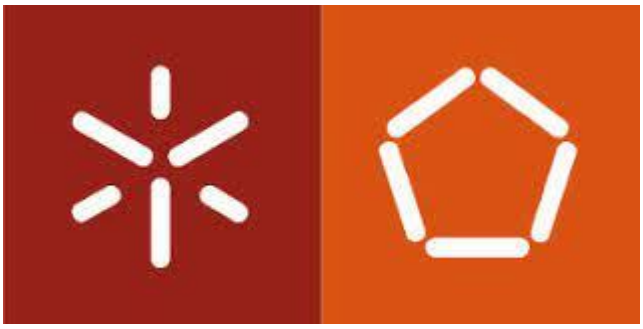


UNIVERSIDADE DO MINHO

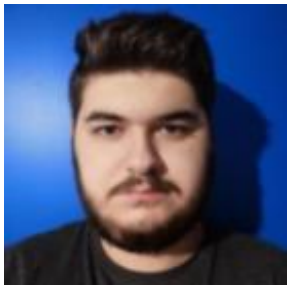
LICENCIATURA EM ENGENHARIA INFORMÁTICA

ANO LETIVO 2022/2023

POO – TRABALHO PRÁTICO

GRUPO 38

António Silva(A100533), José Matos(A100612), Pedro Silva(A100745)



Índice

Capítulo 1

Introdução e principais objetivos

O projeto proposto consiste em criar uma loja chamada Vintage onde os utilizadores podem colocar à venda ou comprar artigos em forma de encomendas, sendo esses artigos enviados por diferentes transportadoras.

Inicialmente, esta loja só terá à venda sapatilhas, t-shirts e malas, mas futuramente poderá ter uma maior variedade de artigos. Alguns desses artigos podem ser de qualidade premium, fazendo com que o seu valor aumente ao longo dos anos, ao contrário de artigos normais que o seu preço não altera, a não ser por motivos específicos, por exemplo, o seu estado.

Tal como uma loja verdadeira, a Vintage tem de ter uma noção de tempo, ou seja, todas as encomendas têm de estar a ser processadas em tempo real ou em saltos de tempo, ou seja, caso avancemos uma semana, tudo o que seria processado nessa semana, o seu estado tem de ser alterado, isto é, encomendas serem entregues, pagamentos serem processados, rejeição de reembolsos.

Também nos foi pedido que guardássemos várias estatísticas da loja, tal como qual é a transportadora mais cara de se utilizador, o utilizador com mais vendas, o que fez mais compras ou quando dinheiro já foi processado dentro da empresa.

Além disso, a qualquer momento podem ser criados novos utilizadores, estes podem por novos artigos à venda, novas encomendas podem ser feitas e transportadoras novas podem começar a existir com novos preços. Como é possível criar novos utilizadores, também é possível removê-los, fazendo com que os seus artigos também saiam do mercado, encomendas podem ser canceladas desde que ainda estejam dentro dos 2 dias de reembolso, transportadoras podem ir à falência e parar de existir, por exemplo, tentamos criar uma loja o mais verídica possível.

Um dos principais objetivos era que no final do trabalho pudéssemos automatizar a loja o mais eficientemente possível, tornando-a de algum modo “autónoma”. Sendo assim, se for adicionado um ficheiro de AutoRun, a loja efetua tudo o que estiver descrito no ficheiro automaticamente, desde a criação de um novo artigo até uma viagem no tempo, que altera tudo o que acontece no programa.

Nós decidimos fazer o trabalho de um modo um pouco mais administrativo, tendo assim um maior controlo sobre tudo o que acontece na empresa, em vez de levarmos uma vertente mais da parte do utilizador, por exemplo, dar login com um utilizador e ver a loja só da perspetiva dele, nós podemos fazer tudo o que quisermos, até eliminar a loja caso seja algo necessário.

Capítulo 2
Classes

2.1. ARTIGO

Quando nos foi proposto o projeto, foi recomendado que fosse possível uma futura implementação de outros tipos de artigos e nós para facilitarmos essa implementação criamos a classe abstrata Artigo que teria todas as características comuns a todos os artigos.

Inicialmente temos três constantes inteiras(MALA, SAPATILHAS e TSHIRT) para identificar os tipos de artigos que temos atualmente. As outras características comuns são:

- private int tipo
- private float estadoUtilizacao
- private int numDonos
- private String descricao
- private String marca
- private int codigo
- private float precoBase
- private int codigoVendedor
- private Transportadora transportadora
- private boolean premiumEstado

Futuramente iremos falar do que são as Transportadoras e os artigos que não são supostos ter premium irão sempre devolver false não interferindo de qualquer forma nos seus cálculos e estatísticas.

Criamos um construtor de artigo de forma que as subclasses que sejam extensões do artigo poupem trabalho na sua criação, informamos que todas têm de ter uma função de calcularPreco e calcularCorrecao dependo da transportadora, por exemplo. Tem a função de calcularCorrecaoPremium já implementada, porque esta será igual para todos que utilizem premium.

A forma que arranjamos de calcular a evolução do preço de um artigo premium foi:

```
if (this.getPremiumEstado() == true) {
    precoPremium += this.getPrecoBase() * ((LocalDateTime.now().getYear() - anoColecao) / 100f);
}
```

Dá para compreender pela função que quanto mais antigo for o Artigo, maior será o preço do Artigo premium.

Finalmente, tem os getters e setters de todos os atributos.

2.2. SAPATILHAS

O primeiro artigo que implementamos foram as sapatilhas. Como mencionamos anteriormente, todos os artigos vão ser uma extensão da classe Artigo, sendo que cada subclasse tem as suas características específicas. No caso das sapatilhas, seria ter atacadores ou atilhos, o tamanho delas, a cor e o ano da coleção.

Estes atributos estão guardados do seguinte modo:

- public static final int ATACADORES = 0;
- public static final int ATILHOS = 1;
- private int tamanho;
- private int atacadores;
- private String cor;
- private int anoColecao;

As contantes ATACADORES e ATILHOS servem para facilitar a compreensão do tipo de atacadores.

Além disso têm o seu próprio construtor, utilizando a funcionalidade super com os atributos da classe Artigo que são usados nas sapatilhas, a sua própria função de calcularPreco que utiliza o precoBase, a calcularCorrecaoPremium, pois sapatilhas podem ser premium ou não e a sua própria calcularCorrecao.

No caso das sapatilhas, a calcularCorrecao, além de se adicionar o preço da Transportadora, caso não sejam novas, ou seja, estado de Utilização diferente de 1, sendo que estes podem ser 0.25,0.5,0.75 ou 1, vão sofrer uma redução de preço quão pior for o seu estado(Pouco Uso 25%, Muito Uso 50% e Estragado 75%) e caso sejam novas e de tamanho maior de 45 recebem ainda mais uma dedução de 25% também.

No nosso caso, o estado nunca terá nada a ver com o número de donos pois achamos que isso é uma correlação pouco baseada, porque dependendo do tipo de dono, o artigo pode não diminuir no seu estado de utilização, por exemplo, algo pode ter 3 donos e continuar praticamente novo e também ter 3 donos e ficar completamente estragado.

Por fim, tem os getters e os setters e na sua toString utiliza funções de utils que serão todas explicadas no seu devido tempo. Todos os tipos de artigo terão sempre no final do ficheiro estas funções, sendo a principal a toString que utiliza as funções dos ficheiros utils.

2.3. TSHIRT

Sendo a TShirt outra extensão da classe Artigo, também só terá de implementar as suas características específicas. No caso da TShirt são o tamanho e o seu padrão.

Para facilitar a identificação de cada usamos as seguintes constantes:

```
// Tipos de padrão
1 usage
public static final int LISO = 0;
no usages
public static final int RISCOS = 1;
no usages
public static final int PALMEIRAS = 2;

// Tamanhos
no usages
public static final String S = "S";
no usages
public static final String M = "M";
no usages
public static final String L = "L";
no usages
public static final String XL = "XL";
```

```
private int padrao;
```

```
private String tamanho;
```

Depois, tal como nas sapatilhas, criamos o construtor utilizando novamente o mesmo método super.

O calcularPreco é implementado, mas ao contrário das Sapatilhas, as TShirts nunca serão premium, logo não utilizará a calcularCorrecaoPremium, e, por fim, tal como as Sapatilhas, tem a sua implementação da calcularCorrecao.

O calcularCorrecao, no caso das TShirts, funciona mais focado no seu padrão. Além de adicionar o valor da transportadora, caso o padrão seja Liso não há mais nenhum desconto, caso contrário, se a TShirt não for nova, independente do padrão, exceto o Liso, teria um desconto fixo de 50%.

2.4. MALA

O último tipo de artigo que implementamos foi a Mala, este tipo de artigo caracteriza-se principalmente pelo facto de o seu preço acabar por depender além das suas características específicas também muito pela sua dimensão.

Este artigo tem várias constantes, sendo as primeiras utilizadas para o tipo de material(TECIDO, PELE, LONA ou VELUDO), as segundas para facilitar a encontrar as dimensões no array delas(COMPRIMENTO, LARGURA e ALTURA) e por fim duas constantes para o calculo do preço (CONSTANTE_CORRECAO e MARGEM_ERRO).

Além disso as suas características específicas são:

- private float[] dimensao; [COMPRIMENTO, LARGURA, ALTURA]
- private int material;
- private int anoColecao

Tal como todas as outras o construtor continua a funcionar da mesma forma, tal como a implementação da calcularPreco, que nas malas também utiliza calcularCorrecaoPremium, porque uma mala pode ser premium ou não.

O calcularCorrecao da mala funciona pegando nas suas dimensões, calculá-la em volume utilizando a função calcularDimensao, adiciona o preço da Transportadora e depois com esta formula:

```
correcao -= (1f / dimensao) * this.getPrecoBase() * CONSTANTE_CORRECAO;
```

A constante de correção foi adicionada, para o caso de as dimensões forem inválidas, por exemplo, dava um preço negativo, o programa possa avisar que essa mala não pode ser adicionada ao programa.

Por fim, tem todos os getters e setters como todos os artigos.

2.5. UTILS(ARTIGOS)

Quando havia necessidade de utilizar funções que seriam partilhadas por várias classes, fomos criando os Utils. Os utils são pequenas classes com algumas funções que servem para corrigir cálculos, facilitá-los ou ajudar na designação de certos parâmetros.

No caso do Utils dos Artigos, este só tem três funções, sendo as duas primeiras para facilitar a demonstração no toString (arredondarDecimas e

arredondarCentesimas), ou seja, apesar de o programa calcular com todas as casas decimais que lhe estão disponíveis, dificultando erros de cálculo, quando demonstrarmos no ecrã, só iremos mostrar as casas decimais principais que desejamos, por exemplo, no preço apresentado, mesmo que este dê 35,1253264, só iremos mostrar 35,13.

```
"Preço Final: " + Utils.arrondarCentesimas(this.calcularPreco()) + "\n" +
```

A terceira função é a calcularPercentagem que trata do valor que é preciso acrescentar ao artigo por causa do valor de expedição que a transportadora cobra pelo transporte.

2.6. UTILIZADOR

Depois de termos implementados os três tipos de artigo pedidos, iniciamos a criação da classe Utilizador. Os utilizadores serão as pessoas que são ambos compradores e vendedores que podem comprar e por à venda os seus artigos. Pode-se criar utilizadores a qualquer momento e apagar, consequentemente perdendo os seus artigos listados, a não ser que estejam numa encomenda. Quando se apaga um utilizador, simplesmente colocamos a conta INATIVA, para não perder informação das encomendas previamente feitas, mas deixamos de listá-lo nos utilizadores na UI.

Por este motivo, inicialmente temos as duas constantes ATIVA e INATIVA e depois os atributos de um utilizador:

```
private int codigo;
4 usages
private String email;
4 usages
private String nome;
4 usages
private String morada;
4 usages
private int numeroFiscal;
7 usages
private int atividade;

8 usages
private List<Integer> listados;
5 usages
private List<Artigo> vendidos;
5 usages
private List<Artigo> comprados;
5 usages
private float valorEmVendas;
3 usages
private float valorEmCompras;
```

- O código de cada utilizador atribuído pelo sistema, o seu email nome, morada, número fiscal e a sua atividade.
- As três listas com os seus artigos listados, os que já foram vendidos e os que ele comprou
- Por fim guardamos o valor que ele ganhou em vendas e quanto gastou em compras.

No construtor, iniciamos o utilizador com três ArrayList para os listados, vendidos e comprados e com os dois valores a zero.

Temos 4 funções que tratam da listagem, da compra e da venda de artigos:

- A função criarListagem recebe um artigo e adiciona-o aos listados do utilizador, caso este for ativo
- A função removerListagem que remove o artigo que é pedido dos listados de um utilizador
- A função venderArtigo que remove um artigo dos listados, acrescenta-o aos vendidos e aumenta o valorEmVendas do valor do artigo, excluindo a parte da transportadora
- A função comprarArtigo recebe a lista de utilizadores e o artigo que vai ser comprado, adiciona o lucro à transportadora usada, descobre qual é o utilizador vendedor, vende-lhe o artigo e depois adiciona este artigo aos comprados do utilizador comprador e aumenta o seu valorEmCompras

No final tem todos os getters e setters e a toString do utilizador também utiliza Utils, sendo que algumas já foram explicadas e outras ainda não.

2.7. ENCOMENDA

Depois de criados os artigos e os utilizadores, finalmente podemos criar a forma como os artigos vão ser vendidos aos utilizadores, em forma de encomendas.

Todas as encomendas têm três estados: pendentes, expedidas ou finalizadas, algo que definidas com constantes.

Depois de criadas, todas as encomendas têm dois dias para serem reembolsadas, ou será impossível retirar o artigo da encomenda, mesmo que ainda seja pendente.

As encomendas terão vários atributos:

```
private int codigo;
6 usages
private int codigoComprador;
11 usages
private List<Integer> artigos;
6 usages
private int dimensaoEncomenda;
7 usages
private int estadoEncomenda;

7 usages
private float precoEncomenda;
5 usages
private LocalDateTime dataCriacao;
5 usages
private LocalDateTime dataEntrega;
```

- Um código que as identifica, um código que identifica o utilizador que a faz e uma lista com o código dos artigos que vão ser comprados
- Vai ter três tipos de dimensão: Grande com 10 artigos, Média com 5 artigos e Pequena com 1 artigo, podendo sempre ter menos
- Além disso, tem o estado, o seu preço, quando foi criada e até quando é a sua entrega, que depende da sua dimensão

No seu construtor, quando a encomenda é criada, o código dela é atribuído pelo sistema, inicia o ArrayList dos artigos, o seu estado será PENDENTE, o seu preço é 0 e a dataCriacao será a data dada pelo sistema.

As funções na classe Encomenda já utilizam o ErrorCode que explicaremos mais à frente.

Na função adicionarArtigo, recebemos a lista de artigos, de encomendas e o código do artigo que queremos adicionar. Testando se a encomenda já foi expedida, se já não está cheia, se o código daquele artigo existe, se não é um artigo do próprio comprador, se este artigo já não está noutra encomenda e só por fim é que sabemos que não nenhum problema e o adicionamos à encomenda. Recalculamos o valor da encomenda e dizemos que não existe qualquer erro.

A segunda função removeArtigo, recebe a lista de artigos e o código do artigo que pedem para ser removido. Se a encomenda já não estiver pendente, já não podemos remover o artigo, se tiver vazia não há nada para remover, se o artigo não existir também não é possível e no fim verifica se o artigo existente está na encomenda. Caso esteja, é removido da encomenda e recalculasse o preço.

Por fim, apesar dos getters e setters serem como o habitual, na toString das encomendas tivemos de meter um if no caso de a encomenda ainda não ter sido expedida, pois nesse caso ainda não temos data de entrega, logo só aparece “Por expedir”, enquanto no outro caso já aparece a data em concreto.

2.8. UTILS(ENCOMENDAS)

No utils das encomendas, simplesmente tem a função que determina o tempo de expedição das encomendas. No caso que ela seja grande, ou seja, tamanho 10, serão 14 dias, tamanho médio(5), 7 dias e uma encomenda pequena demora somente 3 dias a ser entregue.

2.9. TRANSPORTADORA

Por fim, apesar de já termos falado delas, só implementamos as transportadoras no fim e depois é que as adicionamos aos artigos.

Todos os artigos têm a sua transportadora específica, tendo esta de ser especificada no momento que o artigo é criado, uma encomenda pode ter artigos de transportadoras diferentes e, caso uma transportadora seja apagada, todos os artigos que a têm como a sua transportadora, não são apagados, pois a transportadora encontra-se obrigada a terminar o seu trabalho, neste caso, até estes artigos serem vendidos ela continua a "existir".

As transportadoras também poderão ser premium e artigos premium terão de ser entregues obrigatoriamente por estas transportadoras. Para as diferenciar, estas transportadoras também terão uma margem extra além da margem de lucro habitual.

Os atributos são então:

```
private String nome;
6 usages
private float margemLucro;
5 usages
private float margemExtra;
5 usages
private float valorExpedicao;
5 usages
private float lucro;
4 usages
private boolean premiumEstado;
```

- Todas as transportadoras têm o seu nome e a sua margem de lucro, mas só as premium têm a margem extra
- O valor de expedição é depois calculado com umas constantes que estão nas utils e as margens
- Sempre que um artigo é vendido guarda-se o lucro que a transportadora teve
- Por fim, guarda-se se uma transportadora é premium ou não

No construtor, além de se guardar os parâmetros todos calcula-se o valor de expedição.

A função calcularValorExpedicao utiliza as tais constantes e as margens sendo a fórmula deste modo:

```
public float calcularValorExpedicao() {
    return this.premiumEstado ? (VALORBASE * margemLucro * (1 + IMPOSTO)) * margemExtra
            : VALORBASE * margemLucro * (1 + IMPOSTO);
}
```

Tem também a função calcularEntrega que atualiza o lucro da transportadora.

A última função que falta falar das transportadoras é a atualizarValores que autoriza uma transportadora de alterar o valor das suas margens se achar necessário. Não dá para ver na função em concreto, mas nós só autorizamos as transportadoras premium a alterar os seus valores, pois achamos que não faz sentido qualquer transportadora mudar o seu valor de um momento para o outro.

No final, igualmente os getters e setters e a toString utilizando novamente outras funções das Utils, sendo primeiramente introduzidos à que trata do premium.

2.10. UTILS(TRANSPORTADORA)

No caso das utils da transportadora, esta simplesmente guarda as duas constantes que são utilizadas no cálculo do valor de expedição, o VALORBASE e o IMPOSTO.

2.11. SAVELOAD

Foi a partir daqui que começamos a pensar na formulação da nossa UI e como iriamos demonstrar o que acontece dentro do programa, mas antes de começarmos a fazer isso tivemos de pensar como iriamos guardar o estado do programa e quando o voltássemos a abrir dar load a esse estado guardado, de forma que funcione como se fosse um website duma empresa verdadeira.

Para conseguirmos fazer o Save e o Load do programa optamos por utilizar a biblioteca Gson que guarda num ficheiro json o estado do programa e consegue posteriormente lê-lo de forma a obter toda a informação que estava guardada.

Todo esse processo encontra-se feito na classe SaveLoad, sendo a primeira função uma preparação para o load, a segunda a função de save e a terceira a função de load em concreto.

As duas strings constantes iniciais servem para representar a pasta para onde o save deve ir no final do programa e a segunda é onde o programa vai buscar a save (se ele existir) e o nome que ele deve ter.

A função prepareGsonLoader serve para preparar o load do nosso estado guardado, pois apesar de a função de leitura para o load ser muito bem implementada, no caso da utilização de uma classe com extensões, ela precisa que lhe seja especificado os tipos que podem acontecer, ou seja, no nosso caso, por causa do artigo, temos de avisar dos diferentes tipos de

artigos e como serão guardados, de resto ela consegue fazer tudo automaticamente lendo cada classe de forma muito eficaz.

A função que servirá para guardar o estado do programa, a função save, primeiro vê o local onde vai ter de ser guardada essa informação, se essa pasta não existir tem de criá-la e, só depois, é que guarda a informação organizada no ficheiro json com o nome dado no outro filepath utilizando a função toJson, guardando toda a informação da loja dada à função. Caso haja algum erro mostra qual é na consola.

A função load que vai criar a loja Vintage ou buscar o estado que está guardado, primeiro procura o ficheiro e caso este não exista ou se tiver algum erro nele(escrita mal feita, por exemplo, se o ficheiro fosse feito por nós e não pela máquina) devolve uma loja completamente vazia, se existir mas estiver vazio faz a mesma coisa, no último caso, aquele em que há um estado corretamente guardado, utilizando a função fromJson transforma o ficheiro json numa loja Vintage com o estado que se encontrava guardado até ao momento.

2.12. LocalDateTimeTypeAdapter

Esta classe trata da forma como o tempo é guardado no ficheiro save, sendo que o serialize trata da forma como este vai ser guardado no ficheiro e o deserialize trata da forma como este é lido a partir do ficheiro, ambas as funções utilizando o FORMATTER que está guardado nas Utils do Vintage.

2.13. Clock

Uma das funcionalidades principais do projeto seria o facto de podermos saltar no tempo, fazendo com que as encomendas em expedição fossem, caso chegue ou passe o dia de entrega, sejam entregues e atualize tudo o que isso afeta.

O nosso programa além de implementar esses saltos também vai correr em tempo normal, ou seja, se uma entrega demora uma semana a ser entregue e deixarmos o programa a correr durante uma semana, a entrega será finalizada.

A classe Clock, que trata da parte do tempo especificamente, consiste em duas funções, a run e o update. O run tem de ser efetuado sempre que o programa começa para inicializar o relógio, sendo que a hora é definida a partir do save, a não ser que este esteja vazio, nesse caso utiliza a hora do computador, o tempo atual.

A função update serve para demonstrar na UI o relógio a contar os segundos (o nosso Time Travel é feito para qualquer dia, hora e segundo,

desde que seja para o futuro) e mesmo que não esteja no menu principal, o Timer Task faz com que o tempo esteja sempre a correr em background, quase como se fosse uma segunda thread. A cada segundo que passa, a função update também verifica se já passou a hora das encomendas serem entregues e, caso seja verdade, atualiza o seu estado para finalizada.

Para visualizarmos essa alteração, temos de abrir e fechar a janela das encomendas.

2.14. UI

Esta é a classe que trata do menu inicial que é apresentada na nossa UI. Toda a parte gráfica foi formulada utilizando a biblioteca lanterna.

Nesta classe só está definida a função menu que depois nos leva para a parte mais específica do programa que queremos utilizar. Primeiro cria-se o painel onde vai estar todas as opções e cria-se o timer que vai ajudar a controlar o tempo.

Depois disso criamos a label para a data e o tempo e adicionamos esse tempo lá no formato indicado por nós que depois vamos explicar qual é (encontra-se numa das utils).

O nosso programa vai ser constituído por quatro partes principais, logo terá quatro botões: Informação (onde visualizamos os artigos, os utilizadores, as encomendas e as transportadoras com todas as suas informações, Gerir (onde é possível criar novos artigos, utilizadores, encomendas e transportadoras como quisermos, dentro dos possíveis), Controlo (onde se trata do AutoRun, TimeTravel e Wipeout, todas explicadas mais detalhadamente mais à frente) e Estatísticas (onde estão as estatísticas todas da loja).

O último botão é o botão de Sair que nos termina o programa, guardando assim o estado e desligando o timer.

2.15. INFO

Depois de clicarmos no primeiro botão de Informação este leva-nos para o menu de Informação que também terá 4 botões, para visualizar cada uma das partes principais do programa: Artigos, Utilizadores, Encomendas e Transportadoras. Também terá o botão de voltar quando queremos voltar ao menu inicial.

A função menuInformacao trata da criação do painel com estes 5 botões, sendo que cada um depois terá a sua própria função para listar cada

categoria e o botão voltar fará com que o programa volte à função menu mencionada anteriormente.

O primeiro botão Ver Artigos leva-nos para a função listarArtigos que inicialmente irá mostrar uma tabela com algumas das características de cada artigo, sendo estas o seu código, tipo, marca, preço, número de donos, estado e se é premium. Muitas destas informações utilizam Utils que serão explicados no Utils da Info.

Caso a tabela não se encontre vazia, podemos pedir para ver toda a informação detalhada de um artigo, no botão Mais Informação. Esse botão irá abrir um novo painel e utilizar o toString do artigo em questão para mostrar toda a sua informação. Também podemos pedir para este ser removido, que utilizará a função removeArtigo que está definida na função Vintage e será descrita mais à frente.

O segundo botão Ver Utilizadores leva-nos para a função listarUtilizadores que vai tratar da listagem de todos os utilizadores. Tal como a função anterior, vai dar algumas características (código, nome, email, morada, NIF e vendas), mas somente dos utilizadores que se encontram ativos, o botão Mais Informação vai demonstrar de uma forma mais detalhada cada utilizador, utilizando o toString da sua classse e o botão Apagar, apaga o utilizador removendo também os seus artigos associados, salvo algumas exceções que mencionaremos quando falarmos da função apagaUtilizador mais à frente.

O terceiro botão Ver Encomendas leva-nos para a função listarEncomendas que vai listar todas as encomendas. No caso das encomendas, podemos fazer muitas mais coisas em cada uma delas, pois ela são a parte mais importante da aplicação. Para não entrar em muito detalhe, inicialmente aparece na tabela as características mais específicas(código, dimensão, estado, preço e a data de criação), depois em cada encomenda, no botão Mais Informação consegue-se ver mais detalhes, por exemplo, quais artigos estão na encomenda, podemos adicionar novos artigos no botão Adicionar Artigo, remover artigos se ainda tiver dentro do prazo de reembolso no botão Remover Artigo, expedir a encomenda no botão Expedir que depois dependendo do tamanho da encomenda dará a data de entrega e, por fim, podemos cancelar a Encomenda, novamente se tiver dentro do prazo de reembolso, no botão Cancelar, devolvendo os artigos ao mercado, salvo algumas exceções (remover um utilizador, o artigo ainda teria de ser entregue, mas se for cancelada a encomenda esse artigo oficialmente sai do mercado).

O quarto botão Ver Transportadores leva-nos para a função listarTransportadoras que vai listar as transportadoras. No caso das transportadoras, o botão Mais Informação vai mostrar tanta informação quanto se encontra na tabela inicial (Nome, Margem Lucro, Margem Extra, Valor de Expedição e Lucro).

No caso das transportadoras premium, é lhes possível editar os seus valores e todas as transportadoras podem ser apagadas, acontece que os artigos que as têm como transportadoras, simplesmente essas transportadoras são obrigadas a fazer as últimas entregas antes de serem completamente eliminadas do sistema.

2.16. INFOUTILS

Muitas funções desta classe já tinham sido usadas previamente, principalmente em toString's, mas elas foram principalmente criadas por causa da forma como resolvemos mostrar as informações na UI, lidas a partir do objeto em questão. Nesta classe estão definidas 10 funções de parsing:

- parseTipoArtigo: recebe o int que representa o tipo do artigo e transforma-o em String
- parseTipoAtacadores: recebe o int que representa o tipo de atacadores e transforma-o em String
- parsePadrao: recebe o int que representa o padrão da T-Shirt e transforma-o em String
- parseEstadoUtilizacao: recebe o float que representa o estado de utilização e transforma-o em String
- parseDimensao: recebe o int que representa a dimensão de uma encomenda e transforma-o em String
- parseEstadoAtividade: recebe o int que representa se uma conta de utilizador está ativa ou não e transforma-o em String
- parseEstadoEncomenda: recebe o int que representa o estado de uma encomenda e transforma-o em String
- parse ListaArtigos: recebe uma lista de artigos e transforma-a numa representação em texto com os seus códigos (exemplo: [2,4,5])
- parseDimensoes: recebe o array das dimensões de uma mala e transforma-o numa representação em texto (exemplo: 20.0 x 20.0 x 20.0)
- parsePremium: recebe um boolean, se este for verdadeiro então o artigo é premium e mostra "Sim", senão mostra "Não"

2.17. MANAGE

O primeiro botão do menu levava-nos para onde podíamos ver tudo o que está no programa, o segundo botão Gerir autoriza-nos a criar novos artigos, utilizadores, encomendas e transportadores, seguindo passos fáceis e certas regras.

A primeira função menuManutencao funciona de uma forma semelhante à primeira função do Info. Criar 4 botões, cada um para a criação de uma diferente categoria. O primeiro cria artigos, o segundo utilizadores, o terceiro encomendas e por último transportadoras. Igualmente tem o botão de voltar para voltarmos ao menu inicial. Cada um dos quatro primeiro botões vai ter a sua própria função.

O primeiro utiliza a função criarArtigo que é muito complexa, porque dependendo do tipo de Artigo a sua UI vai ter de alterar drasticamente, deixando somente as características comuns e demonstrando as características específicas de cada um dos tipos de artigo.

Todas terão uma caixa para escolher o tipo, que no momento que se altera, altera a configuração da UI demonstrando as características específicas do tipo escolhido. Todas também terão o código do vendedor, o estado, número de donos, descrição, marca, preço base, transportadora e, caso esteja implementado nesse tipo de artigo, o premium.

Na mala, vai ter caixas para colocar comprimento, largura e altura, o material e o ano de coleção. Nas sapatilhas, o tamanho, tipo de atacadores e a cor. Na T-Shirt, o tamanho e padrão.

Quando confirmamos a criação de um artigo este vai ver qual é o seu tipo e depois utiliza a função criaArtigo que se encontra na classe Vintage. No fim da função tem o Listener que é a parte da função que trata de alterar as caixas dependendo do tipo que está a ser utilizado.

O segundo botão utiliza a função criarUtilizador que neste caso é mais simples, pois somente teremos de dizer qual é o seu email, nome, morada e NIF e a função criaUtilizador do Vintage trata do resto.

O terceiro botão utiliza a função criarEncomenda que também só necessita do código do comprador e do tamanho da encomenda para ser criada, pois somente depois na Informação é que iremos adicionar artigos. Também utiliza a função criaEncomenda que se encontra no Vintage.

Por fim, o último botão utiliza a função criarTransportadora que tem um pequeno Listener pois só no caso de ter premium é que terá uma margem extra personalizada, ou seja, todas as transportadoras têm o seu próprio nome e margem de lucro, mas só se estas foram premium é que

também vai aparecer a caixa para acrescentar margem extra. A função criaTransportadora do Vintage trata da criação da Transportadora.

Qualquer erro que possa ocorrer na escrita encontra-se controlado ou pela biblioteca regex ou pelo ErrorHandler que é utiliza dentro das funções do Vintage, que dá o aviso do erro e autoriza a pessoa a alterar o que submetem até aquele momento e quando tudo estiver correto é que a criação é feita.

2.18. MANAGEUTILS

No ManageUtils definidos funções para os casos em que na criação na UI, utilizamos Strings, mas temos de transformar na forma que o objeto em questão os recebe, por exemplo, na primeira função parsePremiumBoolean, nós na UI escolhemos “Sim” ou “Não”, mas o objeto vai ter de receber true ou false e esta função trata desse problema.

O parseEstadoUtilização transforma também a String escolhida nos números utilizados, ou seja, Sem uso é 1.0, Pouco Uso 0.75, Muito Uso 0.5 e Estragado 0.25.

O parseMaterialMala transforma a String do material nos números usados: Tecido é 0, Pele é 1, Lona é 2 e Veludo é 3.

A última parseTamanhoEncomenda transforma a String do tamanho no real tamanho da encomenda: Grande 10 (aguenta 10 artigos), Média 5 e Pequena 1.

2.19. STATS

Foi nos pedido que calculássemos algumas estatísticas no projeto e para as apresentarmos utilizamos a classe Stats, apesar de esta não fazer cálculos nenhums nem tratar do que realmente aparece escrito no ecrã pois isso encontra-se feito noutras classes.

O que esta classe faz é abrir o painel e utilizar o toString da loja Vintage em si que vai apresentar muitas detalhadas informações da loja e certas estatísticas. Contas que não são efetuadas nessa toString são feitas nas StatsUtils.

2.20. STATSUTILS

Esta classe consiste em 8 funções que calculam as estatísticas que têm de ser apresentadas ao administrador. Todas estas funções são utilizadas na toString da Vintage.

As 8 funções são:

- numUtilizadoresAtivos – recebe a lista de utilizadores e diz quantos é que têm a sua atividade como ATIVA
- numUtilizadoresInativos – faz praticamente o mesmo, mas neste caso para utilizadores com atividade INATIVA (utilizadores que foram apagados)
- utilizadorComMaiorFaturacao – percorre todos os utilizadores e envia o código do utilizador que tem o maior valor em vendas
- numEncomendasPendentes – percorre todas as encomendas e calcula quantas estão pendentes
- numEncomendasExpedidas – calcula quantas estão expedidas
- numEncomendasFinalizadas – calcula quantas estão finalizadas
- transportadoraMaiorValorExpedicao – percorre todas as transportadoras e calcula qual delas têm um maior custo de expedição
- transportadoraMaiorLucro – percorre todas as transportadoras e descobre qual foi a que deve maior lucro até agora

2.21. CONTROLCENTER

No menu principal, o botão de controlo leva-nos para a classe ControlCenter que trata da parte do UI das funções de controlo, sendo estas o AutoRun, o TimeTravel e o Wipeout.

Esta classe consiste numa só função, o menuControlo, que cria o painel e apresenta os 3 botões do AutoRun, TimeTravel e Wipeout, mais o botão de Voltar que nos faz voltar ao menu inicial.

O botão AutoRun executa o ficheiro csv que trata da automatização do programa a partir da função readAndExecute da classe AutoRun.

O botão Time Travel abre o menuTimeTravel que está implementado na classe TimeTravel e a partir daí é que trata do salto no tempo.

O botão Wipeout esvazia a loja inteira, deixando-o como se viesse de fábrica sem qualquer informação, para fazer tal coisa utiliza a função wipeAll que está implementada na classe Vintage.

2.22. TIMETRAVEL

Depois de clicarmos no botão do TimeTravel no menu de controlo abrimos o painel que vai tratar do salto temporal, que está implementado nesta classe.

Apesar de no enunciado se falar em saltos temporais diários, nós achamos que um salto temporal específico faria mais sentido, apesar de ser mais complicado de implementar.

Devido a este facto, o painel do TimeTravel pede uma data e uma hora para a qual vai saltar. A função timeTravel que está no Vintage é que vai tratar de todos os acontecimentos e erros, por exemplo, tentar viajar para o passado.

2.23. AUTORUN

Esta classe vai tratar de toda a automatização feita a partir do ficheiro csv. No início da classe, temos documentado a forma como cada comando tem de ser escrito no csv. Felizmente a forma como este autorun funciona, se este encontra algum erro, ele simples ignora-o e não produz, por exemplo, um artigo sem preço ou algo do género.

Inicialmente temos de definir o local onde vai estar o csv ao autorun, guardando assim um RUNNER_FILEPATH.

Depois criamos um HashMap onde vamos guardar todos os métodos de acordo com um texto dado.

A função AutoRun vai à função mapCodesToMethods que vai basicamente transformar certos textos em formas de correr uma função, por exemplo, criaArtigo vai buscar ao Vintage a função criaArtigo, travelTo vai buscar ao Vintage a função timeTravel, coisas desse género.

Depois do AutoRun estar implementado, é necessário criar a função readAndExecute que vai tratar da leitura de cada linha, por exemplo, saber a divisão entre cada parâmetro, se o ficheiro já terminou ou qual o tipo de runMethod que vai precisar de ser usado.

Existem três runMethod, um para se for algo com mais de dois textos, ou seja, que requer uma string de informação (criação de um artigo, por exemplo), um que tenha mais que um, mas menos que três, pois esse é para o runMethod anterior, ou seja, só tem uma string de informação(remover um artigo, só necessita do seu código) e o último que só tem o texto mesmo, logo é só uma função principal (Wipeout, por exemplo).

Foi a partir destes métodos que conseguimos automatizar o nosso programa da melhor forma possível, apesar de a automatizar ser toda feita de uma só vez.

2.24. ERRORCODE

Nós temos duas classes que foram utilizadas para tratar de todos os possíveis erros do programa, sendo elas o ErrorCode e o ErrorHandler. Acontece que o ErrorCode não é bem uma classe, mas sim um Enum para facilitar a procura de cada erro.

Nós encontramos 20 tipos de erros e 2 situações onde também poderia ser necessário mandar mensagem para dizer que tudo funcionou.

Dessas duas situações tem o NO_ERRORS, que simplesmente continua a correr o programa como se nada tivesse acontecido e a outra é o AUTORUN_SUCCESS que avisa quem está a utilizar o programa que o autorun foi feito com sucesso.

Depois temos as 20 situações de erro que são:

NOME	DESCRIÇÃO	EXEMPLO
PARAMETRO_ERRADO	Quando ao criar/alterar alguma coisa, se mete um parâmetro incorreto	Ao criar um artigo deixar parâmetros vazios
CODIGO_INVALIDO	Tentar utilizar códigos de artigo ou utilizador inexistentes	Criar uma encomenda com um utilizador inexistente
DATA_INVALIDA	Escrever a data ou hora de forma errada	Não escrever a data ou hora direito no TimeTravel
ARTIGO_EXPEDIDO	Tentar efetuar uma ação com um artigo já expedido	Tentar remover um artigo que já foi expedido numa encomenda
DIMENSOES_INVALIDAS	Meter dimensões impossíveis numa mala	Escolher uma dimensão que resultaria num preço negativo
EMAIL_INVALIDO	Email com formato errado	Email sem o @
NIF_INVALIDO	NIF com formato errado	NIF com menos de 9 números
SEM_ESPACO	Encomenda cheia	Tentar adicionar um artigo numa encomenda cheia
ENCOMENDA_VAZIA	Encomenda vazia	Tentar remover artigo duma encomenda vazia
EM_ENCOMENDA	Artigo já numa encomenda	Tentar adicionar artigo em duas encomendas diferentes
EM_EXPEDICAO	Encomenda já expedida	Tentar expedir uma encomenda quando esta já foi
SEM_REEMBOLSO	Impossível pedir reembolso	Tentar pedir reembolso depois do prazo autorizado
ARTIGO_DO_COMPRADOR	Artigo do comprador	Tentar adicionar a uma encomenda um artigo do próprio comprador
ARTIGO_INVALIDO	Artigo não se encontra nesta encomenda	Tentar remover um certo artigo de um encomenda apesar de ele não estar lá
TRANSPORTADORA_INVALIDA	Transportadora não existente	Escolher transportadora que não existe
TRANSPORTADORA_NAO_PREMIUM	Transportadora não premium	Escolher uma transportadora não

		premium para um artigo premium
PREMIUM_REQUIRED	Só para transportadoras premium	Tentar alterar valores de transportadoras não premium
TRANSPORTADORA_EM_USO	Transportadora a ser usada	Tentar apagar transportadora que está a ser usada
AUTORUN_ERROR	Erro no autorun	Caso aconteça algum erro inesperado no autorun
UTILIZADOR_INATIVO	Utilizador inativo	Tentar criar um artigo com um utilizador inativo
DATA_PASSADA	Data antes do tempo atual da máquina	Tentar dar TimeTravel para uma data anterior ao momento da máquina

2.25. ERRORHANDLER

A classe ErrorHandler é a classe que vai pegar nos códigos entregues pelo ErrorCode e originar as caixas de texto a avisar qual é o erro em específico. Para isso foi criada a função handleError que recebe a gui e o errorCode que é para ser usado.

Esta função utiliza um switch/case que ao encontrar o código específico origina a caixa de texto específica e depois de a fechares sai da função, menos o NO_ERRORS que não faz nada.

Tal como existem 22 ErrorCodes vão existir 22 casos cada um com a sua mensagem específica, exceto o NO_ERRORS:

CASO	MENSAGEM
ARTIGO_EXPEDIDO	"O artigo já foi expedido."
SEM_ESPACO	"A encomenda está cheia."
ENCOMENDA_VAZIA	"A encomenda está vazia."
EM_ENCOMENDA	"O artigo já está noutra encomenda."
EM_EXPEDICAO	"A encomenda já foi expedida."
SEM_REEMBOLSO	"O prazo de reembolso já ultrapassou."
PARAMETRO_ERRADO	"Erro em parâmetros obrigatórios."
CODIGO_INVALIDO	"O código introduzido é inválido."
TRANSPORTADORA_INVALIDA	"A transportadora introduzida é inválida."
PREMIUM_REQUIRED	"Só transportadoras Premium têm acesso a esta funcionalidade."
TRANSPORTADORA_NAO_PREMIUM	"Artigos Premium têm de ser transportados por Transportadoras Premium"
TRANSPORTADORA_EM_USO	"A transportadora está a ser utilizada"
ARTIGO_DO_COMPRADOR	"O artigo pertence ao comprador da encomenda."
ARTIGO_INVALIDO	"O artigo não pertence a esta encomenda."
DIMENSOES_INVALIDAS	"A mala apresenta dimensões inválidas."
DATA_INVALIDA	"A data introduzida não está formatada corretamente."
EMAIL_INVALIDO	"Formato de e-mail inválido."
NIF_INVALIDO	"NIF inválido."

AUTORUN_ERROR	"Falha ao correr a operação AutoRun."
AUTORUN_SUCCESS	"Operação realizada com sucesso."
UTILIZADOR_INATIVO	"Esse utilizador já não existe"
DATA_PASSADA	"Essa data já passou"

2.26. VINTAGE

Esta é a classe onde vai estar tudo guardado, onde as funções principais de criação, remoção e gestão estão definidas, muitas estatísticas vão estar a ser guardadas e o tempo da máquina.

Os atributos principais são:

- private List<Artigo> artigos (a lista de todos os artigos)
- private List<Encomenda> encomendas (a lista de todas as encomendas)
- private List<Utilizador> utilizadores(a lista de todos os utilizadores)
- private List<Transportadora> transportadoras (a lista de todas as transportadoras)
- private int codigoProximoArtigo (o código do próximo artigo)
- private int numVendas (número de vendas feitas até ao momento)
- private float totalFaturado (total de dinheiro que passou pela Vintage)
- private LocalDateTime tempoAtual (hora atual do programa)

O nosso Vintage tem dois construtores, pois se acontecer algum erro no load consideramos que a loja está vazia ou errada, logo tem de criar uma nova loja de raiz. Novas listas para artigos, encomendas, utilizadores e transportadoras, dar reset ao codidoProximoArtigo, numVendas e totalFaturado e por fim atualiza o tempo para o tempo do computador.

O outro construtor, como o save funciona direito, simplesmente dá get de cada um dos atributos.

Depois dos construtores temos as funções que foram mencionadas, as de criação, remoção e gestão de cada lista. Primeiro, as que tratam dos artigos, depois as das encomendas, utilizadores e no fim as das transportadoras. Também tem a função do timeTravel e a que faz Wipeout da loja, reiniciando-a. No final da classe, tem os getters e setters e a toString que é utilizada nas Stats.

É também nesta classe que tratamos de todos os erros que são possíveis de encontrar, devolvendo a maioria das funções um ErrorCode.

A primeira função dos Artigos neste caso é a função criaArtigo. Algumas destas funções vão estar normalmente envolvidas por um try/catch pois, por exemplo nesta função, caso aconteça alguma Exception, assumimos que há algum erro nos parâmetros e devolvemos o ErrorCode PARAMETRO_ERRADO.

Na criaArtigo, primeiro trata-se dos atributos que são comuns a todos os tipos de artigos, atribuimos o seu código e aumenta-se e vamos buscar a transportadora.

Verificamos se o código de Vendedor existe ou se esse Vendedor ainda está ativo e vemos se a transportadora que tentamos atribuir existe. Depois de verificarmos estes erros, procedemos à tentativa da atribuição das características mais específicas de cada um. Para isso usamos um switch com o tipo de cada artigo.

No caso da mala, pegamos nas informações, verificamos se no caso de a mala ser premium, estamos a atribuir também uma transportadora premium e se as dimensões atribuídas são possíveis. Depois de termos a certeza que é um artigo correto, criamos o artigo, adicionamo-lo à lista de artigos e à lista de artigos do utilizador que o criou, para ir buscar o utilizador usamos uma função dos Utils do Vintage. Nesta classe, iremos utilizar vários vezes funções desse Utils e depois explicaremos cada uma.

No caso das sapatilhas, pegamos nas informações, só temos de verificar o caso da transportadora premium com um artigo premium e procedemos à criação do artigo. Adicioná-lo à lista de artigos e à lista de artigos do utilizador que o criou.

O último caso, a TShirt, não temos de verificar qualquer erro, logo pegamos no resto das informações desse artigo, criamó-lo, adicionamos à lista de artigos e à lista de artigos do utilizador que o criou.

No final, se tudo correr direito, devolvemos o ErrorCode NO_ERRORS. Todos funcionarão dessa forma.

A segunda função dos Artigos é a removeArtigo que trata da remoção de um artigo. Esta função recebe o código do artigo, vai buscá-lo e vê se ele está em alguma encomenda.

Se tiver numa encomenda e está já tiver sido expedida, dá o ErrorCode ARTIGO_EXPEDIDO. Se tiver num encomenda e já tiver passado os dias de reembolso, dá o ErrorCode SEM_REEMBOLSO. Se tiver numa encomenda, mas não tiver passado o tempo de reembolso,

primeiro remove o artigo dessa encomenda. Depois remove-o da lista de listados do utilizador dele e por fim é retirado da lista de artigos.

Depois das funções dos artigos tem as funções sobre as encomendas.

A primeira função é a criaEncomenda. Tal como a criaArtigo também está rodeada por um try/catch, caso não se preencha um parâmetro.

O código da encomenda vai ser o tamanho da lista de encomendas e o codigo do comprador e a sua dimensão são dados pela UI. Caso o código de comprador não exista devolve CODIGO_INVALIDO e caso seja de um utilizador inativo devolve UTILIZADOR_INATIVO.

Por fim, cria a encomenda e a adiciona-a à lista de encomendas.

Para a encomenda temos também o adicionarArtigoEmEncomenda e a removerArtigoEmEncomenda. A primeira tenta adicionar um artigo a uma certa encomenda e caso esteja errado devolve o ErrorCode CODIGO_INVALIDO, ou devolve um dos errors que pode acontecer na função adicionarArtigo que está definida na classe Encomenda. A segunda tenta remover um artigo de uma encomenda. Novamente, tem de ver se o código é válido, mas neste caso também tem de ver se não passou do tempo de reembolso, senão terá de devolver SEM_REEMBOLSO. Verifica os erros da removerArtigo da classe Encomenda e se não houver remove-a da encomenda.

Além disso, também vai ter mais três funções de gestão da encomenda.

A expedirEncomenda que recebe o código da encomenda, vê se esta ainda não foi expedida e se não tiver sido, altera o seu estado para expedida e determina a sua data de entrega dependendo do tamanho da encomenda.

A entregarEncomendas é uma função que no nosso caso é efetuada todos os segundos, pois verifica se as encomendas existentes já foram finalizadas ou não. Se não tiver em estado de EXPEDIDA ignora-a e se a sua data de entrega ainda for anterior ao tempo atual então ainda não temos de alterar o seu estado. Caso contrário, vai ver quem é o comprador, efetua a compra dos artigos todos por parte do comprador, aumenta o totalFaturado da Vintage e o seu número de vendas, aumenta o lucro de cada transportadora e remove o artigo da lista de Artigos. No fim muda o estado da encomenda para finalizada.

A cancelaEncomenda recebe o código da encomenda e caso ainda não tenha passado do tempo de reembolso, simplesmente remove a encomenda da lista de encomendas.

Depois das encomendas temos as duas funções que tratam dos utilizadores.

A função criaUtilizador também funciona com um try/catch para o caso de deixarem parâmetros vazios na UI.

Para definir o código utiliza o tamanho da lista de utilizadores (pois apesar de a próxima função ser apagaUtilizador, eles nunca são apagados), fazemos um pattern para o email e caso não esteja correto devolvemos EMAIL_INVALIDO e caso o NIF não tenho 9 dígitos devolvemos NIF_INVALIDO. Depois dessas verificações adicionamos o novo utilizador à lista de utilizadores.

Na função apagaUtilizador, recebemos o código do utilizador e descobrimos qual é. Definimos as suas características para null e removemos os seus artigos da lista de artigos que não estejam em encomendas expedidas ou encomendas que já ultrapassam o tempo de reembolso. Colocamos a sua lista de listados em null e pomos a sua atividade em NULL.

Por fim, antes das funções de controlo, temos as que tratam das transportadoras.

A criaTransportadora recebe o nome, margemLucro, margemExtra e se é premium. Mesmo que não seja premium vai receber uma margemExtra, pois esta é por default 1 para as que não são premium. Cria a transportadora e adiciona-a à lista de transportadoras. Caso tenha acontecido algum erro, teria mandado PARAMETRO_ERRADO até ser corrigido.

A apagaTransportadora simplesmente remove a transportadora da lista de transportadoras se esta não estiver a ser utilizada por nenhum artigo.

As duas funções de controlo são o timeTravel e o wipeAll.

A função timeTravel recebe uma string com a data e a hora, se está tiver errada devolve DATA_INVALIDA, se tentarmos voltar para uma data anterior ao tempo atual da máquina devolve DATA_PASSADA e caso não aconteça nenhum erro muda o tempo atual da máquina para a data e hora

recebida e verifica as encomendas para ver se alguma foi entregue nessa mudança de tempo.

A função wipeAll simplesmente renova todas as listas de artigos, encomendas, utilizadores, transportadoras, mete o código do próximo artigo a 0, o número de venda e o total faturado também. Por fim mete o tempo atual com o tempo do computador.

2.27. UTILS(VINTAGE)

A classe Vintage tal como muitas outras classes nossas também tem as suas Utils, sendo neste caso principalmente para ir buscar certas partes a certas listas, ou seja, são principalmente getters de listas.

No início tem o FORMATTER para a forma como queremos que se representa a data.

Depois temos 5 funções de getters:

- getArtigo – recebe a lista de artigos e o código de um artigo e caso ele exista envia o Artigo, caso contrário envia NULL
- getEncomenda – recebe a lista de encomenda e o código de uma encomenda e caso ela exista envia a Encomend, caso contrário envia NULL
- getUtilizador – recebe a lista de utilizadores e o código de um utilizador e caso ele exista envia o Utilizador, caso contrário envia NULL
- getTransportadora – recebe a lista de transportadoras e o código de uma transportadora e caso ela exista envia a Transportadora, caso contrário envia NULL
- getEncomendaOfArtigo – recebe a lista de encomendas e um artigo e devolve o código da encomenda em que esse artigo está ou devolve -1 caso esteja em nenhuma

A última função das Utils é a função isArtigoInEncomendaExpedida que recebe a lista de encomendas e um artigo e vê se esse artigo se encontra numa encomenda já expedida ou finalizada, caso seja verdade devolve true, caso contrário devolve false

2.28. MAIN

Na main, começamos com o prepareGsonLoader para iniciar o nosso gson que depois trata do load para criar a loja(Vintage). Criamos o AutoRun e o Timer, iniciamos o relógio e testamos se há entregas por alterar de estado.

Depois disso, criamos a UI toda, começamos o screen e abrimos no menu. No final, fechamos o screen, terminamos o timer e damos save à loja.

Capítulo 3
Diagrama de classes

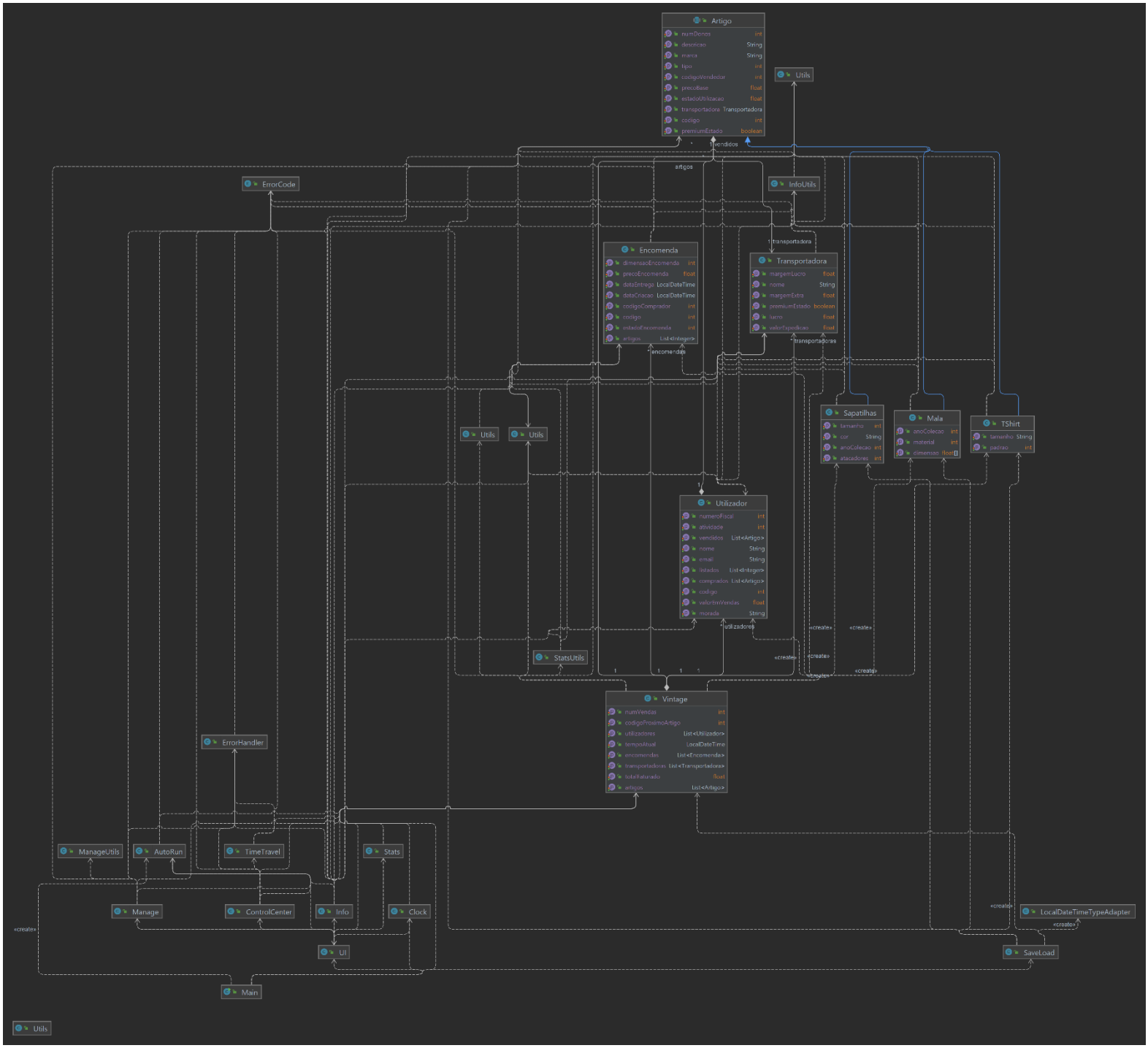


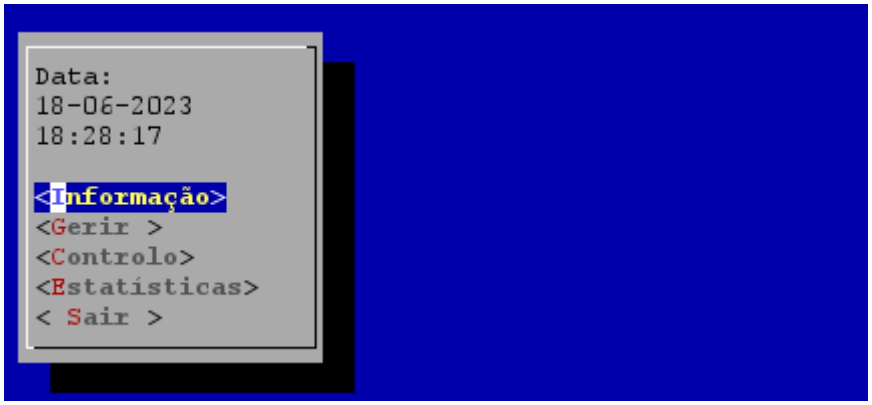
Diagrama de classes feito pelo IntellijA

Capítulo 4

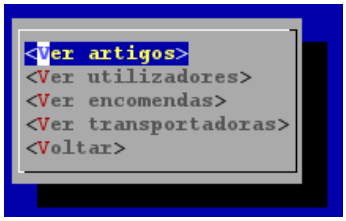
Aplicação desenvolvida

Neste capítulo vamos fazer uma ilustração de todas as funcionalidades, mais em concreto da UI implementada. Uma das características que não vamos apresentar todas, são todas as caixas de erro que aparecem, pois são demasiadas e não achamos que haja necessidade.

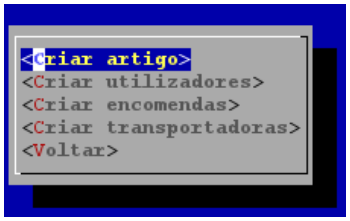
Menu Inicial:



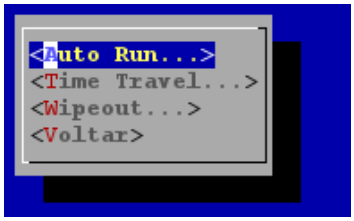
Menu de Informação:



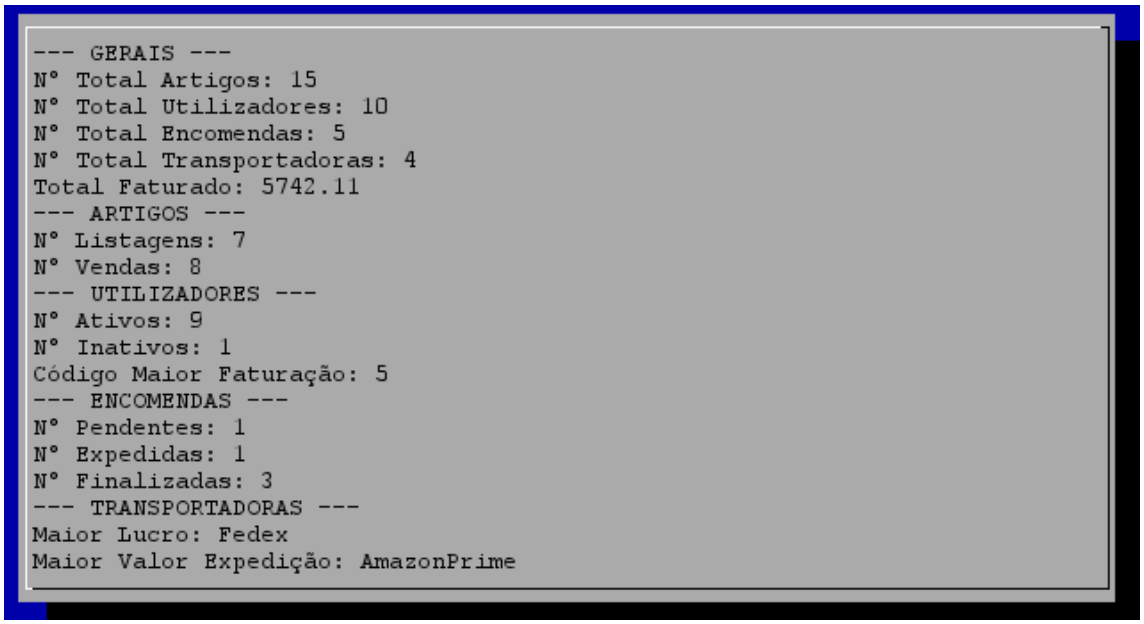
Menu de Gerir:



Menu de Controlo:



Estatísticas de um programa:

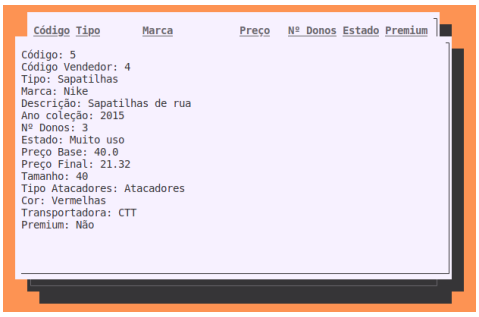


Lista dos artigos

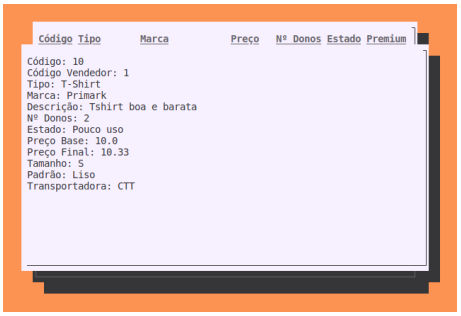
Código	Tipo	Marca	Preço	Nº Donos	Estado	Premium
0	Mala	Gucci	1564.76	1	1.0	Sim
1	Mala	Tous	84.85	1	0.75	Não
2	Mala	Tommy Hillfiger	24.44	4	0.25	Não
3	Mala	Chanel	557.48	2	1.0	Não
4	Mala	Hugo Boss	3613.42	1	0.75	Sim
5	Sapatilhas	Nike	21.32	3	0.5	Não
6	Sapatilhas	Adidas	1368.5	1	1.0	Sim
7	Sapatilhas	Puma	4.18	8	0.25	Não
8	Sapatilhas	Sport Zone	51.51	0	1.0	Não
9	Sapatilhas	Decathlon	20.58	2	1.0	Não
10	T-Shirt	Primark	10.33	2	0.75	Não
11	T-Shirt	Bershka	4.23	4	0.5	Não
12	T-Shirt	H&M	2.65	8	0.25	Não
13	T-Shirt	Lefties	7.93	2	0.75	Não
14	T-Shirt	Primark	20.66	0	1.0	Não

Mais informação de uma mala:

Mais informação de umas sapatilhas:



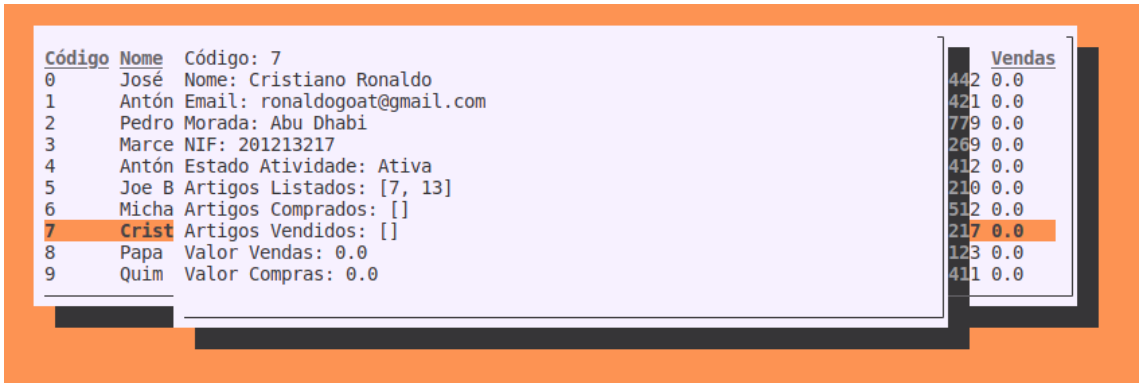
Mais informação de uma T-Shirt:



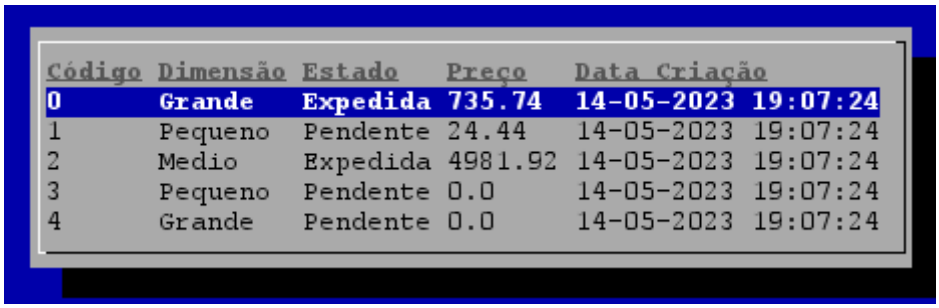
Lista dos utilizadores

Código	Nome	Email	Morada	NIF	Vendas
0	José matos	jose.matos@email.com	Rua Monte de Aldão	255372442	0.0
1	António Silva	antoniofcsilva2003@gmail.com	Rua de São Brás	258877421	0.0
2	Pedro Silva	pedrometaleiro@gmail.com	Rua da Coutada	258891779	0.0
3	Marcelo Sousa	marcelorebelodesousa@gmail.com	Palácio de Belém	150917269	0.0
4	António Costa	antoniocosta@gmail.com	Rua da Amadora	194317412	0.0
5	Joe Biden	joebiden@gmail.com	White House	143219210	0.0
6	Michael Jordan	michaeljordan@gmail.com	Chicago	180213512	0.0
7	Cristiano Ronaldo	ronaldogoat@gmail.com	Abu Dhabi	201213217	0.0
8	Papa Francisco	papafrancisco@hotmail.com	Vaticano	144213123	0.0
9	Quim Barreiros	quimbarreiros@gmail.com	Vila Nova de Cerveira	182123411	0.0

Mais informação de um utilizador

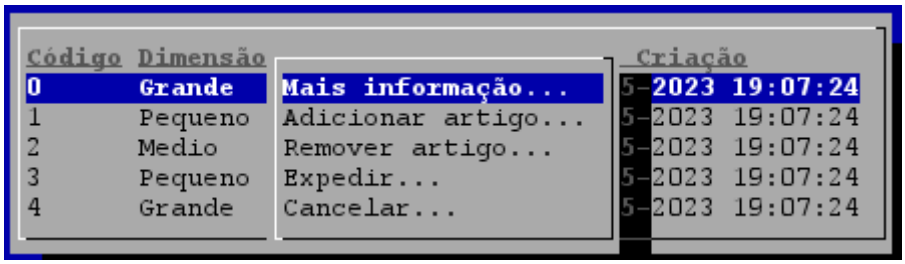


Lista de encomendas

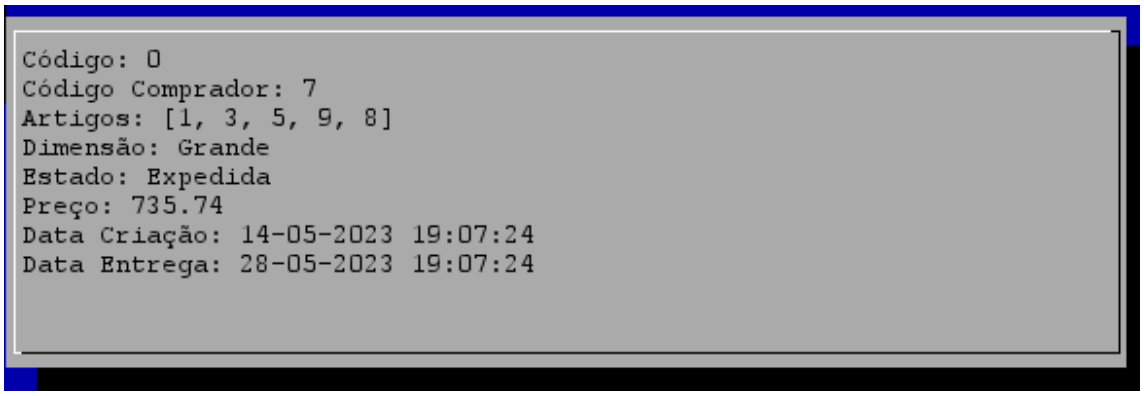


Código	Dimensão	Estado	Preço	Data Criação
0	Grande	Expedida	735.74	14-05-2023 19:07:24
1	Pequeno	Pendente	24.44	14-05-2023 19:07:24
2	Medio	Expedida	4981.92	14-05-2023 19:07:24
3	Pequeno	Pendente	0.0	14-05-2023 19:07:24
4	Grande	Pendente	0.0	14-05-2023 19:07:24

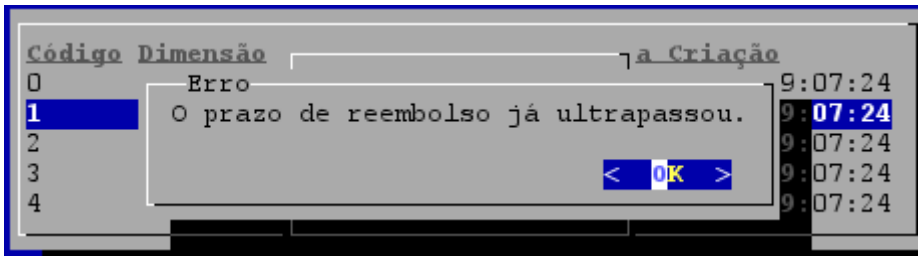
Possíveis manuseamentos de uma encomenda:

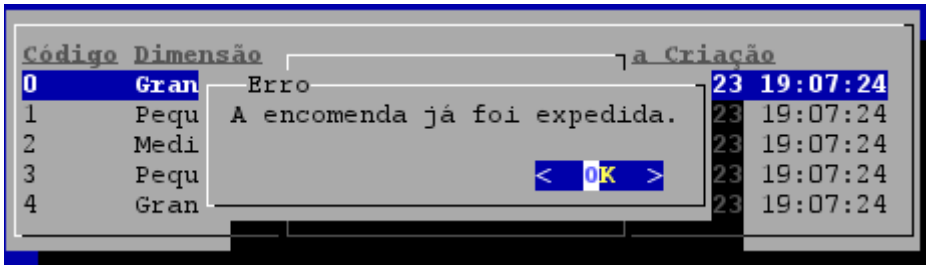


Mais informação de uma encomenda:



Erros comuns de encomenda:



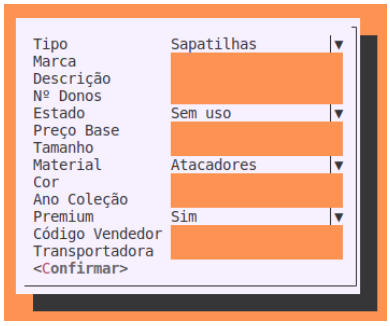


Lista de transportadoras

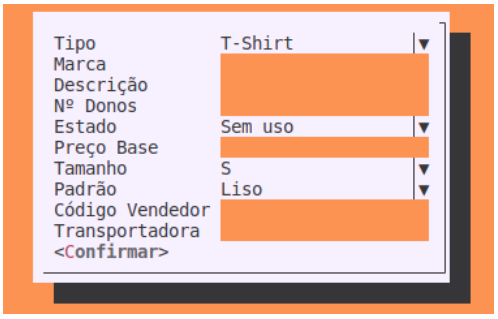
Nome	Margem Lucro	Margem Extra	Valor de Expedição	Lucro	Premium
CTT	1.2	1.0	3.3	0.0	Não
Torrestir	1.1	1.0	3.03	0.0	Não
AmazonPrime	1.5	0.99	4.13	0.0	Sim
UPS	1.05	1.0	2.89	0.0	Não
Fedex	1.4	0.95	3.85	0.0	Sim

Criação de uma mala:

Criação de umas sapatilhas:

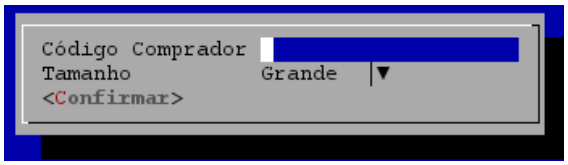


Criação de uma T-Shirt:



Criação de um utilizador:

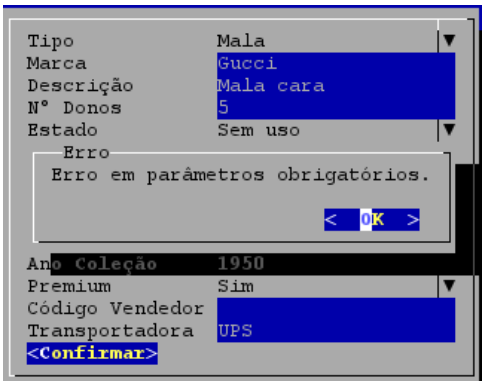
Criação de uma encomenda:



Criação de transportadora normal:

Criação de transportadora premium:

Erros habituais em criações:



Capítulo 5

Conclusão

Com este projeto conseguimos aprender mais sobre herança de classe por causa da forma como programamos os artigos, de forma a conseguir futura implementação de outros tipos.

Achamos que conseguimos respeitar as principais regras de POO, respeitando o encapsulamento e a criação de vários objetos para diferentes necessidades.

Com a criação da nossa UI, agora sabemos apresentar de uma melhor forma UI utilizando bibliotecas externas, sabemos como automatizar um programa utilizando um ficheiro csv que ajuda a tornar um programa mais autónomo e com a atualização do Clock aprendemos a ter mais atenção a forma como o tempo funciona num programa.

A utilização do premium também nos deixou a entender que em qualquer momento uma parte do programa pode necessitar de alterações e que temos de saber como resolver isso da melhor forma.

Concluindo, conseguimos concordar entre nós que temos um trabalho bem composto com todas as funcionalidades pedidas pelos professores no enunciado.