

컬렉션 프레임워크 2

24-1. 컬렉션 기반 알고리즘

정렬

List<E>를 구현한 컬렉션 클래스들은 저장된 인스턴스를 정렬된 상태로 유지하지 않는다.

대신에 정렬을 해야 한다면 다음 메소드를 사용할 수 있다.

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

→ Collections 클래스에 정의되어 있는 제네릭 메소드

→ 인자로 List<T>의 인스턴스는 모두 전달 가능

→ 단, T는 Comparable<T> 인터페이스를 구현한 상태이어야 한다.

리스트 대상 정렬의 예


```
public static void main(String[] args) {
    List<String> list = Arrays.asList("Toy", "Box", "Robot", "Weapon");
    list = new ArrayList<>(list);

    // 정렬 이전 출력
    for(Iterator<String> itr = list.iterator(); itr.hasNext(); )
        System.out.print(itr.next() + '\t');
    System.out.println();

    // 정렬
    Collections.sort(list);

    // 정렬 이후 출력
    for(Iterator<String> itr = list.iterator(); itr.hasNext(); )
        System.out.print(itr.next() + '\t');
    System.out.println();
}
```

`public static <T extends Comparable<T>> void sort(List<T> list)`
`class String extends Object implements Comparable<String>`



```
C:\JavaStudy>java SortCollections
Toy    Box    Robot  Weapon
Box    Robot  Toy    Weapon
C:\JavaStudy>
```

<T extends Comparable<T>> 아니고 <T extends Comparable<? super T>>

임시로

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

실제로

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

이러한 선언이 붙은 이유를 설명하는데 시간이 조금 걸림!

이해했으면 그 이해의 내용을 외우자!

<T extends Comparable<? super T>> 의 이해 1

```
class Car implements Comparable<Car> {  
    private int disp; // 배기량  
    public Car(int d) { disp = d; }  
    . . . .  
    @Override  
    public int compareTo(Car o) {  
        return disp - o.disp;  
    }  
}
```

sort 메소드가 다음과 같다고 가정하자!

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

```
public static void main(String[] args) {  
    List<Car> list = new ArrayList<>();  
    list.add(new Car(1200));  
    list.add(new Car(3000));  
    list.add(new Car(1800));  
    Collections.sort(list); // 정렬  
  
    for(Iterator<Car> itr = list.iterator(); itr.hasNext(); )  
        System.out.println(itr.next().toString() + '\t');  
}
```

<T extends Comparable<? super T>> 의 이해 2

```
class Car implements Comparable<Car> {...}
```

```
class ECar extends Car {...}    // ECar는 Comparable<Car>를 간접 구현
```

sort 메소드가 다음과 같다고 여전히 가정하자!

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

```
public static void main(String[] args) {  
    List<ECar> list = new ArrayList<>();  
    ....  
    Collections.sort(list);    // 이 메소드 호출이 성공할 수 있을까?  
    ....  
}  
    public static <ECar extends Comparable<ECar>> void sort(List<ECar> list)
```

<T extends Comparable<? super T>> 의 이해 3

```
class Car implements Comparable<Car> {...}
```

```
class ECar extends Car {...}    // ECar는 Comparable<Car>를 간접 구현
```

그러나 실제로는 다음과 같으니!

```
    public static <T extends Comparable<? super T>> void sort(List<T>
        list)
public static void main(String[] args) {
    List<ECar> list = new ArrayList<>();
    ....
    Collections.sort(list);    // 이 메소드 호출 가능!
    ....
}
```

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


정렬: Comparator<T> 기반

Collections 클래스에는 호출 시 정렬의 기준을 결정할 수 있는 다음 메소드가 정의되어 있다.

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

```
class Car { . . . }
```

```
// Car의 정렬을 위한 클래스
```

```
class CarComp implements Comparator<Car> {  
    . . . .  
}
```

```
class ECar extends Car { . . . }
```

```
sort(List<Car> list, Comparator<? super Car> c)  
sort(List<ECar> list, Comparator<? super ECar> c)
```

```
public static void main(String[] args) {  
    List<Car> clist = new ArrayList<>();  
    clist.add(new Car(1800));  
    clist.add(new Car(1200));
```

```
    List<ECar> elist = new ArrayList<>();  
    elist.add(new ECar(3000, 55));  
    elist.add(new ECar(1800, 87));
```

```
    CarComp comp = new CarComp();
```

```
    // 각각 정렬  
    Collections.sort(clist, comp);  
    Collections.sort(elist, comp);
```

```
    . . .  
}
```

찾기

```
public static <T> int binarySearch(List<? extends Comparable<? super T>> list, T key)
```

→ list에서 key를 찾아 그 인덱스 값 반환, 못 찾으면 음의 정수 반환

Step 1.

```
public static <T> int binarySearch(List<?> list, T key)
```

Step 2.

```
(List<? extends Comparable<T>> list, T key)
```

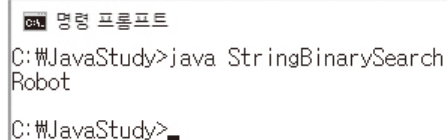
Step 3.

```
(List<? extends Comparable<? super T>> list, T key)
```

찾기의 예

```
public static <T> int binarySearch(List<? extends Comparable<? super T>> list, T key)
```

```
class StringBinarySearch {  
    public static void main(String[] args) {  
        List<String> list = new ArrayList<>();  
        list.add("Box");  
        list.add("Robot");  
        list.add("Apple");  
        Collections.sort(list);    // 정렬이 먼저다!  
        int idx = Collections.binarySearch(list, "Robot");    // 탐색  
        System.out.println(list.get(idx));    // 탐색의 결과 출력  
    }  
}
```



```
C:\JavaStudy>java StringBinarySearch  
Robot  
C:\JavaStudy>
```

찾기: Comparator<T> 기반

```
public static <T> int binarySearch(List<? extends T> list, T key, Comparator<? super T> c)
```

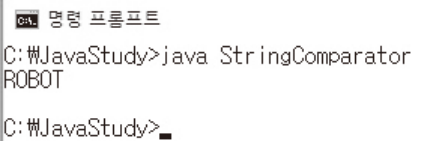
→ list에서 key를 찾는데 c의 기준을 적용하여 찾는다.

Step 1. `public static <T> int binarySearch(List<T> list, T key, Comparator<T> c)`

Step 2. `(List<? extends T> list, T key, Comparator<? super T> c)`

찾기: Comparator<T> 기반의 예

```
class StrComp implements Comparator<String> {  
    @Override  
    public int compare(String s1, String s2) {  
        return s1.compareToIgnoreCase(s2);    // 대문자, 소문자 구분 없이 비교  
    }  
}  
  
class StringComparator {  
    public static void main(String[] args) {  
        List<String> list = new ArrayList<>();  
        list.add("ROBOT");  
        list.add("APPLE");  
        list.add("BOX");  
  
        StrComp cmp = new StrComp(); // 정렬과 탐색의 기준  
        Collections.sort(list, cmp); // 정렬  
        int idx = Collections.binarySearch(list, "Robot", cmp); // 탐색  
        System.out.println(list.get(idx)); // 탐색 결과 출력  
    }  
}
```



```
CA> 명령 프롬프트  
C:\JavaStudy>java StringComparator  
ROBOT  
C:\JavaStudy>
```

복사하기

```
public static <T> void copy(List<? super T> dest, List<? extends T> src)
```

→ src의 내용을 dest로 복사

List<T> dest 아닌 List<? super T> dest 인 이유는?

→ dest에 T형 인스턴스를 넣는 것만 허용하겠다. 꺼내면 컴파일 에러!

List<T> src 아닌 List<? extends T> src 인 이유는?

→ src로부터 T형 인스턴스 꺼내는 것만 허용하겠다. 넣으면 컴파일 에러!