01

Começando com testes de unidade

Transcrição

Todos nós desenvolvedores já escrevemos código que não funciona alguma vez na nossa carreira. E o pior é que nós geralmente só descobrimos que ele não funciona quando esse código já está em produção e o cliente nos liga bravo por que o software não funciona. Infelizmente bugs de software são mais comuns do que deveriam ser, mas por que será que isso acontece? Vamos ver na prática, começando com o nosso sistema de leilão onde um usuário dá lances em um leilão e ao final temos um vencedor.

> Baixe as classes do projeto aqui (http://bit.ly/caelum-testes-cap1). Com o zip em mãos, dezipe-o na sua workspace, no Eclipse vamos importá-lo.

File -> Import -> selecione a opção Existing Projects into Workspace e selecione o diretório que nós acabamos de dezipar, no caso teste-de-unidade. Clique em Finish e pronto, temos o projeto no Eclipse.

Vamos dar uma navegada nas classes que já existem, por exemplo, a classe Lance é composta de um usuário e um valor, a classe Leilao é composta por descrição e vários lances, e por fim a classe Usuário é composta de um id e um nome.

Ótimo, agora precisamos implementar a seguinte funcionalidade, dado um leilão, precisamos saber qual o maior lance que foi dado, para isso vamos criar uma classe CTRL + N (Class.), que vamos chamar de avaliador e colocaremos no pacote de leilão.serviço:

```
class Avaliador {

    private double maiorDeTodos = Double.NEGATIVE_INFINITY;

    public void avalia(Leilao leilao) {

        for(Lance lance : leilao.getLances()) {
            if(lance.getValor() > maiorDeTodos) maiorDeTodos = lance.getValor();
        }
    }

    public double getMaiorLance() {
        return maiorDeTodos;
    }
}
```

Criando a main:

```
class TesteDoAvaliador {

    public static void main(String[] args) {
        Usuario joao = new Usuario("João");
        Usuario jose = new Usuario("José");
        Usuario maria = new Usuario("Maria");

        Leilao leilao = new Leilao("Playstation 3 Novo");
```

```
        leilao.propoe(new Lance(joao, 300.0));
        leilao.propoe(new Lance(jose, 400.0));
        leilao.propoe(new Lance(maria, 250.0));

        Avaliador leiloeiro = new Avaliador();
        leiloeiro.avalia(leilao);

        System.out.println(leiloeiro.getMaiorLance()); // imprime 400.0
    }
}
```

Esse código funciona. Ao receber um leilão, ele varre a lista buscando o maior valor. Veja que a variável `maiorDeTodos` é inicializada com o menor valor que cabe em um `double`. Isso faz sentido, já que queremos que, na primeira vez que o `for` seja executado, ele caia no `if` e substitua o menor valor do double pelo primeiro valor da lista de lances.

A próxima funcionalidade a ser implementada é a busca pelo menor lance de todos. Essa regra de negócio faz sentido estar também na classe `Avaliador`. Basta acrescentarmos mais uma condição, desta vez para calcular o menor valor:

```
class Avaliador {

    private double maiorDeTodos = Double.NEGATIVE_INFINITY;
    private double menorDeTodos = Double.POSITIVE_INFINITY;

    public void avalia(Leilao leilao) {
        for(Lance lance : leilao.getLances()) {
            if(lance.getValor() > maiorDeTodos) maiorDeTodos = lance.getValor();
            else if(lance.getValor() < menorDeTodos) menorDeTodos = lance.getValor();
        }
    }

    public double getMaiorLance() {
        return maiorDeTodos;
    }

    public double getMenorLance() {
        return menorDeTodos;
    }
}


class TesteDoAvaliador {

    public static void main(String[] args) {
        Usuario joao = new Usuario("João");
        Usuario jose = new Usuario("José");
        Usuario maria = new Usuario("Maria");

        Leilao leilao = new Leilao("Playstation 3 Novo");

        leilao.propoe(new Lance(joao, 300.0));
        leilao.propoe(new Lance(jose, 400.0));
        leilao.propoe(new Lance(maria, 250.0));

        Avaliador leiloeiro = new Avaliador();
        leiloeiro.avalia(leilao);
```

```
        System.out.println(leiloeiro.getMaiorLance()); // imprime 400.0
        System.out.println(leiloeiro.getMenorLance()); // imprime 250.0
    }
}
```

Tudo parece estar funcionando. Apareceram na tela o menor e maior valor corretos. Vamos colocar o sistema em produção, afinal está testado!

Mas será que o código está realmente correto? Veja agora um outro teste, muito parecido com o anterior:

```
class TesteDoAvaliador {

    public static void main(String[] args) {
        Usuario joao = new Usuario("João");
        Usuario jose = new Usuario("José");
        Usuario maria = new Usuario("Maria");

        Leilao leilao = new Leilao("Playstation 3 Novo");

        leilao.propoe(new Lance(joao, 250.0));
        leilao.propoe(new Lance(jose, 300.0));
        leilao.propoe(new Lance(maria, 400.0));

        Avaliador leiloeiro = new Avaliador();
        leiloeiro.avalia(leilao);

        System.out.println(leiloeiro.getMaiorLance()); // imprime 400.0
        System.out.println(leiloeiro.getMenorLance()); // INFINITY
    }
}
```

E veja que, para um cenário um pouco diferente, nosso código não funciona! A grande pergunta é: no mundo real, será que teríamos descoberto esse bug facilmente, ou esperaríamos nosso cliente nos ligar bravo porque a funcionalidade não funciona? Infelizmente, bugs em software são uma coisa mais comum do que deveria ser. Bugs nos fazem perder a confiança do cliente, e nos custam muito dinheiro. Afinal, precisamos corrigir o bug e recuperar o tempo perdido do cliente, que ficou parado, enquanto o sistema não funcionava.

Por que nossos sistemas apresentam tantos bugs assim? Um dos motivos para isso é a falta de testes, ou seja, testamos muito pouco! Equipes de software geralmente não gostam de fazer (ou não fazem) os devidos testes. As razões para isso são geralmente a demora e o alto custo para testar o software como um todo. E faz todo o sentido: pedir para um ser humano testar todo o sistema é impossível: ele vai levar muito tempo para isso!

E como resolver esse problema? Fazendo a máquina testar! Escrevendo um programa que teste nosso programa de forma automática! Uma máquina, com certeza, executaria o teste muito mais rápido do que uma pessoa! Ela também não ficaria cansada ou cometeria erros!

Um teste automatizado é muito parecido com um teste manual. Imagine que você está testando manualmente uma funcionalidade de cadastro de produtos em uma aplicação web. Você, com certeza, executará três passos diferentes. Em primeiro lugar, você pensaria em um cenário para testar. Por exemplo, "vamos ver o que acontece com o cadastro de funcionários quando eu não preencho o campo de CPF". Após montar o cenário, você executará a ação que quer testar.

Por exemplo, clicar no botão "Cadastrar". Por fim, você olharia para a tela e verificaria se o sistema se comportou da maneira que você esperava. Nesse nosso caso, por exemplo, esperaríamos uma mensagem de erro como "CPF inválido".

Um teste automatizado é muito parecido. Você sempre executa estes três passos: monta o cenário, executa a ação e valida a saída. Mas, acredite ou não, já escrevemos algo muito parecido com um teste automatizado nesta aula. Lembra da nossa classe `Teste` ? Veja que ela se parece com um teste, afinal ela monta um cenário e ela executa uma ação. Veja o código abaixo:

```
class Teste {

    public static void main(String[] args) {
        // cenario: 3 lances em ordem crescente
        Usuario joao = new Usuario("João");
        Usuario jose = new Usuario("José");
        Usuario maria = new Usuario("Maria");

        Leilao leilao = new Leilao("Playstation 3 Novo");

        leilao.propoe(new Lance(joao, 250.0));
        leilao.propoe(new Lance(jose, 300.0));
        leilao.propoe(new Lance(maria, 400.0));

        // executando a acao
        Avaliador leiloeiro = new Avaliador();
        leiloeiro.avalia(leilao);

        // exibindo a saida
        System.out.println(leiloeiro.getMaiorLance()); // imprime 400.0
        System.out.println(leiloeiro.getMenorLance()); // imprime INFINITY
    }
}
```

E veja que ele é automatizado!