Instituto Politécnico de Beja
Escola Superior de Tecnologia e Gestão

Guia Prático 5
Programação Conduzida por Testes

João Paulo Barros

25 de abril de 2022

Objectivos: Conhecer e iniciar a aplicação de um método para o desenvolvimento de software: a "receita para programar". Realização de uma aplicação Java™ com um método main, utilizando programação conduzida por testes.

1 Um programa para uma máquina dispensadora

Como exemplo de utilização a programação conduzida por testes, em inglês *test driven development (TDD)*, vamos implementar um programa que simula uma máquina de venda de produtos ("dispenser machine"). As funcionalidades principais serão as seguintes:

1. A máquina vende apenas um tipo de produto.
2. A máquina guarda a quantidade de dinheiro que recebe de cada venda.
3. A máquina aceita moedas com os seguintes valores: 5, 10, 20, e 50.
4. Quando o dinheiro é suficiente, o utilizador pode premir um botão para comprar uma unidade do produto. Nessa altura, ocorre um de dois casos:
   (a) Se o dinheiro é igual ou maior do que o preço do produto, a máquina devolve o troco (se existente); subtrai um produto ao total de produtos na máquina e adiciona o preço do produto ao total de dinheiro recebido pela máquina.
   (b) Se o dinheiro é menor do que o preço do produto, a máquina mostra o valor do dinheiro que deve ser adicionado (para atingir o preço do produto).
5. Em qualquer altura, o utilizador pode premir um outro botão para cancelamento da compra. Nesse caso, a máquina devolve o dinheiro já inserido.

1.1 Identificação de objectos

No presente problema há duas classes de objectos bastante óbvias: máquinas e produtos. Por enquanto vamos considerar que todos os produtos são iguais pelo que bastará que a máquina (um objecto) tenha um número inteiro que indica quantos produtos contém. Execute então os seguintes dois passos:

1. Defina um novo projecto (no IntelliJ) para o seu programa com o nome `DispenserMachineSimulator`.

2. Defina uma classe para a máquina de venda (denominada `Dispenser`). Por enquanto, a classe pode ficar vazia.

1.2 Escrita de teste

cenário

Sim. Vamos começar por fazer o teste! No teste vamos dizer em Java™ o que queremos que a máquina faça. O teste será um **CENÁRIO** (algo que queremos que o sistema faça) em código Java™. Para tal teremos de seguir alguns passos até termos um projeto preparado para conter código fonte (pasta *src*) (para a aplicação que queremos fazer) e código de teste (pasta (*src*)) (para testar o model da aplicação que queremos fazer);

1. Crie um novo projecto Java™com o nome "DispenserMachineSimulator"; deve ficar com algo semelhante à Fig. 1;

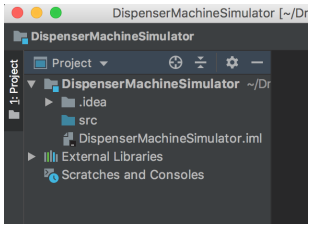

Figura 1: Criação de um novo projecto.

2. Agora vamos adicionar uma pasta para colocar o código de teste; para tal clique com o botão direito do rato no nome do projeto e escolha "New->Directory" (ver Fig. 2;

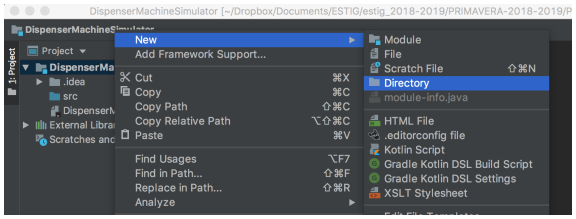

Figura 2: Criação de uma pasta colocar o código de teste.

3. Ainda temos de dizer ao IntelliJ que essa é uma pasta para colocar código de teste; para tal clique com o botão direito do rato na pasta "test" já criada e escolha "Mark Directory as -> Test Sources Root" (ver Fig. 3);

4. Como já identificámos que será necessário um objecto para a modelar a nossa máquina de vendas, vamos começar por criar uma classe `pt.ipbeja.po2.dispenser.model.Dispenser`; o projecto deverá ficar como o apresentado na Fig. 4;

5. Agora vamos criar uma classe onde ficará o código que testa o código da classe `Dispenser`; deve abrir o ficheiro com a classe `Dispenser` e premir Alt + Enter para que surja um menu com a opção "Create Test" (ver Fig. 5);

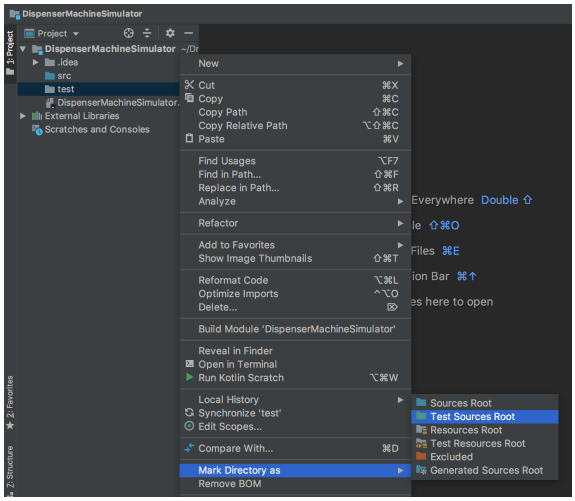

Figura 3: Marcação da pasta para colocar o código de teste.

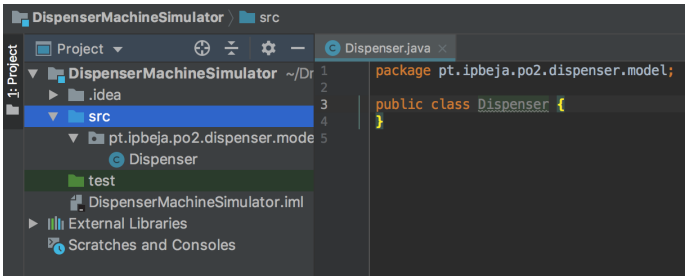

Figura 4: Projecto com classe "Dispenser" na `package` `pt.ipbeja.po2.dispenser.model`.

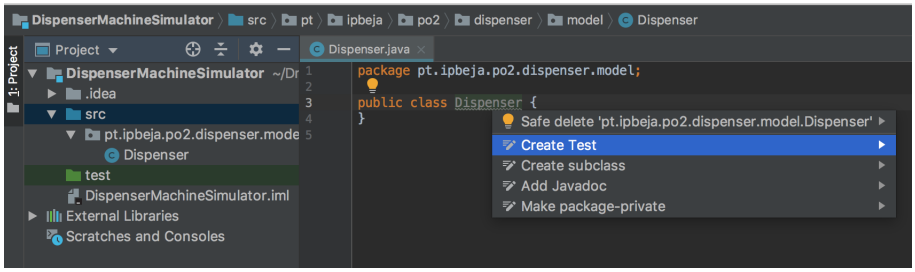

Figura 5: Criação do código de teste.

6. Na caixa de diálogo que surge deve escolher como "Testing library:" a "JUnit 5" (ver Fig. 6); Ainda na mesma caixa de diálogo deve adicionar o código da biblioteca "JUnit 5"ao projeto; para tal deve clicar o botão "Fix" e depois "OK" e novamente "OK" (ver Fig. 7);

7. Neste momento deve ter um projecto com o aspecto da Fig. 8; note o "JUnit5.3" dentro de "External Libraries" e a classe `DispenserTest` na pasta "test"; note que a classe `DispenserTest` está na mesma `package` da classe `Dispenser`, a `package` `pt.ipbeja.po2.dispenser.model`, apesar de estar noutra pasta;

8. Finalmente, podemos agora criar um método de teste; para tal vamos começar por criar um *stub* (minuta) para o mesmo; para isso deve clicar com o botão direito dentro da classe `DispenserTest` e escolher "Generate" (ver Fig. 9) e depois "Test Method"; o seu projeto deverá ficar com o aspeto da

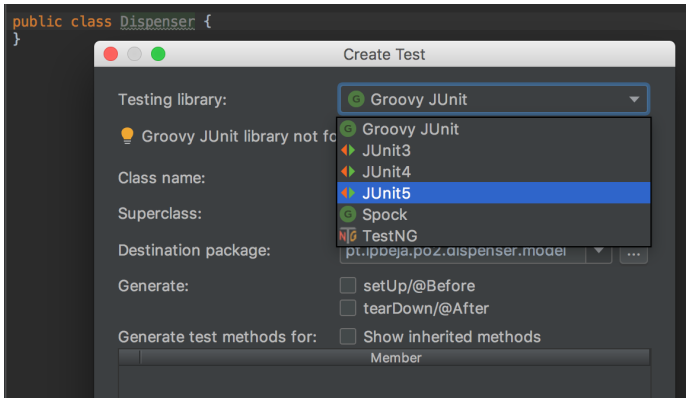

Figura 6: Escolha da JUnit 5.

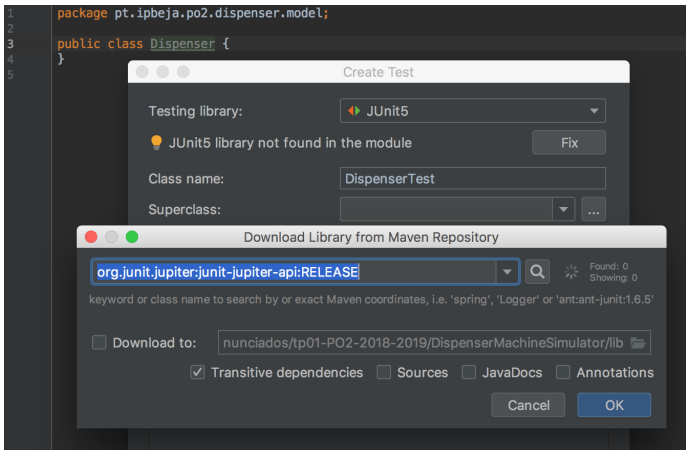

Figura 7: Adição da biblioteca JUnit 5 ao projeto.

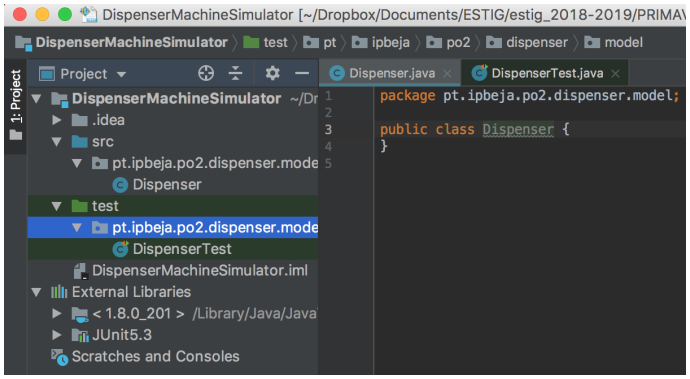

Figura 8: Estado do projeto após adição da "JUnit 5".

Fig. 10;

Agora já podemos escrever o código do nosso método de teste!

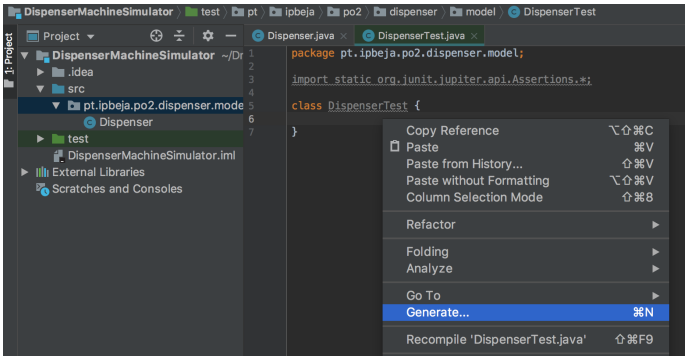

Figura 9: Estado do projeto após adição da "JUnit 5".

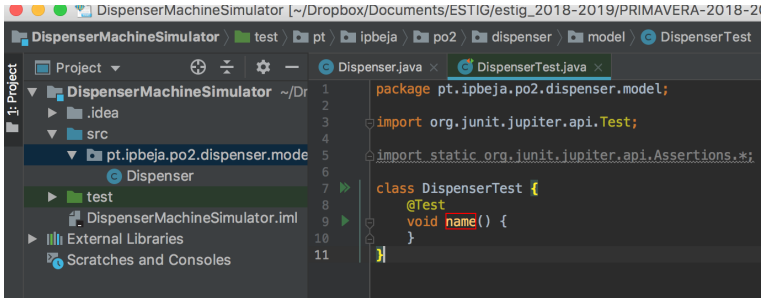

Figura 10: Estado do projeto após adição de um *stub* para um método de teste.

1.3 O primeiro método de teste

Pretendemos definir um teste para inserção de moedas na máquina de venda. Para isso deve mudar o nome do método de teste de "test" para `testInsertCoin`. Note que este teste deve ser definido escrevendo o respectivo código, ou seja, no método `testInsertCoin` na classe `DispenserTest`. Por exemplo:

```
@Test
public void testInsertCoin() {
    // create machine with price 40 for all products
    Dispenser dispenser = new Dispenser(40);

    // inserts one coin
    int balance = dispenser.insertCoin(20);

    // test if method returns the correct (expected) value
    assertEquals(20, balance);

    // inserts another coin
    balance = dispenser.insertCoin(10);

    // test if method returns the correct (expected) value
    assertEquals(30, balance);
}
```

Claro que este teste ainda não pode ser compilado nem executado pois ainda não fizemos o código! Só agora vamos definir os métodos que permitem que o teste passe. Ou seja, ao fazermos o código somos guiados ou conduzidos pelos testes. Eis então a chamada **PROGRAMAÇÃO CONDUZIDA POR TESTES**.

programação conduzida por testes

1.4 Escrever o código necessário para executar o teste com sucesso

Escreva o código necessário para o teste passar.

No exemplo apresentado, tal corresponde a definir o construtor e um método `insertCoin`. Note que irá necessitar de um atributo que guarda a quantidade de dinheiro inserido na máquina: `insertedMoney`. Esse atributo é incrementado no método `insertCoin` o qual deve devolver o valor actual do dinheiro inserido.

Já pode verificar se o código passa no teste.

1.5 Execução do teste

1. Execute o teste para confirmar que o código escrito está de acordo com esse teste. Para tal clique com o botão direito na classe de teste `DispenserTest` e escolha "Run 'DispenserTest' ".

error
failure

2. Se dá erro verifique se o erro é um **ERROR** (erro na execução do código) ou uma **FAILURE** (falha o teste, ou seja o definido num `assertEquals`). Em qualquer dos casos, corrija o código de forma a que o teste passe (zero *Errors* e zero *Failures*).

2 Completar o programa

Pode agora continuar a resolução seguindo o mesmo método.

Agora irá também necessitar dos seguintes atributos:

1. Um atributo que guarda o preço dos produtos: `productPrice`.
2. Um atributo que guarda a quantidade de produtos: `nProducts`.
3. Um atributo que guarda a quantidade de dinheiro recebido pela máquina que corresponde à venda de produtos: `salesMoney`.

Inicialmente a quantidade de dinheiro inserido (`insertedMoney`) está a zero (no construtor que "cria"uma máquina).

1. Deve definir um novo teste, ou seja, repetir o ponto 8 da secção 1.2. Em resumo: deve executar um novo passo iterativo na construção do seu programa. Deve seguir a seguinte ordem:
   (a) Identificar objectos.
   (b) Definir um teste.
   (c) Definir o código necessário para o teste passar.
   (d) Executar o teste para confirmar que o código escrito está de acordo com esse teste.
   (e) Dar os comandos (rato e teclado) ao colega.

(f) Passar para o ponto 1b.

2. Claro que em qualquer momento qualquer um dos dois programadores pode identificar novos objectos e portanto, eventualmente novas classes.

3. Continue a resolução do problema proposto utilizando esta receita. Para tal, considere os seguintes teste adicionais:

   (a) Cancelamento da compra com obtenção do dinheiro inserido.
      i. Cria um objecto `Dispenser` com um preço fixo de 40 para todos os produtos.
      ii. Insere dinheiro na máquina (`dispenser.insertCoin(20);`);
      iii. Insere mais dinheiro na máquina (`dispenser.insertCoin(10);`);
      iv. Pede à máquina para cancelar a compra (`dispenser.cancel();` e verifica se o valor devolvido pelo método é o valor inserido, ou seja igual a 30.
      v. Verifica também se a quantidade de produtos na máquina se manteve igual.
      vi. Verifica se a quantidade de dinheiro recebido pela máquina se manteve igual.

   (b) Compra e obtenção do troco:
      i. Cria um objecto `Dispenser` com um preço fixo de 40 para todos os produtos (parâmetro do construtor).
      ii. Insere dinheiro na máquina (`dispenser.insertCoin(50);`);
      iii. Pede à máquina para lhe vender produto (`dispenser.buyProduct();` e verifica se o o troco é igual a 10.
      iv. Para já, deve considerar que os produtos são representados unicamente por um contador que indica quantos produtos se encontram na máquina. Este contador é decrementado quando é feita uma venda. Assim, a máquina deve ter ficado com menos um produto. Verifique que assim é perguntando à máquina (objecto dispenser), antes e depois da venda, quantos objectos tem.
      v. Verifica também se a quantidade de produtos na máquina diminuiu de uma unidade.
      vi. Verifica se a quantidade de dinheiro recebido pela máquina aumentou 40.

   (c) Tentativa de compra com dinheiro insuficiente. Semelhante ao anterior mas o método `buyProduct` devolve um valor negativo que corresponde ao dinheiro em falta:
      i. Cria um objecto `Dispenser` com um preço fixo de 40 para todos os produtos.
      ii. Insere dinheiro na máquina (`dispenser.insertCoin(20);`);
      iii. Insere mais dinheiro na máquina (`dispenser.insertCoin(5);`);
      iv. Pede à máquina para lhe vender produto (`dispenser.buyProduct();` e verifica se o o troco é igual a -15.
      v. Verifica também se a quantidade de produtos na máquina se manteve igual.
      vi. Verifica se a quantidade de dinheiro recebido pela máquina se manteve igual.

Note que em todos os testes se começa por criar um objecto da classe `Dispenser`. Pode evitar esta repetição de código colocando este código comum, e que corresponde a criar o objecto necessário para o teste, no método `setUp` da classe de teste. Este método `setUp` é executado antes de cada teste pelo que o efeito final será igual ao código já feito.

3 Mais algumas operações a realizar pela máquina

Defina testes e (só depois!) os respectivos métodos para cada uma das seguintes funcionalidades reservadas ao administrador:

1. O administrador da máquina pode fixar um preço para os produtos.
2. O administrador da máquina pode fixar uma quantidade de produtos.
3. O administrador da máquina pode perguntar, e ficar a saber, quantos produtos estão na máquina.
4. O administrador da máquina pode perguntar, e ficar a saber, quanto dinheiro (resultante de compras já efectuadas) está na máquina.

4 Funcionalidades EXTRA

1. O administrador pode trocar o preço dos produtos para um múltiplo 10.
2. O utilizador pode comprar mais que um produto ao mesmo tempo. Quando o utilizador indicar o número de produtos a comprar, a máquina só vende a quantidade pretendida.
3. A máquina guarda moedas de 5, 10, 20, e 50 em separado e devolve o troco de acordo com as moedas existentes.

5 Uma receita para programar

Futuramente, pode passar a seguir o procedimento que pode ser resumido na seguinte "receita":

1. **Analise o problema e a informação disponível sobre mesmo**. Para tal leia o enunciado com cuidado e várias vezes. O enunciado contém muitas indicações sob a forma de melhor estruturar o seu programa. **Represente essa informação sob a forma de classes/objectos e respectivos atributos**. Deve também identificar relações entre os diversos objectos, mas disto falaremos mais tarde.
2. **Escreva um pequeno teste**.Este teste deve ter a forma de um método de teste numa classe de teste. Este código de teste não irá em rigor pertencer ao programa a implementar mas é extremamente importante para o sucesso do seu desenvolvimento. Caso o código de teste se torne muito longo defina outros métodos na classe de teste como forma de o decompor.
3. **Implemente o teste**:
   (a) (Opcional como rascunho do passo seguinte): para cada construtor e método que identificou no passo anterior, escreva um esqueleto de **dados (variáveis) para cada método**. Este esqueleto de dados é constituído pelos **cabeçalhos dos métodos** e por corpos (definições desses métodos) que contêm apenas os dados que serão utilizados por cada método (ainda sem saber exactamente como serão utilizados).

(b) **Defina o corpo de cada método**.

4. Teste o programa utilizando o código de teste já desenvolvido no passo 2.

A "receita para programar"é uma adaptação da Design Recipe (in Viera K. Proulx and Tanya Cashorali, "Calculator problem and the design recipe" ACM SIGPLAN Notices Volume 40 , Issue 3 (March 2005) `http://doi.acm.org/10.1145/1057474.1057478`

Deve terminar a resolução deste guia fora das aulas. Traga as dúvidas para a próxima aulas ou coloque-as no fórum de dúvidas da disciplina. As sugestões para melhorar este texto também são bem-vindas.