Programação Orientada por Objetos 2021/2022

Ficha de Laboratório #2

Objetivo

> - Introdução à utilização de testes unitários.

Programa

Protótipo de uma aplicação para facilitar a gestão de um stand. A aplicação deve permitir registar: os clientes, os vendedores, os veículos comercializados e as vendas realizadas pelo stand (os vários registos não podem aceitar duplicados).

Regras de implementação

- Criar a aplicação utilizando o IDE BlueJ.
- Implementar o código necessário e testar no fim de cada nível.
- **Atualizar a versão do programa no repositório no mínimo no fim de cada nível** (é aconselhado no final de cada funcionalidade implementada e testada com sucesso). Não é necessário incluir na versão os ficheiros compilados (extensão .class).
- Use as convenções de codificação adotadas para a linguagem Java (ver **Notas**).

Implementação

Nível 1:

As classes fornecidas correspondem à estrutura das classes da aplicação, com as assinaturas dos métodos públicos, sem a implementação do código interno. Vamos preparar o programa para que possa compilar e vamos criar alguns testes básicos que serão executados repetidamente ao longo da implementação do código interno da aplicação.

1. Inclua todas as classes fornecidas com este enunciado `User`, `Vehicle`, `Sell` e `Stand` num projeto BlueJ deste laboratório, no repositório local clonado com o GitHub Desktop.
2. Altere os métodos de forma a retornar valores fixos de modo a possibilitar a compilação das classes. Por exemplo, os métodos que retornam referências de objetos retornam o valor `null`, os métodos que retornam inteiros retornam o valor -1, etc.

A classe `Vehicle` representa um veículo. Crie a classe de teste `VehicleTest`, para a classe `Vehicle`.

3. Crie o método de teste `testConstructor` que testa o construtor. Este método de teste deve criar o objeto vehicle1 (ver dados na **Tabela 1**) como uma Fixture e verificar que o objeto é criado com os valores fornecidos, recorrendo aos seletores do preço e do modelo para isso.

4. Implemente o código interno do construtor e também de todos os seletores da classe (necessários para verificar que o construtor inicializou o objeto com os valores corretos).
5. Execute o teste criado (quando completar a implementação do código interno de todos os métodos utilizados no teste, a execução do teste deve ter sucesso).
6. Quando o teste passar com sucesso, crie uma nova versão da aplicação.

Identificador	Modelo	Preço
vehicle1	Renault Captur	34450.0

Tabela 1 - Veículo a criar para o teste do construtor

Nível 2:

O método `toString` da classe `Vehicle` deve retornar uma string no formato apresentado na listagem 1.

1. Crie o método de teste `testToString`.
2. Implemente o código interno do método `toString`.
3. Execute todos os testes criados.

```
Veículo:
Modelo        : Renault Captur
Preço         : 34450.0 Euros
```

Listagem 1 - Exemplo de String a devolver pelo método toString para o veículo da tabela 1

Nível 3:

A classe `Stand` representa um stand de venda de veículos.

Identificador	Nome	Telefone	E-mail
client1	Helder Oliveira	911111111	ho356@yahoo.com
client2	António Parreira	922222222	toparreira@hotmail.com
seller1	Paulo Figueira	966777101	paulo.figueira@renault.pt
seller2	Pedro Pereira	966777152	pedro.pereira@renault.pt

Tabela 2 - Objetos da fixture para a classe StandTest

1. Crie a classe de teste `StandTest`, para a classe `Stand`.
2. Defina na classe `StandTest` uma Fixture com um objeto `Stand` e todos os objetos da tabela 2.

3. Crie, nesta classe, o método de teste `testConstructor` que testa o construtor. Este método de teste cria um objeto stand2 e verifica que o construtor instancia as coleções do stand, ou seja, que após a criação os atributos não são null.
4. Implemente o código interno do construtor e dos seletores da classe.
5. Execute todos os testes criados.

Nível 4:

1. Crie o método de teste `testRegisterClient` que regista um cliente e o método de teste `testRegisterClients` que regista dois clientes.
2. Crie método de teste negativo `testRegisterClientDuplicate`, que tenta registar o mesmo cliente duas vezes.
3. Crie método de teste negativo `testRegisterClientNull`, que tenta registar um cliente null.
4. Em seguida, implemente o código interno do método `registerClient`.
5. Execute todos os testes criados.
6. Aplique a mesma abordagem do que nas alíneas 1 a 5 para testar o registo de um vendedor e o registo de um veículo.

Nível 5:

1. Crie na classe de teste **StandTest** o método de teste `testCreateSell`.
2. Em seguida, implemente o código interno do método `createSell`.
3. Execute todos os testes criados.
4. Crie o método de teste `testCalculateSellsOfTheYear`. Em seguida, implemente o código interno do método `calculateSellsOfTheYear`.
5. Execute todos os testes criados.
6. Crie o método de teste `testFindSellerOfTheYear`. Em seguida, implemente o código interno do método `findSellerOfTheYear`.
7. Execute todos os testes criados.

Identificador	Cliente	Vendedor	Veículo
sell1	client1	seller1	vehicle1
sell2	client2	seller2	vehicle2

Tabela 3 - Exemplo de vendas a utilizar nos testes

Notas:

Para os identificadores siga as convenções adotadas normalmente, em particular:

1. A notação **camelCase** para o nome das variáveis locais e identificadores de atributos e métodos.
2. A notação **PascalCase** para os nomes das classes.
3. Não utilize o símbolo '_', nem abreviaturas nos identificadores.