1 2 9 0

UNIVERSIDADE D
COIMBRA

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 o algoritmo de execução e, de seguida, 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.

```
// loop where the application runs
1. while (true) { // Loop until user from inside the loop
2.     Products products = new Products(); // load Products object
3.     Promotions promotions = new Promotions(); // load Promotions object
4.
5.
6.     new Auth(sc,products,now,promotions);  //Interface for login
7. }
```

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 em contas distintas sem ter que fechar a aplicação.

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

```
Welcome to the Java SuperMarket Chain!
Today's date is: 5/12/2022
Please login to continue.

1 - Login
2 - Exit
Input: 1   O utilizador pretende entrar
Please login by entering your email: rodas@gmail.com



Welcome Rodrigo!  O Login foi feito com sucesso
What do you wish to do?
1 - Make an order
2 - View previous orders
3 - Logout
Input: 3  O utilizador pretende sair



Thank you for using the Java SuperMarket Chain!  Ao sair o menu principal reaparece.
Welcome to the Java SuperMarket Chain!
Today's date is: 5/12/2022
Please login to continue.

1 - Login
2 - Exit
Input:
```

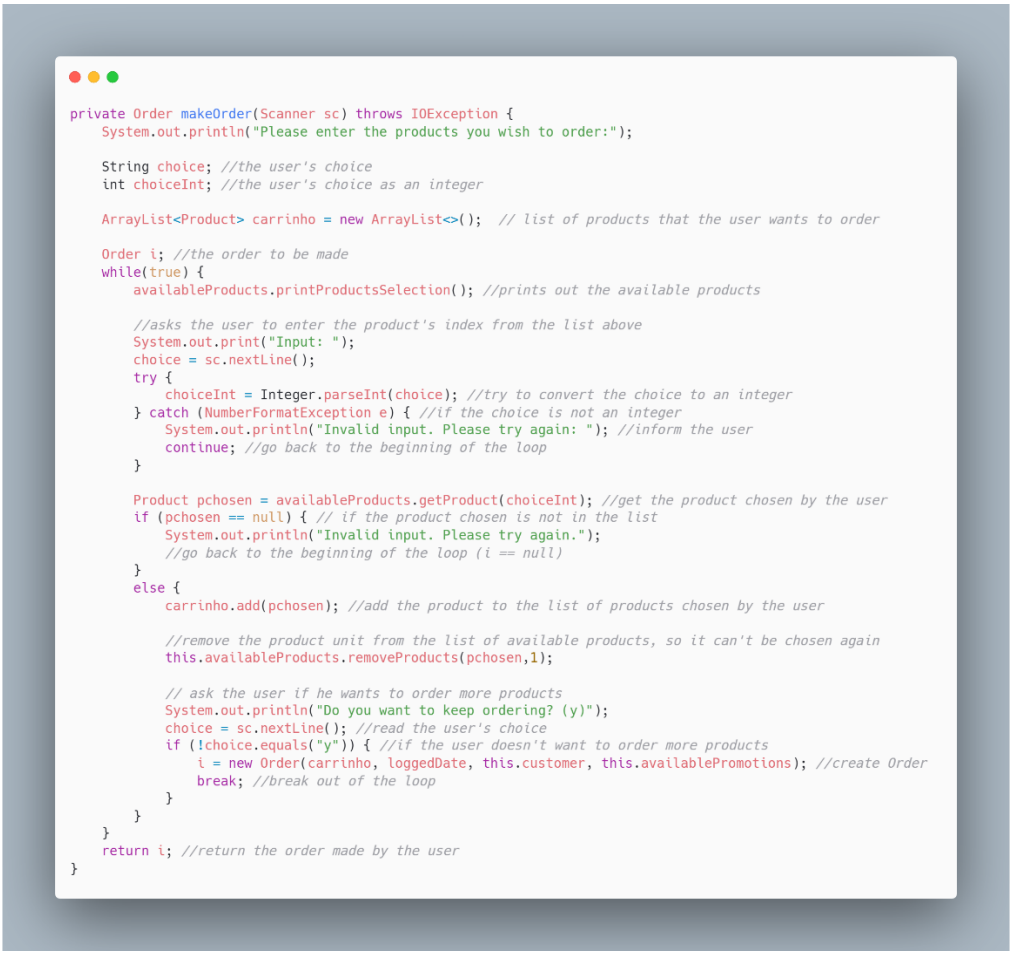
Exemplo de um utilizador que fez Login e Logout, é importante notar que o menu de Login reaparece.

Uma vez selecionado e feito o *Login* com sucesso, é criado um novo objeto (uma nova instância da classe **LoggedIn**) que, no seu construtor chama o menu principal de interação com o utilizador onde este pode fazer compras, ver as suas compras passadas e fazer **Logout**.

```
private void menu(Scanner sc) {
    String choice; //the user's choice
    int choiceInt; //the user's choice as an integer
    while (true) {
        System.out.print("1 - Make an order\n2 - View previous orders\n3 - Logout\nInput: ");
        choice = sc.nextLine(); //read the user's choice
        try {
            choiceInt = Integer.parseInt(choice);

        } catch (NumberFormatException e) { //if the choice is not an integer
            continue; //go back to the beginning of the loop
        }

        switch (choiceInt) {
            case 1:
                Order Ords = makeOrder(sc); //returns an Order object created by the user
                this.customer.appendOrders(Ords); //updates all customers with the new orders
                this.cs.savecustomersOBJ(); //saves the customers in .obj file
                break;
            case 2:
                viewOrders();
                break;

            case 3:
                System.out.println("\n\n\n\nThank you for using the Java SuperMarket Chain!");
                return;

            default: // if the choice is not 1, 2 or 3
                System.out.println("Invalid input. Please try again.");
                break;
        }
    }
}
```

Função menu que faz a principal interação com o utilizador LoggedIn

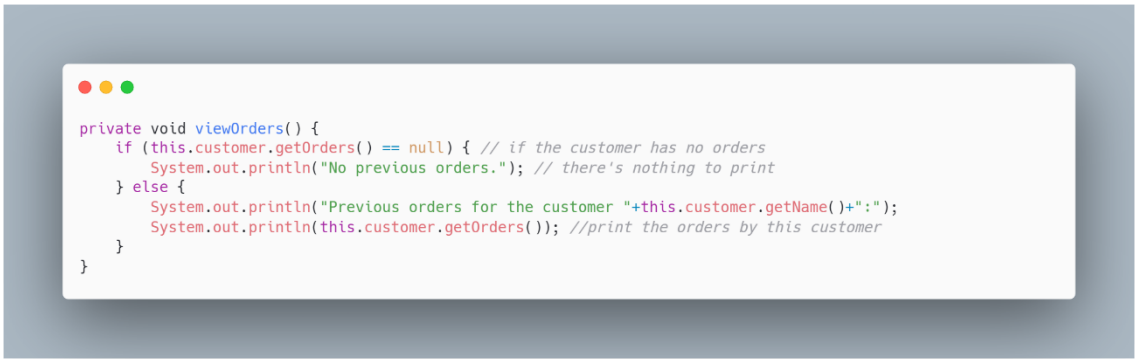
Para fazer compras, ou seja, pressionando 1, é retornada uma nova compra pelo utilizador, sendo esta associada ao cliente **LoggedIn** e, imediatamente a seguir, é salvo todos os clientes para o ficheiro de objetos de modo a atualizar a nova compra do **Cliente**. Para criar uma compra, é ser apresentado ao utilizador todos os **Produtos** disponíveis juntamente com um número lhe associado de modo a facilitar a escolha destes. Quando o **Cliente** seleciona o produto desejado, este adiciona-se a uma ***ArrayList*** de **Produtos** (descontando-se, também por um a **o stock do Produto selecionado** para, no caso de não haver mais produtos escolhidos, o utilizador não os poder escolher) e, quando o utilizador está satisfeito com as compras feitas, sai do loop criando assim uma nova **Compra** e retornando-a.



```java
private Order makeOrder(Scanner sc) throws IOException {
    System.out.println("Please enter the products you wish to order:");

    String choice; //the user's choice
    int choiceInt; //the user's choice as an integer

    ArrayList<Product> carrinho = new ArrayList<>();  // list of products that the user wants to order

    Order i; //the order to be made
    while(true) {
        availableProducts.printProductsSelection(); //prints out the available products

        //asks the user to enter the product's index from the list above
        System.out.print("Input: ");
        choice = sc.nextLine();
        try {
            choiceInt = Integer.parseInt(choice); //try to convert the choice to an integer
        } catch (NumberFormatException e) { //if the choice is not an integer
            System.out.println("Invalid input. Please try again: "); //inform the user
            continue; //go back to the beginning of the loop
        }

        Product pchosen = availableProducts.getProduct(choiceInt); //get the product chosen by the user
        if (pchosen == null) { // if the product chosen is not in the list
            System.out.println("Invalid input. Please try again.");
            //go back to the beginning of the loop (i == null)
        }
        else {
            carrinho.add(pchosen); //add the product to the list of products chosen by the user

            //remove the product unit from the list of available products, so it can't be chosen again
            this.availableProducts.removeProducts(pchosen,1);

            // ask the user if he wants to order more products
            System.out.println("Do you want to keep ordering? (y)");
            choice = sc.nextLine(); //read the user's choice
            if (!choice.equals("y")) { //if the user doesn't want to order more products
                i = new Order(carrinho, loggedDate, this.customer, this.availablePromotions); //create Order
                break; //break out of the loop
            }
        }
    }
    return i; //return the order made by the user
}
```

Função que permite ao utilizador fazer novas compras

Para ver as suas compras, ou seja, pressionando 2, é chamada uma função que com o atributo da Classe **LoggedIn** que corresponde ao **Cliente** que fez a autenticação chama todas as compras do **Cliente** em causa.



```java
private void viewOrders() {
    if (this.customer.getOrders() == null) { // if the customer has no orders
        System.out.println("No previous orders."); // there's nothing to print
    } else {
        System.out.println("Previous orders for the customer "+this.customer.getName()+":");
        System.out.println(this.customer.getOrders()); //print the orders by this customer
    }
}
```

*Função que mostra todas as compras do **Cliente***

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** (independentemente do seu tipo) na base de dados, guardados, depois de lidos, numa *ArrayList*, 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. Estas são guardadas numa *ArrayList*

de compras. Esta classe é essencial para o programa visto que ela 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)**
  A classe **Produtos** representa todos os produtos disponíveis **de todos os tipos**. Esta classe é de extrema importância pois se um determinado **Produto** não estiver disponível, ele não consta nesta classe e, portanto, não aparece na lista de seleção quando um **Cliente** estiver a fazer uma **Compra**. Além de conter todos os **Produtos**, esta classe também é responsável por guardar e ler no respetivo ficheiro os produtos disponíveis atualizados, por exemplo, se a loja ficar sem stock num **Produto** qualquer, ele não será guardado no ficheiro de objetos, porque já não está mais em stock.

- **Superclasse – Promoção (Promotion) e as suas Subclasses**
  Esta classe representa uma **Promoção**, uma promoção é apenas composta pela sua **Data** de início e **Data** do fim. O método mais importante da Superclasse é ver se a **Compra** é aplicável sendo que os diferentes algoritmos de cálculo do seu desconto veem nas suas *Subclasses*:
    - **Promoção Pague-Três-Leve-Quatro (PTTF)**
    - **Promoção Pague-Menos (PL)**

- **Classe – Promoções (Promotions)**
  E finalmente, a classe **Promoções** que salva e lê as **Promoções** existentes para o respetivo ficheiro (.txt ou .obj). É importante existir esta classe para podermos verificar qual é a promoção numa **Compra** especifica ou até mesmo se é aplicável alguma. Se existir, encontramos a promoção aplicável percorrendo a *ArrayList* desta classe.

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 ou que seja desinteressante para o programa geral, é sim uma classe 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** a que será associado ao **Cliente** que fez a **Autenticação.**

- **Classe – Login (LoggedIn)**
  Esta classe, juntamente com a classe de **Autenticação,** 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** ou se pretende apenas ver os seus **Pedidos** passados**. É nesta classe que acontece a maior parte de interação entre o utilizador e a aplicação** visto que as **Compras** e a sua consulta são chamadas a partir desta classe.

- **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 assim que o programa é lançado, rapidamente.