11. 자료 구조: 컬렉션과 제네릭

정렬
제네릭
Comparator
컬렉션 클래스

pp.352-355

ArrayList 원소 정렬하기

- ArrayList 원소들을 정렬하려면 어떻게 해야 할까?

- API를 뒤져서 정렬 기능을 제공하는 메서드가 있는지 확인!!
  - ArrayList에는 java.util.List 인터페이스에 정의된 default 메서드 sort가 있습니다.
  - **default void sort(Comparator<? super E> c)** → List 인터페이스에 정의된 기본 메서드임.
  - (참고) java.util.Comparator<T> 인터페이스의 메서드: **int compare(T o1, T o2)**

- 혹시 다른 컬렉션 객체를 쓸 수 있는지 확인
  - 자동으로 정렬되는 TreeSet 같은 클래스를 활용하는 것도 좋은 대안이 될 수 있습니다.

(참고) ArrayList.sort() 사용법

Comparator는 두 개의 객체 비교
Comparable은 한 개의 다른 객체와의 비교에 사용됨!

```
import java.util.*;

public class SongSort {
    public void run() {
        ArrayList<String> songs = new ArrayList<>();
        songs.add("Lemon");
        songs.add("Apple");
        songs.add("Mango");
        songs.add("Carrot");
        songs.add("Banana");
        System.out.println("정렬 전: songs = " + songs);

        // songs.sort((o1, o2) -> o1.compareTo(o2));
        MyComparator mc = new MyComparator();
        songs.sort(mc);

        System.out.println("정렬 후: songs = " + songs);
    }

    public static void main(String[] args) {
        new SongSort().run();
    }

    private class MyComparator implements
                                    Comparator<String> {
        @Override
        public int compare(String o1, String o2) {
            return o1.compareTo(o2);
        }
    }
}
```

$ java SongSort.java (javac 사용하지 않고도 실행 가능)
정렬 전: songs = [Lemon, Apple, Mango, Carrot, Banana]
정렬 후: songs = [Apple, Banana, Carrot, Lemon, Mango]

순서대로 입력해 봅시다: JShell 사용

```
$ jshell
jshell> List<String> d1 = List.of("I", "am", "a", "Java", "master")      → Unmodifiable List 반환
jshell> d1.getClass()                                                    → class java.util.ImmutableCollections$ListN
jshell> d1.sort( (e1, e2) -> e1.compareTo(e2) )  → java.lang.UnsupportedOperationException
jshell> List<String> d2 = new ArrayList<>(d1)  → 복사 생성자
jshell> Collections.sort(d2)  또는
jshell> d2.sort( (e1, e2) -> e1.compareTo(e2) )  → d2 정렬
jshell> d2
d2 ==> [I, Java, a, am, master]
```

(참고)

```
List<String> d1 = java.util.Arrays.stream("I am a Java master".split(" "))
                                  .collect(Collectors.toList())    → ArrayList<String> 반환
```

(참고) Unmodified Set

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

public class Main {
    public static void main(String[] args) {
        Set<String> modifiableSet = new HashSet<>();
        modifiableSet.add("A");
        modifiableSet.add("B");

        Set<String> unmodifiableSet = Collections.unmodifiableSet(modifiableSet);

        // 시도 시 UnsupportedOperationException 발생
        // unmodifiableSet.add("C");

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

(참고) Unmodifiable Map

```
import java.util.Collections;
import java.util.HashMap;
import java.util.Map;

public class Main {
    public static void main(String[] args) {
        Map<String, String> modifiableMap = new HashMap<>();
        modifiableMap.put("key1", "value1");
        modifiableMap.put("key2", "value2");

        Map<String, String> unmodifiableMap = Collections.unmodifiableMap(modifiableMap);

        // 시도 시 UnsupportedOperationException 발생
        // unmodifiableMap.put("key3", "value3");

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

p.356

Collections 인터페이스

- **sort(List<T> list)**
- binarySearch()
- reverse()
- min(), max()
- isEmpty()
- add()
- remove()
- stream()

I *Collections*

```
public static final Set EMPTY_SET
public static final List EMPTY_LIST
public static final Map EMPTY_MAP
```

```
public static <T extends Comparable<? super T>> void sort(List<T> list)
public static <T> void sort(List<T> list, Comparator<? super T> c)
public static <T> int binarySearch(List<? extends Comparable<? super T>> list, T key)
public static void reverse(List<?> list)
public static void shuffle(List<?> list)
public static void swap(List<?> list, int i, int j)
public static <T> void fill(List<? super T> list, T obj)
public static <T> void copy(List<? super T> dest, List<? extends T> src)
public static <T extends Object & Comparable<? super T>> T min(Collection<? extends T> coll)
public static <T> T min(Collection<? extends T> coll, Comparator<? super T> comp)
public static <T> T max(Collection<? extends T> coll, Comparator<? super T> comp)
public static <T> boolean replaceAll(List<T> list, T oldVal, T newVal)
boolean isEmpty()
public static final <T> Set<T> emptySet()
public static final <T> List<T> emptyList()
public static final <K,V> Map<K,V> emptyMap()
Object[] toArray()
boolean add(E e)
boolean remove(Object o)
void clear()
default Stream<E> stream()
```

Collections.sort() 메서드

- java.util.Collections 클래스의 클래스 메서드인 sort() 메서드를 활용하면 됩니다.

  public static **void sort(List<T> list)**

- java.util.Collections.sort() 메서드를 이용하면 임의의 List 객체를 알파벳순으로 정렬할 수 있습니다.

- 하지만 String 객체가 아닌, 다른 복잡한 객체로 구성된 리스트는 어떻게 정렬할까요?

Collections.sort() 메서드

Method Detail

sort

```
public static <T extends Comparable<? super T>> void sort(List<T> list)
```

Sorts the specified list into ascending order, according to the *natural ordering* of its elements. All elements in the list must implement the `Comparable` interface. Furthermore, all elements in the list must be *mutually comparable* (that is, `e1.compareTo(e2)` must not throw a `ClassCastException` for any elements `e1` and `e2` in the list).

<T extends Comparable<? super T>>

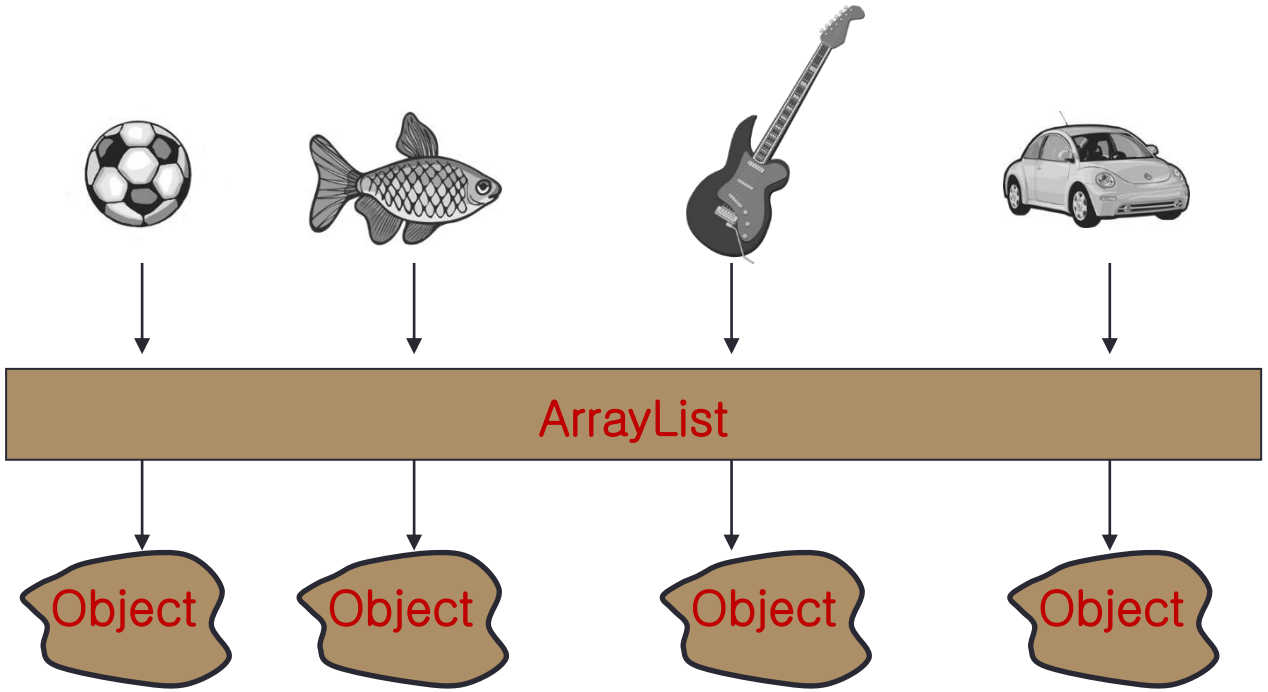
List<T> list

- 처음 보는 이상한 기호들이 등장했습니다.
- 이런 문법들을 이해하기 위해서 제네릭을 공부해야 되겠습니다.

p.362

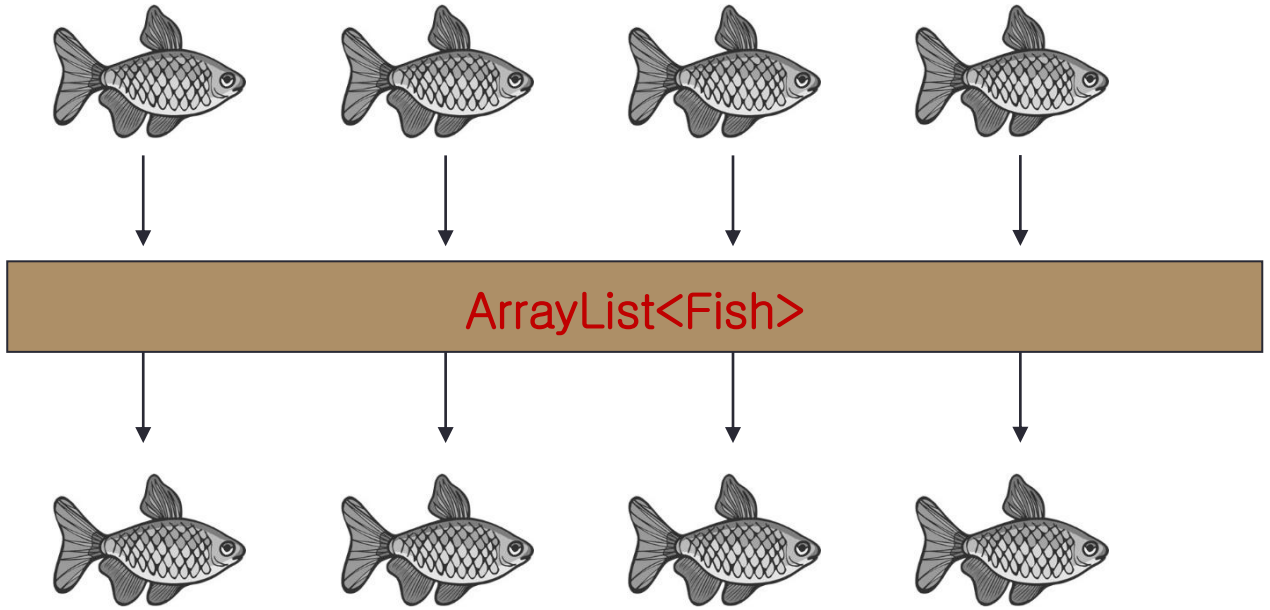
제네릭과 형안전성

- 제네릭을 활용하지 않는 경우
  - 일단 컬렉션에 들어가고 나면 어떤 유형이었는지를 잊어버리게 됩니다.



제네릭과 형안전성

- 제네릭을 활용하는 경우
  - 유형이 엉뚱하게 바뀌지 않습니다.
  - 형안전성이 확보됩니다.



p.363

제네릭 사용법

- 제네릭을 사용하는 클래스

  new ArrayList<Song> 또는 new ArrayList<>()

- 제네릭형 변수 선언 및 대입

  List<Song> songList = new ArrayList<Song>() 또는 List<Song> songList = new ArrayList<>()

- 제네릭형을 받아들이는 메서드 선언 및 호출

  void foo(List<Song> list)

  x.foo(songList)

public <T extends Animal> void takeThing(ArrayList<T> list) ← ArrayList<Dog>, ArrayList<Cat> 객체도 list에 대입 가능

≠

public void takeThing(ArrayList<Animal> list) ← ArrayList<Animal> 객체만 list에 대입 가능

p.369

제네릭 사용법

generics에서 extends는 upper bound,
super는 lower bound를 의미함!

- extends 키워드의 의미
  - 제네릭에서의 extends 키워드는 확장(extends)과 구현(implements)을 모두 의미합니다.
- <T extends Comparable<? super T>> → T는 Comparable의 하위 클래스임을 의미
- 위 코드가 무엇을 뜻하는지 답해보세요.

Comparable 제네릭은 타입 매개변수가
최소한 T 클래스 또는 T의 상위 클래스여야 함.

List<T>: Comparable을 구현하는
T 클래스 객체 사용 가능

```
public static <T extends Comparable<? super T>> void sort(List<T> list)
```

This says "Whatever 'T' is must be of type Comparable."

(Ignore this part for now. But if you can't, it just means that the type parameter for Comparable must be of type T or one of T's supertypes.)

You can pass in only a List (or subtype of list, like ArrayList) that uses a parameterized type that "extends Comparable".

p.372

Comparable 사용

- Comparable<T> 인터페이스의 compareTo() 메서드 구현하면 List에 넣어서 Collections.sort() 이용하면 쉽게 정렬 가능

```
class Song implements Comparable<Song> {
   String title;
   String artist;
   String rating;
   String bpm;

   public int compareTo(Song s) {
       return title.compareTo(s.getTitle());
   }

   Song(String t, String a, String r, String b) {
      title = t;
      artist = a;
      rating = r;
      bpm = b;
   }
```

The sort() ... to see how ... which the ...

p.374

Comparator

- 정렬 순서를 정하는 기준을 직접 정하고 싶다면?
  또는
- Comparable을 구현하지 않는 클래스를 정렬하고 싶다면?

- java.util.**Comparator**<T> 사용 : **int compare(T o1, T o2)** (참고. @FunctionalInterface)
- java.util.**Collections.sort(List o, Comparator c)** 메서드에서도 사용

```
public static <T extends Comparable<? super T>> void sort(List<T> list)
public static <T> void sort(List<T> list, Comparator<? super T> c)
```

Collections.sort(List<T> list)에서 T는
Comparable 인터페이스 구현해야 함.

- (참고) **Comparable**<T> 인터페이스: **int compareTo(T o)**
  → *정렬하기 위한 클래스 안에 비교 연산자 추가 (Comparator는 두 개의 객체 간 순서 비교)*

Comparator 활용

```
class ArtistCompare implements Comparator<Song> {
  public int compare(Song one, Song two) {
    return one.getArtist().compareTo(two.getArtist());
  }
}

.
.
.

ArtistCompare artistCompare = new ArtistCompare();
Collections.sort(songList, artistCompare);
```

ComparableTest.java

```
package ch11_3;

import java.util.ArrayList;
import java.util.Collections;
import java.util.List;


public class ComparableTest {

    public static void main(String[] args) {
        List<Person> data = new ArrayList<>();
        data.add(new Person(20, "이민우"));
        data.add(new Person(10, "김철희"));
        data.add(new Person(15, "박가네"));
        data.add(new Person(7, "최가네"));

        System.out.print("정렬 전: ");
        System.out.println(data);

        Collections.sort(data);

        System.out.print("정렬 후: ");
        System.out.println(data);
    }

}

class Person implements Comparable<Person> {
    private int age;
    private String name;

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

    public int getAge() {
        return age;
    }

    public String getName() {
        return name;
    }
```

```
    /**
     * 나이 오름차순으로 정렬하도록 함.
     * @param o 비교할 Person 객체
     * @return Comparable 인터페이스의 compareTo() 참조
     */
    @Override
    public int compareTo(Person o) {
        return (age - o.age);
    }

    @Override
    public String toString() {
        return String.format("%s (%d)", name, age);
    }

}
```

```
==========================

$ java -Dfile.encoding=UTF-8 ComparableTest.java

정렬 전: [이민우 (20), 김철희 (10), 박가네 (15), 최가네 (7)]
정렬 후: [최가네 (7), 김철희 (10), 박가네 (15), 이민우 (20)]
```

ComparableTest2.java*

record를 사용해서 클래스 정의하면 매개변수 목록을 사용하는 생성자와 getter/~~setter~~ 자동으로 정의됨. getter이름은 int age(), String name() 형태로 됨. (since 14). toString()도 자동 정의됨.

```
package ch11_3;

import java.util.ArrayList;
import java.util.Collections;
import java.util.List;


public class ComparableTest2 {
    public static void main(String[] args) {
        List<Person> data = new ArrayList<>();
        data.add(new Person(20, "이민우"));
        data.add(new Person(10, "김철희"));
        data.add(new Person(15, "박가네"));
        data.add(new Person(7, "최가네"));

        System.out.print("정렬 전: ");
        System.out.println(data);

        Collections.sort(data);

        System.out.print("정렬 후: ");
        System.out.println(data);
    }
}
```

```
record Person(int age, String name) implements Comparable<Person> {

    @Override
    public int compareTo(Person o) {
        return age - o.age;
    }

    @Override
    public String toString() {
        return String.format("%s (%d)", name, age);
    }

}
```

==========

```
$ java -Dfile.encoding=UTF-8 ComparableTest2.java

정렬 전: [이민우 (20), 김철희 (10), 박가네 (15), 최가네 (7)]
정렬 후: [최가네 (7), 김철희 (10), 박가네 (15), 이민우 (20)]
```

ComparatorTest.java

```
package ch11_3;

import java.util.*; // ArrayList, Collections, Comparator, List


public class ComparatorTest {

    public static void main(String[] args) {
        List<Person> data = new ArrayList<>();
        data.add(new Person(20, "이민우"));
        data.add(new Person(10, "김철희"));
        data.add(new Person(15, "박가네"));
        data.add(new Person(7, "최가네"));

        System.out.print("정렬 전: ");
        System.out.println(data);

    // Collections.sort(data, (o1, o2) -> o1.getAge() – o2.getAge());
        Collections.sort(data, new PersonAgeComparator());

        System.out.print("나이 정렬 후: ");
        System.out.println(data);

        Collections.sort(data, new PersonNameComparator());

        System.out.print("이름 정렬 후: ");
        System.out.println(data);
    }

}

class Person {
    private int age;
    private String name;

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

    public int getAge() {
        return age;
    }
```

```
    public String getName() {
        return name;
    }

    @Override
    public String toString() {
        return String.format("%s (%d)", name, age);
    }

}


class PersonAgeComparator implements Comparator<Person> {

    @Override
    public int compare(Person o1, Person o2) {
        return o1.getAge() - o2.getAge();
    }

}
```

```
class PersonNameComparator implements Comparator<Person> {

    @Override
    public int compare(Person o1, Person o2) {
        return o1.getName().compareTo(o2.getName());
    }

}
```

==========

$ java -Dfile.encoding=UTF-8 ComparatorTest.java

```
정렬 전: [이민우 (20), 김철희 (10), 박가네 (15), 최가네 (7)]
나이 정렬 후: [최가네 (7), 김철희 (10), 박가네 (15), 이민우 (20)]
이름 정렬 후: [김철희 (10), 박가네 (15), 이민우 (20), 최가네 (7)]
```

ComparatorTest2.java*

```
package ch11_3;

import java.util.*; // ArrayList, Collections, Comparator, List

public class ComparatorTest2 {

    public static void main(String[] args) {
        List<Person> data = new ArrayList<>();
        data.add(new Person(20, "이민우"));
        data.add(new Person(10, "김철희"));
        data.add(new Person(15, "박가네"));
        data.add(new Person(7, "최가네"));

        System.out.print("정렬 전: ");
        System.out.println(data);

        Collections.sort(data, (o1, o2) -> o1.age() - o2.age());

        System.out.print("나이 정렬 후: ");
        System.out.println(data);

        Collections.sort(data,
                (o1, o2) -> o1.name().compareTo(o2.name()));

        System.out.print("이름 정렬 후: ");
        System.out.println(data);
    }

}

record Person(int age, String name) {
    @Override
    public String toString() {
        return String.format("%s (%d)", name, age);
    }
}
```

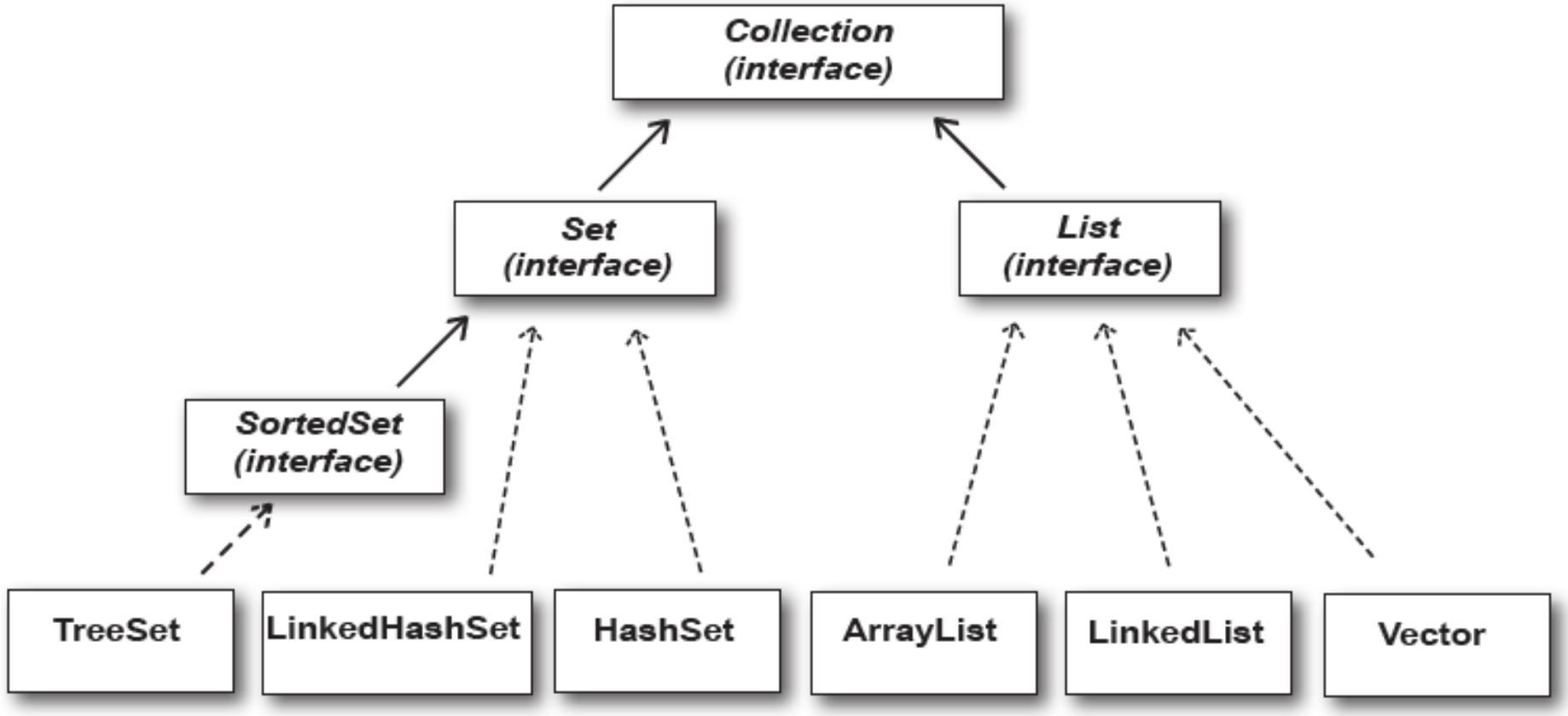
==========

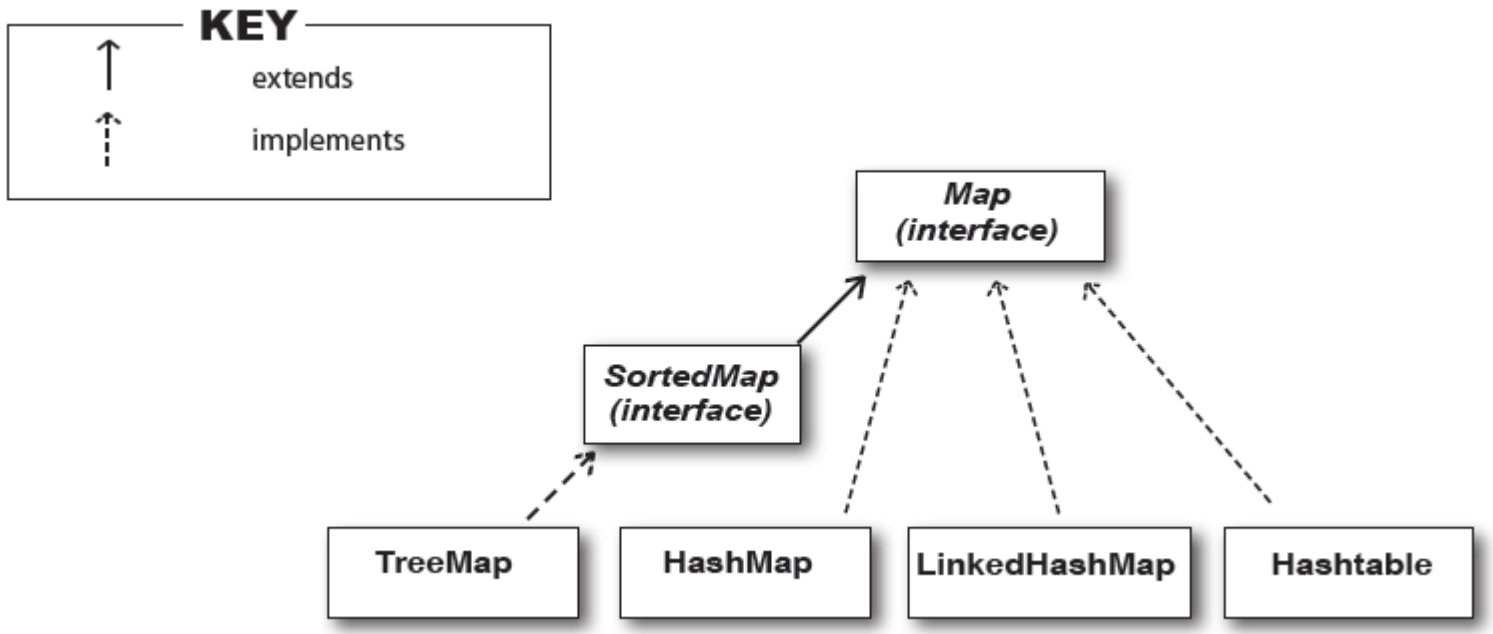
```
$ java -Dfile.encoding=UTF-8 ComparatorTest2.java

정렬 전: [이민우 (20), 김철희 (10), 박가네 (15), 최가네 (7)]
나이 정렬 후: [최가네 (7), 김철희 (10), 박가네 (15), 이민우 (20)]
이름 정렬 후: [김철희 (10), 박가네 (15), 이민우 (20), 최가네 (7)]
```

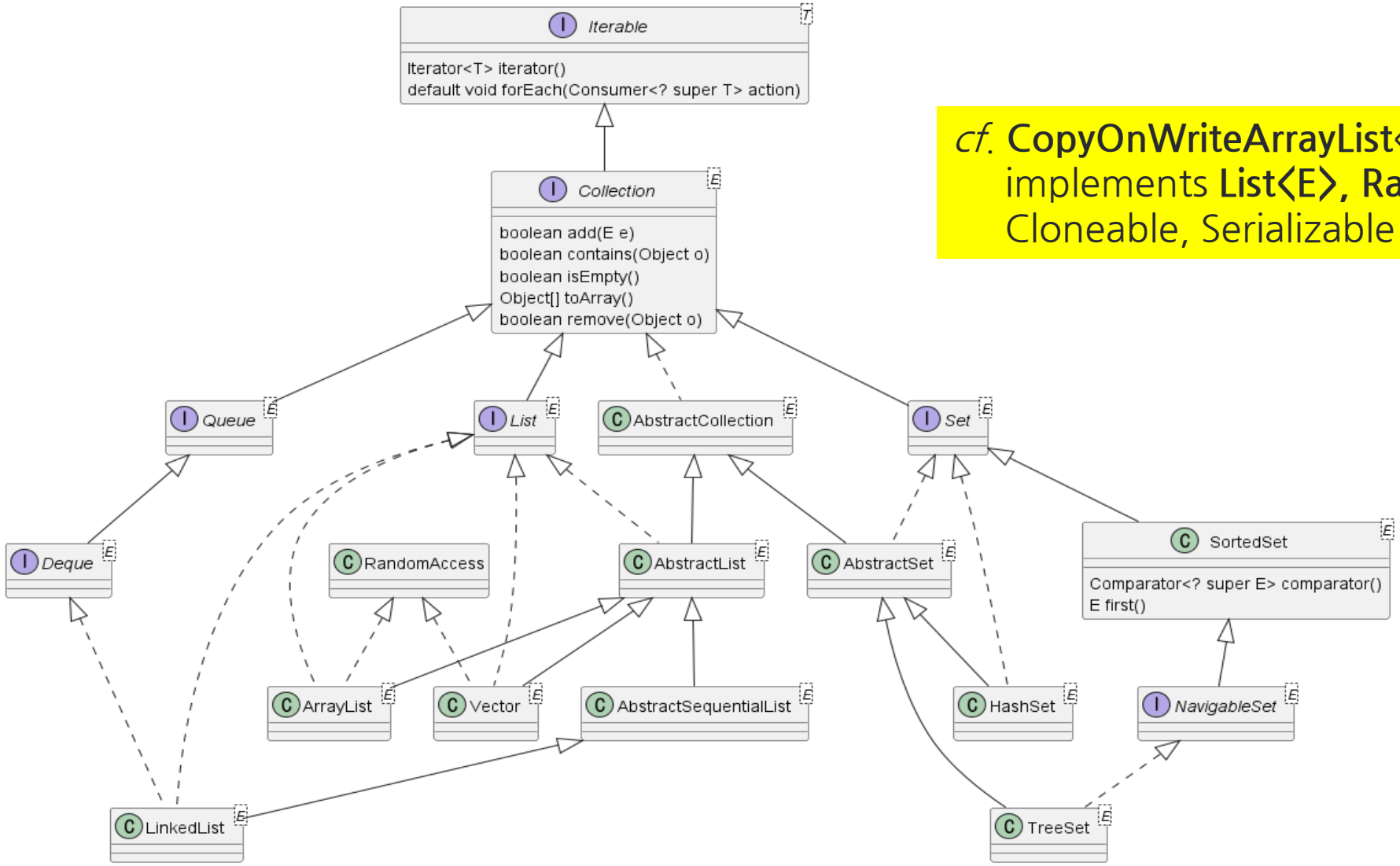
p.388

컬렉션의 종류 (원본)



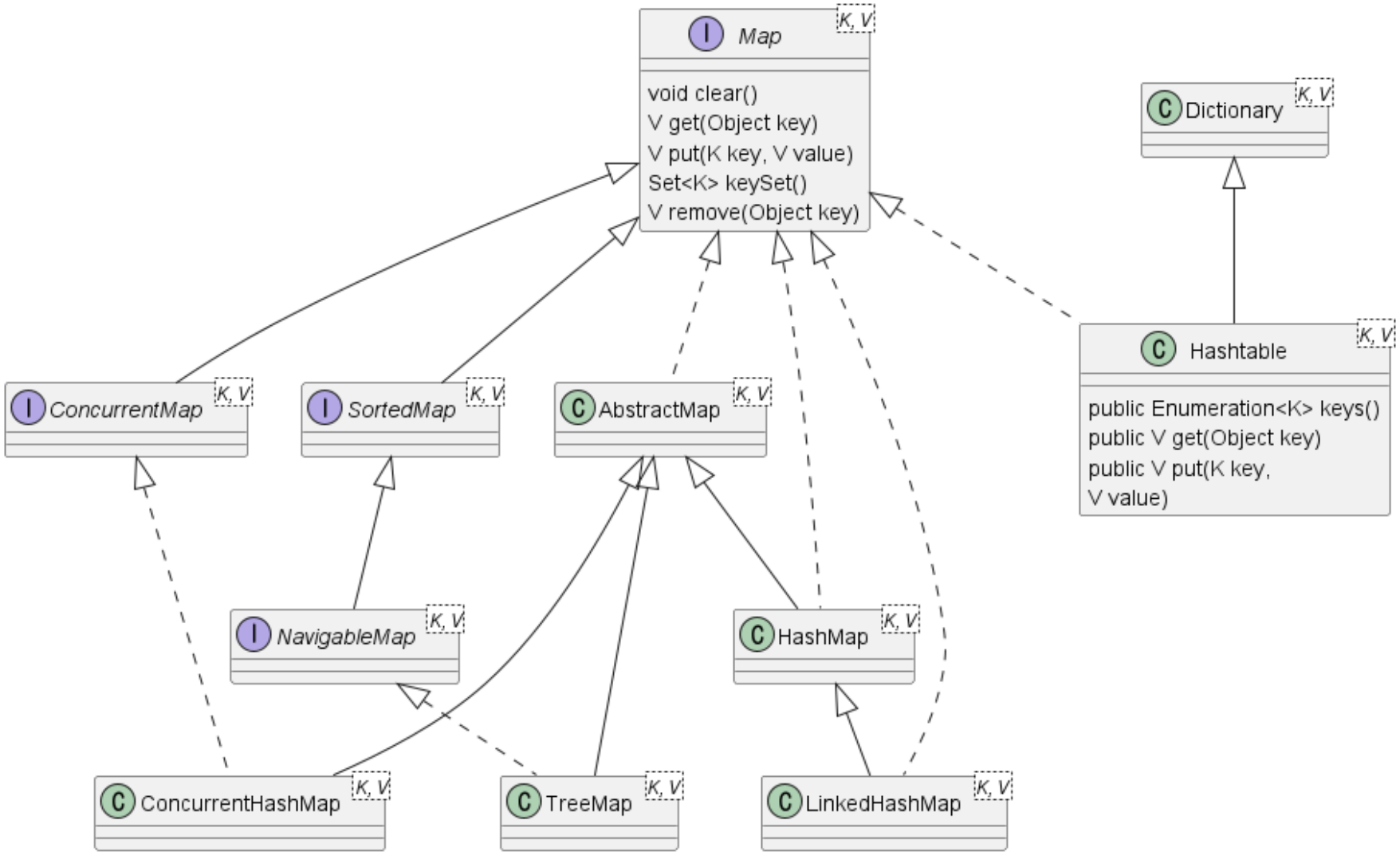

KEY
extends
implements
Map
(interface)
SortedMap
(interface)
TreeMap
HashMap
LinkedHashMap
Hashtable

Collection 인터페이스 구조



cf. **CopyOnWriteArrayList<E>** implements **List<E>, RandomAccess**, Cloneable, Serializable

Map 인터페이스 구조


Map
K, V
void clear()
V get(Object key)
V put(K key, V value)
Set<K> keySet()
V remove(Object key)
Dictionary
K, V
Hashtable
K, V
public Enumeration<K> keys()
public V get(Object key)
public V put(K key,
V value)
ConcurrentMap
K, V
SortedMap
K, V
AbstractMap
K, V
NavigableMap
K, V
HashMap
K, V
ConcurrentHashMap
K, V
TreeMap
K, V
LinkedHashMap
K, V

List

- List
  - 인덱스 제공
  - ListIterator를 줄 수 있음
  - ArrayList
    - 상당히 빠르고 크기를 마음대로 조절할 수 있는 초강력 배열
  - Vector
    - ArrayList의 구형 버전. 모든 메서드가 동기화되어 있음
    - 요즘은 대부분 ArrayList를 사용함
  - LinkedList
    - 컬렉션 중간에서 원소를 추가하거나 삭제하는 작업을 더 빠르게 처리
    - 목록 끝에 원소를 추가하거나 끝에 있는 원소를 쉽게 제거할 수 있는 메서드 제공
    - 스택, 큐, 양방향 큐 등을 만들기 위한 용도로 많이 쓰임

Set

- Set
  - 중복된 항목이 들어가는 것을 방지함
  - equals() 메서드를 이용하여 중복 확인

  - HashSet
    - 가장 빠른 임의 접근 속도
    - 순서를 전혀 예측할 수 없음
  - LinkedHashSet
    - 추가된 순서, 또는 가장 최근에 접근한 순서대로 접근 가능
  - TreeSet
    - **정렬된 순서**대로 보관. 정렬 방법을 지정할 수 있음

SetTest.java

```java
package ch11_3;

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


public class SetTest {
    public static void main(String[] args) {
        Set<Integer> s1 = Set.of(1, 2, 3, 4, 5);
        Set<Integer> s2 = Set.of(4, 5, 6, 7, 8, 9, 10);
        System.out.printf("s1 = %s%n", s1);
        System.out.printf("s2 = %s%n", s2);

        // 합집합
        Set<Integer> s3 = new HashSet<>(s1);
        s3.addAll(s2);
        System.out.printf("합집합 s3 = %s%n", s3);

        // 교집합
        Set<Integer> s4 = new HashSet<>(s1);
        s4.retainAll(s2);
        System.out.printf("교집합 s4 = %s%n", s4);

        // 차집합
        Set<Integer> s5 = new HashSet<>(s1);
        s5.removeAll(s2);
        System.out.printf("차집합 s5 = %s%n", s5);
    }
}
```

```
$ java SetTest.java
s1 = [3, 4, 5, 1, 2]
s2 = [4, 5, 6, 7, 8, 9, 10]
합집합 s3 = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
교집합 s4 = [4, 5]
차집합 s5 = [1, 2, 3]
```

p.395

TestTree.java

```java
package ch11;

import java.util.*;

public class TestTree {
  public static void main(String[] args) {
    new TestTree().go();
  }

  public void go() {
    Book b1 = new Book("How Cats Work");
    Book b2 = new Book("Remix your Body");
    Book b3 = new Book("Finding Emo");

    Set<Book> tree = new TreeSet<>();
    tree.add(b1);
    tree.add(b2);
    tree.add(b3);
    System.out.println(tree);
  }
}
```

```java
class Book implements Comparable<Book> {
  String title;
  public Book(String t) {
    title = t;
  }

  @Ovedrride
  public int compareTo(Book other) {
    return title.compareTo(other.title);
  }

  @Override
  public String toString() {
    return "Book{" +
           "title='" + title + '\'' +
           '}';
  }
}
```

```
$ java TestTree.java
[Book{title='Finding Emo'}, Book{title='How Cats Work'}, Book{title='Remix your Body'}]
```

p.396

TestTreeComparator.java

```java
package ch11;

import java.util.*;



public class TestTreeComparator {
  public static void main(String[] args) {
    new TestTreeComparator().go();
  }

  public void go() {
    Book b1 = new Book("How Cats Work");
    Book b2 = new Book("Remix your Body");
    Book b3 = new Book("Finding Emo");

    BookCompare bookCompare = new BookCompare();
    Set<Book> tree = new TreeSet<>(bookCompare);
    tree.add(b1);
    tree.add(b2);
    tree.add(b3);
    System.out.println(tree);
    Set<Book> tree2 = new TreeSet<>((o1, o2)
            -> o1.title.compareTo(o2.title));
    tree2.add(b1);
    tree2.add(b2);
    tree2.add(b3);
    System.out.println(tree2);
  }
}


class Book {
  String title;
  public Book(String t) {
    title = t;
  }

  @Override
  public String toString() {
    return "Book{" +
            "title='" + title + '\'' +
            '}';
  }
}
```

```
48  class BookCompare implements Comparator<Book> {
49    public int compare(Book one, Book two) {
50      return one.title.compareTo(two.title);
51    }
52  }
```

```
 $ java TestTreeComparator.java
[Book{title='Finding Emo'}, Book{title='How Cats Work'}, Book{title='Remix your Body'}]
[Book{title='Finding Emo'}, Book{title='How Cats Work'}, Book{title='Remix your Body'}]
```

Map

- Map
  - 키와 값을 대응시키는 기능을 제공
  - 키와 값은 모두 객체여야만 함

  - HashMap
    - 가장 빠른 임의 접근 기능을 제공: get(), put() 메서드 실행을 상수 시간에 가능
  - Hashtable
    - HahsMap의 구형 버전
  - LinkedHashMap
    - LinkedHashSet과 유사. 입력된 순서 또는 가장 최근에 접근된 순서대로 보관
  - TreeMap
    - **정렬된 순서**대로 유지하기에 좋음

p.397

TestMap.java

```java
package ch11;

import java.util.*;

public class TestMap {
  public static void main(String[] args) {
    Map<String, Integer> scores = new HashMap<>();

    scores.put("Kathy", 42);
    scores.put("Bert", 343);
    scores.put("Skyler", 420);

    System.out.println(scores);
    System.out.println(scores.get("Bert"));
  }
}
```

```
$ java TestMap.java
{Kathy=42, Skyler=420, Bert=343}
343
```

p.390

객체가 같은지 확인하는 방법

- 레퍼런스 동치
  - 두 레퍼런스가 같은 객체를 참조하는 경우
  - a == b 가 참이 되어야 함: a.hashCode() == b.hashCode()와 동일함!

- 객체 동치
  - equals() 메서드에 대해 true가 리턴되는 것
  - 똑같은 객체를 참조해야 하는 것은 아님

```
String a = “abc”;
if (a.equals(new String(“abc”)) {
```

- 어떤 조건에서 두 객체가 같다고 인정할 수 있을지 결정한 다음 equals() 메서드를 오버라이드해야 함.
- 이 때 hashCode() 메서드도 반드시 오버라이드해야 함: == 연산자와 관련 有

와일드카드

```
public <T extends Animal> void takeThing(ArrayList<T> list)
```

```
public void takeThing(ArrayList<? extends Animal> list)
```

- 와일드카드를 쓸 때는 목록에 새로운 원소를 추가할 수 없음

(참고) EqualityTest.java*

```
package cse.oop2.ch16.equal_test;


public class EqualityTest {

    public static void main(String[] args) {
        String s1 = "Hello";
        String s2 = "Hello";
        System.out.printf("Hash code for s1 = %d, s2 = %d%n",
                s1.hashCode(), s2.hashCode());
        System.out.printf("동일 여부: %s%n%n",
                s1.equals(s2));

        Person p1 = new Person(10);
        Person p2 = new Person(10);
        System.out.printf("Hash code for p1 = %d, p2 = %d%n",
                p1.hashCode(), p2.hashCode());
        System.out.printf("Person 레퍼런스 동치 여부: %s%n",
                p1 == p2);
        System.out.printf("Person 객체 동치 여부: %s%n%n",
                p1.equals(p2));

        NewPerson n1 = new NewPerson(10);
        NewPerson n2 = new NewPerson(10);
        System.out.printf("Hash code for n1 = %d, n2 = %d%n",
                n1.hashCode(), n2.hashCode());
        System.out.printf("NewPerson 레퍼런스 동치 여부: %s%n",
                n1 == n2);
        System.out.printf("NewPerson 객체 동치 여부: %s%n%n",
                n1.equals(n2));
    }

}


class Person {
    private int age;

    public Person(int age) {
        this.age = age;
    }
}
```

```
class NewPerson {
    private Integer age;

    public NewPerson(int age) {
        this.age = age;
    }

    @Override
    public int hashCode() {
        return age.hashCode();
    }

    @Override
    public boolean equals(Object obj) {
        if (this == obj) {
            return true;
        }
        if (obj == null) {
            return false;
        }
        if (getClass() != obj.getClass()) {
            return false;
        }
        final NewPerson other = (NewPerson) obj;
        return this.age == other.age;
    }

}
```

실행 결과

Hash code for s1 = 69609650, s2 = 69609650
동일 여부: true

Hash code for p1 = 381707837, p2 = 589446616
Person 레퍼런스 동치 여부: false
Person 객체 동치 여부: false

Hash code for n1 = 10, n2 = 10
NewPerson 레퍼런스 동치 여부: false
NewPerson 객체 동치 여부: true

(참고) <? extends T> vs <? super T>

- URL: https://velog.io/@kasania/Java-Generic%EC%97%90-%EB%8C%80%ED%95%9C-%EA%B4%80%EC%B0%B0-2
- extends 예시
  - public **<T extends Readable>** void func(**T** readable) { ... }
- super 예시

```
public void addLoggers(List<? super Writer> list){
    list.add(new BufferedWriter(new OutputStreamWriter(System.out)));
    list.add(new FileWriter("log.txt"));
} // ?는 최소한 Writer의 기능까지는 구현이 보장되어야 하며, 그 하위 타입인지는 관심이 없다
```

- PECS (Producer Extends, Consumer Super)
  - 외부에서 데이터를 생산한다면(Producer), extends를, 외부에서 데이터를 소모한다면(Consumer), super를 사용하라

- (참고) Guidelines for Wildcard Use (The Java™ Tutorials > Learning the Java Language > Generics (Updated)) (oracle.com)

- An "In" Variable
  - An "in" variable serves up data to the code. Imagine a copy method with two arguments: copy(src, dest). The src argument provides the data to be copied, so it is the "in" parameter.
- An "Out" Variable
  - An "out" variable holds data for use elsewhere. In the copy example, copy(src, dest), the dest argument accepts data, so it is the "out" parameter.

```
public void copyList(List<? extends Reader> in, List<? super Reader> out){
    for (Reader integer : in) {
        out.add(integer);
    }
}
```

TODO

- 본문을 다시 한 번 꼼꼼히 읽어봅시다.
- 본문 중간에 나와있는 코드를 직접 입력해서 컴파일하고 실행해 봅시다.