

STARTTHRIVE – GESTOR DE EMPRESAS

PROJETO FINAL

Leonor Reis, nº 2019210146
Francisco Mendes, nº 2019222823

Prof. Responsável: Marília Curado
Programação Orientada aos Objetos
Ano letivo 2022/23

Índice

Introdução.....	3
Diagrama de classes em UML	4
Explicação do UML	4
Classes	6
Classe GestorEmpresas.....	5
Classe SuperProjeto	5
Classe Empresa	5
Classes EmpresaRestauracao e EmpresaMercearia	5
Restantes classes	6
Classes abstratas.....	6
Privacidade dos atributos	7
Coisas que não estão explicitamente definidas no enunciado	7
Organização dos ficheiros de texto e objetos.....	7
Explicação detalhada sobre a implementação	8
Listar empresas.....	8
Criar empresa	9
Editar Empresa	10
Apagar Empresa	12
Listar os dois maiores restaurantes	13
Relatório empresa	14
Conclusão.....	16

Introdução

Este projeto tem como finalidade desenvolver e testar os conhecimentos adquiridos ao longo deste semestre na cadeira de Programação Orientada aos Objetos. Para isso, foi proposta a construção de um programa para a empresa StarThrive, que tem como objetivo gerir várias empresas. O foco do projeto está em atingir os seguintes aspetos: a possibilidade de listar todas as empresas geridas pela StarThrive, assim como a respetiva informação; apresentar para cada tipo de empresa, a empresa com maior receita anual (nome e valor), a empresa com menor despesa anual (nome e valor) e a empresa com maior lucro anual (nome e valor do lucro); apresentar as 2 empresas do tipo restauração com maior capacidade de clientes por dia.

De forma a atingir os objetivos, irá ser feita uma gestão das empresas, sendo que se dividem em empresas de restauração (cafés, pastelarias, restaurantes locais e restaurantes fast-food) e em empresas de mercearia (mercados e frutarias). Tudo isto irá dar origem num total de 13 classes. Neste relatório, as classes serão explicadas ao detalhe, e irão ser mostradas todas as decisões tomadas ao longo do projeto. Segue-se a análise dos resultados do programa e finalmente uma breve conclusão.

Todas as interações com o programa serão feitas através de uma interface gráfica, e que os dados estão contidos em ficheiros de texto, que irão ser passados a ficheiros de objetos para serem lidos pelo programa.

O projeto terá ainda um diagrama UML, que contém a sua estrutura geral.

Diagrama de classes em UML

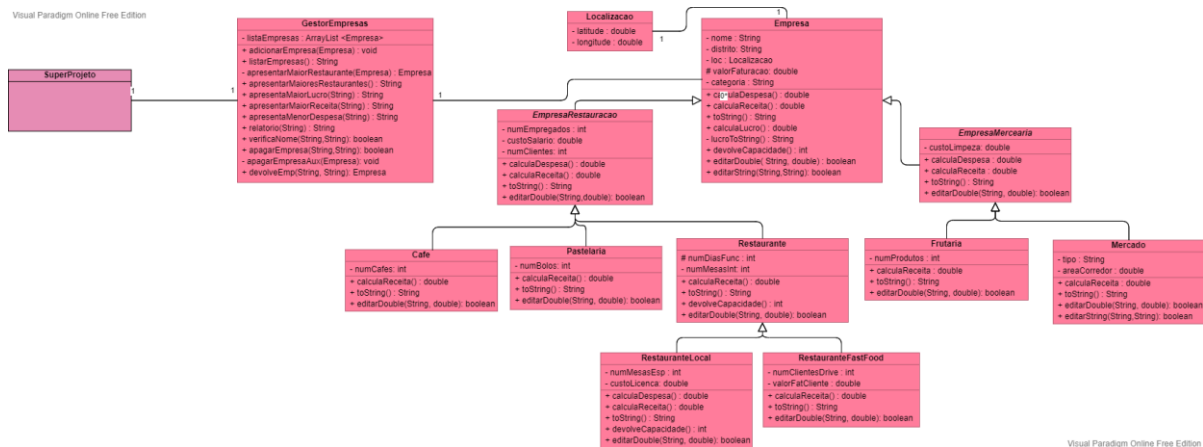


Imagem 1 - UML

EXPLICAÇÃO DO UML

(- Explicar porque é que os atributos passaram para as classes pai

- Justificar porque não criamos uma classe abstrata entre café e pastelaria)

Como podemos ver pela imagem 1, o diagrama UML apresenta várias relações de herança, sendo que, em conjunto com o polimorfismo, são os pontos principais deste projeto. Temos o GestorEmpresas, que pode gerir 0 ou mais empresas, que contém a lista das empresas que gere, em conjunto com as operações principais que o projeto pede nos métodos que apresenta; a classe Empresa, que tem 1 localização (composta pela longitude e pela latitude), e é nela que começam as relações de herança. Apresenta os atributos comuns a todo o tipo de empresas, assim como os métodos que usufruem do polimorfismo; no primeiro nível de herança, estão as classes EmpresaResturacao e EmpresaMercearia, abstratas, que contém os atributos comuns a todas as empresas desse tipo, assim como os métodos que usam polimorfismo; a classe Restaurante, abstrata, que apresenta os atributos comuns aos 2 tipos de restaurantes existentes, assim como os métodos que usam polimorfismo; e finalmente, no último nível de herança, apresentam-se as classes Cafe, Pastelaria, RestauranteLocal, RestauranteFastFood, Frutaria e Mercado, que contém os atributos específicos dessas empresas, e os métodos que usam polimorfismo.

Dependendo de quando já existem os atributos suficientes de forma a se calcular a receita ou a despesa de uma empresa, é que vamos saber em que classe os métodos calcularReceita() e calculaDespesa() realizam as operações pedidas.

Classes

CLASSE GestorEmpresas

Nesta classe, é onde se encontra o ArrayList com a lista das empresas, e é onde se encontram as funções que permitem adicionar empresas novas, listar todas as empresas, apresentar as 2 empresas de restauração com maior capacidade de clientes através da função compareCapacidade() que se encontra na classe Empresa, e apresentar a empresa com maior lucro, maior receita e menor despesa, através dos métodos compareLucro(), compareReceita() e compareDespesa() que fazem parte da classe Empresa. Estas três últimas operações são executadas pelo método relatório.

CLASSE SuperProjeto

É a classe “main” do projeto. Esta classe inicia o programa ao criar um objeto do tipo GestorEmpresas, e dependendo se já existe ficheiro de objetos ou não, executa a leitura ou do ficheiro de texto, ou do ficheiro de objetos. Apresenta ainda a inicialização dos componentes da interface gráfica.

CLASSE Empresa

Esta classe é importante principalmente porque contém os métodos que servem para a classe GestorEmpresas fazer as comparações necessárias, de forma a encontrar a empresa com maior lucro, a com maior receita, a com menor despesa, e as duas do tipo restauração com maior capacidade.

CLASSES EmpresaRestauracao E EmpresaMercearia

São as classes de primeiro nível de herança, sendo que herdam as características da classe Empresa. Nelas, através do polimorfismo, é calculada a despesa e a receita de cada empresa, caso já tenham informação suficiente.

REstantes CLASSES

As restantes classes são principalmente as classes de segundo nível de herança que vão dar origem aos objetos que identificam cada empresa, sendo elas: Cafe, Pastelaria, Mercado, Frutaria, *RestauranteLocal* e *RestauranteFastFood*. Nelas, através do polimorfismo, é calculada a receita e a despesa de cada uma, caso ainda não tenha sido calculada nas classes das quais é herdeira. Existe ainda uma classe *Localização*, que serve para determinar as coordenadas de cada empresa (latitude e longitude).

O método *calculaDespesa()* está redefinido na classe abstrata *EmpresaRestauração* porque a despesa de todas as empresas de restauração inclui o produto do número de funcionários pelo salário. Logo, só na subclasse *RestauranteLocal* é que este método é redefinido porque existem mais despesas, para além das despesas anteriormente referidas.

O método *devolveCapacidade()* está definido na classe *Empresa* e volta a ser redefinido na classe abstrata *Restaurante* porque a capacidade de todos os restaurantes inclui o número de mesas interiores. Logo, só na subclasse *RestauranteLocal* é que este método é redefinido porque existem mais despesas, para além das despesas anteriormente referidas.

CLASSES ABSTRATAS

Neste contexto, fez sentido a criação das classes *EmpresaRestauracao*, *EmpresaMercearia* e *Restaurante* como classes abstratas porque não é necessário gerar instâncias destas classes. Além disso, permite-nos normalizar a linguagem a partir deste ponto na hierarquia e tirar partido do polimorfismo.

Nestas classes também foram criados métodos construtores para que seja possível que as subclasses tenham também métodos construtores por omissão. Foram criados métodos construtores que recebem argumentos de inicialização dos atributos porque este construtor é chamado pelas subclasses.

PRIVACIDADE DOS ATRIBUTOS

O atributo `valorFaturacao` na classe *Empresa* é `protected` porque nas subclasses é necessário o acesso a este atributo para o cálculo da receita anual.

Os atributos `numEmpregados` e `custoSalario` na classe *EmpresaRestauracao* são `protected` porque nas subclasses são necessários para o cálculo da despesa anual.

O atributo `numDiasFunc` na classe *Restaurante* é `protected` porque nas subclasses é necessário o acesso a este atributo para o cálculo da receita anual.

COISAS QUE NÃO ESTÃO EXPLICITAMENTE DEFINIDAS NO ENUNCIADO

Os métodos `calculaLucro()` e `lucroToString()` são responsáveis pelo cálculo do lucro e correspondência à string (“SIM”/“NÃO”) adequada, respetivamente. Caso a empresa tenha um lucro igual a zero, é considerado que a empresa não teve lucro.

Caso duas empresas da mesma categoria tenham a mesma receita, lucro ou despesa, é considerado o restaurante que aparece primeiro na lista de empresas.

Organização dos ficheiro de texto e de objetos

Quanto ao ficheiro de texto, este apresenta todas as empresas que o programa ao iniciar, vai carregar para lista das empresas, assim como todos os seus respetivos atributos, dependendo do tipo de empresa. Os atributos estão separados por “;”, de forma a facilitar a atribuição de cada atributo à variável correspondente. De referir ainda que a distinção entre os tipos de empresas acontece logo na primeira palavra das linhas do ficheiro, sendo que representa a categoria.

Após o programa fazer a primeira leitura, os objetos são escritos para um ficheiro de objetos, e é a partir dele que o programa passa a ler os dados daí em diante. Qualquer alteração que um utilizador faça quando corre o programa, é também guardada no ficheiro de objetos.

Explicação detalhada sobre a implementação

LISTAR EMPRESAS

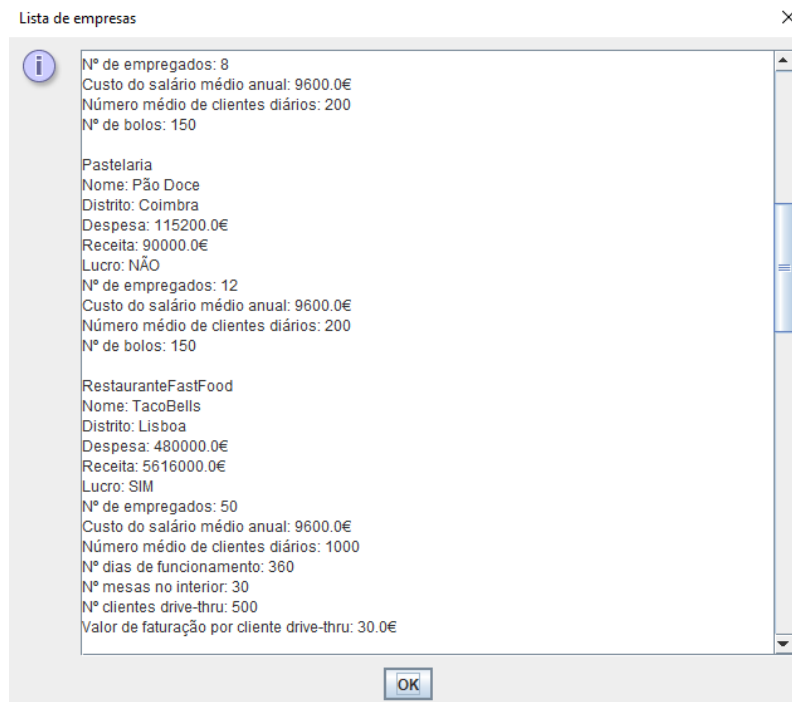


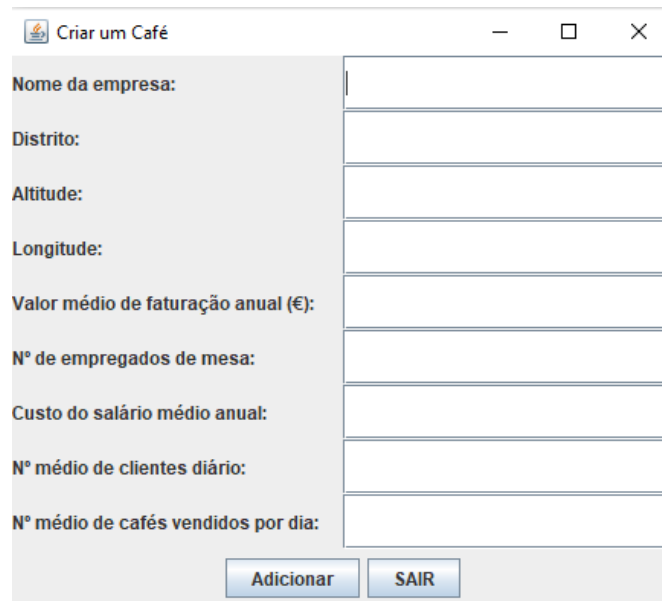
Imagem 2- Interface gráfica da ação "Listar Empresas"

Nesta funcionalidade, pretende-se listar todas as empresas registadas atualmente no sistema, assim como, as características específicas de cada uma.

Para a implementação desta funcionalidade utilizamos o método `listarEmpresas()` da classe `GestorEmpresas` que percorre a lista de empresas e chama o método `toString()` de cada empresa. Este método foi implementado em cada classe e tira partido da relação de herança.

Caso não existam empresas registadas no sistema, é emitida uma mensagem de erro "Não existem empresas registadas no sistema.".

CRIAR EMPRESA



Nome da empresa:	<input type="text"/>
Distrito:	<input type="text"/>
Altitude:	<input type="text"/>
Longitude:	<input type="text"/>
Valor médio de faturação anual (€):	<input type="text"/>
Nº de empregados de mesa:	<input type="text"/>
Custo do salário médio anual:	<input type="text"/>
Nº médio de clientes diário:	<input type="text"/>
Nº médio de cafés vendidos por dia:	<input type="text"/>

Adicionar SAIR

Imagem 3 - Interface gráfica da ação "Criar café"

Para criar a empresa é apresentada uma interface ao utilizador onde tem de preencher obrigatoriamente todas as características da empresa.

Como dependendo da categoria, a empresa apresenta diferentes características, foi criada uma classe `GuiCriarEmpresa` que contém as labels, campos de texto e botões que todas as categorias apresentam em comum. Posteriormente, foram criadas classes GUI para cada categoria de empresa que herdaram os atributos da classe `GuiCriarEmpresa`.

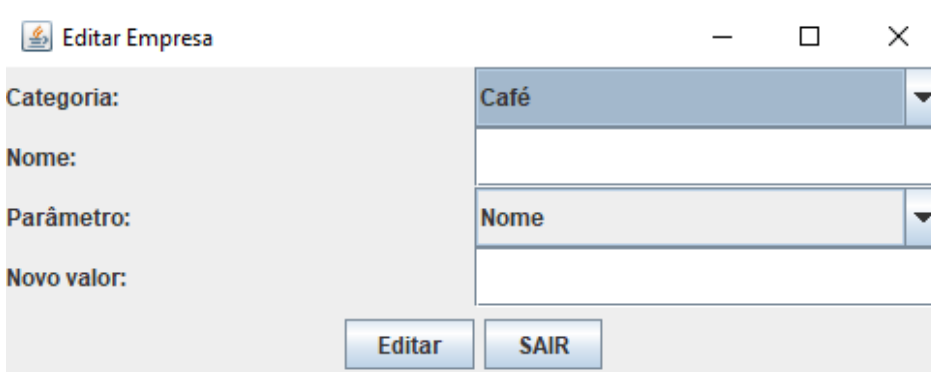
O comportamento do programa segue esta ordem:

1. Considera-se que o utilizador tem de preencher todos os dados pedidos na criação da empresa. Posto isto, primeiramente é verificado se todos os parâmetros estão preenchidos. Caso contrário, é emitida uma mensagem de erro "Todos os parâmetros são de preenchimento obrigatório." e a empresa não é criada.

2. Seguidamente, é chamado o método `verificaNome(String, String)` da classe `GestorEmpresas` e verifica se já exista alguma empresa com a mesma categoria e com o mesmo nome. Devolve *false* caso exista e *true* caso não exista. Considera-se que não é permitido que duas empresas da mesma categoria tenham o mesmo nome. Posto isto, é necessário um cuidado extra quando o utilizador cria uma empresa.
3. Caso exista uma empresa da mesma categoria com o mesmo nome, é emitido um erro "Não é possível a utilização do nome inserido." e a empresa não é criada.
4. Por último, todos os dados preenchidos pelo utilizador que sejam do tipo *int* ou *double* são convertidos. Caso não seja possível a conversão, é emitida uma mensagem de erro "Valores inválidos." e a empresa não é criada.
5. Se a empresa foi criada, é emitida uma mensagem de sucesso.

Nota: Para qualquer outra exceção não especificada anteriormente, é emitido o "Erro na criação da empresa." e a empresa não é criada.

EDITAR EMPRESA



The image shows a Java Swing window titled "Editar Empresa". It has a standard title bar with minimize, maximize, and close buttons. The window is divided into two main sections. On the left, there are four labels: "Categoria:", "Nome:", "Parâmetro:", and "Novo valor:". On the right, there are four corresponding input fields. The "Categoria:" field is a dropdown menu currently showing "Café". The "Nome:" field is a text box that is currently empty. The "Parâmetro:" field is a dropdown menu currently showing "Nome". The "Novo valor:" field is a text box that is currently empty. At the bottom of the window, there are two buttons: "Editar" and "SAIR".

Imagem 4- Interface gráfica da ação "Editar Empresa"

Para editar a empresa é apresentada uma interface ao utilizador onde pode escolher a categoria da empresa, o nome, o parâmetro que quer alterar e o novo valor para esse mesmo parâmetro. Sublinha-se que neste programa permitimos que existam duas empresas de categorias diferentes com o mesmo nome. Logo, a categoria é utilizada meramente para saber qual é a empresa que o utilizador quer editar.

O comportamento do programa segue esta ordem:

1. É chamado o método `devolveEmp(String, String)` da classe `GestorEmpresas` que devolve o objeto `Empresa` que tem a o nome e a categoria dadas anteriormente. Realça-se que esta funcionalidade é *non case-sensitive*.
2. Caso não exista nenhuma empresa nestas condições, devolvemos o erro "*A empresa não está registada no sistema.*" e a empresa não é editada.
3. Caso exista uma empresa nestas condições, guardamos o objeto numa variável.
4. Se o parâmetro a alterar corresponder a um atributo do tipo `double`, é necessário fazer a conversão para do tipo `String` para `double`. Caso não seja possível, emitimos a mensagem de erro "Valores inválidos.", a célula do novo valor é limpa e a empresa não é editada.
5. Dependendo do tipo do parâmetro que se pretende editar (`double` ou `String`) é chamado o método `editarDouble(String, double)` ou `editarString(String, String)` que está definido dentro de cada classe tirando partido do polimorfismo. Estes dois métodos devolvem `true` se o parâmetro é um atributo do objeto e foi editado com recurso aos métodos `set` de cada classe. Caso contrário, é devolvido `false`.
6. Caso esse parâmetro não seja um atributo da empresa, é devolvido o erro "Não é possível editar esse parâmetro na empresa." e a empresa não é editada.
7. Por último, se o parâmetro da empresa foi editado com sucesso, devolvemos a mensagem "Empresa editada com sucesso."

Nota: Para qualquer outra exceção não especificada anteriormente, é emitido o erro "Erro a editar a empresa!" e a empresa não é editada.

APAGAR EMPRESA



Imagem 5- Interface gráfica da ação "Apagar Empresa"

Para apagar a empresa é apresentada uma interface ao utilizador onde pode escolher a categoria da empresa e o nome. O pedido ao utilizador da categoria tem os mesmos motivos que na alínea imediatamente anterior.

O comportamento do programa segue esta ordem:

1. É chamado o método `apagarEmpresa(String, String)` da classe `GestorEmpresas` que devolve `true` caso tenha encontrado uma empresa na lista de empresas com a categoria e o nome indicados pelo utilizador e a tenha removido da lista. Realça-se que esta funcionalidade é *non case-sensitive*.
2. Se os dados indicados não corresponderem a nenhuma empresa presente na lista é devolvido o erro *"A empresa não está registada no sistema."*
3. Caso contrário, é devolvida a mensagem de sucesso *"Empresa apagada com sucesso."*

Nota: Para qualquer outra exceção não especificada anteriormente, é emitido o erro "Erro a apagar a empresa!" e a empresa não é apagada.

LISTAR OS DOIS MAIORES RESTAURANTES

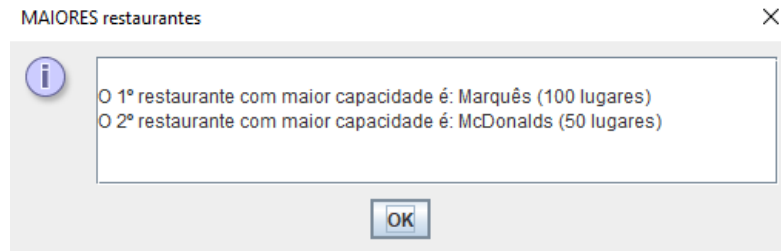


Imagem 6 - Interface gráfica da ação "Listar os 2 maiores restaurantes"

Nesta funcionalidade, o pretendido é saber quais são os dois restaurantes com maior capacidade de clientes por dia, ou seja, maior número de mesas. Os restaurantes locais têm mesas interiores e mesas na esplanada. No caso dos restaurantes de fast-food só têm mesas interiores.

O comportamento do programa segue esta ordem:

1. É chamado o método `apresentarMaioresRestaurantes()` da classe `GestorEmpresas` que devolve uma string com informação sobre os dois maiores restaurantes. Esta string que será posteriormente mostrada na *message dialog box*.
2. Dentro do método, é primeiramente chamado o método `apresentarMaiorRestaurante(Empresa emp)` da mesma classe. Este método devolve o objeto empresa que tem a maior capacidade se `emp=null` e o segundo maior restaurante se `emp=restaurante` com maior capacidade.
3. Dentro do método `apresentarMaiorRestaurante(Empresa emp)` adotou-se o seguinte comportamento:
 - 3.1. Inicializa-se uma variável auxiliar capacidade a -1 e a `maiorEmp` a null.
 - 3.2. Percorre-se a lista de empresas e utiliza-se o método `devolveCapacidade()` que devolve o número de mesas caso a empresa seja um restaurante e caso contrário, devolve -1.
 - 3.3. Caso a empresa seja diferente da empresa dada como argumento e tenha maior capacidade que a última maior capacidade registada, o valor da capacidade e de `maiorEmp` são atualizados.

4. Caso não existam restaurantes na lista de empresas, o método `apresentarMaioresRestaurantes()` devolve "0" e é devolvida a mensagem de erro "A empresa não está registada no sistema."
5. Caso contrário, o método `apresentarMaioresRestaurantes()` volta a chamar o método `apresentarMaiorRestaurante(Empresa emp)` mas agora passando como argumento a empresa com maior capacidade apurada imediatamente antes.
6. Caso exista apenas um restaurante na lista de empresas, é devolvida apenas a informação do restaurante (único) com maior capacidade.
7. Caso contrário, é enviada a string com as informações dos dois maiores restaurantes.

Nota: O método `devolveCapacidade()` volta a ser redefinido na classe `Restaurante` e devolve o número de mesas interiores. Logo, não é preciso redefini-lo na classe `RestauranteFastFood`. No entanto, na classe `RestauranteLocal` é chamado o método definido na classe pai e adicionado o número de mesas de esplanada. Desta forma tiramos máximo partido das relações de herança e do polimorfismo.

RELATÓRIO EMPRESA

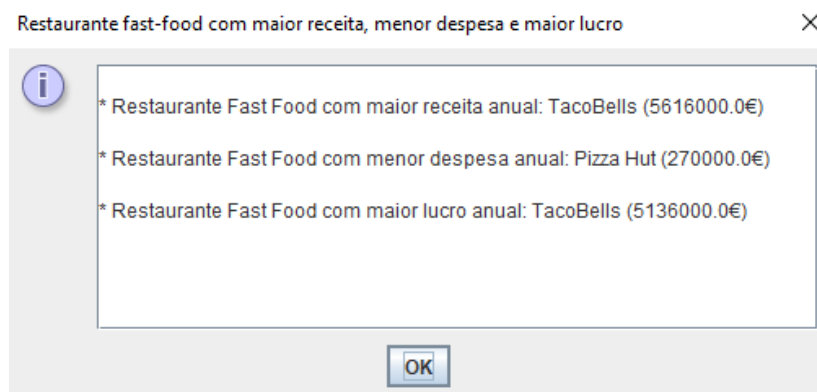


Imagem 7- Interface gráfica da ação "Relatório restaurante fast-food"

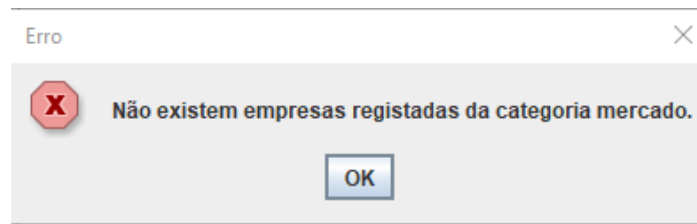


Imagem 8- Interface gráfica de um erro originado pela ação "Relatório mercado"

Nesta funcionalidade, o pretendido é saber qual a empresa, de uma determinada categoria, com maior receita anual, menor despesa anual e maior lucro anual. Para a implementação desta funcionalidade, é chamado o método `relatorio(String)` que recebe como argumento a categoria e devolve uma `String` com as informações pretendidas.

O comportamento do programa segue esta ordem:

1. O método `relatório (String)` definido na classe `GestorEmpresas` chama os métodos `apresentaMaiorLucro(String)`, `apresentaMenorDespesa(String)` e `apresentaMaiorReceita(String)` que calculam, respetivamente, a empresa da categoria com maior lucro anual, menor despesa anual e maior receita anual.
2. Estes três métodos comportam-se de forma similar. O comportamento destes métodos segue esta ordem:
 - 2.1. Percorremos a lista de empresas e caso a empresa seja da categoria desejada, analisamos se é a primeira vez que aparece uma empresa desta categoria.
 - 2.2. Se for a primeira vez, inicializamos a variável auxiliar com a receita/despesa/lucro da empresa e guardamos o objeto empresa também numa variável auxiliar.
 - 2.3. Caso não seja a primeira vez, verificamos se a receita/despesa/lucro é maior/menor (no caso da despesa) que o valor guardado na variável auxiliar.
 - 2.3.1. O cálculo da receita e despesa anual da empresa é feita com recurso aos métodos `calculaReceita()` e `calculaDespesa()` que estão definidos em cada classe ou na classe pai. O método `calculaLucro()` está apenas definido na classe `Empresa`, uma vez, que esse cálculo é apenas a diferença da receita com o lucro.
 - 2.4. Se tal se verificar, as variáveis auxiliares são atualizadas.
 - 2.5. É devolvida uma string com informações sobre a empresa com maior receita/lucro ou menor despesa.

3. Caso não exista nenhuma empresa da categoria desejada, é devolvida a mensagem de erro “Não existem empresas registadas da categoria x.”.
4. Se existirem, é mostrada a string com as informações desejadas.

Conclusão

Ao longo da realização deste projeto, desenvolveram-se as capacidades de programação e raciocínio lógico, e foi ainda adquirida prática e competência no que toca a usar os conceitos de Java e de Programação Orientada a Objetos, tais como classes, herança, polimorfismo, classes abstratas, ficheiros, diagramas de classes UML, utilização do JavaDoc, interfaces gráficas, entre outros.

Considera-se que o projeto foi realizado com sucesso, pois todas as funcionalidades estão a funcionar de acordo com o pedido, e usando os conceitos pedidos. As únicas áreas de melhoria seriam no aspeto de proteger o código, de forma a não deixar o utilizador inserir certos tipos de dados, que acabariam por estar errados. Mas, de uma forma geral, o projeto apresenta todos os aspetos fundamentais apresentados no enunciado.