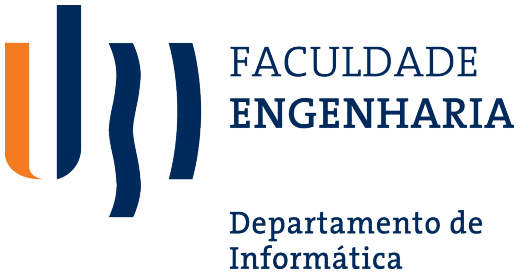
Universidade da Beira Interior
Departamento de Informática



Trabalho Prático de Programação Orientada a Objetos - 2021/2022

Trabalho Elaborado Por:

Nº 45600 - André Pais
Nº 45601 - Miguel Pais
Nº 45609 - Miguel Pereira
Nº 45894 - Guilherme Nunes

Orientador:

Professora Doutora Maria Paula Prata de Sousa

19 de dezembro de 2021

Conteúdo

Capítulo

1

Introdução

Este trabalho foi proposto no âmbito da Unidade Curricular de Programação Orientada a Objetos, lecionada pela Professsora Doutora Maria Paula Prata de Sousa, pretende-se de que sirva como elemento de avaliação para testar os conhecimentos dos alunos sobre a linguagem de programação Java e sobre as linguagens de programação orientadas a objetos.

1.1 Conceito

O objetivo deste trabalho era criar uma plataforma para uma escola de formação profissional em que fosse possível administrar e obter informações acerca da mesma, entre as funcionalidades do programa é possível atribuir cursos aos alunos, consultar quanto cada docente recebe por mês, ver qual o aluno com melhor média por curso, consultar o número de horas por curso, entre outras funcionalidades.

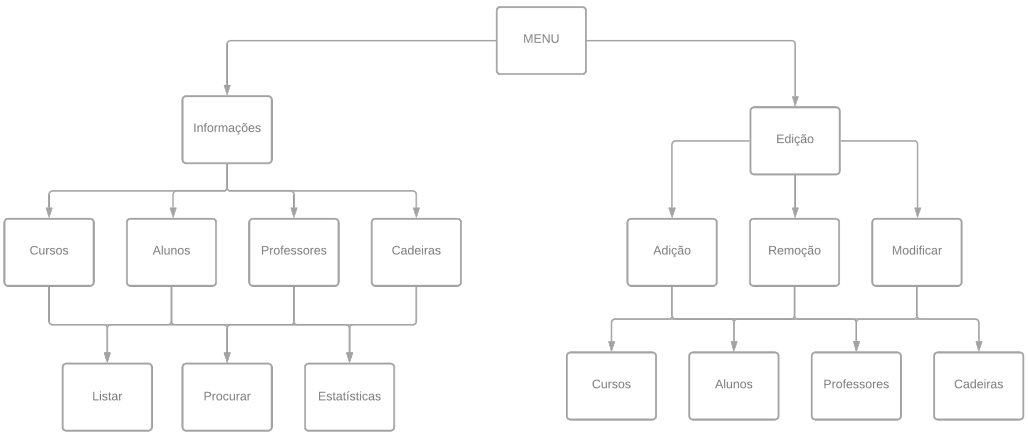
Em questões de organização a escola profissional encontra-se organizada do seguinte modo, a escola tem quatro cursos sendo eles "Redes", "Informática Web", "Engenharia Informática", "Segurança Informática", cada curso tem um número diferente de horas que está distribuido pelas cadeiras (Unidades Curriculares) do mesmo e nenhum curso excede as 5 cadeiras. As cadeiras podem ser lecionadas por mais que um professor e um professor pode lecionar mais que uma cadeira. As notas às respetivas cadeiras valem a mesma percentagem no cálculo da média do aluno. Na execução do código as relações procedem-se de um modo parecido no entanto um pouco diferente, os professores "estão dentro"das cadeiras, as cadeiras "dentro"dos cursos e os cursos nos alunos, podendo assim relacioná-los facilmente.

1.2 Abordagem Inical

A primeira abordagem ao trabalho (a primeira ideia e tentativa de concessão do projeto) foi criar uma superclasse a que se chamaria "escola"e se seguida criar subclasses para os alunos, as cadeiras, os professores e os cursos. No entanto esta ideia não foi avante, a ideia que acabou por ser implementada foi uma abordagem em que todas as classes são equivalentes,isto é, não há super nem subclasses.

1.3 Esquematização do trabalho

Para um maior entendimento do trabalho foi criado um esquema de opções do menu inicial para que as funções do programa se interligassem e para que em termos de programação se conseguisse criar as associações e que quando algo fosse alterado as dependências permitissem perceber todos os sitios em que seriam necessárias alterações.



Capítulo

2

Trechos de Código

Embora o trabalho funcione muito na base de classes e de funções básicas relacionas com a utilização das classes em java como por exemplo o "equals", "getters", "seters", "clone"existerm algumas funções com um grau de complexidade mais elevado, ste caítulo serve para explicar algumas delas.

2.1 Função BUSCARFILES

Esta função vai buscar a informação que está nos ficheiros ("Files") e transfere-a para os "Arraylists"respetivos, desta forma podemos manipular a informação que está presente nos ficheiros da forma que entender-mos.

2.2 Função GUARDARFILES

Após a manipulação da informação presente nos Arraylists, se necessário, substitui a informação antiga nos Files pela sua versão mais atualizada.

2.3 Função REPOR

Esta função tem como objetivo fazer uma modificação/remoção "segura"do elemento em questão, isto é, sem que danifique o resto da informação à qual está associada. Por exemplo, se removermos uma cadeira ela é consequentemente retirada da lista de cadeiras, da lista de cadeiras do curso e de todas as outras listas em que se possa encontrar.

```
ArrayList<Cadeira> cadeirastemp = cadeiras;
```

```
        for (int i = 0; i < cadeirastemp.size(); i++) {
            ArrayList<Professor> tempprof = cadeirastemp.get(i).
                getProfessores();
            ArrayList<Integer> exprof = new ArrayList<Integer>();
            for (int k = 0; k < professores.size(); k++) {
                int ex = 0;
                for (int j = 0; j < tempprof.size(); j++) {
                    if (tempprof.get(j).equals(professores.get(k))) {
                        ex = ex + 1;
                    }
                }
                exprof.add(ex);
            }
            ArrayList<Professor> tempprof2 = new ArrayList<Professor>();
            for (int j = 0; j < professores.size(); j++) {
                if (exprof.get(j) != 0) {
                    tempprof2.add(professores.get(j));
                }
            }
            int cadeiratemounao = 0;
            for (int j = 0; j < exprof.size(); j++) {
                if (exprof.get(j) != 0) {
                    cadeiratemounao = 1;
                }
            }
            if (cadeiratemounao == 1) {
                cadeirastemp.get(i).setProfessores(tempprof2);
            }
        }
        cadeiras = cadeirastemp;
```

A função começa por criar um Arraylist temporário que irá retirar a informação do array original para puder modificá-la, as alterações nunca acontecem no array original, o array cópia vai sofrer as alterações necessárias e após a conclusão do processo irá substituir aquilo que se encontra no array original por aquilo que foi modificado no array cópia.

A segunda parte do código pode ser dividida em duas ações, numa primeira instância vamos correr o array professores e vamos compará-lo com o array de professores que pertence a uma determinada cadeira (esta ação irá acontecer de forma sistemática até ter percorido todas as cadeiras, isto acontece por conta do ciclo for, por isso vai correr para todos os professores de todas as cadeiras) caso a informação nos dois arrays seja coincidente iremos alterar o que está no array de professores cópia.

Num segundo momento criamos um aray list temporário para os professores das cadeiras. O programa irá fazer uma comparação com a lista de professores da escola e da lista de professores da cadeira, se o professor da cadeira existir na lista de professores total, este será adicionado à lista de professores temporária essa lista irá então substituir antiga lista de professores de cadeira.

Este procedimento irá repetir-se para os cursos e para os alunos, por conta da hierárquia estabelecida na origem do programa este irá começar pela atualização das cadeiras, de seguida dos cursos e só por fim dos alunos.

2.4 Função STATS_P

Esta funcão tem como finalidade de indicar qual o professor com maior número de cadeiras associadas, a função começa por criar um array list temporário para guardar o número de cadeiras de cada professor, de seguida percorre o arraylist dos professores para determinar o número de cadeiras de cada um. A função está munida de uma estrutura composta por 2 ciclos for que vão correr o arraylist das cadeiras e ver quais os professores presentes em cada cadeiras e ver se o professor está presente nessa cadeira, se o resultado for positivo irá adicionar mais um ao número de cadeiras do respetivo professor, a função faz isto para todos os professores.

A função irá então comprar a quantidade de cadeiras que cada professor tem, irá ser criada uma variàvel temporária para guardar o maior valor, respetivo ao professor que tem o maior número de cadeiras, de seguida a função procede a fazer comparações entre os valores de quantidade de cadeiras de cada professor, a variavel temporária vai ser iniciada a zeros e sempre que aparecer um número maior nas comparações vai ser substituido pelo novo valor maior.

Capítulo

3

Problemas e Dificuldades

3.1 Dificuldades

Em geral, a maior dificuldade associada a este trabalho foi a estruturação e conceção de alguns trechos de código e funções em específico, aquelas funções que podem ser consideradas mais complexas, não em termos de complexidade da linguagem de programação mas sim em questões de lógica de programação, estas foram explicadas no capítulo anterior.

3.2 Problemas

Durante a elaboração deste trabalho houve um problema em específico que foi particularmente difícil de resolver. Na impletementação da ferramenta de modificação reparamos que quando removemos ou modificamos um qualquer atributo de um objeto, essa modificação não é transmitida às outras instâncias desse objeto.

3.3 Resolução dos Problemas

Para a resolução deste problema criamos a função que foi apresentada na secção 2.3. A ideia inicial foi criar uma função que percorresse a todas as listas à procura do objeto que está a ser removido ou alterado até encontrar todas as suas instâncias, depressa percebemos que não era a melhor abordagem porque precisávamos de algo mais generalizado que pudesse ser utilizado em qualquer parte do código.

Capítulo

4

Conclusão

4.1 Possíveis Melhorias

Considerando o que foi pedido no enunciado do trabalho, o programa cumpre com todos os requisitos necessários porém este ainda tem espaço para melhorias. Com conhecimentos mais avançados na área da programação orientada a objetos e na linguagem de programação Java é possível optimizar o programa para que este funcione de forma mais eficiente asism demorando menos tempo a entregar os valores desejados (ainda que este já esteja bastante rápido na entrega dos outputs) e a que consuma menos poder de computação durante a sua execução.

4.2 Conhecimentos Retidos

Este trabalho contribuiu para solidificação dos conhecimentos dos alunos que foram falados nas aulas teóricas e trabalhados nas aulas práticas, foi uma oportunidade de trabalhar com objetos de uma forma mais desafiante do que aquela que foi trabalhada nas aulas devido À relação entre as diferentes classes do projeto.