Relatório Terceiro Projeto de POO

Gestor de compras online



# Eduardo José Gonçalves Nunes nr.2020217675

# Introdução

Este relatório tem por base explicar a estrutura e os algoritmos do projeto desenvolvido. Começando por descrever a estrutura em geral do programa e o propósito de todas as classes

# Estrutura em geral do programa

## Algoritmo de execução

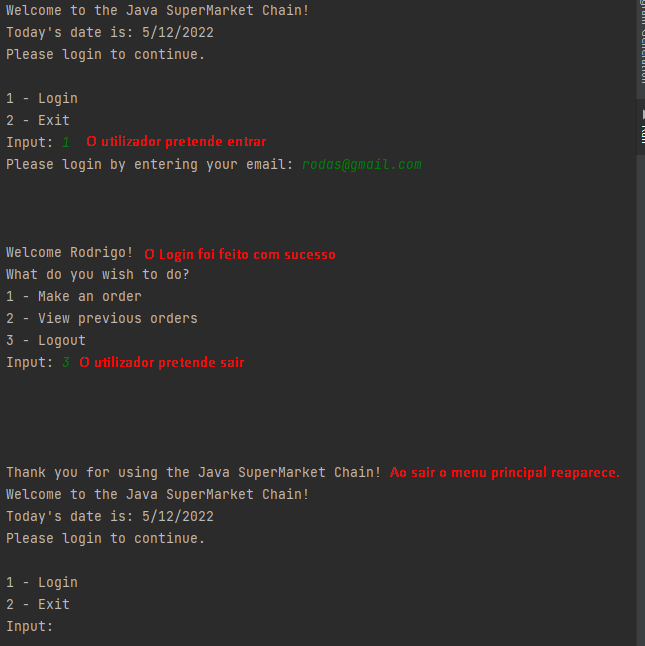
O programa baseia-se num loop infinito. Cada vez que o utilizador faça *Logout*, o menu de autenticação reaparece, tal como numa aplicação normal.

Uma imagem com texto

Descrição gerada automaticamente

Snippet de código da função Main

As variáveis **Products** e **Promotions** que podem ser modificadas são recarregadas (construídas) por cada autenticação feita, sendo a classe ***Auth*** a classe responsável pela autenticação do utilizador. Uma vez o *Logout* feito, é apresentado outra vez um menu de Autenticação ao utilizador caso pretenda entrar como outro **Cliente**.

O principal uso do loop é para o utilizador poder estar sempre a entrar com contas distintas sem ter que fechar a aplicação.

Este Loop apenas volta a criar um objeto **Auth** cada vez que o utilizador faz Logout.

## Classes Principais (essenciais ao funcionamento do programa)

* **Classe – Main**

Nesta classe iniciamos a nossa aplicação, aqui inicializamos alguns objetos necessários ao funcionamento da nossa aplicação, tais como:

* O objeto **Scanner**, que nos vai permitir interagir com o utilizador;
* O objeto ***Date now***, que representa a data a que o utilizador acede ao programa. Este objeto serve para ver quais são as promoções aplicáveis no instante que o utilizador acede ao programa.
* O objeto **Promotions**, que contem todas as promoções aplicáveis dependendo da Data atual (o objeto *Date now)*.
* O objeto **Products**, contido num “loop” (devido ao facto de no *logout* do cliente queremos atualizar os produtos disponíveis para o próximo *login)*
* E, por fim, o objeto de Autenticação **(Auth)**, também contido num “loop” caso o utilizador queira sair da aplicação,que gera os menus de autenticação, com a ajuda do **Scanner**, para o utilizador aceder aos seus produtos.
* **Superclasse – Cliente (Customer) e as suas Subclasses.**

Uma classe que representa um Cliente juntamente com todos os seus dados e operações atribuídas dependendo da sua *subclasse*:

* **Cliente Regular (Regular Customer)**
* **Cliente Frequente (Frequente Customer)**

É necessário criar subclasses da superclasse **Cliente** porque o cálculo para o valor monetário de transporte (o método para obter o preço do transporte) varia consoante o tipo de **Cliente**.

* **Classe – Clientes (Customers)**

O propósito desta classe é aceder a todos os clientes na base de dados e fazer operações neles. Aqui carregamos os **Clientes** disponíveis do ficheiro de texto (se o ficheiro de objetos não existir) para o ficheiro de objetos e verificamos se a nossa base de dados é defeituosa, juntamente com outras operações úteis para o resto do programa, tais como, retornar um **Cliente** somente pelo seu email (algo muito útil quando se faz **Login**) e ver se um **Cliente** já existe na base de dados atual (algo muito útil para validar se os ficheiros como base de dados são defeituosos ou, no futuro, se se implementar um método de **registo** **de clientes** além do **Login**).

* **Classe – Compra (Order)**

O propósito desta classe é conter os dados de uma compra feita por um **Cliente**, como por exemplo, o valor da compra em euros, a data a que foi feita, os produtos escolhidos…

* **Classe – Orders (Compras)**

A função desta classe é simplesmente organizar todas as compras de um **Cliente**, visto que, um **Cliente** pode fazer mais que uma compra. Esta classe é essencial para o programa visto que é ela que permite ao utilizador verificar todas as suas compras na aplicação, juntamente com a sua **Data**.

* **Superclasse – Produto (Product) e as suas Subclasses**

Esta classe representa um **Produto** juntamente com todos os seus dados, como por exemplo, o seu stock, algo muito importante para averiguar se o **Produto** ainda pode ser comprado por **Clientes**, o seu preço por unidade, o seu nome, o seu preço de transporte, entre outros dados também, alguns somente pertencentes às suas extensões. As suas *subclasses* são importantes devido aos novos dados que um produto pode assumir, como a **toxicidade** caso este seja um **Produto de Limpeza** e ao cálculo do seu transporte que pode variar se for um **Produto de Mobiliário** e o seu peso for maior que 15kg. Todas as suas *subclasses* são:

* **Produto de Limpeza (Cleaning Product)**
* **Produto de Mobiliário (Furniture Product)**
* **Produto Alimentar (Food Product)**
* **Classe – Produtos (Products)**
* **Superclasse – Promoção (Promotion) e as suas Subclasses**
* **Classe – Promoções (Promotions)**

E finalmente, a classe **Promoções** que salva e lê as Promoções existentes para o respetivo ficheiro. É importante existir esta classe para podermos verificar qual é a promoção numa **Compra** especifica ou até mesmo se é aplicável alguma. Encontramos

### Classes Secundárias (que facilitam a utilização e organização do programa)

*É importante referir que uma classe secundária não é uma classe que pode ser removida do programa, mas sim uma que complementa o seu uso tornando a experiência do utilizador e do desenvolvedor mais agradáveis.*

* **Classe – Autenticação (Auth)**

Esta classe ajuda a estruturar o código pois separa o loop principal da função *main* da autenticação e validação da mesma que o utilizador tem que fazer. Esta valida o *login* e, se este for bem-sucedido, cria um objeto do tipo **Login (LoggedIn)** a que será associado um **Cliente (Customer)**.

* **Classe – Login (LoggedIn)**

Esta classe, juntamente com a classe de **Autenticação (Auth)**, simplificam a estrutura do programa porque separam a autenticação e a pós autenticação em dois objetos diferentes. Ao ser iniciada (construída), é apresentado um menu ao utilizador autenticado, este pode escolher se pretende fazer um novo **Pedido (Order)** ou se pretende apenas ver os seus **Pedidos (Orders)** passados**. É nesta classe que acontece a maior parte de interação entre o utilizador e a aplicação.**

* **Classe – Data (Data)**

Esta classe presta-nos auxílio a fazer operações com Datas num só objeto, tais como, ver se uma **Data** está depois ou antes de outra, ver se uma **Data** é válida, ou seja, se é possível de existir, ver se duas Datas são iguais… Esta é útil devido à necessidade de verificar se uma promoção se aplica à data a que o programa é lançado.