**Collection**

2016-11-07

이승진

**학습목표**

Collection Framework 개념을 이해한다.

Collection 인터페이스 메소드 사용법을 학습한다.

Collection 인터페이스 메소드를 사용하여 집합 연산을 구현한다.

**목차**

[1. Collection Framework 2](#_Toc466919554)

[1) Collection Framework 이란? 2](#_Toc466919555)

[2) Collection Framework 구성요소 2](#_Toc466919556)

[3) Collection Framework의 Interface들 3](#_Toc466919557)

[2. Collection interface 5](#_Toc466919558)

[1) Collection interface 메소드 5](#_Toc466919559)

[3. Collection 예제 6](#_Toc466919560)

[1) 예제1 6](#_Toc466919561)

[2) 예제2 7](#_Toc466919562)

[3) 예제3 8](#_Toc466919563)

[4) 예제4 9](#_Toc466919564)

[5) 예제5 10](#_Toc466919565)

[6) 예제6 11](#_Toc466919566)

[7) 예제7 12](#_Toc466919567)

[8) 실습 13](#_Toc466919568)

[4. StringSet 14](#_Toc466919569)

[1) 구현1 14](#_Toc466919570)

[2) 구현2 16](#_Toc466919571)

[3) 구현3 18](#_Toc466919572)

# Collection Framework

## Collection Framework 이란?

### Class library

자주 사용하는 클래스들을 컴파일해서 jar 파일로 묶어 놓은 것을 우리는 클래스 라이브러리(class library)라고 부른다. JDK에 포함된 클래스 라이브러리를 Java 표준 클래스 라이브러리라고 부른다.

### Framework

클래스 라이브러리 중에서 규모가 좀 크고, 클래스들의 구조가 간단하지 않을 것들을, 프레임웍 클래스 라이브러리(framework class library)라고 부른다. 간단히 줄여서 프레임웍(framework)이라고 부르기도 한다.

framework에서 중요한 부분은, 라이브러리 규모가 크다는 점 보다, 클래스들이 서로 연결되고 실행되는 구조가 단순하지 않고 세련되었다는 점이다.

즉 객체지향적으로 세련된 구조를 갖는 클래스 라이브러리를 간단히 프레임웍이라고 부른다.

### Collection Class

내부에 자료 목록을 보관하기 위한 클래스를 컬렉션 클래스라고 부른다.

컬렉션 클래스에 보관할 수 있는 자료는 참조형이어야 한다.

기본 자료형은 컬렉션 클래스에 보관할 수 없다.

기본 자료형을 컬렉션에 보관할 때는, 자동으로 참조형으로 변환되어 보관된다. (auto-boxing)

우리가 잘 알고 있는 ArrayList 클래스가 바로 대표적인 컬렉션 클래스이다.

ArrayList 뿐만아니라 많은 컬렉션 클래스들이 Java 표준 라이브러리에 포함되어 있다.

LinkedList 클래스, Stack 클래스도 대표적인 컬렉션 클래스이다.

### Collection Framework

Java 표준 라이브러리에 포함된 컬렉션 클래스들의 수가 많고, 그 구조가 간단하지 않기 때문에, 이들을 Collection Framework 이라고 부른다.

## Collection Framework 구성요소

Java 표준 라이브러리의 Collection Framework 구성 요소들은 다음과 같다.

### Interfaces

컬렉션 클래스들을 서로 호환되는 몇 개의 그룹으로 나눌 수 있다.

각 그룹을 대표하는 java interface 들이 있다.

예를 들어 List 그룹을 대표하는 것은 List interface이고,

Deque 그룹을 대표하는 것은 Deque interface 이고,

Map 그룹을 대표하는 것은 Map interface 이다.

### implementations

컬렉션 클래스 그룹에 속하는 클래스들은 그 그룹 대표 java interface를 구현(implements)해야 한다.

예를 들어 List 그룹을 대표하는 것은 List interface 이다.

이 그룹에 속하는 ArrayList 클래스와 LinkedList 클래스는 이 List interface를 구현했다.

Deque 그룹을 대표하는 것은 Deque interface 이다.

이 그룹에 속하는 ArrayDeque 클래스는 Deque interface를 구현했다.

Map 그룹을 대표하는 것은 Map interface 이다.

이 그룹에 속하는 HashMap 클래스는 Map interface를 구현했다.

각 그룹에 속하는 클래스들은 그 그룹의 대표 interface를 구현했기 때문에 서로 호환된다.

### Algorithms

collection class 들을 대상으로 정렬(sorting), 탐색(searching) 등의 연산을 구현한 메소드들을 collection framework의 algorithms 라고 부른다. 이 알고리즘 메소드들은 다형성(polymorphism)을 활용하여 구현되었다.

예를들어 ArrayList 클래스와 LinkedList 클래스는 서로 호환된다.

ArrayList와 LinkedList의 public method가 동일하기 때문에, 이 public method 들을 호출하는 형태로 정렬을 한 번만 구현하면 된다. 그렇기 때문에 ArrayList, LinkedList 클래스에 대한 정렬(sort)을 따로 구현할 필요가 없고 하나만 구현해도 된다.

비유를 하자면, 스마트폰 충전 케이블이 서로 호환되기 때문에, 스마트폰 모델마다 충전 케이블을 따로 만들지 않아도 되는 것과 비슷하다.

즉 서로 호환되는 컬렉션 클래스들의 그룹을 대상으로 정렬(sorting), 탐색(searching) 등의 연산을 구현할 때, 클래스 각각에 대해서 따로 구현할 필요 없이, 그룹에 대해서 한 번만 구현하면 된다.

이렇게 구현된 메소드들을 collection framework의 algorithms 라고 부른다.

java.util.Collections 클래스에 알고리즘 메소드들이 구현되어 있다.

## Collection Framework의 Interface들

컬렉션 클래스들의 그룹을 대표하는 java interface들을 collection framework의 interfaces 라고 부른다.

java collection framework에 들어있는 interfaces 들은 다음과 같다.



위 그림의 트리구조는 상속 관계를 나타낸다.

위쪽이 부모 interface이고 아래쪽이 자식 interface 이다.

### Collection interface

대다수의 자바 컬렉션 클래스를 대표하는 interface 이다.

대다수의 자바 컬렉션 클래스들에 Collection interface의 메소드들이 구현되어 있다.

달리 표현하자면, 자바 컬렉션 클래스 대다수에 적용되는 공통 메소드가 Collection interface에 정의되어 있다.

### Set interface

수학에서 집합(set)은 값이 같은 항목을 여러 개 포함할 수 없다.

이 특징을 구현한 컬렉션 클래스 그룹을 대표하는 것이 Set interface 이다.

### List interface

List interface를 구현한 대표적인 클래스가 ArrayList 이다.

배열과 비슷한 방법으로 사용할 수 있는 컬렉션 클래스들을 대표하는 것이 List interface 이다.

배열처럼 인덱스(index) 위치에 값을 저장한다. 인덱스는 0부터 시작한다.

### Queue interface

자료구조 시간에 배운 큐(queue)처럼 사용할 수 있는 컬렉션 클래스들을 대표하는 것이 Queue interface 이다.

Queue에 데이터를 넣고 꺼내는 방식은 FIFO(first-in, first-out) 이다.

### Deque interface

자료구조 시간에 배운 큐(queue)는 데이터를 넣고 꺼내는 방향이 정해져 있어서, 한쪽에서만 넣을 수 있고, 다른쪽에서만 꺼낼 수 있다.

deque는 양방향 큐라고 생각하면 된다. 양쪽에서 넣거나 꺼낼 수 있다. 그래서 큐(queue)처럼 FIFO 방식으로 사용할 수도 있고, 스택(stack)처럼 LIFO 방식으로 사용할 수도 있다.

### Map interface

키(key)로 값(value)을 등록하거나 꺼내는 방식으로 사용하는 컬렉션 클래스들을 대표하는 interface 이다.

### SortedSet interface, SortedMap interface

SortedSet, SortedMap은 각각 Set, Map의 자식 인터페이스이다.

그래서 각각 Set, Map에 정의된 메소드들을 상속 받는다.

SortedSet, SortedMap은 데이터를 정렬된 순서로 보관한다.

데이터가 정렬된 순서이기 때문에 가능한 메소드들이 SortedSet, SortedMap 인터페이스에 포함되어 있다.

# Collection interface

java.util.Collection 인터페이스

자료 목록을 보관하는 용도의 클래스를 컬렉션 클래스라고 부른다.

Java 표준 라이브러리에 들어있는 대부분의 컬렉션 클래스들은 Collection interface를 implements 하였다.

Collection interface를 implements한 클래스는 Collection interface의 메소드를 전부 구현하였다.

## Collection interface 메소드

|  |
| --- |
| **boolean add(E e)**  객체 e를 내부 목록에 추가한다.  추가 작업이 성공했으면 true를, 실패면 false를 리턴한다. |
| **boolean addAll(Collection<E> c)**  목록 객체 c에 들어있는 항목들을 전부 내부 목록에 추가한다.  추가 작업이 성공했으면 true를, 실패면 false를 리턴한다. |
| **void clear()**  내부 목록에 들어있는 항목 전체를 제거한다. 그 결과 내부 목록은 비어 있는 상태가 된다. |
| **boolean contains(Object o)**  파라미터 o와 동일한 값이 내부 목록에 들어있다면 true를 리턴한다.  동일한 값이 있는지 찾을 때 equals 메소드를 호출해서 비교한다. |
| **boolean containsAll(Collection<E> c)**  컬렉션 객체 c에 들어있는 항목들이 전부 this 객체의 내부 목록에 들어있는지 확인한다.  전부 들어있다면 true를 리턴한다.  동일한 값이 있는지 찾을 때 equals 메소드를 호출해서 비교한다. |
| **boolean equals(Object o)**  객체 o와 this 객체의 equality를 비교한다.  즉 this 객체와 o 객체의 클래스가 같고,  목록 o에 들어있는 항목의 수가, this의 그것과 동일하고,  양쪽 목록에 들어있는 항목들 각각이 서로 동일한 경우에 true 를 리턴한다.  항목들 각각이 서로 동일한지 비교할 때 equals 메소드를 호출해서 비교한다. |
| **int hashCode()**  내부 객체 목록까지 고려하여 hash code 값을 계산해서 리턴한다. |
| **boolean isEmpty()**  목록이 비어 있으면 true 를 리턴한다. |
| **Iterator<E> iterator()**  내부 목록에 들어있는 항목을 하나씩 탐색하기 위한 Iterator 객체를 생성하여 리턴한다. |
| **boolean remove(Object o)**  파라미터 o와 동일한 값(equals)을 목록에서 찾아서 제거한다.  파라미터 값이 null 이면, null을 찾아서 제거한다.  제거에 성공했으면 true를, 제거 실패면 false를 리턴한다.  예를 들어 파라미터와 동일한 값이 목록에 없어서 제거할 것이 없는 경우에 false를 리턴한다. |
| **boolean removeAll(Collection c)**  파라미터로 주어진 목록 객체 c에 들어있는 항목들을 this 객체에서 찾아서 전부 제거한다.  제거된 항목이 한 개 이상이면 true를 리턴한다. |
| **boolean retainAll(Collection c)**  파라미터로 주어진 목록 객체 c에 들어있지 않은 항목들을 this 객체에서 찾아서 전부 제거한다.  그래서 c에 들어있는 항목들만 this 객체에 남아있게 된다.  제거된 항목이 한 개 이상이면 true를 리턴한다. |
| **int size()**  내부 목록에 들어있는 항목의 수를 리턴한다. |
| **Object[] toArray()**  목록 객체에 들어있는 항목들을 배열에 채워서 리턴한다.  리턴되는 배열의 타입은 Object[] 이다. |
| **E[] toArray(E[] a)**  목록 객체에 들어있는 항목들을 배열에 채워서 리턴한다.  리턴되는 배열의 타입은 파라미터로 주어진 배열의 타입과 같다. |

# Collection 예제

Collection 인터페이스의 메소드를 사용하는 예제들을 살펴보자.

## 예제1

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17 | package net.skhu.collection;  import java.util.ArrayList;  import java.util.Collection;  public class Example01 {    public static void main(String[] args) {  Collection<String> c = new ArrayList<String>();    c.add("one");  c.add("two");  c.add("three");  for (String s : c)  System.out.printf("%s ", s);  }  } |

실행 결과 출력

|  |
| --- |
| one two three |

(줄9)

Collection<String> c = new ArrayList<String>();

ArrayList 클래스는 Collection 인터페이스를 구현(implements) 하였다.

따라서 Collection 인터페이스가 ArrayList 클래스의 부모 타입이다.

자식 타입의 객체에 대한 참조를 부모 타입의 변수에 대입하는 것은 Ok 이다.

(줄11)

c.add("one");

c.add("two");

c.add("three");

부모 타입의 변수로 메소드를 호출해야 다형성 호출이다.

(줄14)

for (String s : c)

System.out.printf("%s ", s);

배열뿐만 아니라, Collection 인터페이스를 구현한 클래스 객체들도

위와 같은 for 문으로 탐색할 수 있다.

## 예제2

|  |  |
| --- | --- |
| 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 | package net.skhu.collection;  import java.util.ArrayList;  import java.util.Collection;  import java.util.LinkedList;  public class Example02 {    static void addData(Collection<String> c) {  c.add("one");  c.add("two");  c.add("three");  }    static void printCollection(Collection<String> c) {  for (String s : c)  System.out.printf("%s ", s);  System.out.println();  }    public static void main(String[] args) {  Collection<String> c1 = new LinkedList<String>();  Collection<String> c2 = new ArrayList<String>();    addData(c1);  addData(c2);  printCollection(c1);  printCollection(c2);  }  } |

실행 결과 출력

|  |
| --- |
| one two three  one two three |

(줄22)

Collection<String> c1 = new LinkedList<String>();

LinkedList 클래스도 Collection 인터페이스를 구현(implements) 하였다.

따라서 Collection 인터페이스가 LinkedList 클래스의 부모 타입이다.

자식 타입의 객체에 대한 참조를 부모 타입의 변수에 대입하는 것은 Ok 이다.

(줄9~13)

static void addData(Collection<String> c) {

c.add("one");

c.add("two");

c.add("three");

}

addData 메소드가 처음 호출될 때 파라미터 c는 LinksedList<String> 타입의 객체이고,

그 다음 호출 될 때 파라미터 c는 ArrayList<String> 타입의 객체이다.

줄10~12의 add 메소드 호출은 다형성 호출이다.

## 예제3

|  |  |
| --- | --- |
| 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.collection;  import java.util.ArrayList;  import java.util.Arrays;  import java.util.Collection;  import java.util.LinkedList;  public class Example03 {    static void addData(Collection<String> c) {  c.add("one");  c.add("two");  c.add("three");  }    static void printCollection(Collection<String> c) {  Object[] a = c.toArray();  String s = Arrays.toString(a);  System.out.println(s);  }    public static void main(String[] args) {  Collection<String> c1 = new LinkedList<String>();  Collection<String> c2 = new ArrayList<String>();    addData(c1);  addData(c2);  printCollection(c1);  printCollection(c2);  }  } |

실행 결과 출력

|  |
| --- |
| one two three  one two three |

(줄17)

Object[] a = c.toArray();

Collection의 toArray 메소드는 Object[] 타입의 배열을 리턴한다.

리턴되는 a 배열에는, c 컬렉션 객체에 들어 있던 문자열들이 다 들어있다.

정확히 표현하자면, a 배열에 들어있는 것은 문자열 객체들이 아니고,

문자열 객체에 대한 참조값들이 들어 있다.

즉 a 배열과 c 컬렉션 객체에, 동일한 참조값 들이 들어 있다.



## 예제4

|  |  |
| --- | --- |
| 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 | package net.skhu.collection;  import java.util.ArrayList;  import java.util.Arrays;  import java.util.Collection;  import java.util.LinkedList;  public class Example04 {    static void printCollection(Collection<String> c) {  String[] a = c.toArray(new String[0]);  System.out.println(Arrays.toString(a));  }    public static void main(String[] args) {  Collection<String> c1 = new LinkedList<String>();  Collection<String> c2 = new ArrayList<String>();    c1.add("one");  c1.add("two");    c2.addAll(c1);  c2.add("three");    printCollection(c1);  printCollection(c2);  }  } |

실행 결과 출력

|  |
| --- |
| [one, two]  [one, two, three] |

(줄22)

c2.addAll(c1);

c1 컬렉션 객체에 들어있는 문자열 개체들에 대한 참조들을 모두 c2 컬렉션 객체에 add 한다.

즉 줄22는 아래의 코드와 동일한 일을 한다.

|  |
| --- |
| for (String s : c1)  c2.add(s); |

줄22를 위 for 문으로 수정하여 실행하고 결과가 동일함을 확인해 보라.

## 예제5

|  |  |
| --- | --- |
| 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  32 | package net.skhu.collection;  import java.util.ArrayList;  import java.util.Arrays;  import java.util.Collection;  import java.util.LinkedList;  public class Example05 {    static void printCollection(Collection<String> c) {  String[] a = c.toArray(new String[0]);  System.out.println(Arrays.toString(a));  }    public static void main(String[] args) {  Collection<String> c1 = new LinkedList<String>();  Collection<String> c2 = new ArrayList<String>();    c1.add("one");  c1.add("two");    c2.addAll(c1);  c2.add("three");  c2.add("four");    printCollection(c1);  printCollection(c2);    c2.removeAll(c1);  printCollection(c2);  }  } |

실행 결과 출력

|  |
| --- |
| [one, two]  [one, two, three, four]  [three, four] |

(줄29)

c2.removeAll(c1);

c1 컬렉션 객체에 들어있는 문자열 개체들에 대한 참조들을 모두 c2 컬렉션 객체에서 remove 한다.

즉 줄29는 아래의 코드와 동일한 일을 한다.

for (String s : c1)

c2.remove(s);

줄29를 위 for 문으로 수정하여 실행하고 결과가 동일함을 확인해 보라.

## 예제6

|  |  |
| --- | --- |
| 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  32 | package net.skhu.collection;  import java.util.ArrayList;  import java.util.Arrays;  import java.util.Collection;  import java.util.LinkedList;  public class Example06 {    static void printCollection(Collection<String> c) {  String[] a = c.toArray(new String[0]);  System.out.println(Arrays.toString(a));  }    public static void main(String[] args) {  Collection<String> c1 = new LinkedList<String>();  Collection<String> c2 = new ArrayList<String>();    c1.add("one");  c1.add("two");    c2.addAll(c1);  c2.add("three");  c2.add("four");    printCollection(c1);  printCollection(c2);    c2.retainAll(c1);  printCollection(c2);  }  } |

실행 결과 출력

|  |
| --- |
| [one, two]  [one, two, three, four]  [one, two] |

(줄29)

c2.retainAll(c1);

c1 컬렉션 객체에 들어있지 않은 문자열 개체들에 대한 참조들을 모두 c2 컬렉션 객체에서 remove 한다.

아래의 설명은 틀렸음.

즉 줄29는 아래의 코드와 동일한 일을 한다.

for (String s : c2)

if (c1.contains(s) == false)

c2.remove(s);

줄29를 위 for 문으로 수정하여 실행하고 결과가 동일함을 확인해 보라.

for 루프로 탐색 하는 collection에서 add, remove를 할 수 없음.

이 이슈에 대한 설명은 Iterator에서...

## 예제7

|  |  |
| --- | --- |
| 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  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56 | package net.skhu.collection;  import java.util.ArrayList;  import java.util.Arrays;  import java.util.Collection;  import java.util.LinkedList;  import java.util.Stack;  public class Example07 {    static void printCollection(String s, Collection<String> c) {  String[] a = c.toArray(new String[0]);  Arrays.sort(a);  System.out.printf("%s: %s\n", s, Arrays.toString(a));  }    public static void main(String[] args) {  Collection<String> c1 = new Stack<String>();  Collection<String> c2 = new LinkedList<String>();  Collection<String> c3 = new ArrayList<String>();  for (int i = 0; i < 20; i += 2) {  String s = String.format("%02d", i);  c1.add(s);  }  printCollection("c1 (2의 배수)", c1);    for (int i = 0; i < 20; i += 3) {  String s = String.format("%02d", i);  c2.add(s);  }  printCollection("c2 (3의 배수)", c2);    c3.clear();  c3.addAll(c1);  c3.retainAll(c2);  printCollection("c1, c2 교집합", c3);  c3.clear();  c3.addAll(c1);  c3.removeAll(c2);  printCollection("c1, c2 차집합", c3);  c3.clear();  c3.addAll(c2);  c3.removeAll(c1);  printCollection("c2, c1 차집합", c3);  c3.clear();  c3.addAll(c1);  c3.removeAll(c2);  c3.addAll(c2);  printCollection("c1, c2 합집합", c3);  }  } |

실행 결과 출력

|  |
| --- |
| c1 (2의 배수): [00, 02, 04, 06, 08, 10, 12, 14, 16, 18]  c2 (3의 배수): [00, 03, 06, 09, 12, 15, 18]  c1, c2 교집합: [00, 06, 12, 18]  c1, c2 차집합: [02, 04, 08, 10, 14, 16]  c2, c1 차집합: [03, 09, 15]  c1, c2 합집합: [00, 02, 03, 04, 06, 08, 09, 10, 12, 14, 15, 16, 18] |

## 실습

예제7의 코드를 수정하라.

addAll, retainAll, removeall 메소드를 사용하지 말고,

add, remove, contains 메소드만 사용하도록 수정하라.

예제4,5,6의 설명을 참고하라.

# StringSet

3학년 알고리즘 경진대회 문제. 참고.

## 구현1

|  |
| --- |
| package net.skhu.stringSet;  import java.util.ArrayList;  import java.util.Arrays;  class StringSet1 {  ArrayList<String> list = new ArrayList<String>();    public void add(String s) {  list.add(s);  }  public void remove(String s) {  list.remove(s);  }  public boolean contains(String s) {  return list.contains(s);  }  public StringSet1 union(StringSet1 set) {  StringSet1 newSet = new StringSet1();  newSet.list.addAll(this.list);  newSet.list.removeAll(set.list);  newSet.list.addAll(set.list);  return newSet;  }  public StringSet1 intersection(StringSet1 set) {  StringSet1 newSet = new StringSet1();  newSet.list.addAll(this.list);  newSet.list.retainAll(set.list);  return newSet;  }  public StringSet1 difference(StringSet1 set) {  StringSet1 newSet = new StringSet1();  newSet.list.addAll(this.list);  newSet.list.removeAll(set.list);  return newSet;  }  @Override  public boolean equals(Object obj) {  if (obj instanceof StringSet1 == false)  return false;  StringSet1 set = (StringSet1) obj;  return this.list.equals(set.list);  }  public String[] toArray() {  return list.toArray(new String[0]);  }  }  public class Example1 {    static void print(String msg, StringSet1 set) {  String[] a = set.toArray();  Arrays.sort(a);  String temp = Arrays.toString(a);  System.out.printf("%s: %s\n", msg, temp);  }    public static void main(String[] args) {  StringSet1 A = new StringSet1();  for (int i = 0; i < 20; ++i) {  String s = String.format("%02d", i);  A.add(s);  }  print("A", A);  StringSet1 B = new StringSet1();  for (int i = 0; i < 20; i += 2) {  String s = String.format("%02d", i);  B.add(s);  }  print("B", B);  StringSet1 C = new StringSet1();  for (int i = 0; i < 20; i += 3) {  String s = String.format("%02d", i);  C.add(s);  }  print("C", C);  print("A 합집합 B", A.union(B));  print("A 교집합 B", A.intersection(B));  print("A 합집합 C", A.union(C));  print("A 교집합 C", A.intersection(C));  print("B 합집합 C", B.union(C));  print("B 교집합 C", B.intersection(C));  print("A 차집합 B", A.difference(B));    }  } |

## 구현2

|  |
| --- |
| package net.skhu.stringSet;  import java.util.Arrays;  import java.util.TreeSet;  class StringSet2 {  TreeSet<String> list = new TreeSet<String>();    public void add(String s) {  list.add(s);  }  public void remove(String s) {  list.remove(s);  }  public boolean contains(String s) {  return list.contains(s);  }  public StringSet2 union(StringSet2 set) {  StringSet2 newSet = new StringSet2();  newSet.list.addAll(this.list);  newSet.list.removeAll(set.list);  newSet.list.addAll(set.list);  return newSet;  }  public StringSet2 intersection(StringSet2 set) {  StringSet2 newSet = new StringSet2();  newSet.list.addAll(this.list);  newSet.list.retainAll(set.list);  return newSet;  }  public StringSet2 difference(StringSet2 set) {  StringSet2 newSet = new StringSet2();  newSet.list.addAll(this.list);  newSet.list.removeAll(set.list);  return newSet;  }  @Override  public boolean equals(Object obj) {  if (obj instanceof StringSet2 == false)  return false;  StringSet2 set = (StringSet2) obj;  return this.list.equals(set.list);  }  public String[] toArray() {  return list.toArray(new String[0]);  }  }  public class Example2 {    static void print(String msg, StringSet2 set) {  String[] a = set.toArray();  Arrays.sort(a);  String temp = Arrays.toString(a);  System.out.printf("%s: %s\n", msg, temp);  }    public static void main(String[] args) {  StringSet2 A = new StringSet2();  for (int i = 0; i < 20; ++i) {  String s = String.format("%02d", i);  A.add(s);  }  print("A", A);  StringSet2 B = new StringSet2();  for (int i = 0; i < 20; i += 2) {  String s = String.format("%02d", i);  B.add(s);  }  print("B", B);  StringSet2 C = new StringSet2();  for (int i = 0; i < 20; i += 3) {  String s = String.format("%02d", i);  C.add(s);  }  print("C", C);  print("A 합집합 B", A.union(B));  print("A 교집합 B", A.intersection(B));  print("A 합집합 C", A.union(C));  print("A 교집합 C", A.intersection(C));  print("B 합집합 C", B.union(C));  print("B 교집합 C", B.intersection(C));  print("A 차집합 B", A.difference(B));  }  } |

## 구현3

|  |
| --- |
| package net.skhu.stringSet;  import java.util.Arrays;  import java.util.TreeSet;  class StringSet3 {  TreeSet<String> list = new TreeSet<String>();    public void add(String s) {  list.add(s);  }    public void remove(String s) {  list.remove(s);  }    public boolean contains(String s) {  return list.contains(s);  }    public StringSet3 union(StringSet3 set) {  StringSet3 newSet = new StringSet3();  newSet.list.addAll(this.list);  for (String s : set.list)  if (newSet.contains(s) == false)  newSet.add(s);  return newSet;  }  public StringSet3 intersection(StringSet3 set) {  StringSet3 newSet = new StringSet3();  for (String s : list)  if (set.contains(s))  newSet.add(s);  return newSet;  }  public StringSet3 difference(StringSet3 set) {  StringSet3 newSet = new StringSet3();  for (String s : list)  if (set.contains(s) == false)  newSet.add(s);  return newSet;  }    @Override  public boolean equals(Object obj) {  if (obj instanceof StringSet3 == false) return false;  StringSet3 set = (StringSet3)obj;  return this.list.equals(set.list);  }  public String[] toArray() {  return list.toArray(new String[0]);  }  }  public class Example3 {    static void print(String msg, StringSet3 set) {  String[] a = set.toArray();  Arrays.sort(a);  String temp = Arrays.toString(a);  System.out.printf("%s: %s\n", msg, temp);  }    public static void main(String[] args) {  StringSet3 A = new StringSet3();  for (int i = 0; i < 20; ++i) {  String s = String.format("%02d", i);  A.add(s);  }  print("A", A);  StringSet3 B = new StringSet3();  for (int i = 0; i < 20; i += 2) {  String s = String.format("%02d", i);  B.add(s);  }  print("B", B);  StringSet3 C = new StringSet3();  for (int i = 0; i < 20; i += 3) {  String s = String.format("%02d", i);  C.add(s);  }  print("C", C);  print("A 합집합 B", A.union(B));  print("A 교집합 B", A.intersection(B));  print("A 합집합 C", A.union(C));  print("A 교집합 C", A.intersection(C));  print("B 합집합 C", B.union(C));  print("B 교집합 C", B.intersection(C));  print("A 차집합 B", A.difference(B));  }  } |