



Desenvolvimento de Serviços com Spring Boot

Etapa/Aula 04.2

Professor(a): Flávio Neves

E-mail: flavio.neves@prof.infnet.edu.br

Roteiro da Aula

- Exemplo de validação de Status usando Exceptions.

validação de Status

- Os códigos de status de resposta HTTP indicam se uma solicitação HTTP específica foi concluída com êxito.

validação de Status

- As respostas são agrupadas em cinco classes:
 - Respostas Informativas (100 – 199)
 - Respostas bem-sucedidas (200 – 299)
 - Mensagens de redirecionamento (300 – 399)
 - Respostas de erro do cliente (400 – 499)
 - Respostas de erro do servidor (500 – 599)

https://developer.mozilla.org/pt-BR/docs/Web/HTTP>Status#respostas_informativas

Project

application-exception C:\Users\Flavio\Dropbox\INFNET\Aulas\

- > .idea
- > .mvn
- < src
 - < main
 - < java
 - com.infnet.applicationexception
 - > api
 - > exception
 - > handler
 - > model
 - < service
 - FinanceiroService
 - > resources
 - > test
 - > target
 - .gitignore
 - HELP.md
 - > mvnw
 - > mvnw.cmd
 - pom.xml
 - > External Libraries
 - > Scratches and Consoles

Project

application-exception C:\Users\Flavio\Dropbox\INFNET\Aulas\

- > .idea
- > .mvn
- < src
 - < main
 - < java
 - < com.infnet.applicationexception
 - < api
 - FinanceiroAPI
 - < exception
 - ValorZeroException
 - < handler
 - ValorZeroExceptionHandler
 - < model
 - ErrorModel
 - FinanceiroModel
 - < service
 - FinanceiroService
 - ApplicationExceptionApplication
 - > resources
 - > test

Resumo rápido dos princípios SOLID

- **S - Single Responsibility Principle (SRP):** Uma classe deve ter apenas uma única responsabilidade.
- **O - Open/Closed Principle (OCP):** Software deve estar aberto para extensão, mas fechado para modificação.
- **L - Liskov Substitution Principle (LSP):** Subtipos devem poder substituir seus tipos base sem alterar o comportamento esperado.
- **I - Interface Segregation Principle (ISP):** Muitas interfaces específicas são melhores do que uma interface única e geral.
- **D - Dependency Inversion Principle (DIP):** Dependa de abstrações, não de implementações concretas.

Vamos para o código

Análise das classes

- Single Responsibility Principle (SRP):

- ApplicationExceptionApplication
- FinanceiroService
- FinanceiroModel
- ValorZeroException
- ErrorModel
- ValorZeroExceptionHandler
- FinanceiroAPI

Análise das classes

- ValorZeroException
 - **Liskov Substitution Principle (LSP)**: Sim, pois estende RuntimeException e pode ser usada no lugar dela.
 - **Open/Closed Principle (OCP)**: Sim, por ser uma exceção específica, ela estende comportamento de exceções padrão sem.

Análise das classes

- FinanceiroAPI
 - **Dependency Inversion Principle (DIP)**: Sim, depende do serviço via injeção (@Autowired) — embora não use interface, está desacoplado da implementação concreta.
 - Poderia ser melhorada usando interfaces para FinanceiroService (DIP mais forte).

Injeção de Dependência

- É um padrão de design que permite que uma classe receba suas dependências de fora, ao invés de criar ou gerenciar diretamente essas dependências.
- Isso aumenta o desacoplamento, facilita testes e manutenção.
- No Spring, isso é feito principalmente via anotações, como **@Autowired**.

Injeção de Dependência

- Aqui, a dependência FinanceiroService é automaticamente injetada pelo Spring no controlador REST FinanceiroAPI.
- Isso significa que o Spring cuida de criar e fornecer uma instância de FinanceiroService para o FinanceiroAPI usar, sem que este precise instanciar diretamente.
- O controlador delega as operações de negócio para o serviço, mantendo-se focado na parte de controle e interface com o cliente.

```
@Autowired
```

```
private FinanceiroService financeiroService;
```

Injeção de Dependência

- Onde mais?
 - O **FinanceiroService** está anotado com **@Service**, o que indica ao Spring que essa classe é um componente gerenciado e disponível para injeção.
 - Assim, quando qualquer outra classe precisar do serviço financeiro, pode receber via injeção.

Análise das classes

- Single Responsibility Principle (SRP) é o mais bem aplicado em todas as classes: cada classe tem uma função clara e única.
- Open/Closed Principle (OCP) está implícito, mas pode ser melhor explorado em serviços e tratamento de exceções, permitindo extensão sem modificação.
- Liskov Substitution Principle (LSP) aparece na exceção personalizada, que estende RuntimeException.
- Interface Segregation Principle (ISP) não se aplica diretamente neste projeto, pois não há interfaces definidas para clientes consumirem.
- Dependency Inversion Principle (DIP) é parcialmente seguido: há injeção de dependência, mas poderia ser melhor usando interfaces para abstrair serviços.

Análise das classes