02

Simulando comportamentos com Mock Objects

Você pode fazer o download do projeto aqui (http://bit.ly/caelum-mocks)

Escrever testes de unidade, além de divertido, é parte vital para o desenvolvimento de um software de fácil manutenção e evolução. Durante o curso, trabalharemos em cima de um sistema de leilão. Usuários podem dar lances, leilões são encerrados etc. Veja a classe abaixo, responsável por encerrar leilões:

```
public class EncerradorDeLeilao {

    private int total = 0;

    public void encerra() {

        LeilaoDao dao = new LeilaoDao();
        List<Leilao> todosLeiloesCorrentes = dao.correntes();

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

    private boolean comecouSemanaPassada(Leilao leilao) {
        return diasEntre(leilao.getData(), Calendar.getInstance()) >= 7;
    }

    private int diasEntre(Calendar inicio, Calendar fim) {
        Calendar data = (Calendar) inicio.clone();
        int diasNoIntervalo = 0;
        while (data.before(fim)) {
            data.add(Calendar.DAY_OF_MONTH, 1);
            diasNoIntervalo++;
        }
        return diasNoIntervalo;
    }

    public int getTotalEncerrados() {
        return total;
    }
}
```

Ela é razoavelmente simples de entender: esse código percorre toda a lista de leilões e, caso o leilão tenha sido iniciado semana passada, ele é encerrado e persistido no banco de dados através do DAO. Nosso DAO é convencional. Ele faz uso de JDBC e envia comandos SQL para o banco.

Pensar em cenários de teste para esse problema não é tão difícil:

- Uma lista com leilões a serem encerrados

- Uma lista sem nenhum leilão a ser encerrado
- Uma lista com um leilão um dia antes de ser encerrado
- Uma lista com um leilão no dia exato de ser encerrado

Escrever estes testes não deve ser uma tarefa tão difícil. Vamos lá:

```
public class EncerradorDeLeilaoTest {

    @Test
    public void deveEncerrarLeiloesQueComecaramUmaSemanaAtras() {

        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();

        // mas como passo os leiloes criados para o EncerradorDeLeilao,
        // já que ele os busca no DAO?

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

        assertTrue(leilao1.isEncerrado());
        assertTrue(leilao2.isEncerrado());
    }
}
```

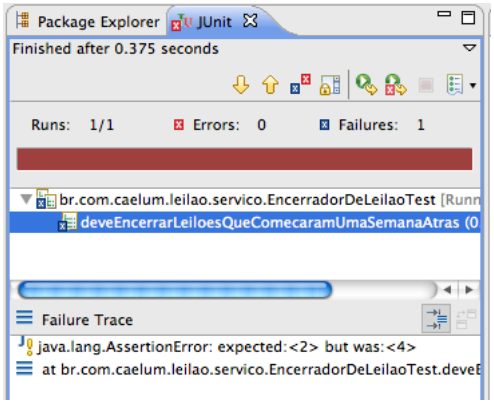
Mas o problema é: como passamos o cenário para a classe `EncerradorDeLeilao`, já que ela busca esses dados do banco de dados?

Uma solução seria adicionar todos esses leilões no banco de dados, um a um, para cada cenário de teste. Ou seja, faríamos diversos INSERTs no banco de dados e teríamos o cenário pronto lá. No nosso caso, vamos usar o próprio DAO para inserir os leilões. Além disso, já que o DAO criará novos objetos, precisamos buscar os leilões novamente no banco, para garantir que eles estão encerrados:

```
    @Test
    public void deveEncerrarLeiloesQueComecaramUmaSemanaAtras() {

        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();

        LeilaoDao dao = new LeilaoDao();
        dao.salva(leilao1);
        dao.salva(leilao2);

        EncerradorDeLeilao encerrador = new EncerradorDeLeilao(dao);
```

```
        encerrador.encerra();

        // busca no banco a lista de encerrados
        List<Leilao> encerrados = dao.encerrados();

        // vamos conferir tambem o tamanho da lista!
        assertEquals(2, encerrados.size());
        assertTrue(encerrados.get(0).isEncerrado());
        assertTrue(encerrados.get(1).isEncerrado());
    }
```

Isso resolve o nosso problema, afinal agora o teste passa! Mas isso não é suficiente: se rodarmos o teste novamente, ele agora quebra!



Isso aconteceu porque, como persistimos o cenário no banco de dados, na segunda execução do teste já existiam leilões lá! Então precisamos lembrar sempre de limpar o cenário antes do início de cada teste, dando um `DELETE` ou um `TRUNCATE TABLE`.

Veja quanto trabalho para testar uma simples regra de negócio! Isso sem contar que o teste agora leva muito mais tempo para ser executado, afinal conectamos em um banco de dados todas as vezes que rodamos o teste!

Precisamos conseguir testar essa classe `EncerradorDeLeilao` sem depender do banco de dados. A grande pergunta é: como fazer isso?

Uma ideia seria simular o banco de dados. Veja que, para a classe `EncerradorDeLeilao`, não importa como o DAO faz o serviço dela. O encerrador está apenas interessado em alguém que saiba devolver uma lista de leilões e que saiba persistir um leilão. Como isso é feito, para o encerrador pouco importa.

Vamos criar uma classe que finge ser um DAO. Ela persistirá as informações em uma simples lista:

```
public class LeilaoDaoFalso {

    private static List<Leilao> leiloes = new ArrayList<Leilao>();

    public void salva(Leilao leilao) {
        leiloes.add(leilao);
    }

    public List<Leilao> encerrados() {

        List<Leilao> filtrados = new ArrayList<Leilao>();
        for(Leilao leilao : leiloes) {
```

```
            if(leilao.isEncerrado()) filtrados.add(leilao);
        }

        return filtrados;
    }

    public List<Leilao> correntes() {

        List<Leilao> filtrados = new ArrayList<Leilao>();
        for(Leilao leilao : leiloes) {
            if(!leilao.isEncerrado()) filtrados.add(leilao);
        }

        return filtrados;
    }

    public void atualiza(Leilao leilao) { /* faz nada! */ }
}
```

Agora vamos fazer com que o `EncerradorDeLeilao` e o `EncerradorDeLeilaoTest` usem o DAO falso. Além disso, vamos pegar o total de encerrados agora pelo próprio `EncerradorDeLeilao`, já que pega pelo DAO não adianta mais (o DAO é falso!):

```
public class EncerradorDeLeilaoTest {

    @Test
    public void deveEncerrarLeiloesQueComecaramUmaSemanaAtras() {

        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();

        // dao falso aqui!
        LeilaoDaoFalso daoFalso = new LeilaoDaoFalso();
        daoFalso.salva(leilao1);
        daoFalso.salva(leilao2);

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

        assertEquals(2, encerrador.getTotalEncerrados());
        assertTrue(leilao1.isEncerrado());
        assertTrue(leilao2.isEncerrado());
    }
}

public class EncerradorDeLeilao {
    public void encerra() {

        // DAO falso aqui!
        LeilaoDaoFalso dao = new LeilaoDaoFalso();
        List<Leilao> todosLeiloesCorrentes = dao.correntes();
```

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

        // classe continua aqui...
    }
```

Rodamos o teste novamente, e pronto! Veja que agora ele rodou rápido! Nosso teste está mais simples, mas ainda assim não é ideal. Sempre que criarmos um método novo no DAO, precisaremos escrevê-lo também no `LeilaoDaoFalso`. Se precisarmos fazer testes de casos excepcionais como, por exemplo, uma exceção lançada no método `salva()`, precisaríamos de vários DAOs falsos.

Ou seja, a ideia de objetos falsos é boa; precisamos apenas encontrar uma maneira ideal de implementá-la. Objetos que simulam os comportamentos dos objetos reais é o que chamamos de mock objects. Mock objects ou, como foi traduzido para o português, objetos dublês, são objetos que fingem ser outros objetos. Eles são especialmente úteis em testes como esses, em que temos objetos que se integram com outros sistemas (como é o caso do nosso DAO, que fala com um banco de dados).

E o melhor: os frameworks de mock objects tornam esse trabalho extremamente simples! Neste curso, estudaremos o Mockito. Ele é um dos frameworks de mock mais populares do mercado.

A primeira coisa que devemos fazer é criar um mock do objeto que queremos simular. No nosso caso, queremos criar um mock de `LeilaoDao`:

```
LeilaoDao daoFalso = mock(LeilaoDao.class);
```

Veja que fizemos uso do método `mock`. Esse método deve ser importado estaticamente da classe do próprio Mockito:

```
import static org.mockito.Mockito.*;
```

Pronto! Temos um mock criado! Precisamos agora ensiná-lo a se comportar da maneira que esperamos. Vamos ensiná-lo, por exemplo, a devolver a lista de leilões criados quando o método `correntes()` for invocado:

```
        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();

        List<Leilao> leiloesAntigos = Arrays.asList(leilao1, leilao2);

        // criando o mock!
        LeilaoDao daoFalso = mock(LeilaoDao.class);
        // ensinando o mock a reagir da maneira que esperamos!
        when(daoFalso.correntes()).thenReturn(leiloesAntigos);
```

Veja que o método `when()`, também do Mockito, recebe o método que queremos simular. Em seguida, o método `thenReturn()` recebe o que o método falso deve devolver. Veja que simples! Agora, quando invocarmos `daoFalso.correntes()`, ele devolverá `leiloesAntigos`!

Vamos levar esse código agora para nosso método de teste:

```
@Test
public void deveEncerrarLeiloesQueComecaramUmaSemanaAtras() {

    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();
    List<Leilao> leiloesAntigos = Arrays.asList(leilao1, leilao2);

    // criamos o mock
    LeilaoDao daoFalso = mock(LeilaoDao.class);
    // ensinamos ele a retornar a lista de leiloes antigos
    when(daoFalso.correntes()).thenReturn(leiloesAntigos);

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

    assertTrue(leilao1.isEncerrado());
    assertTrue(leilao2.isEncerrado());
    assertEquals(2, encerrador.getQuantidadeDeEncerrados());
}
```

Mas, ao executar o teste, ele falha! Por que? Porque a classe `EncerradorDeLeilao` não faz uso do mock que criamos! Veja que ela instancia o DAO falso ainda! Precisamos fazer com que o `EncerradorDeLeilao` receba o mock na hora do teste e receba a classe de verdade quando o sistema estiver em produção.

Uma solução é receber o `LeilaoDao` no construtor. Nesse caso, o teste passaria o mock para o Encerrador:

```
public class EncerradorDeLeilao {

    private int encerrados;
    private final LeilaoDao dao;

    public EncerradorDeLeilao(LeilaoDao dao) {
        this.dao = dao;
    }

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

        for(Leilao leilao : todosLeiloesCorrentes) {
            if(comecouSemanaPassada(leilao)) {
                encerrados++;
                leilao.encerra();
```

```
                    dao.salva(leilao);
                }
            }
        }

        // codigo continua aqui
    }

    public class EncerradorDeLeilaoTest {

        @Test
        public void deveEncerrarLeiloesQueComecaramUmaSemanaAtras() {

            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();
            List<Leilao> leiloesAntigos = Arrays.asList(leilao1, leilao2);

            LeilaoDao daoFalso = mock(LeilaoDao.class);
            when(daoFalso.correntes()).thenReturn(leiloesAntigos);

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

            assertTrue(leilao1.isEncerrado());
            assertTrue(leilao2.isEncerrado());
            assertEquals(2, encerrador.getQuantidadeDeEncerrados());
        }
    }
```

Agora sim! Nosso teste passa! Ao invocar o método `encerra()`, o DAO que é utilizado é o mock; ele, por sua vez, nos devolve a lista "de mentira", e conseguimos executar o teste!

Veja que, agora, foi fácil escrevê-lo, afinal o Mockito facilitou a nossa vida! Conseguimos simular o comportamento do DAO e testar a classe que queríamos sem precisarmos montar cenários no banco de dados!

Mock objects são uma ótima alternativa para facilitar a escrita de testes de unidade para classes que dependem de outras classes!