```java
interface LightControl{
    public void turn_on();
    public void turn_off();
    public void blink_on();
}
class Lamp implements LightControl{
    // implements하면 LightControl class의 객체화가 필요 없음.
    // 아래 led나 street lamp는 class 내부에 인터페이스에 대한 객체를 생성한 것에 반면
    // lamp class에서는  implements를 사용하여 해당 class에서
    // interface 내부의 미구현 method를 구현해줌으로서 동작을 하게 한다.
        @Override
        public void turn_on() { System.out.println("<램프>에 전기신호를 일정하게 주고 지속적으로 켜져있게 한다"); }
        @Override
        public void turn_off() { System.out.println("<램프>에 전기신호를 차단한다"); }
        @Override
        public void blink_on() { System.out.println("<램프>에 전기신호를 일정한 주기로 주었다가 껐다를 반복하여 깜빡이게 한다."); }
    }
class Led{
    LightControl led_control = new LightControl() {
        @Override
        public void turn_on() { System.out.println("<led>에 전기신호를 일정하게 주고 지속적으로 켜져있게 한다"); }
        @Override
        public void turn_off() { System.out.println("<led>에 전기신호를 차단한다"); }
        @Override
        public void blink_on() { System.out.println("<led>에 전기신호를 일정한 주기로 주었다가 껐다를 반복하여 깜빡이게 한다."); }
    };
}
```

2021.06.02

Java

〈Interface_implements〉

Q. interface 자체가 추상 메서드이기 때문에 abstract class에 대해서 따로 알 필요가 없는지..?

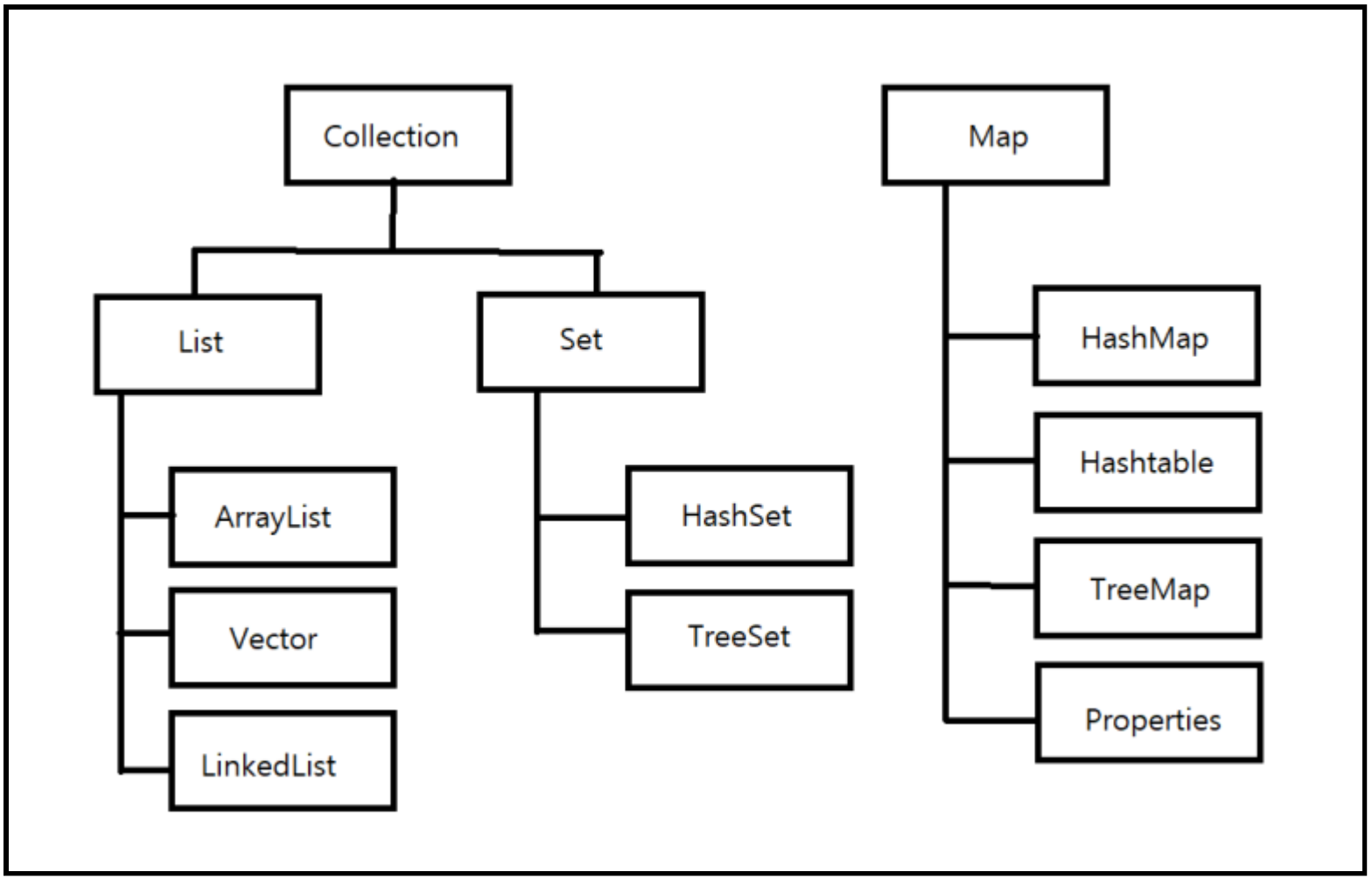
```java
class LampMethod2 {
    public void lightOn() {
        System.out.println("상속을 사용했을 때");
    }
    public void lightOff() {
        System.out.println("상위 클래스에 종속된다.");
    }
}
class Lamp2 extends LampMethod2 {

}
class StreetLamp2 extends LampMethod2 {

}
class Led2 extends LampMethod2 {

}
public class _6h_InterfaceVersusExtends{
    public static void main(String[] args) {
        Lamp2 lamp = new Lamp2();
        lamp.lightOn();
        lamp.lightOff();

        StreetLamp2 streetLamp = new StreetLamp2();
        streetLamp.lightOn();
        streetLamp.lightOff();
```

```
"C:\Program Files\Ja
상속을 사용했을 때
상위 클래스에 종속된다.
상속을 사용했을 때
상위 클래스에 종속된다.
상속을 사용했을 때
상위 클래스에 종속된다.
```

위에서 class와 class를 interface로 연결한 것과
지금처럼 class들의 상속관계일 때와 비교할 때의 장점은
class마다 독립적인 프로그래밍이 가능하다는 것.

interface의 경우에는
interface에 미구현된 method를 각각의 class가 독립적으로
method를 구현하는데 반하여

상속(extends)의 경우에는 상위 class에 종속.



〈Collection / Map 프레임워크〉

여기서 짚고 넘어갈 한 가지 개념.

》 ArrayList / Vector / LinkedList는 List 인터페이스의 구현클래스.

》 HashSet / TreeSet도 마찬가지 개념으로 Set의 구현클래스.

List Collection은 저장순서를 유지

Set Collection은 저장순서를 유지하지 않지만 객체의 중복을 허용하지 않는다.

))참고로 Set은 인덱스로 관리하지 않기 때문에 인덱스를 매개값으로 가지는 메소드가 없다.

기능	메소드	설명
객체 추가	boolead add(E e)	주어진 객체를 저장, 객체가 성공적으로 저장되면 true를 반환하고 중복되는 객체면 false를 리턴
객체 검색	boolean contains(Object o)	주어진 객체가 저장되어있는지 여부
	boolean isEmpty()	컬렉션이 비어 있는지 조사
	Iterator<E> iterator()	저장된 객체를 한 번씩 가져오는 반복자 리턴
	int size()	저장되어 있는 전체 객체 수 리턴
객체 삭제	void clear()	저장된 모든 객체를 삭제
	boolean remove(Object o)	주어진 객체를 삭제

<Set Collection에 공통적으로 사용 가능한 Set Interface method>

- **retainAll = 교집합 개념**
- **addAll = 합집합 개념**
- **removeAll = 차집합 개념**
- **….. 등**

```java
import java.util.HashSet;
public class _2nd_HashSetEdu {
    public static void main(String[] args) {
        HashSet<String> food = new HashSet<String>();

        food.add("우유");
        food.add("빵");
        food.add("베이컨");
        food.add("Ham");
        food.add("소시지");
        food.add("파스타");
        food.add("Ham"); //  Ham 두개 써도 하나만 add 됨.
        food.add("ham");
        //...
        System.out.println(food);
    }
```

```
_2nd_HashSetEdu
"C:\Program Files\Java\jdk-16\bin\java.ex
[빵, Ham, ham, 우유, 파스타, 베이컨, 소시지]

Process finished with exit code 0
```

〈HashSet〉

- javaLI에서 존재하는 collection 중 가장 속도가 빠르다.
- 집합의(set)의 특성을 가지고 있어 중복을 허용하지 않는다. ex) Ham
- 일종의 배열. ArrayList와 비슷함.
- 단 순서는 HashSet 자체의 알고리즘대로 순서가 정해짐. 〉 순서가 중요한 알고리즘에는 사용하지 말 것. 순서가 중요하다면 ArrayList 사용
- HashSet으로 만들어진 배열을 ArrayList로 만들 수는 있지만 순서 보장x

```java
import java.util.HashSet;
import java.util.Set;

public class _2nd_HashSetEdu {
    public static void main(String[] args) {
        Set<String> food = new HashSet<String>();

        food.add("우유");
        food.add("빵");
```

Q. HashSet은 Set의 구현클래스이기 때문에
이런식으로도 사용해도 문제가 없을까요..?

〈HashSet '중복허용x'의 활용〉

```java
import java.util.HashSet;
import java.util.Set;

public class _4th_HowToUseHashSet {
    public static void main(String[] args) {
        //중복을 허용하지 않는 HashSet 특성 사용의 예
        HashSet<String> sayhello = new HashSet<>();

        String[] sample = { "안녕", "hi", "헬로", "안녕", "안녕", "hi" };

        for (String str : sample){
            if(!sayhello.add(str)){
                // Returns: true if this set contained the specified element << ctrl + b로 ".add" 확인
                // add 할 때 이미 집합 내용에 포함되어 있지 않으면 true > !니까 반대
                System.out.println("중복되었습니다: " + str);
            }
        }
        System.out.println("중복을 제외한 단어들: " + sayhello );
    }
```

```
"C:\Program Files\Java\jdk-16\bin\j
중복되었습니다: 안녕
중복되었습니다: 안녕
중복되었습니다: hi
중복을 제외한 단어들: [hi, 안녕, 헬로]
```

```java
public boolean add(E e) {
    return map.put(e, PRESENT)==null;
}
```

Removes the specified element from this set if it is present. More formally, removes an element e such that `Objects.equals(o, e)`, if this set contains such an element. Returns `true` if this set contained the element (or equivalently, if this set changed as a result of the call). (This set will not contain the element once the call returns.)

Params: o – object to be removed from this set, if present

Returns: `true` if the set contained the specified element

만약 add 하려고 하는 element가
이미 set에 있다면 remove(중복의 제거).
그 set이 element를 contain한다면(성공적으로
add 됐다면으로 의역하면 될 듯) return true.

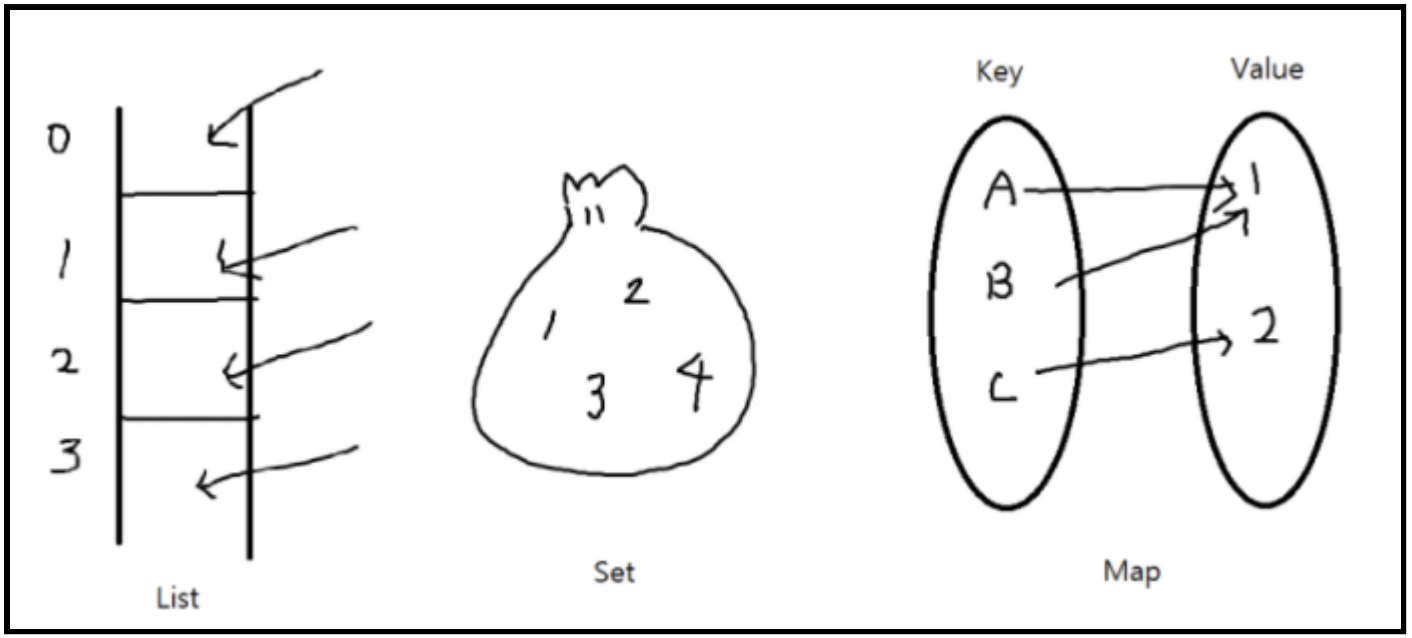
```java
import java.util.HashSet;
import java.util.Set;
public class _5th_SetFeatureEduWithHashSet {
    public static void main(String[] args) {
        Set<String> s1 = new HashSet<>();
        Set<String> s2 = new HashSet<>();

        s1.add("Apple");
        s1.add("Tesla");
        s1.add("Microsoft");

        s2.add("Tesla");
        s2.add("Alphabet");
        s2.add("Texas Instruments");

        Set<String> union_s1_s2 = new HashSet<>(s1);
        union_s1_s2.addAll(s2);
        System.out.println("합집합: " + union_s1_s2);

        Set<String> intersection_s1_s2 = new HashSet<>(s1);
        intersection_s1_s2.retainAll(s2);
        System.out.println("교집합: " + intersection_s1_s2);
    }
```

```
"C:\Program Files\Java\jdk-16\bin\java.exe" -javaagent:C:\User
합집합: [Alphabet, Texas Instruments, Apple, Tesla, Microsoft]
교집합: [Tesla]
```

〈HashSet의 method (= Set의 method)〉

addAll: 합집합 개념

retainAll: 교집합 개념



〈Map 컬렉션〉

Map 컬렉션은 키(Key)와 값(Value)로 구성된 Entry 객체를 저장하는 구조를 가지고 있다.

Key와 Value는 모두 객체

Key는 중복저장 불가능 / Value는 중복저장 가능

기존에 있던 Key와 동일한 Key가 등록이 된다면 기존의 Key는 사라지고 새로운 Key가 등록된다.

기능	메소드	설명
객체 추가	V put(K key, V value)	주어진 키로 값을 저장, 새로운 키일 경우 null을 리턴하고 동일한 키가 있을 경우 대체하고 이전 값을 리턴
객체 검색	boolean containsKey(Object o)	주어진 키가 있는지 여부
	boolean containsValue(Object value)	주어진 값이 있는지 여부
	Sat<Map.Entry<K,V>>enrtySet()	키와 값의 쌍으로 구성된 모든 Map.Entry 객체를 set에 담아서 리턴
	V get(Object Key)	주어진 키가 있는 값을 리턴
	boolean isEmpty()	컬렉션이 비어 있는지 여부
	Set<K> KeySet()	모든 키를 Set 객체에 담아서 리턴
	int size()	저장된 키의 총 수를 리턴
	Collection<V> values()	자징된 모든 값을 Collection에 담아서 리턴
객체 삭제	void clear()	주어진 Map.Enrty(키와 값)을 삭제
	V remove(Object key)	주어진 키와 일치하는 Map.Entry를 삭제하고 값을 리턴

<Map컬렉션에서 공통으로 사용 가능한 Map Interface Method>

리턴타입에 K와 V라는 파라미터가 있는데 Map인터페이스가 제네릭 타입이기 때문.

〈HashMap〉

```java
import java.util.HashMap;
import java.util.Map;
class Student {
    int age;
    String name;

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

    @Override
    public String toString() {
        return "Student{" +
                "age=" + age +
                ", name='" + name + '\'' +
                '}';
    }
```

```java
public class _7th_HashMapEdu {
    public static void main(String[] args) {
        // Map의 특성중 하나가 key와 value가 분리됨
        // Map<Key, Value> > 특별히 특정 데이터타입을 지켜줄 필요가 없다.
        //                 > 배열, ArrayList, Integer, String.. 다 가능.
        Map<Integer, Student> st = new HashMap<Integer, Student>();
        // 앞에 오는 숫자는 인덱스가 아니다. 단지 사물함을 여는데 필요한 열쇠일 뿐.
        st.put(7, new Student( age: 42,  name: "Bob"));
        st.put(2, new Student( age: 33,  name: "Chris"));
        st.put(44, new Student( age: 27,  name: "Denis"));
        st.put(3, new Student( age: 29,  name: "David"));
        System.out.println(st);

        st.remove( key: 2);
        System.out.println(st); // Christ 사라짐

        st.put(3, new Student( age: 77,  name: "Jesica"));
        System.out.println(st); // David의 Jessica로 대체

        for (Map.Entry<Integer, Student> s : st.entrySet()) {
            Integer key = s.getKey();
            Student value = s.getValue();
            System.out.println("key = " + key + ", value = " + value);
        }
        System.out.println(st.entrySet());
        System.out.println(st);
```

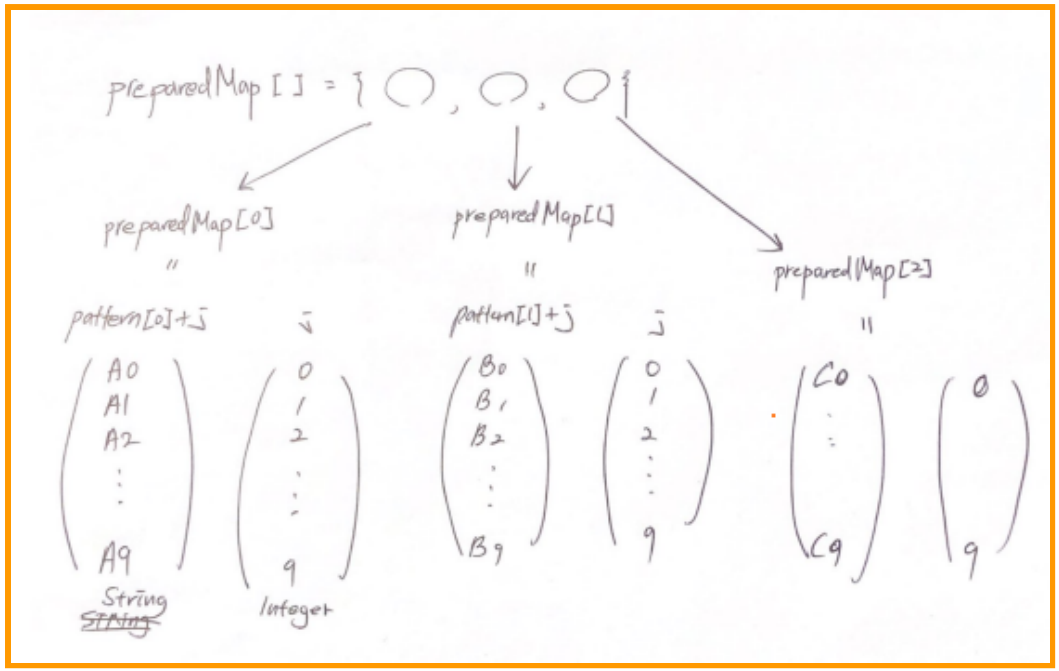
Q. print(st.entrySet()); 과 print(st)는 차이가 없는게 맞는지..? 아니면 개념적인 부분에서 차이가 있는지..?

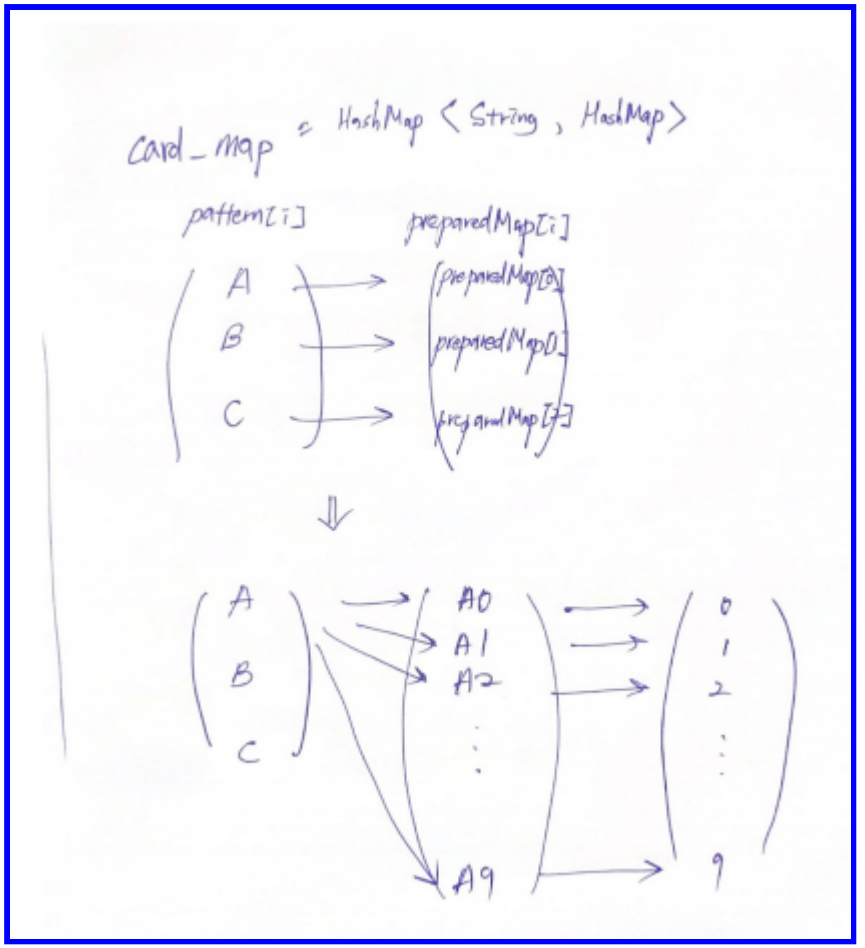
```java
Map<String, String> strMap = new HashMap<String, String>();
strMap.put("열쇠", "아무거나 다 된다");
// 추상화의 연장선 관점에서 아래 사항을 준수하여 코딩하면
// HashMap에서 어떤 상황에서든 이 방식으로 key, value값을 얻을 수 있다.
// Entry<키 데이터타입, 밸류 데이터타입> 형식은 지켜야 한다.
for (Map.Entry<String, String> map : strMap.entrySet()) {
    String key = map.getKey();
    String value = map.getValue();
    System.out.println("key = " + key + ", value = " + value);
}
```

```
"C:\Program Files\Java\jdk-16\bin\java.exe" -javaagent:C:\Users\Samuel\AppData\Local\JetBrains\Toolbox\apps\IDEA-C\ch-0\211.7142.45\
{2=Student{age=33, name='Chris'}, 3=Student{age=29, name='David'}, 7=Student{age=42, name='Bob'}, 44=Student{age=27, name='Denis'}}
{3=Student{age=29, name='David'}, 7=Student{age=42, name='Bob'}, 44=Student{age=27, name='Denis'}}
{3=Student{age=77, name='Jesica'}, 7=Student{age=42, name='Bob'}, 44=Student{age=27, name='Denis'}}
key = 3, value = Student{age=77, name='Jesica'}
key = 7, value = Student{age=42, name='Bob'}
key = 44, value = Student{age=27, name='Denis'}
[3=Student{age=77, name='Jesica'}, 7=Student{age=42, name='Bob'}, 44=Student{age=27, name='Denis'}]
{3=Student{age=77, name='Jesica'}, 7=Student{age=42, name='Bob'}, 44=Student{age=27, name='Denis'}}
key = 열쇠, value = 아무거나 다 된다
```

〈Quiz.56〉

```java
import java.util.ArrayList;
import java.util.HashMap;
import java.util.Map;
public class _8th_Quiz56 {
    public static void main(String[] args) {
        /* 숫자 0~9까지의 카드가 존재
        카드별로 문양이 있고 문양의 종류는 3가지.
        문양의 종류는 spear=A, sword=B, arrow=C
        카드는 총 30장 / 각각의 문양이 10개씩.
        사용자에게 랜덤하게 4장의 카드를 나눠준다
         */
        Map<String, Map<String, Integer>> card_map = new HashMap<String, Map<String, Integer>>();
        // <String, Map<String, Integer>>형태의 HashMap datatype인 card_map을 만들겠다.
        Map<String, Integer>[] preparedMap = new HashMap[3];
        // HashMap<String, Integer>형태의 인자 3개를 갖는 배열 preparedMap 만들겠다.

        String[] pattern = {"A", "B", "C"};

        for (int i = 0; i < 3; i++) {
            preparedMap[i] = new HashMap<String, Integer>();
            // prepareMap[0] ~ preparedMap[2]는 HashMap.
            for (int j = 0; j < 10; j++) {
                preparedMap[i].put(pattern[i] + j, j);
            }
        }

        for (int i = 0; i < 3; i++) {
            card_map.put(pattern[i], preparedMap[i]);
        }
```


preparedMap[] = { , , }
preparedMap[0]
=
pattern[0]+j
i
A0
A1
A2
A9
String
0
1
2
9
Integer
preparedMap[1]
=
pattern[1]+j
j
B0
B1
B2
B9
0
1
2
9
preparedMap[2]
=
C0
C9
0
9


card_map = HashMap<String, HashMap>
pattern[i]
preparedMap[i]
A
B
C
A0
A1
A2
A9
0
1
2
9

```java
System.out.println(card_map);
System.out.println("카드 분배");

ArrayList<Integer> number_duplicate_check = new ArrayList<Integer>();
boolean dupilcate_checking = true;
while (!(number_duplicate_check.size() == 4)) {
    String card_pattern = pattern[(int) (Math.random() * 3)];
    int randNum = (int) (Math.random() * 10);
        if (number_duplicate_check.indexOf(randNum) == -1) {
            number_duplicate_check.add(randNum);
            dupilcate_checking = false;
        } else {
            continue;
        }
        // number_duplicate_check라는 ArrayList를 만들고
        // while을 사용해서
        // card_pattern과 randNum 구하고
        // if에서 number_duplicate_check에 중복 발견시 다시 위로 continue
        // 중복 없으면 나온 randNum을 number_duplicate_check[]에 add
        // number_duplicate_check의 size가 4가 될때까지 돌린다.

        // 이런식으로 하면 같은 문양의 같은 숫자는 안 나오게 되는데
        // 대신 다른 문양이면서 같은 숫자인 것도 안 나오게 된다..
        // 문양을 넣는 ArrayList를 만들어서 분배하는 식으로 해야하나..?
    System.out.println("사용자에게 분배된 카드는 = " + card_pattern +
                " 문양의 " + randNum + " 카드입니다!");
    }
```

```
"C:\Program Files\Java\jdk-16\bin\java.exe" -javaagent:C:\Users\Samuel\AppData\Local\JetB
{A={A1=1, A2=2, A3=3, A4=4, A5=5, A6=6, A7=7, A8=8, A9=9, A0=0}, B={B2=2, B3=3, B4=4, B5=
카드 분배
사용자에게 분배된 카드는 = C 문양의 5 카드입니다!
사용자에게 분배된 카드는 = B 문양의 2 카드입니다!
사용자에게 분배된 카드는 = B 문양의 1 카드입니다!
사용자에게 분배된 카드는 = C 문양의 9 카드입니다!
```