Universidade do Minho

Escola de Engenharia

Programação Orientada aos Objetos

Licenciatura em Engenharia Informática

Vintage

Grupo 101

Gonçalo Gonçalves - [A90969]
Henrique Pereira - [A100831]
Henrique Vaz - [A95533]

Maio, 2023

Conteúdo

1. Introdução

Este relatório tem como objetivo apresentar um sistema de marketplace chamado Vintage para compra e venda de artigos novos e usados de vários tipos. O sistema permite aos utilizadores registados assumir o papel de vendedor ou comprador, adicionando novos artigos para venda ou optando pela compra dos artigos disponíveis.

O sistema Vintage mantém controlo sob o stock dos artigos disponíveis, das encomendas efetuadas pelos compradores e das vendas efetuadas pelos vendedores.

Será possível a um vendedor publicar os seus artigos e escolher a transportadora que tratará da expedição, bem como ao comprador adicionar vários artigos a uma encomenda e concluir a compra. Além disso, o sistema prevê a devolução da encomenda, mas somente dentro de um prazo de 48 horas após a entrega da encomenda.

O sistema é também capaz de efetuar estatísticas sobre vários aspetos da loja.

Também é importante destacar que o sistema permite a simulação de passagem de tempo, o que permite avançar para uma data no futuro e testar a mudança de estado das encomendas.

Este relatório abordará detalhadamente os requisitos funcionais e não funcionais do sistema, a arquitetura proposta, as tecnologias e metodologias usadas.

2. Arquitetura

A arquitetura deste programa segue o MVC (Model-View-Controller). Assim, é observável que existem três componentes principais neste trabalho: o Model, que assegura a parte das regras e camada computacional, sendo constituído por todas as classes exceto as classes *Controller*, *View* e *Populator*; a View, que corresponde ao código que faz a interação com o utilizador recolhendo informação e comunicando-a ao *Controller*, assim como transmitindo informação proveniente do *Controller* para o utilizador; e o Controller, que faz a ligação entre a View e o Model. Com este modelo de arquitetura temos então o código mais organizado e é mais fácil de localizar métodos quando necessário.

3. Diagrama de Classes

Figura 3.1: Diagrama de Classes

4. Classes

4.1 Model

4.1.1 Transportadora

```
public final double PEQUENAS;
public final double MEDIAS;
public final double GRANDES;
public final double IMPOSTO;
private double lucro;
private boolean premium;
private String nome;
private double dinehiro_feito;
```

Classe que representa transportadoras.

As variáveis de instância apresentadas têm o seguinte significado:

- novo - Se o artigo é novo ou não (true ou false)
- n_donos - O número de donos caso o artigo seja usado
- estado - O estado do artigo (pessimo, mau, razoavel, bom e muito bom)
- desc - Uma breve descrição do artigo
- marca - A marca do artigo
- codAlfaNum - O código alfanumérico do artigo
- preco_base - O preço base do artigo
- transp - A transportadora que será usada para transportar o artigo

4.1.2 TransportadoraPremium

Sub-classe da classe transportadora que implementa a interface premium, é usada para distinguir quais transportadoras podem transportar artigos premium e quais apenas podem transportar artigos normais. O calculo do preço é também atualizado.

4.1.3 Artigo

```
public enum Estado {PESSIMO,MAU,RAZOAVEL,BOM,MUITO_BOM};

private boolean novo;
private int n_donos;
private Estado estado;
private String desc;
private String marca;
private String codAlfaNum;
private double preco_base;
private Transportadora transp;
```

Esta classe é abstrata e é a superclasse das classes Mala, Sapatilha e Tshirt (subclasses). Como todos os artigos são comprados e vendidos da mesma forma então faz sentido usar o conceito de herança e atribuir a cada uma destas 3 classes uma implementação base em comum(Eg.:Mala e Sapatilha sao do tipo Artigo e partilham em comum uma marca, descrição e um estado).

As variáveis de instância apresentadas têm o seguinte significado:

- novo - Se o artigo é novo ou não (true ou false)
- n_donos - O número de donos caso o artigo seja usado
- estado - O estado do artigo (pessimo, mau, razoavel, bom e muito bom)
- desc - Uma breve descrição do artigo
- marca - A marca do artigo
- codAlfaNum - O código alfanumérico do artigo
- preco_base - O preço base do artigo
- transp - A transportadora que será usada para transportar o artigo

4.1.4 Mala

```
public enum Dim {PEQUENO,MEDIO,GRANDE}

private Dim dimensao;
private String material;
private LocalDate colecao;
private double preco_curr;
```

Esta classe representa artigos do tipo Mala.

As variáveis de instância apresentadas têm o seguinte significado:

- dimensao - Dimensão da mala (pequeno,medio,grande)

- material - Descrição do material da mala
- colecao - Data da coleção da mala
- preco_curr - Preço atual do artigo

4.1.5 MalaPremium

Sub-classe da classe mala que implementa a interface premium, é usada para calcular o preço de uma mala premium.

4.1.6 Sapatilha

```
private double tamanho;
private boolean atacadores;
private String cor;
private LocalDate colecao;
```

Esta classe representa artigos do tipo sapatilha.

As variáveis de instância apresentadas têm o seguinte significado:

- tamanho - Tamanho das sapatilhas
- atacadores - Se a sapatilha têm atacadores ou não
- cor - A cor da sapatilha
- colecao - Data da coleção da mala

4.1.7 SapatilhaPremium

Sub-classe da classe sapatilha que implementa a interface premium, é usada para calcular o preço de uma sapatilha premium.

4.1.8 Tshirt

```
public enum Tamanho {S,M,L,XL};

public enum Padrao {Riscas,Liso,Palmeiras};

private Tamanho tamanho;
private Padrao padrao;
private double preco_curr;
```

Esta classe representa artigos do tipo tshirt.

As variáveis de instância apresentadas têm o seguinte significado:

- tamanho - Tamanho da tshirt (S,M,L,XL)
- padrao - Padrão da tshirt (riscas,liso,palmeiras)
- preco_curr - Preço atual da tshirt

4.1.9 Encomenda

```
public enum Embalagem {Pequeno,Medio,Grande};

public enum Padrao {Pendente,Finalizada,Expedida};
private static int count;
private Map⟨String, Artigo⟩ artigos;
private Embalagem dim;
private double preco;
private LocalDate data;
private State estado;
```

Esta classe representa encomendas

As variáveis de instância apresentadas têm o seguinte significado:

- count - Contador que mantém o número de encomendas criadas
- artigos - Map dos artigos que pertencem à encomenda
- dim - Dimensão da embalagem da encomenda (pequeno,medio,grande)
- preco - Preço totol da encomenda
- data - Data em que foi efetuada a encomenda
- estado - Estado da encomenda (pendente, finalizada, expedida)

Para guardar os artigos na encomenda foi escolhido um map com o código da encomenda como chave e o objeto artigo como conteúdo, desta forma é possivel procurar instantâneamente a coleção usando o código alfanumérico.

4.1.10 Utilizador

```
private String email;
private String nome;
private String morada;
```

```
private Set⟨String⟩ vendido;
private Set⟨String⟩ para_vender;
private double dinheiro_vendas;
private double dinheiro_compras;
```

Esta classe representa utilizadores.

As variáveis de instância apresentadas têm o seguinte significado:

- email - Email do utilizador
- nome - Nome do utilizador
- morada - Morada do utilizador
- vendido - Conjunto de artigos vendidos pelo utilizador
- para_vender - Conjunto de artigos que o utilizador têm para vender
- dinheiro_vendas - Dinheiro ganho pelo utilizador em vendas
- dinheiro_compras - Dinheiro gasto pelo utilizador em compras

Para os artigos vendidos e para vender escolhemos usar sets pois não teremos de fazer muitas pesquisas e o propósito maior destes sets será serem impressos para o utilizador, pelo que teriam sempre de ser totalmente percorridos.

4.1.11 Populator

A classe populator cria os objetos necessários das variadas classes para popular a loja com dados de teste.

4.1.12 Vintage

```
private Map⟨Integer,Encomenda⟩ encomendas;
private Map⟨Integer,String⟩ encomendas_utilizadores_ligacao;
private Map⟨String,Utilizador⟩ utilizadores
private Map⟨String,Transportadora⟩ transportadoras
private Map⟨String,String⟩ artigos_utilizadores_ligacao
private Map⟨String,Artigo⟩ artigos
private Map⟨String,Artigo⟩ artigos_vendidos
```

- encomendas - Map das encomendas existentes
- encomendas_utilizadores_ligacao - Map de ligação entre encomendas e utilizadores
- utilizadores - Map de utilizadores existentes

- transportadoras - Map de transportadoras existentes
- artigos_utilizadores_ligacao - Map de ligação entre artigos e utilizadores
- artigos - Map de artigos existentes
- artigos_vendidos - Map de artigos vendidos

Para as encomendas escolhemos usar um Map pela possibilidade de encontrar as encomendas instantâneamente pelo seu id.

Com o id de uma encomenda é possível encontrar o utilizador que a comprou usando o Map encomendas_utilizadores_ligacao.

O Map dos utilizadores permite encontrar utilizadores usando o seu e-mail.

O Map das transportadoras permite encontar uma transportadora através do seu nome.

Com o código alfanumérico de um artigo é possível encontrar o seu vendedor usando o Map artigos_utilizador_ligacao

O Map artigos têm o código alfanumérico dos artigos como chave e como conteúdo o objeto artigo.

O Map artigos_vendidos têm o código alfanumérico dos artigos como chave e como conteúdo o objeto artigo que foi vendido.

4.2 View

A classe view não têm variáveis de instância, o seu propósito é apenas servir de "interface"entre o utilizador e o programa.

4.3 Controller

```
private View view;
private Vintage vintage;
private boolean run;
private String current_user;
private boolean logged;
```

O controller terá apenas um objeto da classe view e vintage, será o controller que fará a ligação entre ambos.

5. View

Todas as interações feitas pelo utilizador com a vintage são geridas pela classe *View*, esta classe é responsável por fazer print dos menus e informação para o utilizador e também por receber o input do utilizador.

Para além disso, ao interagir com o utilizador assegura sempre que os valores possíveis de serem escritos são assegurados, assim como o tipo de classe que é suposto.

6. Controller

O *Controller* é a classe responsável pela ligação entre o *Model* e a *View*, recorrendo à *View* para interagir com o utilizador e receber dados provenientes dele de forma a poder enviar dados para serem armazenados ou serem efetuados cálculos no *Model*, caso seja pedido, esses mesmos dados podem ser enviados de volta para a *View* de modo a serem apresentados ao utilizador. Resumidamente, serve como um intermediário, uma vez que conhece toda a implementação.

É no *Controller* que se encontram todos os métodos que arrancam os vários menus, procurando sempre respeitar o encapsulamento e fornecer a máxima proteção através do uso de *conditions* associadas a ciclos e *exceptions* de forma a que o programa nunca seja forçado a encerrar.

Para além disso, o Controller guarda também o utilizador que dê *login* na aplicação, tentando simular ao máximo um website do género, e dois *booleans* que têm o objeto de indicar se já está alguem *logged in*, para que vários utilizadores possam entrar e sair ao longo do mesmo programa e o outro *boolean* para dizer ao programa quando encerrar.

7. Execução

7.1 Carregamento de dados

No arranque do programa encontram-se duas opções de carregamento de dados, carregar dados usando a classe Populator, que funciona como um *script*, ou o carregamento do estado a partir de um ficheiro de objetos recorrendo ao processo de serialização.

```
Indique como pretende carregar os dados:
1. Popular com os dados de teste
2. Ficheiro de Objetos
```

Figura 7.1: Menu de carregamento de dados

Assim que existirem dados em memória, podemos começar a realizar operações. Com isto, iremos demonstrar como utilizar o resto do programa.

7.2 Login/Sign-in

Após os dados carregados o utilizador pode efetuar o login/sign-in ou terminar o programa.

```
--------Vintage--------
1. Login/Sign-in
0. Terminar Programa
```

Figura 7.2: Menu de login

Se o utilizador escolher a opção de login será então pedido o seu email, caso o utilizador já exista então o login é efetuado, se não existir será então pedida mais informação ao utilizador para criar a sua conta.

```
--------Vintage--------
Insira o seu email:
```

Figura 7.3: Email

7.3 Menu principal

Após o login efetuado é então apresentado ao utilizador todas as operações que pode realizar na nossa loja, sendo elas as opções representadas na Fig 7.4:

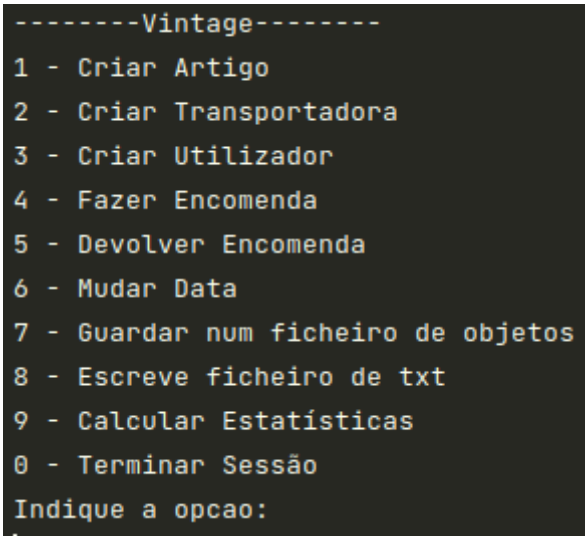

Figura 7.4: Menu principal

Passaremos agora a explicar cada uma das opções do menu.

7.4 Criar Artigo

Quando o utilizador escolhe criar um artigo terá de preencher a informação relativa ao mesmo.

Primeiramente terá de escolher se o artigo é normal ou premium.

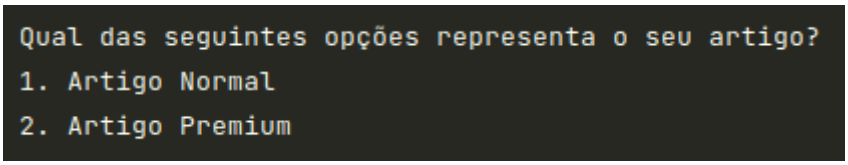

Figura 7.5: Tipo de artigo

De seguida o utilizador escolherá a transportadora a usar, serão apresentadas as transportadoras premium caso o artigo seja premium ou as transportadoras normais caso o artigo seja normal.

```
Transportadora :: dhl, Lucro: 0.06, Dinheiro em Vendas: 0.0
Transportadora :: ctt, Lucro: 0.07, Dinheiro em Vendas: 0.0
Transportadora :: gls, Lucro: 0.05, Dinheiro em Vendas: 0.0
De entre as Transportadoras anteriores escreva o nome da que pretende:
```

Figura 7.6: Escolher transportadora

o utilizador têm então de especificar o tipo de artigo que quer criar.

```
Que tipo de artigo queres criar?
1. Mala
2. Sapatilha
3. Tshirt
0 - Voltar ao Menu Inicial
```

Figura 7.7: Género de artigo

Posto isto serão então pedidas as informações respetivas ao artigo tal como visto na explicação das classes 4.1.3.

7.5 Criar transportadora

Quando o utilizador escolhe adicionar uma transportadora ao sistema apenas terá de introduzir a seguinte informação:

- Nome da transportadora
- Taxa de lucro da transportadora (entre 0.00 e 1.00)
- Se a transportadora é premium ou não

7.6 Criar utilizador

Na criação de um novo utilizador serão repetidos os passos feitos quando é necessário criar uma conta ao fazer login sendo que o utilizador que foi agora criado não fica com o login feito, apenas é introduzido no sistema (Pode servir por exemplo para um administrador ajudar um cliente a criar a sua conta).

7.7 Fazer encomenda

Ao realizar a encomenda o utilizador pode escolher uma das opções abaixo

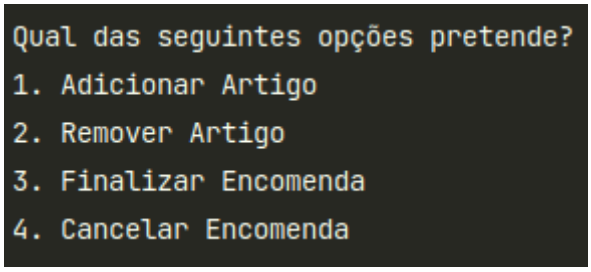

Figura 7.8: Menu fazer encomenda

Ao escolher a opção de adicionar artigo serão então apresentados ao utilizador todos os artigos da loja (menos os que o utilizador está a vender), para adicionar o artigo à encomenda basta escrever o seu código alfanumérico.

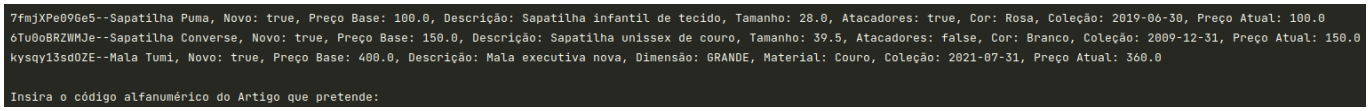

Figura 7.9: Apresentação dos artigos

7.8 Devolver encomenda

Para devolver encomendas o utilizador apenas têm de introduzir o número da encomenda a devolver, sendo que só lhe serão apresentadas como devolviveis encomendas com menos de 48h.

Se o utilizador escolher uma encomenda que não exista ou que não esteja atribuida a si o sistema irá emitir um aviso.

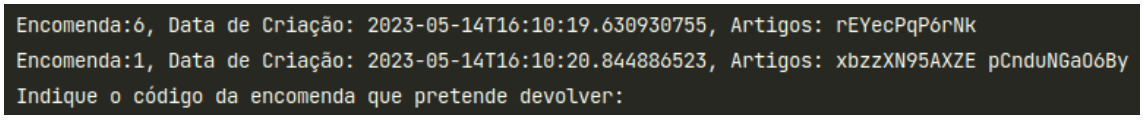

Figura 7.10: Devolver encomenda

7.9 Mudar data

A opção de mudança de data simula a passagem do tempo, é pedido um número de horas para avançar no sistema.

7.10 Guardar num ficheiro de objetos

A opção de guardar o estado num ficheiro de objetos guarda toda a informação atualmente carregada na loja.

7.11 Escreve ficheiro de txt

A opção de escrever o estado num ficheiro de texto permite aos administradores a visualização do estado da loja num dado momento.

7.12 Calcular estatisticas

Quando o utilizador escolhe calcular estatísticas estas serão calculadas com base na informação na secção 8.

8. Estatísticas

8.1 Qual é o vendedor que mais faturou num período ou desde sempre

Para o cálculo desta estatística decidimos que não seria necessário separar o cálculo de um determinado período de tempo de para desde sempre, uma vez que desde sempre basta colocar uma data final superior à data atual guardada no sistema.

Deste modo, primeiramente recolhemos duas datas, sendo que a primeira tem de ser sempre inferior à segunda, voltando sempre a pedir novas datas quando isto não se verifica.

De seguida, armazenamos num *Map*, cuja *Key* é o nome do utilizador (vendedor), os *values* que correspondem a uma *List* com todas as encomendas que ele vendeu no período de tempo recolhido anteriormente.

Por último, convertemos este *Map* numa *Stream* e com auxílio de um *Comparator* pré-definido para *Double* ordenamos pelos valores resultantes do cálculo da receita feita em todas as vendas armazenadas, às quais o método *max* filtra o maior, cuja *Key* corresponderá ao vendedor que mais faturou.

8.2 Qual o transportador com maior volume de faturação

Esta estatística é feita de uma forma quase automática uma vez que o volume de faturação de cada transportadora está armazenado numa das suas variáveis de instância, pelo que é só necessário converter a coleção das transportadoras em *Stream*, ordenar recorrendo ao *Comparator* mencionado na estatística anterior, aplicado ao *getter* dessa variável de instância. Depois também o método *max* devolverá a transportadora com maior volume de faturação.

8.3 Listar as encomendas emitidas por um vendedor

Para esta estatística é inicialmente necessário recolher o nome de um utilizador para poder ser efetuada. Este utilizador caso não exista, ou caso não tenha encomendas emitidas associadas estará protegido por *Exceptions*. De seguida será guardado num *Set* todas as encomendas que o utilizador escolhido vendeu, informação essa que está armazenada em cada utilizador, pelo que basta recorrer a um *getter*.

De seguida, caso existam encomendas, estas serão listadas no *stdout*.

8.4 Fornecer uma ordenação dos maiores compradores/-vendedores do sistema durante um período a determinar

Para esta estatística, dividimos em compradores e vendedores o seu cálculo, no entanto não vimos necessidade em dividir na sua apresentação, pelo que ao efetuar esta estatística serão apresentados ambos.

Assim, de forma semelhante à estatística apresentada em 8.1, vão ser recolhidas duas datas correspondentes ao período de tempo.

Tanto num como noutro vão ser recolhidas todas as encomendas associadas ao utilizador nesse período de tempo e armazenadas num *Map*, sendo que para os vendedores vão ser obtidas as encomendas onde um dos seus artigos foi vendido, e para os compradores as encomendas que este comprou.

De seguida, é criado um *Map* cuja *Key* vai ser o nome do utilizador e o *Value* o dinheiro gasto/feito caso seja comprador/vendedor. Assim, é criado um ciclo pelo

```
Map<String,List<Encomenda>>
```

, calculado o valor total correspondente a todas as encomendas e armazenado no último *Map* que foi criado. No fim, basta criar uma *Stream* do

```
Map<String,Double>
```

criado, ordenar pelos *Values*, e guardar numa *List* todos os nomes dos utilizadores, que serão depois apresentados.

8.5 Determinar quanto dinheiro ganhou o Vintage no seu funcionamento

Considerando que o dinheiro que a Vintage ganha corresponde às taxas de satisfação cobradas por artigo em cada encomenda, para calcular este valor a estratégia adotada foi percorrer a coleção das Encomendas recorrendo a *Streams*, por cada encomenda verificar se o artigo é novo ou usado, uma vez que a taxa é diferente e realizar a soma total. No fim de percorrer todas as encomendas, com auxílio do método *reduce* é calculado o total entre todas as encomendas e é originado o dinheiro total que a vintage ganhou.

9. Conclusão

A realização deste trabalho permitiu-nos aprofundar vários conceitos da programação orientada a objetos.

Foi também desenvolvida a técnica de encapsulamento, ou seja, não aceder diretamente aos dados mas usar métodos para o fazer.

Além disso, a aplicação de conceitos como herança permitiu-nos, por um lado, não ter que reescrever grande parte do código já que temos objetos a herdarem caracterısticas/métodos de outros.

Com este trabalho ficou claro quais os métodos de programação orientada a objetos a usar e a evitar.