


# 이펙티브 자바 CP.9

🕒 작성 일시	@2023년 3월 16일 오후 11:42
🕒 최종 편집 일시	@2023년 3월 17일 오전 12:31
📄 유형	이펙티브 자바
👤 작성자	 종현 박
👥 참석자	
🗣️ 언어	

## 9 예외

- 69. 예외는 진짜 예외 상황에만 사용하라.
- 70. 복구할 수 있는 상황에는 검사 예외를, 프로그래밍 오류에는 런타임 예외를 사용하라
- 71. 필요 없는 검사 예외 사용은 피해라
- 72. 표준 예외를 사용하라
- 73. 추상화 수준에 맞는 예외를 던져라
- 74. 메서드가 던지는 모든 예외를 문서화하라
- 75. 예외의 상세 메시지에 실패 관련 정보를 담으라
- 76. 가능한 한 실패 원자적으로 만들라
- 77. 예외를 무시하지 말라

## 9 예외

### ▼ 69. 예외는 진짜 예외 상황에만 사용하라.

- 예외
  - 제대로 사용한다면 프로그램의 가독성, 신뢰성, 유지보수성이 높아진다
  - 하지만 반대로, 잘못 사용하면 그 반대의 효과를 보게 된다.
- ex) 예외를 잘 못 사용한 예

```
try {
    int i = 0;
    while(true)
        range[i++].climb();
}
```

```

} catch (ArrayIndexOutOfBoundsException e) {
}

```

- 전혀 직관적이지 않다는 사실 하나 만으로도 코드를 이렇게 작성하면 안된다. (아이템 67)
- 무한 루프를 돌다가 배열의 끝에 도달해 에러가 발생하면 끝을 내는 것이다...
- 표준적인 관용구 표현으로는  
`for (Mountain m : range) m.climb();` 이다.

- 예외를 써서 루프를 종료한 이유

- JVM 은 배열에 접근할 때마다, 경계를 넘지 않는지 검사하는데, 일반적인 반복 문도 배열 경계에 도달하면 종료한다. 따라서 이 검사를 반복문에도 명시하면 같은 일이 중복됨으로 하나를 생략한 것이다.
- 하지만 이는 3 가지 면에서 잘못된 추론이다.
  1. 예외는 예외 상황에 쓸 용도로 설계되었으므로 JVM 구현자 입장에서는 명확한 검사만큼 빠르게 만들어야 할 동기가 약하다. (최적화에 별로 신경 쓰지 않았을 가능성이 크다)
  2. 코드를 `try-catch` 블록 안에 넣으면 JVM이 적용할 수 있는 최적화가 제한된다.
  3. 배열을 순회하는 표준 관용구는 앞서 걱정한 중복 검사를 수행하지 않는다. JVM이 알아서 최적화해 없애준다.

- 즉, 예외를 사용한 쪽이 오히려 더 느리게 동작한다.

- 예외를 다른 곳(진짜 예외가 아닌 경우)에 사용한 경우

- 코드를 헛갈리게 하고 성능을 떨어뜨린다.
- 심지어, 제대로 동작하지 않을 수도 있다.
- 반복문 내 버그가 숨어 있다면 흐름 제어에 쓰인 예외가 이 버그를 숨겨 디버깅을 어렵게 할 것이다.

- 예외는 오직 예외 상황에서만 써야 한다.

절대로 일상적인 제어 흐름용으로 쓰여서는 안 된다.

- 잘 설계된 API 라면 클라이언트가 정상적인 제어 흐름에서 예외를 사용할 일이 없게 해야함.

- 특정 상태에서만 호출 할 수 있는 '상태 의존적 메서드'를 제공하는 클래스는, '상태 검사 메서드'도 함께 제공해야 한다.

- `Iterator` 인터페이스의 `next` (상태 의존적 메서드), `hasNext` (상태 검사 메서드)

```
for (Iterator<Foo> i = collection.iterator(); i.hasNext(); ) {
    Foo foo = i.next();
    ...
}
```

- 만약, `hasNext` 가 없다면

```
try {
    Iterator<Foo> i = collection.iterator();
    while(true) {
        Foo foo = i.next(0);
        ...
    }
} catch (ArrayIndexOutOfBoundsException e) {
}
```

- 굉장히 맨 처음 ex 과 비슷해보인다.
  - 반복문에 예외를 사용하면 상황하고 헷갈리며 속도도 느리고, 엉뚱한 곳에서 발생한 버그를 숨기기도 한다.
- 상태 검사 메서드 대신 사용할 수 있는 선택지도 있다.  
올바르지 않은 상태 일 때 빈 옵셔널 (아이템 55), 혹은 `null` 같은 특수 값 반환
    1. 외부 동기화 없이 **여러 스레드가 동시에 접근** 할 수 있거나, **외부 요인으로 상태가 변할** 수 있다면 옵셔널이나 특정 값을 사용한다.  
상태 검사 메서드와 상태 의존적 메서드 호출 사이에 객체의 상태가 변할 수도 있기 때문이다.
    2. 성능이 중요한 상황에서 **상태 검사 메서드가 상태 의존적 메서드의 작업 일부를 중복** 수행한다면 옵셔널이나 특정 값을 선택한다.
    3. 다른 모든 경우엔 상태 검사 메서드 방식이 조금 더 낫다.  
가독성이 더 좋고, 잘못 사용시 발견하기 쉽다.  
상태 검사 메서드 호출을 하지 않았다면 상태 의존적 메서드가 예외를 던져 버그를 들어 낼 것이다. 반면 특정 값을 검사하지 않고 지나쳐도 발견하기 어렵다.  
(옵셔널은 해당하지 않는 문제)

- ▼ 70. 복구할 수 있는 상황에는 검사 예외를, 프로그래밍 오류에는 런타임 예외를 사용하라
- ▼ 71. 필요 없는 검사 예외 사용은 피해라
- ▼ 72. 표준 예외를 사용하라
- ▼ 73. 추상화 수준에 맞는 예외를 던져라
- ▼ 74. 메서드가 던지는 모든 예외를 문서화하라
- ▼ 75. 예외의 상세 메시지에 실패 관련 정보를 담으라
- ▼ 76. 가능한 한 실패 원자적으로 만들라
- ▼ 77. 예외를 무시하지 말라