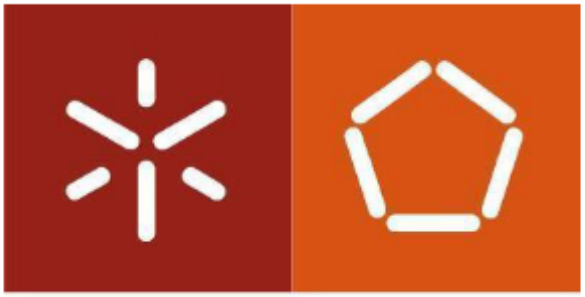


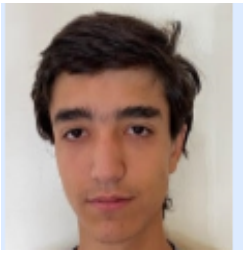
Universidade do Minho

Engenharia Informática - Programação Orientada aos Objetos

Relatório Trabalho Prático

Gonçalo Daniel Machado Costa

A100824



José Eduardo Silva Monteiro Santos Oliveira

A100547

Pedro Afonso Moreira Lopes

A100759

ÍNDICE

1.Introdução....2
2.Arquitetura de classes....2
2.1.Artigos....2
2.1.1.Sapatilhas....3
2.1.2.T-shirt....3
2.1.3.Malas....3
2.2.Utilizadores....4
2.3.Encomendas....4
2.4.Transportadoras....5
2.4.1.Fórmulas....5
2.5.Menus de interação....5
2.6.Estado....6
2.7.Controlador....6
2.8.Programa....6
2.9.Classes de Exceções....7
3.Estatísticas do estado do programa....7
3.1.Vendedor que mais faturou num período ou desde sempre....7
3.2.Transportador com maior volume de faturação....8
3.3.Listar as encomendas emitidas por um vendedor....8
3.5.Quanto dinheiro ganhou o Vintage no seu funcionamento....9
4.Diagrama de classes....9
5.Abordagens diferentes ao enunciado....10
6.Conclusão....10

1.Introdução

Foi proposta a realização de uma aplicação Java capaz de construir um sistema de marketplace Vintage que permite a compra e venda de artigos novos e usados de vários tipos, onde os utilizadores assumem o papel de vendedor ou comprador. Para além disso o sistema tem de ser capaz de gerir todas as compras e vendas assim como controlar o stock disponível dos artigos.

Durante o desenvolvimento do projeto, visamos sempre manter os princípios fundamentais do JAVA, nomeadamente:

- **Encapsulamento**, através da criação de métodos que permitem aceder e modificar atributos de qualquer classe;
- **Hierarquia**, através da utilização de superclasses que englobam métodos comuns a determinadas classes;
- **Polimorfismo**, que resulta da invocação dos mesmos métodos para subclasses distintas.

2.Arquitetura de classes

2.1.Artigos

A classe Artigo é a classe abstrata que complementa a estrutura dos outros artigos. Dado o Artigo ser uma classe abstrata, permite a adição de novos tipos de produtos sem ser necessário repetir o código para cada novo tipo de Artigo. As variáveis de instância presentes são:

- **Estado** estado - estado do artigo, definido pela Enum Estado, com valores novo e usado;
- **String** avaliacaoEstado - avaliação do estado, caso este seja usado;
- **int** numDonos - número de donos do artigo;
- **String** descricao - descricao do artigo;
- **String** marca - marca do artigo;
- **String** ID - ID do artigo;
- **double** precoBase - preço base do artigo;
- **double** correcaoPreco - preço depois de descontos aplicados;
- **String** idTransportadora - id da transportadora;
- **String** idVendedor - id do dono do artigo;
- **String** idExVendedor - id do ex-dono do artigo;

Esta classe, sendo abstrata, vai servir como extensão para as classes que definem cada um dos diferentes artigos disponíveis(Sapatilhas,T-shirts e Malas). Ela também possui um método abstrato para o cálculo do preço.

- public abstract **double** calculaPreco();

Esta classe também possui dois métodos extra, um para o cálculo do preço da expedição e o cálculo do lucro da transportadora.

- public **double** precoExpedicao(int opcao, double margemLucro, double imposto);
- public **double** lucroTransportadoraPorArtigo(int opcao,double margemLucro, double imposto);

2.1.1.Sapatilhas

Na classe Sapatilhas estão definidas as informações pedidas no enunciado que definem este Artigo. As variáveis de instância presentes são:

- **int** tamanho - tamanho das sapatilhas;
- **boolean** atacadores - se as sapatilhas possuem atacadores ou não;
- **String** cor - cor das sapatilhas;
- **LocalDate** dataDeLancamento - data em que as sapatilhas foram lançadas;
- **int** idade - idade das sapatilhas;

Também é possível que as sapatilhas sejam Sapatilhas Premium, que é uma subclasse que se estende às Sapatilhas. A variável de instância presente é uma String autor, com o nome do autor das Sapatilhas. Ambas as classes possuem o seu próprio método calculaPreco.

- calculoPreco Sapatilhas:
    - Se Tamanho > 45 e Estado Novo - Desconto: PreçoBase*(Tamanho/45*3);
    - Se Idade > 1 ano e Estado Usado - Desconto: PreçoBase*(numeroDonos+idade)/25;
- calculoPreco Sapatilhas Premium:
    - Aumento do Preço - PreçoBase + 5*Idade;

2.1.2.T-shirt

Na classe Sapatilhas estão definidas as informações pedidas no enunciado que definem este Artigo. As variáveis de instância presentes são:

- **Tamanho** tamanha - tamanho da T-shirt, definido por um enum com os campos S, M, L, XL;
- **Padrao** padrao - padrão da T-shirt, definido por um enum com os campos liso, riscas ou palmeiras;

A T-shirt tem o seu método calculaPreco.

- Se Padrão != liso e Estado == Usado - Desconto de 50%;

2.1.3.Malas

Na classe Mala estão definidas as informações pedidas no enunciado que definem este Artigo. As variáveis de instância presentes são:

- **int** anoColecao - ano da Coleção;
- **int** tamanho - tamanho da Mala;
- **int** estado_material - Estado do material;
- **int** idade - idade da Mala;

Também é possível que as malas sejam Malas Premium, que é uma subclasse que se estende às Malas. A variável de instância presente é uma int valorização, com o valor da sua valorização. Ambas as classes possuem o seu próprio método calculaPreco.

- calculaPreco Mala:
    - Desconto: 1/tamanho + 1/estado_material + 1/idade;
- calculaPreco Mala Premium:
    - Aumento de Preço - Preço*(valorização/100)*idade;

2.2.Utilizadores

Na classe Utilizadores estão definidas as informações pedidas no enunciado para cada utilizador, que pode servir como cliente e vendedor, e ambas não estão separadas. As variáveis de instância presentes são:

- **String** codigo - codigo do utilizador;
- **String** email - email do utilizador;
- **String** nome - nome do utilizador;
- **String** morada - morada do utilizador;
- **int** numeroFiscal - NIF do utilizador;
- **Map<String,Artigo>** produtosVendidos - coleção de produtos vendidos pelo utilizador
- **Map<String,Artigo>** produtosParaVenda - coleção de produtos que pode vender
- **Map<String,Artigo>** produtosAdquiridos - coleção de produtos comprados
- **Map<String,Artigo>** carrinho - coleção com os produtos do carrinho
- **Map<String,Encomenda>** comprasFeitas - coleção de encomendas feitas
- **double** valorVendas - valor conseguido por vendas;

Esta classe possui dois métodos, um para adicionar artigos para venda e outro para uma encomenda. Caso a encomenda não exista, manda-se uma IdEncomendaNaoExistenteException.

- public **void** adicionaArtigoParaVenda(Artigo artigo);
- public **String** verificaEncomenda(String ID) throws IdEncomendaNaoExistenteException;

2.3.Encomendas

Na classe Encomendas estão definidas as informações pedidas no enunciado para cada encomenda. As variáveis de instância presentes são:

- **String** ID - id da encomenda;
- **Map<String,Artigo>** lista_artigos- coleção de artigos da encomenda;
- **Dimensao** dimensao; //dimensao da encomenda;
- **double** preco_final - preço final da encomenda;
- **double** custos_de_expedicao - custo de expedição da encomenda, dependendo de cada transportadora;
- **Estado** estado - estado da encomenda (começa como pendente, passa a expedida depois de ser feita e passa a finalizada depois de 2 dias da criação da encomenda;
- **LocalDate** data_de_criacao - data da criação da encomenda;

Também possui vários métodos para adicionar e remover artigos e calcular o valor de cada encomenda:

- public **void** adicionaArtigo(Artigo artigo);
- public **void** removeArtigos();
- public **void** removeArtigo(String id);
- public **double** valorFinalEncomenda(Map<String,Transportadora> mapa);
- public **double** precoExpedicaoEncomenda(Map<String,Transportadora> mapa);
- public **double** valorVintage();
- public **Dimensao** dimensao();
- public **void** faturaEncomenda(Map<String,Transportadora> mapa);
- public **int** idadeEncomenda(LocalDate data);
- public **void** encomendaFinalizada(LocalDate data);

2.4.Transportadoras

Na classe Transportadoras estão definidas as informações pedidas no enunciado para cada transportadora disponível no mercado. As variáveis de instância presentes são:

- **String** nome - nome da transportadora;
- **int** formula - fórmula utilizada para cálculo do lucro;
- **double** valorPequeno;
- **double** valorMedio ;
- **double** valorGrande;
- **double** imposto - imposto aplicado;
- **double** margemLucro - margem de lucro da transportadora;
- **double** totalLucro - lucro total da transportadora;

Não existem métodos de realce na classe Transportadoras.

2.4.1.Fórmulas

A classe Fórmulas é a classe utilizada no cálculo do preço de expedição das transportadoras. Esta classe não possui variáveis de instância, porém possui 3 métodos diferentes para o cálculo do preço de expedição.

- Fórmula 1 - precoBase * margemLucro * (1 + imposto) * 0.9;
- Fórmula 2 - precoBase * (margemLucro + imposto);
- Fórmula 3 - (precoBase/imposto) * margemLucro;

2.5.Menus de interação

Aqui, decidimos apenas utilizar e adaptar um pouco a classe Menu que nos foi fornecida pela UC.

2.6.Estado

A classe Estado é a classe utilizada para a verificação do estado do programa em um dia específico. É usada também sempre que é necessário fazer operações de avanço de tempo.

- **Set<String>** utilizadores - coleção de utilizadores num dia;
- **Set<String>** encomendas - coleção de encomendas num dia;
- **LocalDate** diaAtual - dia do estado;

2.7.Controlador

A classe Controlador é uma das 2 classes mais fulcrais de todas neste projeto, pois esta classe vai servir como a base de dados do programa. Em outras palavras, para futuro uso no programa, tudo o que for criado vai ser armazenado aqui, facilitando assim o acesso a objetos que sejam necessários para cálculos e métodos.

- **Map<String,Utilizador>** utilizadores - coleção de todos os utilizadores criados;
- **Map<String,Artigo>** artigos - coleção de todos os artigos;
- **Map<String,Transportadora>** transportadoras - coleção de todas as transportadoras;
- **Map<String,Encomenda>** encomendas - coleção de todas as encomendas criadas;
- **double** valorVintage - valor ganho pela Vintage;
- **Map<LocalDate,Estado>** estados - coleção de todos os estados aka dias em que houve atividade no site;
- **LocalDate** diaAtual - dia atual, que vai ser usado na criação dos estados;

É nesta classe também onde estão definidos alguns dos métodos que realizam alguns dos requisitos definidos no enunciado.

2.8.Programa

A classe Programa é onde tudo o que foi feito vai ser aplicado, sendo assim possível a execução e realização do programa em si.

- **Controlador** controller - controlador do programa aka o que vai conter tudo (tipo uma base de dados);
- **Utilizador** user - user que está logado no programa;
- **Map<String,Artigo>** artigosParaCompra - artigos que o user que está logado pode comprar;
- **final Scanner** scan;

Estão aqui também definidos métodos que nos vão permitir a introdução de input do utilizador, usando para isto os menus previamente mencionados, o controlador que vai ser usado para armazenar tudo e o utilizador que será logado e utilizará o programa.

2.9.Classes de Exceções

As seguintes classes implementadas descrevem o comportamento do programa quando este se depara com um erro nelas descrito:

- **DevolucaoImpossivelException -** comportamento do programa caso a devolução de uma encomenda seja impossível devido a ultrapassar o tempo limite de devolução.
- **EmailExistenteException -** comportamento do programa caso exista uma tentativa de inserir um email já existente.
- **EmailNaoExistenteException -** comportamento do programa caso exista uma tentativa de aceder a um email que não existe.
- **IIdArtigoExistenteException -** comportamento do programa caso exista uma tentativa de inserir um identificador de artigo já existente.
- **IdArtigoNaoExistenteException -** comportamento do programa caso exista uma tentativa de aceder a um identificador de artigo que não existe.
- **IdEncomendaNaoExistenteException -** comportamento do programa caso exista uma tentativa de aceder a um identificador de encomenda que não existe.
- **TransportadoraExistenteException -** comportamento do programa caso exista uma tentativa de inserir uma transportadora já existente.
- **TransportadoraNaoExistenteException -** comportamento do programa caso exista uma tentativa de aceder a uma transportadora que não existe.

3.Estatísticas do estado do programa

3.1.Vendedor que mais faturou num período ou desde sempre

```
   public Utilizador vendedorMaisFaturouSempre (){

         return this.utilizadores.entrySet()

              .stream()

              .max((entry1, entry2) -> entry1.getValue().getValorVendas()
> entry2.getValue().getValorVendas() ? 1 : -1)

              .get()

              .getValue();

    }
```

Aqui vamos obter o maior valor que um vendedor teve desde a criação do programa. Será feita a organização de todos os utilizadores a partir do seu valor de venda, obtendo-se no final o maior valor, utilizando o max.

3.2.Transportador com maior volume de faturação

```
   public Transportadora transportadoraComMaisFaturacao (){

        return this.transportadoras.entrySet()

            .stream()

            .max((entry1, entry2) -> entry1.getValue().getTotalLucro()
> entry2.getValue().getTotalLucro() ? 1 : -1)

            .get()

            .getValue();

    }
```

Muito semelhante à forma como calculamos o ponto anterior em termos de lógica, aqui vamos obter a transportadora com o maior volume de faturação.

3.3.Listar as encomendas emitidas por um vendedor

```
   public void listarEncomendasDeVendedor(String utilizador) {

        for(Map.Entry<String,Encomenda> encomenda :
this.encomendas.entrySet()){

            for(Map.Entry<String,Artigo> artigo :
encomenda.getValue().getListaArtigos().entrySet()){


if(artigo.getValue().getIdVendedor().equals(utilizador)){

                    System.out.println("Encomenda " +
encomenda.getKey() + " com os artigos " +
encomenda.getValue().getListaArtigos());

                    break;

                }

            }

        }

    }
```

Aqui simplesmente filtramos de todos os artigos de todas as encomendas aqueles que pertencem ao vendedor cujo identificador é enviado para o método.

3.5.Quanto dinheiro ganhou o Vintage no seu funcionamento

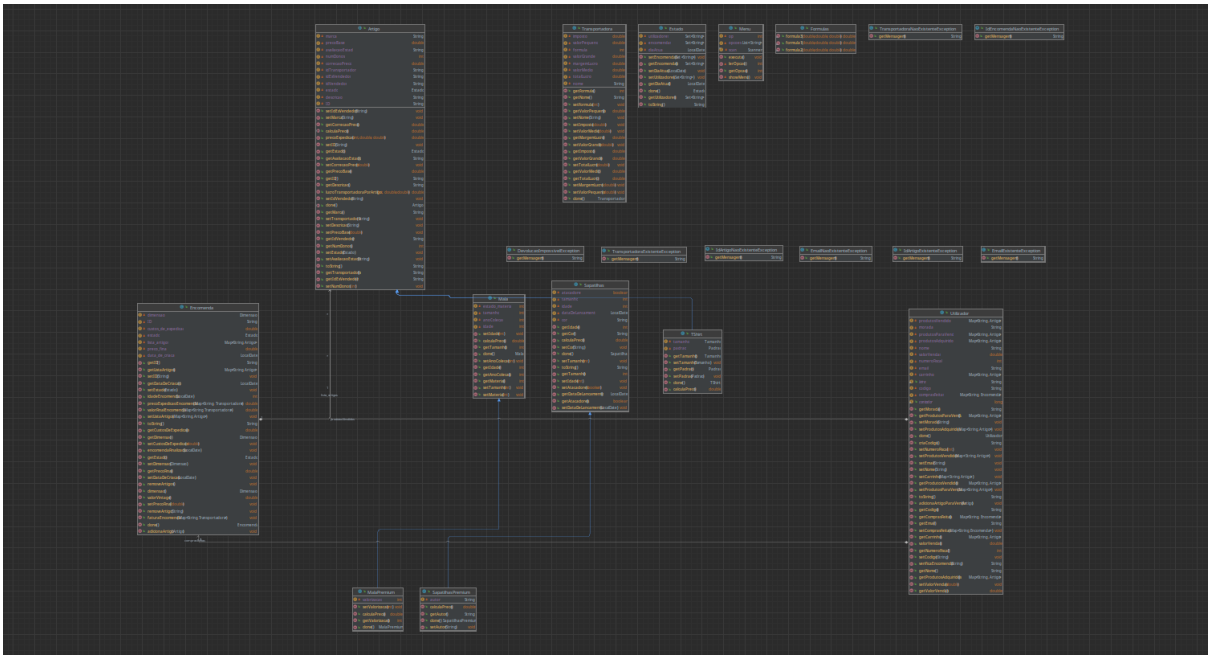
```
public double valorVintageTotalFuncionamento(){

      return valorVintage;

 }
```

Neste método, como armazenamos o valor total da Vintage no controlador sempre que é feito uma compra, decidimos então apenas retornar este valor do controlador.

4.Diagrama de classes



5.Abordagens diferentes ao enunciado

De seguida, iremos explicar as mudanças feitas ao longo do trabalho em relação ao enunciado. Uma das mudanças que fizemos foi não usarmos o tamanho da encomenda para o cálculo do preço de expedição da encomenda, pois as transportadoras transportam artigos e não encomendas, daí o uso do tamanho da encomenda ser inútil, a nosso ver.
Outra mudança que fizemos foi a declaração do atributo material nas Malas como um valor inteiro, pois não nos sentimos confortáveis no cálculo dos preços das malas com uma String na fórmula.

6.Conclusão

Em suma, acreditamos que fomos capazes de realizar um sólido projeto, mas sem dúvida deparamos-nos com diversas dificuldades o que impediu de concluir e resolver tudo o que nos foi proposto, como por exemplo a implementação de transportadoras premium e algumas complicações com o uso do utilizador logado e o mesmo utilizador no controlador. Gerimos de forma eficiente as relações entre as classes e implementamos métodos que facilitam o funcionamento geral do programa. Apesar disso, acreditamos que atingimos bons resultados pois, tal como mencionamos em cima, o projeto foi bem conseguido dentro dos possíveis.