UNIVERSIDADE D
COIMBRA

StarThrive

Gestor de Empresas

2022/2023

Ana Carolina Morais 2021222056
Fernanda Margarida Fernandes 2021216620

Programação Orientada aos Objetos

Índice

Introdução

Este projeto tem como principal objetivo adquirir sensibilidade para as questões fundamentais da programação orientada aos objetos, em particular, UML, polimorfismo, ficheiros de texto e de objetos, herança e, por último de GUI.

Como tal, os conceitos fundamentais enumerados acima, vão ser aplicados através da elaboração de uma aplicação que permite realizar operações importantes na gestão de várias empresas.

Para o desenvolvimento do código, utilizámos o *IntelliJ IDEA 2022.2 (Edu)*. Através do mesmo garantimos que todo o código cumpre os requisitos associados ao *JAVA*.

Da mesma forma que garantimos que o código cumpre os requisitos, ao implementar as bibliotecas necessárias, estamos a contribuir para o mesmo.

Classes e heranças

Antes de mais, iremos enunciar a forma como decidimos estruturar o nosso projeto. Começámos, primeiro, por criar a classe abstrata "Empresa" que implementa a interface *Serializable*, que irá ser útil, mais tarde, para o tratamento do ficheiro de objetos, é abstrata pois irá conter métodos abstratos. A partir desta, criámos mais duas classes a classe abstrata "Mercearia" e a classe "Restauração", que estendem da classe "Empresa", numa relação de herança (Pai-filho).

Uma vez que as empresas da categoria "Mercearia" podem ser frutaria ou mercado, optámos por criar uma classe para cada subtipo, ou seja, uma classe "Frutaria" e uma classe "Mercado", que estendem da classe "Mercearia".

Dado que as empresas de categoria de restauração podem ser café, pastelaria ou restaurante, também criámos uma classe para cada subtipo, estendendo também da classe "Restauração". Como o enunciado também nos diz, os restaurantes podem ser locais ou fast-food, daí criarmos uma classe "Local" e outra "FastFood", ambas estendendo da classe abstrata "Restaurante".

Após a criação destas classes bases, elaborámos mais uma classe, "GerirEmpresas", cuja função é ter uma lista de empresas com as diferentes operações sobre elas, esta classe também implementa a interface *Serializable*.

Como um dos atributos da classe "Empresa" é a localização, achámos, por bem, criar uma classe "GPS", uma vez que existem diferentes tipos de localização, ao qual escolhemos o sistema longitude/latitude.

Relativamente aos ficheiros, uma vez que é necessário construir um ficheiro de texto para uma primeira leitura, optámos por criar uma classe, "FicheiroTexto", precisamente para essa função. Como também é necessário, posteriormente, ler e guardar num ficheiro de objetos, também criámos uma classe, "FicheiroObjetos", de modo a atingir esse mesmo objetivo.

Para garantir a validação de todos os dados a tratar, em vez de repetirmos código ao garantir essa verificação, enunciámos uma classe," VerificaDados", cujos métodos se aplicam ao tipo de dados que estamos a tratar.

Relativamente ao GUI, concebemos três classes, a classe "Interface", a classe "JanelaCategoria" e classe "JanelaEmpresa". A primeira classe trata apenas de apresentar o menu, ou seja, apresenta o tipo de opções que se podem realizar, ao escolhermos uma dessas opções, tanto pode abrir a janela que contém as várias operações que se podem realizar com as empresas, que são: listar, criar, editar ou apagar, ou pode abrir a janela que contém as várias operações que se podem realizar com as categorias, que são: apresentar para cada tipo de empresa, a empresa com maior receita anual, com menor despesa, com maior lucro e ainda apresentar as duas empresas do tipo restauração com maior capacidade de clientes diário.

Por último, mas não menos importante, temos a classe "Main", que é o executável, contém o código que irá correr.

Atributos e métodos

Relativamente aos atributos, uns são fácies de aceder a partir do enunciado, e, portanto, é de fácil escrita na sua respetiva classe. Mas, ao ler o enunciado, existem alguns casos especiais, como é o caso da repetição de atributos em diferentes classes e, por isso, em vez de repetirmos o mesmo atributo em diferentes classes, decidimos colocar só uma vez o atributo na classe "pai", uma vez que os "filhos" também herdam esse mesmo atributo, como por exemplo: a frutaria é caraterizada pelo custo anual de limpeza do estabelecimento e o mercado também contém esse mesmo atributo, por isso decidimos colocá-lo na classe "Mercearia". Outro exemplo: o café é caraterizado pelo número médio de clientes diário, também o é a pastelaria e ainda o restaurante, daí esse atributo ser colocado na classe "Restauração", em vez de o repetirmos nas três classes.

Nas classes "Empresa", "Mercearia", "Restauração" e "Restaurante" os atributos são "protected" para indicar que os métodos e variáveis podem ser acedidos a partir de subclasses, mas não das restantes classes.

Referindo agora os métodos, existem alguns métodos abstratos nas classes abstratas referidas acima, uma vez que ao recorrermos ao polimorfismo nesses métodos, essas classes não tenham que devolver ou apresentar qualquer informação, desta forma garantimos que o polimorfismo ocorra nas classes pretendidas, como, por exemplo: o método "despesaAnual()" da classe abstrata "Empresa", não tem nenhuma fórmula, em comparação com as cinco classes que a contêm, e como nós queremos que sejam essas classes a calcular a sua respetiva fórmula, decidimos colocar este método abstrato nas classes abstratas.

A classe "GerirEmpresas", é a classe onde irá ocorrer a base deste projeto, que tem como atributo único a lista de empresas do tipo "ArrayList<Empresa>", é com esta lista que vamos aceder ao GUI e aos ficheiros. Os métodos principais desta classe são o inserir," insert ()", e o remover, que têm como método auxiliar o "pesquisaNome ()", de forma a garantir que não se insere empresas duplicadas. Já o método "pesquisaEmpresa ()", tem como função encontrar uma empresa na lista e a devolver, de forma a podermos aceder aos seus atributos para os editar, como iremos realizar no GUI.

É também, nesta classe, que iremos dar uso ao polimorfismo, uma vez que um dos objetivos deste projeto é apresentar para cada tipo de empresa a empresa com maior receita anual, com menor despesa anual e com maior lucro anual e, também, as duas empresas do tipo restauração com maior capacidade e, para atingirmos o mesmo, foi necessário criar um método para cada função, em que, por exemplo, no método "maiorReceita()", foram enunciadas uma variável de cada tipo de empresa e, fomos percorrer a nossa lista de empresas e para cada empresa víamos qual era o tipo e, consoante, o mesmo íamos calcular a maior receita dessa empresa e, também, passamos essa variável por argumento, de forma a fazermos a comparação da maior receita dentro da classe tipo correspondente. Utilizámos esta mesma forma de pensar para o cálculo da menor despesa e do maior lucro.

No que diz respeito ao cálculo da maior capacidade, elaborámos duas variáveis auxiliares, "aux" do tipo *'int',* e duas auxiliares, "auxNome" do tipo *'String'*, dado que o resultado irão ser duas empresas, a partir dai basta percorrer a nossa lista empresas e verificar qual a capacidade da empresa que lê, é de realçar que o método "capacidade()" só se encontra nas classes correspondentes, ou seja, na classe "Local" e na classe "FastFood", pois são as únicas classes que permite calcular a capacidade através do número de mesas, este método nas classes abstratas está abstrato, assim, no final irá ser devolvido uma *'string'* com o nome e valor da capacidade das duas empresas com maior capacidade.

Quanto aos ficheiros de texto, só contém um método de leitura, uma vez que só é necessário quando a aplicação iniciar, dado que o ficheiro de objetos ainda não foi criado. A função deste método é carregar os dados das empresas deste ficheiro para a lista. Este método tem como variável uma lista do tipo "GerirEmpresas" e outra do tipo "VerificaDados". Neste caso, a escrita diretamente no ficheiro segue a seguinte forma:

Categoria da empresa:nome,distrito,latitude,longitude, ... (consoante o número de atributos da categoria correspondente).

Ex: `CAFE:CELEIRO,COIMBRA,30000,2100,30,20,9600,18,18,180`

Tendo em conta a escrita no ficheiro, depois ao ler é de fácil compreensão o 'split' de cada linha e, como sabemos que o primeiro elemento do 'array' da linha é a categoria da empresa, basta verificar qual a categoria corresponde através de um 'switch' e a partir desse momento adicionámos cada elemento do "array" da linha a uma variável de cada 'case' da categoria correspondente ao 'case', não esquecendo de verificar cada elemento, através da variável do tipo "VerificaDados", esta verificação só se aplica ao tipo de dados 'float' e 'int'.

Uma vez a aplicação iniciada e após ter feito uma primeira leitura a partir do ficheiro de texto, vamos passar, então, a tratar os dados a partir de um ficheiro de objetos, para isso criámos dois métodos um de escrita e outro de leitura, em que, no método "escrita ()" passamos por argumento a nossa lista e vai escrevendo cada empresa no mesmo, já o método "lerFicheiroObjetos ()", que tem como variável a lista do tipo "GerirEmpresas", lê cada empresa do nosso ficheiro e insere-o na lista.

Decidimos que após a leitura do ficheiro de texto, o programa irá logo escrever no ficheiro de objetos, garantindo, assim, a criação deste ficheiro. Sempre que o utilizador realizar alguma operação, como por exemplo: apagar ou inserir uma empresa, iremos reescrever no ficheiro de objetos, para assim, garantir a sua atualização continua.

Dado que a interação com o utilizador é através de uma interface gráfica e, como já referimos, anteriormente, a estrutura que utilizámos para abordar este assunto, é suficiente afirmar, que na classe "JanelaEmpresa", vamos verificar qual a opção que o utilizador deseja, através, da seleção do botão respetivo, e, no caso de ser o botão "Voltar", a janela anterior, classe "Interface", fica visível e a janela atual fica invisível.

No caso, de se selecionar o botão "Apagar", irá aparecer uma *JList* contendo a lista de empresas e, neste caso, o utilizador seleciona a empresa que deseja apagar. Utiliza-se a mesma metodologia para o botão "Editar", só que irá aparecer ao utilizador os atributos todos da empresa que selecionou anteriormente e, ele decide quais, ou qual quer editar, também estes verificados através dos métodos da classe "VerificaDados".

Já no caso do botão "Criar", aparece uma caixa onde o utilizador pode selecionar qual a categoria da empresa a inserir, após a seleção da categoria irá aparecer os atributos para o utilizador inserir os respetivos dados, não esquecendo,

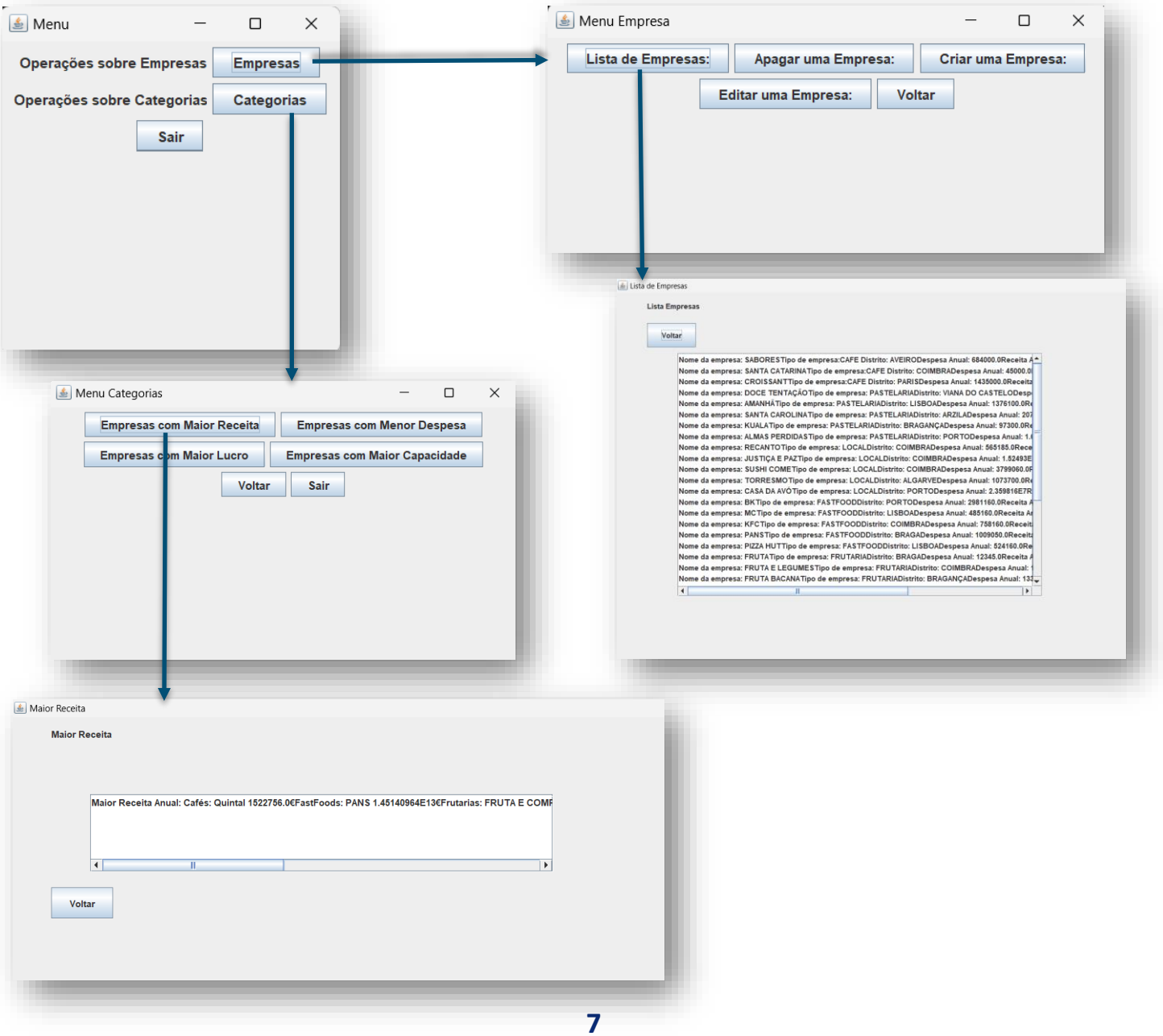
antes, de serem verificados através dos métodos da classe "VerificaDados" que se adequam ao *GUI*.

Estes métodos, métodos da classe "VerificaDados", diferenciam- se dos métodos que aplicámos aos ficheiros de textos, na medida em que, passamos por argumento o campo que contém a *string* com o que o utilizador escreve e, também uma string com o que corresponde essa *string*. Nestes métodos fazemos uma verificação dos dados através de um ciclo, que só permite avançar nos atributos que o utilizador deseja inserir/editar se o valor for válido.

À medida que as opções vão sendo selecionadas, o programa irá também imprimir na consola o resultado dessas operações, ou seja, caso o utilizador decida listar todas as empresas, essa lista irá aparecer numa nova janela e, também irá aparecer na consola do *Intellij IDEA*.

Da mesma forma que estabelecemos um seguimento de ações para a "JanelaEmpresa", aplicámos à classe "JanelaCategorias".

A seguir, colocámos uma representação do nosso *GUI*.



NOTA:

Relativamente ao UML, no caso das classes do GUI, não colocámos a classe privada que implementa o *ActionListener*, mas queremos, apenas, deixar claro que sabemos que não está no UML, mas achámos por bem colocar esta nota.

Conclusão

Em suma, após a criação deste projeto podemos afirmar que conseguimos adquirir melhor sensibilidade a respeito da programação orientada aos objetos.

Apesar de termos encontrado alguns percalços, sentimos que ultrapassámos as nossas dificuldades. Após uma análise dos resultados, podemos garantir que o objetivo foi comprido.

Referências

- Parte 4.3 – Herança e Polimorfismo (Slides Aula)
- Parte 5 – Classes Abstratas-Interfaces (Slides Aula)
- Parte 6 – UML (Slides Aula)
- Parte 7 – Ficheiros e Exceções (Slides Aula)
- Parte 8 – GUI (Slides Aula)
- Exercício nº2 (Trabalho POO)
- Ficha 07 -Ficheiros e Exceções (Ficha Aula)
- Ficha 08 – GUI (Ficha Aula)