



아이템 14. Comparable을 구현할지 고려하라

- Primitive 자료형은 쉽게 비교 및 정렬이 가능

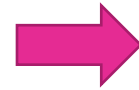
```
public class Application {  
    public static void main(String[] args) {  
        int a = 1;  
        int b = 2;  
  
        System.out.println(a > b);  
    }  
}  
  
public class Application {  
    public static void main(String[] args) {  
        int[] integers = {1, 3, 5, 2, 11, 4, 7};  
        Arrays.sort(integers);  
        System.out.println(Arrays.toString(integers));  
    }  
}
```

부등호로 비교하여
false 값 반환

[1, 2, 3, 4, 5, 7, 11]로 정렬

객체의 비교는 어떻게 해야 할까?

```
public class Book {  
    private final String title;  
    private final int price;  
  
    public Book(String title, int price) {  
        this.title = title;  
        this.price = price;  
    }  
  
    public String getTitle() {  
        return title;  
    }  
  
    public int getPrice() {  
        return price;  
    }  
}
```



이 객체를 비교하고
정렬할 수 있을까?

Comparable 이란?

```
public interface Comparable<T> {  
    public int compareTo(T o);  
}
```

- interface
- 제네릭 <T> 자리에 비교할 타입이 들어감
- 구현체는 반드시 compareTo 메서드를 오버라이딩 하여 사용
- 자바 라이브러리의 모든 값 클래스와 열거 객체는 Comparable을 구현

- String의 정렬

```
public class Application {  
    public static void main(String[] args) {  
        String a = "apple";  
        String b = "book";  
        List<String> words = new ArrayList<>();  
  
        words.add(b);  
        words.add(a);  
        System.out.println(words);  
  
        Collections.sort(words);  
        System.out.println(words);  
    }  
}
```

[book, apple]

[apple, book]

- 다른 객체의 정렬

```
public class Application {  
    public static void main(String[] args) {  
        Book effectiveJava = new Book("이펙티브 자바", 30000);  
        Book modernJavaInAction = new Book("모던 자바 인 액션", 20000);  
        List<Book> books = new ArrayList<>();  
  
        books.add(effectiveJava);  
        books.add(modernJavaInAction);  
        System.out.println(books);  
  
        Collections.sort(books);  
        System.out.println(books);  
    }  
}
```

컴파일 에러!

error: no suitable method found for sort(List<Book>)

String은 Comparable을 구현했지만 Book은 Comparable을 구현하지 않았다

```
public final class String
    implements java.io.Serializable, Comparable<String>,
    CharSequence {
    ...
}
```

```
public class Book {
    private final String title;
    private final int price;

    public Book(String title, int price) {
        this.title = title;
        this.price = price;
    }

    public String getTitle() {
        return title;
    }

    public int getPrice() {
        return price;
    }
}
```

**compareTo를 구현할 때는
규약을 지키자**

- compareTo의 구현 방법

```
public interface Comparable<T> {  
    public int compareTo(T o);  
}
```

- compareTo를 호출하는 인스턴스와 매개변수로 주어지는 인스턴스의 순서를 비교한다.
- 이 객체를 기준으로, 주어진 객체보다 작으면 음의 정수를, 주어진 객체와 같으면 0을, 주어진 객체보다 크면 양의 정수를 반환한다.
 - (일반적으로 -1, 0, 1을 반환하나 반드시 -1, 1일 필요는 없다.)
- 비교할 수 없는 타입의 인스턴스가 주어지면 ClassCastException을 던진다.

- 첫 번째 규약

- `x.compareTo(y)`와 `y.compareTo(x)`의 부호는 반대여야 한다.
- `x.compareTo(y)`가 0이면 `y.compareTo(x)`도 0이다.
 - 즉, $x < y$ 면 $y > x$ 이고, $x == y$ 이면 $y == x$ 여야 한다.
- `x.compareTo(y)`가 예외를 던지면 `y.compareTo(x)`도 예외를 던져야 한다.

-> 비교의 순서가 바뀌어도 반드시 예상한 결과가 나와야 한다.

- 두 번째 규약

- $x.compareTo(y) > 0$, $y.compareTo(z) > 0$ 이면 $x.compareTo(z) > 0$ 이다.
 - 즉, $x > y$ 이고 $y > z$ 이면 $x > z$ 여야 한다.

- 세 번째 규약

- `x.compareTo(y) == 0` 이면 `x.compare(z)`와 `y.compare(z)`의 값이 같다.
 - 즉, `x`와 `y`가 같다면 모든 `z`에 대하여 `x`와 `y`의 비교 결과는 같아야 한다.

- 네 번째 규약(권고)

- $(x.compareTo(y) == 0) == (x.equals(y))$ 여야 한다.
 - 즉, `compareTo`로 수행한 동치성 테스트의 결과가 `equals`의 결과와 같아야 한다.
 - 이 규약을 잘 지키면 `compareTo`로 줄지은 순서와 `equals`의 결과가 일관된다.
- 지키지 않는 클래스는 그 사실을 명시해야 한다.

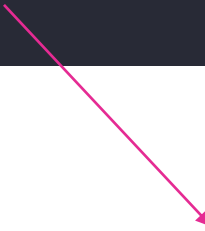
- 이 규약을 지키지 않은 예시: `BigDecimal`

- `new BigDecimal("1.0")`과 `new BigDecimal("1.00")`
- `compareTo`로 비교하면 두 인스턴스가 같으나 `equals`로 비교하면 서로 다르다.
- 따라서 `TreeSet`에서는 원소를 1개만, `HashSet`에서는 원소를 2개 갖는다.

Comparable 구현 시 주의사항

- 원시 타입을 비교할 때 <나 > 대신 wrapper 클래스의 compare를 사용하자

```
public class Position implements Comparable<Position> {  
    private final int position;  
  
    ...  
  
    @Override  
    public int compareTo(Position target) {  
        return Integer.compare(position, target.position);  
    }  
}
```

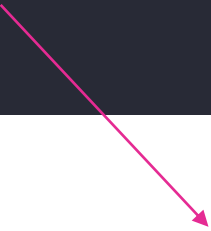


Integer 클래스가 제공하는 정적 메서드 compare 사용

- 정렬 기준인 필드가 여러 개일 때

- Book 클래스를 가격 순으로, 만약 가격이 같으면 제목 순으로 정렬하고 싶을 때

```
@Override
public int compareTo(Book o) {
    int result = Integer.compare(price, o.price);
    if (result == 0) {
        result = String.CASE_INSENSITIVE_ORDER.compare(title, o.title);
    }
    return result;
}
```



price 값이 같을 경우 title에 대한 비교 수행
만약 비교하려는 필드가 추가될 경우 if 안에 같은 방법으로 또다른 필드 비교 추가

- 정렬 기준인 필드가 여러 개일 때
 - Comparator 인터페이스의 비교자 생성 메서드를 활용

```
private static final Comparator<Book> COMPARATOR =  
    Comparator.comparingInt((Book book) -> book.price)  
                .thenComparing(book -> book.title);  
  
@Override  
public int compareTo(Book o) {  
    return COMPARATOR.compare(this, o);  
}
```



기존 방법 대비 성능 저하가 있음

여기서 잠깐,
Comparable하고 Comparator가
뭐가 다르죠?

Comparable과 Comparator의 비교

- Comparable

- 자기 자신과 매개변수 객체를 비교
- 주로 정렬해야 하는 클래스를 구현체로 만들어서 사용
- lang 패키지 -> import 필요 x

- Comparator


- 두 매개변수 객체를 비교
- 주로 익명 객체로 구현해서 사용
- util 패키지 -> import 필요

- Quiz. 이렇게도 가능하지 않을까?

```
public class Position implements Comparable<Position> {  
    private final int position;  
  
    ...  
  
    @Override  
    public int compareTo(Position target) {  
        return position - target.position;  
    }  
}
```

- 안티 패턴: 값의 차를 기준으로 비교

```
public class Position implements Comparable<Position> {  
    private final int position;  
  
    ...  
  
    @Override  
    public int compareTo(Position target) {  
        return position - target.position;  
    }  
}
```



뿔셈 과정에서 자료형의 범위를 넘어버리는 경우가 있고
실수 뿔셈의 경우에는 부동소수점 계산에 따른 오류가 날 수 있다

- Tip: Comparator 자리에 Comparable의 compareTo를 넣을 수 있다.

```
private Position getMaxPosition(List<Car> cars) {  
    return cars.stream()  
        .map(Car::getPosition)  
        .max(Position::compareTo)  
        .orElse(Position.fromStartLine());  
}
```

max() 안에는 원래 Comparator가 들어간다.

보통 람다식으로 처리하는데, compareTo를 대신해서 넣어도 작동한다.