예외

Chapter 10

예외는 진짜 예외 상황에만 사용하라

- 예외는 진짜 예외 상황에서만 사용하자

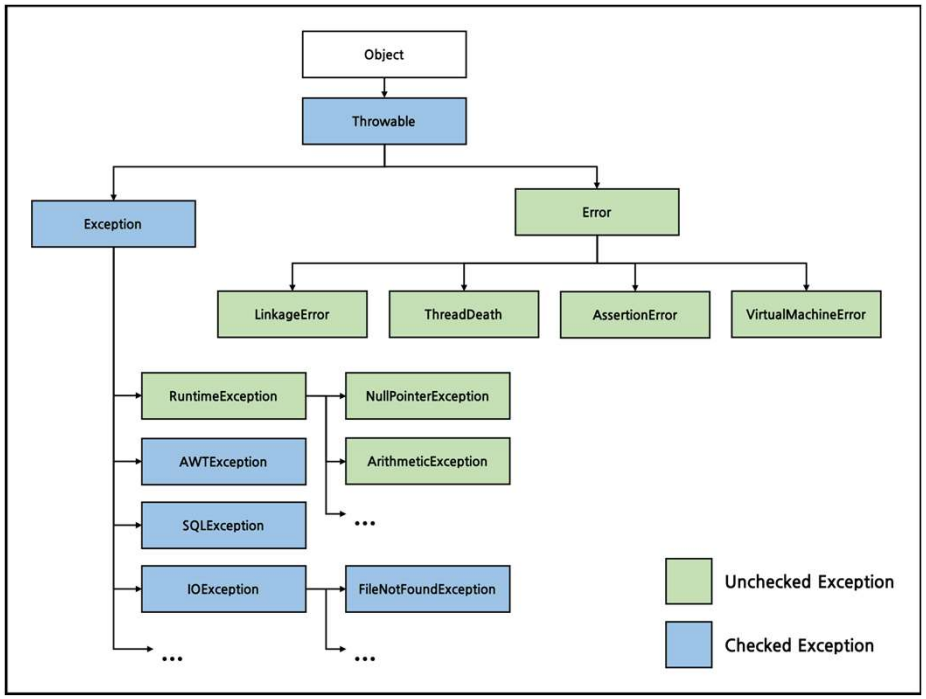
```
try {
    int i = 0;
    while(true)
        range[i++].climb();
} catch (ArrayIndexOutOfBoundsException e) {

}
```

마지막 인덱스를 넘는 경우 예외처리로 종료

1. 예외는 예외상황에서만, 명확한 검사만큼 빠르지 않다.
2. 코드를 try-catch 블록 안에 넣으면 최적화가 제한된다
3. 배열의 경우 중복 검사 수행X

검사 예외, 런타임 예외

- 복구할 수 있는 상황이면 검사 예외를, 프로그래밍 오류에는 런타임 예외를 사용하자



검사예외

- RuntimeException을 상속하지 않는 클래스

비검사예외

- RuntimeException을 상속하는 클래스
- Exception의 일종, but 명시적으로 예외 처리를 하지 않아도 됨

필요 없는 검사 예외 사용은 피하라

- 새롭게 추가하거나 단 하나의 검사 예외를 던질 때는?

코드 71-1 검사 예외를 던지는 메서드 - 리팩터링 전

```
try {
    obj.action(args);
} catch (TheCheckedException e) {
    ... // 예외 상황에 대처한다.
}
```

코드 71-2 상태 검사 메서드와 비검사 예외를 던지는 메서드 - 리팩터링 후

```
if (obj.actionPermitted(args)) {
    obj.action(args);
} else {
    ... // 예외 상황에 대처한다.
}
```

검사 예외를 던지는 메서드를
2개로 쪼개서 비검사 예외로 바꾸었다

표준 예외

- 표준 예외를 사용하고, 표준 내에서 재사용 하는 것이 좋다

예외	주요 쓰임
IllegalArgumentException	허용하지 않는 값이 인수로 건네졌을 때(null은 따로 NullPointerException으로 처리)
IllegalStateException	객체가 메서드를 수행하기에 적절하지 않은 상태일 때
NullPointerException	null을 허용하지 않는 메서드에 null을 건넸을 때
IndexOutOfBoundsException	인덱스가 범위를 넘어섰을 때
ConcurrentModificationException	허용하지 않는 동시 수정이 발견됐을 때
UnsupportedOperationException	호출한 메서드를 지원하지 않을 때

Exception, RuntimeException, Throwable, Error는 직접 재사용하지 말자!

추상화 수준에 맞는 예외를 던지라

- 추상화 수준에 맞는 예외로 바꿔 던져야 한다
  - 예외 번역 기법

```
try {
    // 저수준 추상화를 이용한다.
} catch (LowerLevelException e) {
    // 추상화 수준에 맞게 번역한다.
    throw new HigherLevelException(...);
}
```

가능하다면 저수준 메서드가 반드시 성공할 수 있도록!

상위에서 매개변수 미리 검사

메서드가 던지는 모든 예외를 문서화하라

- 각 메서드가 던지는 예외는 왠만하면 문서화하자
  - 검사 예외의 문서화

```
/**
 * blah blah...
 *
 * @param fileName
 * @throws IOException
 */
public void someMethod(String fileName) {
    try (BufferedReader br = new BufferedReader(new FileReader(fileName))) {
    } catch (IOException e) {
        // exception handling
    }
}
```

@throw 태그 사용

메서드가 던지는 모든 예외를 문서화하라

- 각 메서드가 던지는 예외는 왠만하면 문서화하자
  - 비검사 예외의 문서화

```
/**
 * blah blah...
 *
 * @param divisor
 * @throws ArithmeticException
 *     Exception may occur when divisor is zero
 */
public int someMethod(int divisor) throws ArithmeticException {
    // throws 선언에는 제외하는 것을 권장한다.
}
```

가능한 한 실패 원자적으로 만들라

- 호출한 메서드가 실패해도 호출 전 상태를 유지하는 것
  - 불변 객체로 설계
  - 로직 수행 전 매개변수의 유효성 검사

```
public Object pop() {
    if (size == 0)
        throw new EmptyStackException();
    Object result = elements[--size];
    elements[size] = null; // 다 쓴 참조 해제
    return result;
}
```

내부 상태 변경 전 잠재적 예외의 가능성을 걸러 낸다

가능한 한 실패 원자적으로 만들라

- 호출한 메서드가 실패해도 호출 전 상태를 유지하는 것
  - 실패할 가능성이 있는 모든 코드를 객체 상태를 바꾸는 코드 앞에 두자
  - 객체의 임시 복사본에서 작업 후 완료 되면 원래 객체와 교체하자

항상 실패 원자성을 지킬 수는 없다

예외를 무시하지 말라

- 예외를 선언했으면 적절한 조치를 취해야 한다

```
try {
    ...
} catch (SomeException ignored) {
    // 변수 이름은 ignored 등으로 바꾸고,
    // 예외를 무시하되 관련 로그를 남겨둔다.
}
```