**Map 인터페이스**

2017-10-24

이승진

**학습목표**

Map 인터페이스 사용방법을 학습한다.

equals, hashCode 메소드 재정의 방법을 학습한다.

**목차**

[1. Map 인터페이스 2](#_Toc496620820)

[1) Map<K,V> interface 2](#_Toc496620821)

[2) Map 인터페이스를 구현한 클래스 3](#_Toc496620822)

[3) Map<K,V> interface 메소드 4](#_Toc496620823)

[4) 예제1 5](#_Toc496620824)

[5) 예제2 6](#_Toc496620825)

[6) 예제3 7](#_Toc496620826)

[2. equals, hashCode 메소드 재정의 8](#_Toc496620827)

[1) equals, hashCode 메소드를 재정의 해야함 8](#_Toc496620828)

[2) Object 클래스의 eauals, hashCode 메소드의 문제점 8](#_Toc496620829)

[3) Objects 클래스 9](#_Toc496620830)

[4) 예제4 10](#_Toc496620831)

[5) 예제5 11](#_Toc496620832)

[6) 목록 출력 예제 12](#_Toc496620833)

# Map 인터페이스

java.util.Map 인터페이스



### Map interface

데이터 항목 한 개는 key와 value로 구성된다.

데이터 항목을 등록할 때는, key와 value를 같이 등록한다.

데이터 항목을 꺼낼 때는, key로 찾아서 value를 리턴한다.

## Map<K,V> interface

Map 인터페이스를 사용할 때, Map<K, V> 형태로 사용한다.

여기서 K, V 부분에 각각 실제 클래스 타입을 지정해야 한다.

K는 key의 타입이고, V는 value의 타입이다.

### 객체 생성의 예

Map<String, Person> map = new HashMap<String, Person>();

key는 String 객체이고, value는 Person 객체이다.

### 사용 예

map.put("가", new Person("홍길동")); // (key: "가", value: 홍길동 Person 객체) 등록

map.put("나", new Person("임꺽정")); // (key: "나", value: 임꺽정 Person 객체) 등록

Person p1 = map.get("가"); // (key: "가") 로 등록했던 value를 리턴한다.

// 홍길동 Person 객체 리턴됨.

Person p2 = map.get("나"); // (key: "나") 로 등록했던 value를 리턴한다.

// 임꺽정 Person 객체 리턴됨.

Person p3 = map.get("다"); // (key: "다") 로 등록했던 value를 리턴한다.

// 등록되지 않았으므로 null 리턴됨.

### 치환

동일한 key로 다른 value를 등록하면 기존의 value의 치환된다.

사용 예:

map.put("가", new Person("홍길동")); // (key: "가", value: 홍길동 Person 객체) 등록

map.put("가", new Person("임꺽정")); // (key: "가", value: 임꺽정 Person 객체) 등록된다.

Person p1 = map.get("가"); // (key: "가") 로 등록했던 value를 리턴한다.

// 임꺽정 Person 객체 리턴된다.

### 제거

map.put("가", new Person("임꺽정")); // (key: "가", value: 임꺽정 Person 객체) 등록된다.

Person p1 = map.get("가"); // (key: "가") 로 등록했던 value를 리턴한다.

// 임꺽정 Person 객체 리턴된다.

map.remove("가"); // (key: "가") 로 등록했던 데이터를 제거한다.

p1 = map.get("가"); // null 이 리턴된다.

## Map 인터페이스를 구현한 클래스

### HashMap 클래스

해시 테이블 알고리즘으로 구현된 클래스

해시 테이블 알고리즘의 성능

평균: O(1)

최악의 경우: O(*n*)

### TreeMap 클래스

Red Black Tree 알고리즘으로 구현된 클래스

Red Black Tree 알고리즘의 성능

평균: O(log*n*)

최악의 경우: O(log*n*)

TreeMap 클래스는 SortMap 인터페이스를 구현하였다.

등록된 데이터 항목들이 key 순서대로 정렬된다. (Red Black Tree 도 Binary Tree 이므로)

### 다형성

위 두 클래스 모두 Map 인터페이스를 구현했기 때문에,

사용법이 같고 서로 호환된다.

## Map<K,V> interface 메소드

Map 인터페이스를 사용할 때, Map<K, V> 형태로 사용한다.

여기서 K, V 부분에 각각 실제 클래스 타입을 지정해야 한다.

K는 key의 타입이고, V는 value의 타입이다.

|  |
| --- |
| **void clear()**  내부 목록에 들어있는 항목 전체를 제거한다. 그 결과 내부 목록은 비어 있는 상태가 된다. |
| **boolean containsKey(Object k)**  등록된 key 목록 중에서 k 가 포함되어 있으면 true, 없으면 false 리턴 |
| **boolean containsValue(Object v)**  등록된 value 목록 중에서 v 가 포함되어 있으면 true, 없으면 false 리턴 |
| **Set<Map.Entry<K,V>> entrySet()**  등록된 데이터 항목들의 목록을 Set 컬렉션 타입으로 리턴한다.  key의 타입은 **K** 이고, value의 타입은 **V** 이고,  데이터 항목 한 개의 타입은 **Map.Entry<K,V>** 타입이고,  리턴되는 데이터 항목의 목록의 타입은 **Set<Map.Entry<K,V>>** 타입이다. |
| **V get(Object k)**  (key: k) 로 등록했던 value를 리턴한다. |
| **boolean isEmpty()**  등록된 데이터가 0 개이면 true를 리턴한다. |
| **Set<K> keySet()**  등록된 key 목록을 Set 타입으로 리턴한다. |
| **V put(K k, V v)**  (key: k, value: v) 등록한다.  (key: k)로 이미 등록되어 있던 value 가 있다면, v로 치환된다.  리턴 값은 (key: k)로 이미 등록되어 있던 value 이다. 등록되어 있던 value 가 없을 경우 null을 리턴한다. |
| **void putAll(Map map)**  map 에 등록되어 있는 데이터 항목들을 전부 this에 등록한다. |
| **V remove(K v)**  (key: k)로 등록되어 있던 데이터를 제거한다.  리턴 값은 (key: k)로 이미 등록되어 있던 value 이다. 등록되어 있던 value 가 없을 경우 null을 리턴한다. |
| **int size()**  등록된 데이터 항목의 수를 리턴한다. |
| **Collection<V> values()**  등록된 value 목록을 Collection 타입으로 리턴한다. |

## 예제1

Map 인터페이스의 메소드를 사용하는 예제를 살펴보자

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16 | package net.skhu.map;  import java.util.HashMap;  import java.util.Map;  public class Example1 {  public static void main(String[] args) {  Map<String, Integer> map = new HashMap<String, Integer>();  map.put("a", 101);  map.put("b", 102);  map.put("c", 103);  System.out.printf("%d %d %d\n", map.get("a"), map.get("b"), map.get("c"));  }  } |

실행결과 출력

|  |
| --- |
| 101 102 103 |

## 예제2

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19 | package net.skhu.map;  import java.util.HashMap;  import java.util.Map;  public class Example2 {  public static void main(String[] args) {  Map<Integer, String> map = new HashMap<Integer, String>();  map.put(101, "a");  map.put(102, "b");  map.put(103, "c");  System.out.printf("%s %s %s\n", map.get(101), map.get(102), map.get(103));  map.remove(102);  System.out.printf("%s %s %s\n", map.get(101), map.get(102), map.get(103));  }  } |

실행결과 출력

|  |
| --- |
| a b c  a null c |

## 예제3

Person.java

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16 | package net.skhu.map;  public class Person {  String name;  int age;  public Person(String name, int age) {  this.name = name;  this.age = age;  }  @Override  public String toString() {  return String.format("Person(\"%s\",%d)", name, age);  }  } |

Example3.java

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18 | package net.skhu.map;  import java.util.HashMap;  import java.util.Map;  public class Example3 {  public static void main(String[] args) {  Map<Integer, Person> map = new HashMap<Integer, Person>();  map.put(101, new Person("홍길동", 18));  map.put(102, new Person("임꺽정", 19));  System.out.printf("%s %s %s\n", map.get(101), map.get(102), map.get(103));  map.remove(102);  System.out.printf("%s %s %s\n", map.get(101), map.get(102), map.get(103));  }  } |

실행결과 출력

|  |
| --- |
| Person("홍길동",18) Person("임꺽정",19) null  Person("홍길동",18) null null |

# equals, hashCode 메소드 재정의

## equals, hashCode 메소드를 재정의 해야함

HapMap 클래스는 해시 테이블 알고리즘으로 구현되었다.

해시 테이블 알고리즘에서 저장할 데이터의 해시(hash) 값을 계산할 때,

hashCode 메소드를 사용한다.

HaspMap 에 key 로 저장할 클래스는

equals, hashCode 메소드를 재정의 해야 한다.

## Object 클래스의 eauals, hashCode 메소드의 문제점

hashCode 메소드는 Object 클래스에 정의되어 있기 때문에

모든 Java 클래스에 상속된다.

그렇지가 Object 클래스에 구현된 hashCode 메소드를 사용하는 것은 바람직하지 않다.

equals 메소드는 Object 클래스에 정의되어 있기 때문에

모든 Java 클래스에 상속된다.

그렇지가 Object 클래스에 구현된 equals 메소드를 사용하는 것은 바람직하지 않다.

equals 메소드는 객체의 내용이 동일한지 비교해야 하는데 (equality 비교),

Object 클래스에 구현된 equals 메소드는 identity를 비교한다.

그래서 내용이 동일한 두 객체를 비교해도 false가 될 수 있다.

내용이 동일한 두 객체를 equals 메소드로 비교하면 true 이어야 한다.

hashCode 메소드는 객체의 내용을 가지고 계산해야 하는데

Object 클래스에 구현된 hashCode 메소드는 identity를 가지고 계산한다.

그래서 내용이 동일한 두 객체의 hashCode 값이 다를 수 있다.

내용이 동일한 두 객체의 hashCode 값은 같아야 한다.

## Objects 클래스

equals 메소드와 hashCode 메소드를 재정의할 때 편리하게 사용할 수 있는 메소드가

Obejcts 클래스에 포함되어 있다.

### hashCode 메소드 재정의

|  |
| --- |
| @Override  public int hashCode() {  return Objects.hash(name, age);  } |

return Objects.hash(....);

.... 부분에 모든 멤버 변수를 나열하면 된다.

Objects 클래스의 hash 메소드가 해시 값을 계산해 준다.

### equals 메소드 재정의

|  |
| --- |
| @Override  public boolean equals(Object obj) {  if (this == obj) return true;  if (obj instanceof Person2 == false) return false;  Person2 p = (Person2)obj;  return this.age == p.age && Objects.equals(this.name, p.name);  } |

return 문에서, 모든 멤버 변수를 비교해야 한다.

기본 자료형은 this.age == p.age 형태로 비교하고,

참조형은 Objects.equals(this.name, p.name) 형태로 비교하면 된다.

Objects.equals 메소드를 사용하지 않으려면

this.name == null ? p.name == null : this.name.equals(p.name) 형태로 비교하면 되는데,

Objects.equals(this.name, p.name) 형태가 훨씬 갈결하다.

### 사례

|  |
| --- |
| package net.skhu.objects;  import java.util.Date;  import java.util.Objects;  public class Data {  int year;  String address;  Date startDate;  @Override  public int hashCode() {  return Objects.hash(year, address, startDate);  }  @Override  public boolean equals(Object obj) {  if (this == obj) return true;  if (obj instanceof Data == false) return false;  Data d = (Data)obj;  return this.year == d.year &&  Objects.equals(this.address, d.address) &&  Objects.equals(this.startDate, d.startDate);  }  } |

## 예제4

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16 | package net.skhu.map;  import java.util.HashMap;  import java.util.Map;  public class Example4 {  public static void main(String[] args) {  Map<Person, Integer> map = new HashMap<Person, Integer>();  map.put(new Person("홍길동", 18), 101);  map.put(new Person("임꺽정", 19), 102);  System.out.println(map.get(new Person("홍길동", 18)));  System.out.println(map.get(new Person("임꺽정", 19)));  }  } |

실행결과 출력

|  |
| --- |
| null  null |

Person 클래스에 equals, hashCode 메소드가 재정의 되어 있지 않아서,

key를 찾지 못한다.

그래서 null 이 출력된다.

## 예제5

Person2.java

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31 | package net.skhu.map;  import java.util.Objects;  public class Person2 {  String name;  int age;  public Person2(String name, int age) {  this.name = name;  this.age = age;  }  @Override  public String toString() {  return String.format("Person(\"%s\",%d)", name, age);  }  @Override  public int hashCode() {  return Objects.hash(name, age);  }  @Override  public boolean equals(Object obj) {  if (this == obj) return true;  if (obj instanceof Person2 == false) return false;  Person2 p = (Person2)obj;  return this.age == p.age && Objects.equals(this.name, p.name);  }  } |

Example5.java

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16 | package net.skhu.map;  import java.util.HashMap;  import java.util.Map;  public class Example5 {  public static void main(String[] args) {  Map<Person2, Integer> map = new HashMap<Person2, Integer>();  map.put(new Person2("홍길동", 18), 101);  map.put(new Person2("임꺽정", 19), 102);  System.out.println(map.get(new Person2("홍길동", 18)));  System.out.println(map.get(new Person2("임꺽정", 19)));  }  } |

실행결과 출력

|  |
| --- |
| 101  102 |

Person2 클래스에 equals, hashCode 메소드가 재정의되어 있기 때문에

정상적으로 작동한다.

## 목록 출력 예제

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22 | package net.skhu.map;  import java.util.HashMap;  import java.util.Map;  public class Example6 {  public static void main(String[] args) {  Map<String, Integer> map = new HashMap<String, Integer>();  map.put("a", 101);  map.put("b", 102);  map.put("c", 103);  for (String key : map.keySet())  System.out.printf("(%s, %d) ", key, map.get(key));  System.out.println();  for (Map.Entry<String,Integer> entry : map.entrySet())  System.out.printf("(%s, %d) ", entry.getKey(), entry.getValue());  System.out.println();  }  } |

실행결과 출력

|  |
| --- |
| (a, 101) (b, 102) (c, 103)  (a, 101) (b, 102) (c, 103) |

## 구현실습

|  |
| --- |
| package net.skhu.scanner;  import java.io.IOException;  import java.nio.file.Paths;  import java.util.Scanner;  public class Example1 {  public static void main(String[] args) throws IOException {  String filePath = "e:/a.java";  Scanner scanner = new Scanner(Paths.get(filePath));  scanner.useDelimiter("[^a-zA-Z]+");  while (scanner.hasNext()) {  String s = scanner.next();  System.out.println(s);  }  scanner.close();  }  } |

위 소스 코드는

텍스트 파일에서 영어 알파벳으로 구성된 단어 목록을 출력한다.

Scanner scanner = new Scanner(Paths.get(filePath));

경로명이 filePath 인 텍스트 파일을 읽기 위한 Scanner 객체를 생성한다.

scanner.useDelimiter("[^a-zA-Z]+");

이 Scanner 객체를 이용해서 단어(token)를 한 개씩 읽을 때,

단어 사이 구분 문자를 정규식으로 지정한다.

"[^a-zA-Z]+" 정규식

영어 소문자도 아니고, 대문자도 아닌 어떤 문자가 한 개 이상 연속된 문자열과 일치하는 패턴

scanner.hasNext()

텍스트 파일에서 다음 단어(token)를 읽어서 리턴한다.

"[a-zA-Z]+" 정규식

영어 소문자나 대문자가 한 개 이상 연속된 문자열과 일치하는 패턴

### 구현 실습 문제 #1

텍스트 파일에서 각 단어의 수를 세어서 출력하라

단어와 그 단어의 출현 빈도수(count)를 출력하라.

단어: 영어 알파벳만으로 구성된 단어

힌트 Map 객체 활용

### 구현 실습 문제 #2

출현 빈도수 내림차순으로 출력하라.

힌트 출력할 데이터를 ArrayList 에 넣고 정렬한 후 출력해야 함.