Qualidade de Software 2014/15
Instituto Superior Técnico
Enunciado do 1º Projecto

Data de Entrega: 7 de Novembro de 2014

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 primeiro projecto de Qualidade de Software consiste no desenho de casos de teste a partir de uma especificação dada. Esta especificação modela o sistema informático, designado como *REST*, que é um portal de restaurantes que permite a venda de refeições a clientes registados no portal.

Este projecto tem como objectivo principal de aprendizagem que os alunos ganhem experiência no desenho de casos de testes aplicando os padrões de desenho de teste leccionados nas aulas teóricas.

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. Finalmente, a secção 3 identifica os testes a realizar e a forma como a resolução do projecto vai ser avaliada.

2 Descrição do sistema

O portal de restaurantes a testar é uma entidade que mantém informação sobre todos os restaurantes e clientes registados no sistema e é responsável pela gestão das compras de refeições realizadas pelos clientes. As entidades principais deste sistema são: *Rest*, *Restaurant*, *Dish*, *ShoppingTray* e *Client*. De seguida descrevem-se estas entidades.

2.1 A entidade Dish

Os pratos confeccionados por cada restaurante são representados pela entidade *Dish*. Um prato tem uma descrição, nome, restaurante e a indicação se é um prato vegetariano ou não. A figura 1 apresenta a interface desta entidade.

```
public class Dish {

  //creates a dish with a given name and description
  public Dish(String name, String description, boolean isVegetarian)   { /* .... */ }

  //returns the name of this dish
  public String getName() { /* .... */ }

  //sets the Restaurant of this dish
  public void setRestaurant(Restaurant r) { /* .... */ }

  //returns the Restaurant of this dish
  public Restaurant getRestaurant() { /* .... */ }

  //returns the description of this dish
  public String getDescription() { /* .... */ }

  //checks if this dish is vegetarian
  public boolean isVegetarian() { /* .... */ }
  ...
}
```

Figura 1: A interface da classe *Dish*.

Cada prato está associado a um só restaurante.

2.2 A entidade Restaurant

Cada restaurante tem um nome e uma morada. O restaurante é responsável por gerir a informação sobre os seus pratos. Cada restaurante tem que satisfazer as seguintes condições. Um restaurante pode ser vegetariano e neste caso, apenas pode oferecer pratos vegetarianos. Cada restaurante tem que oferecer pelo menos 5 pratos e máximo pode oferecer até 15 pratos. Um restaurante não pode oferecer dois pratos com o mesmo nome, ou seja, o nome de cada prato é único dentro de cada restaurante. Cada restaurante define o preço de cada um dos seus pratos. No entanto, existe um preço máximo (igual para todos os restaurantes e definido pelo sistema) que não pode ser ultrapassado. Os restaurantes podem ter pratos grátis (o que normalmente apenas deve acontecer em casos especiais). Caso a invocação de um dos métodos da classe invalide uma destas condições, o método não deverá ter qualquer efeito e deverá lançar a excepção *InvalidInvocationException.* A figura 2 apresenta a interface desta classe.

O método *add* da classe *Restaurant* adiciona ao restaurante o prato indicado

com o preço dado. Caso o prato já esteja registado no restaurante, então apenas actualiza o preço do prato no restaurante com o novo preço recebido como argumento.

```
public class Restaurant {

  //creates a restaunrant with the given name, address, initial set of
  //dishes and respective prices. By default the restaurant is not vegetarian
  public Restaurant(String name, String address, List<Dish> dishes, int[] dishPrices)   { /* .... */ }

  //changes the type of this restaurant
  public void setVegetarian(boolean vegetarian) { /* .... */ }

  //checks the type of this restaurant
  public boolean isVegetarian() { /* .... */ }

  //returns the name of this restaurant
  public String getName() { /* .... */ }

  //returns the address of this restaurant
  public String getAddress() { /* .... */ }

  //adds a dish with the given price to this restaurant. If the dish is already present
  //in the restaurant, then it just updates the price asociated with the restaurant.
  public void addDish(Dish d, int price) { /* .... */ }

  //changes price of the dish
  //throws DishNotFoundException if dish does not exist in this restaurant.
  public void changePrice(Dish d, int newPrice) { /* .... */ }

  //removes the dish from this restaurant
  //throws DishNotFoundException if dish does not exist in this restaurant
  public void remove(Dish dish) { /* .... */ }

  //returns the price of the dish in this restaurant
  //throws DishNotFoundException if dish does not exist in this restaurant
  public int getprice(Dish dish)  { /* .... */ }

  //returns the list of dishes of this restaurant
  public List<Dish> getDishes()  { /* .... */ }
  ...
}
```

Figura 2: A interface da classe *Restaurant*.

2.3 A entidade Client

Cada cliente tem um nome e correio eletrónico de contacto. Cada cliente pode ainda registar o seu número de identificação fiscal para constar na emissão da fatura eletrónica. Os clientes podem ser classificados como clientes VIPs ou

clientes normais. Esta classificação pode ser alterada a qualquer momento pelo sistema. A figura 3 apresenta a interface desta entidade.

```
public enum ClientType {
  STANDARD, VIP;
}

public class Client {

  //creates a client with the given name and email.
  public Client(String name, String email)   { /* .... */ }

  //sets the tax identification number of this client
  public void setIdentificationNumber(int taxNumber) { /* .... */ }

  //gets the tax identification number of this client
  public int getIdentificationNumber() { /* .... */ }

  //returns the name of this client
  public String getName() { /* .... */ }

  //returns the email of this client
  public String getEmail() { /* .... */ }

  //returns the type of this client
  public ClientType getClientType() { /* .... */ }

  //returns the total value of shopping made by this client
  //in the last 365 days
  public int getAccumulatedShoppings() { /* .... */ }

  //sets the type of this client
  public void setClientType(ClientType newType) { /* .... */ }

  ...
}
```

Figura 3: A interface da classe *Client*.

2.4 A entidade ShoppingTray

Cada cliente tem um tabuleiro de compras (representado pela classe *ShoppingTray*) que mantém os pratos e respectiva quantidade que o cliente pretende encomendar. Em cada interacção com o portal, o cliente pode actualizar o seu tabuleiro de compras de acordo com determinadas regras. A figura 4 apresenta a interface desta entidade. De seguida descreve-se o comportamento correcto desta classe. Considere que qualquer invocação de um método desta classe numa situação não descrita corresponde a uma invocação inválida e deverá dar origem ao lançamento da excepção *InvalidInvocationException*.

```
public class ShoppingTray {
  public ShoppingTray() { /* .... */ }

  //removes the dish from this shopping tray
  public void remove(Dish dish)   { /* .... */ }

  //adds the dish with the given quantity to this shopping tray
  public void add(Dish dish, int quantity) { /* .... */ }

  //begins the checkout process for this shopping tray
  public void checkout()  { /* .... */ }

  //confirms this shopping tray
  public void confirm(String address)()  { /* .... */ }

  //pays this shopping tray
  public void pay()   { /* .... */ }

  //cancels this shopping tray
  public void cancel()   { /* .... */ }

  //computes the price of this shopping tray
  public int computePrice()   { /* .... */ }

  //returns the invoice of this payed shopping tray
  public Invoide getInvoice()   { /* .... */ }

  //returns the quantity associated with the selected dish
  public int getQuantity(Dish dish)   { /* .... */ }

  //returns the set of dishes presented in this shopping tray
  public List<Dish> getDishes()   { /* .... */ }
}
```

Figura 4: A interface da classe *ShoppingTray*.

Um tabuleiro de compras pode estar nos seguintes estados: *aberto*, *fechado*, *confirmado*, *pago* e *cancelado*. O estado inicial de cada tabuleiro de compras é o *aberto*. O conteúdo do tabuleiro de compras apenas pode ser alterado enquanto o tabuleiro está *aberto*. O conteúdo de um tabuleiro de compras é alterado através dos métodos *add* e *remove*. A invocação do método *add* é válida desde que a quantidade indicada do prato seja um número maior do que zero. A invocação do método *checkout* no estado *aberto* fecha o tabuleiro, não sendo então possível mais alterações ao seu conteúdo. O cliente tem agora que confirmar os pratos encomendados e indicar o endereço de entrega da encomenda (via método *confirm*) ou pode cancelar a encomenda (via *cancel)*. No primeiro caso, o tabuleiro ficará no estado *confirmado*, enquanto que no segundo ficará no estado *cancelado*. No estado *confirmado*, o cliente pode pagar o tabuleiro de compras através do método *pay*, passando o tabuleiro para o estado *pago,* ou então tem ainda uma

última oportunidade para cancelar o pagamento do tabuleiro invocando o método *cancel*. Neste caso, o tabuleiro passará para o estado *cancelado*. É possível obter a factura relativa a um tabuleiro de compras pago através do método *getInvoice*. O método *computePrice* determina o preço dos vários pratos escolhidos pelo cliente e apenas é válido para tabuleiros que ainda não foram pagos ou cancelados.

Em qualquer instante é possível saber o conteúdo do carrinho de compras através dos métodos *getDishes* e *getQuantity*.

2.5 A entidade Rest

A entidade *Rest* gere a informação sobre os restaurantes e clientes registados no sistema. Cada restaurante é identificado univocamente pelo seu nome. Um cliente é identificado univocamente pelo seu endereço de email. A figura 5 apresenta a interface desta entidade.

```
public class Rest {

  //creates a portal with the given name.
  public Rest(String name)   { /* .... */ }

  //registers a new client. The email of this client must be unique
  //throws ClientNotUniqueException if email is not unique
  public void add(Client client) { /* .... */ }

  //registers a new restaurant. The name of this restaurant must be unique.
  //throws RestaurantNotUniqueException if name is not unique
  public void add(Restaurant restaurant) { /* .... */ }

  //gets all restaurants
  public List<Restaurant> getRestaurants() { /* .... */ }

  //gets all clients
  public List<Client> getClients() { /* .... */ }

  //gets the client with the specified email
  public Client getClient(String email) { /* .... */ }

  //computes the discount for the specified client and shopping tray
  public void computeDiscount(Client c, ShoppingTray tray)   { /* .... */ }

  ...
}
```

Figura 5: A interface da classe *Rest*.

O método *computeDiscount* determina o desconto a aplicar a uma compra tendo em conta o conteúdo do tabuleiro de compras, o tipo do cliente e o seu historial de compras. O cálculo deste desconto tem a seguinte forma:

- Se o tabuleiro de compras tiver um custo superiror a 200 euros, o desconto a aplicar é 10% para clientes VIP e 5% para clientes normais.
- Se o tabuleiro de compras tiver um preço entre 100 e 200 euros (inclusive), o desconto é 6% caso o cliente tenha compras no último ano superiores a 2000 euros. Se as compras forem menores ou iguais a 2000 euros, então o desconto a aplicar é igual a 5% para clientes VIP e 3% para clientes normais.
- Se o tabuleiro de compras tiver um custo inferior a 100 euros, o desconto a aplicar é igual a 3% caso o cliente tenha compras no último ano superiores a 2000 euros ou seja um cliente VIP. Se for um cliente normal e tiver compras no último ano inferiores ou iguais a 2000, então o desconto é igual a 0%.

3 Avaliação do projecto

Os casos de teste a desenhar são os seguintes:

- Todos os grupos têm que desenhar os casos de teste ao nível de classe das classes *Restaurant* e *ShoppingTray*.
- Todos os grupos têm que desenhar os casos de teste correspondentes ao método *computeDiscount* da classe *Rest*.
- Todos os grupos têm que desenhar os casos de teste correspondentes ao método *add* da classe *Restaurant*.
- Adicionalmente, é necessário concretizar 6 casos de teste da bateria de testes que testa a classe *Restaurant*. Esta concretização deve ser feita utilizando a framework de testes *JUnit* 4.X.

Todos os casos de teste devem ser desenhados aplicando os padrões de desenho de testes mais apropriados. Na avaliação do projecto serão consideradas as seguintes componentes:

1. O desenho dos casos de teste relacionados com a classe *ShoppingTray* vale entre 0 e 6,5 valores.
2. O desenho dos casos de teste relacionados com a classe *Restaurant* vale entre 0 e 3,5 valores.
3. O desenho dos casos de teste relacionados com o método *computeDiscount* vale entre 0 e 3,5 valores.
4. O desenho dos casos de teste relacionados com o método *add* vale entre 0 e 3,5 valores.

5. A concretização de seis casos de teste relativos à bateria de testes que testa a classe *Restaurant* 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 cadeira. Alunos cuja nota no primeiro teste seja inferior em mais de 5 valores à nota do projecto terão de realizar uma discussão. Caso haja lugar a uma discussão, a nota final do projecto poderá ser individualizada e será a nota obtida 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 de todos os alunos envolvidos (incluindo os que possibilitaram a ocorrência) à disciplina de Qualidade de Software neste ano lectivo.

3.3 Entrega do projecto

O protocolo de entrega do projecto será disponibilizado oportunamente na página da cadeira bem como qual a documentação a entregar.

O prazo de entrega do projecto é dia **7 de Novembro de 2014 às 16:59:00**.

4 Notas finais

Toda a informação referente a este projecto estará disponível na página da cadeira na Secção *Projectos*. O material de apoio tal como links, manuais, e FAQs

encontram-se na mesma Secção. O esclarecimento de dúvidas deve ser feito preferencialmente nas sessões de dúvidas.

Bom Trabalho!