FACULDADE DE CIÊNCIAS E TECNOLOGIAS DA UNIVERSIDADE DE COIMBRA

Bib – Biblioteca

Relatório do Projecto da cadeira de Programação Orientada a Objectos

Ano lectivo 2013/2014

André Martins Carrilho Costa Baptista 2012137523
Rosa Manuela Rodrigues de Faria 2005128014

Índice

Estrutura Geral

Este projecto consiste num sistema de gestão de uma biblioteca. Permite que os administradores possam gerir a biblioteca, efectuando vários tipos de operções, tais como registar artigos e utilizadores. Permite também aos leitores, a pesquisa e requisição facilitada de artigos existentes na biblioteca.

O projecto tem dois tipos de classes: as classes principais e as classes relativas à interface gráfica.

As classes principais contêm todos os métodos necessários para o tratamento de informação, processamento de dados, cálculos estatísticos, procura e comparação de objectos, entre outros, que irão ser aludidos ao longo do relatório. Em suma, estas classes são o cérebro do programa e realizam todo o trabalho interno, do qual o utilizador não se apercebe de forma directa.

As classes relativas à interface gráfica contém métodos que permitem a interacção entre o utilizador e o programa. O utilizador pode utilizar os componentes da interface para recorrer aos métodos das classes principais, enviando pedidos de processamento de informação e, posteriormente, receber o resultado. As classes relativas à interface gráfica permitem apresentar o resultado num dos vários componentes da interface.

Quando o programa é iniciado carrega para a memória todos os dados que estão guardados nos ficheiros apresentados na tabela seguinte. Caso os ficheiros não existam são criados para que seja possível guardar os dados, de forma a poderem ser reutilizados em cada execução.

Nome do ficheiro	Tipo de ficheiro	Descrição
leitores	Ficheiro de objectos	Contém um ArrayList de objectos do tipo Leitor
admin	Ficheiro de objectos	Contém um ArrayList de objectos do tipo Admin
books	Ficheiro de texto	Informações sobre vários artigos do tipo Livro
DVDs	Ficheiro de texto	Informações sobre vários artigos do tipo DVD
Reqs	Ficheiro de objectos	Contém um ArrayList de objectos do tipo Requisicao

Estes dados são carregados para ArrayLists, recorrendo às funções da classe Login: loadReaders(), loadAdmins(), loadBooks(), loadDVDs() e loadReqs(). Para evitar o uso da *keyword* **static**, estes ArrayLists são passados como parâmetro para todos os métodos que precisam de as utilizar, de forma a que seja possível adicionar, remover ou alterar objectos.

Depois do carregamento dos dados, o utilizador deve-se autenticar no sistema. Exitem dois tipos principais de utilizador: o leitor e o administrador, cada um com privilégios distintos.

O leitor pode procurar e requisitar artigos, visualizar e alterar os seus dados pessoais (através do menu “Eu” na janela principal – onde pode também fazer *logout*), visualizar as suas requisições e entregar artigos (através das suas requisições). O administrador pode procurar e adicionar artigos, visualizar e alterar dados pessoais, adicionar ou remover utilizadores, alterar

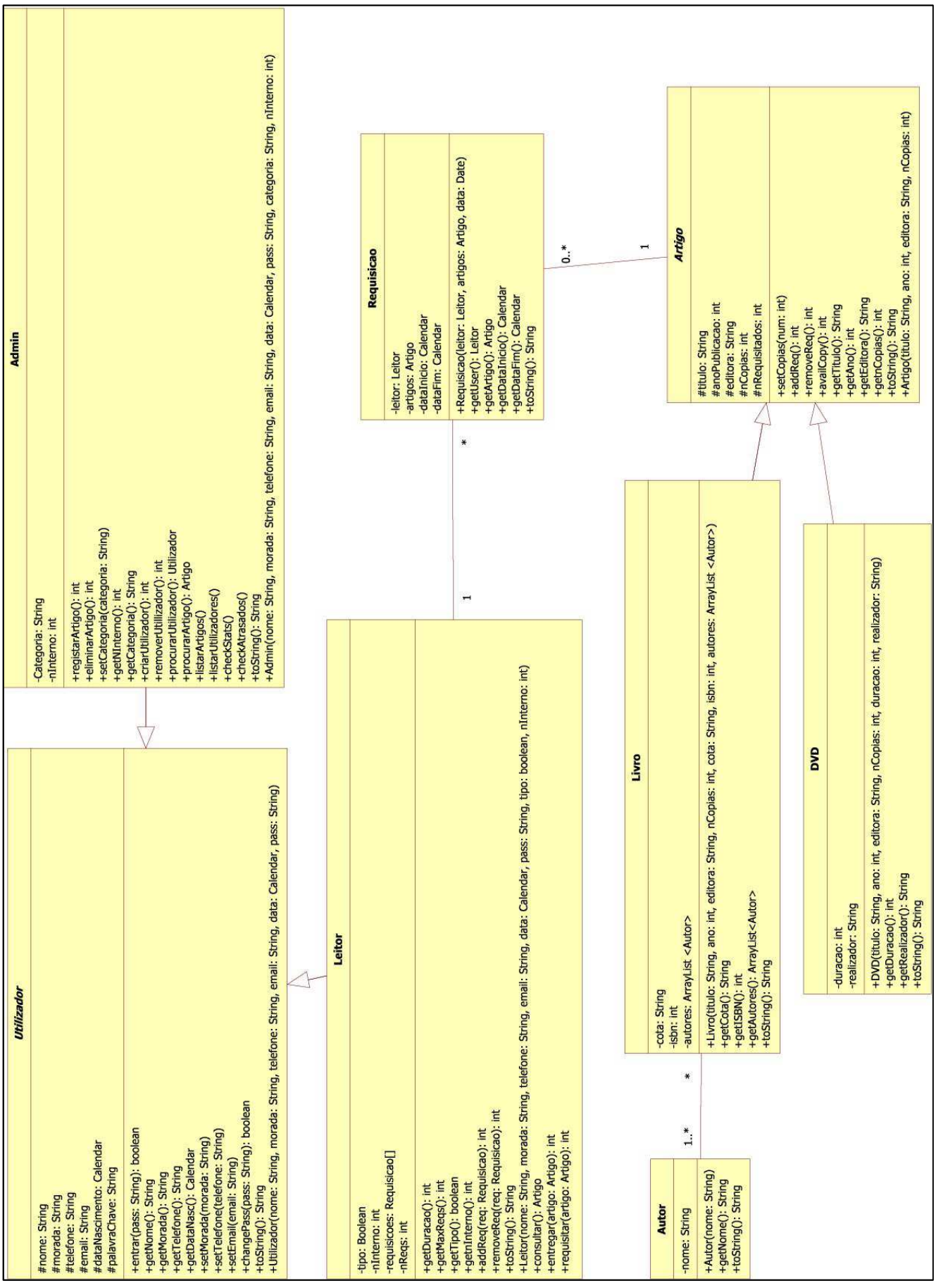
os dados dos utilizadores, visualizar estatísticas para um determinado mês, visualizar as requisições atrasadas e os artigos que se encontram requisitados actualmente. Qualquer tipo de utilizador pode, ainda, mudar a sua password.

O programa calcula e apresenta vários dados estatísticos para um determinado mês: O número médio de requisições, o dia em que houve mais requisições, o livro e DVD mais requisitados, os artigos que não foram requisitados e ainda um gráfico que representa a evolução do número de requisições ao longo desse mês; mostra também as requisições atrasadas. Todas estas opções estão numa janela separada por tabs para ser facilmente navegável. Nesta janela há ainda um pequeno menu "Limpar" que permite limpar todos os campos e informações da janela.

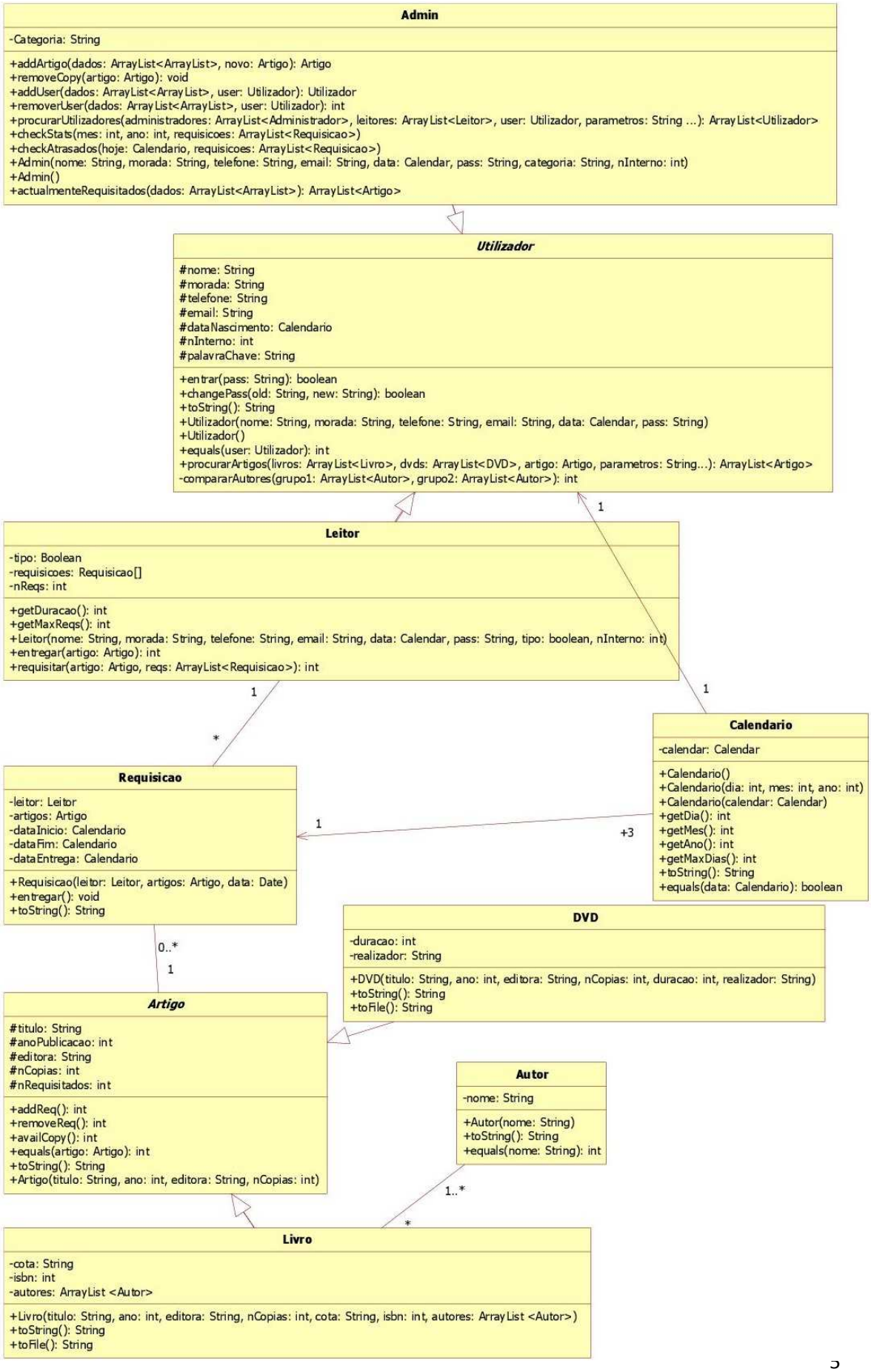
Foram ainda implementadas algumas protecções, tais como: verificação da validade dos dados que o utilizador fornece, verificação das datas e tratamento de excepções. Foi ainda utilizado um *Window Listener*, com o objectivo de fechar todas as janelas criadas após o início de sessão do utilizador, no momento em que o utilizador termina sessão.

Diagramas de classes

Inicial



Final


Admin
-Categoria: String
+addArtigo(dados: ArrayList<ArrayList>, novo: Artigo): Artigo
+removeCopy(artigo: Artigo): void
+addUser(dados: ArrayList<ArrayList>, user: Utilizador): Utilizador
+removerUser(dados: ArrayList<ArrayList>, user: Utilizador): int
+procurarUtilizadores(administradores: ArrayList<Administrador>, leitores: ArrayList<Leitor>, user: Utilizador, parametros: String ...): ArrayList<Utilizador>
+checkStats(mes: int, ano: int, requisicoes: ArrayList<Requisicao>)
+checkAtrasados(hoje: Calendario, requisicoes: ArrayList<Requisicao>)
+Admin(nome: String, morada: String, telefone: String, email: String, data: Calendar, pass: String, categoria: String, nInterno: int)
+Admin()
+actualmenteRequisitados(dados: ArrayList<ArrayList>): ArrayList<Artigo>
Utilizador
#nome: String
#morada: String
#telefone: String
#email: String
#dataNascimento: Calendario
#nInterno: int
#palavraChave: String
+entrar(pass: String): boolean
+changePass(old: String, new: String): boolean
+toString(): String
+Utilizador(nome: String, morada: String, telefone: String, email: String, data: Calendar, pass: String)
+Utilizador()
+equals(user: Utilizador): int
+procurarArtigos(livros: ArrayList<Livro>, dvds: ArrayList<DVD>, artigo: Artigo, parametros: String...): ArrayList<Artigo>
-compararAutores(grupo1: ArrayList<Autor>, grupo2: ArrayList<Autor>): int
Leitor
-tipo: Boolean
-requisicoes: Requisicao[]
-nReqs: int
+getDuracao(): int
+getMaxReqs(): int
+Leitor(nome: String, morada: String, telefone: String, email: String, data: Calendar, pass: String, tipo: boolean, nInterno: int)
+entregar(artigo: Artigo): int
+requisitar(artigo: Artigo, reqs: ArrayList<Requisicao>): int
Calendario
-calendar: Calendar
+Calendario()
+Calendario(dia: int, mes: int, ano: int)
+Calendario(calendar: Calendar)
+getDia(): int
+getMes(): int
+getAno(): int
+getMaxDias(): int
+toString(): String
+equals(data: Calendario): boolean
Requisicao
-leitor: Leitor
-artigos: Artigo
-dataInicio: Calendario
-dataFim: Calendario
-dataEntrega: Calendario
+Requisicao(leitor: Leitor, artigos: Artigo, data: Date)
+entregar(): void
+toString(): String
DVD
-duracao: int
-realizador: String
+DVD(titulo: String, ano: int, editora: String, nCopias: int, duracao: int, realizador: String)
+toString(): String
+toFile(): String
Artigo
#titulo: String
#anoPublicacao: int
#editora: String
#nCopias: int
#nRequisitados: int
+addReq(): int
+removeReq(): int
+availCopy(): int
+equals(artigo: Artigo): int
+toString(): String
+Artigo(titulo: String, ano: int, editora: String, nCopias: int)
Autor
-nome: String
+Autor(nome: String)
+toString(): String
+equals(nome: String): int
Livro
-cota: String
-isbn: int
-autores: ArrayList <Autor>
+Livro(titulo: String, ano: int, editora: String, nCopias: int, cota: String, isbn: int, autores: ArrayList <Autor>)
+toString(): String
+toFile(): String
1
*
1
1
1
+3
0..*
1
1..*
*

Descrição das principais estruturas de dados

As classes principais deste trabalho são, como é visível no diagrama anterior, as classes Utilizador e suas classes derivadas (Leitor e Admin), Requisição e Artigo e suas classes derivadas, Livro e DVD. Temos ainda duas classes, Autor e Calendario.

A classe Utilizador é a representação dos utilizadores genéricos, sendo portanto abstracta, já que um Utilizador ou é Admin ou é Leitor. Assim, os atributos desta classe são os atributos que são comuns a ambos os tipos de utilizadores, ou seja, nome, morada, telefone, e-mail, data de Nascimento, o nº interno de identificação no sistema e a password. Da mesma forma, terá as funções que lhes são comuns: entrar no sistema, mudar a password, e procurar artigos. Deste modo terá os métodos correspondentes a essas funções, getters e setters, toString e equals, e um método auxiliar ao procurarArtigos, compararAutores.

Por seu lado, a classe Admin é a classe cujos objectos são os administradores do sistema, tendo portanto como funções, para além das do Utilizador, a gestão das listas de utilizadores e das listas de artigos (adicionar e remover). É também a única classe que pode procurar utilizadores (dado que um Leitor apenas poderá ver e editar os seus próprios dados) e editá-los. Uma outra função desta classe é a de calcular as estatísticas mensais, bem como listar as requisições atrasadas e os artigos requisitados no momento. Esta classe tem como atributo, além das anteriores, a categoria profissional do mesmo.

Quanto aos objectos do tipo Leitor, têm como principal objectivo requisitar e entregar Artigos (para além dos do Utilizador), tendo como métodos auxiliares a esses métodos para adicionar e remover requisições do seu array de requisições e métodos que permitem determinar qual o número máximo de requisições que podem fazer e a duração máxima das mesmas, consoante o tipo de Leitor. Para este efeito terá como atributos extra um array de requisições, com tamanho igual ao número máximo de requisições que pode fazer, e um booleano que representa o seu tipo (true para professor, false para aluno).

A classe Artigo define os objectos do tipo Artigo, representativos dos items existentes na biblioteca. Esta é, mais uma vez, abstracta, pois os items ou são Livros ou DVDs. Novamente, os seus atributos e métodos são os que são comuns aos dois tipos: como atributos, título, ano de publicação, editora, nº de cópias existentes e nº de cópias requisitadas; como funções, adicionar e remover cópias existentes e cópias requisitadas, e determinar o número de cópias disponíveis.

As suas classes derivadas apenas têm diferentes atributos, dado que são utilizadas da mesma forma; a classe Livro tem como atributos a mais a cota, o ISBN e a lista de autores, e a classe DVD, o realizador e a duração.

A classe Requisicao é, assim, o cerne da questão, já que liga um utilizador a um artigo, com as datas da criação da requisição, de limite de entrega, e a data em que foi de facto entregue o artigo (se foi entregue). O único método particular a esta classe é o que tem como

função representar a entrega do artigo, que nada mais faz a não ser definir a data de entrega como a data actual.

A classe Autor tem apenas como propósito identificar o autor que tem pelo menos um Livro no sistema, não tendo qualquer função em particular, além das funções standard do Java.

Finalmente, a classe Calendario foi criada para auxiliar no tratamento das datas. Esta tem como atributo um calendar do tipo Calendar, e funciona como "caixa negra" deste tipo de objectos: determina o dia, mês e ano (excluindo assim as unidades menor que o dia), compara, converte para String e calcula o número máximo de dias no mês em questão, e permitindo que se use o construtor tanto com um parâmetro do tipo Calendar, como com os parâmetros inteiros dia, mês e ano.

Instruções de utilização

Para correr o programa a partir do IDE NetBeans deve ser aberto o projecto (pasta Bib) no mesmo. Seguidamente é necessário importar a biblioteca jmathplot.jar. Esta biblioteca está na pasta Bib/src/bib/resources/. Para importar a biblioteca dever-se-á clicar com o botão direito no nome do projecto do NetBeans, clicando em Properties. Na secção Libraries poderá ser encontrada a biblioteca com um erro, bastando substituí-la pelo ficheiro acima mencionado.

Para executar o programa com uma JVM basta abrir o ficheiro Projecto.jar, tendo em atenção que se deve ter na mesma directoria os ficheiros da base de dados admin, dvd.txt, livros.txt, readers e reqs, bem como a pasta lib com a biblioteca jmathplot. Caso os ficheiros da base de dados não estejam na mesma pasta do executável o programa irá criar um utilizador "root" do tipo Administrador, com password "root", não tendo mais nenhuns dados, podendo ser iniciado um registo completamente novo. Para efeitos de segurança recomenda-se que na primeira utilização seja criado um novo administrador e removido o utilizador "root".

Anexo - Listagem do programa

```java
package bib;

import java.io.Serializable;
import java.util.ArrayList;
import java.util.Calendar;

/** Classe que representa um Utilizador do tipo Administrador.
 * <p> Cada administrador tem a função de gerir utilizadores e artigos na biblioteca.
 *
 * @author André Baptista (2012137523)
 * @author Rosa Faria (2005128014)
 */
public class Admin extends Utilizador implements Serializable{
    private String categoria;
    static final long serialVersionUID = 6784420523545791913L;

    /** Construtor vazio */
    Admin(){}

    /** Construtor do tipo Administrador.
     * <p> Chama o construtor super e define a categoria e número interno do Administrador
     *
     * @param nome nome do utilizador
     * @param morada morada do utilizador
     * @param telefone telefone do utilizador
     * @param email e-mail do utilizador
     * @param data data de nascimento do utilizador
     * @param pass palavra-chave da conta do utilizador
     * @param categoria categoria profissional do administrador
     * @param nInterno número interno do Administrador
     */
    Admin(String nome, String morada, String telefone, String email, Calendar data, String pass, String categoria, int nInterno) {
        super(nome, morada, telefone, email, data, pass);
        this.categoria = categoria;
        this.setnInterno(nInterno);
    }

    /** Método para remover uma cópia do artigo se houver alguma cópia não requisitada
     * @param artigo artigo a que se quer remover uma cópia
     */
    public void removeCopy (Artigo artigo) {
        if (artigo.availCopy()>0) artigo.setnCopias(artigo.getnCopias()-1);
    }

    /** Método para criar um novo artigo.
     * <p> Começa por verificar que o artigo ainda não existe na base de dados
     * <p> Se não existir adiciona, pondo o número de cópias a 1
     * @param dados ArrayList com todos os dados da biblioteca
     * @param novo Artigo a inserir
     * @return ponteiro para novo artigo, null se artigo já existia
     */
    public Artigo addArtigo(ArrayList<ArrayList> dados, Artigo novo) {
        if (novo instanceof Livro) {
            /**dados.get(1) é o ArrayList de livros do sistema**/
            for (int i = 0; i<dados.get(1).size();i++) {
```

```
56                  if (novo.equals(dados.get(1).get(i)))
57                      /**encontrou um artigo igual**/
58                      return null;
59              }
60              novo.setnCopias(1);
61              dados.get(1).add(novo);
62              return (Artigo)dados.get(1).get(dados.get(1).size()-1);
63          }
64          else {
65              /**dados.get(0) é o ArrayList de DVDs do sistema**/
66              for (int i = 0; i<dados.get(0).size();i++) {
67                  if (novo.equals(dados.get(0).get(i)))
68                      /**encontrou um artigo igual**/
69                      return null;
70              }
71              novo.setnCopias(1);
72              dados.get(0).add(novo);
73              return (Artigo) dados.get(0).get(dados.get(0).size()-1);
74          }
75      }
76
77      /** Método para adicionar utilizador.
78       * <p> Começa por verificar se ainda não existe o utilizador
79       * <p> Se não existir, adiciona e atribui-lhe um id igual ao
80       * id do último utilizador da lista +1
81       *
82       * @param dados ArrayList com os dados da biblioteca
83       * @param user Utilizador novo a inserir
84       * @return ponteiro para novo utilizador, null se o utilizador já existia
85       */
86      public Utilizador addUser(ArrayList<ArrayList> dados, Utilizador user) {
87          if (user instanceof Leitor) {
88              /**dados.get(5) é o ArrayList de leitores do sistema**/
89              ArrayList<Leitor> leitor = dados.get(5);
90              for (int i = 0; i<leitor.size();i++) {
91                  if (user.equals(leitor.get(i)))
92                      /**encontrou um Utilizador igual**/
93                      return null;
94              }
95              /**Cria novo leitor com os mesmos dados e ID = ID do último no array +1**/
96              Leitor aux = new Leitor(user.getNome(), user.getMorada(), user.
                getTelefone(), user.getEmail(), user.getDataNascimento().getCalendar(),
                user.getPalavraChave(), ((Leitor)user).getTipo(), leitor.get(leitor.size
                ()-1).getnInterno()+1);
97              dados.get(5).add(aux);
98              return (Utilizador)dados.get(5).get(leitor.size()-1);
99          }
100         else {
101             /**dados.get(4) é o ArrayList de administradores do sistema**/
102             ArrayList<Admin> admins = dados.get(4);
103             for (int i = 0; i<admins.size();i++) {
104                 if (user.equals(admins.get(i)))
105                     /**encontrou um Utilizador igual**/
106                     return null;
107             }
108             /**Cria novo administrador com os mesmos dados e ID = ID do último no
                array +1**/
```

```
109             Admin aux = new Admin(user.getNome(), user.getMorada(), user.getTelefone
                (), user.getEmail(), user.getDataNascimento().getCalendar(), user.
                getPalavraChave(), ((Admin)user).getCategoria(),admins.get(admins.size()-
                1).getnInterno()+1);
110             dados.get(4).add(aux);
111             return (Utilizador) dados.get(4).get(admins.size()-1);
112         }
113     }
114 
115     /** Método para procurar o utilizador comparando os parametros dados
116      * com todos os utilizadores do mesmo tipo
117      * @param administradores ArrayList dos administradores do sistema
118      * @param leitores ArrayList dos leitores do sistema
119      * @param utilizador Utilizador a procurar
120      * @param parametros Lista de Strings representando os parâmetros a comparar.
121      * Parâmetros possíveis: nome, morada, telefone, email, data, tipo, categoria,
      id(dependendo do tipo de Utilizador)
122      * @return lista com utilizadores cujos parametros são identicos, null se não
      existir nenhum
123      */
124     public ArrayList<Utilizador> procurarUtilizadores(ArrayList<Admin>
      administradores, ArrayList<Leitor> leitores, Utilizador utilizador, String ...
      parametros){
125         /* Copiar todos os utilizadores do mesmo tipo para um array auxiliar
126          * Desse array vão sendo retirados todos os que são diferentes do inserido
127          */
128         ArrayList<Utilizador> aux = new ArrayList<>();
129         if (utilizador instanceof Leitor)
130             for (int i = 0; i<leitores.size();i++)
131                 aux.add(leitores.get(i));
132         else
133             for (int i = 0; i<administradores.size();i++)
134                 aux.add(administradores.get(i));
135 
136 
137         for (String param : parametros){
138             /* Percorre todos os parâmetros
139              * Para cada um percorre todos os utilizadores do array aux
140              * e vê se coincidem. Se não coincidirem remove do array aux
141              */
142             try {
143                 switch(param) {
144                     case "nome":
145                         for (int i = 0; i<aux.size();i++) {
146                             if (!aux.get(i).getNome().contains(utilizador.getNome()))
147                                 aux.remove(i--);
148                         }   break;
149                     case "morada":
150                         for (int i = 0; i<aux.size();i++) {
151                             if (!aux.get(i).getMorada().contains(utilizador.getMorada
                            ()))
152                                 aux.remove(i--);
153                         }   break;
154 
155                     case "telefone":
156                         for (int i = 0; i<aux.size();i++) {
157                             if (!aux.get(i).getTelefone().contains(utilizador.
```

```
                                getTelefone()))
158                                 aux.remove(i--);
159                         }   break;
160                     case "email":
161                         for (int i = 0; i<aux.size();i++) {
162                             if (!aux.get(i).getEmail().contains(utilizador.getEmail
                                ()))
163                                 aux.remove(i--);
164                         }   break;
165                     case "data":
166                         for (int i = 0; i<aux.size();i++) {
167                             if (!aux.get(i).getDataNascimento().equals(utilizador.
                                getDataNascimento()))
168                                 aux.remove(i--);
169                         }   break;
170                     case "tipo":
171                         for (int i = 0; i<aux.size();i++) {
172                             if (((Leitor)aux.get(i)).getTipo()!=((Leitor)utilizador
                                ).getTipo()))
173                                 aux.remove(i--);
174                         }   break;
175                     case "categoria":
176                         for (int i = 0; i<aux.size();i++) {
177                             if (!((Admin)aux.get(i)).getCategoria().equals(((Admin)
                                utilizador).getCategoria()))
178                                 aux.remove(i--);
179                         }   break;
180                     case "id":
181                         for (int i = 0; i<aux.size();i++) {
182                             if (aux.get(i).getnInterno()!=utilizador.getnInterno())
183                                 aux.remove(i--);
184                         }   break;
185                     default:
186                         System.out.println("Parâmetro não existente: "+param);
187                 }
188             }
189             catch (ClassCastException e){
190                 System.out.println("Parâmetro inválido para o tipo de utilizador: "+
                    param);
191             }
192         }
193         return aux;
194     }
195 
196     /** Método para remover utilizador do sistema.
197      * <p> Verifica se o utilizador existe,
198      * entrega todos os artigos que tiver requisitado,
199      * adiciona a palavra "(removido)"
200      * ao seu nome para manter a referência ao utilizador no registo das
201      * requisições feitas antes de ser eliminado
202      * @param dados ArrayList dos dados da Biblioteca
203      * @param user Utilizador a apagar
204      * @return 1 se for removido, -1 se não for encontrado (nem removido)
205      */
206     public int removeUser(ArrayList<ArrayList> dados, Utilizador user) {
207         if (dados.get(4).contains(user)) {
208             user.setNome(user.getNome()+" (removido)");
```

```
209             dados.get(4).remove(user);
210             return 1;
211         }
212         if (dados.get(5).contains(user)) {
213             for (int i = 0; i<((Leitor)user).getnReqs();i++)
214                 ((Leitor)user).entregar(((Leitor)user).getRequisicoes()[i].getArtigo
                    ());
215             user.setNome(user.getNome()+" (removido)");
216             dados.get(5).remove(user);
217             return 1;
218         }
219         else return -1;
220     }
221 
222     /** Método que calcula as estatísticas do mês dado.
223      * <p> Calcula o dia com mais requisições, a média das requisições diárias
224      * os artigos mais requisitados, e os que nunca foram requisitados
225      * @param mes (Int) mês cujas estatísticas queremos
226      * @param ano (Int) ano em que o mês se encontra
227      * @param dados ArrayList com os dados do sistema
228      * @return Arraylist que contém o número de requisições para cada dia do mês, o
        número médio de requisições por mês e o dia do mês em que houve
        mais requisições.
229      */
230     public ArrayList checkStats(int mes, int ano, ArrayList <ArrayList> dados) {
231         ArrayList<Requisicao> requisicoes = dados.get(3);
232         ArrayList<Livro> livros = dados.get(1);
233         ArrayList<DVD> dvds = dados.get(0);
234 
235         Calendario input = new Calendario(1,mes-1,ano), data;
236         double dias[] = new double[input.getMaxDias()], media = 0;
237         int max = 0, diaMaximo = 0, maxL = 0, maxD = 0, indexL = -1, indexD = -1;
238         int livrosStats[] = new int[livros.size()];
239         int dvdsStats[] = new int[dvds.size()];
240         ArrayList<Artigo> nenhumArtigo = new ArrayList<>();
241 
242         ArrayList stats = new ArrayList<>();
243 
244         /* Percorre o ArrayList de requisições
245          * Verifica se cada uma das requisições foi realizada no mês e ano pretendidos
246          * Incrementa o número de requisições de cada dia
247         */
248         for (int i = 0; i<requisicoes.size();i++) {
249             data = requisicoes.get(i).getDataInicio();
250             if ((data.getMes() == mes) && (data.getAno()  == ano)) {
251                 dias[data.getDia()-1]++;
252                 //Incrementa contador das requisições do artigo
253                 if (requisicoes.get(i).getArtigo() instanceof Livro)
254                     livrosStats[livros.indexOf(requisicoes.get(i).getArtigo())]++;
255                 else
256                     dvdsStats[dvds.indexOf(requisicoes.get(i).getArtigo())]++;
257             }
258         }
259 
260         //Procura livro mais requisitado e livros não requisitados
261         for (int i = 0; i<livros.size();i++) {
262             if (livrosStats[i] == 0) {
```

```
263                 nenhumArtigo.add(livros.get(i));
264             }
265             else if (livrosStats[i]>maxL) {
266                 maxL = livrosStats[i];
267                 indexL = i;
268             }
269         }
270 
271         //Procura DVD mais requisitado e DVDs não requisitados
272         for (int i = 0; i<dvds.size();i++) {
273             if (dvdsStats[i] == 0) {
274                 nenhumArtigo.add(dvds.get(i));
275             }
276             else if (dvdsStats[i]>maxD) {
277                     maxD = dvdsStats[i];
278                     indexD = i;
279             }
280         }
281 
282         //Percorre o array de dias e obtém o índice com mais requisições
283         for (int i=0;i<dias.length;i++){
284             media+=dias[i];
285             if (dias[i] > max) {
286                 max = (int)dias[i];
287                 diaMaximo = i;
288             }
289         }
290         diaMaximo++;
291         media /= dias.length;
292 
293         //juntar todos os arrays e dados no ArrayList stats para ser devolvido
294         stats.add(dias); //0
295         stats.add(media); //1
296         stats.add(diaMaximo); //2
297         stats.add(nenhumArtigo); //3
298         if (indexL==-1) stats.add(null);
299         else stats.add(livros.get(indexL)); //4
300         if (indexD==-1) stats.add(null);
301         else stats.add(dvds.get(indexD)); //5
302         return stats;
303     }
304 
305     /**
306      * Função para listar os artigos que estão actualmente requisitados
307      * @param dados Dados do sistema
308      * @return ArrayList com os artigos que têm pelo menos 1 cópia requisitada
309      */
310     public ArrayList<Artigo> actualmenteRequisitados(ArrayList<ArrayList> dados) {
311         ArrayList<Artigo> aux = new ArrayList<>();
312         for (int i = 0; i< dados.get(0).size();i++) {
313             if (((Artigo)dados.get(0).get(i)).getnRequisitados()>0) {
314                 aux.add((Artigo)dados.get(0).get(i));
315             }
316         }
317         for (int i = 0; i< dados.get(1).size();i++) {
318             if (((Artigo)dados.get(1).get(i)).getnRequisitados()>0) {
319                 aux.add((Artigo)dados.get(1).get(i));
```

```
320                 }
321             }
322             return aux;
323         }
324 
325         /** Método que verifica todas as requisicoes atrasadas
326          * @param hoje Calendario com a data de hoje
327          * @param requisicoes ArrayList com todas as requisições do sistema
328          * @return ArrayList com as requisicoes cuja data de entrega é anterior à data
          de hoje
329          */
330         public ArrayList <Requisicao> checkAtrasados(Calendario hoje, ArrayList <
          Requisicao> requisicoes) {
331             ArrayList <Requisicao> atrasadas = new ArrayList <>();
332             for (int i = 0; i<requisicoes.size();i++)
333                 if (((hoje.getCalendar()).after(requisicoes.get(i).getDataFim().
                    getCalendar()))&&(requisicoes.get(i).getDataEntrega()==null))
334                     atrasadas.add(requisicoes.get(i));
335             return atrasadas;
336         }
337 
338         /** Compara o Administrador this com o Administrador user.
339          * Compara os campos nome, nº interno, telefone, morada e e-mail
340          * @param user Utilizador a comparar
341          * @return true se todos forem iguais, false caso contrário
342          */
343         @Override
344         public boolean equals(Utilizador user) {
345             Admin leitor = (Admin) user;
346             return (this.getNome().equals(leitor.getNome()))&&(this.getnInterno() ==
                leitor.getnInterno())&&(this.getTelefone().equals(leitor.getTelefone()))&&(
                this.getMorada().equals(leitor.getMorada()))&&(this.getEmail().equals(leitor.
                getEmail()));
347         }
348 
349         @Override
350         public String toString() {
351             return this.getNome() +  ", " + this.getEmail() + ", Telefone: " + this.
                getTelefone() +", Data de nascimento: " + this.getDataNascimento();
352         }
353 
354         /** Setter do parâmetro correspondente à categoria profissional do Administrador
355          * @param categoria String com a categoria profissional do Administrador
356          */
357         public void setCategoria(String categoria) {
358             this.categoria = categoria;
359         }
360 
361         /** Getter da categoria profissional do Administrador
362          * @return categoria profissional do Administrador
363          */
364         public String getCategoria() {
365             return this.categoria;
366         }
367     }
```

```java
package bib;

import java.io.Serializable;

/** Classe abstracta Artigo que representa cada item existente na Biblioteca
 * @author André Baptista (2012137523)
 * @author Rosa Faria (2005128014)
 */
public abstract class Artigo implements Serializable {
    private String titulo;
    private int anoPublicacao;
    private String editora;
    private int nCopias;
    private int nRequisitados;
    static final long serialVersionUID = -5889329237349479458L;

    /** Construtor vazio */
    public Artigo(){}

    /** Construtor de tipo que define alguns atributos do objecto
     * @param titulo Titulo da obra
     * @param ano Ano de publicação
     * @param editora Editora
     * @param nCopias Número de cópias existentes
     */
    public Artigo (String titulo, int ano, String editora, int nCopias) {
        this.titulo = titulo;
        this.anoPublicacao = ano;
        this.editora = editora;
        this.nCopias = nCopias;
    }

    /** Método para incrementar contador de número de cópias requisitadas
     * @return -1 se já não houver cópias disponíveis, 1 se a requisicão foi bem sucedida
     */
    public int addReq() {
        if (this.getnRequisitados() >= this.getnCopias()) return -1;
        this.setnRequisitados(this.getnRequisitados() + 1);
        return 1;
    }

    /** Método para decrementar contador de número de cópias requisitadas
     * @return -1 se não houver nenhuma cópia requisitada, 1 se a remoção foi bem sucedida
     */
    public int removeReq() {
        if (this.getnRequisitados() <=0) return -1;
        this.setnRequisitados(this.getnRequisitados() - 1);
        return 1;
    }

    /** Método que calcula o nº de cópias disponíveis
     * @return nº de cópias disponíveis
     */
    public int availCopy() {
        return (this.getnCopias()-this.getnRequisitados());
```

```
56      }
57
58      @Override
59      public String toString() {
60          return "Título: " + this.getTitulo() +", ano: " + this.getAnoPublicacao() +
            ", editora: " + this.getEditora() + ", número de copias: " + this.getnCopias
            ();
61      }
62
63      abstract boolean equals(Artigo a);
64
65      /** Getter para o atributo título
66       * @return título do artigo
67       */
68      public String getTitulo() {
69          return this.titulo;
70      }
71
72      /** Getter para o atributo editora
73       * @return editora do artigo
74       */
75      public String getEditora() {
76          return this.editora;
77      }
78
79      /**
80       * Getter para o atributo nº de cópias
81       * @return nº de cópias existentes do artigo
82       */
83      public int getnCopias() {
84          return this.nCopias;
85      }
86
87      /** Getter para o atributo ano de publicação
88       * @return ano de publicação
89       */
90      public int getAnoPublicacao() {
91          return anoPublicacao;
92      }
93
94      /** Getter para o atributo nº de cópias requisitadas
95       * @return nº de cópias requisitadas
96       */
97      public int getnRequisitados() {
98          return nRequisitados;
99      }
100
101     /** Setter para o atributo titulo
102      * @param titulo Titulo da obra
103      */
104     public void setTitulo(String titulo) {
105         this.titulo = titulo;
106     }
107
108     /** Setter para o atributo ano de publicação
109      * @param anoPublicacao Ano de publicação da obra
110      */
```

```
111         public void setAnoPublicacao(int anoPublicacao) {
112             this.anoPublicacao = anoPublicacao;
113         }
114 
115         /** Setter para o atributo editora
116          * @param editora Editora
117          */
118         public void setEditora(String editora) {
119             this.editora = editora;
120         }
121 
122         /** Setter para o atributo nCopias
123          * @param nCopias Nº de cópias existentes
124          */
125         public void setnCopias(int nCopias) {
126             this.nCopias = nCopias;
127         }
128 
129         /** Setter para o numero de atributos requisitados
130          * @param nRequisitados Nº de cópias requisitadas
131          */
132         public void setnRequisitados(int nRequisitados) {
133             this.nRequisitados = nRequisitados;
134         }
135     }
136 
```

```java
package bib;

import java.io.Serializable;

/**Classe que representa um autor de pelo menos uma obra incluída na Biblioteca
 * @author André Baptista (2012137523)
 * @author Rosa Faria (2005128014)
 */
public class Autor implements Serializable{
    private final String nome;
    static final long serialVersionUID = -5315745972852757639L;

    /** Construtor do tipo que define o nome do objecto
     * @param nome Nome do autor
     */
    public Autor(String nome) {
        this.nome = nome;
    }

    /** Getter para o atributo nome
     * @return nome do autor
     */
    public String getNome() {
        return this.nome;
    }

    @Override
    public String toString() {
        return this.nome;
    }

    /** Método que compara dois autores pelo nome
     * @param autor Autor a comparar com o que chamou o método
     * @return true se nomes dos autores são iguais, false caso contrário.
     */
    public boolean equals (Autor autor) {
        return this.nome.equals(autor.getNome());
    }
}

```


xi

```java
package bib;

import java.io.Serializable;
import java.text.DateFormat;
import java.text.SimpleDateFormat;
import java.util.Calendar;

/** Classe que representa datas com dia, mês e ano.
 * <p> Implementa um tratamento mais fácil de datas ignorando as unidades menores que o dia
 * e faz conversão para String das mesmas
 * @author André Baptista (2012137523)
 * @author Rosa Faria (2005128014)
 */

public class Calendario implements Serializable{
    private Calendar calendar;
    static final long serialVersionUID = 3469165778394055780L;

    Calendario(){
        this.calendar = Calendar.getInstance();
    }

    Calendario(Calendar calendar){
        this.calendar = calendar;
    }

    Calendario (int dia, int mes, int ano) {
        this.calendar = Calendar.getInstance();
        this.calendar.set(ano, mes, dia);
    }

    /** Getter para o parametro calendario
     * @return o atributo calendar do tipo Calendar
     */
    public Calendar getCalendar() {
        return calendar;
    }

    /** Método para determinar o dia do mês
     * @return dia do mês
     */
    public int getDia(){
        return this.calendar.get(Calendar.DAY_OF_MONTH);
    }

    /** Método para determinar o mês
     * @return mês
     */
    public int getMes(){
        return this.calendar.get(Calendar.MONTH)+1;
    }

    /** Método para determinar o ano
     * @return ano
     */
    public int getAno(){
```

```
57          return this.calendar.get(Calendar.YEAR);
58      }
59
60      /** Método para determinar o número de dias do mês
61       * @return número de dias do mês em questão
62       */
63      public int getMaxDias(){
64          return this.calendar.getActualMaximum(Calendar.DAY_OF_MONTH);
65      }
66
67      /** Define o valor do atributo calendar
68       * @param calendar variável do tipo Calendar
69       */
70      public void setCalendar(Calendar calendar) {
71          this.calendar = calendar;
72      }
73
74      @Override
75      public String toString(){
76          DateFormat dateFormat = new SimpleDateFormat("dd/MM/yyyy");
77          return dateFormat.format(this.calendar.getTime()).toString();
78      }
79
80      /** Compara o valor de dois calendarioss
81       * @param data Calendario com a data a comparar
82       * @return true se iguais, false caso contrário
83       */
84      public boolean equals(Calendario data) {
85          return this.calendar.equals(data.getCalendar());
86      }
87  }
88
```

```java
package bib;

import java.io.Serializable;

/** Classe que representa um Artigo do tipo DVD.
 * @author André Baptista (2012137523)
 * @author Rosa Faria (2005128014)
 */
public class DVD extends Artigo implements Serializable{
    private int duracao;
    private String realizador;
    static final long serialVersionUID = -7743761448586186551L;

    /** Construtor vazio */
    public DVD () {}

    /** Construtor do tipo que define os vários atributos do artigo
     * @param titulo Titulo da obra
     * @param ano Ano de publicação
     * @param editora Editora
     * @param nCopias Nº de cópias
     * @param duracao Duração do DVD
     * @param realizador Realizador do DVD
     */
    public DVD(String titulo, int ano, String editora, int nCopias, int duracao,
    String realizador) {
        super(titulo, ano, editora, nCopias);
        this.duracao = duracao;
        this.realizador = realizador;
    }

    /** Getter para o atributo duracao
     * @return duração do DVD
     */
    public int getDuracao() {
        return this.duracao;
    }

    /** Getter para o atributo realizador
     * @return nome do realizador
     */
    public String getRealizador() {
        return this.realizador;
    }

    /** Setter para o parâmetro duração
     * @param duracao Duração do DVD
     */
    public void setDuracao(int duracao) {
        this.duracao = duracao;
    }

    /** Setter para o parâmetro realizador
     * @param realizador Realizador do DVD
     */
    public void setRealizador(String realizador) {
        this.realizador = realizador;
```

```
57      }
58
59      @Override
60      public String toString() {
61          return "\"" + this.getTitulo() +"\" ("+ this.getAnoPublicacao() + ") de "+
            this.getRealizador()+", " + this.getDuracao()+" min";
62      }
63
64      /** Método que compara dois DVDs
65       * @param artigo Artigo a comparar
66       * @return true se forem iguais, false caso contrário
67       */
68      @Override
69      public boolean equals(Artigo artigo) {
70          DVD dvd = (DVD) artigo;
71          return (this.getTitulo().equals(dvd.getTitulo()))&&(this.getRealizador().
            equals(dvd.getRealizador()))&&(this.getAnoPublicacao() == dvd.
            getAnoPublicacao())&&(this.getEditora().equals(dvd.getEditora()))&&(this.
            getDuracao() == dvd.getDuracao());
72      }
73
74      /** Método para criar a String que vai ser guardada no ficheiro
75       * @return String a imprimir
76       */
77      public String toFile() {
78          String output=this.getTitulo()+"\n"+this.getEditora()+"\n"+this.
            getAnoPublicacao()+"\n"+this.getnCopias()+"\n"+this.getDuracao()+"\n"+this.
            getRealizador();
79          return output+"\n";
80      }
81  }
82
```

```java
package bib;

import java.io.Serializable;
import java.util.ArrayList;
import java.util.Calendar;

/** Classe que representa um Leitor.
 * <p> Com permissões para requisitar, entregar e consultar Artigos
 * @author André Baptista (2012137523)
 * @author Rosa Faria (2005128014)
 */
public class Leitor extends Utilizador implements Serializable {
    private Boolean tipo;
    private Requisicao[] requisicoes;
    private int nReqs;
    static final long serialVersionUID = -4185566377128093394L;

    /** Construtor vazio */
    public Leitor(){}

    /** Construtor do tipo que define os dados e nº de identificação do leitor
     * @param nome Nome do utilizador
     * @param morada Morada do utilizador
     * @param telefone Telefone do utilizador
     * @param email E-mail do utilizador
     * @param data Data do utilizador
     * @param pass Palavra-chave do utilizador
     * @param tipo Tipo (true - professor, false - aluno) de Leitor
     * @param nInterno Nº interno do Leitor
     */
    public Leitor(String nome, String morada, String telefone, String email, Calendar data, String pass, boolean tipo, int nInterno) {
        super(nome, morada, telefone, email, data, pass);
        this.tipo = tipo;
        this.setnInterno(nInterno);
        this.requisicoes = new Requisicao[this.getMaxReqs()];
    }

    /** Método para calcular a duração máxima de um empréstimo consoante o tipo de Leitor
     * @return 90 se for professor (tipo = true), 5 se for aluno (tipo = false)
     */
    public int getDuracao() {
        if (tipo) return 90;
        else return 5;
    }

    /** Método para calcular o número máximo de artigos a emprestar consoante o tipo de Leitor
     * @return 5 se for professor (tipo = true), 2 se for aluno (tipo = false)
     */
    public final int getMaxReqs() {
        if (tipo) return 5;
        else return 2;
    }

    /** Getter para o atributo tipo de Leitor
```

```
55          * @return true se for professor, false se for aluno
56          */
57         public boolean getTipo() {
58             return this.tipo;
59         }
60 
61         /** Getter para o atributo com a lista de requisições
62          * @return Array de requisições do utilizador
63          */
64         public Requisicao[] getRequisicoes() {
65             return requisicoes;
66         }
67 
68         /** Getter para o atributo número de requisições
69          * @return nº de requisições actuais do utilizador
70          */
71         public int getnReqs() {
72             return this.nReqs;
73         }
74 
75         /** Setter para o atributo tipo de Leitor
76          * @param tipo Tipo do Leitor (true - professor, false - aluno)
77          */
78         public void setTipo(Boolean tipo) {
79             this.tipo = tipo;
80         }
81 
82         @Override
83         public String toString() {
84             return this.getNome() +  ", "+ this.getEmail() + ", Telefone: " + this.
                getTelefone() +", Data de nascimento: " + this.getDataNascimento();
85         }
86 
87         /** Compara o Leitor this com o Leitor user.
88          * Compara os campos nome, nº interno, telefone, morada e e-mail
89          * @param user Utilizador a comparar
90          * @return true se todos forem iguais, false caso contrário
91          */
92         @Override
93         public boolean equals(Utilizador user) {
94             Leitor leitor = (Leitor) user;
95             return (this.getNome().equals(leitor.getNome()))&&(this.getnInterno() ==
                leitor.getnInterno())&&(this.getTelefone().equals(leitor.getTelefone()))&&(
                this.getMorada().equals(leitor.getMorada()))&&(this.getEmail().equals(leitor.
                getEmail()));
96         }
97 
98 
99         /** Método para utilizador entregar um artigo
100          * @param artigo Artigo a entregar
101          * @return 1 se entregar com sucesso, -1 se não for encontrada a requisição
102          */
103         public int entregar(Artigo artigo) {
104             int i = 0, j;
105 
106             //Procura a requsição que contém o artigo no ArrayList de requisições
107             while (i<this.nReqs){
```

```
108             if (this.getRequisicoes()[i].getArtigo().equals(artigo)) break;
109             i++;
110         }
111         if (i==this.nReqs) return -1;
112         j = i;
113 
114         //Remove a requisição e actualiza o ArrayList de requisições
115         this.getRequisicoes()[i].entregar();
116         for (j = i; j<nReqs-1; j++)
117             this.requisicoes[j] = this.getRequisicoes()[j+1];
118         this.requisicoes[j] = null;
119         nReqs--;
120 
121         //Coloca o artigo como não requisitado
122         artigo.removeReq();
123         return 1;
124     }
125 
126     /** Método para utilizador requisitar um artigo
127      * @param artigo Artigo a requisitar
128      * @param reqs Lista de requisições do sistema
129      * @return 1 se for bem sucedido, 0 se falhar
130      */
131     public int requisitar(Artigo artigo, ArrayList<Requisicao> reqs) {
132 
133         //Requisita caso o leitor possa requisitar mais artigos e o artigo esteja
            disponível
134         if ((this.nReqs<this.requisicoes.length)&&(artigo.availCopy()>0)) {
135             Requisicao nova = new Requisicao(this, artigo, Calendar.getInstance());
136             this.requisicoes[nReqs++] = nova;
137             reqs.add(nova);
138             artigo.addReq();
139             return 1;
140         }
141         return 0;
142     }
143 }
144 
```

```java
package bib;

import java.io.Serializable;
import java.util.ArrayList;

/** Classe que representa um Artigo do tipo Livro.
 * @author André Baptista (2012137523)
 * @author Rosa Faria (2005128014)
 */
public class Livro extends Artigo implements Serializable {
    private String cota;
    private String isbn;
    private ArrayList <Autor> autores;
    static final long serialVersionUID = -1631655505216644749L;

    /** Construtor vazio */
    public Livro(){}

    /**  Construtor do tipo que define as características do Artigo
     * @param titulo Titulo do livro
     * @param ano Ano de publicação
     * @param editora Editora
     * @param nCopias Número de cópias existentes
     * @param cota Cota
     * @param isbn ISBN
     * @param autores Lista de autores
     */


    public Livro(String titulo, int ano, String editora, int nCopias, String cota,
    String isbn, ArrayList <Autor> autores) {
        super(titulo,ano,editora,nCopias);
        this.cota = cota;
        this.isbn = isbn;
        this.autores = autores;
    }

    /** Getter para o atributo cota
     * @return cota do Livro
     */
    public String getCota() {
        return this.cota;
    }

    /** Getter para o atributo ISBN
     * @return ISBN do livro
     */
    public String getIsbn() {
        return this.isbn;
    }

    /** Getter para o atributo autores
     * @return ArrayList dos autores
     */
    public ArrayList<Autor> getAutores() {
        return this.autores;
    }
```


xix

```
57
58      /** Método para comparar o livro com outro livro
59       * @param artigo Livro a ser comparado
60       * @return true se forem iguais, false caso contrario
61       */
62      @Override
63      public boolean equals(Artigo artigo){
64          Livro livro = (Livro) artigo;
65          return ((this.getTitulo().equals(livro.getTitulo())) && (this.
            getAnoPublicacao() == livro.getAnoPublicacao()) && (this.getEditora().equals(
            livro.getEditora())) && (this.getCota().equals(livro.getCota())) && (this.
            getIsbn().equals(livro.getIsbn())));
66      }
67
68      @Override
69      public String toString() {
70          String output =  "\"" + this.getTitulo() +"\" (" + this.getAnoPublicacao() +
            ")";
71          for (int i=0;i<this.getAutores().size();i++){
72              output += "; " + this.getAutores().get(i);
73          }
74          output+=", Cota: "+this.getCota();
75          return output;
76      }
77
78
79      /** Método para criar a String que vai ser guardada no ficheiro
80       * @return String a imprimir
81       */
82      public String toFile() {
83          String output=this.getTitulo()+"\n"+this.getEditora()+"\n"+this.
            getAnoPublicacao()+"\n"+this.getnCopias()+"\n"+this.getCota()+"\n"+getIsbn()+
            "\n"+this.getAutores().size();
84          for  (int i = 0; i<this.getAutores().size();i++)
85              output+="\n"+this.getAutores().get(i);
86          return output+"\n";
87      }
88
89      /** Setter para o atributo Cota
90       * @param cota cota do Livro
91       */
92      public void setCota(String cota) {
93          this.cota = cota;
94      }
95
96      /** Setter para o atributo ISBN
97       * @param isbn ISBN do Livro
98       */
99      public void setIsbn(String isbn) {
100         this.isbn = isbn;
101     }
102
103     /** Setter para o atributo com a lista de autores
104      * @param autores ArrayList<Autor> com os autores do Livro
105      */
106     public void setAutores(ArrayList <Autor> autores) {
107         this.autores = autores;
```

```
108         }
109     }
110
```

```java
package bib;

import java.io.Serializable;
import java.util.Calendar;

/** Classe que representa uma requisição.
 * <p> Liga um leitor a um artigo, com data em que foi feita,
 * data em que é suposto entregar e data em que de facto foi entregue
 * @author André Baptista (2012137523)
 * @author Rosa Faria (2005128014)
 */
public class Requisicao implements Serializable{
    private Leitor leitor;
    private Artigo artigo;
    private Calendario dataInicio;
    private Calendario dataFim;
    private Calendario dataEntrega;
    static final long serialVersionUID = 3398822842236017323L;

    /** Construtor do tipo definindo o Leitor, artigo e datas de início e fim
     * @param leitor Leitor que fez requisição
     * @param artigo Artigo que requisita
     * @param data Data de criação da Requisição
     */
    public Requisicao(Leitor leitor, Artigo artigo, Calendar data) {
        this.leitor = leitor;
        this.artigo = artigo;
        this.dataInicio = new Calendario(data);
        this.dataFim = new Calendario((Calendar)data.clone());
        this.dataFim.getCalendar().add(Calendar.DAY_OF_MONTH,leitor.getDuracao());
    }

    /** Método para entregar artigo.
     * <p> Põe no atributo data de entrega a data de Hoje
     */
    public void entregar () {
        this.dataEntrega = new Calendario(Calendar.getInstance());
    }

    /** Getter para o atributo artigo
     * @return artigo requisitado
     */
    public Artigo getArtigo() {
        return this.artigo;
    }

    /**  Getter para a data da criação da requisição
     * @return data da requisição
     */
    public Calendario getDataInicio() {
        return this.dataInicio;
    }

    /** Getter para a data limite da requisição
     * @return data limite da requisição
     */
    public Calendario getDataFim() {
```

```
58          return this.dataFim;
59      }
60
61      /** Getter para o atributo leitor
62       * @return Leitor que fez a requisição
63       */
64      public Leitor getLeitor() {
65          return leitor;
66      }
67
68      /** Getter para o atributo data de Entrega
69       * @return data de entrega do artigo
70       */
71      public Calendario getDataEntrega() {
72          return dataEntrega;
73      }
74
75      @Override
76      public String toString() {
77          String output = "";
78          if (this.getDataEntrega() == null) output+= "Não entregue: ";
79          else output+= "Entregue: ";
80          output +=""+this.getLeitor().getNome() + " || \"" + this.artigo.getTitulo() +
             "\"";
81          if (this.getDataEntrega() == null) output+="|| Termina a "+this.getDataFim();
82          return output;
83      }
84
85  }
86
```

```java
package bib;

import java.io.Serializable;
import java.util.ArrayList;
import java.util.Calendar;

/** Classe abstracta que define os atributos de um utilizador genérico
 * @author André Baptista (2012137523)
 * @author Rosa Faria (2005128014)
 */
public abstract class Utilizador implements Serializable{

    private String nome;
    private String morada;
    private String telefone;
    private String email;
    private Calendario dataNascimento;
    private String palavraChave;
    private int nInterno;
    static final long serialVersionUID = -7403151711204682368L;

    /** Construtor vazio */
    public Utilizador(){}

    /** Construtor do tipo definindo todos os atributos à excepção do nº interno
     * @param nome Nome do utilizador
     * @param morada Morada do utilizador
     * @param telefone Telefone do utilizador
     * @param email E-mail do utilizador
     * @param data Data de nascimento do utilizador
     * @param pass Palavra-chave da conta do utilizador
     */
    public Utilizador(String nome, String morada, String telefone, String email,
    Calendar data, String pass) {
        this.nome = nome;
        this.morada = morada;
        this.telefone = telefone;
        this.email = email;
        this.dataNascimento = new Calendario(data);
        this.palavraChave = pass;
    }

    /** Método para utilizador fazer login
     * @param pass Palavra-chave inserida
     * @return true se a password inserida corresponde à definida, false caso
     contrário
     */
    public boolean entrar(String pass) {
        return this.getPalavraChave().equals(pass);
    }

    /** Getter do atributo nome
     * @return nome do utilizador
     */
    public String getNome() {
        return this.nome;
    }
```

```
56
57      /** Getter para o atributo morada
58       * @return morada do utilizador
59       */
60      public String getMorada() {
61          return this.morada;
62      }
63
64      /** Getter para o atributo telefone
65       * @return telefone do utilizador
66       */
67      public String getTelefone() {
68          return this.telefone;
69      }
70
71      /** Getter para o atributo e-mail
72       * @return e-mail do utilizador
73       */
74      public String getEmail() {
75          return this.email;
76      }
77
78      /** Getter para o atributo palavra chave
79       * @return palavra-chave do utilizador
80       */
81      public String getPalavraChave() {
82          return palavraChave;
83      }
84
85      /**
86       * Getter para o atributo nInterno
87       * @return numero interno do utilizador
88       */
89      public int getnInterno() {
90              return this.nInterno;
91      }
92
93      /** Getter para o atributo dataNascimento
94       * @return data de nascimento
95       */
96      public Calendario getDataNascimento() {
97          return dataNascimento;
98      }
99
100     /** Setter para o parâmetro morada
101      * @param morada Morada do utilizador
102      */
103     public void setMorada(String morada) {
104         this.morada = morada;
105     }
106
107     /** Setter para o parâmetro telefone
108      * @param telefone Telefone do utilizador
109      */
110     public void setTelefone(String telefone) {
111         this.telefone = telefone;
112     }
```

```
113
114         /** Setter para o parâmetro e-mail
115          * @param email E-mail do utilizador
116          */
117         public void setEmail(String email) {
118             this.email = email;
119         }
120
121         /** Setter para o atributo nome
122          * @param nome Nome do utilizador
123          */
124         public void setNome(String nome) {
125             this.nome = nome;
126         }
127
128         /** Setter para o atributo data de nascimento
129          * @param dataNascimento Data de nascimento do utilizador
130          */
131         public void setDataNascimento(Calendario dataNascimento) {
132             this.dataNascimento = dataNascimento;
133         }
134
135         /** Setter para o atributo palavra-chave
136          * @param palavraChave Palavra-chave do utilizador
137          */
138         public void setPalavraChave(String palavraChave) {
139             this.palavraChave = palavraChave;
140         }
141
142         /** Setter para o número interno
143          * @param nInterno Nº interno do utilizador
144          */
145         public void setnInterno(int nInterno) {
146             this.nInterno = nInterno;
147         }
148
149         /** Método para utilizador alterar a password
150          * @param old Palavra-Passe antiga
151          * @param pass Nova palavra-passe
152          * @return true se a palavra passe anterior está correcta, false caso contrário
153          */
154         public boolean changePass(String old, String pass) {
155             if (this.getPalavraChave().equals(old)){
156                 this.setPalavraChave(pass);
157                 return true;
158             }
159             return false;
160         }
161
162         abstract boolean equals(Utilizador user);
163
164         /** Método para determinar se algum utilizador do grupo 1 está contido no grupo 2
165          * (mesmo que seja apenas uma parte do nome)
166         **/
167         private static int compararAutores (ArrayList<Autor> grupo1, ArrayList<Autor>
        grupo2) {
168             for(int i = 0; i<grupo2.size();i++)
```

```
169             for (int j = 0; j<grupo1.size();j++) {
170                 if (grupo2.get(i).getNome().contains(grupo1.get(j).getNome())) {
171                     return 1;
172                 }
173             }
174         return 0;
175     }
176 
177     /** Método para procurar artigos comparando os parametros dados
178      * com todos os artigos do mesmo tipo
179      * @param livros ArrayList dos livros existentes na Biblioteca
180      * @param dvds ArrayList dos DVDs existentes na Biblioteca
181      * @param artigo Artigo a procurar
182      * @param parametros Lista de Strings representando os parâmetros a comparar.
183      * Parâmetros possíveis: titulo, ano, editora, cota, isbn, autores, realizador,
     duracao (dependendo do tipo de Artigo)
184      * @return ArrayList com os artigos cujos atributos coincidem ou contêm os dados
185      */
186     public ArrayList<Artigo> procurarArtigos(ArrayList<Livro> livros, ArrayList<DVD>
     dvds, Artigo artigo, String... parametros) {
187         /* Copiar todos os artigos do mesmo tipo para um array auxiliar
188          * Desse array vão sendo retirados todos os que são diferentes do inserido
189          */
190         ArrayList<Artigo> aux = new ArrayList<>();
191         if (artigo instanceof Livro)
192             for (int i = 0; i<livros.size();i++)
193                 aux.add(livros.get(i));
194         else
195             for (int i = 0; i<dvds.size();i++)
196                 aux.add(dvds.get(i));
197         for (String param: parametros)
198             /* Percorre todos os parâmetros
199              * Para cada um percorre todos os artigos do array aux e verifica se
                coincidem
200              * Se não coincidirem remove o artigo desse array
201              */
202             try {
203                 switch (param) {
204                     case "titulo":
205                         for (int i = 0; i<aux.size();i++) {
206                             if (!aux.get(i).getTitulo().contains(artigo.getTitulo()))
207                                 aux.remove(i--);
208                         }   break;
209                     case "ano":
210                         for (int i = 0; i<aux.size();i++) {
211                             if (aux.get(i).getAnoPublicacao()!=artigo.
                            getAnoPublicacao())
212                                 aux.remove(i--);
213                         }   break;
214                     case "editora":
215                         for (int i = 0; i<aux.size();i++) {
216                             if (!aux.get(i).getEditora().contains(artigo.getEditora
                            ()))
217                                 aux.remove(i--);
218                         }   break;
219                     case "cota":
220                         for (int i = 0; i<aux.size();i++) {
```

```
221                             if (!((Livro)aux.get(i)).getCota().equals(((Livro)artigo
                                ).getCota()))
222                                 aux.remove(i--);
223                         }   break;
224                     case "isbn":
225                         for (int i = 0; i<aux.size();i++) {
226                             if (!((Livro)aux.get(i)).getIsbn().equals(((Livro)artigo
                                ).getIsbn()))
227                                 aux.remove(i--);
228                         }   break;
229                     case "autores":
230                         for (int i = 0; i<aux.size();i++) {
231                             if (compararAutores(((Livro)artigo).getAutores(), ((Livro
                                )aux.get(i)).getAutores())==0)
232                                 aux.remove(i--);
233                         }   break;
234                     case "realizador":
235                         for (int i = 0; i<aux.size();i++) {
236                             if (!((DVD)aux.get(i)).getRealizador().equals(((DVD)
                                artigo).getRealizador()))
237                                 aux.remove(i--);
238                         }   break;
239                     case "duracao":
240                         for (int i = 0; i<aux.size();i++) {
241                             if (((DVD)aux.get(i)).getDuracao()!=(((DVD)artigo).
                                getDuracao()))
242                                 aux.remove(i--);
243                         }   break;
244                     default:
245                         System.out.println("Parâmetro inválido: "+param);
246                         break;
247                 }
248             }
249             catch (ClassCastException e){
250                 System.out.println("Invalid parameter: "+param);
251             }
252         return aux;
253     }
254 }
255
256
```