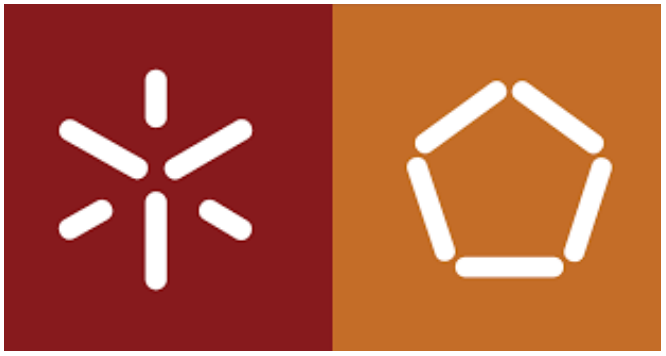


Universidade Do Minho
Engenharia Informática

Trabalho Prático
POO

2022/2023

Relatório Grupo83 data de entrega: **14/05/2023**

Composição do Grupo 83

Tiago Pinheiro Silva
a93285

Tiago Alexandre Ferreira Silva
a93182

Carlos Alberto Fernandes Dias Da Silva
a93199

Índice

Introdução

Introdução

Neste trabalho prático da Unidade Curricular de Programação Orientada aos Objetos, foi nos proposto a criação de um programa que simula um "Market Place" usando a linguagem de programação Java.
Foi nos requerido para esta aplicação criar as funcionalidades necessárias para: A compra e venda de artigos, podendo estes serem novos e/ou usados, a possibilidade de controlar o stock de certos artigos dos utilizadores, decidir a transportadora a quando da venda de um dos seus artigos, entre outras.
A cada artigo ser-lhe-ão atribuídas características que os tornam únicos, como, no caso das malas, a sua dimensão e o material de que são estas mesmas feitas.

Classes

Artigo.java
esta é a classe que ditará os atributos gerais e métodos fundamentais a criação de todas as suas subclasses
Esta classe apresenta-se no topo da Hierarquia.

Malas.java
subclasse do artigo, que especializa o comportamento deste tipo de artigo:
- dimensão
- material
- ano de coleção

Sapatilhas.java
subclasse do artigo, que especializa o comportamento deste tipo de artigo:
- tamanho
- atacadores
- cor
- ano de coleção

T_shirt.java
subclasse do artigo, que especializa o comportamento deste tipo de artigo:
- tamanho
- padrão

Encomenda.java
embora não seja subclasse de artigo, nesta temos listas destes mesmos, consoante a encomenda de um utilizador.
aqui foi escolhido usar uma map (Map<Integer ,Artigo> artigos) para guardarmos os produtos que se encontram a esta associada, uma vez que permite uma fácil consulta e alteração da mesma através do id dos artigos, coisa que é feita com frequência quando estamos a realizar uma encomenda.
como cada artigo tem uma transportadora associada, por este motivo será possível que numa encomenda exista mais que uma transportadora, em que cada uma pode praticar um preço diferente, seja por ter margens de lucro diferentes, seja por ter dimensões de expedição

diferentes, de forma a garantir o cálculo correto do preço de expedição usamos um map (Map<Integer,Integer> dimensoes) ao qual esta associado os ids das transportadoras dos produtos que se encontram na encomenda e o número de artigos que possuem dessa mesma encomenda. Assim calculando de forma correta o preço de expedição das encomendas.

Transportadora.java

embora não seja subclasse de artigo, um artigo é associado a uma transportadora aquando da colocação no sistema

NewMenu.java

classe fornecida pelos professores de menus

Utilizador.java

classe onde se estipula o que é um utilizador, as suas características e funções no sistema (como de comprador e de vendedor)
para guardar os artigos vendidos de cada utilizador usamos um map (Map<Integer, Integer> artigos_vendidos;) em que guardamos o id do artigo na primeira parte do map e o id da encomenda na qual este foi vendido de forma, sendo assim possível ter acesso fácil e rápido às faturas da encomenda em que se encontra o artigo.
para criar as faturas usamos um map (Map<Encomenda, Double> faturas) em que guardamos toda a informação da encomenda na primeira parte, sendo assim possível ter acesso a toda a informação presente nesta. Na segunda parte encontra-se o valor final gasto pelo comprador somando o valor dos produtos, o valor das taxas da vintage e portes de envio de todas as transportadoras associadas.

TextUI.java

segundo o modelo MVC*, esta classe identifica-se como uma view e controller juntos.

Vintage.java

classe responsável pelo manuseamento dos dados da model, segundo o modelo MVC*

VintageAPP.java

classe responsável por executar o programa. (main)

*Modelo MVC

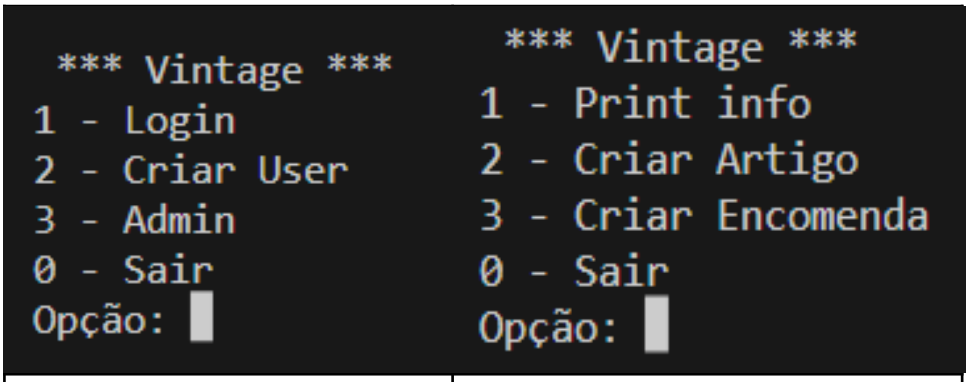
Model-View-Controller é um modelo utilizado com o intuito de separar a lógica de apresentação da interface. Separar o programa em várias secções onde cada um tem as suas próprias funções dentro do projeto ajudará no desenvolvimento do mesmo

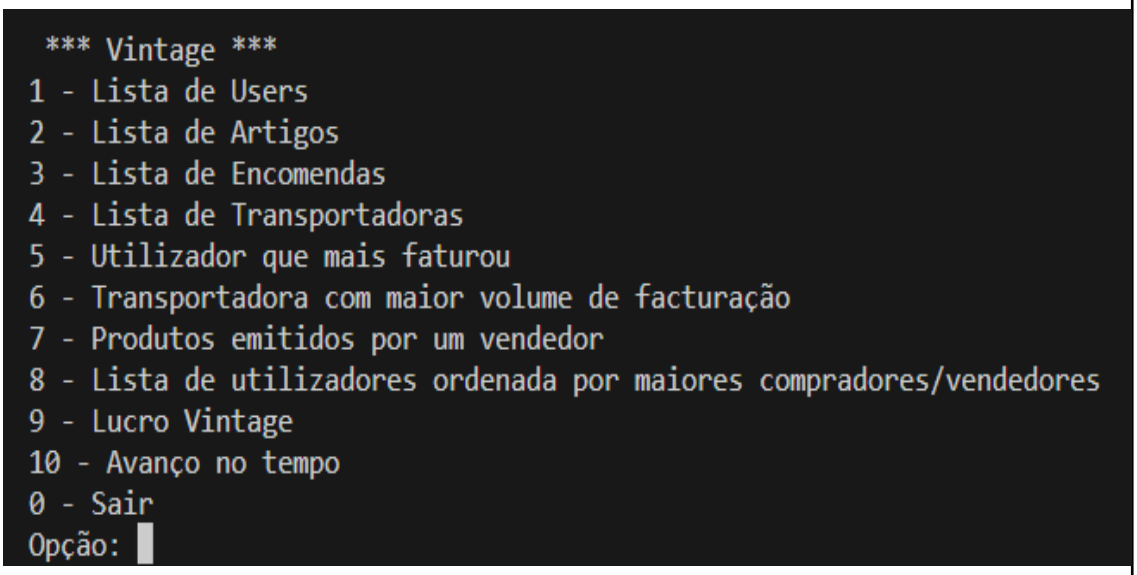
a nossa implementação:

Decidimos abordar este projecto tendo em conta a arquitetura do MVC, mas juntando a parte da view com o controller, fazendo parte do Model tudo o resto.

Sendo assim, temos a TextUI.java, onde se agrupam os métodos responsáveis por conseguir passar os dados necessários a visualização de uma interface por parte de um Utilizador (View) e os métodos responsáveis por intermediar entre esta interface e o resto do model, que administrará o input dado (Controller).

O resto das classes pertencem então ao Model, onde os dados correspondentes ao MarketPlace serão lidos, armazenados e tratados.

	
Menu inicial	Menu pós-login



Menu Admin

Conclusão

Este projeto foi fundamental para consolidar os conceitos aprendidos em sala de aula e compreender a importância das boas práticas na programação para, por exemplo, um projeto poder ser expandido infinitamente.
Fizemos por aplicar os principais pilares de POO, como o encapsulamento, abstração, herança e polimorfismo, que foram ensinados ao longo do semestre. Concluímos então que seguir esses princípios permite-nos ter um código mais organizado e menos redundante, tornando mais eficaz.

Apesar das adversidades, acabamos o projeto seguros que nos tornamos melhores e mais eficientes nesta área da programação.