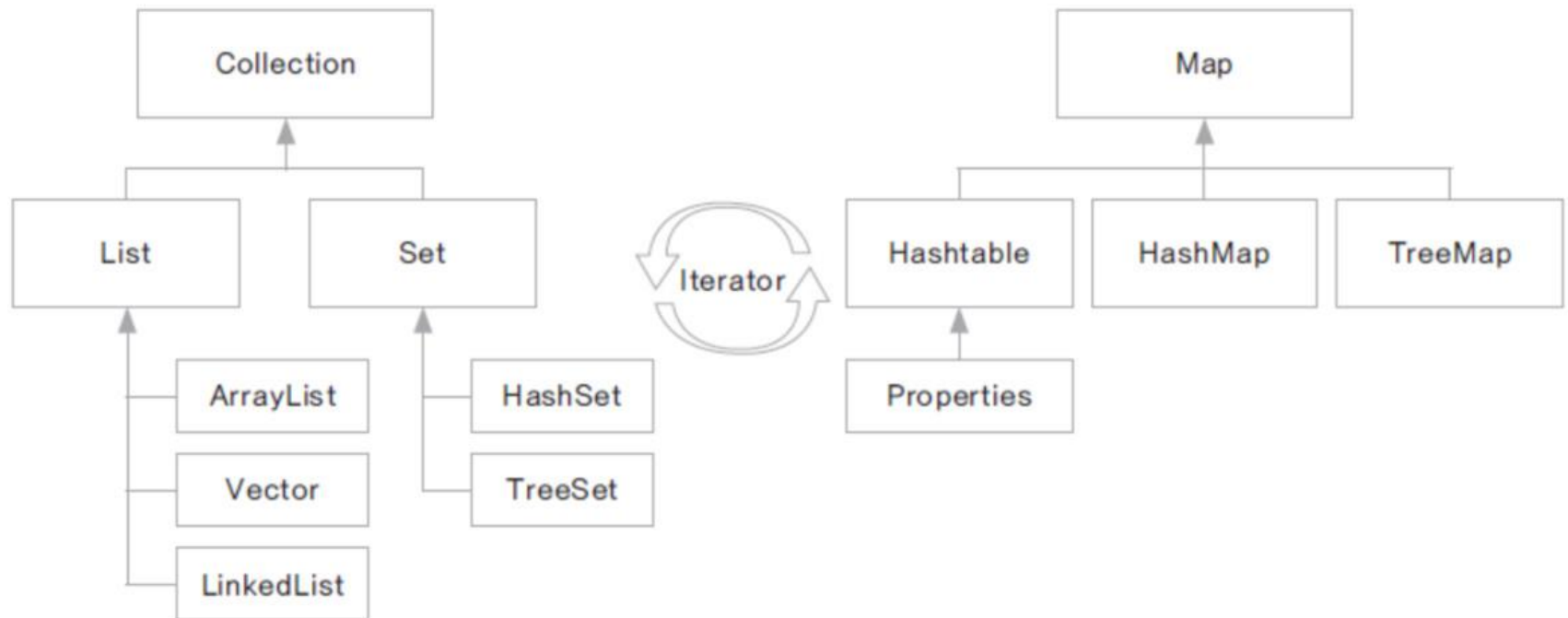


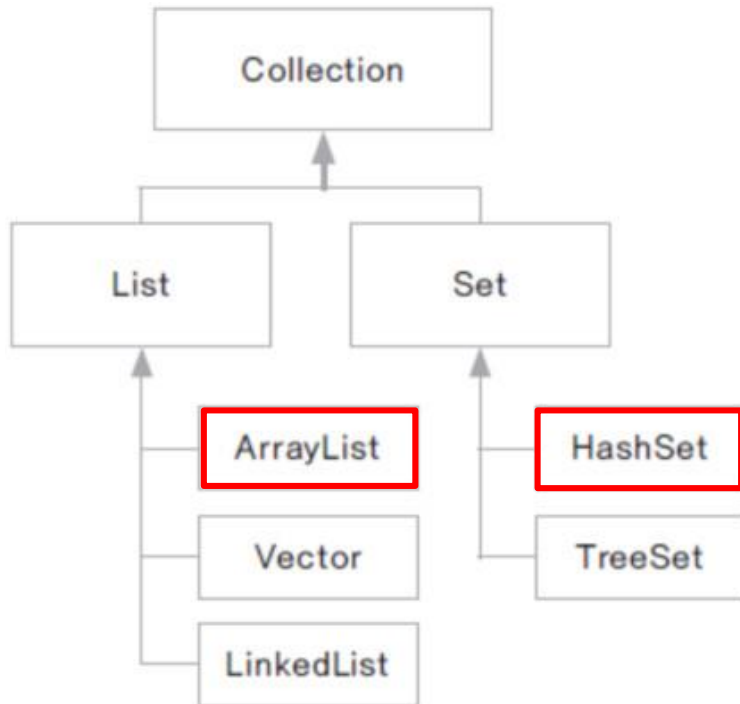
■ 컬렉션 프레임워크

- 프로그램 구현에 필요한 자료구조를 구현해 놓은 라이브러리
- java.util 패키지에 구현되어 있음
- 개발에 소요되는 시간을 절약하면서 최적화 된 알고리즘 사용 가능
- 여러 인터페이스와 구현 클래스의 사용 방법을 이해해야 함



■ 컬렉션 프레임워크

- List / Set 인터페이스
- 하위 여러 클래스들이 구현되어 있음



분류	설명
List 인터페이스	순서가 있는 자료 관리, 중복 허용. 이 인터페이스를 구현한 클래스는 ArrayList, Vector, LinkedList, Stack, Queue 등이 있음
Set 인터페이스	순서가 정해져 있지 않음, 중복을 허용하지 않음. 이 인터페이스를 구현한 클래스는 HashSet, TreeSet 등이 있음

■ 컬렉션 프레임워크

- Map 인터페이스
- 쌍으로 이루어진 객체를 관리하는데 사용
- key - value 쌍으로 이루어짐
- key는 중복 될 수 없음



메서드	설명
V put(K key, V value)	key에 해당하는 value 값을 map에 넣습니다.
V get(K key)	key에 해당하는 value 값을 반환합니다.
boolean isEmpty()	Map이 비었는지 여부를 반환합니다.
boolean containsKey(Object key)	Map에 해당 key가 있는지 여부를 반환합니다.
boolean containsValue(Object value)	Map에 해당 value가 있는지 여부를 반환합니다.
Set keyset()	key 집합을 Set로 반환합니다(중복 안 되므로 Set).
Collection values()	value를 Collection으로 반환합니다(중복 무관).
V remove(key)	key가 있는 경우 삭제합니다.
boolean remove(Object key, Object value)	key가 있는 경우 key에 해당하는 value가 매개변수와 일치할 때 삭제합니다.

■ List 인터페이스

- 객체를 순서에 따라 저장하고 관리
- 배열의 불편한 부분을 개선
 - 배열 생성 후 크기 변경 불가
 - 중간의 데이터를 삭제해도 요소가 존재하는 것으로 처리

배열

0	1	2	3	4	5	6	7	8	9
●	●	×	●	×	●	×	●	●	×

● 자주 사용되는 메서드

메서드	설명
<code>boolean add(E e)</code>	Collection에 객체를 추가합니다.
<code>void clear()</code>	Collection의 모든 객체를 제거합니다.
<code>Iterator<E> iterator</code>	Collection을 순환할 반복자(Iterator)를 반환합니다.
<code>boolean remove(Object o)</code>	Collection에 매개변수에 해당하는 인스턴스가 존재하면 제거합니다.
<code>int size()</code>	Collection에 있는 요소의 개수를 반환합니다.

■ ArrayList / Vector

- 일반적으로 ArrayList가 가장 자주 사용
- 멀티 스레드 상태에서 동기화가 필요한 경우 Vector 사용

※ 동기화(Synchronization)

두 개 이상으로 병렬 처리(쓰레드)로 하나의 자원(변수/메소드)에 접근하여 작업 할 때 데이터 오류가 발생하지 않도록 하는 기능

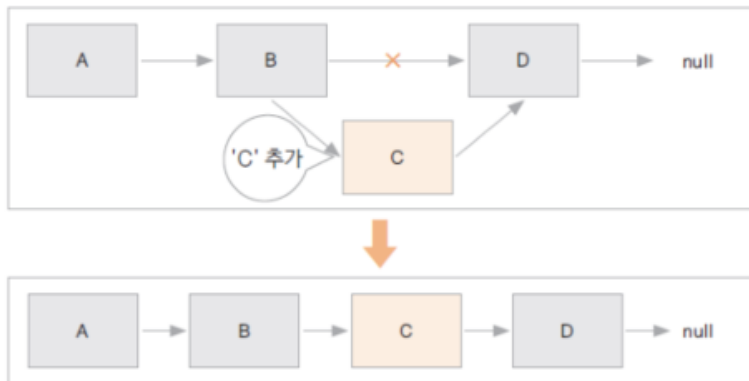
■ LinkedList

- 논리적으로 순차적인 자료구조가 구현된 클래스
- 다음 요소에 대한 참조값을 가지고 있음

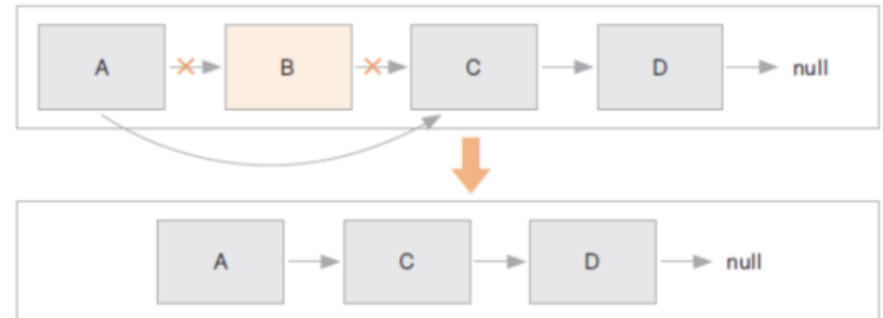


- 요소의 추가 / 삭제에 드는 비용이 배열보다 적음

- 추가



- 삭제

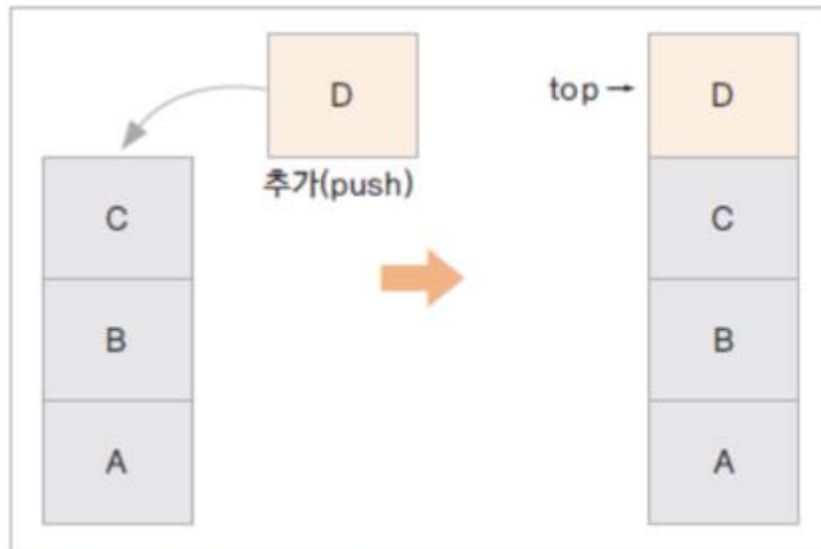


■ Stack / Queue

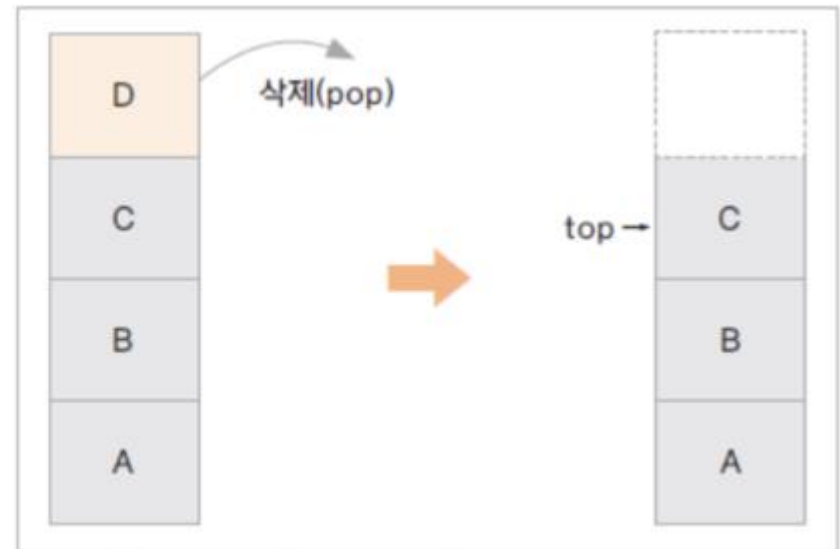
● Stack : Last In First Out (LIFO)

- 마지막에 추가된 요소가 가장 먼저 꺼내지는 자료 구조

게임의 undo, 앱 사용 중 back키를 이용한 이전 화면으로 전환 등



스택에 요소 추가(push)하기

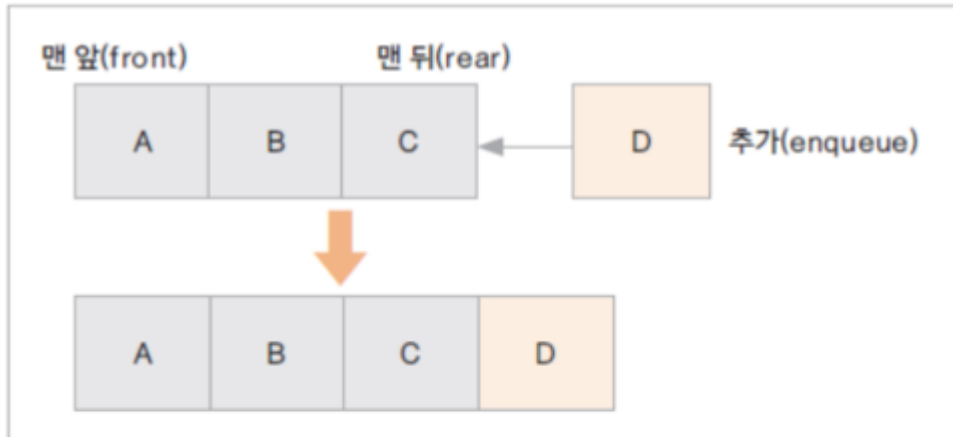


스택에서 요소 꺼내어(pop) 삭제하기

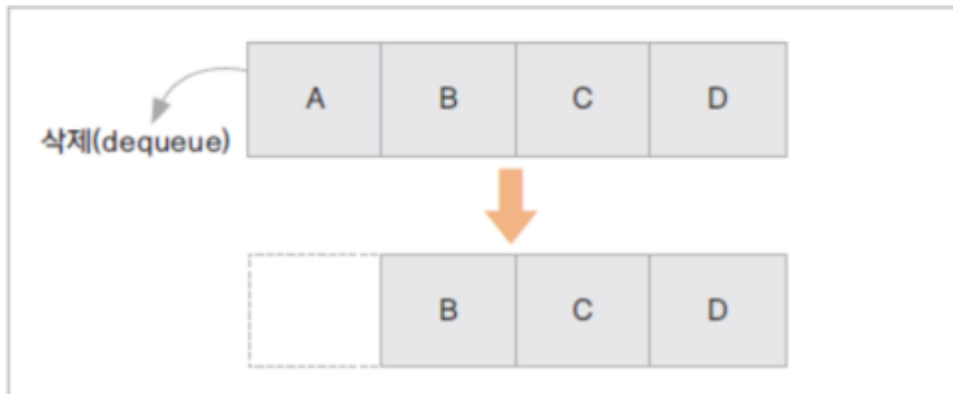
■ Stack / Queue

● Queue : First In First Out (FIFO)

- 먼저 저장된 자료가 먼저 꺼내지는 대기열 자료 구조



큐에서 요소 추가(enqueue)하기



큐에서 요소 삭제(dequeue)하기

■ ArrayList 사용

```
import java.util.ArrayList;

public class ArrayListExam {
    public static void main(String[] args) {
        // arrayList 이름을 가진 ArrayList 생성
        ArrayList<String> arrayList = new ArrayList<String>();

        // ArrayList 에 element 추가
        arrayList.add("1번");
        arrayList.add("2번");
        arrayList.add("3번");

        // ArrayList 에 element 추가
        arrayList.add("=> 4번");
        arrayList.add("=> 5번");

        for (int index = 0; index < arrayList.size(); index++) {
            System.out.println(arrayList.get(index));
        }
    }
}
```

1번
2번
3번
=> 4번
=> 5번

■ ArrayList 사용

// ArrayList의 4번째와 5번째 element를 삭제

```
arrayList.remove(4);
```

```
arrayList.remove(3);
```

// ArrayList 에 index를 부여한 element 추가

```
arrayList.add(0, "> 4번");
```

```
arrayList.add(1, "> 5번");
```

=> 4번

=> 5번

1번

2번

3번

```
for (int index = 0; index < arrayList.size(); index++) {
```

```
    System.out.println(arrayList.get(index));
```

```
}
```

```
}
```

```
}
```

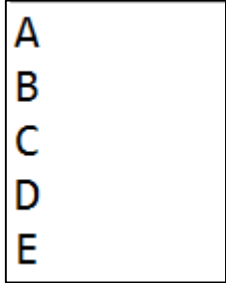
■ LinkedList 사용

```
import java.util.LinkedList;

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

        linkedList.add("A");
        linkedList.add("B");
        linkedList.add("C");
        linkedList.add("D");
        linkedList.add("E");

        for (int index = 0; index < linkedList.size(); index++) {
            System.out.println(linkedList.get(index));
        }
    }
}
```



A
B
C
D
E

■ LinkedList 사용

```
linkedList.remove(4);  
linkedList.remove(3);
```

```
linkedList.add(0, "=> D");  
linkedList.add(1, "=> E");
```

```
=> D  
=> E  
A  
B  
C
```

```
for (int index = 0; index < linkedList.size(); index++) {  
    System.out.println(linkedList.get(index));  
}  
}  
}
```

■ 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 Wrapper 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

실행 결과

Auto Boxing과 Auto Unboxing
의 도움으로 정수 단위의 데이터
입출력이 매우 자연스럽다!

■ ArrayList 와 LinkedList 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 |

■ Iterator

- Collection의 개체를 순회하는 인터페이스
- iterator() 호출

```
Iterator ir = memberArrayList.iterator( );
```

- 자주 사용되는 메서드

메서드	설명
boolean hasNext()	이후에 요소가 더 있는지를 체크하는 메서드이며, 요소가 있다면 true를 반환합니다.
E next()	다음에 있는 요소를 반환합니다.

■ Iterator

```
public boolean removeMember(int memberId) {  
    Iterator<Member> ir = arrayList.iterator( );  
    while(ir.hasNext( )) {  
        Member member = ir.next( );  
        int tempId = member.getMemberId( );  
        if(tempId == memberId) {  
            arrayList.remove(member);  
            return true;  
        }  
    }  
    //끝날 때까지 삭제하려는 값을 찾지 못한 경우  
    System.out.println(memberId + "가 존재하지 않습니다");  
    return false;  
}
```

// Iterator 반환
// 요소가 있는 동안
// 다음 회원을 반환받음
// 회원 아이디가 매개변수와 일치하면
// 해당 회원 삭제
// true 반환

■ Iterator

```
import java.util.ArrayList;
import java.util.Iterator;

public class IteratorExam {
    public static void main(String[] args) {
        ArrayList<String> data = new ArrayList<String>();

        data.add("1번");
        data.add("2번");
        data.add("3번");

        Iterator<String> iter = data.iterator();
        while(iter.hasNext()) {
            String item = iter.next();
            System.out.println(item);
        }
    }
}
```

1번
2번
3번

■ Set 인터페이스

- 중복을 허용하지 않음
- 아이디, 주민등록번호, 사번, 학번 등 유일한 값이나 객체를 관리할 때 사용
- List와 달리 순서(index)가 없음
- Set에 저장되는 순서와 출력 순서가 다를 수 있음
- get 메서드가 제공되지 않으므로 요소를 확인하기 위해서는 Iterator 사용

■ HashSet

- Set 인터페이스를 구현한 클래스
- 중복을 허용하지 않음
- 순서가 없음

```
import java.util.HashSet;

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

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

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

[1, A, B, 2, 3, C]

■ TreeSet

- Set 인터페이스를 구현한 클래스
- 중복을 허용하지 않음
- 순서가 있음 (오름차순 또는 내림차순)

```
import java.util.TreeSet;

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

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

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

[1, 2, 3, A, B, C]

■ Set - Iterator

```
import java.util.HashSet;
import java.util.Iterator;

public class HashSetIteratorExam {
    public static void main(String[] args) {
        HashSet<String> data = new HashSet<String>();

        data.add("1번");
        data.add("2번");
        data.add("3번");

        Iterator<String> iter = data.iterator();
        while(iter.hasNext()) {
            String item = iter.next();
            System.out.println(item);
        }
    }
}
```

3번
2번
1번

■ Map 인터페이스

- key – value 쌍으로 이루어짐
- key는 중복 될 수 없음
- 순서가 없음
- 검색을 위한 자료 구조
- Map 내의 모든 요소를 확인하기 위해서는 key의 집합을 만든 후 출력

■ HashMap

- Map 인터페이스를 구현한 클래스

```
import java.util.HashMap;

public class HashMapExam {
    public static void main(String[] args) {
        HashMap<String, String> map = new HashMap<String, String>();
        map.put("A", "1");
        map.put("B", "2");
        map.put("C", "3");
        map.put("D", "4");
        System.out.println(map);

        map.put("A", "10");
        System.out.println(map);
    }
}
```

[1, A, B, 2, 3, C]

[1, 2, 3, A, B, C]

■ HashMap

● 단어 세기

```
import java.util.HashMap;
import java.util.Iterator;
import java.util.Set;

public class WordCount {
    public static void main(String[] args) {
        StringBuffer text = new StringBuffer();
        text.append("안축축한 초코칩 나라에 살던 안축축한 초코칩이 ");
        text.append("축축한 초코칩 나라의 축축한 초코칩을 보고 ");
        text.append("축축한 초코칩이 되고 싶어서 축축한 초코칩 나라에 갔는데 ");
        text.append("축축한 초코칩 나라의 축축한 초코칩 문지기가 ");
        text.append("넌 축축한 초코칩이 아니고 안축축한 초코칩이니까 ");
        text.append("안축축한 초코칩나라에서 살아 라고해서 ");
        text.append("안축축한 초코칩은 축축한 초코칩이 되는 것을 포기하고 ");
        text.append("안축축한 초코칩 나라로 돌아갔다. ");

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

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


■ HashMap

● 단어 세기

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

```
나라에 - 2  
초코칩 - 6  
라고해서 - 1  
살아 - 1  
년 - 1  
초코칩을 - 1  
초코칩은 - 1  
나라로 - 1  
돌아갔다. - 1  
나라의 - 2  
되고 - 1  
초코칩이 - 4  
아니고 - 1  
촉촉한 - 8  
초코칩나라에서 - 1  
싫어서 - 1  
되는 - 1  
갔는데 - 1  
초코칩이니까 - 1  
포기하고 - 1  
것을 - 1  
보고 - 1  
살던 - 1  
안촉촉한 - 6  
문지기가 - 1
```