예외처리

예외 처리란?

예외 처리는 프로그램이 **잘못된 행동을 하려고 할 때** 그 문제를 **잡아서** 프로그램이 멈추지 않도록 도와주는 방법이에요. 만약 예외 처리가 없으면, 프로그램이 **갑자기 멈출 수** 있어요. 예외 처리를 통해 프로그램이 **문제가 생겨도 계속 작동**할 수 있게 하는 것이 중요합니다.

예외 처리의 주요 키워드

1. `try`: 여기에는 **문제가 생길 수 있는 코드**가 들어갑니다. 예를 들어, 0으로 나누기 같은 연산에서 문제가 발생할 수 있어요.
2. `catch`: 문제가 발생하면, 이 부분에서 **그 문제를 처리**해요. 문제가 없을 경우 이 블록은 실행되지 않습니다.
3. `finally`: 문제가 발생하든 말든, **항상 실행되는 코드**가 들어갑니다. 자원 정리나 로그 기록 같은 마무리 작업에 주로 사용됩니다.

코드 예시

자바스크립트에서는 **0으로 나누기**가 오류로 간주되지 않고, 결과가 **Infinity**로 처리됩니다. 그래서 `catch` 블록이 실행되지 않아요. 이 문제를 해결하려면, 명시적으로 예외를 발생시켜야 합니다.

수정된 코드

```
function divideNumbers() {
    try {
        let number = 10;
        let zero = 0;

        if (zero === 0) {
            throw new Error("0으로는 나눌 수 없습니다!");  // 명시적으로 오류를 던집니다.
        }
```

```
        let result = number / zero;  // 자바스크립트에서는 실제 오류 발생하지 않음
        console.log("결과: " + result);
    } catch (e) {
        console.log("문제가 생겼어요: " + e.message);  // 던진 오류 메시지 출력
    } finally {
        console.log("프로그램이 끝났어요.");
    }
}

divideNumbers();
```

설명

1. `try`: 여기에서 **10을 0으로 나누려는 코드**가 실행됩니다. 자바스크립트에서는 0으로 나누면 오류가 발생하지 않으므로, 직접 오류를 던집니다.
2. `catch`: 오류를 던진 후 `catch` **블록**에서 이를 잡아 **"0으로는 나눌 수 없습니다!"**라는 메시지를 출력합니다.
3. `finally`: 오류 발생 여부와 상관없이 **항상 실행**되어 **"프로그램이 끝났어요."**라는 메시지가 출력됩니다.

REST API를 사용하여 프론트엔드에서 백엔드로 데이터 전송하기

이제 프론트엔드에서 백엔드로 데이터를 전송할 때 **REST API**를 어떻게 사용하는지 보여드릴게요. JavaScript에서는 주로 **`fetch`**를 사용하여 데이터를 서버로 보내고, 예외 처리를 통해 오류 상황을 관리할 수 있습니다.

REST API 데이터 전송 예시 (JavaScript -> 프론트엔드 → 백엔드)

1. 프론트엔드 (JavaScript)

프론트엔드에서 데이터를 서버로 전송할 때, 오류가 발생하면 이를 `catch`로 처리하고, 요청이 완료되면 `finally`를 통해 완료 메시지를 출력할 수 있어요.

```
async function sendData() {
    const data = {
        name: "Alice",
        age: 25
```

```
    };

    try {
        const response = await fetch('<http://localhost:8080/api/users>', {
            method: 'POST',
            headers: {
                'Content-Type': 'application/json'
            },
            body: JSON.stringify(data)
        });

        if (!response.ok) { // 서버에서 오류 응답이 오면 예외로 처리
            throw new Error(`서버 오류: ${response.status}`);
        }

        const result = await response.json();
        console.log("서버 응답:", result);
    } catch (error) {
        console.error("데이터 전송 중 문제가 발생했습니다:", error.message);
    } finally {
        console.log("데이터 전송 작업이 완료되었습니다.");
    }
}

sendData();
```

설명

1. `try`: 데이터를 `fetch`로 서버에 보내는 코드가 들어갑니다. 서버로 데이터를 **POST** 방식으로 전송하고, 서버가 응답을 줄 때까지 기다립니다. 오류가 없다면, 서버에서 받은 데이터를 처리합니다.
2. `catch`: 만약 네트워크 오류나 서버 오류가 발생하면, `catch`에서 이 오류를 처리하여 콘솔에 출력합니다.

3. `finally`: 데이터 전송 작업이 완료되면 **항상 실행**됩니다. 여기에서는 작업 완료 메시지를 출력합니다.

백엔드 (Spring Boot)

백엔드에서 데이터를 받는 방법은 **Spring Boot**를 사용하여 REST API 엔드포인트를 만들어 처리할 수 있습니다.

```
import org.springframework.web.bind.annotation.*;

@RestController
@RequestMapping("/api")
public class UserController {

    @PostMapping("/users")
    public ResponseEntity<String> createUser(@RequestBody U
ser user) {
        System.out.println("받은 데이터: " + user.getName() +
", 나이: " + user.getAge());

        return ResponseEntity.ok("데이터가 성공적으로 전송되었습니
다.");
    }
}

// 사용자 데이터를 받는 DTO 클래스
class User {
    private String name;
    private int age;

    // Getter와 Setter 메소드
    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }
```

```
    public int getAge() {
        return age;
    }

    public void setAge(int age) {
        this.age = age;
    }
}
```

요약

1. **프론트엔드에서 데이터 전송**: 자바스크립트에서 `fetch`를 사용하여 데이터를 백엔드로 전송하고, 예외가 발생하면 이를 `catch`로 처리합니다.
2. **백엔드에서 데이터 처리**: 스프링 부트에서는 **REST API**를 사용해 클라이언트로부터 데이터를 받습니다. 데이터를 처리한 후, 성공 메시지를 반환합니다.

이렇게 하면 프론트엔드에서 서버로 데이터를 안전하게 전송하고, 예외를 처리하여 안정적인 웹 애플리케이션을 만들 수 있습니다.