

TestsFone - Unitário

FoneControllerTests/FoneEntityTests/FoneServiceTests

ACFA | JOCOCO | 02/04/2025

# **JaCoCo 100%**

o que o espera para atingir uma cobertura próxima de **100%**:

* **Testar todos os métodos públicos** das classes de serviço e controller.
* **Cobrir cenários positivos e negativos** (sucesso e falha).
* **Incluir testes para exceções e fluxos alternativos**.
* **Testar todas as branches de decisões condicionais** (if, else, switch).
* **Garantir que todos os construtores sejam chamados nos testes**.
* **Testar DTOs, models e métodos utilitários**, mesmo que só tenham getters e setters.
* **Mockar dependências externas** corretamente para evitar falhas inesperadas.
* **Evitar código não testável**, como lógica complexa em blocos static.
* **Executar testes automatizados com frequência** para acompanhar a cobertura.

importantes:

✅ **Testar exceções esperadas**:

* Quando um método lança uma exceção, crie um teste que confirme esse comportamento.
* Use assertThrows() no JUnit para validar a exceção correta.

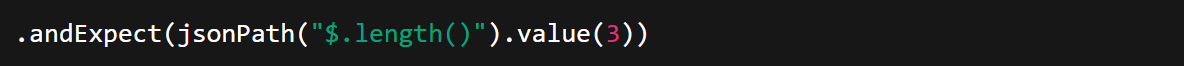
✅ **Cobrir fluxos de erro**:

* Simule cenários que levem a exceções, como entradas inválidas ou falhas no banco de dados.
* Teste o catch em blocos try-catch para garantir que o código de tratamento seja executado.

✅ **Mocks e exceções**:

* Configure Mockito.when().thenThrow() para simular falhas em serviços dependentes.

***Constatações:***  
001 - Durante a execução do teste testFindAll(), relativo a classe FoneControllerTests, foi identificado um problema de *looping infinito* durante a serialização da entidade Fone, especificamente na linha:



Esse erro estava sendo causado por uma associação bidirecional entre as entidades Pessoa e Fone. A entidade Pessoa possui uma lista de objetos Fone, e cada Fone possui uma referência de volta para a Pessoa. Essa referência cíclica gerou uma recursão infinita durante a serialização automática para JSON, realizada pelo MockMvc.

**Causa técnica:** O mecanismo de serialização do Jackson não consegue, por padrão, lidar com estruturas bidirecionais sem entrar em recursão infinita, o que leva a StackOverflowError ou falha no parse do JSON.

**Solução aplicada:** Para controlar esse comportamento, foram adicionadas as anotações @JsonManagedReference e @JsonBackReference nas respectivas entidades:

* Na classe Pessoa, foi adicionada a anotação @JsonManagedReference no lado *"pai"* da relação (a lista de telefones):



* Na classe Fone, foi utilizada a anotação @JsonBackReference no lado *"filho"* (a referência à pessoa):



**Resultado:** Com essas anotações, o Jackson serializa corretamente os objetos, evitando o ciclo infinito: ele inclui os ‘Fone’ dentro de ‘Pessoa’, mas ignora a referência reversa da ‘Pessoa’ dentro de cada ‘Fone’. Isso tornou possível executar os testes corretamente e verificar o conteúdo JSON retornado, sem erros de serialização.