Ricardo Reis, 200262024

Rodrigo Nogueira, 200262002

2020/2021  Professor Fábio Varanda

Projeto POO

Contents

[1. Introdução 2](#_Toc74499509)

[2. Classes Implementadas 3](#_Toc74499510)

[2.1 Restaurante.java 3](#_Toc74499511)

[2.2 Table.java 5](#_Toc74499512)

[2.3 Order.java 6](#_Toc74499513)

[2.4 Item.java 7](#_Toc74499514)

[2.5 Product.java 8](#_Toc74499515)

[2.6 Dish.java 9](#_Toc74499516)

[2.7 History.java 11](#_Toc74499517)

[2.8 Management.java 13](#_Toc74499518)

[2.8.1 addProduct() 13](#_Toc74499519)

[2.8.2 addDrink() 15](#_Toc74499520)

[2.8.3 removeProduct() 16](#_Toc74499521)

[2.8.4 bookTable() 17](#_Toc74499522)

[18](#_Toc74499523)

[2.8.5 editTable() 19](#_Toc74499524)

# Introdução

O objetivo deste relatório é descrever a aplicação desenvolvida para a gestão de um restaurante, utilizando a linguagem *Java* e o paradigma de programação orientada por objetos (*POO*). Foi pedido uma aplicação que permita aplicar soluções para os conceitos aprendidos no decorrer da unidade curricular, os quais permitem satisfazer um número de princípios fundamentais da Engenharia de Software, nomeadamente: a modularidade conceptual e de implementação, a correção de erros e eficiência.

# Classes Implementadas

## Restaurante.java

Graphical user interface, text, application, email

Description automatically generated

Fig. 1 - Classe Restaurant

Text

Description automatically generated

Fig. 2 - Classe RestaurantManagement

A classe *Restaurant*(a *main*) chama um método estático da classe *RestaurantManagement* para não ser necessário criado um objeto desta mesma classe, uma vez que o método serve para verificar se o ficheiro binário já existe. Se esse ficheiro já existir, vai ser lido, caso contrário é criado o objeto da classe *Management* e inicializa o programa.

Graphical user interface, text, application, email

Description automatically generated

Fig. 3 - Método para guardar no ficheiro binário

Graphical user interface, text, application, email

Description automatically generatedEste é o método dentro da classe RestaurantFileHandler que permite ao programa guardar os dados do restaurante em um ficheiro binário.

Fig. 4 - Ler o ficheiro binário

Este método também está dentro da classe *RestaurantFileHandler* que permite ao programa carregar dados do ficheiro binário.

## Table.java

Graphical user interface, text, application

Description automatically generated

Fig. 5 - Classe Table

Nesta classe é definido o número da mesa, o estado de ocupação da mesma e contém apenas um pedido.

## Order.java

Text

Description automatically generated

Fig. 6 - Classe Order

Text

Description automatically generated with low confidence

Fig. 7 - Estados do pedido

A classe *Order* é composta por um conjunto de *Itens*, o seu estado(Fig. 7), a data/hora em que o pedido é aberto e a data/hora de fecho do mesmo. Sempre que um pedido é fechado iguala-se a data/hora atual à variável de fecho(*closeHour*).

## Item.java

Text

Description automatically generated

Fig. 8 - Classe Item

Nesta classe define-se a quantidade de cada produto.

## Product.java

Graphical user interface, text, application

Description automatically generated

´

Fig. 9 - Classe Product

Graphical user interface, text, application

Description automatically generatedA classe *Product* é uma classe abstrata pois existem quatro tipos de produtos e cada um com a própria classe, métodos e atributos. Cada Subclasse derivante herda os atributos: nome, preço e *iva*. O iva é sempre introduzido como número inteiro mas convertido para decimal para facilitar os cálculos.

Fig. 10 - Classe Drink

Na Subclasse *Drink* é definida a capacidade da bebida, tendo um mínimo de 0,33L , e se é alcoólica.

## Dish.java

Graphical user interface, text, application

Description automatically generated

Fig. 11 - Classe Dish

Graphical user interface, text, application, email

Description automatically generatedA classe *Dish* contém a descrição do prato, pois o nome, preço e iva são atributos da classe Product.

Fig. 12 - Classe Snack

Na classe *Snack* é definida a quantidade apesar de estre atributo não afetar o preço total do pedido. Apenas a quantidade definida no Item irá influenciar o preço final.

Graphical user interface, text, application

Description automatically generated

Fig. 13 - Classe Sweet

A classe Sweet assim como a classe *Dish* também tem uma descrição, adicionalmente existe um boolean que indica se o doce é confecionado ou não no próprio restaurante.

## History.java

Graphical user interface, text, application, email

Description automatically generated

Fig. 14 - Método do histórico para ordenar pedidos do mais recente para o mais antigo

A classe *History* tem uma *arraylist* de pedidos(*Order*) como atríbuto e como método o *sortOrder* que é chamado cada vez que é adicionado um novo pedido.

No ciclo do-while existe um objeto da classe *Order*(*temporaryOrder*) que é utilizado para transporte de pedidos entre os índices do orderList e uma variável booleana que por defeito é falsa e convertida para verdadeira sempre que é feita uma ordenação dos índices para sinalizar o recomeço do ciclo pois pode existir mais índices por ordenar.

No ciclo que percorre o arraylist da lista de pedidos(*for-loop*) é verificado se o índice após ao atual é nulo, ou seja, chegou-se ao final do ciclo, se for verdade é lançada uma exeção que dá por terminado o final do ciclo. Em contraste no else-if é comparada a data/hora de abertura do pedido atual com a data/hora do pedido do índice seguinte, caso o pedido atual seja mais antigo que o pedido seguinte, que é retornado um número inteiro negativo, é feita a troca do objeto do pedido do índice atual pelo objeto do índice seguinte.

## Management.java

## addProduct()

Text

Description automatically generated with low confidence

Fig. 15 - Método addProduct() – PARTE 1

Na classe principal para adicionar produtos é utilizado o método addProduct, sempre que o utilizador escolhe o tipo de produto que pretende adicionar é perguntado o nome e em seguinda é verificado se já existe esse mesmo produto baseado no nome.

A picture containing text

Description automatically generated

Fig. 16 - Método addProduct() - PARTE 2

Graphical user interface, text, application, email

Description automatically generatedSe o utilizador introduzir um novo nome para um produto é pedido o preço e iva. Este três atributos são comuns a todos os produtos no programa.

Fig. 17 - Menu de distribuição

Este excerto de código iniciar o método respetivo para a criação de cada Produto.

## addDrink()

Graphical user interface, text, application, email

Description automatically generated

Fig. 18 - Método addDrink()

Os métodos respetivos a cada Produto regem-se pelo mesmo padrão onde é introduzido o valor dos atributos em comum(nome, preço, iva) e em seguida é pedido ao utilizador os dados pertencentes às variáveis das devidas Subclasses do *Product* antes de serem adicionadas ao *arraylist* dos Produtos.

## removeProduct()

Graphical user interface, text, application

Description automatically generated

Fig. 19 - Método removeProduct()

A picture containing application

Description automatically generatedÉ mostrado uma lista numerada de produtos introduzidos previamente, bastando apenas ser necessário inserir o número correspondente ao produto.

Fig. 20 - Exemplo da remoção de produtos

## Graphical user interface, text, application, email Description automatically generated2.8.4 bookTable()

Fig. 21 - Método bookTable() - PARTE 1

Graphical user interface, text, application, email

Description automatically generatedÉ verificado se existem mesas disponiveis para uma reserva, caso existam é demostrado uma lista com as mesas do restaurante e é pedido ao utilizador que mesa, que não esteja já ocupada, deseja efetuar uma reserva.

Fig. 22 - Método bookTable() - PARTE 2

A picture containing text

Description automatically generatedEsta parte do método pergunta ao utilizador se pretende reservar a mesa para uma data futura ou no momento. Se reserva para o momento o programa abre o pedido.

## 

Fig. 23 - Data para reversa futura

Caso seja feita uma reserva para uma data futura, o programa irá pedir pela data que se pretende reservar assim como as hora e minutos com as respetivas validações.

## Text Description automatically generated with medium confidenceeditTable()

Fig. 24 - Método editTable()

Neste método é apresentado mais uma vez a lista de produtos para adicionar um pedido da respetiva mesa, para tal, é auxiliado pelo método *selectTable* que é utilizado para receber a mesa como parametro.

A picture containing logo

Description automatically generated

Fig. 25 - Utilização do selectTable() no menu principal

Também é utilizado como auxilio o método checkTableOrderState para caso a data atual não for superior à data do pedido da mesa, ser possível fechar o pedido ou para serem adicionados mais produtos, passar o estado do pedido para servido e fechar o pedido, sendo este introduzido no histórico.