02

Organizando o teste de integração

Nossa bateria de testes de integração está crescendo, e já conseguimos reparar alguns padrões nela: em todo começo de teste, abrimos uma "Session", e no fim a fechamos. Em nosso primeiro curso, aprendemos que sempre que algo ocorre no começo e fim de todo o teste, a boa prática é não repetir código, mas sim fazer uso de métodos anotados com `@Before` e `@After`. Você se lembra? Esses métodos são executados respectivamente antes e depois de todos os testes.

Vamos lá. Criaremos dois atributos na classe, uma `Session` e um `UsuarioDao` e faremos o método com @Before instanciar esses objetos. No método com @After, fecharemos a sessão. Com isso, os métodos de testes ficarão mais enxutos:

```
public class UsuarioDaoTests {

    private Session session;
    private UsuarioDao usuarioDao;

    @Before
    public void antes() {
        // criamos a sessao e a passamos para o dao
        session = new CriadorDeSessao().getSession();
        usuarioDao = new UsuarioDao(session);
    }

    @After
    public void depois() {
        // fechamos a sessao
        session.close();
    }

    @Test
    public void deveEncontrarPeloNomeEEmail() {
        Usuario novoUsuario = new Usuario
                ("João da Silva", "joao@dasilva.com.br");
        usuarioDao.salvar(novoUsuario);

        Usuario usuarioDoBanco = usuarioDao
                .porNomeEEmail("João da Silva", "joao@dasilva.com.br");

        assertEquals("João da Silva", usuarioDoBanco.getNome());
        assertEquals("joao@dasilva.com.br", usuarioDoBanco.getEmail());
    }

    @Test
    public void deveRetornarNuloSeNaoEncontrarUsuario() {
        Usuario usuarioDoBanco = usuarioDao
                .porNomeEEmail("João Joaquim", "joao@joaquim.com.br");

        assertNull(usuarioDoBanco);
    }
}
```

Excelente. Muito melhor! E nossos testes continuam passando!

Vamos agora começar a testar nosso `LeilaoDao`. Um dos métodos desse DAO retorna a quantidade de leilões que ainda não foram encerrados. Veja:

```
public Long total() {
    return (Long) session.createQuery("select count(l) from "+
                            "Leilao l where l.encerrado = false")
            .uniqueResult();
}
```

Ele faz um simples "SELECT COUNT". Para testar essa consulta, adicionaremos dois leilões: um encerrado e outro não encerrado. Dado esse cenário, esperamos que o método `total()` nos retorne 1. Vamos ao teste.

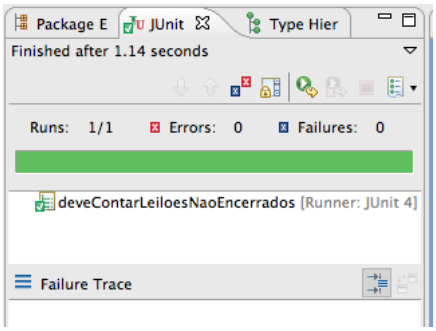
Repare que, para criar um Leilão, precisaremos criar um Usuário e persistí-lo no banco também, afinal o Leilão referencia um Usuário; para fazer isso, utilizaremos o `UsuarioDao`, que sabe persistir um `Usuario`!

Essa é uma das dificuldades de se escrever um teste de integração: montar cenário é mais difícil. Dê uma olhada no código abaixo, ele é extenso, mas está comentado:

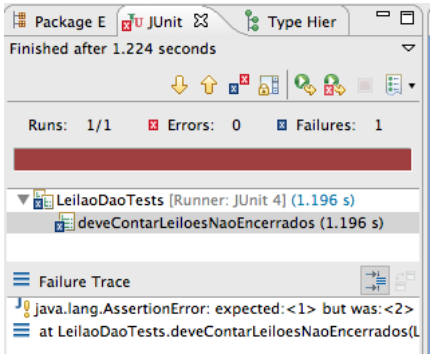
```
public class LeilaoDaoTests {
    private Session session;
    private LeilaoDao leilaoDao;
    private UsuarioDao usuarioDao;

    @Before
    public void antes() {
        session = new CriadorDeSessao().getSession();
        leilaoDao = new LeilaoDao(session);
        usuarioDao = new UsuarioDao(session);
    }

    @After
    public void depois() {
        session.close();
    }

    @Test
    public void deveContarLeiloesNaoEncerrados() {
        // criamos um usuario
        Usuario mauricio =
                new Usuario("Mauricio Aniche", "mauricio@aniche.com.br");

        // criamos os dois leiloes
        Leilao ativo =
                new Leilao("Geladeira", 1500.0, mauricio, false);
        Leilao encerrado =
                new Leilao("XBox", 700.0, mauricio, false);
        encerrado.encerra();

        // persistimos todos no banco
        usuarioDao.salvar(mauricio);
        leilaoDao.salvar(ativo);
        leilaoDao.salvar(encerrado);

        // invocamos a acao que queremos testar
```

```
        // pedimos o total para o DAO
        long total = leilaoDao.total();

        assertEquals(1L, total);
    }
}
```

Se rodarmos o teste, ele passa!



Mas esse teste ainda não está legal. Nesse momento, ele passa porque estamos rodando ele usando o banco de dados HSQLDB. Se estivéssemos rodando em um MySql, por exemplo, esse teste poderia falhar. Veja, cada vez que rodamos o teste, ele insere 2 linhas no banco de dados. Se rodarmos o teste 2 vezes, por exemplo, teremos 2 leilões não encerrados, o que faz com que o teste falhe:



A melhor maneira de garantir que, independente do banco que você esteja rodando o teste, o cenário esteja sempre limpo para aquele teste, é ter a base de dados limpa. Uma maneira simples de fazer isso é executar cada um dos testes dentro de um contexto de transação e, ao final do teste, fazer um "rollback". Com isso, o banco rejeitará tudo o que aconteceu no teste e continuará limpo.

Isso é fácil de ser implementado. Basta mexermos nos métodos `@Before` e `After` :

```
@Before
public void antes() {
    session = new CriadorDeSessao().getSession();
    leilaoDao = new LeilaoDao(session);
    usuarioDao = new UsuarioDao(session);

    // inicia transacao
    session.beginTransaction();
}

@After
public void depois() {
    // faz o rollback
```

```
        session.getTransaction().rollback();
        session.close();
    }
```

Pronto. Agora, mesmo no MySql, esse teste passaria. Iniciar e dar rollback na transação durante testes de integração é prática comum. Faça uso do "@Before" e "@After" para isso, e dessa forma, seus testes ficam independentes e fáceis de manter.