한 기본 자료형 저장

```
ArrayList<int> arr=new ArrayList<int>( );          error

LinkedList<int> link=new LinkedList<int>( );       error
```

기본 자료형 정보를 이용해서 제네릭 인스턴스 생성 불가능! 따라서 Wraper 클래스를 기반으로 컬렉션 인스턴스를 생성한다.

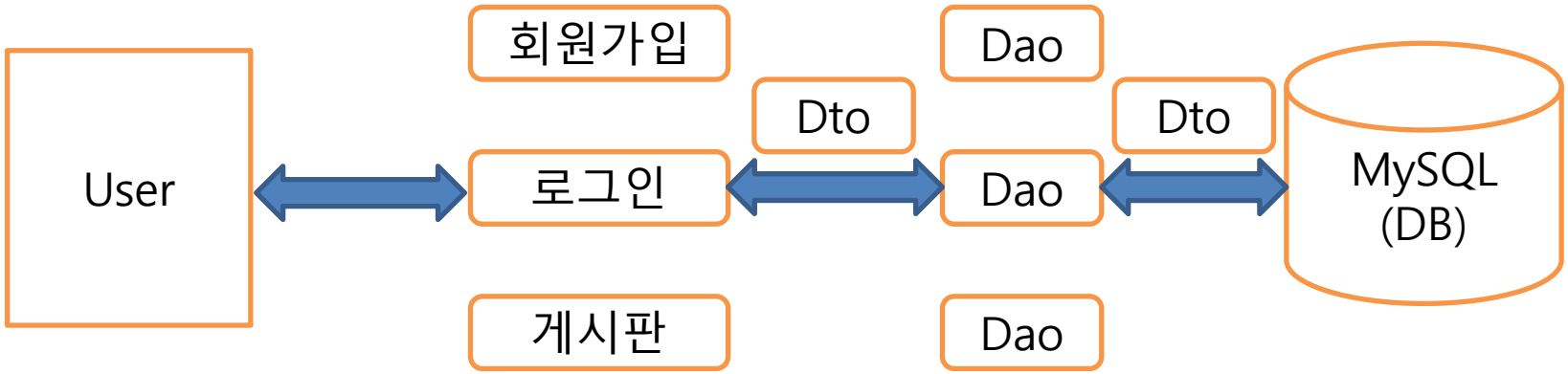
```
public static void main(String[] args)
{
    LinkedList<Integer> list=new LinkedList<Integer>();
    list.add(10);       // Auto Boxing
    list.add(20);       // Auto Boxing
    list.add(30);       // Auto Boxing

    Iterator<Integer> itr=list.iterator();

    while(itr.hasNext())
    {
        int num=itr.next();     // Auto Unboxing
        System.out.println(num);
    }
}
```

```
10
20
30
```

실행결과

Auto Boxing과 Auto Unboxing의 도움으로 정수 단위의 데이터 입출력이 매우 자연스럽다!

■ 데이터베이스 조회 (1 / 4)



■ 데이터베이스 조회 (2 / 4)

```
public class MemberDto {
    private String id;
    private String pass;

    public String getId() {
        return id;
    }
    public void setId(String id) {
        this.id = id;
    }
    public String getPass() {
        return pass;
    }
    public void setPass(String pass) {
        this.pass = pass;
    }
}
```

■ 데이터베이스 조회 (3 / 4)

```
public class MemberDao {
    public List<MemberDto> getMemberList() {
        List<MemberDto> memberList = new ArrayList<MemberDto>();
        for(int i = 0; i < 5; i++) {
            MemberDto member = new MemberDto();
            member.setId("id-" + i);
            member.setPass("pass-" + i);
            memberList.add(member);
        }
        return memberList;
    }
}
```

■ 데이터베이스 조회 (4 / 4)

```
public class DaoLauncher {
    public static void main(String[] args) {
        MemberDao dao = new MemberDao();

        List<MemberDto> list = dao.getMemberList();

        for(int i = 0; i < list.size(); i++) {
            MemberDto dto = list.get(i);

            String id = dto.getId();
            String pass = dto.getPass();
            System.out.println(id + ", " + pass);
        }
    }
}
```

■ Set<E> 인터페이스, HashSet<E> 클래스

- List<E>를 구현하는 클래스들과 달리 Set<E>를 구현하는 클래스들은 데이터의 저장순서를 유지하지 않는다.
- List<E>를 구현하는 클래스들과 달리 Set<E>를 구현하는 클래스들은 데이터의 중복저장을 허용하지 않는다. 단, 동일 데이터에 대한 기준은 프로그래머가 정의
- 즉, Set<E>를 구현하는 클래스는 '집합'의 성격을 지닌다.

```
public static void main(String[] args)
{
    HashSet<String> hSet=new HashSet<String>();
    hSet.add("First");
    hSet.add("Second");
    hSet.add("Third");
    hSet.add("First");

    System.out.println("저장된 데이터 수 : "+hSet.size());

    Iterator<String> itr=hSet.iterator();
    while(itr.hasNext())
        System.out.println(itr.next());
}
```

동일한 문자열 인스턴스는 저장되지 않았다. 그렇다면 동일 인스턴스를 판단하는 기준은?

실행결과

```
저장된 데이터 수 : 3
Third
Second
First
```

■ HashSet 사용

```
public class HashSetExam {
    public static void main(String args[]) {
        HashSet<String> hs = new HashSet<String>();

        hs.add("1"); hs.add("2"); hs.add("3");
        hs.add("B"); hs.add("A"); hs.add("C");
        hs.add("나"); hs.add("가"); hs.add("다");

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

■ 로또 번호 생성기 – 1 (List 사용)

```
public class LottoList {
    public static void main(String[] args) {
        List<Integer> ts = new ArrayList<Integer>();
        Random random = new Random();
        while(true) {
            int num = random.nextInt(45) + 1;
            if(!ts.contains(num)) {
                ts.add(num);
            }
            if(ts.size() >= 6) {
                break;
            }
        }
        System.out.println(ts);
    }
}
```

■ 로또 번호 생성기 – 2 (Set 사용)

```
public class LottoSet {
    public static void main(String[] args) {
        Set<Integer> ts = new TreeSet<Integer>();
        Random random = new Random();
        while(ts.size() < 6) {
            int randomNum = random.nextInt(45) + 1;
            ts.add(randomNum);
        }
        System.out.println(ts);
    }
}
```

■ TreeSet<E> 클래스

- TreeSet<E> 클래스는 트리라는 자료구조를 기반으로 데이터를 저장한다.
- 데이터를 정렬된 순서로 저장하며, HashSet<E>와 마찬가지로 데이터의 중복저장 않는다.
- 정렬의 기준은 프로그래머가 직접 정의한다.

```
public static void main(String[] args)
{
    TreeSet<Integer> sTree=new TreeSet<Integer>();
    sTree.add(1);
    sTree.add(2);
    sTree.add(4);
    sTree.add(3);
    sTree.add(2);

    System.out.println("저장된 데이터 수 : "+sTree.size());

    Iterator<Integer> itr=sTree.iterator();
    while(itr.hasNext())
        System.out.println(itr.next());
}
```

데이터는 정렬되어 저장이 되며, 때문에 iterator 메소드의 호출로 생성된 반복자는 오름차순의 데이터 참조를 진행한다.

실행결과

```
저장된 데이터 수 : 4
1
2
3
4
```

■ TreeSet 사용

```
public class TreeSetExam {
    public static void main(String[] args) {
        TreeSet<String> ts = new TreeSet<String>();

        ts.add("1"); ts.add("2"); ts.add("3");
        ts.add("B"); ts.add("A"); ts.add("C");
        ts.add("나"); ts.add("가"); ts.add("다");

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

■ Iterator<E>

Iterator<E> 인터페이스

- Collection<E> 인터페이스에는 iterator라는 이름의 메소드가 다음의 형태로 정의
  ✈ Iterator<E> iterator( ) { . . . . }
- iterator 메소드가 반환하는 참조 값의 인스턴스는 Iterator<E> 인터페이스를 구현하고 있다.
- iterator 메소드가 반환한 참조 값의 인스턴스를 이용하면, 컬렉션 인스턴스에 저장된 인스턴스의 순차적 접근이 가능함.
- iterator 메소드의 반환형이 Iterator<E>이니, 반환된 참조 값을 이용해서 Iterator<E>에 선언된 함수들만 호출하면 된다.

Iterator<E> 인터페이스에 정의된 메소드

• boolean hasNext( )	참조할 다음 번 요소(element)가 존재하면 true를 반환
• E next()	다음 번 요소를 반환
• void remove()	현재 위치의 요소를 삭제

■ Iterator<E> 사용 예

```
public static void main(String[] args)
{
    LinkedList<String> list=new LinkedList<String>();
    list.add("First");
    list.add("Second");
    list.add("Third");
    list.add("Fourth");

    Iterator<String> itr=list.iterator();

    System.out.println("반복자를 이용한 1차 출력과 \"Third\" 삭제");
    while(itr.hasNext())
    {
        String curStr=itr.next();
        System.out.println(curStr);
        if(curStr.compareTo("Third")==0)
            itr.remove();
    }

    System.out.println("\n\"Third\" 삭제 후 반복자
    itr=list.iterator();
    while(itr.hasNext())
        System.out.println(itr.next());
}
```

iterator 메소드가 생성하는 인스턴스를 가리켜 '반복자'라 한다.

실행결과

```
반복자를 이용한 1차 출력과 "Third" 삭제
First
Second
Third
Fourth

"Third" 삭제 후 반복자를 이용한 2차 출력
First
Second
Fourth
```

■ Iterator<E> 사용 – 1 (1 / 2)

```
public class IteratorExam {
    public static void main(String[] args) {
        ArrayList<String> aList =
                new ArrayList<String>();
        aList.add("1");
        aList.add("2");
        for (int i = 0; i < aList.size(); i++) {
            System.out.println(aList.get(i));
        }

        LinkedList<String> lList =
                new LinkedList<String>();
        lList.add("A");
        lList.add("B");
        for (int i = 0; i < lList.size(); i++) {
            System.out.println(lList.get(i));
        }
        HashSet<String> hSet =
                new HashSet<String>();
        hSet.add("가");
        hSet.add("나");
        for (int i = 0; i < hSet.size(); i++) {
            System.out.println(hSet.get(i));
        }
    }
}
```

■ Iterator<E> 사용 – 1 (2 / 2)

```
public class IteratorExam {
    public static void main(String[] args) {
        ArrayList<String> aList =
                new ArrayList<String>();
        aList.add("1");
        aList.add("2");
        for (int i = 0; i < aList.size(); i++) {
            System.out.println(aList.get(i));
        }

        LinkedList<String> lList =
                new LinkedList<String>();
        lList.add("A");
        lList.add("B");
        for (int i = 0; i < lList.size(); i++) {
            System.out.println(lList.get(i));
        }
        HashSet<String> hSet =
                new HashSet<String>();
        hSet.add("가");
        hSet.add("나");
        Iterator<String> iter = hSet.iterator();
        while(iter.hasNext()) {
            String item = iter.next();
            System.out.println(item);
        }
    }
}
```

■ Iterator 를 사용하는 이유

- 반복자를 사용하면, 컬렉션 클래스의 종류에 상관없이 **동일한 형태의 데이터 참조방식을 유지**할 수 있다.
- 따라서 컬렉션 클래스의 교체에 큰 영향이 없다.
- 컬렉션 클래스 별 데이터 참조방식을 별도로 확인할 필요가 없다.

```
public static void main(String[] args)
{
    LinkedList<String> list=new LinkedList<String>();
    list.add("First");
    list.add("Second");
    list.add("Third");
    list.add("Fourth");

    Iterator<String> itr=list.iterator();

    System.out.println("반복자를 이용한 1차 출력과 \"Third\" 삭제");
    while(itr.hasNext())
    {
        String curStr=itr.next();
        System.out.println(curStr);
        if(curStr.compareTo("Third")==0)
            itr.remove();
    }

    System.out.println("\n\"Third\" 삭제 후 반복자를 이용한 2차 출력 ");
    itr=list.iterator();
    while(itr.hasNext())
        System.out.println(itr.next());
}
```

왼편은 앞서 소개한 예제이다. 그런데, 이 예제는 반복자를 사용했기 때문에, LinkedList<E>가 어울리지 않아서, 컬렉션 클래스를 HashSet<E>로 변경해야 할 때, 다음과 같이 변경이 매우 용이하다.

변경의 전부!

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

```
HashSet<String> set
    =new HashSet<String>( );
```

■ Iterator<E> 사용 – 2

```
public class IteratorExam2 {
    public static void main(String[] args) {
        ArrayList<String> aList =
                new ArrayList<String>();
        aList.add("1");
        aList.add("2");
        showData(aList);

        LinkedList<String> lList =
                new LinkedList<String>();
        lList.add("A");
        lList.add("B");
        showData(lList);

        HashSet<String> hSet =
                new HashSet<String>();
        hSet.add("가");
        hSet.add("나");
        showData(hSet);
    }

    public static void showData(
                Collection<String> c) {
        Iterator<String> iter = c.iterator();
        while(iter.hasNext()) {
            String item = iter.next();
            System.out.println(item);
        }
    }
}
```

■ Map<K, V> 인터페이스, HashMap<K, V> 클래스

- Map<K, V> 인터페이스를 구현하는 컬렉션 클래스는 key-value 방식의 데이터 저장을 한다.
- value는 저장할 데이터를 의미하고, key는 value를 찾는 열쇠를 의미한다.
- Map<K, V>를 구현하는 대표적인 클래스로는 HashMap<K, V>와 TreeMap<K, V>가 있다.
- TreeMap<K, V>는 정렬된 형태로 데이터가 저장된다.

```
public static void main(String[] args)
{
    HashMap<Integer, String> hMap=new HashMap<Integer, String>();
    hMap.put(new Integer(3), "나삼번");
    hMap.put(5, "윤오번");
    hMap.put(8, "박팔번");
    System.out.println("6학년 3반 8번 학생 : "+hMap.get(new Integer(8)));
    System.out.println("6학년 3반 5번 학생 : "+hMap.get(5));
    System.out.println("6학년 3반 3번 학생 : "+hMap.get(3));

    hMap.remove(5);     // 5번 학생 전학 감
    System.out.println("6학년 3반 5번 학생 : "+hMap.get(5));
}
```

실행 결과

```
6학년 3반 8번 학생 : 박팔번
6학년 3반 5번 학생 : 윤오번
6학년 3반 3번 학생 : 나삼번
6학년 3반 5번 학생 : null
```

■ HashMap 사용

```
public class HashMapExam {
    public static void main(String[] args) {
        Map<String, String> map =
                new HashMap<String, String>();
        map.put("111", "aaa");
        map.put("112", "aab");
        map.put("113", "aac");
        map.put("114", "aad");

        Set<String> set = map.keySet();
        Iterator<String> iter = set.iterator();
        while(iter.hasNext()) {
            String key = iter.next();
            String value = map.get(key);
            System.out.println(key + " : " + value);
        }
        for(Map.Entry<String, String> s :
                map.entrySet()) {
            String key = s.getKey();
            String value = s.getValue();
            System.out.println(key + " : " + value);
        }
    }
}
```

■ 단어 개수 세기 (1 / 2)

```
public class WordCount {
    public static void main(String[] args) {
        StringBuffer text = new StringBuffer();
        text.append("Python is an easy to learn, powerful programming language. ");
        text.append("It has efficient high-level data structures and a simple ");
        text.append("but effective approach to object-oriented programming. ");
        text.append("Python's elegant syntax and dynamic typing, ");
        text.append("together with its interpreted nature, ");
        text.append("make it an ideal language for scripting and ");
        text.append("rapid application development in many areas on most platforms.");

        String[] words = text.toString().split(" ");

        Map<String, Integer> wordMap = new HashMap<String, Integer>();
        for(String word : words) {
            boolean isContain = wordMap.containsKey(word);
            int count = 1;
            if(isContain) {
                count = wordMap.get(word);
                wordMap.put(word, ++count);
            }
            wordMap.put(word, count);
        }
```

■ 단어 개수 세기 (2 / 2)

```
        Set<String> keys = wordMap.keySet();
        Iterator<String> iter = keys.iterator();
        while(iter.hasNext()) {
            String key = iter.next();
            int value = wordMap.get(key);
            System.out.println(key + " :: " + value);
        }
    }
}
```