02

Isolando para testar

Temos mais uma regra de negócio para implementar. A data do pagamento nunca pode ser de fim de semana; se "hoje" for sábado ou domingo, devemos empurrar a data para o primeiro dia útil. A implementação não é difícil: basta verificarmos o dia da semana e empurrar 1 dia se for domingo, 2 dias se for sábado.

Vamos lá:

```
public class GeradorDePagamento {

private final RepositorioDePagamentos pagamentos;
private final RepositorioDeLeiloes leiloes;
private final Avaliador avaliador;

public GeradorDePagamento(RepositorioDeLeiloes leiloes,
            RepositorioDePagamentos pagamentos,
            Avaliador avaliador) {
this.leiloes = leiloes;
this.pagamentos = pagamentos;
this.avaliador = avaliador;
}

public void gera() {

List<Leilao> leiloesEncerrados = leiloes.encerrados();
for(Leilao leilao : leiloesEncerrados) {
avaliador.avalia(leilao);

// agora empurramos para o próximo dia útil
Pagamento novoPagamento =
            new Pagamento(avaliador.getMaiorLance(), primeiroDiaUtil());
pagamentos.salva(novoPagamento);
}
}

private Calendar primeiroDiaUtil() {
Calendar data = Calendar.getInstance();
int diaDaSemana = data.get(Calendar.DAY_OF_WEEK);

if(diaDaSemana == Calendar.SATURDAY) data.add(Calendar.DAY_OF_MONTH, 2);
else if(diaDaSemana == Calendar.SUNDAY) data.add(Calendar.DAY_OF_MONTH, 1);

return data;
}
}
```

Agora vamos testar. O que faremos é verificar que a data gerada para o pagamento é uma segunda-feira. Vamos usar o `ArgumentCaptor`, que estudamos no capítulo passado:

```
@Test
public void deveEmpurrarParaOProximoDiaUtil() {
```

```
RepositorioDeLeiloes leiloes = mock(RepositorioDeLeiloes.class);
RepositorioDePagamentos pagamentos = mock(RepositorioDePagamentos.class);

Leilao leilao = new CriadorDeLeilao()
.para("Playstation")
.lance(new Usuario("José da Silva"), 2000.0)
.lance(new Usuario("Maria Pereira"), 2500.0)
.constroi();

when(leiloes.encerrados()).thenReturn(Arrays.asList(leilao));

GeradorDePagamento gerador =
            new GeradorDePagamento(leiloes, pagamentos, new Avaliador());
gerador.gera();

ArgumentCaptor<Pagamento> argumento = ArgumentCaptor.forClass(Pagamento.class);
verify(pagamentos).salva(argumento.capture());
Pagamento pagamentoGerado = argumento.getValue();

assertEquals(Calendar.MONDAY, pagamentoGerado.getData().get(Calendar.DAY_OF_WEEK));
}
```

Nosso teste falha! Veja a implementação do método `primeiroDiaUtil`. Ele verifica se o dia de hoje é sábado ou domingo. Ou seja, esse teste só passará se você estiver fazendo esse curso no fim de semana!

A pergunta é: como mudar a data atual? Precisamos fazer com que ela seja sábado para esse teste! Poderíamos mudar a data da nossa máquina, mas essa não é uma solução elegante. Sabemos também que é impossível mockar um método estático, ou seja, não conseguimos mockar `Calendar.getInstance()`.

Uma possível solução para o problema é criar uma abstração de um relógio; uma interface que teria um método `hoje()`, por exemplo, responsável por devolver a data atual. Teríamos também uma implementação concreta, que simplesmente faria `Calendar.getInstance()`. Essa abstração é facilmente mockável e, se nosso `GeradorDePagamento` fizer uso da abstração em vez de usar o Calendar diretamente, conseguiríamos testá-la.

Vamos então criar a interface `Relogio`, passá-la como dependência para `GeradorDePagamento` e pedir a hora atual para ela. Para manter a compatibilidade, manteremos o construtor antigo também:

```
public interface Relogio {
Calendar hoje();
}

public class RelogioDoSistema implements Relogio{
public Calendar hoje() {
return Calendar.getInstance();
}
}

public class GeradorDePagamento {

private final RepositorioDePagamentos pagamentos;
private final RepositorioDeLeiloes leiloes;
private final Avaliador avaliador;
```

```
private final Relogio relogio;

public GeradorDePagamento(RepositorioDeLeiloes leiloes,
            RepositorioDePagamentos pagamentos,
            Avaliador avaliador,
            Relogio relogio) {
this.leiloes = leiloes;
this.pagamentos = pagamentos;
this.avaliador = avaliador;
this.relogio = relogio;
}

public GeradorDePagamento(RepositorioDeLeiloes leiloes,
        RepositorioDePagamentos pagamentos,
        Avaliador avaliador) {
this(leiloes, pagamentos, avaliador, new RelogioDoSistema());
}

// ...

private Calendar primeiroDiaUtil() {
Calendar data = relogio.hoje();
int diaDaSemana = data.get(Calendar.DAY_OF_WEEK);

if(diaDaSemana == Calendar.SATURDAY)
    data.add(Calendar.DAY_OF_MONTH, 2);
else if(diaDaSemana == Calendar.SUNDAY)
    data.add(Calendar.DAY_OF_MONTH, 1);

return data;
}
```

Agora, tendo o `Relogio` como dependência, conseguimos facilmente criar um mock para ele e fazer com que o método `hoje()` devolva a data de sábado.

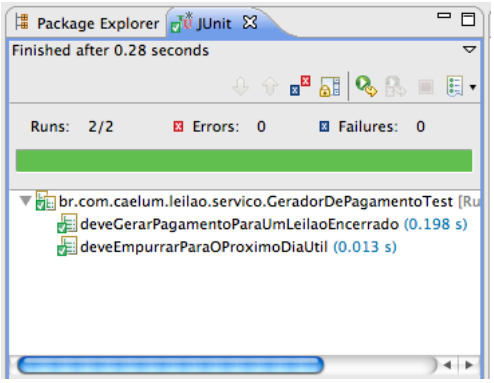
Vamos lá:

```
@Test
public void deveEmpurrarParaOProximoDiaUtil() {
RepositorioDeLeiloes leiloes = mock(RepositorioDeLeiloes.class);
RepositorioDePagamentos pagamentos = mock(RepositorioDePagamentos.class);
Relogio relogio = mock(Relogio.class);

// dia 7/abril/2012 eh um sabado
Calendar sabado = Calendar.getInstance();
sabado.set(2012, Calendar.APRIL, 7);

// ensinamos o mock a dizer que "hoje" é sabado!
when(relogio.hoje()).thenReturn(sabado);

Leilao leilao = new CriadorDeLeilao()
.para("Playstation")
.lance(new Usuario("José da Silva"), 2000.0)
.lance(new Usuario("Maria Pereira"), 2500.0)
.constroi();
```

```
when(leiloes.encerrados()).thenReturn(Arrays.asList(leilao));

GeradorDePagamento gerador =
            new GeradorDePagamento(leiloes, pagamentos, new Avaliador(), relogio);
gerador.gera();

ArgumentCaptor<Pagamento> argumento = ArgumentCaptor.forClass(Pagamento.class);
verify(pagamentos).salva(argumento.capture());
Pagamento pagamentoGerado = argumento.getValue();

assertEquals(Calendar.MONDAY, pagamentoGerado.getData().get(Calendar.DAY_OF_WEEK));
assertEquals(9, pagamentoGerado.getData().get(Calendar.DAY_OF_MONTH));
}
```

Veja que criamos um `Calendar` para o dia 07/04/2012, que é um sábado, e ensinamos o mock de `Relogio` a responder ao método `hoje()` com essa data. Agora conseguimos simular o "dia de hoje"!

Nosso teste agora passa!



Sempre que tivermos dificuldade de testar algum trecho de código (geralmente os que fazem uso de métodos estáticos), é comum criarmos abstrações para facilitar o teste. A abstração de relógio é muito comum em sistemas bem testados.