



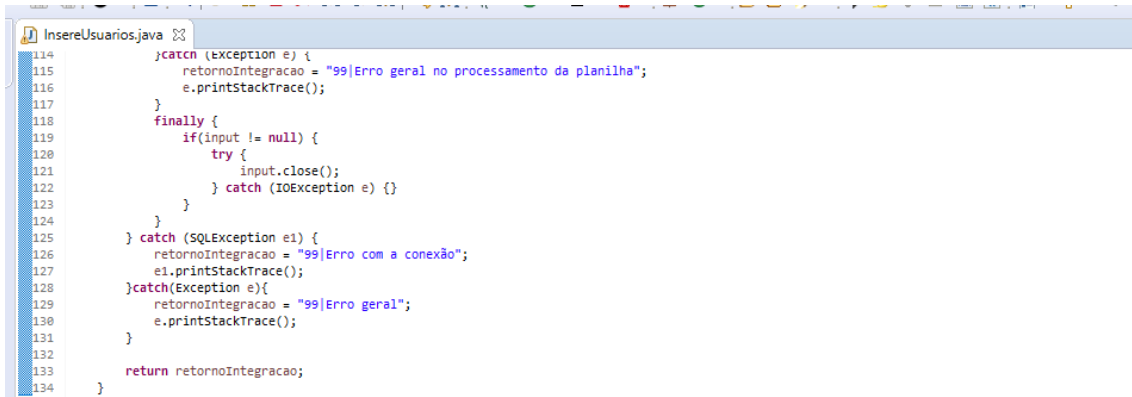
# PROJETO CUSTOMIZAÇÕES

Inserir Usuários a partir de uma planilha

Amanda Alvim

```

1 package com.lecom.workflow.integracao.passagem_etapa;
2
3 import static br.com.lecom.api.factory.ECHFactory.documento();
4
5
6
7
8
9
10 @IntegrationModule("InsererUsuario")
11 @Version(1,0,0)
12 public class InsererUsuarios {
13
14     private static final Logger logger = Logger.getLogger(InsererUsuarios.class);
15
16
17
18
19
20
21 @Execution
22 public String insererUsuario(IntegracaoVO integracaoVO) {
23
24     @SuppressWarnings("unchecked")
25     Map<String, String> parametros = integracaoVO.getMapCamposFormulario();
26
27     integracaoVO.setConexao("workflow");
28     String retornoIntegracao = "";
29     try {
30         Connection connectionWF = integracaoVO.getConexao();
31         InputStream input = null;
32         try {
33             connectionWF.setAutoCommit(false);
34
35             String anexoi = parametros.get("USUARIOS_ANEXOS");
36
37             input = getWFFilePath(anexoi);
38             Workbook wb = WorkbookFactory.create(input);
39             Sheet sheet = wb.getSheetAt(0);
40             Row row = null;
41             int l = 1; // linha 2 da planilha
42             boolean existeLinhas = true;
43             List<String> usuariosNaoCadastrados = new ArrayList<String>();
44             while (existeLinhas) { // A cada linha cria uma
45                 try {
46                     row = sheet.getRow(l);
47                     //logger.debug("row = "+row);
48                     if (row != null) {
49                         String nomCli = verificaTipoCell(row.getCell(0));
50                         if (!nomCli.equals("")) {
51                             String loginCli = verificaTipoCell(row.getCell(1));
52                             String emailCliente = verificaTipoCell(row.getCell(2));
53                             String senhaCliente = verificaTipoCell(row.getCell(3));
54                             if (!existeUsuarioEmailLogin(connectionWF, loginCli.trim(), emailCliente.trim())) {
55                                 if (!existeUsuarioLogin(connectionWF, loginCli.trim())) {
56                                     insererUsuarioBanco(connectionWF, nomCli, loginCli.trim(), senhaCliente, emailCliente, integracaoVO);
57                                     usuariosNaoCadastrados.add(" Nome = "+nomCli+" email = "+emailCliente);
58                                 }
59                             }
60                         }
61                     }
62                 }
63                 else {
64                     Long codUsuario = getCodUsuario(connectionWF, loginCli);
65                     inserirGrupo(connectionWF, codUsuario, 11);
66                 }
67             }
68             existeLinhas = false;
69         }
70         else {
71             existeLinhas = false;
72         }
73     }
74     logger.debug(" existeLinhas = "+existeLinhas);
75 }
76 catch (Exception e) {
77     retornoIntegracao = "99|Erro no processamento da planilha";
78     e.printStackTrace();
79 }
80 }
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
```



```

114     } catch (Exception e) {
115         retornoIntegracao = "99|Erro geral no processamento da planilha";
116         e.printStackTrace();
117     }
118     finally {
119         if(input != null) {
120             try {
121                 input.close();
122             } catch (IOException e) {}
123         }
124     }
125     } catch (SQLException e1) {
126         retornoIntegracao = "99|Erro com a conexão";
127         e1.printStackTrace();
128     } catch (Exception e){
129         retornoIntegracao = "99|Erro geral";
130         e.printStackTrace();
131     }
132 }
133 return retornoIntegracao;
134 }

```

Nas imagens acima é o método principal que fará a leitura da planilha e a cada linha lida será feito a inserção do usuário no banco de dados, forma essa que pode sofrer alterações no futuro para utilizações de apis do produto.

Nas **linhas 39 e 40** são as anotações para dizer que é um tipo de integração e qual a versão que está a classe, a versão nos ajuda no momento de subir o jar no Lecom BPM para assim saber que está sendo atualizada.

Na **linha 56** estamos pegando a conexão do workflow e setando um autocommit false por conta de na nossa classe utilizar inserts e updates na base por isso vamos comitar somente depois de fazer essas ações.

Na **linha 58** estamos pegando o valor do campo USUARIOS\_ANEXOS que no meu exemplo é o campo do tipo template que o usuário vai fazer o upload da planilha, e na **linha 60** eu pego o inputstream, o método getWfFilePath estará utilizando a api do ecm para transformar o nome criptografado em stream para trabalharmos com esse dado.

Nas **linhas 61 a 67** estamos utilizando a lib da poi para poder fazer a leitura dessa planilha, e assim ele instancia classes workbook, sheet, row e faz um while enquanto existir linhas para continuar lendo a planilha.

Na **linha 69** pegamos a segunda linha, por conta da inicialização da variável l na **linha 64**, e verificamos na **linha 71** se está diferente de null, após isso pegamos cada célula dessa linha e os valores, verificando no caso se existe a primeira coluna, caso já não exista o valor da primeira coluna ele já fala que não existe mais linhas e não continuará o while, caso encontre o valor da primeira coluna ele irá começar algumas validações em cima das informações da planilha como na **linha 79** que ele chama a função para validar se esse login e e-mail já existem na base de dados, caso exista ele irá somente inserir um grupo de permissão para esse usuário, caso não exista ele valida se existe o login se existir o login, esse login é gravado numa lista de String, **linha 83**, para que posteriormente seja enviado um e-mail ao iniciador com um aviso sobre esses usuários que existem porem os e-mails que vieram na planilha estão diferentes dos cadastrados, caso não exista o login, ele chama na **linha 81** a inserção do usuário no bando do dados.

Na **linha 98** está sendo tratado uma exceção que pode vir da inserção da leitura da planilha e assim ele coloca uma mensagem padrão para ser retornado na tela para o usuário.

Nas **linhas 104 a 107** ele testa se tem algum usuário preenchido que existe o login, mas o e-mail está diferente para realizar o envio do e-mail ao usuário iniciador falando essa lista de usuários.

Na **linha 109** está sendo colocado o valor retornado quando se tem sucesso da leitura da planilha, nas **linhas 112, 115, 126, 129** está sendo colocado valores de acordo com o local de tratamento do erro para no final na **linha 133** retornar para o usuário na tela onde ele caiu.

```

135
136 private String verificaTipoCell(Cell cell){
137
138     if(cell != null){
139
140         if(cell.getCellTypeEnum() == CellType.STRING) {
141             return cell.getRichStringCellValue().getString();
142         } else if(cell.getCellTypeEnum() == CellType.NUMERIC) {
143             if(DateUtil.isCellDateFormatted(cell)) {
144                 Date data = cell.getDateCellValue();
145
146                 Calendar calendar = Calendar.getInstance();
147                 calendar.setTime(data);
148
149                 return calendar.get(Calendar.DAY_OF_MONTH) + "/" + calendar.get(Calendar.MONTH) + "/" + calendar.get(Calendar.YEAR) ;
150             }
151         } else {
152             BigDecimal valor = new BigDecimal(cell.getNumericCellValue());
153
154             String retorno = String.valueOf(valor.setScale(2, RoundingMode.HALF_UP));
155
156             if(retorno.indexOf(".") > -1){
157                 retorno = retorno.substring(0, retorno.length()-3);
158             }
159
160             return retorno;
161         }
162     } else if(cell.getCellTypeEnum() == CellType.FORMULA) {
163         return cell.getCellFormula();
164     } else {
165         return "";
166     }
167 }
168 else {
169     Logger.debug("Celula vazia !");
170     return "";
171 }
172 }

```

Na imagem acima é o método `verificaTipoCell` onde nele, ele verificará se é uma String, se é numérico, se é data para formatar, e retornar o tipo de célula e o valor delas.

```

173
174 private InputStream getWfFilePath(String anexo) throws Exception {
175
176     InputStream is = null;
177
178     if (anexo != null && !"".equals(anexo)) {
179
180         // Realiza o split do valor do campo template, pois o primeiro valor @ o nome do arquivo físico e o segundo o uniqueID do documento, os dois separados por ":"
181         String[] valores = anexo.split(":");
182         String identificacaoDocumento = valores[1];
183
184         // InputStream do arquivo retornado por mtda da API do ECM
185         is = documento().lerArquivo(identificacaoDocumento);
186
187     }
188     return is;
189 }
190
191 private boolean existeUsuarioemailLogin(Connection connectionWF, String loginuser, String email){
192     String sql = "select count(*) from usuario where des_login = ? and des_email = ? ";
193     int qtd = 0;
194     try(PreparedStatement pst = connectionWF.prepareStatement(sql)){
195         pst.setString(1,loginuser);
196         pst.setString(2,email);
197         try(ResultSet rs = pst.executeQuery()){
198             if(rs.next()){
199                 qtd = rs.getInt(1);
200             }
201         }
202     } catch (SQLException e) {
203         // TODO Auto-generated catch block
204         e.printStackTrace();
205     }
206     if(qtd > 0){
207         return true;
208     } else{
209         return false;
210     }
211 }

```

Na imagem acima temos 2 métodos que são os `getWfFilePath` e o `existeUsuarioemailLogin`, o método do `getWfFilePath`, onde nesse método ele pega o valor do campo, que vem nome do arquivo:unique id do ecm, e passa o unique id do ecm para a api do ecm e retorna o `InputStream` do arquivo. No método `existeUsuarioemailLogin`, ele faz um `select count` para identificar se existe usuário com mesmo login e e-mail do que veio na planilha.

```

212
213 private boolean existeUsuarioByLogin(Connection connectionWF, String loginUser){
214     String sql = "select count(*) from usuario where des_login = ? ";
215     int qtd = 0;
216     try(PreparedStatement pst = connectionWF.prepareStatement(sql)){
217         pst.setString(1,loginUser);
218         try(ResultSet rs = pst.executeQuery()){
219             if(rs.next()){
220                 qtd = rs.getInt(1);
221             }
222         }
223     } catch (SQLException e) {
224         // TODO Auto-generated catch block
225         e.printStackTrace();
226     }
227     if(qtd > 0){
228         return true;
229     }else{
230         return false;
231     }
232 }
233
234 private Map<String,String> getDadosUsuario(Connection connectionWF, int codUsuario){
235     String sql = "select nom_usuario,des_email from usuario where cod_usuario = ? ";
236     Map<String,String> dadosUsuario = new HashMap<String,String>();
237     try(PreparedStatement pst = connectionWF.prepareStatement(sql)){
238         pst.setInt(1,codUsuario);
239         try(ResultSet rs = pst.executeQuery()){
240             if(rs.next()){
241                 dadosUsuario.put("NOME", rs.getString("nom_usuario"));
242                 dadosUsuario.put("EMAIL", rs.getString("des_email"));
243             }
244         }
245     } catch (SQLException e) {
246         // TODO Auto-generated catch block
247         e.printStackTrace();
248     }
249     return dadosUsuario;
250 }

```

Na imagem acima temos 2 métodos o **existeUsuarioByLogin** e o **getDadosUsuario**, sendo que o método **existeUsuarioByLogin**, como o explicado acima estará validando se existe um usuário com esse login que está preenchido na planilha, o método **getDadosUsuario**, retornará o nome e o email do usuário iniciador do processo, por isso ele recebe um código como parâmetro, que é passado na **linha 105** pegando a informação do objeto **IntegracaoVO**.

```

251
252
253 private Long getCodUsuario(Connection connectionWF, String loginUser){
254     String sql = "select cod_usuario from usuario where des_login = ?";
255     Long cod = 0L;
256     try(PreparedStatement pst = connectionWF.prepareStatement(sql)){
257         pst.setString(1,loginUser);
258         try(ResultSet rs = pst.executeQuery()){
259             if(rs.next()){
260                 cod = rs.getLong(1);
261             }
262         }
263     } catch (SQLException e) {
264         // TODO Auto-generated catch block
265         e.printStackTrace();
266     }
267     return cod;
268 }
269

```

Na imagem acima é o método **getCodUsuario**, ele é responsável por retornar o código do usuário do login/e-mail já existente pois conforme visto no método principal ele pega esse código para dar permissão a esse usuário a um grupo específico.

```
InserirUsuarios.java
419
271 private boolean inserirUsuarioBanco(Connection connectionWF, String nameUser, String loginUser, String senhaUser, String emailUser, IntegracaoVO integracaoVO) throws Exception{
272
273     Logger.info("INICIO da inserção de usuários!");
274
275     boolean ret = false;
276
277     StringBuilder sql = new StringBuilder();
278     sql.append(" INSERT INTO USUARIO ( ");
279     sql.append("                NOM_USUARIO, DES_LOGIN, DES_SENHA,");
280     sql.append("                IDE_ACESSO_ADM, NUMERO_SENHA, DAT_ALTERA_SENHA, DES_EMAIL, IDE_ACESSO_PESQ, ");
281     sql.append("                IDE_ACESSO_ESTAT, IDE_ACESSO_MOD, COD_SUBSTITUTO, IDE_ACESSO_EXTERNO, IDE_CAMPO_ESPECIAL, ");
282     sql.append("                COD_GRUPO_ACESSO, COD_DEPTO, COD_IDIOMA, COD_LIDER, EXPIRA_SENHA, IDE_USUARIO_INATIVO, DES_CAMPO_INTEGRACAO1 ");
283     sql.append("            ) VALUES ( ");
284     sql.append("                ?, ?, ?, ");
285     sql.append("                'N', 0, '1900-01-01 00:00:00.000', ?, 'P', ");
286     sql.append("                'N', 'N', -1, 'N', 'N', ");
287     sql.append("                1, 1, 1, ?, 1, 'N', ? ");
288     sql.append("            ) ");
289
290     try {
291
292         Integer retorno = null;
293         Long codusuario = 0L;
294         try(PreparedStatement pst = connectionWF.prepareStatement(sql.toString(), Statement.RETURN_GENERATED_KEYS)){
295             pst.setString(1, nameUser);           //NOM_USUARIO
296             pst.setString(2, loginUser);          //DES_LOGIN
297             pst.setString(3, criptografaSenha(loginUser, senhaUser)); //DES_SENHA
298             pst.setString(4, emailUser);          //DES_EMAIL
299             pst.setString(5, "1");               //COD_LIDER
300
301             retorno = pst.executeUpdate();
302
303             ResultSet rs = pst.getGeneratedKeys();
304             if (rs.next()) {
305                 codusuario = rs.getLong(1);
306             }
307         }
308         Logger.debug(" codusuario = "+codusuario+" retorno = "+retorno);
309         if(codusuario > 0){
310
311             ret = true;
312
313             String updateLider = "update usuario set cod_lider = ? where cod_usuario = ?";
314             try(PreparedStatement pstupdate = connectionWF.prepareStatement(updateLider)){
315                 pstupdate.setLong(1, codusuario);
316                 pstupdate.setLong(2, codusuario);
317
318                 pstupdate.executeUpdate();
319             }
320
321             inserirGrupo(connectionWF, codusuario, 1); // inserindo grupo AcessoBPM
322
323             //enviaEmailUsuario(nameUser, loginUser, emailUser, senhaUser, integracaoVO);
324
325             connectionWF.commit();
326         }
327     }
328
329     } catch (Exception e) {
330         e.printStackTrace();
331         Logger.error("erro", e);
332         try {
333             connectionWF.rollback();
334         } catch (SQLException e1) {
335             // TODO Auto-generated catch block
336             e1.printStackTrace();
337         }
338         throw new Exception();
339     }
340
341     Logger.info("FIM da inserção de usuários!");
342
343     return ret;
344 }
345 }
```

Nas imagens acima é o método que fará a inserção do usuário diretamente na base de dados, ele monta um insert nas **linhas 278 a 288**, com os campos e informações padrões que precisamos passar para cadastrar esse usuário por fora.

Na **linha 298** é chamado o método para criptografar a senha do usuário, nesse caso ele precisa do login e da senha escolhida para fazer a criptografia.

Nesse método é usado algumas coisas diferentes ao preparar a query e recuperar a informação do código usuário inserido no banco de dados.

Na **linha 311** é verificado se o código do usuário é maior zero, significando que foi inserido com sucesso e assim realizamos um update para informar que o líder será ele mesmo nesse momento.

Após isso na **linha 323** está sendo chamada função para inserir um grupo padrão para esse usuário.

Nesse método inteiro vemos também a questão dos rollbacks ou commit da conexão, levando em consideração o momento adequado para fazer tal chamadas.

```

348 private void insereGrupo(Connection connectionIF, Long codUsuario, int codGrupo) throws SQLException {
349     Logger.debug("insere grupo codUser = "+codUsuario+" grupo = "+codGrupo);
350     StringBuilder sqlGrupoAberturaChamado = new StringBuilder();
351     sqlGrupoAberturaChamado.append(" INSERT INTO grupo_usuario(COD_GRUPO, COD_USUARIO) ");
352     sqlGrupoAberturaChamado.append(" VALUES(?, ?) ");
353
354     try(PreparedStatement pstGrupo = connectionIF.prepareStatement(sqlGrupoAberturaChamado.toString());{
355         pstGrupo.setLong(1, codGrupo); //COD_GRUPO
356         pstGrupo.setLong(2, codUsuario); //COD_USUARIO
357         pstGrupo.executeUpdate();
358     }
359 }
360
361 private void enviaEmailUsuarioIniciador(Map<String,String> dadosUsuarios,List<String> usuariosNaoCadastrados,IntegracaoVO integracaoVO){
362     Logger.debug(" enviaEmailUsuarioIniciador ");
363     StringBuilder mensagem = new StringBuilder();
364     mensagem.append(" Prezado "+dadosUsuarios.get("NOME")+",<br/><br/> ");
365     mensagem.append(" Temos alguns usuários que não foram inseridos na base pois o login já existem na base porém com emails diferentes.<br/> ");
366     mensagem.append(" Por favor crie outra planilha com outros logins para usuários abaixo citado:<br/><br/> ");
367     for (String string : usuariosNaoCadastrados) {
368         mensagem.append(string+"<br/>");
369     }
370
371
372
373     try {
374         List<String> to = new ArrayList<>();
375         to.add(dadosUsuarios.get("EMAIL"));
376         Logger.debug(" enviaEmailUsuarioIniciador de = "+integracaoVO.getDesFrom());
377         String assunto = "[Inserção Usuários] - Falhas";
378         EmailMessage emailMessage = new EmailMessage(assunto, mensagem.toString(), integracaoVO.getDesFrom(),to, true);
379         Logger.debug(" enviaEmailUsuarioIniciador emailMessage = "+emailMessage);
380         integracaoVO.enviaEmailMessage(emailMessage);
381     } catch (Exception e) {
382         Logger.debug(" enviaEmailUsuarioIniciador -> ERRO ");
383         e.printStackTrace();
384     }
385
386     Logger.debug(" enviaEmailUsuarioIniciador -> fim ");
387 }
388

```

Na imagem acima temos 2 métodos **insereGrupo** e o **enviaEmailUsuarioIniciador**, o método do **insereGrupo**, nesse exemplo ele está fazendo o insert direto utilizando o código do usuário inserido e um código de grupo que vocês teriam que passar. No método do **enviaEmailUsuarioIniciador** ele está montando uma mensagem fixa para o usuário falando que os usuários existem na base, porém não com mesmo email que está vindo na planilha.

```

375     Logger.debug(" enviaEmailUsuarioIniciador de = "+integracaoVO.getDesFrom());
376     String assunto = "[Inserção Usuários] - Falhas";
377     EmailMessage emailMessage = new EmailMessage(assunto, mensagem.toString(), integracaoVO.getDesFrom(),to, true);
378     Logger.debug(" enviaEmailUsuarioIniciador emailMessage = "+emailMessage);
379     integracaoVO.enviaEmailMessage(emailMessage);
380 } catch (Exception e) {
381     Logger.debug(" enviaEmailUsuarioIniciador -> ERRO ");
382     e.printStackTrace();
383 }
384
385     Logger.debug(" enviaEmailUsuarioIniciador -> fim ");
386 }
387
388
389
390 private String criptografaSenha(String login, String senha) throws Exception {
391     String retorno = "";
392     String aux = login.trim() + "\uA12C\u067B" + senha.trim();
393
394     Md5 md5 = new Md5(aux);
395     md5.getDigest();
396
397     retorno = md5.getStringDigest() + geraAleatorio(8); //adicionando um "sujeira" na string já criptografada
398
399     return retorno;
400 }
401
402 private String geraAleatorio(int tamanho) {
403     Random random = new Random();
404     int i = 0;
405     int iHex = 0;
406     String sSaida = "";
407
408     for (i = 0; i < tamanho; i++) {
409         iHex = random.nextInt(16);
410         sSaida += Integer.toHexString(iHex).toString();
411     }
412     return sSaida;
413 }
414 }

```

Na imagem acima temos os 2 métodos finais da classe, **criptografaSenha** e **geraAleatorio**, esse último é chamado pelo primeiro, a criptografia é usada a md5 para gerar uma informação binaria dessa senha e assim essa informação é que é gravada no banco de dados.