RELATÓRIO DO PROJETO

Programação Orientada a Objetos
Engenharia Informática

• U  C •

Realizado por

Teresa Salazar 2015234237 | José Ferreira 2014192844

INTRODUÇÃO

O objetivo deste trabalho é desenvolver uma aplicação para gestão de exames. A aplicação, desenvolvida e Java, tem em conta os princípios da programação orientada a objetos.

CLASSES

Para a realização do projeto, utilizámos várias classes. Cada exame é constituído por uma disciplina (classe Disciplina), uma data (classe Data), duração (inteiro), sala (inteiro), um docente responsável (classe Docente), um conjunto de vigilantes (Arraylist de Docentes), um conjunto de funcionários não docentes de apoio à realização do exame (Arraylist de NaoDocentes) e um conjunto de Alunos e respetiva classificação (Arraylist de AlunosClassificação). Há três tipos de exame: Exame Normal, Exame Recurso e Exame Especial, cada um representado por uma classe filho do Exame.

Cada Pessoa constituída por um nome (String) e um email (String). Cada Aluno, que é uma pessoa, é constituído por um número (inteiro), um ano de matrícula (inteiro), um curso (classe Curso) e um regime (String). Cada Funcionário, que é uma pessoa, é constituído por um número mecanográfico (inteiro) e uma categoria (String). Cada Docente, que é um Funcionário, é constituído por uma área de investigação (String). Cada NaoDocente, que é um funcionário, é constituído por um cargo (String).

Cada Curso é constituído por um nome (String), uma duração (inteiro), um grau (String) e um conjunto de disciplinas (Arraylist de Disciplinas).

Cada Disciplina é constituída por um nome (String), um docente (classe Docente), um conjunto de outros docentes (Arraylist de Docentes) e um conjunto de alunos (Arraylist de alunos).

Cada AlunoDisciplina é constituída por um Aluno (classe Aluno) e pela sua classificação (inteiro).

Finalmente, cada Data é constituída por 5 números inteiros: dia, mês, ano, hora, minuto.

FICHEIROS

Todos os dados da aplicação estão guardados em ficheiros e são carregados para as estruturas de dados adequadas no arranque do programa. As Disciplinas, as pessoas e os exames estão guardados em ficheiros de objetos, enquanto que os utilizadores da aplicação estão guardados num ficheiro de texto.

ESTRUTURA GERAL DO PROGRAMA

No arranque do programa, é perguntado ao utilizador se este pretende fazer o seu registo ou o login. Se pretender fazer o registo, é pedido que introduza o username e a password, que serão escritos no ficheiro “Autenticacao.txt”, onde estão guardadas todas as informações sobre os clientes na forma de um csv. Caso pretenda fazer o login, é pedido que introduza o username e a password. Estes serão comparados com as

informações do ficheiro e verificados se são válidos. Se sim, o utilizador é redirecionado para o menu. Caso contrário, é lhe pedido que introduza novamente o username e a password, até estarem corretos.

Para um melhor funcionamento do programa criámos várias funções que são essenciais ao longo de toda a sua execução. Estas são:

- listarPessoas() - lista todo o tipo de pessoas – alunos, docentes e não docentes
- listarDisciplinas() – lista todas as disciplinas
- listarExames() – lista todo o tipo de exames – normal, recurso e especial
- escolherPessoa() – devolve uma pessoa pretendida pelo utilizador (aluno, docente ou não docente) conforme a classe que recebe como argumento
- escolherDisciplina() – devolve uma disciplina
- escolherExame() – devolve um exame
- devolveInteiroValido() – como todos os inputs do utilizador deverão ser números (excepto no login/registo), esta é a única proteção de input necessária do programa. Recebe como argumento um número máximo e devolve um número inteiro válido entre 1 e o número máximo. Fica em ciclo caso o input seja uma string ou um número inválido
- compararData – recebe como argumento 2 datas e 2 durações. Faz o somatório da data para transformar a data em um único número inteiro. Devolve 0 caso data coincida ou 1 caso contrário. Esta função é utilizada para verificar se docente não tem vigilâncias sobrepostas e para confirmar que a sala não seja usada durante um determinado tempo

Dentro da função menu() o cliente tem várias opções de escolha.

Se escolher a opção 1, a função adicionarExame() é chamada. Aí é pedido que escolha disciplina do exame e que introduza a data, a duração e o tipo de exame (normal (1), recurso (2) ou especial (3)). Posteriormente, o docente da disciplina é adicionado ao docente responsável do exame automaticamente e os outros docentes da disciplina são adicionados aos vigilantes, também de forma automática, se estiverem disponíveis. Finalmente, depois de o exame ser adicionado ao Arraylist de exames, o ficheiro de exames é atualizado.

Se escolher a opção 2, a função configurarSala() é chamada. É primeiro pedido ao cliente que escolha o exame para o qual deseja marcar sala. Depois é pedido que introduza o número da sala (salas válidas de 1 a 10). Se houver disponibilidade daquela sala àquele horário, a sala do exame fica marcada e o ficheiro de exame é atualizado.

Caso escolha a opção 3, a função convocarFuncionários() é chamada. Primeiro, é pedido ao cliente que escolha o exame para o qual pretende convocar funcionários. Caso pretenda associar docentes a vigilância de um exame, são listados todos os docentes. Depois de escolher o docente, é verificado se este tem disponibilidade. Em caso afirmativo, é verificado se o docente já tinha sido adicionado ao exame. Se não tinha, o docente é adicionado ao exame. É feito o mesmo caso for pretendido adicionar funcionários não docentes de apoio ao exame, com exceção da verificação da disponibilidade, pois este tipo de funcionário pode dar apoio a vários exames. Finalmente, o ficheiro de exames é atualizado.

Se escolher a opção 4, a função inscreverAluno() é chamada. Primeiro é pedido que escolha o aluno que quer inscrever. Depois é pedido que o cliente escolha em que exame pretende inscrever o aluno. Se o aluno estiver inscrito na disciplina do exame, se tiver

acesso ao tipo de exame pretendido e se ainda não tinha sido inscrito neste exame, o aluno é então inscrito no exame. Finalmente, o ficheiro de exames é atualizado.

Se o cliente escolher a opção 5, pede-se que este escolha o exame para o qual quer lançar notas. Depois é chamado o método lançar notas, que permite classificar os alunos. Caso não haja nenhum aluno inscrito no exame, o cliente é informado.

Caso escolha a opção 6, são listados todos os exames com recurso à função listarExames().

Se escolher a opção 7, pede-se que este escolha o exame para o qual quer listar notas. Caso não haja nenhum aluno inscrito no exame, o cliente é informado. Se há alunos inscritos no exame, usando o método listarNotas() da classe Exame, são listados os alunos e as respetivas classificações referentes àquele exame.

Caso o cliente escolha a opção 8, a função listarExamesAluno() é chamada. Primeiro, é pedido ao cliente que escolha o aluno. Esta função percorre todos os exames e verifica se aluno está inscrito no exame e qual a sua classificação. Se está, essa informação é mostrada na consola. O cliente é informado se o aluno não estiver inscrito em nenhum exame.

Se escolher a opção 9, é pedido ao utilizador que escolha o exame para o qual pretende listar docentes e funcionários. Depois é chamado o método listarFuncionarios() desse exame e são impresso na consola o docente responsável, os vigilantes e os não docentes de apoio ao exame. Caso não haja vigilantes ou não docentes de apoio associados a este exame o cliente é informado.

Caso escolha a opção 10, a função listarExamesFuncionario() é chamada. Depois do cliente escolher o funcionário, os exames são percorridos e verificado se o funcionário consta em cada exame. Se sim, o exame é impresso na consola. Se o funcionário não estiver inscrito em nenhum exame, o utilizador é informado.

Finalmente, se escolher a opção 11, o programa é terminado.

DIAGRAMA DE CLASSES

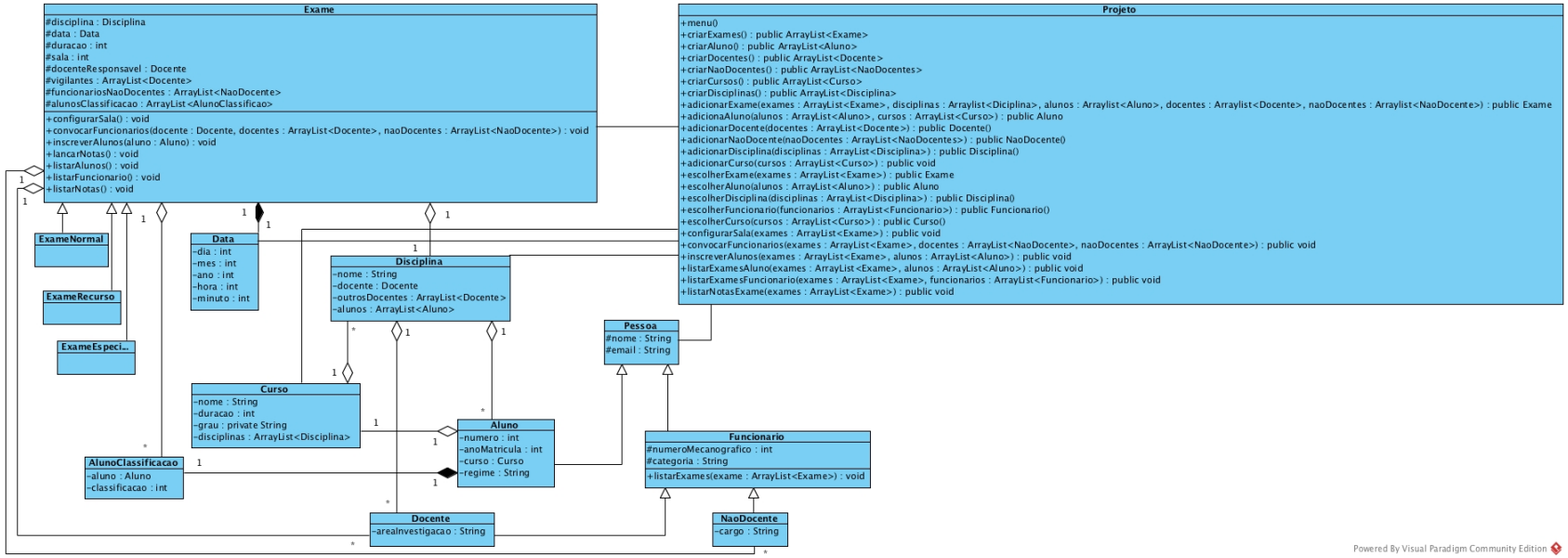
Relativamente à meta 1, os diagramas de classes alteraram-se um pouco. O diagrama da direita na próxima página refere-se à meta 1 e o diagrama da esquerda é o final.

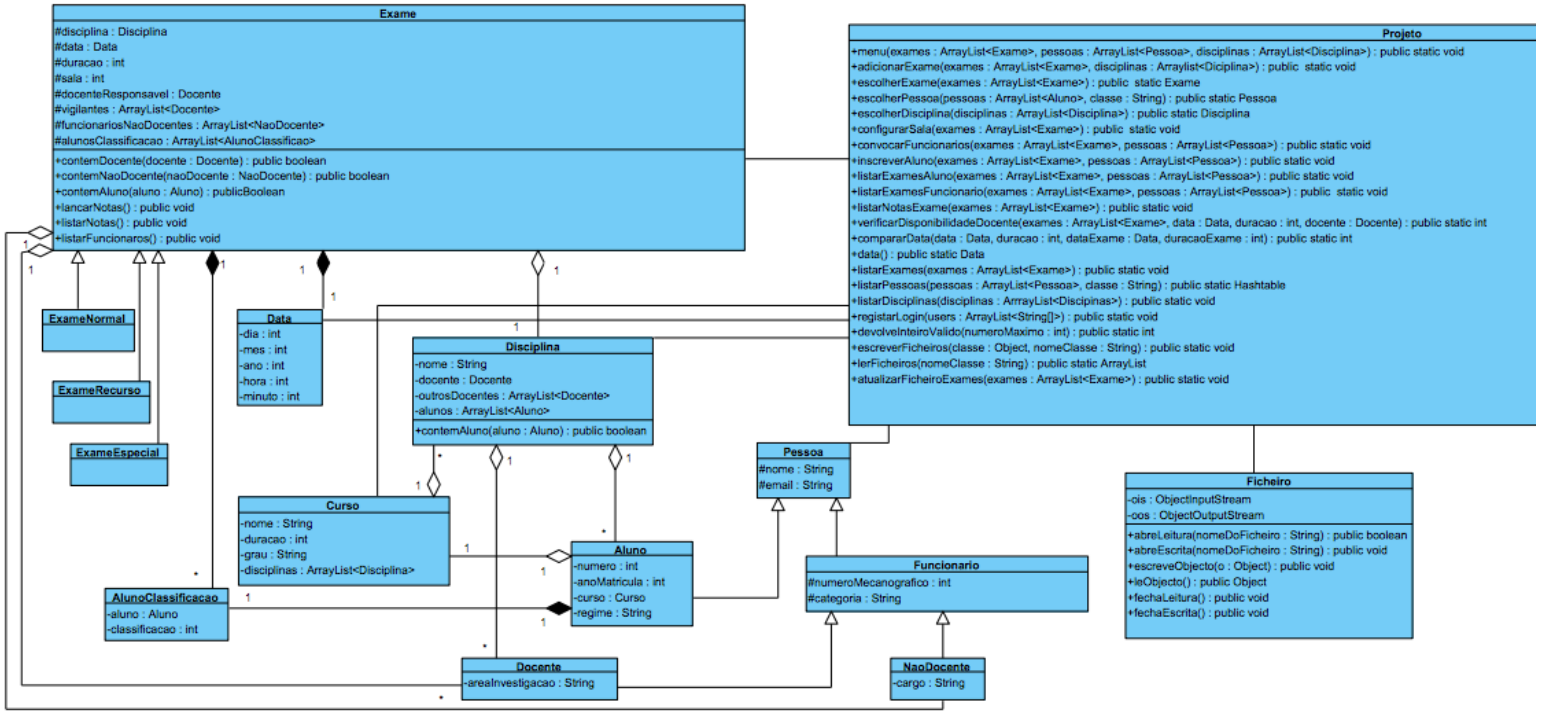
Como é possível observar, há poucas diferenças entre os dois UMLs.

CONCLUSÃO

Para concluir, pensamos que a aplicação está bem conseguida e tem em conta os Princípios da Programação Orientada a objetos. A interface está bastante “user friendly” dado que o utilizador apenas tem que inserir números, o que torna a utilização do programa mais rápida. Os comentários ao longo do programa também ajudam quem está a ler o código a perceber como o este está estruturado.

É de salientar que para além de ter todas as funcionalidades pedidas, implementamos funcionalidades extra como o Registo e o Login.


Exame
#disciplina : Disciplina
#data : Data
#duracao : int
#sala : int
#docenteResponsavel : Docente
#vigilantes : ArrayList<Docente>
#funcionariosNaoDocentes : ArrayList<NaoDocente>
#alunosClassificacao : ArrayList <AlunoClassificao>
+configurarSala() : void
+convocarFuncionarios(docente : Docente, docentes : ArrayList<Docente>, naoDocentes : ArrayList<NaoDocente>) : void
+inscreverAlunos(aluno : Aluno) : void
+lancarNotas() : void
+listarAlunos() : void
+listarFuncionario() : void
+listarNotas() : void
Projeto
+menu()
+criarExames() : public ArrayList<Exame>
+criarAluno() : public ArrayList<Aluno>
+criarDocentes() : public ArrayList<Docente>
+criarNaoDocentes() : public ArrayList<NaoDocentes>
+criarCursos() : public ArrayList<Curso>
+criarDisciplinas() : public ArrayList<Disciplina>
+adicionarExame(exames : ArrayList<Exame>, disciplinas : Arraylist<Diciplina>, alunos : Arraylist<Aluno>, docentes : Arraylist<Docente>, naoDocentes : Arraylist<NaoDocente>) : public Exame
+adicionaAluno(alunos : ArrayList<Aluno>, cursos : ArrayList<Curso>) : public Aluno
+adicionarDocente(docentes : ArrayList<Docente>) : public Docente()
+adicionarNaoDocente(naoDocentes : ArrayList<NaoDocentes>) : public NaoDocente()
+adicionarDisciplina(disciplinas : ArrayList<Disciplina>) : public Disciplina()
+adicionarCurso(cursos : ArrayList<Curso>) : public void
+escolherExame(exames : ArrayList<Exame>) : public Exame
+escolherAluno(alunos : ArrayList<Aluno>) : public Aluno
+escolherDisciplina(disciplinas : ArrayList<Disciplina>) : public Disciplina()
+escolherFuncionario(funcionarios : ArrayList<Funcionario>) : public Funcionario()
+escolherCurso(cursos : ArrayList<Curso>) : public Curso()
+configurarSala(exames : ArrayList<Exame>) : public void
+convocarFuncionarios(exames : ArrayList<Exame>, docentes : ArrayList<NaoDocente>, naoDocentes : ArrayList<NaoDocente>) : public void
+inscreverAlunos(exames : ArrayList<Exame>, alunos : ArrayList<Aluno>) : public void
+listarExamesAluno(exames : ArrayList<Exame>, alunos : ArrayList<Aluno>) : public void
+listarExamesFuncionario(exames : ArrayList<Exame>, funcionarios : ArrayList<Funcionario>) : public void
+listarNotasExame(exames : ArrayList<Exame>) : public void
ExameNormal
ExameRecurso
ExameEspeci...
Data
-dia : int
-mes : int
-ano : int
-hora : int
-minuto : int
Disciplina
-nome : String
-docente : Docente
-outrosDocentes : ArrayList<Docente>
-alunos : ArrayList<Aluno>
Pessoa
#nome : String
#email : String
Curso
-nome : String
-duracao : int
-grau : private String
-disciplinas : ArrayList<Disciplina>
Aluno
-numero : int
-anoMatricula : int
-curso : Curso
-regime : String
Funcionario
#numeroMecanografico : int
#categoria : String
+listarExames(exame : ArrayList<Exame>) : void
AlunoClassificacao
-aluno : Aluno
-classificacao : int
Docente
-areaInvestigacao : String
NaoDocente
-cargo : String
Powered By Visual Paradigm Community Edition


Exame
#disciplina : Disciplina
#data : Data
#duracao : int
#sala : int
#docenteResponsavel : Docente
#vigilantes : ArrayList<Docente>
#funcionariosNaoDocentes : ArrayList<NaoDocente>
#alunosClassificacao : ArrayList<AlunoClassificao>
+contemDocente(docente : Docente) : public boolean
+contemNaoDocente(naoDocente : NaoDocente) : public boolean
+contemAluno(aluno : Aluno) : publicBoolean
+lancarNotas() : public void
+listarNotas() : public void
+listarFuncionaros() : public void
Projeto
+menu(exames : ArrayList<Exame>, pessoas : ArrayList<Pessoa>, disciplinas : ArrayList<Disciplina>) : public static void
+adicionarExame(exames : ArrayList<Exame>, disciplinas : Arraylist<Diciplina>) : public static void
+escolherExame(exames : ArrayList<Exame>) : public static Exame
+escolherPessoa(pessoas : ArrayList<Aluno>, classe : String) : public static Pessoa
+escolherDisciplina(disciplinas : ArrayList<Disciplina>) : public static Disciplina
+configurarSala(exames : ArrayList<Exame>) : public static void
+convocarFuncionarios(exames : ArrayList<Exame>, pessoas : ArrayList<Pessoa>) : public static void
+inscreverAluno(exames : ArrayList<Exame>, pessoas : ArrayList<Pessoa>) : public static void
+listarExamesAluno(exames : ArrayList<Exame>, pessoas : ArrayList<Pessoa>) : public static void
+listarExamesFuncionario(exames : ArrayList<Exame>, pessoas : ArrayList<Pessoa>) : public static void
+listarNotasExame(exames : ArrayList<Exame>) : public static void
+verificarDisponibilidadeDocente(exames : ArrayList<Exame>, data : Data, duracao : int, docente : Docente) : public static int
+compararData(data : Data, duracao : int, dataExame : Data, duracaoExame : int) : public static int
+data() : public static Data
+listarExames(exames : ArrayList<Exame>) : public static void
+listarPessoas(pessoas : ArrayList<Pessoa>, classe : String) : public static Hashtable
+listarDisciplinas(disciplinas : ArrrayList<Discipinas>) : public static void
+registarLogin(users : ArrayList<String[]>) : public static void
+devolveInteiroValido(numeroMaximo : int) : public static int
+escreverFicheiros(classe : Object, nomeClasse : String) : public static void
+lerFicheiros(nomeClasse : String) : public static ArrayList
+atualizarFicheiroExames(exames : ArrayList<Exame>) : public static void
ExameNormal
ExameRecurso
ExameEspecial
Data
-dia : int
-mes : int
-ano : int
-hora : int
-minuto : int
Disciplina
-nome : String
-docente : Docente
-outrosDocentes : ArrayList<Docente>
-alunos : ArrayList<Aluno>
+contemAluno(aluno : Aluno) : public boolean
Pessoa
#nome : String
#email : String
Ficheiro
-ois : ObjectInputStream
-oos : ObjectOutputStream
+abreLeitura(nomeDoFicheiro : String) : public boolean
+abreEscrita(nomeDoFicheiro : String) : public void
+escreveObjecto(o : Object) : public void
+leObjecto() : public Object
+fechaLeitura() : public void
+fechaEscrita() : public void
Curso
-nome : String
-duracao : int
-grau : String
-disciplinas : ArrayList<Disciplina>
Aluno
-numero : int
-anoMatricula : int
-curso : Curso
-regime : String
Funcionario
#numeroMecanografico : int
#categoria : String
AlunoClassificacao
-aluno : Aluno
-classificacao : int
Docente
-areaInvestigacao : String
NaoDocente
-cargo : String