7주차 발표 - 메서드

Method

JooYeon Han
한 주 연

Item - 49

01 매개변수가 유효한지 검사

1. 메서드 매개변수의 유효성을 검사하여 오류를 사전에 처리하여야 한다.
2. 공개 API의 메서드는 매개변수 값이 잘못됐을 때 던지는 예외를 문서화.

Item - 49

01 매개변수가 유효한지 검사

* 자바의 null 검사 기능

```
public void someMethod(Integer val) {
  Integer integer = Objects.requireNonNull(val, "매개변수가 null!");
  System.out.println(integer);
}
```

* 범위 검사 기능

```
List<String> list = List.of("a", "b", "c");
// Exception in thread "main" java.lang.IndexOutOfBoundsException:
//      Index 4 out of bounds for length 3
Objects.checkIndex(4, list.size());
```

* 단언문(`assert`) 사용

```
private void someMethod(int arr[], int length) {
    assert arr != null;
    assert length >= 0 && arr.length == length;
```

Item - 50

02 방어적 복사

객체의 허락 없이 외부에서 내부를 수정 하여서는 안된다.

클라이언트가 불변식을 깨뜨리지 않도록 방어적으로 프로그래밍 하라

Item - 50

02 방어적 복사

* 생성자로 받은 가변 매개변수로 인하여 불변식이 깨지는 경우

```
Date start = new Date();
Date end = new Date();
Period period = new Period(start, end);
end().setMonth(3);    // period의 내부를 수정했다.
```

* 해결

```
// 방어적 복사를 적용한 생성자
public Period(Date start, Date end) {
  this.start = new Date(start.getTime());
  this.end = new Date(end.getTime());

  // 유효성 검사 전에 복사해야 한다.
  if(start.compareTo(end) > 0) {
      throw new IllegalArgumentException(start + " after " + end);
  }
}
```

Item - 50

02 방어적 복사

* 접근자로 인하여 불변식이 깨지는 경우

```
Date start = new Date();
Date end = new Date();
Period period = new Period(start, end);
// period의 인스턴스를 수정하여 불편식을 해치고 있다.
period.end().setMonth(3);
```

* 해결

```
public Date start() {
  return new Date(start.getTime()); // 새로운 Date 객체를 만들어 반환
}
public Date end() {
  return new Date(end.getTime());
}
```

Item - 51

03 매서드 시그니처

- 메서드 이름을 신중히 짓자.
- 편의 메서드를 너무 많이 만들지 말자
- 매개변수 목록은 짧게 유지하자
- 매개변수 타입으로는 클래스보다는 인터페이스가 더 낫다

Item - 51

03 매서드 시그니처

* 독립된 기능을 하는 메서드로 구분

```
List<String> list = Lists.of("a", "b", "c", "d");
// 전체가 아닌 지정된 범위의 부분 리스트에서 인덱스를 찾는 경우
List<String> newList = list.subList(1, 3);  // 부분 리스트 추출
int index = newList.indexOf("b");           // 추출한 리스트에서 인덱스 찾기
```

* 도우미(Helper) 클래스

```
// Helper 클래스 적용
class HelperClass {
  String a;
  String b;
  String c;
  String d;
}

public void someMethod(HelperClass someHelper) {
```

* 빌더 패턴

```
NutritionFacts cocaCola = new NutritionFacts.Builder(240, 8).calories(100).sodium(35).
carbohydrate(27).build();
```

Item - 52

04 다중정의

재정의(overriding): 런타임에 동적으로 선택

다중 정의(overloading): 컴파일 타임에 호출 여부가 정해진다.

```
class ColectionClassifier {
  public static String classify(Set<?> set) {
    return "집합";
  }

  public static String classify(List<?> list) {
    return "리스트";
  }

  public static String classify(Collection<?> collection) {
    return "그 외"
  }

  public static void main(String[] args) {
    Collection<?>[] collections = {
      new HashSet<String>(),
      new ArrayList<Integer>(),
      new HashMap<String, String>().values()
    };

    for (Collection<?> c : collections) {
      System.out.println(classfy(c));
```

Item - 52

04 다중정의

```
public class ObjectOutputStream
    extends OutputStream implements ObjectOutput, ObjectStreamConstants
{
public void writeBoolean(boolean val) throws IOException {
    bout.writeBoolean(val);
}
public void writeByte(int val) throws IOException  {
    bout.writeByte(val);
}
public void writeShort(int val)  throws IOException {
    bout.writeShort(val);
}
```

```
public class ObjectInputStream
    extends InputStream implements ObjectInput, ObjectStreamConstants
{
public boolean readBoolean() throws IOException {
    return bin.readBoolean();
}
public byte readByte() throws IOException  {
    return bin.readByte();
}
```

Item - 53

05 가변 인수

가변 인수란?

- 명시한 타입을 0개 이상 받을 수 있는 인수
- 가변 인수 메서드를 호출하면, 인수의 개수와 길이가 같은 배열을 만들어 건네 준다.

Item - 53

05 가변 인수

* 필수 매개변수를 받아야 하는 경우

```
static int min(int firstArg, int... remainingArgs) {
  int min = firstArg;
  for (int arg : remainingArgs) {
    if (arg < min) {
      min = arg;
    }
  }
  return min;
}
```

Item - 53

05 가변 인수

* 메서드 호출의 95% 이상이 3개 이하의 인수를 사용

```
public void foo() {}
public void foo(int arg1) {}
public void foo(int arg1, arg2) {}
public void foo(int arg1, arg2, arg3) {}  // 메서드 호출의
public void foo(int arg1, arg2, arg3, int... restArg) {}
```

Item - 54

06 Null 반환 금지

* List가 비어 있을 시 Null을 반환하는 getter

```
private final List<Cheese> cheesesInStock = ...;

public List<Cheese> getCheeses() {
  return cheesesInStock.isEmpty() ? null : new ArrayList<>(cheesesInStock);
}
```

* List가 비어 있을 시 빈 컨테이너를 반환하는 getter

```
public List<Cheese> getCheeses() {
  // cheesesInStock의 값이 없다면 빈 컨테이너 반환
  return new ArrayList<>(cheesesInStock);
}
```

```
public List<Cheese> getCheeses() {
  return cheesesInStock.isEmpty() ?
  Collections.emptyList() : new ArrayList<>(cheesesInStock);
}
```

Item - 54

06 Null 반환 금지

* 배열의 경우 길이가 0인 배열을 반환

```
// 매번 새로 할당하지 않게 하는 방법
private static final Cheese[] EMPTY_CHEESE_ARRAY = new Cheese[0];

public Cheese[] getCheeses() {
  return cheesesInStock.toArray(EMPTY_CHEESE_ARRAY);
  // 다음과 같이 미리 할당하는 것은 성능을 저하한다.
  // return cheesesInStock.toArray(new Cheese[cheesesInStock.size()]);
}
```

Item - 55

07 옵셔널 반환

Optional<T>

null이 아닌 T 타입 참조를 하나 담거나 또는 아무것도 담지 않는 객체

원소를 최대 1개 가질 수 있는 불변 컬렉션이며, null-safe를 보장한다.

Item - 55

07 옵셔널 반환

* 옵셔널 사용 예 - 1

```
public static <E extends Comparable<E>> Optional<E> max(Collection<E> c) {
  if (c.isEmpty()) {
    return Optional.empty();
  }

  E result = null;
  for (E e : c) {
    if (result == null || e.compareTo(result) > 0)
      result = Objects.requireNonNull(e);
  }
  return Optional.of(result);   // null을 넣으면 안된다.
}
```

Item - 55

07 옵셔널 반환

* 옵셔널 사용 예 - 2

```
public static <E extends Comparable<E>>
  Optional<E> max(Collection<E> c) {
  return c.stream().max(Comparator.naturalOrder());
}

public static void main(String[] args) {
  List<Integer> comp = Arrays.asList(1, 5, 23, 6, 7);
  System.out.println(max(comp));  // output : Optional[23]
}
```

Item - 55

07 옵셔널 반환

메서드 이름	설명
Optional.empty()	내부 값이 비어있는 Optional 객체 반환
Optional.of(T value)	내부 값이 value인 Optional 객체 반환
Optional.ofNullable(T value)	value가 null이면, empty Optional을 반환하고, 값이 있으면 Optional.of로 생성
T get()	Optional 내의 값을 반환
boolean isPresent()	Optional 내부 값이 null이면 false, 있으면 true
Optional<T> filter()	Optional에 filter 조건을 걸어 조건에 맞을 때만 Optional 내부 값 리턴
Optional<U> map()	Optional 내부의 값을 Function을 통해 가공
T orElse(T other)	Optional 내부의 값이 null인 경우 other을 반환
T orElseGet()	Optional 내부의 값이 null인 경우 supplier을 실행한 값을 반환
T orElseThrow()	Optional 내부의 값이 null인 경우 Exception 발생

Item - 56

08 JavaDoc

태그	용도	설명
@param	모든 매개변수	매개 변수에 대한 설명을 표시한다.
@return	void가 아닌 반환	반환 값에 대한 설명(데이터 유형 및 범위 등)을 표시 할 때 사용한다.
@throws	발생할 가능성 있는 모든 예외	태그는 발생할 수 있는 예외에 대한 설명을 표시 할 때 사용한다.
@exception	발생할 가능성 있는 모든 예외	@throws 태그와 동일하게 사용한다
@code	코드용 폰트로 렌더링	Javadoc에 예제 코드 작성시 사용된다, HTML 요소나 다른 자바독 태그를 무시한다.
@literal	HTML 요소 무시	@code와 다르게 코드용 폰트로 렌더링하지 않는다.
@implSpec	구현스펙 안내	해당 메서드와 하위 클래스 사이의 계약 설명
@index	색인화	지정한 용어를 색인화할 수 있다.
@inheritDoc	상속	상위 타입의 문서화 주석 일부를 재사용한다.
@summary	요약 설명	해당 설명에 대한 요약

감사합니다

THANK YOU

JooYeon Han
한 주 연