02

Mocks que lançam exceções

Nosso sistema evoluiu. Agora os usuários precisam ser avisados sobre os leilões encerrados. Para implementar essa funcionalidade, suponha uma interface `Carteiro`:

```
public interface Carteiro {
    void envia(Leilao leilao);
}
```

Precisamos fazer nossa classe `EncerradorDeLeilao` fazer uso dela, e enviar o e-mail logo após encerrar o leilão. Para isso, receberemos o Carteiro no construtor e o invocaremos logo após persistir no DAO:

```
public class EncerradorDeLeilao {

    private int total = 0;
    private final RepositorioDeLeiloes dao;
    private final Carteiro carteiro;

    public EncerradorDeLeilao(RepositorioDeLeiloes dao, Carteiro carteiro) {
        this.dao = dao;
        // guardamos o carteiro como atributo da classe
        this.carteiro = carteiro;
    }
    public void encerra() {
        List<Leilao> todosLeiloesCorrentes = dao.correntes();

        for (Leilao leilao : todosLeiloesCorrentes) {
            if (comecouSemanaPassada(leilao)) {
                leilao.encerra();
                total++;
                dao.atualiza(leilao);
                // agora enviamos por email tambem!
                carteiro.envia(leilao);
            }
        }
    }

    // ... codigo continua
}
```

Nesse momento, provavelmente todos nossos testes devam quebrar. Isso acontecerá pois agora o construtor da classe `EncerradorDeLeilao` recebe um `Carteiro`. Passe um mock de `Carteiro` para todos eles. A mudança deve ser parecida com a abaixo:

```
Carteiro carteiroFalso = mock(Carteiro.class);
EncerradorDeLeilao encerrador = new EncerradorDeLeilao(daoFalso, carteiroFalso);
```

Nosso método `encerra()` agora, além de encerrar um leilão, ainda persiste a informação na base de dados e notifica o sistema de envio de emails. Já sabemos que, sempre que lidamos com infraestrutura, precisamos nos precaver de possíveis

problemas que podem acontecer: o banco pode estar fora do ar, o SMTP pode recusar nosso envio de e-mail e assim por diante.

Nosso sistema deve saber se recuperar da falha e continuar para garantir que nosso `EncerradorDeLeilao` não pare por causa de um erro de infraestrutura. Para isso, vamos adicionar um try-catch dentro do loop e, caso o DAO lance uma exceção, o encerrador continuará a tratar os próximos leilões da lista. Vamos lá:

```
public void encerra() {
    List<Leilao> todosLeiloesCorrentes = dao.correntes();

    for (Leilao leilao : todosLeiloesCorrentes) {
        try {
            if (comecouSemanaPassada(leilao)) {
                leilao.encerra();
                total++;
                dao.atualiza(leilao);
                carteiro.envia(leilao);
            }
        }
        catch(Exception e) {
            // salvo a excecao no sistema de logs
            // e o loop continua!
        }
    }
}
```

Como sempre, vamos garantir que esse comportamento realmente funciona. Sabemos que, se passarmos dois leilões e o DAO lançar uma exceção no primeiro, ainda sim o segundo deve ser processado.

Para simular essa exceção, faremos uso do método `doThrow()` do Mockito. Esse método recebe um parâmetro: a exceção que deve ser lançada. Em seguida, passamos para ele a mesma instrução `when()` que já estamos acostumados. Veja o exemplo:

```
doThrow(new RuntimeException()).when(daoFalso).atualiza(leilao1);
```

Estamos dizendo ao mock que, quando o método `atualiza(leilao1)` for invocado no `daoFalso`, o Mockito deve então lançar a Exception que foi passada para o `doThrow`. Simples assim!

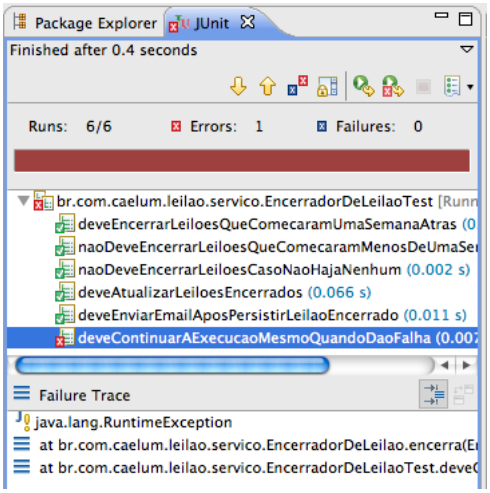
Vamos ao teste agora:

```
@Test
public void deveContinuarAExecucaoMesmoQuandoDaoFalha() {
    Calendar antiga = Calendar.getInstance();
    antiga.set(1999, 1, 20);

    Leilao leilao1 = new CriadorDeLeilao().para("TV de plasma")
        .naData(antiga).constroi();
    Leilao leilao2 = new CriadorDeLeilao().para("Geladeira")
        .naData(antiga).constroi();

    RepositorioDeLeiloes daoFalso = mock(RepositorioDeLeiloes.class);
    when(daoFalso.correntes()).thenReturn(Arrays.asList(leilao1, leilao2));
```

```
        doThrow(new RuntimeException()).when(daoFalso).atualiza(leilao1);

        EnviadorDeEmail carteiroFalso = mock(EnviadorDeEmail.class);
        EncerradorDeLeilao encerrador =
            new EncerradorDeLeilao(daoFalso, carteiroFalso);

        encerrador.encerra();

        verify(daoFalso).atualiza(leilao2);
        verify(carteiroFalso).envia(leilao2);
    }
```

Veja que ensinamos nosso mock a lançar uma exceção quando o `leilao1` for passado; mas nada deve acontecer com o `leilao2`. Ao final, verificamos que o DAO e o carteiro receberam `leilao2` (afinal, a execução deve continuar!). Nosso teste passa!

Por curiosidade, comente o try-catch e rode o teste novamente. Ele falhará:



Pronto! Garantimos que nosso sistema continua funcionando mesmo se uma exceção ocorrer! É comum termos tratamentos diferentes dado uma exceção; o Mockito faz com que esses testes sejam fáceis de serem escritos! Veja que, sem o Mockito, como faríamos esse teste? Desligaríamos o MySQL para simular banco de dados fora do ar? Mocks realmente são úteis.