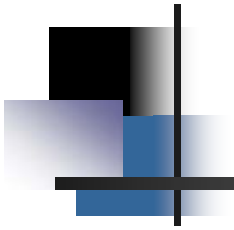


Collection과 Generic

남 광 우

Head First Java

기본 : Java Arrays

- 배열의 선언

```
int[] myArray;
int[] myArray = new int[5];
String[] stringArray = new String[10];
String[] strings = new String[] {“one”, “two”};
```

- 배열의 길이

```
int arrayLength = myArray.length;
```

- 배열의 반복 접근

```
for(int I=0; I<myArray.length; i++)
{
   String s = myArray[i];
}
```

기본 : Java Arrays

- 배열의 복사

```
int array1[] = new int[10];
int array2[] = new int[10];
//assume we add items to array1

//copy array1 into array2
System.arrayCopy(array1, 0, array2, 0, 10);
//copy last 5 elements in array1 into first 5 of array2
System.arrayCopy(array1, 5, array2, 0, 5);
```

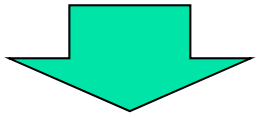
- 배열의 정렬

```
int myArray[] = new int[] {5, 4, 3, 2, 1};
Arrays.sort(myArray);
//myArray now holds 1, 2, 3, 4, 5
```

기본 : Java Arrays

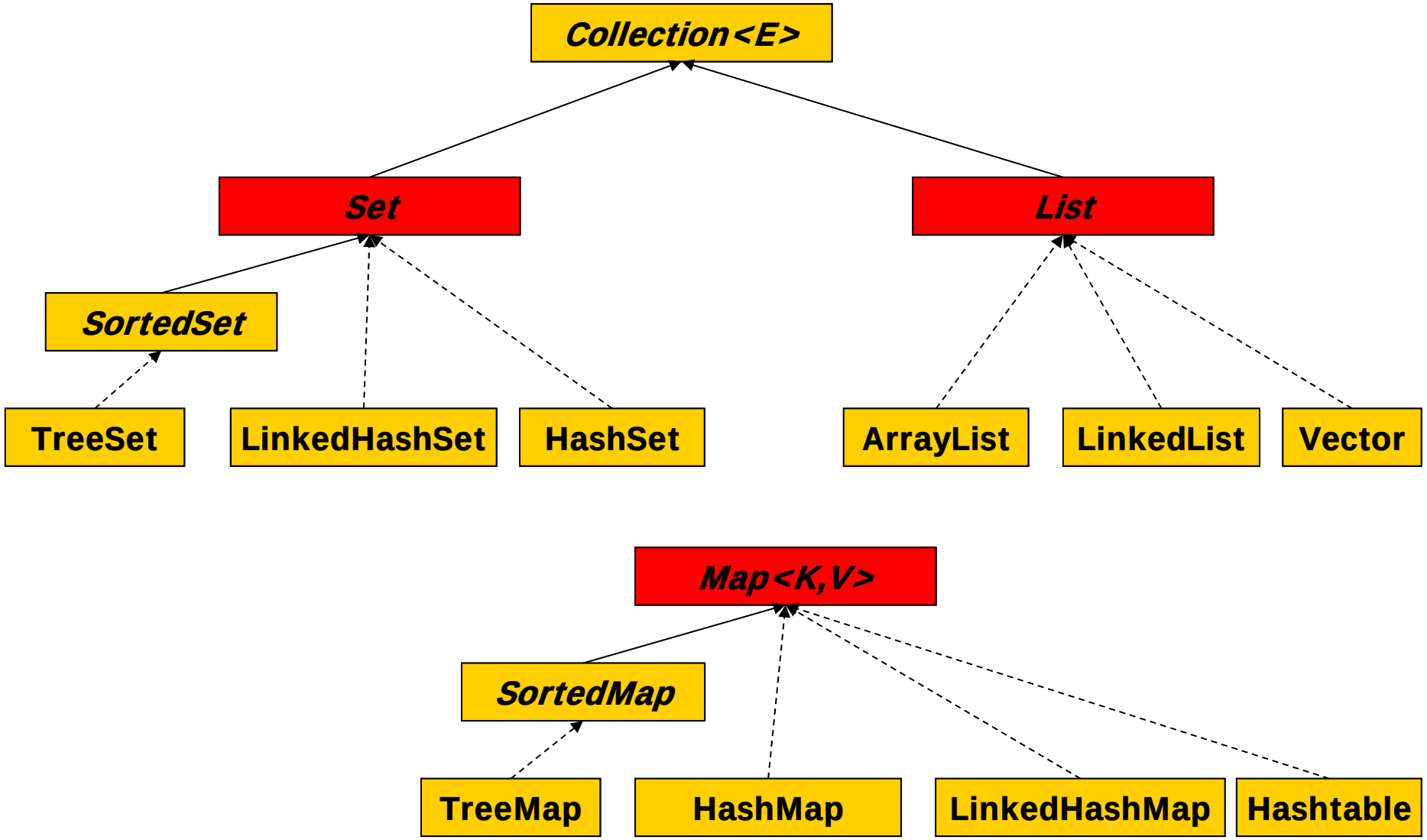
- **배열의 장점**
  - 데이터 접근과 사용이 매우 효율적
  - Type-safe : 배열에 선언된 데이터 타입만 추가될수 있음

- **배열의 단점**
  - 고정 크기, 복사와 크기 재조정에 오버헤드 존재
  - 배열안에 몇 개의 데이터가 있는지 일일이 검사해야함
  - 제한된 기능, 좀더 일반적인 기능이 필요함(링크드 리스트)



Java Collection Framework

컬렉션의 종류


Collection<E>
Set
List
SortedSet
TreeSet
LinkedHashSet
HashSet
ArrayList
LinkedList
Vector
Map<K,V>
SortedMap
TreeMap
HashMap
LinkedHashMap
Hashtable

Interface Collection

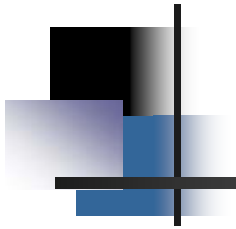
- 인터페이스

• `add(o)`	Add a new element
• `addAll(c)`	Add a collection
• `clear()`	Remove all elements
• `contains(o)`	Membership checking.
• `containsAll(c)`	Inclusion checking
• `isEmpty()`	Whether it is empty
• `iterator()`	Return an iterator
• `remove(o)`	Remove an element
• `removeAll(c)`	Remove a collection
• `retainAll(c)`	Keep the elements
• `size()`	The number of elements

Interface List

- 인터페이스

- `add(i,o)` Insert `o` at position `i`
- *`add(o)`* Append `o` to the end
- `get(i)` Return the `i`-th element
- `remove(i)` Remove the `i`-th element
- `set(i,o)` Replace the `i`-th element with `o`
- `indexOf(o)`
- `lastIndexOf(o)`
- `listIterator()`
- `sublist(i,j)`



Interface Map

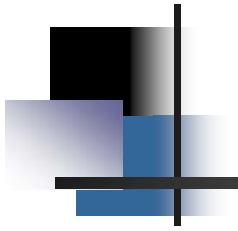
• `clear()`	Remove all mappings
• `containsKey(k)`	Whether contains a mapping for `k`
• `containsValue(v)`	Whether contains a mapping to `v`
• `entrySet()`	Set of key-value pairs
• `get(k)`	The value associated with `k`
• `isEmpty()`	Whether it is empty
• `keySet()`	Set of keys
• `put(k,v)`	Associate `v` with `k`
• `remove(k)`	Remove the mapping for `k`
• `size()`	The number of pairs
• `values()`	The collection of values

Collection:List

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

Collection:Set

- Set
  - 중복된 항목이 들어가는 것을 방지함
  - equals() 메소드를 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 순서대로 보관. 정렬 방법을 지정할 수 있음



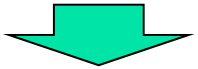
Map

- Map
  - 키와 값을 대응시키는 기능을 제공
  - 키와 값은 모두 객체여야만 함
  - Hashtable(Thread-safe)
    - HahsMap의 구형 버전
  - HashMap(=>Thread? then ConcurrentHashMap)
    - 가장 빠른 임의 접근 기능을 제공
  - LinkedHashMap
    - LinkedHashSet과 유사. 입력된 순서 또는 가장 최근에 접근된 순서대로 보관
  - TreeMap
    - 정렬된 순서대로 유지하기에 좋음

Collection의 기본 : 생성

- Collection의 생성 방법

```
List myList = new ArrayList();
List otherList = new ArrayList(5);
Map database = new HashMap();
Set things = new HashSet();
```



1.5이후 generic

```
List<String> myList = new ArrayList<String>();
List <String> otherList = new ArrayList <String> (5);
Set<String> things = new HashSet<String> ();

Map<String, Object> database = new HashMap<String, Object> ();
```

Collection의 기본 : 데이터 추가

- Collection 의 데이터 추가 : add()

```
List myList = new
ArrayList();
myList.add(“A String”);
myList.add(“Other String”);
```

- Map의 데이터 추가 : put()

```
Map myMap = new HashMap();
myMap.put(“google”, “http://www.google.com”);
mpMap.put(“yahoo”, “http://www.yahoo.com”);
```

Collection의 기본 : 복사

- 데이터의 복사 : addAll()

```
List myNewList = new ArrayList();
//assume we add items to the list

List otherList = new ArrayList();
myNewList.addAll(myList);
```

Collection의 기본 : 데이터 접근

- Collection에서의 데이터 접근 : get()

```
String s = (String)myList.get(1);
String s2 = (String)myList.get(10);
```

- Map에서의 데이터 접근

```
String s = (String)myMap.get("google");
String s2 = (String)mpMap.get("yahoo");
```

Collection의 기본 : 반복문

- Collection에서 Iterator를 이용한 반복문

```
List myList = new ArrayList();
//we add items

Iterator iterator = myList.iterator();
while (iterator.hasNext())
{
  String s = (String)iterator.next();
  //do something with it
}
```

Collection의 기본 : 반복문

- Set에서의 예

```
Set set = new HashSet(); // instantiate a concrete set
// ...
set.add(obj); // insert an elements
// ...
int n = set.size(); // get size
// ...
if (set.contains(obj)) {...} // check membership

// iterate through the set
Iterator iter = set.iterator();
while (iter.hasNext()) {
  Object e = iter.next();
  // downcast e
  // ...
}
```

Collection의 기본 : 반복문

- Map에서의 예

```
Map map = new HashMap(); // instantiate a concrete map
// ...
map.put(key, val); // insert a key-value pair
// ...
// get the value associated with key
Object val = map.get(key);
map.remove(key); // remove a key-value pair
// ...
if (map.containsValue(val)) { ... }
if (map.containsKey(key)) { ... }
Set keys = map.keySet(); // get the set of keys
// iterate through the set of keys
Iterator iter = keys.iterator();
while (iter.hasNext()) {
  Key key = (Key) iter.next();
  // ...
}
```

Collection의 예 :Word 카운팅

- Set을 이용한 서로다른 단어의 수 count 하기

```
public class CountWords {
  static public void main(String[] args) {
    Set words = new HashSet();
    BufferedReader in =
         new BufferedReader(
             new InputStreamReader(System.in));
    String delim = " \t\n.,:;?!-/()[]\"\'";
    String line;
    int count = 0;
 try {
      while ((line = in.readLine()) != null) {
        StringTokenizer st =
           new StringTokenizer(line, delim);
        while (st.hasMoreTokens()) {
          count++;
          words.add(
            st.nextToken().toLowerCase());
        }
      }
    } catch (IOException e) {}
    System.out.println("Total number of words: "
                       + count);
    System.out.println("Number of different words: "
                       + words.size());
  }
}
```

Generic 없으므로 warning

Collection의 예 :Word Frequency

- HashMap을 이용한 각 단어의 출현 빈도 카운팅하기

```
public class Count {

  public String word;
  public int i;

  public Count(String word, int i) {
    this.word = word;
    this.i = i;
  }
}
```

Collection의 예 :Word Frequency

```
public class WordFrequency {

  static public void main(String[] args) {
    Map words = new HashMap();
    String delim = " \t\n.,:;?!-/()[]\"\'";
    BufferedReader in =
      new BufferedReader(
        new InputStreamReader(System.in));
    String line, word;
    Count count;

try {
      while ((line = in.readLine()) != null) {
        StringTokenizer st =
          new StringTokenizer(line, delim);
        while (st.hasMoreTokens()) {
          word = st.nextToken().toLowerCase();
          count = (Count) words.get(word);
          if (count == null) {
            words.put(word,
                      new Count(word, 1));
          } else {
            count.i++;
          }
        }
      }
    } catch (IOException e) {}
```

Collection의 예 :Word Frequency

```
 Set set = words.entrySet();
    Iterator iter = set.iterator();
    while (iter.hasNext()) {
      Map.Entry entry =
                  (Map.Entry) iter.next();
      word = (String) entry.getKey();
      count = (Count) entry.getValue();
      System.out.println(word +
          (word.length() < 8 ? "\t\t" : "\t") +
          count.i);
    }
  }
}
```

ArrayList 원소 정렬하기

- ArrayList 원소들을 정렬하려면 어떻게 해야 할까?

- API를 뒤져서 정렬 기능을 제공하는 메소드가 있는지 확인!!
  - ArrayList에는 정렬용 메소드가 없습니다.

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

Collections.sort() 메소드

- java.util.Collections 클래스의 클래스 메소드인 sort() 메소드를 활용하면 됩니다.

  public static void sort(List list)

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

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

Collections.sort() 메소드

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).

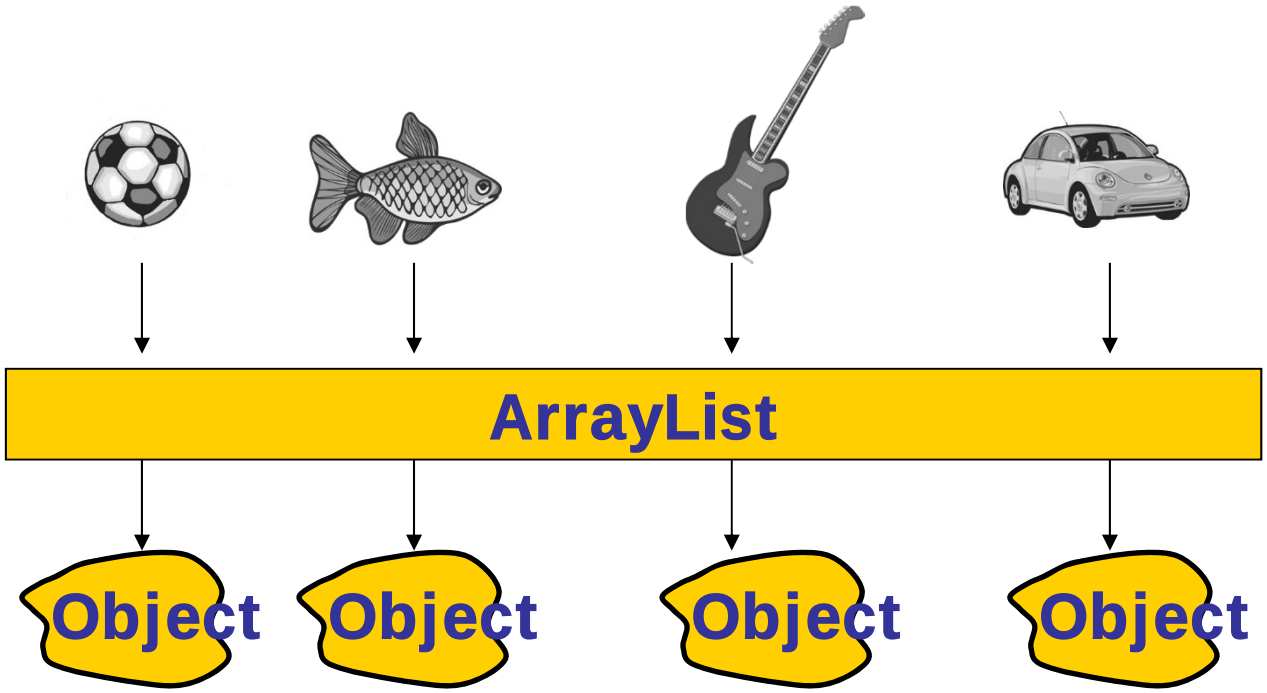
<T extends Comparable<? super T>>

List<T> list

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

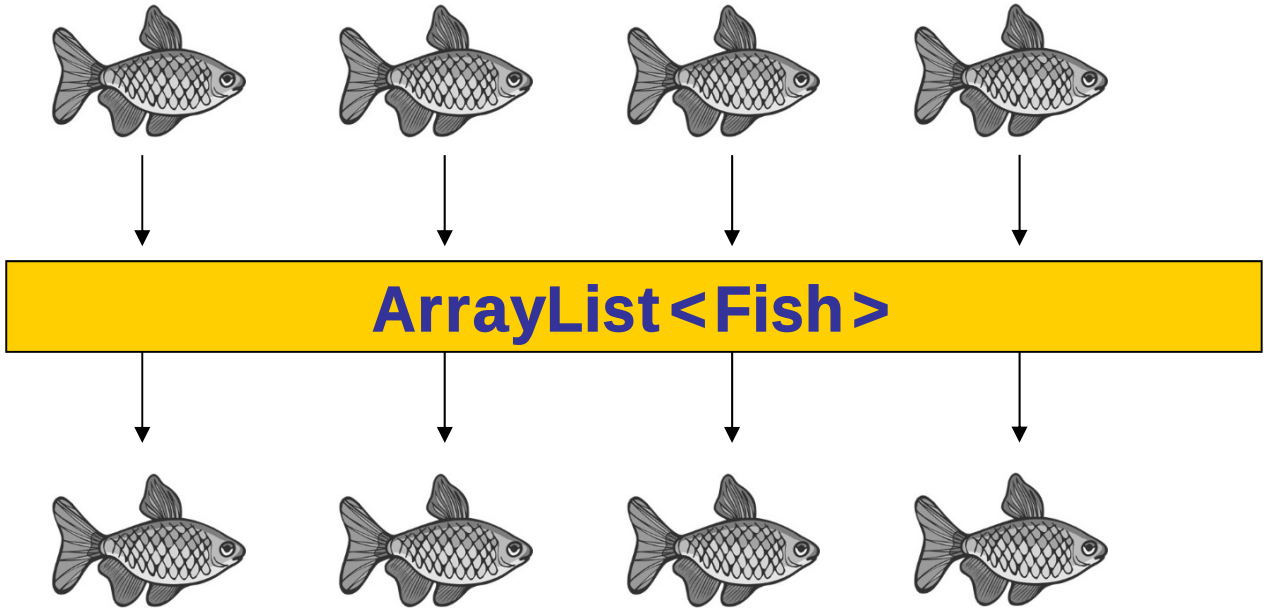
제네릭과 형안전성

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



제네릭과 형안전성

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



제네릭 사용법

- 제네릭을 사용하는 클래스

  new ArrayList<Song>()

- 제네릭형 변수 선언 및 대입

  List<Song> songList = new ArrayaList<Song>()

- 제네릭형을 받아들이는 메소드 선언 및 호출

  void foo(List<Song> list)

  x.foo(songList)

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

  public void takeThing(ArrayList<Animal> list)

제네릭 사용법

- extends 키워드의 의미
  - 제네릭에서의 extends 키워드는 확장과 구현을 모두 의미합니다.

- <T extends Comparable<? super T>>

- 위 코드가 무엇을 뜻하는지 답해보세요.

제네릭 사용법

Comparator

- 정렬 순서를 정하는 기준을 직접 정하고 싶다면?

  또는

- Comparable을 구현하지 않는 클래스를 정렬하고 싶다면 ?

- java.util.Comparator 사용
- sort(List o, Comparator c) 메소드 사용

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);
```

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