# Universidade da Beira Interior


Departamento de Informática
Informática Web

# Projeto Final

Alunos:
Bruno Salvado, Nº35251
João Nobre, Nº38464
Rita Correia, Nº38254

Unidade Curricular: Programação Orientada a Objetos

Professora: Paula Prata

Dezembro
2017

# Conteúdo

## Acrónimos

**UBI** Universidade da Beira Interior

**UC** Unidade Curricular

**POO** Programação Orientada a Objetos

# Lista de Figuras

# Capítulo 1

# Introdução

## 1.1 Contexto e Motivação

*Quando uma equipa pensa, reflete e aceita o novo como algo construtivo, é mais fácil assumir novos conhecimentos.*[1]

Inicío este relatório com a citação apresentada no parágrafo anterior, pois penso que esta transmite uma forte ideia do enquadramento e objetivo deste trabalho, bem como a importância do trabalho em equipa. Este projeto é dirigido para os alunos do segundo ano, tanto do curso de Informática Web como do curso de Engenharia Informática, da Universidade da Beira Interior (UBI), que frequentam a Unidade Curricular (UC) de Programação Orientada a Objetos (POO).

Sendo hoje em dia a Informática, uma das áreas de estudo mais abrangentes e com maiores prespetivas de crescimento, é cada vez mais importante, principalmente para estudantes de cursos desta natureza, o procurar saber mais, toda a informação é pouca num mundo que evolui a um ritmo cada vez mais acelarado e desenvolvido, seja através deste tipo de trabalhos, como por iniciativa e pesquisa própria, para que mais tarde seja possível alcançar de forma mais rápida, exigente e eficaz certos objetivos profissionais, multiplicando ou triplicando os resultados até à data existentes.

## 1.2 Objetivo

O objetivo deste projeto é promover a solidificação de conhecimentos da UC em questão e a consequente aprendizagem de um conjunto de conceitos, metedologias e ferramentas no domínio da programação orientada a objetos, utilizadas neste caso, em Java.

[1] Citação de Helyane Dianno

# Capítulo 2

# Escola de teatro

## 2.1 Resumo e funcionamento

### 2.1.1 Resumo da aplicação

A aplicação realizada consiste na gestão de uma escola de teatro, que tem diversos sub-menus, com diversas funcionalidades distintas e opções.

No primeiro sub-menu, é possível fazer a gestão dos alunos: criar um novo aluno, modificar um aluno, listar todos os alunos e lançar uma nota para cada disciplina que o aluno tem. Cada aluno tem um id distinto, um nome, um email, um nível, um curso e as consequentes disciplinas.

À semelhança do sub-menu da gestão dos alunos, existe também um sub-menu para gerir os professores, onde também é possível criar, listar e modificar um professor. Cada professor tem também um id próprio, um nome, um email, um número de telemovel, um nib, um salário, um curso e uma disciplina onde vai lecionar.

De seguida, existe um sub-menu para gerir os cursos exixtentes na escola. Sendo que a aplicação já vem com 3 cursos pré-definidos, é possível nesse sub-menu, criar novos cursos, modificá-los, listá-los ou eliminá-los. Cada curso é também definido por um id, um nome, um nível e um conjunto de disciplinas.

Foi também implementada, no menu principal, uma opção para procurar um utilizador (aluno ou professor) por id, e uma outra para eliminar um utilizador também pelo seu id.

Por último, existe o sub-menu das estatísticas, onde existe uma opção para ver a média do melhor aluno de cada curso, e a melhor nota de uma determinada disciplina.

### 2.1.2 Funcionamento da aplicação

A aplicação desenvolvida está estruturada em 10 classes.

A primeira classe, com o nome de IDGenerator, é uma classe que serve para criar automaticamente o id dos alunos, professores, cursos e das unidades curriculares.Ao fazermos o gerador de id numa classe à parte ficamos com um codigo mais organizado, menos repetições de código,constantes e atributos,tendo uma classe exclusivamente focada nisso.

A segunda classe é a classe Ler, que foi usada em vários trabalhos práticos, e foi usada para detetar erros de input do utilizador ao escrever para uma variável.

A terceira classe chama-se Aluno, e é uma classe que vai criar uma variável nova para os alunos.A variável alunos vai conter o nome, email, curso, uma lista de notas, um nível(ano académico) e a média do aluno. A classe do aluno extende-se à classe do utilizador, que iremos explicar mais tarde. Para criar a lista de notas é usada uma List<>. Esta é uma interface abstrata e é usada, por exemplo, quando são utilizadas superclasses. Uma List<> não necessita de ser instânciada pois é um conceito abstrato. Quando uma List<> é instânciada esta passa a ser uma ArrayList. A vantagem da implementação de uma List<> é que esta pode mudar sem afetar o resto do código. Na classe aluno foram implementados os getters e os setters, assim também como o método toString.

A quarta classe tem o nome de Curso, que vai criar uma variável nova para o curso. A variável Curso contém o nome do curso, uma ArrayList de disciplinas e o id do curso.Para criar uma lista de disciplinas foi usada uma ArrayList, porque é dinâmica, facilitando todos os casos pois permite a escalabilidade do programa em termos de memória. Foram também implementados os setters, getters e o método toString na classe Curso.

A quinta classe chama-se Professor, e vai criar uma variável nova para os professores. A variável Professor contém a disciplina que ele vai ensinar, o seu salário, o seu nib, o email e o número de telemóvel. A classe Professor também se extende à classe Utilizador. Foram implementados os setters, getters e o método toString à classe.

A sexta classe, Utilizador, é definida como uma classe abstrata pois como não existem objetos do tipo utilizador, esta não necessita de ser instânciada, visto que os utilizadores é o conjunto de alunos e professores que partilham atributos e métodos. A classe contém os setters e os getters.

A sétima classe, com o nome Nota, cria uma variável nova para a nota de um aluno. A variável contém uma disciplina e um valor(nota). A classe também contém os setters, getters e um método toString.

A oitava classe chama-se Disciplina, que cria uma variável Disciplina. Essa variável vai conter o nome, o id e o nível da disciplina. A classe também contém os getters/setters e um método toString.

A nona classe chamada Manager é uma *singleton*. Este tipo de classes é

dos padrões de desenho de software mais usado, representando classes com uma só instância. Como vai ser apenas uma instância a tratar da gestão do programa, não vão haver conflitos. Nesta classe é possivel fazer a gestão da memória de toda a aplicação, não sendo necessário aceder diretamente ao modelo. Para este efeito foi criada o método "public static synchronized Manager getInstance()", que é definido como static para ser possível chamá-lo sem o instânciar. O *synchronized* serve de proteção às *threads*. Cada vez que o método for chamado este vai devolver a instância criada do manager. Se esta não existir ele vai criá-la, por outro lado, caso esta exista, ela vai retorná-la. Foram definidas uma lista de alunos e uma lista de cursos, instânciadas como sendo uma ArrayList. Foi definido também um método Populate que vai inicializar três ArrayList que correspondem às disciplinas de três cursos diferentes estabelecidos num primeiro momento. Estas ArrayList das disciplinas vão depois adiciona-las à classe Disciplina, anteriormente referida. Este processo foi também utilizado aquando a criação dos cursos. O restante código implementado nesta classe serve para criar todos os métodos e funções necessárias que vão ser depois chamados no main, referentes à gestão dos alunos, professores e cursos. Por último foi criado um método para escrever e posteriomente ler todos estes dados de um ficheiro aí designado, porém o grupo apresentou dificuldades na implementação deste.

A décima e última classe é a Main, que é a classe que o cliente vai utilizar para executar o programa. Visualmente, num momento inicial, vai aparecer no output o Menu Principal, onde o cliente vai escolher a opção que pretende usar. Inicialmente vai ser chamada a instância que foi criada na classe manager, anteriormente referida. O grupo optou por utilizar um switch/case para cada menu e sub-menu criado, devido à facilidade e organização da implementação do código. Cada *case* executa uma função diferente, de acordo com o que o cliente pretende. A imagem abaixo representada mostra o output do menu principal do programa:

```
run:

********** GESTÃO DA ESCOLA**********
O que pretende?

0- Sair do programa
1- Gestão de alunos;
2- Gestão de Professores;
3- Gestão de cursos;
4- Procurar um utilizador pelo número;
5- Apagar um utilizador;
6- Estatísticas;
```

Figura 2.1: Menu Principal

Se o cliente optar pela opção número 0, o programa encerra. Em contrapartida, se este selecionar a opção número 1, vai ser redirecionado para o sub-menu da gestão dos alunos. Neste sub-menu, é possível criar, modificar, listar e lançar as notas para cada disciplina, de todos os alunos até à data criados, tal como mostra a figura abaixo:

```
O que pretende?
****Gestão de Alunos****

0- Voltar para o menu principal;
1- Inserir novo aluno;
2- Listar alunos;
3- Modificar um aluno;
4 - Lançar nota
1

Digite o nome do aluno: João Nobre
Digite o email do aluno: nobre@ubi.pt
Lista de Cursos:

1 - Design de cena

2 - Especialização em Artes Performativas

3 - Especialização em produção
---------------------------
Digite o id do curso: 3
Aluno inserido com sucesso!
---------------------------
```

Figura 2.2: Gestão de alunos

Similarmente à opção número 1, a opção número 2 faz a gestão dos professores que, tal como na anterior, também é possível adicionar, listar e modificar quaisquer professores adicionados. No entanto, no processo da criação de um professor, contrariamente aos alunos, é necessário introduzir dados adicionais, tais como o nib, número de telemóvel, salário e disciplina que este vai lecionar.

De seguida, é apresentada a opção que permite gerir os cursos existentes na escola de teatro, sendo possível também listar os cursos existentes, criar, modificar ou apagar um curso.Cada um deles vai ter um nome, lista de

```
O que pretende?
****Gestão de Professores****

0- Voltar para o menu principal;
1- Inserir novo professor;
2- Listar todos os professores;
3- Modificar um professor;
1

Digite o nome do professor: Paula Prata
Digite o email do professor: pprata-poo@di.ubi.pt
Digite o número de telemóvel do professor: 969420420
Digite o nib do professor: 0123456789
Defina o salário do professor: 1420
Escolha o curso em que o professor vai estar a lecionar:

Lista de Cursos:

1 - Design de cena

2 - Especialização em Artes Performativas

3 - Especialização em produção
--------------------------
Digite o id do curso: 1

Lista de disciplinas do curso 1 - Design de cena:

2001 - Desenho I(nível 1)
2002 - Teoria e História do Design de Cena I(nível 1)
2003 - Teorias da Arte Teatral I(nível 1)
2004 - História da Arte I(nível 1)
2005 - Desenho II(nível 2)
2006 - Design de Cena I(nível 2)
2007 - Teoria e História do Design de Cena II(nível 2)
2008 - Iniciação à Produção de Cena(nível 2)
2009 - Design de Cena II(nível 3)
2010 - Problemas da Arte Contemporânea(nível 3)
2011 - Escrita de Relatório(nível 3)
2012 - Escritas Dramáticas da Contemporaneidade(nível 3)
--------------------------
Digite o id da disciplina que o professor vai ensinar: 2011
Professor inserido com sucesso!
--------------------------
```

Figura 2.3: Gestão de professores

disciplinas e o nível(que pode ir de 1 a 3), sendo que este último parâmetro representa o ano em que um aluno está. Na imagem abaixo está representado um exemplo da criação de um novo curso:

```
****Gestão de Cursos****

O que pretende?

0- Voltar para o menu principal;
1- Inserir novo curso;
2- Listar Cursos;
3- Modificar um curso;
4- Apagar um curso
1

Nome do curso?
InfoWeb aplicada ao teatro
Quantas disciplinas tem o curso?
2
Nome da disciplina nº1:
POO
Nivel da disciplina nº1:
1
Nome da disciplina nº2:
Seminários
Nivel da disciplina nº2:
2
Curso criado com sucesso!
--------------------------
```

Figura 2.4: Gestão de cursos

Na opção 4 é possível procurar um utilizador(aluno ou professor) pelo seu id, definido na classe IDGenerator.

```
********** GESTÃO DA ESCOLA**********

O que pretende?

0- Sair do programa
1- Gestão de alunos;
2- Gestão de Professores;
3- Gestão de cursos;
4- Procurar um utilizador pelo número;
5- Apagar um utilizador;
6- Estatísticas;
4

Digite o numero do utilizador a procurar: 1001
Número: 1001  Nome: João Nobre  Email: nobre@ubi.pt  Curso: Especialização em produção  Nivel: 1
```

Figura 2.5: Procurar um utilizador

Na penúltima opção do menu principal, o cliente pode eliminar um utilizador pelo seu id.

```
O que pretende?

0- Sair do programa
1- Gestão de alunos;
2- Gestão de Professores;
3- Gestão de cursos;
4- Procurar um utilizador pelo número;
5- Apagar um utilizador;
6- Estatísticas;
5

Digite o numero do utilizador a apagar: 1001
Utilizador apagado com sucesso!
```

Figura 2.6: Eliminar um utilizador

Por último, o cliente pode consultar a opção número 5, referente às estatísticas. Nesta opção é possível ver a maior média de um determinado curso, assim como ver a nota mais alta dum curso, tendo em conta todas as disciplinas deste.

```
**** Estatísticas ****

O que pretende?

0- Voltar para o menu principal;
1- Ver a melhor média do curso;
2- Ver a melhor nota do curso;
```

Figura 2.7: Estatísticas

Consequentemente, nesta mesma classe, foram implementadas todas as funções necessárias para estas operações. Em todas estas funções, sempre que foi necessário utilizar um ciclo "for", foi decidido usar a implementação do "for each", que, por exemplo, para cada disciplina da List<Dsiciplinas>, a cada iteração, a Disciplina d vai ser uma instância de uma disciplina dessa lista. O uso do "for each"foi devido à facilidade de implementação e à conveniência da linguagem, pois pode não haver necessidade de alterar os valores de um Array.

# Capítulo 3

# Considerações Finais

O desenvolvimento deste projeto foi bastante importante para o grupo ganhar e solidificar conhecimentos na área de POO, em Java.

A construção desta escola de teatro fictícia permitiu-nos também ter uma leve ideia de todo o esforço, trabalho e pormenor existente, por detrás da criação de uma simples aplicação, como esta.

A desistência de 2 elementos do grupo, fez-nos perceber o quão importante é o trabalho em equipa, pois quando um grupo de pessoas está focado para atingir determinado objetivo, torna-se claramente mais facil alcançá-lo, sendo essa a maior dificuldade que encontramos.

Futuramente, numa empresa, o trabalho em equipa é uma componente crucial para o sucesso, pois assim é possível arranjar mais e melhores soluções para um determinado problema , demorando menos tempo a realizar uma determinada tarefa, e tornando o trabalho gerado mais efciente. Trabalhar em equipa cria sinergia, ou seja, existe uma associação concomitante de várias pessoas executoras de uma determinada função, que contribuem para uma ação coordenada, criando-se um somatório de esforços em prol de um mesmo fim. O efeito resultante desta sinergia tem, normalmente, um efeito bastante superior ao efeito de cada pessoa, se trabalhasse individualmente, sem um objetivo final comum previamente estabelecido. Como se costuma dizer: "O todo supera a soma das partes".

De qualquer maneira, foi possível ganhar noções bastante enriquecedoras de POO aplicado em java, e tendo em conta os fatores apresentados anteriormente, e a destabilização do grupo devido às desistências ocorridas, ficámos bastante satisfeitos com o resultado final do trabalho, embora um pouco incompleto, pois tivemos a oportunidade de aprender bastante, e de aplicar a matéria aprendida em contexto de aula.