EJ

이펙티브 자바 CP.7

작성 일시	@2023년 2월 22일 오후 8:28
최종 편집 일시	@2023년 2월 23일 오후 9:01
유형	이펙티브 자바
작성자	종현 박
참석자	

7️⃣ 메소드

▼ 선행 내용

- 메서드 설계시 주의점
  - 매개변수와 반환 값 처리
  - 메서드 시그니처 설계화, 문서화
  - 상당 부분은 메서드 뿐만 아니라 생성자에 적용된다.
- 사용성, 견고성, 유연성에 집중

▼ 49. 매개변수가 유효한지 검사하라

- 오류는 가능한 빨리 잡아야 한다.

    - 메서드와 생성자 대부분은 입력 매개변수의 값이 특정 조건을 만족하기를 바란다.
        - 인덱스 값이 음수이면 안 되며, 객체 참조는 `null` 이 아니어야 하는 시깅다.
    - 이런 제약은 반드시 문서화를 해야 하며 메서드 몸체가 시작되기 전에 검사해야 한다.
    - 오류를 발생한 즉시 잡지 못하면 해당 오류를 감지하기 어려워지고, 감지하더라도 오류 발생 지점을 찾기 어려워진다.
- 메서드가 실행되기 전에 매개변수를 체크한다면 잘못된 값이 넘어 올 때 즉각적이고 깔끔하게 예외를 처리 할 수 있다.
    - 매개변수 검사를 제대로 하지 못할 떄 문제
        - 메서드가 수행되는 중간에 모호한 예외를 던지며 실패할 수 있다.
        - 메서드가 수행되었지만 잘못된 결과를 반환한다.
        - 메서드는 문제없이 수행되었지만, 어떤 객체를 이상한 상태로 만들어서 미래의 알수 없는 시점에 문제를 일으키는 경우
        - 즉, 매개변수 검사에 실패하면 실패 원자성(아이템 76)을 어기는 결과를 낳을 수 있다.
- `public`, `protected` 메서드는 매개변수 값이 잘못됐을 때, 던지는 예외를 문서화해야 한다.
    - `@throws` 자바독 태그를 사용하면 된다. (아이템 74)
        - 주로, `IllegalArgumentException`, `IndexOutOfBoundsException`, `NullPointException` 중 하나 가 될 것이다. (아이템 72)
    - 매개변수의 제약을 문서화 한다면 그 제약을 어겼을 때 발생하는 예외도 함께 기술해야 한다.
    - 이런 간단한 방법으로 API 사용자가 제약을 지킨 가능성을 높일 수 있다.
    - ex) `mod` 연산

```
/*
 * 현재 값 mod m 값을 반환한다. 이 메서드는
 * 항상 음이 아닌 BigInteger 를 반환한다는 점에서 remainer 메서드와 다름
 *
 * @param m 계수 (반드시 양수)
 * @return 현재 값 mod m
 * @throws ArithmeticException m 이 0보다 작거나 같으면 발생
 */
public BigInteger mod (BigInteger m) {
```

```
    if (m.signum() <= 0)
        throw new ArithmeticException("계수 (m)는 양수여야 합니다");
    ...
}
```

  - 만약 `m` 이 `null` 이면 `signum` 호출시 `NullPointerException` 을 던진다.
    - 해당 내용은 메서드 설명에 없다.
      왜냐면 (개별 메서드가 아닌) `BigInteger` 클래스 수준에서 기술했기 때문이다.
    - `@Nullable` 이나 이와 비슷한 애너테이션을 사용해 특정 매개변수는 `null` 의 가능성을 알려줄 수 있지만, 표준적인 방법이 아니다.
  - 클래스 수준 주석은 그 클래스의 모든 `public` 메서드에 적용되므로 각 메서드에 일일이 기술하는 것 보다 훨씬 깔끔한 방법이다.

- `java.util.Objects.requireNonNull`
  - Java 7 에 추가된 메서드이며, 유연하고 사용하기 편하니 ,더 이상 수동으로 `null` 검사를 하지 않아도 된다. 원하는 예외 메시지도 지정할 수 있다.
  - 입력을 그대로 반환함으로 값을 사용하는 동시에 `null` 검사를 수행 할 수 있다.

```
public static <T> T requireNonNull(T obj) {
    if (obj == null)
        throw new NullPointerException();
    return obj;
}

public static <T> T requireNonNull(T obj, String message) {
    if (obj == null)
        throw new NullPointerException(message);
    return obj;
}
```

- `Objects` 의 `checkFromIndexSize`, `checkFromToIndex`, `checkIndex` 메서드
  - Java 9 에 범위 검사 기능으로 추가된 메서드
  - `null` 검사 메소드 만큼 유연하지는 않고, 예외 메시지를 지정할 수 없고, 리스트와 배열 전용으로 설계되었다.
  - 닫힌 범위 (closed range: 양 끝단 값을 포함)는 다루지 못한다.
- `public` 이 아닌 메서드라면 단언문(`assert`)을 사용해 매개변수 유호성을 검증하자
  - `public` 이 아닌 메서드라면 메서드가 호출되는 상황을 통제할 수 있음으로, 오직 유효한 값만이 메서드에 넘겨지리라는 것을 보증할 수 있다.

- ex) 재귀 정렬용 `private` 도우미 함수

```
private static void sort(long a[], int offset, int length) {
    assert a != null;
    assert offset >= 0 && offset <= a.length;
    assert length >= 0 && length <= a.length - offset;
    ...
}
```

  - 단언문들은 자신이 단언한 조건이 무조건 참이라고 선언한다는 것이다.
  - 단언문은 일반적인 유효성 검사와 다르다.
    - 실패하면 `AssertionError` 를 던진다.
    - 런타임에서 아무런 효과도, 아무런 성능 저하도 없다.
      (java 실행시 `-ea`, `-enableassertions` 플래그를 설정하면 런타임에 영향줌)

- 매개변수 유효성을 검사 규칙 예외
  - 유효성 검사 비용이 지나치게 높거나 실용적이지 않을 때, 혹은 계산과정에서 암묵적으로 검사가 수행될 때다.
  - 하지만, 암묵적 검사에 의존했다가는 실패 원자성을 해칠 수 있다.
- 생성자
  - 생성자는 “나중에 쓰려고 저장하는 매개변수의 유효성을 검사하라”는 원칙의 특수한 사례이다.
  - 생성자 매개변수의 유효성 검사는 클래스 불변식을 어기는 객체가 생성되지 않도록 해준다.
- 정리
  - 메서드나 생성자를 작성할 때면 그 매개변수들에 어떤 제약이 있을지 생각해야 한다.
  - 그 제약들을 문서화하고 메서드 시작 부분에서 명시적으로 검사해야 한다.
  - 유효성 검사시 실제 오류를 걸러낼 때 빛을 본다. (Spring 의 `@Vaild`)
  - 이번 아이템은 “매개변수에 제약을 두는게 좋다” 로 해석하면 안된다.
    사실 그 반대로 메서드는 최대한 범용적으로 설계해야 한다.

▼ 50. 적시에 방어적 복사본을 만들라

▼ 51. 메서드 시그니처를 신중하게 설계하라

▼ 52. 다중 정의는 신중히 사용하라

▼ 53. 가변인수는 신중히 사용하라

▼ 54. null 이 아닌, 빈 컬렉션이나 배열을 반환하라

▼ 55. 옵셔널 반환은 신중히 하라

▼ 56. 공개된 API 요소에는 항상 문서화 주석을 작성하라.