02

Garantindo que os métodos foram invocados

Agora que conseguimos simular nosso banco de dados, o teste ficou fácil de ser escrito. Mas veja que ainda não testamos o método `encerra()` da classe `EncerradorDeLeilao` por completo. Veja o código dele abaixo:

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

    for (Leilao leilao : todosLeiloesCorrentes) {
        if (comecouSemanaPassada(leilao)) {
            leilao.encerra();
            total++;
            dao.atualiza(leilao);
        }
    }
}
```

Testamos que leilões são ou não encerrados, mas ainda não garantimos que os leilões são atualizados pelo DAO após serem encerrados! Como garantir que o método `atualiza()` foi invocado?

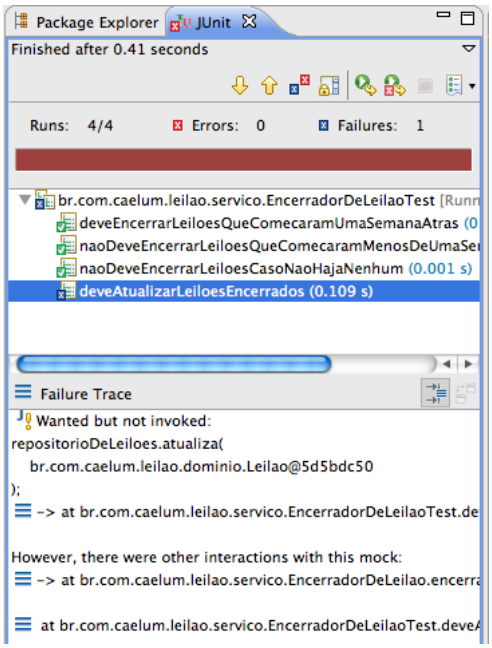
Se não estivéssemos "mockando" o DAO seria fácil: bastaria fazer um SELECT no banco de dados e verificar que a coluna foi alterada! Mas agora, com o mock, precisamos perguntar para ele se o método foi invocado!

Para isso, faremos uso do método `verify` do Mockito. Nele, indicaremos qual método que queremos verificar se foi invocado! Veja o teste abaixo:

```
@Test
public void deveAtualizarLeiloesEncerrados() {

    Calendar antiga = Calendar.getInstance();
    antiga.set(1999, 1, 20);

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

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

    EncerradorDeLeilao encerrador = new EncerradorDeLeilao(daoFalso);
    encerrador.encerra();

    // verificando que o metodo atualiza foi realmente invocado!
    verify(daoFalso).atualiza(leilao1);
}
```

Veja que passamos para o método `atualiza()` a variável `leilao1` . O Mockito é inteligente: ele verificará se o método `atualiza()` foi invocado com a variável `leilao1` sendo passada. Caso passemos um outro leilão, por exemplo, o teste falhará.

Nesse momento, nosso teste passa! Por curiosidade, se comentarmos a linha `dao.atualiza(leilao);` na classe `EncerradorDeLeilao`, o teste falhará com a seguinte mensagem:

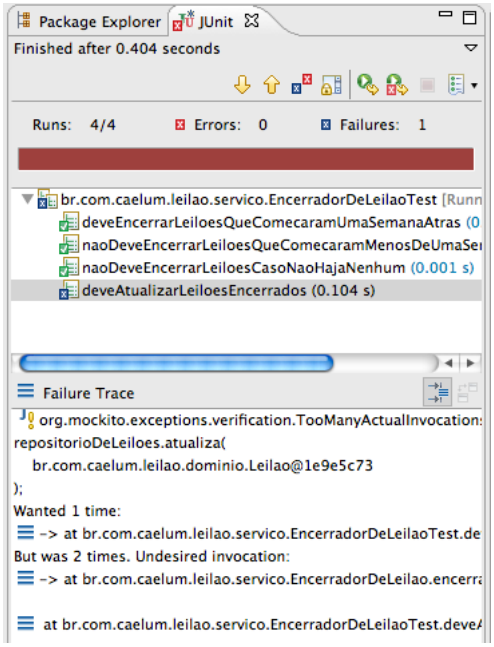


Veja que a mensagem diz que o método não foi invocado. Pronto! Dessa forma conseguimos testar a invocação de métodos.

Podemos melhorar ainda mais essa verificação. Podemos dizer ao `verify()` que esse método deve ser executado uma única vez, e que, caso ele seja invocado mais de uma vez, o teste deve falhar:

```
@Test
public void deveAtualizarLeiloesEncerrados() {

    Calendar antiga = Calendar.getInstance();
    antiga.set(1999, 1, 20);

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

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

    EncerradorDeLeilao encerrador = new EncerradorDeLeilao(daoFalso);
    encerrador.encerra();

    verify(daoFalso, times(1)).atualiza(leilao1);
}
```

Veja o `times(1)`. Ele também é um método da classe `Mockito`. Ali passamos a quantidade de vezes que o método deve ser invocado; poderia ser 2, 3, 4, ou qualquer outro número. Por curiosidade, se fizermos a classe `EncerradorDeLeilao` invocar duas vezes o DAO, nosso teste falhará. Ele nos avisará que o método foi invocado 2 vezes, o que não era esperado:



Através do `verify()`, conseguimos então testar que métodos são invocados, garantindo o comportamento de uma classe por completo.