Residência em Software

Módulo Programação JAVA (Avançado)

MÊS 01

INSTITUIÇÃO EXECUTORA

cepedi UESC

COORDENADORA

MCTI FUTURO Softex

APOIO

MINISTÉRIO DA CIÊNCIA, TECNOLOGIA E INOVAÇÃO

GOVERNO FEDERAL BRASIL UNIÃO E RECONSTRUÇÃO

Residência em Software

Escrevendo testes usando **JUnit e Mockito**

Parte 02

O que estar, onde e quando testar?

A solução específica dependerá dos detalhes da aplicação e de como você quer que seus testes sejam estruturados. Nossa missão é garantir que o teste está focado no que você planeja testar.

Residência em Software

Escrevendo testes usando **JUnit e Mockito**

Parte 02

Exemplo:

Em uma arquitetura típica de aplicação seguindo o padrão **MVC (Model-View-Controller)** com a **separação de responsabilidades**, a interação com o repositório (ou **DAO - Data Access Object**) para operações **CRUD (Create, Read, Update, Delete)** e outras operações de banco de dados deve ser feita pela camada de serviço. A camada de serviço age como **um intermediário** entre a **camada de apresentação** (controladores, APIs) e a **camada de acesso a dados** (repositórios), encapsulando a lógica de negócios da aplicação.

Residência em Software

Escrevendo testes usando **JUnit e Mockito**

Parte 02

Nesse contexto, temos:

Camada de Repositório (EmployeeRepository): Responsável pela comunicação direta com o banco de dados. Ela utiliza o Spring Data JPA (ou qualquer outra tecnologia de persistência) para realizar operações de banco de dados. Esta camada não deve conter **lógica de negócios**, apenas lógica para acessar, inserir, atualizar e deletar dados do banco.

Residência em Software

Escrevendo testes usando **JUnit e Mockito**

Parte 02

Camada de Serviço (EmployeeService): Contém a lógica de negócios e chama a camada de repositório quando necessário. É responsável por realizar operações de mais alto nível, como validações, transformações de dados, chamadas a outros serviços ou repositórios, etc. A camada de serviço é onde você decidiria, por exemplo, o que fazer se um nome de funcionário já existir no banco de dados antes de tentar salvá-lo.

Residência em Software

Escrevendo testes usando **JUnit e Mockito**

Parte 02

Controladores/APIs: Chamam a camada de serviço para processar requisições e devolver respostas ao usuário ou sistema cliente. Esta camada não deve conter lógica de negócios diretamente; em vez disso, deve delegar operações de negócios complexas para a camada de serviço.

Residência em Software

Escrevendo testes usando **JUnit e Mockito**

Parte 02

No contexto dos testes unitários com **mock**, o que fazemos é simular o comportamento esperado do repositório em resposta a certas ações. Por exemplo, ao verificar a unicidade do nome envolveria mais a lógica interna do repositório (e, por extensão, do banco de dados) do que do serviço em si, já que a classe **Service** delega essa responsabilidade.

Residência em Software

Escrevendo testes usando **JUnit e Mockito**

Parte 02

Assim, é correto que os métodos na classe **EmployeeService** estejam chamando a classe **EmployeeRepository** para realizar operações de banco de dados. A classe **Service** deve conter a lógica de negócios e chamar o **repositório** para acessar ou modificar os dados.

Residência em Software

Escrevendo testes usando **JUnit e Mockito**

Parte 02

Testes de Serviço

Nos testes de serviço, como EmployeeServiceTest, devemos focar em testar a lógica de negócios implementada na camada de serviço. Isso normalmente envolve:

- Simular chamadas ao repositório (EmployeeRepository) usando mocks.
- Verificar se a lógica de negócios do serviço está correta, dadas as respostas simuladas do repositório.
- Garantir que o serviço lide adequadamente com várias situações, como entradas válidas e inválidas, e lançamento de exceções esperadas.

Nesses testes, **não estamos testando a interação real com o banco de dados**. Em vez disso, testamos como o serviço responde com base nas simulações (mocks) das chamadas ao repositório.

Residência em Software

Escrevendo testes usando **JUnit e Mockito**

Parte 02

Testes de Repositório

Os testes de repositório visam verificar se nossas consultas ao banco de dados, realizadas através do repositório, estão corretas. Isso geralmente envolve:

- Executar testes utilizando um banco de dados em memória (como H2) ou um banco de dados de teste configurado.
- Verificar se as operações de CRUD funcionam como esperado.
- Testar consultas específicas do repositório para garantir que elas retornem os resultados esperados dada uma base de dados conhecida.

Esses testes devem ser separados dos testes de serviço, por terem focos diferentes. Enquanto os testes de serviço verificam a lógica de negócios sem se preocupar com os detalhes de como os dados são armazenados ou recuperados, os testes de repositório focam especificamente nessa interação com o banco de dados.

Residência em Software

Escrevendo testes usando **JUnit e Mockito**

Parte 02

Importância do uso do Faker nos testes:

Utilizar o Faker para gerar dados nos testes é uma abordagem incentivada, por ele ajudar a criar testes mais dinâmicos e flexíveis, minimizando a chance de testes que falham ou passam por acaso devido a dados hardcoded(“escritos de uma forma fixa”).
Além disso, o uso do Faker pode tornar os testes mais legíveis e fáceis de manter, ao evitar a repetição de blocos de código para a criação de objetos de teste.

Residência em Software

Escrevendo testes usando **JUnit e Mockito**

Parte 02

Seguindo as **melhores práticas** mantemos a **lógica de acesso a dados** separada da **lógica de negócios**. Isso facilita a **manutenção e a testabilidade** do código, além de permitir uma maior flexibilidade para mudanças tanto na lógica de negócios quanto nas operações de banco de dados.

Residência em Software

Contato

rogerio.jesus@cepedi.org.br

https://moodle.residenciatic18.cepedi.org.br/