**Escrevendo o primeiro teste de integração**

Mas para criar o DAO, precisamos passar uma Session do Hibernate; e dessa vez não vamos mockar. A classe CriadorDeSessao cria a Session. Devemos então passá-la para o DAO. No teste:

@Test

public void deveEncontrarPeloNomeEEmail() {

Session session = new CriadorDeSessao().getSession();

UsuarioDao usuarioDao = new UsuarioDao(session);

Usuario usuario = usuarioDao.porNomeEEmail(

"João da Silva", "joao@dasilva.com.br");

}

Ótimo. Se tudo deu certo, espera-se que a instância usuario contenha o nome e e-mail passados. Vamos escrever os asserts então:

@Test

public void deveEncontrarPeloNomeEEmail() {

Session session = new CriadorDeSessao().getSession();

UsuarioDao usuarioDao = new UsuarioDao(session);

Usuario usuario = usuarioDao

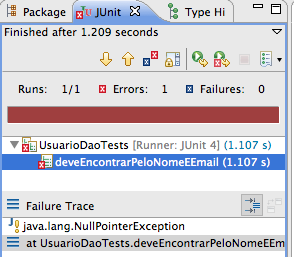
.porNomeEEmail("João da Silva", "joao@dasilva.com.br");

assertEquals("João da Silva", usuario.getNome());

assertEquals("joao@dasilva.com.br", usuario.getEmail());

}

O teste está pronto, mas se o rodarmos, ele falhará.



O motivo é simples: para que o teste passe, o usuário "João da Silva" deve existir no banco de dados! Precisamos salvá-lo no banco antes de invocar o método porNomeEEmail. Essa é a principal diferença entre testes de unidade e testes de integração: precisamos montar o cenário, executar a ação e validar o resultado esperado no software externo.

Para salvar o usuário, basta invocarmos o método salvar() do próprio DAO. Veja o código abaixo, onde criamos um usuário e o salvamos. Não podemos também esquecer de fechar a sessão com o banco de dados (afinal, sempre que consumimos um recurso externo, precisamos fechá-lo!):

@Test

public void deveEncontrarPeloNomeEEmail() {

Session session = new CriadorDeSessao().getSession();

UsuarioDao usuarioDao = new UsuarioDao(session);

// criando um usuario e salvando antes

// de invocar o porNomeEEmail

Usuario novoUsuario = new Usuario

("João da Silva", "joao@dasilva.com.br");

usuarioDao.salvar(novoUsuario);

// agora buscamos no banco

Usuario usuarioDoBanco = usuarioDao

.porNomeEEmail("João da Silva", "joao@dasilva.com.br");

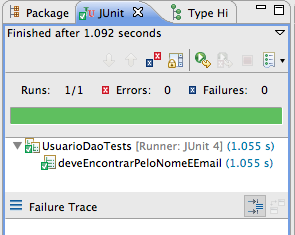
assertEquals("João da Silva", usuarioDoBanco.getNome());

assertEquals("joao@dasilva.com.br", usuarioDoBanco.getEmail());

session.close();

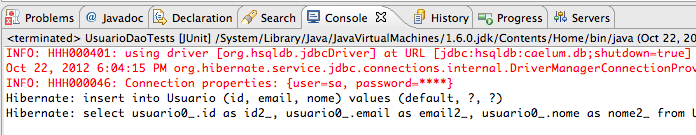
}

Agora sim, o teste passa!



Veja então que escrever um teste para um DAO não é tão diferente; é só mais trabalhoso, afinal precisamos nos comunicar com o software externo o tempo todo, para montar cenário, para validar se a operação foi efetuada com sucesso e etc. Em nosso caso, criamos uma "Session" (uma conexão com o banco), inserimos um usuário no banco (um INSERT, da SQL), e depois uma busca (um SELECT).

Isso pode inclusive ser visto pelo log do Hibernate, no console do Eclipse:



Chamamos esses testes de **testes de integração**, afinal estamos testando o comportamento da nossa classe integrada com um serviço externo real. Testes como esse são úteis para classes como nossos DAOs, cuja tarefa é justamente se comunicar com outro serviço.

**Teste de unidade x teste de integração**

Um teste de unidade isola a classe de suas dependências, e a testa independente delas. Testes de unidade fazem sentido quando nossas classes contém regras de negócio, mas dependem de infra estrutura. Nesses casos, fica fácil isolar a infra estrutura.

Já testes de integração testam a classe de maneira integrada ao serviço que usam. Um teste de DAO, por exemplo, que bate em um banco de dados de verdade, é considerado um teste de integração. Testes como esses são especialmente úteis para testar classes cuja responsabilidade é se comunicar com outros serviços.

**Ações no fim do teste**

Podemos fazer diversas coisas ao final de cada teste. Mas uma coisa que geralmente não é opcional é limpar o banco de dados para que o próximo teste consiga executar sem problemas.

Fazemos isso dando rollback no banco de dados, ou mesmo executando uma sequência de TRUNCATE TABLEs. Você pode escolher qual maneira agrada mais!