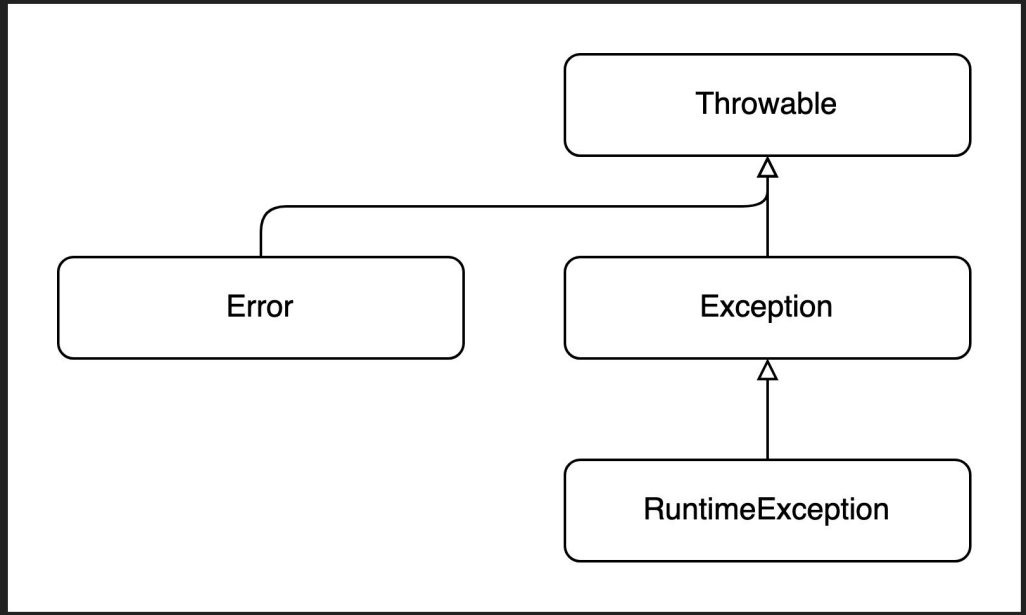
Luke's Language로 파싱된 이펙티브 자바

예쁜 예외

제작. 홍성혁

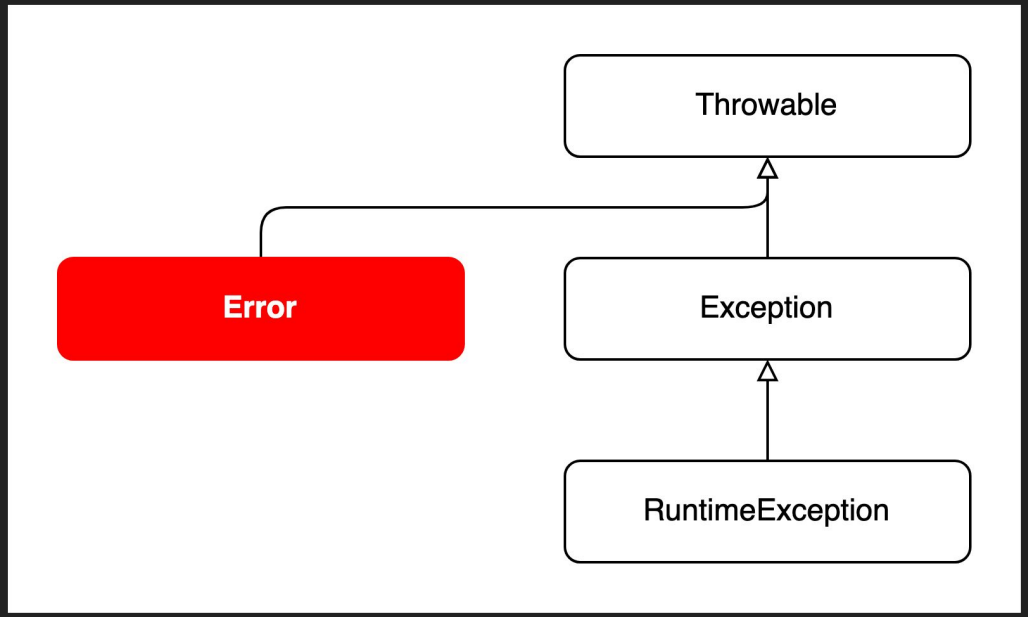
다 아는거 - 예외가 뭐에요?

1. Throwable
2. Exception
3. RuntimeException
4. Error



다 아는거 - 예외가 뭐에요?

1. Throwable
2. Exception
3. RuntimeException
4. Error



다 아는거 - 예외가 뭐에요?

Error vs Exception

1. Exception
   - 애플리케이션에서 발생할 수 있는, 복구 가능한 문제
     - 영속성, 파일이 없는 경우, 네트워크 문제
     - Checked Exception, Unchecked Exception
2. Error
   - 애플리케이션이 복구할 수 없는 심각한 문제
     - OutOfMemmoryError, StackOverFlowError

다 아는거 - 예외가 뭐에요?

Error vs Exception vs Defect

소프트웨어의 **예외**는 **결함**이 아닙니다.
예외는 소프트웨어의 시스템에 **정의해둔 기능**일 뿐 입니다.
이런 **예외**는 소프트웨어가 동작하는데 아무런 문제가 되지 않습니다.

결함은 개발자가 미처 인식하지 못한 **취약점**을 의미합니다.
결함은 ***장애***를 유발할 수 있습니다.
개발자는 소프트웨어가 **결함**을 일으키지 않도록, 소프트웨어 상에 모든 **예외**를 설계해야 합니다.

추상화 수준에 맞는 예외를 던지라

Item 73 - 예외 추상화

1. 예외 번역 Exception translation
2. 예외 연쇄 Exception chaining

추상화 수준에 맞는 예외를 던지라

Item 73 - 예외 추상화

1. 예외 번역 Exception translation

- 상위 계층에서는 저수준 예외를 잡아 자신의 추상화 수준에 맞는 예외로 바꿔 던져야 한다.
- 하위 계층의 예외를 그대로 보여주는 것은 how를 보여주는 것이다.
- 예외를 추상화하여, what을 보여주도록 한다.

코드 73-1 예외 번역

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

추상화 수준에 맞는 예외를 던지라

Item 73 - 예외 추상화

1. 예외 번역 Exception translation

- 상위 계층에서는 저수준 예외를 잡아 자신의 추상화 수준에 맞는 예외로 바꿔 던져야 한다.
- 하위 계층의 예외를 그대로 보여주는 것은 how를 보여주는 것이다.
- 예외를 추상화하여, what을 보여주도록 한다.

코드 73-1 예외 번역

```
ListIterator<E> i = listIterator(index);
try {
    return i.next();
} catch (NoSuchElementException e) {
    throw new IndexOutOfBoundsException("인덱스: " + index);
}
```

추상화 수준에 맞는 예외를 던지라

Item 73 - 예외 추상화

2. 예외 연쇄 Exception Chaining

- 추상화 수준에 맞는 예외로 바꿀 때, 저수준의 예외를 담는다.
- 근본 원인도 추적 가능하게 하는 것이 이유

코드 73-2 **예외 연쇄**

```
try {
    ... // 저수준 추상화를 이용한다.
} catch (LowerLevelException cause) {
    // 저수준 예외를 고수준 예외에 실어 보낸다.
    throw new HigherLevelException(cause);
}
```

메서드가 던지는 모든 예외를 문서화하라

Item 74 - 예외 문서화

메서드를 만들때

1. 모든 검사 예외를 선언하고,
2. 각 예외가 발생하는 상황을 자바독의 **@throws** 태그를 사용하여 정확히 문서화하자

Optional 클래스 정적 메서드인 of()의 javadoc

```
/**
 * Returns an {@code Optional} describing the given non-{@code null}
 * value.
 *
 * @param value the value to describe, which must be non-{@code null}
 * @param <T> the type of the value
 * @return an {@code Optional} with the value present
 * @throws NullPointerException if value is {@code null}
 */
public static <T> Optional<T> of( @Flow(targetIsContainer = true) T value) {
    return new Optional<>(Objects.requireNonNull(value));
}
```

메서드가 던지는 모든 예외를 문서화하라

Item 74 - 예외 문서화

1. 모든 검사 예외를 선언
- Exception이나 Throwable이 아닌, 상황에 맞는 적절한 예외를 던질 것

Optional 클래스 정적 메서드인 of()의 javadoc

```
/**
 * Returns an {@code Optional} describing the given non-{@code null}
 * value.
 *
 * @param value the value to describe, which must be non-{@code null}
 * @param <T> the type of the value
 * @return an {@code Optional} with the value present
 * @throws NullPointerException if value is {@code null}
 */
public static <T> Optional<T> of( @Flow(targetIsContainer = true) T value) {
    return new Optional<>(Objects.requireNonNull(value));
}
```

메서드가 던지는 모든 예외를 문서화하라

Item 74 - 예외 문서화

1. 모든 검사 예외를 선언
    1. 비검사 예외는 메서드에 담지 말 것
    2. 검사 예외는 모든 예외 케이스를 일일히 선언하고, 문서화 할 것
    3. 하나의 클래스에 많은 메서드들이 동일한 이유로 예외를 낸다면, 클래스 단위로 설명
        - ex) 클래스의 모든 메서드들이, 인자에 null일 경우 NPE 를 낼 때

예외의 상세 메시지에 실패 관련 정보를 담으라

Item 75 - 예외 메시지

실패의 원인을 분석해야하는 개발자나, SRE 가 얻을 수 있는 유일한 정보인 경우가 많다.

예외의 상세 메시지에 실패 관련 정보를 담으라

Item 75 - 예외 메시지

실패의 원인을 분석해야하는 개발자나, SRE 가 얻을 수 있는 유일한 정보인 경우가 많다.

잠깐, SRE가 뭐야?

사이트 안정성 엔지니어(SRE)는 시스템 가용성 , 지연 , 성능 , 효율성, 변경 관리 , 모니터링 , 비상 대응 및 용량 계획 의 조합에 대한 책임을 집니다 . [10] SRE는 종종 소프트웨어 엔지니어링 , 시스템 엔지니어링 및/또는 시스템 관리 분야의 배경을 가지고 있습니다 . [11] SRE의 초점에는 자동화 , 시스템 설계 및 시스템 복원력 개선이 포함됩니다 . [11]

SRE는 DevOps 의 구체적인 구현으로 간주됩니다 . [12] 특히 안정적인 시스템 구축에 초점을 맞추는 반면 DevOps는 더 광범위한 운영 범위를 포괄합니다. [13] [14] [15] 초점은 다르지만 일부 회사에서는 운영 팀을 SRE 팀으로 리브랜딩했습니다. [5]

예외의 상세 메시지에 실패 관련 정보를 담으라

Item 75 - 예외 메시지

실패의 원인을 분석해야하는 개발자나, SRE 가 얻을 수 있는 유일한 정보인 경우가 많다.

- 즉 사후 분석을 위해, 실패 순간에 대한 상황에 대한 정보를 상세히 담아야 한다.

    1. 발생한 예외에 관련한, 모든 매개변수와 필드 값

        - ex) IndexOutBoundsException 의 상세 메시지는
          범위의 최소와 최대값(필드), 범위를 벗어나는 인덱스 값(매개변수 값)을 담아야 한다.
        - 셋 중 어떤 것이 잘못된 것인지 모르므로!

예외의 상세 메시지에 실패 관련 정보를 담으라

Item 75 - 예외 메시지

실패의 원인을 분석해야하는 개발자나, SRE 가 얻을 수 있는 유일한 정보인 경우가 많다.

- 즉 사후 분석을 위해, 실패 순간에 대
  한다.
  1. 발생한 예외에 관련한, 모든 매개
     - ex) IndexOutBoundsException 의
       범위의 최소와 최대값(필드), 범위
     - 셋 중 어떤 것이 잘못된 것인지 또

```
/**
 * IndexOutOfBoundsException을 생성한다.
 *
 * @param lowerBound 인덱스의 최솟값
 * @param upperBound 인덱스의 최댓값 + 1
 * @param index 인덱스의 실젯값
 */
public IndexOutOfBoundsException(int lowerBound, int upperBound,
                                 int index) {
    // 실패를 포착하는 상세 메시지를 생성한다.
    super(String.format(
            "최솟값: %d, 최댓값: %d, 인덱스: %d",
            lowerBound, upperBound, index));

    // 프로그램에서 이용할 수 있도록 실패 정보를 저장해둔다.
    this.lowerBound = lowerBound;
    this.upperBound = upperBound;
    this.index = index;
}
```

예외의 상세 메시지에 실패 관련 정보를 담으라

Item 75 - 예외 메시지

실패의 원인을 분석해야하는 개발자나, SRE 가 얻을 수 있는 유일한 정보인 경우가 많다.

- 즉 사후 분석을 위해, 실패 순간에 대
  한다.
  1. 발생한 예외에 관련한, 모든 매개
     - ex) IndexOutBoundsException 의
     - 셋 중 어떤 것이 잘못된 것인지 노

=> 단, 이런 스택 추적 정보가 개인정보나 암호키를 담아서는 안될 것!

```
/**
 * IndexOutOfBoundsException을 생성한다.
 *
 * @param lowerBound 인덱스의 최솟값
 * @param upperBound 인덱스의 최댓값 + 1
 * @param index 인덱스의 실젯값
 */
public IndexOutOfBoundsException(int lowerBound, int upperBound,
                                 int index) {
    // 실패를 포착하는 상세 메시지를 생성한다.
            lowerBound, upperBound, index));

    // 프로그램에서 이용할 수 있도록 실패 정보를 저장해둔다.
    this.lowerBound = lowerBound;
    this.upperBound = upperBound;
    this.index = index;
}
```

예외의 상세 메시지에 실패 관련 정보를 담으라

Item 75 - 예외 메시지

실패의 원인을 분석해야하는 개발자나, SRE 가 얻을 수 있는 유일한 정보인 경우가 많다.

- 즉 사후 분석을 위해, 실패 순간에 대한 상황에 대한 정보를 상세히 담아야 한다.

    2. 예외가 발생한 파일 이름, 줄 번호, 스택에서 호출한 메서드의 정보를 담는다.

        - 분석하는 개발자는 보통 소스코드를 같이 살펴본다. 문서와 소스코드에서 얻을 수 있는 정보를 나열하지 않는다.
        - 가독성이 중요!
        - 최종 사용자(user)에게 보여줄 오류 메시지와는 다르다. (이건 친절하게 최종 사유만 작성한다.)

예외의 상세 메시지에 실패 관련 정보를 담으라

Item 75 - 예외 메시지

실패의 원인을 분석해야하는 개발자나, SRE 가 얻을 수 있는 유일한 정보인 경우가 많다.

- 즉 사후 분석을 위해, 실패 순간에 대한 상황에 대한 정보를 상세히 담아야 한다.

3. 접근자 메서드(get) 이용

- 저자는 예외의 접근자 메서드 이용을 적극 권장하고 있다.
- 예외의 의도를 명확히 보여주기 위함
- 고품질의 상세 메시지

E.O.D.