[item75] 예외의 상세 메세지에 실패 관련 정보를 담으라

예외 메세지
예외 메세지엔 어떤 정보가 들어가야 할까?
그렇다고 예외 메세지에 모든 것을 적으란 이야기는 아니다
꿀팁. 필요한 정보를 생성자에 받도록 하여 상세 메세지를 미리 정의 해놓는 방법

예외 메세지

- 예외를 잡지 못해 프로그램이 실패하면 자바 시스템은 그 예외의 스택 추적 (stack trace) 정보를 자동으로 출력한다
- 스택 추적은 예외 객체의 toString 메서드를 호출해 얻는 문자열로 딱히 오버라이딩 하지 않았다면 Throwable 클래스의 toString() 을 사용합니다. - 클래스 이름 + 상세 메세지

```java
public class Sample {

    public static void main(String[] args) {
        fun1();
    }

    private static void fun1() {
        fun2();
    }

    private static void fun2() {
        fun3();
    }

    private static void fun3() {
        RuntimeException ex = new RuntimeException("메롱~");
        System.out.println(ex.toString());
        throw ex;
    }

}
```


```java
481     /**
482      * Returns a short description of this throwable.
483      * The result is the concatenation of:
484      * <ul>
485      * <li> the {@linkplain Class#getName() name} of the class of this object
486      * <li> ": " (a colon and a space)
487      * <li> the result of invoking this object's {@link #getLocalizedMessage}
488      *      method
489      * </ul>
490      * If {@code getLocalizedMessage} returns {@code null}, then just
491      * the class name is returned.
492      *
493      * @return a string representation of this throwable.
494      */
495     public String toString() {
496         String s = getClass().getName();
497         String message = getLocalizedMessage();
498         return (message != null) ? (s + ": " + message) : s;
499     }
```

Throwable.java

- 스택 추적 정보가 실패 원인을 분석해야 하는 프로그래머 혹은 사이트 신뢰성 엔지니어(Site reliability engineer, SRE)가 얻을 수 있는 유일한 정보인 경우가 많고 재현하기 쉽게끔 만들기 위해서는

  → `예외의 toString()`에 `실패 원인에 관한 정보`를 `가능한 한 많이 담아 반환`해야 한다
- 위처럼 메롱~ 하면 안되고 원인을 정확히 밝혀주세요

예외 메세지엔 어떤 정보가 들어가야 할까?

- 발생한 예외에 관여된 모든 매개변수와 필드의 값을 실패 메세지에 담아야 합니다

- IndexOutOfBoundsException 가 발생했다고 했을 때 메세지엔 어떤 정보가 들어가야 할까?

  ▼ 👀

  범위의 최솟값, 최댓값 그 범위를 벗어났다는 인덱스의 값

  발생 가능한 예외상황

  - 인덱스가 최솟값보다 1만큼 작거나
  - 인덱스가 최댓값과 같거나 (인덱스는 0부터 시작하므로 최댓값이 같으면 안 된다)
  - 불변식이 깨져서 최솟값이 최댓값보다 클 수도 있다 `→ ????`

  → 모두 원인이 다르므로 예외 메세지를 잘 써주면 무엇을 고쳐야 할지 분석하는데 큰 도움이 된다.

> 💡 **비밀번호나 암호키를 예외 메세지에 표시하면 안된다**
> 문제를 진단하고 해결하는 과정에서 스택 추적 정보를 많은 사람들이 볼 수 있으므로 보안과 관련된 정보는 예외 메세지에 표시하면 안됩니다.

그렇다고 예외 메세지에 모든 것을 적으란 이야기는 아니다

- 관련 데이터를 모두 담아야 하지만 장황하면 안된다

  문제를 분석하는 사람은 스택 추적뿐 아니라 관련 문서와 소스코드도 찾아볼 수 있다.

  스택 추적에는 예외가 발생한 파일 이름, 줄번호 뿐 아니라

  스택에서 호출한 다른 메서드들의 파일 이름과 줄번호까지 나오는게 보통이다.

  문서와 소스코드에서 얻을 수 있는 정보까지 길게 예외 메세지에 장황하게 적어둘 필요는 없다

  → 👷 예외 메세지의 주 소비층은 프로그래머와 SRE 엔지니어

- 예외의 상세 메세지와 최종 사용자에게 보여줄 오류 메세지를 혼동해서는 안 된다.

  최종 사용자에게는 친절한 안내 메세지를 보여줘야 하는 반면,

예외 메세지는 가독성보다는 담긴 내용이 훨씬 중요하다

꿀팁. 필요한 정보를 생성자에 받도록 하여 상세 메세지를 미리 정의 해놓는 방법

- 현재 IndexOutOfBoundsException

  생성자에서 String (예외 메세지) 이나 int (인덱스 값) 을 받고 있다.

```
package java.lang;

public class IndexOutOfBoundsException extends RuntimeException {

    public IndexOutOfBoundsException() {
        super();
    }

    public IndexOutOfBoundsException(String s) {
        super(s);
    }

    /**
     * Constructs a new {@codeIndexOutOfBoundsException} class with an
     * argument indicating the illegal index.
     *
     * <p>The index is included in this exception's detail message.  The
     * exact presentation format of the detail message is unspecified.
     *
     * @paramindexthe illegal index.
     * @since9
     */
    public IndexOutOfBoundsException(int index) {
        super("Index out of range: " + index);
    }

}
```

- 조슈아 블로크가 강력히 권장하는 꿀팁 !!

  IndexOutOfBoundsException 의 생성자에서 최솟값, 최댓값, 인덱스를 받아 예외 메세지를 작성하는 포맷을 만들어 이용하기

```
public class IndexOutOfBoundsException extends RuntimeException {

    private final int lowerBound;
    private final int upperBound;
    private final int index;
```

```
    /**
     * Constructs an IndexOutOfBoundsException.
     *
     *@paramlowerBoundthe lowest legal index value
     *@paramupperBoundthe highest legal index value plus one
     *@paramindexthe actual index value
     */
    public IndexOutOfBoundsException(int lowerBound,int upperBound,int index) {
        // Generate a detail message that captures the failure
        super(String.format(
                "Lower bound: %d, Upper bound: %d, Index: %d",
                lowerBound, upperBound, index));

        // Save failure information for programmatic access
        this.lowerBound = lowerBound;
        this.upperBound = upperBound;
        this.index = index;
    }
}
```

→ 이렇게 해두면 프로그래머가 던지는 예외는 자연스럽게 실패를 더 잘 포착한다