

Effective Java Study

Woowacourse_study 4th



Item 52 by 알파

다중정의란..?

흔히 아는 오버로딩

리턴 타입과 메서드 네이밍은 같으나,

파라미터의 개수나 타입이 다른 메서드를 작성하는 것

다중정의란..?

무엇이 출력될까?

```
public class CollectionClassifier {  
    public static String classify(Set<?> s) {  
        return "집합";  
    }  
  
    public static String classify(List<?> lst) {  
        return "리스트";  
    }  
  
    public static String classify(Collection<?> c) {  
        return "그 외";  
    }  
  
    public static void main(String[] args) {  
        Collection<?>[] collections = {  
            new HashSet<String>(),  
            new ArrayList<BigInteger>(),  
            new HashMap<String, String>().values()  
        };  
  
        for (Collection<?> c : collections)  
            System.out.println(classify(c));  
    }  
}
```

다중정의란..?

collections 배열의 타입이
Collection 타입이기 때문에
마지막 classify가 실행

```
public class CollectionClassifier {  
    public static String classify(Set<?> s) {  
        return "집합";  
    }  
  
    public static String classify(List<?> lst) {  
        return "리스트";  
    }  
  
    public static String classify(Collection<?> c) {  
        return "그 외";  
    }  
  
    public static void main(String[] args) {  
        Collection<?>[] collections = {  
            new HashSet<String>(),  
            new ArrayList<BigInteger>(),  
            new HashMap<String, String>().values()  
        };  
  
        for (Collection<?> c : collections)  
            System.out.println(classify(c));  
    }  
}
```

다중정의란..?

즉, 컴파일러 시점에서 오버로딩이 정해짐 = 정적

오버라이딩의 경우 런타임 시점에서 정해짐 = 동적

재정의는..?

같은 타입의 Wine이지만,
최하위 클래스가 정의한
Name을 호출한다

```
class Wine {
    String name() { return "와인"; }
}

class SparklingWine extends Wine {
    @Override
    String name() {
        return "스파클링 와인";
    }
}

class Champagne extends SparklingWine {
    @Override
    String name() {
        return "샴페인";
    }
}

public static void main(String[] args) {
    List<Wine> wineList = List.of(
        new Wine(), new SparklingWine(), new Champagne());

    for (Wine wine : wineList)
        System.out.println(wine.name());
}
```

다중정의의 단점

**사용자가 파라미터를 넘기면서 어떤
오버로딩 메서드가 호출될 지 모른다면 오작동
→ 혼란 발생**

해결책

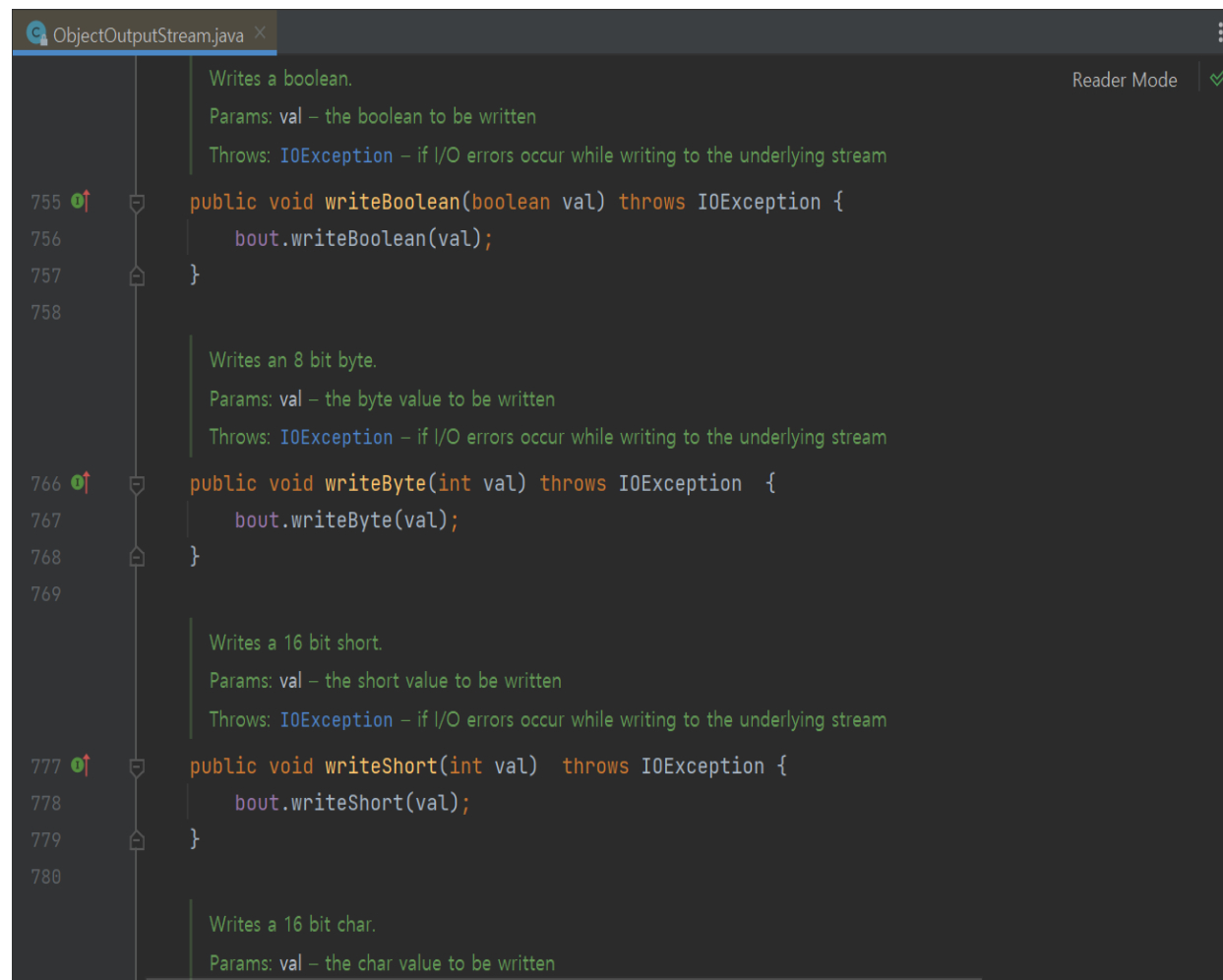
웬만하면 오버로딩을 하지말자

가변인수를 쓰는 메서드는 아예 시도조차 하지 말자

가장 간단한 방법 = 메서드 네이밍을 다르게 지어주기

해결책 예시

ObjectOutputStream



```
ObjectOutputStream.java x
Reader Mode ✓

Writes a boolean.
Params: val – the boolean to be written
Throws: IOException – if I/O errors occur while writing to the underlying stream
755 public void writeBoolean(boolean val) throws IOException {
756     bout.writeBoolean(val);
757 }
758

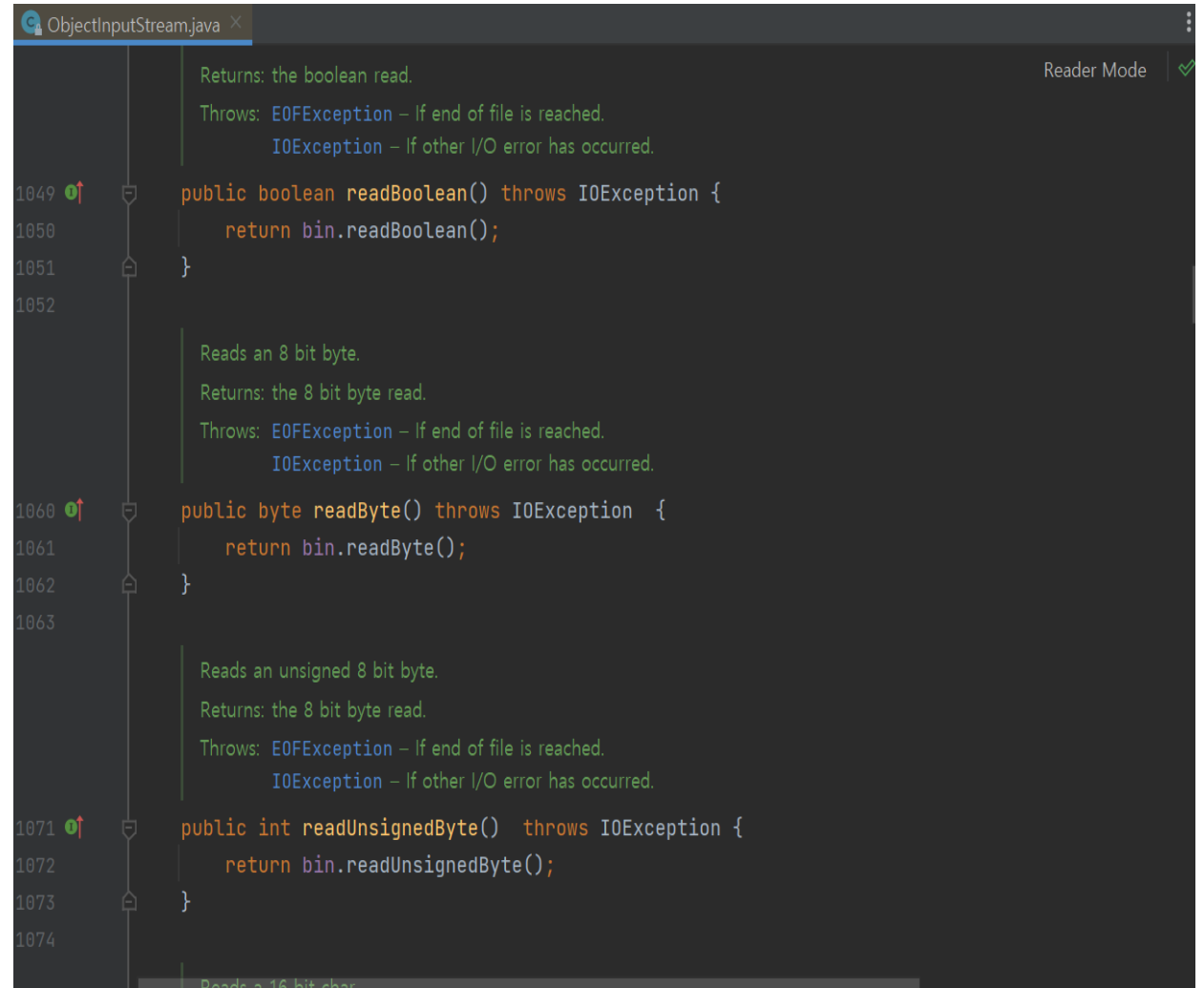
Writes an 8 bit byte.
Params: val – the byte value to be written
Throws: IOException – if I/O errors occur while writing to the underlying stream
766 public void writeByte(int val) throws IOException {
767     bout.writeByte(val);
768 }
769

Writes a 16 bit short.
Params: val – the short value to be written
Throws: IOException – if I/O errors occur while writing to the underlying stream
777 public void writeShort(int val) throws IOException {
778     bout.writeShort(val);
779 }
780

Writes a 16 bit char.
Params: val – the char value to be written
```

해결책 예시

ObjectInputStream



```
ObjectInputStream.java x
Returns: the boolean read.
Throws: EOFException – If end of file is reached.
        IOException – If other I/O error has occurred.
1049 public boolean readBoolean() throws IOException {
1050     return bin.readBoolean();
1051 }
1052
Returns: the 8 bit byte read.
Throws: EOFException – If end of file is reached.
        IOException – If other I/O error has occurred.
1060 public byte readByte() throws IOException {
1061     return bin.readByte();
1062 }
1063
Returns: the 8 bit byte read.
Throws: EOFException – If end of file is reached.
        IOException – If other I/O error has occurred.
1071 public int readUnsignedByte() throws IOException {
1072     return bin.readUnsignedByte();
1073 }
1074
Returns: the 16 bit char.
```

생성자는..?

미션을 진행하면서 주 생성자, 부 생성자 등

여러 생성자를 만든 상황이 존재

만약, 파라미터 개수만 같은 생성자들끼리는 어떻게 처리할까?

생성자는..?

파라미터 중 한 개 이상이 근본적으로 다른 타입이면 가능

생성자는..?

**파라미터 중 한 개 이상이 근본적으로 다른 타입이면 가능
근본적으로 다른 타입이란, 서로 캐스팅 할 수 없는 것을 의미**

생성자는..?

Constructs an empty list with the specified initial capacity.

Params: `initialCapacity` – the initial capacity of the list

Throws: `IllegalArgumentException` – if the specified initial capacity is negative

```
public ArrayList( @Range(from = 0, to = java.lang.Integer.MAX_VALUE) int initialCapacity) {
    if (initialCapacity > 0) {
        this.elementData = new Object[initialCapacity];
    } else if (initialCapacity == 0) {
        this.elementData = EMPTY_ELEMENTDATA;
    } else {
        throw new IllegalArgumentException("Illegal Capacity: "+
                                         initialCapacity);
    }
}
```

Constructs a list containing the elements of the specified collection, in the order they are returned by the collection's iterator.

Params: `c` – the collection whose elements are to be placed into this list

Throws: `NullPointerException` – if the specified collection is null

```
public ArrayList( @NotNull @Flow(sourceContainers = true, targetContainers = true) Collection<? extends E> c) {
    Object[] a = c.toArray();
    if ((size = a.length) != 0) {
        if (c.getClass() == ArrayList.class) {
            elementData = a;
        } else {
            elementData = Arrays.copyOf(a, size, Object[].class);
        }
    } else {
        // replace with empty array.
        elementData = EMPTY_ELEMENTDATA;
    }
}
```

하지만..

자바 5 이상부터 오토박싱이 도입되었다!

하지만..

어떤 결과가 나올까?

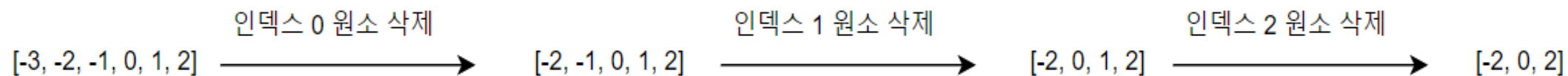
```
public class SetList {  
    public static void main(String[] args) {  
        Set<Integer> set = new TreeSet<>();  
        List<Integer> list = new ArrayList<>();  
  
        for (int i = -3; i < 3; i++) {  
            set.add(i);  
            list.add(i);  
        }  
        for (int i = 0; i < 3; i++) {  
            set.remove(i);  
            list.remove(i);  
        }  
        System.out.println(set + " " + list);  
    }  
}
```


하지만..

[-3, -2, -1] [-2, 0, 2]

```
public class SetList {  
    public static void main(String[] args) {  
        Set<Integer> set = new TreeSet<>();  
        List<Integer> list = new ArrayList<>();  
  
        for (int i = -3; i < 3; i++) {  
            set.add(i);  
            list.add(i);  
        }  
        for (int i = 0; i < 3; i++) {  
            set.remove(i);  
            list.remove(i);  
        }  
        System.out.println(set + " " + list);  
    }  
}
```

왜 [-2, 0, 2]..?



Remove쿤..어째서..?

원래 바랬던 것은 원소 내용을 기준으로

없애는 remove(Object)였으나,

For문 안의 타입이 int이기 때문에

Index를 기준으로 없애는 remove(int)가 컴파일 시점에

정해졌기 때문

Removeくん..어째서..?

**자바 4 이전에는 제네릭도 없었고 오토박싱도 없었으나,
자바 5 이후로는 제네릭과 오토박싱이 도입되어
더 이상 Object와 int가 근본적으로
다른 타입이 아니기 때문!**

메소드 참조..?

둘 다 runnable을

받는 생성자, 메소드

→그런데 컴파일 에러?

```
public static void main(String[] args) {  
    new Thread(System.out::println).start();  
  
    ExecutorService es = Executors.newCachedThreadPool();  
  
    es.submit(System.out::println);  
}
```

메소드 참조..?

Submit의 경우

Callable, Runnable로

오버로딩

```
@NotNull  
<T> Future<T> submit( @NotNull Callable<T> task);
```

Submits a Runnable task for execution and returns a Future representing that task. The Future's get method will return the given result upon successful completion.

Params: task – the task to submit
result – the result to return

Returns: a Future representing pending completion of the task

Throws: `RejectedExecutionException` – if the task cannot be scheduled for execution
`NullPointerException` – if the task is null

```
@NotNull  
<T> Future<T> submit( @NotNull Runnable task, T result);
```

Submits a Runnable task for execution and returns a Future representing that task. The Future's get method will return null upon *successful* completion.

Params: task – the task to submit

Returns: a Future representing pending completion of the task

Throws: `RejectedExecutionException` – if the task cannot be scheduled for execution
`NullPointerException` – if the task is null

```
@NotNull  
Future<?> submit( @NotNull Runnable task);
```

Executes the given tasks, returning a list of Futures holding their status and results when all complete. `Future.isDone` is true for each element of the returned list. Note that a *completed* task could have

메소드 참조..?

System.out.println의 경우는 void니까 Runnable 아닌가?

이 정도는 쉽게 추론할 수 있지 않나?

메소드 참조..?

System.out.println의 경우도 역시 오버로딩

Submit도 오버로딩 되었기 때문에

기대하는 동작이 이루어지지 않음

→오버로딩 추론 규칙이 복잡해짐

메소드 참조..?

**비록 서로 다른 함수형 인터페이스라도 파라미터 위치가
같으면 혼란이 발생한다**

서로 다른 함수형 인터페이스라도 **근본적으로 다르지 않다는 뜻**

메소드 참조..?

근본적으로 다르지 않다는 뜻은 서로서로 캐스팅이 된다는 뜻

그렇다면, System.out이 Runnable과 Callable 둘 다

Implements한다는 의미?

→ 내부 코드에서는 그렇지 않음

In java..

```
@Contract(pure = true)
public boolean contentEquals( @NotNull StringBuffer sb) {
    return contentEquals((CharSequence)sb);
}
```

```
Returns: true if this String represents the same sequence of char values as the specified sequence,
         false otherwise
Since: 1.5

public boolean contentEquals( @NotNull CharSequence cs) {
    // Argument is a StringBuffer, StringBuilder
    if (cs instanceof AbstractStringBuilder) {
        if (cs instanceof StringBuffer) {
            synchronized(cs) {
                return nonSyncContentEquals((AbstractStringBuilder)cs);
            }
        } else {
            return nonSyncContentEquals((AbstractStringBuilder)cs);
        }
    }
    // Argument is a String
```

In java..

**이번 아이템의 주제와 반대되지만,
단순히 캐스팅을 한 후 동일한 동작을 하고 있으므로
큰 문제는 없다**

In java..

아이템에서 제시하는
문제 발생 가능

Returns the string representation of the `Object` argument.

Params: `obj` – an `Object`.

Returns: if the argument is `null`, then a string equal to `"null"`; otherwise, the value of `obj`.
`toString()` is returned.

See Also: `Object.toString()`

```
public static String valueOf(Object obj) { return (obj == null) ? "null" : obj.toString(); }
```

Returns the string representation of the `char` array argument. The contents of the character array are copied; subsequent modification of the character array does not affect the returned string.

Params: `data` – the character array.

Returns: a `String` that contains the characters of the character array.

`@NotNull` `@Contract(pure = true)`

```
public static String valueOf( @NotNull char data[]) { return new String(data); }
```

In java..

아이템에서 제시하는
문제 발생 가능

```
public class Foo {  
    private Object some = new char[] {'1', '2', '3'};  
  
    public Object getSome() {  
        return some;  
    }  
}  
  
public static void main(String[] args) {  
    System.out.println(String.valueOf(new Foo().getSome()));  
}  
"C:\Program Files (x86)\Java\jdk1.8.0_311\bin\java.exe" ..  
[C@16d3586  
  
Process finished with exit code 0
```

결론

**무조건 다중정의를 이용하기 보다는,
아이템에서 제시한 여러 조건들을 고려했을 때
다중정의를 쓰지 않을 수 있다면 쓰지 않는 것이 좋다**

References

Joshua Bloch, 『Effective Java 3/E』, 이복연 역 (서울 : 인사이트, 2018), pp. 312 – 319

E.O.D



Item 52 by 알파