Disciplina: **Teste**

Antonio Carlos Ferreira de Almeida - RA:2438194

Aberto: segunda, 20 fev 2023, 08:00
Vencimento: domingo, 5 mar 2023, 23:59
Teste de integração

1. Apresente as diferença entre testes de unidade e integração.
2. Como limpar o banco de dados durante os testes de integração?
3. Quais as vantagens e/ou desvantagens de usar Scripts para popular o banco durante os testes de integração?
4. Devemos testar até métodos simples (por exemplo, salvar um usuário utilizando o Hibernate) durante os testes de integração?

Entregar em .pdf.

Respostas:

01. Apresente as diferenças entre testes de unidade e integração.

Ambos são importantes para garantir a qualidade do software, mas diferem em seus objetivos, escopo, dependências, tempo de execução e ambiente, dentre outros.

Objetivos: O objetivo dos testes de unidade é verificar se as unidades individuais de código (métodos, classes, etc.) estão funcionando corretamente. Já o objetivo dos testes de integração é verificar se as diferentes unidades de código estão funcionando bem juntas e se a integração entre elas está funcionando corretamente.

Escopo: Os testes de unidade têm um escopo limitado e focam em testar apenas uma única unidade de código. Eles normalmente são executados dentro do ambiente de desenvolvimento, com o objetivo de encontrar erros rapidamente e facilitar a depuração do código. Já os testes de integração têm um escopo mais amplo, testando a interação entre diferentes unidades de código, como diferentes classes, módulos ou até mesmo sistemas inteiros.

Dependências: Os testes de unidade são realizados em unidades de código isoladas, geralmente com a ajuda de ferramentas de mock ou stub para simular o comportamento de outras unidades de código que a unidade testada depende. Já os testes de integração testam a interação real entre as diferentes unidades de código, sem a utilização de ferramentas de mock ou stub.

Tempo de execução: Os testes de unidade geralmente são rápidos de serem executados, pois testam apenas uma unidade de código. Já os testes de integração podem ser mais lentos e demorados, pois envolvem a interação entre diferentes unidades de código, o que pode exigir mais recursos e tempo para serem executados.

Ambiente de execução: Os testes de unidade são executados no ambiente de desenvolvimento, geralmente utilizando ferramentas como JUnit ou TestNG. Já os testes de integração podem ser executados em diferentes ambientes, como um ambiente de teste dedicado ou um ambiente de pré-produção.

02. Como limpar o Banco de dados durante os testes de integração?

Para limpar um banco de dados durante os testes de integração em Java, há algumas abordagens que podem ser adotadas:

Uma abordagem comum é usar um banco de dados separado para testes, que é criado e destruído antes e depois dos testes. Essa abordagem garante que os dados de teste não interferirão nos dados reais do banco de dados.

Outra abordagem é usar transações para garantir que os dados criados durante os testes sejam revertidos após cada teste. Isso pode ser feito usando a anotação @Transactional nos testes, que garante que cada teste seja executado em uma transação separada e que a transação seja revertida ao final do teste.

Outra abordagem é executar scripts SQL para limpar o banco de dados antes de cada teste. Isso pode ser feito usando frameworks de teste como o JUnit, que permitem a execução de código antes e depois dos testes.

03. Quais as vantagens e/ou desvantagens de usar Scripts para popular o banco durante os testes de integração?

Vantagens:

Facilidade de uso: Scripts são relativamente fáceis de criar e executar, o que pode tornar o processo de popular o banco de dados de testes mais simples.

Consistência dos dados: Usando scripts, é possível garantir que os dados utilizados nos testes sejam consistentes e padronizados, o que pode aumentar a precisão dos testes.

Reutilização: Scripts podem ser reutilizados em diferentes testes, permitindo que os dados sejam reutilizados sem a necessidade de recriá-los.

Desvantagens:

Dificuldade de manutenção: Scripts podem se tornar difíceis de manter e atualizar à medida que o sistema evolui e novas funcionalidades são adicionadas. Isso pode exigir esforços adicionais de manutenção, o que pode ser demorado e difícil.

Dificuldade de criar dados complexos: Scripts podem ser limitados na capacidade de criar dados complexos e realistas que simulam diferentes cenários. Isso pode levar a testes de integração insuficientes, o que pode reduzir a eficácia dos testes.

Conflito com dados existentes: Se não for feita uma verificação adequada, os dados utilizados nos testes podem entrar em conflito com dados existentes no banco de dados, causando problemas indesejados.

Falta de flexibilidade: Scripts podem não ser flexíveis o suficiente para lidar com situações imprevistas, o que pode limitar a capacidade de teste em certos cenários.

04. Devemos testar até métodos simples (por exemplo, salvar um usuário utilizando o Hibernate) durante os testes de integração?

Sim, é importante testar até mesmo métodos simples, como salvar um usuário utilizando o Hibernate, durante os testes de integração. Embora possa parecer trivial testar métodos simples como esse, é importante lembrar que mesmo as partes mais simples de um sistema podem ter impacto significativo no seu funcionamento geral.

Testar um método simples pode ser valioso por várias razões:

Verificação de lógica básica: Mesmo em métodos simples, pode haver algum tipo de lógica básica envolvida que precisa ser testada para garantir que o método esteja funcionando corretamente.

Identificação de problemas de integração: Testar métodos simples em conjunto com outros métodos pode ajudar a identificar problemas de integração que podem não ser evidentes quando testados isoladamente.

Garantia de que o método está funcionando como esperado: Testar métodos simples pode ajudar a garantir que eles estão funcionando corretamente e que todos os requisitos de negócios foram atendidos.

Facilitação de manutenção e refatoração: Testes de integração bem escritos para métodos simples podem ajudar a garantir que o código possa ser facilmente mantido e refatorado no futuro.