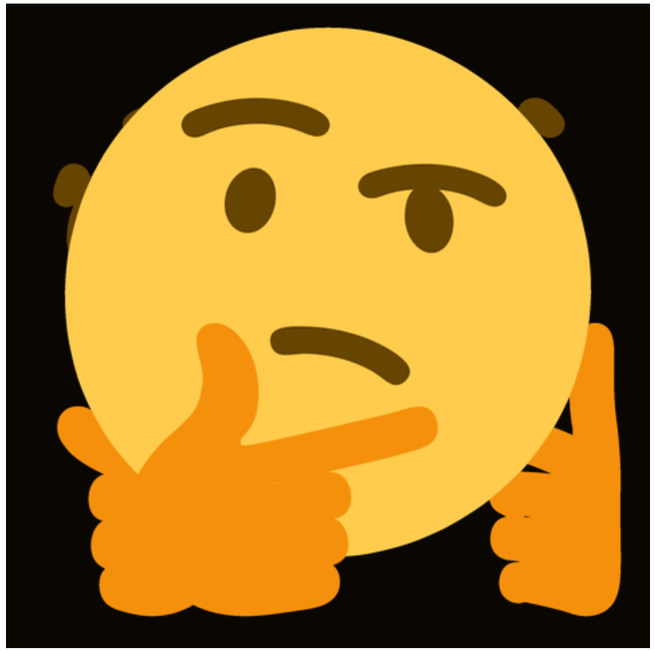
ITEM 74

메서드가 던지는 모든 예외를 문서화하라

강철원

메서드가 던지는 예외는 그 메서드를 올바로 사용하는 데 아주 중요한 정보다.



왜 예외를 문서화 해야할까?

1. 클라이언트 코드의 안정성 향상

2.API 사용성 개선

3.유지보수성 증진

1. 클라이언트 코드의 안정성 향상

예상치 못한 예외로 인한 런타임 에러를 방지하고, 애플리케이션의 안정성을 높임

```java
/**
 * 데이터베이스에서 사용자 정보를 조회합니다.
 *
 * @param userId 조회할 사용자의 ID
 * @return 사용자 정보 객체
 * @throws SQLException 데이터베이스 접근 중 오류가 발생한 경우
 * @throws UserNotFoundException 해당 ID의 사용자가 존재하지 않는 경우
 */
public User getUserById(int userId) throws SQLException, UserNotFoundException {
    // 구현 코드
}

```

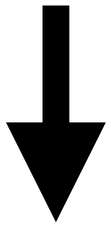
2. API 사용성 개선

명확한 예외 문서화는 API의 사용성을 높입니다.

개발자는 메서드의 동작과 예외 상황을 정확히 이해하고, 이를 기반으로 올바른 코드를 작성할 수 있습니다.

3.유지보수성 증진

코드 변경이나 확장 시에도 예외 처리 로직을 쉽게 파악



유지보수가 용이

예외 문서화 방법

🔥 예외 문서화 방법

1. 자바독(@throw)을 활용한 문서화
2. 구체적인 예외 정보 제공

1. 자바독('@throws')을 활용한 문서화

자바독의 @throws 태그를 사용하여 메서드가 던질 수 있는 예외를 명시적으로 문서화

```java
/**
 * 주어진 파일에서 데이터를 읽어옵니다.
 *
 * @param filePath 읽어올 파일의 경로
 * @return 파일에서 읽은 데이터
 * @throws IOException 파일을 읽을 수 없는 경우
 * @throws NullPointerException 파일 경로가 null인 경우
 */
public String readData(String filePath) throws IOException {
    // 구현 코드
}
```

2. 구체적인 예외 정보 제공

예외가 발생하는 상황과 원인에 대한 구체적인 정보를 제공하여 개발자가 이해하기 쉽게 만들기

```java
/**
 * 네트워크를 통해 데이터를 전송합니다.
 *
 * @param data 전송할 데이터
 * @throws IllegalArgumentException 데이터가 null이거나 빈 문자열인 경우
 * @throws NetworkException 네트워크 연결에 실패한 경우
 */
public void sendData(String data) throws IllegalArgumentException, NetworkException {
    if (data == null || data.isEmpty()) {
        throw new IllegalArgumentException("전송할 데이터가 null이거나 빈 문자열입니다.");
    }
    // 구현 코드
}
```

❗ 예외 문서화 시 주의사항

1. 구현 세부사항 노출 방지
2. 일관성 유지
3. Deprecated 예외 제거
4. 비검사 예외는 메서드 선언의 throws 목록에 포함 ❌

1. 구현 세부사항 노출 방지

❗ 내부적으로 사용하는 라이브러리의 예외를 그대로 노출하면 안 됩니다.

```java
/**
 * 데이터를 처리합니다.
 *
 * @param data 처리할 데이터
 * @throws SomeInternalLibraryException 내부 라이브러리 오류
 */
public void processData(Data data) throws SomeInternalLibraryException {
    // 구현 코드
}
```

1. 구현 세부사항 노출 방지

좋은 예시

```java
/**
 * 데이터를 처리합니다.
 *
 * @param data 처리할 데이터
 * @throws DataProcessingException 데이터 처리 중 오류가 발생한 경우
 */
public void processData(Data data) throws DataProcessingException {
    try {
        // 내부 라이브러리 호출
    } catch (SomeInternalLibraryException e) {
        throw new DataProcessingException("데이터 처리 중 오류가 발생했습니다.", e);
    }
}
```

2. 일관성 유지

📌 모든 메서드에 대해 일관된 방식으로 예외를 문서화하여 API 사용자에게 예측 가능한 정보를 제공

📌 예외의 이름, 메시지, 문서화 스타일 등을 통일

3. Deprecated 예외 제거

📌 사용하지 않는 예외나 더 이상 던지지 않는 예외는 문서에서 제거하여 혼란을 방지

📌 코드 리팩토링 시 예외 문서화도 함께 업데이트해야 합니다.

4.비검사 예외는 메서드 선언의 throws 목록에 포함 ❌

why?

1️⃣ 비검사 예외의 특성 : 비검사 예외는 주로 프로그래밍 오류나 예측 불가능한 런타임 상황을 나타냄

👉이러한 예외는 호출자가 직접 처리하기보다는 애플리케이션의 전역 예외 처리기에 의해 처리

2️⃣ 코드의 간결성과 가독성 : 모든 비검사 예외를 throws 절에 포함시키면 메서드 선언이 불필요하게 길어지고 복잡

3️⃣ 일관성 있는 관례 준수 : 자바 표준 라이브러리와 대부분의 자바 코드는 비검사 예외를 throws절에 명시 ❌

👉 이를 따름으로써 코드의 일관성을 유지

4.비검사 예외는 메서드 선언의 throws 목록에 포함 ❌

잘못된 방법 - 비검사 예외를 throws절에 포함시킨 경우

```java
/**
 * 문자열을 정수로 변환합니다.
 *
 * @param input 변환할 문자열
 * @return 변환된 정수 값
 * @throws NumberFormatException 문자열이 유효한 정수가 아닌 경우
 */
public int parseInt(String input) throws NumberFormatException {
    return Integer.parseInt(input);
}
```

4.비검사 예외는 메서드 선언의 throws 목록에 포함 ❌

올바른 방법

```java
/**
 * 문자열을 정수로 변환합니다.
 *
 * @param input 변환할 문자열
 * @return 변환된 정수 값
 * @throws NumberFormatException 문자열이 유효한 정수가 아닌 경우
 */
public int parseInt(String input) {
    return Integer.parseInt(input);
}

```

📚 마무리

📌 메서드가 던지는 모든 예외를 철저히 문서화하는 것은 견고한 API 설계의 핵심

👉 클라이언트 코드의 안정성 향상

👉 API의 사용성 개선

👉 유지보수성 증진

📌 메서드의 동작과 예외 상황을 정확히 이해

👉 안정하고 효율적인 코드를 작성

📌 예외 문서화는 단순히 예외 클래스를 나열하는 것 ❌

👍 예외가 발생하는 상황, 원인, 그리고 해결 방안까지 고려