Teste e Validação de Software 2019/20
Instituto Superior Técnico

Enunciado do Projecto

Data de Entrega: 10 de Abril de 2020

1 Introdução

O desenvolvimento de grandes sistemas de informação é um processo complexo e requer diversos níveis de abstracção. Um primeiro passo é a especificação do problema de uma forma rigorosa. Após a especificação rigorosa do problema, é possível efectuar provas de correcção do mesmo ou, caso este esteja incorrecto, descobrir falhas que de outra forma seriam difíceis de detectar. Depois, dever-se-á começar a codificar a solução e a testá-la simultaneamente por forma a detectar os erros o mais cedo possível.

O projecto da disciplina Teste e Validação de Software consiste no desenho de casos de teste a partir de uma especificação disponibilizada. Esta especificação diz respeito a algumas entidades do sistema informático, designado como *PostOffice*, responsável pela gestão de uma estação de correios.

Este projecto tem como objectivo principal de aprendizagem que os alunos ganhem experiência no desenho de casos de testes ao aplicarem os padrões de teste leccionados nas aulas teóricas a um conjunto de classes e métodos cujo comportamento está descrito neste documento.

Este documento está organizado da seguinte forma. A secção 2 descreve as entidades que participam no sistema a testar assim como alguns detalhes de concretização destas entidades e que serão importantes para os casos de teste a desenhar. A secção 3 identifica os testes a realizar e a forma como a resolução do projecto vai ser avaliada.

2 Descrição do sistema

O sistema a testar é responsável pela gestão dos clientes, produtos e facturas de uma estação de correios. Todas estas entidades têm um identificador único no sistema. O sistema é responsável por assegurar que não existem duas entidades do mesmo tipo com o mesmo identificador.

2.1 A entidade Invoice

A entidade `Invoice` representa uma factura passada pelo sistema como resultado de um serviço efectuado. Esta entidade gere uma determinada compra a ser efectuada por um dado cliente desde o início da interação até ao pagamento da factura. A figura 1 apresenta a interface desta entidade.

Quando um cliente pretende realizar uma compra na estação de correios é criada uma instância de `Invoice`. Quando uma factura é criada ela está aberta. Esta instância será responsável por manter a lista de produtos que o cliente pretende comprar. Enquanto uma factura estiver aberta é possível adicionar (ou remover) produtos à factura indicando qual o produto desejado e a respectiva quantidade. O produto é indicado pelo seu identificador (que é uma cadeia que caracteres e representa o nome do produto). Em caso de sucesso (foi possível adicionar ou remover o produto indicado), então os métodos devolvem `true`. Se não for possível adicionar o produto (porque não existe na estação de correios ou não tem a quantidade desejada) ou remover o produto (porque não está presente na factura) então o método em causa não tem qualquer efeito e devolve `false`.

Quando um dado cliente quer comprar os produtos presentes numa factura aberta então deve confirmar a factura invocando o método `checkout`. Por razões de segurança não é possível ter facturas com um valor superior a 100

```
enum InvoiceMode {
  OPEN, CONFIRMED, PAYD, WAITING_FOR_PAYMENT, CANCELLED;
}

public class Invoice {

  // creates a new invoice with the specified identifier.
  public Invoice(int id) {/*....*/}

  // adds a product p with the given name and given quantity to this invoice if the invoice is open
  public boolean add(String name, quantity q) { /* .... */ }

  // removes the product whith the specified name  from this invoice if the invoice is open. If the product
  // does not exist in the invoice it does nothing.
  public boolean remove(String name) { /* .... */ }

  // cancels a confimerd invoice
  public void cancel() { /* .... */ }

  // confirms this invoice if the invoice is open and ita value is less than 100 euros.
  // Returns true if it was possible to confirm the open invoice, false otherwise.
  public boolean checkout(Client client)  { /* .... */ }

  // pays a confirmed or "waiting for payment" invoice.
  public void pay() {/*....*/}

  // pays a confirned invoice by credit.
  public void payByCredit() { /* .... */ }

  // computes the current value to pay for this invoice taking into account the client
  // and the number of days of delay.
  float computeCreditBill() { /* .... */ }

  // returns the list of products of this invoice
  List<Product> getProducts() { /* .... */ }

  // returns the mode of this invoice
  InvoiceMode getMode() { /* .... */ }
  ...
}
```

Figura 1: A interface da classe `Invoice`.

euros. Assim, a factura passa a confirmada caso o valor da compra seja menor ou igual a 100 euros. Se for superior, a factura continua aberta. O método `checkout` devolve um boleano a indicar se foi possível ou não confirmar a factura. É possível cancelar uma factura confirmada, tornando-a aberta outra vez, através do método `cancel`.

Após se ter confirmado com sucesso uma factura é necessário ainda realizar o seu pagamento. O pagamento depende da forma de pagamento a realizar pelo cliente. Uma factura confirmada pode ser paga a pronto (através do método `pay`) ou o pagamento pode ser feito a crédito (via método `payByCredit`) e neste caso o cliente terá um prazo para pagar a compra. No primeiro caso, a factura fica paga enquanto que no segundo fica a aguardar pagamento. Neste último caso, a factura ficará paga quando o cliente finalmente efectuar o pagamento, o que é feito através do método `pay`. O método `computeCreditBill` indica qual o valor a pagar para uma factura a aguardar pagamento de acordo com a lógica de negócio descrita no fim desta secção.

Em qualquer situação é possível invocar os métodos `getProducts` e `getMode`. O primeiro devolve os produtos e respectivas quantidades presentes na factura enquanto que o segundo indica a situação em que a factura se encontra (aberta, confirmada, ...).

Invocações dos métodos de `Invoice` em situações não descritas anteriormente não deverão ter qualquer efeito e deverão lançar a excepção *InvalidInvocationException*.

O método `computeCreditBill` calcula o valor a pagar para uma dada compra feita a crédito e tem em conta o número de compras e o valor total das compras que o cliente já efectuou e se este está a realizar o pagamento

dentro ou fora do prazo. Uma compra feita a crédito tem 14 dias para ser paga sem qualquer penalização. Este método primeiro determina o custo base da factura. O custo base é igual ao somatório do custo unitário de cada produto presente na factura multiplicado pela respectiva quantidade. O método determina de seguida qual o desconto ou penalização a aplicar a este custo base da seguinte forma:

- No caso de a factura ser paga dentro do prazo, não existe penalização e o desconto a aplicar depende do número de compras feitas pelo cliente. Se estiver entre 30 e 15 (inclusive), o desconto é de 3%. Se for menor que 15, o desconto é apenas de 1%. Finalmente, se o número de compras for superior a 30 então o desconto depende do valor total das compras feitas pelo cliente. Caso este valor seja superior a 1000 euros, então o desconto é 10%, caso contrário o desconto é 6%.
- Se a factura for paga apenas com um dia de atraso, os clientes com um número de compras superior a 30 têm 2% de desconto. Caso o cliente tenha um número de compras menor ou igual então é aplicada uma penalização de 4% (o que pode ser visto como um desconto de -4%).
- Finalmente, se o número de dias de atraso for maior do que um, a penalização é igual a 10%.

2.2 A entidade Client

Cada cliente de `PostOffice` tem um nome (cadeia de caracteres) e é identificado pelo seu número de identificação fiscal (inteiro). O número de identificação fiscal identifica univocamente o cliente. Um cliente mantém também as compras já efectuadas na estação de correios. Esta entidade é representada pela classe `Client`, apresentada na figura 2.

```
public class  Client {
  // Creates a client with the given name and unique identifier
  public Client(String name, int id) { /* .... */ }
  // adds an invoice bought by this client
  void addInvoice(Invoice v) { /* .... */ }
  // returns the name of this client.
  public void getName() { /* .... */ }
  // returns the unique identifier of this object.
  public int getId() { /* .... */ }
  ...
}
```

Figura 2: A interface da classe *Client*.

2.3 A entidade Product

Existem vários produtos à venda numa estação de correios (por exemplo, selos e livros). Cada produto tem um nome, uma descrição do produto, um preço e uma quantidade crítica (utilizado na gestão de existências da estação de correios). A figura 3 apresenta a interface desta classe.

2.4 A entidade PostOffice

Uma estação de correios é responsável por fazer a gestão dos vários produtos que oferece. A estação de correios mantém as existências de cada produto oferecido pela estação de correios. No âmbito da estação de correios, cada produto é identificado pelo seu nome sendo por isso impossível ter dois produtos com o mesmo nome registados na mesma estação de correios. Para cada produto, a estação de correios guarda o número de unidades do produto disponíveis na estação de correios, o preço unitário do produto, a sua quantidade crítica e o nome do produto. A figura 4 apresenta a interface da entidade que representa uma estação de correios.

Cada estação de correios tem uma capacidade de armazenamento distinta, pelo que o número máximo de produtos pode não ser igual para todas as estações de correios. Este número máximo pode variar entre 2 e 20 e é indicado no momento da criação da estação de correios. Na criação de uma estação de correios é ainda indicado o conjunto inicial de produtos disponível na estação de correios. É possível alterar o valor máximo de produtos através

```
public class Product {
  // creates a product
  public Product(String name, String description, int price, int criticalValue) { /* .... */ }

  public String getDescription() { /* .... */ }

  public int getPrice() { /* .... */ }

  public int setPrice(int newPrice) { /* .... */ }

  public int getCurrentQuantity() { /* .... */ }

  public void store(int numberOfUnits) { /* .... */ }

  public String getName() { /* .... */ }
  ...
}
```

Figura 3: A interface da classe *Product*.

do método `setMaxNumberOfProducts`. O preço associado a cada produto não pode ser negativo. O número de unidades de cada produto não pode ser um número negativo.

É possível actualizar o preço ou a quantidade de um dado produto via método `update`. É possível adicionar novos produtos a uma estação de correios através do método `addNewProduct(Product p)` que irá adicionar um novo produto, referenciado por `p`, à estação de correios caso nenhuma das condições indicadas anteriormente não seja violada e o produto p não faça já parte do inventário da estação de correios em causa. É também possível remover um produto (identificado pelo seu nome) dos produtos associados à estação de correios caso não se esteja a invalidar nenhuma das condições indicadas anteriormente. Existe ainda a restrição adicional que o número de unidades do produto a remover é igual a 0. Os métodos `addNewProduct`, `removeProduct`, `updateProduct` e `setMaxNumberOfProducts` devolvem todos um valor boleano que indica se o método teve sucesso ou não em realizar a funcionalidade pretendida. Caso a operação em causa seja inválida (por exemplo, coloca o preço de um produto a um valor negativo), então o método devolve `false` e não tem qualquer efeito na estação de correios em causa.

```
public class PostOffice {
  public PostOffice(int maxNumberOfProducts, List<Product> products) { /* .... */ }

  // adds a new product to this post office.
  public boolean addNewProduct(Product p) { /* .... */ }

  // updates quantity and/or price of the given product
  public boolean update(String productName, int newPrice, int newQuantity) { /* .... */ }

  // removes a product from this post office
  public boolean removeProduct(String productName) { /* .... */ }

  public boolean setMaxNumberOfProducts(int newMaxQuantity) { /* .... */ }

   // accessor methods
  public List<Product> getProducts() { /* .... */ }
  public int getMaxNumberOfProducts() { /* .... */ }
  public void getMaxNumberOfProducts() { /* .... */ }
  ...
}
```

Figura 4: A interface da classe *PostOffice*.

3 Avaliação do projecto

Todos os casos de teste a realizar devem ser determinados aplicando os padrões de desenho de testes mais apropriados. Os casos de testes a determinar são os seguintes:

- Casos de teste ao nível de classe da classe `PostOffice`. Valem entre 0 e 3,5 valores.

- Casos de teste ao nível de classe da classe `Invoice`. Valem entre 0 e 6,5 valores.
- Casos de teste correspondentes ao método `computeCreditBill` da classe *Invoice*. Valem entre 0 e 3,5 valores.
- Casos de teste correspondentes ao método `addNewProduct` da classe *PostOffice*. Valem entre 0 e 3,5 valores.
- Adicionalmente, é necessário concretizar 6 casos de teste da bateria de testes desenhada para exercitar a classe `PostOffice`. Esta concretização deve ser feita utilizando a framework de testes *TestNG* e vale entre 0 e 3 valores.

Para cada método ou classe a testar é necessário indicar o seguinte:

- O nome do padrão de teste aplicado.
- Caso seja aplicável, indicar o resultado dos vários passos da aplicação do padrão, utilizando o formato apresentado nas aulas teóricas.
- A descrição dos casos de teste resultantes da aplicação do padrão de teste escolhido.

Caso algum dos pontos mencionados acima apenas seja satisfeito parcialmente a nota será dada na proporção realizada.

3.1 Discussão do projecto

Poderá existir uma discussão que avalie a capacidade dos alunos em realizar testes semelhantes aos realizados no projecto. Esta decisão cabe exclusivamente ao corpo docente da disciplina. Alunos cuja nota no exame seja inferior em mais de 5 valores à nota do projecto terão de realizar uma discussão e neste caso a nota do projecto será individualizada e dependerá do desempenho na discussão.

3.2 Detecção de cópias

A submissão de um projecto pressupõe o **compromisso de honra** que o trabalho incluso foi realizado pelos alunos referenciados nos ficheiros/documentos submetidos para avaliação. A quebra deste compromisso, ou seja a tentativa de um grupo se apropriar de trabalho realizado por colegas, tem como consequência a reprovação nesta disciplina de todos os alunos envolvidos (incluindo os que possibilitaram a ocorrência).

3.3 Notas Finais

O prazo de entrega do projecto é dia **10 de Abril de 2020 às 23:59**. O protocolo de entrega do projecto será disponibilizado oportunamente na página da disciplina bem como qual a documentação a entregar.

Toda a informação referente a este projecto estará disponível na página da disciplina na secção *Projecto*. O esclarecimento de dúvidas deve ser feito preferencialmente nas sessões de dúvidas.

Bom Trabalho!