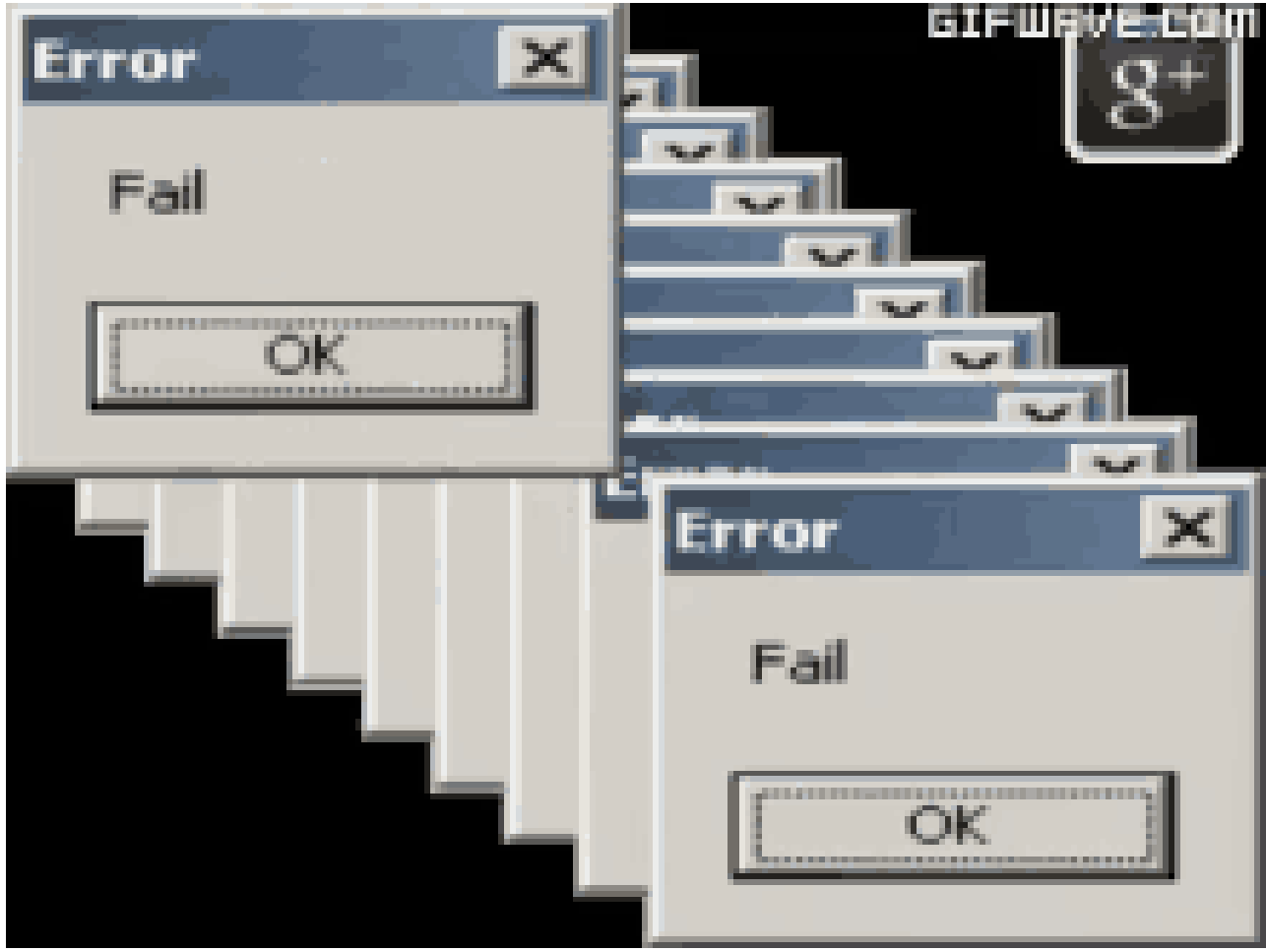
TÓPICO 16 - TRATAMENTO DE EXCEÇÕES

Disciplina de Backend - Professor Ramon Venson - SATC 2026.1

Tratamento de Exceções

Tratamento de exceções é o processo de lidar com comportamentos não ideais que podem ocorrer durante a execução de uma aplicação.



Tratamento de Exceções no Spring

O Spring possui uma estrutura de tratamento de exceções que permite definir diferentes comportamentos para diferentes tipos de exceções.

- Tratamento de exceções automáticas
- Tratamento de exceções com a anotação `@ExceptionHandler`
- Tratamento de exceções com `@RestControllerAdvice`

Tratamento de exceções automáticas

Podemos configurar o Spring para tratar automaticamente as exceções que ocorrem durante a execução de uma aplicação.

Para isso vamos adicionar a seguinte configuração no arquivo `application.properties`:

```
server.error.include-binding-errors=always
server.error.include-message=always
```

> Incluir erros de binding significa que o Spring inclui os erros de binding (erros de validação) na resposta.

Tratamento de exceções com a anotação `@ExceptionHandler`

Podemos definir um método que recebe uma exceção e retorna uma resposta personalizada, dentro de um controller:

```
@RestController
public class JogadorController {
    // ... Outros métodos do controller
    @ExceptionHandler(MethodArgumentNotValidException.class)
    @ResponseStatus(HttpStatus.BAD_REQUEST)
    public String erroValidacao() {
        return "Ocorreu um erro de validação";
    }
}
```

Este método será invocado automaticamente quando ocorrer um erro do tipo `MethodArgumentNotValidException`.

Tratamento de exceções com `@RestControllerAdvice`

Uma das melhores formas de tratar exceções é criar uma classe específica chamada `GlobalExceptionHandler` que implementa a interface `@RestControllerAdvice`.

```
@RestControllerAdvice
public class RestExceptionHandler {
    @ExceptionHandler(MethodArgumentNotValidException.class)
    @ResponseStatus(HttpStatus.BAD_REQUEST)
    public String erroValidacao() {
        return "Ocorreu um erro de validação";
    }
}
```

> Dessa forma, o Spring automaticamente invoca o método `erroValidacao` quando ocorrer um erro do tipo `MethodArgumentNotValidException`.

Criando exceções personalizadas

Podemos criar exceções personalizadas para tratarmos as exceções que ocorrem durante a execução de uma aplicação.

Para isso, vamos criar uma nova classe que herda de `RuntimeException`:

```
public class JogadorConflictException extends RuntimeException {
    public JogadorConflictException(String mensagem) {
        super(mensagem);
    }
}
```

Essa estratégia permite tratar exceções de forma mais específica de acordo com o tipo de exceção, tomando cuidado com o contexto em que a exceção ocorreu.

Invocando exceções personalizadas

Para invocar uma exceção personalizada, podemos incluir na lógica de negócio (service) uma instrução `throw`:

```
@GetMapping("/jogador/{id}")
public JogadorDto getJogador(@PathVariable UUID id) {
    Jogador jogador = jogadorService.getJogador(id);
    if (jogador == null) {
        throw new JogadorConflictException("O jogador não existe");
    }
    return jogadorMapper.toDto(jogador);
}
```

> Essa exceção será capturada pelo Spring. Se tivermos um @ExceptionHandler anotado (numa classe separada ou no próprio controller), ele será invocado.

O que aprendemos hoje

- Tratamento de exceções no Spring;
- Tratamento de exceções com a anotação `@ExceptionHandler`;
- Tratamento de exceções com `@RestControllerAdvice`.