PROJETO CUSTOMIZAÇÕES

Alteração de Data Limite

Amanda Alvim

1. Objetivo

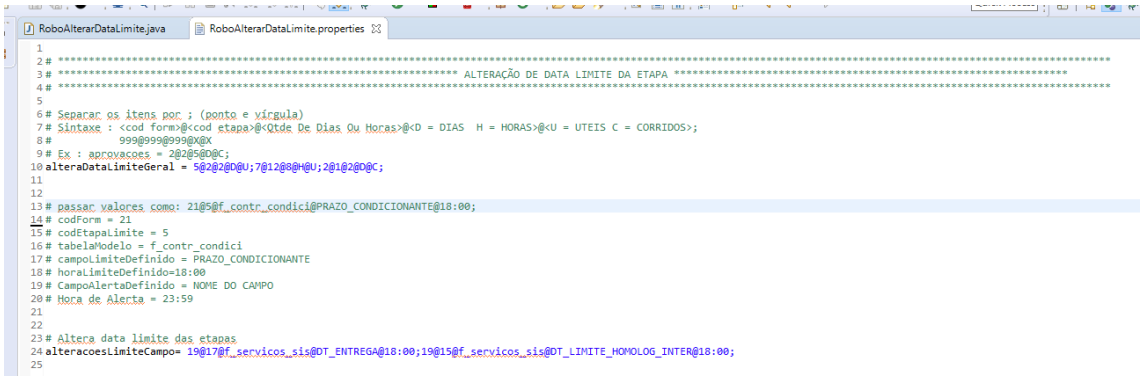
O objetivo dessa customização é realizar alteração da data limite do processo configurado, de vez de ser configurada pelo studio para aquela atividade, ela precisa ser dinâmica, então esse robô ajudará a fazer isso de forma fácil.

O robô contendo essa implementação se encontra no Projeto Base no nosso git (http://git.lecom.com.br/PSP/Projeto-Base-BPM) com o nome de **RoboAlterarDataLimite**, que está no package com.lecom.workflow.robo.exemplos, para o funcionamento dele será utilizado um properties do mesmo nome **RoboAlterarDataLimite.properties** que está no diretório upload/cadastros/config.

2. Como usar

No momento de utilizar essa classe você precisará colocar o properties com as informações do seu(s) processo(s) no diretório do servidor do cliente /opt/lecom/app/tomcat/webapps/bpm/upload/cadastros/config para ambientes com o modo antigo de atualização das customizações ou no diretório /opt/lecom/custom/web-content/config para modo novo de atualização e depois utilizar o jar gerado desse robô que está no diretório JAR do projeto base e subir no seu ambiente, ele já está gerado para ajudar na utilização como ele usa várias classes externas este jar já contempla tudo junto.

3. Properties



O properties acima contém 2 parâmetros para que possam utilizar de maneira genérica, vamos entender o que dever ser colocado em cada um desses parâmetros:

O parâmetro **alteraDataLimiteGeral**, consiste em realizar alteração da data limite do processo para as etapas configuradas sendo que pode ser somado dias ou horas, corridas ou uteis a mais da data de início dessa atividade, as informações que precisa passar separadas por @ são:

- Código do formulário
- Código da etapa que será alterado
- Quantidade de dias ou horas
- Informativo se é Dia ou hora
- Informativo se é corrido ou uteis

Ficando no final a configuração por exemplo: 5@2@2@D@U;5@4@5@H@C;

O outro parâmetro **alteracoesLimiteCampo** consiste em realizar alteração de data limite do processo/etapa a partir de um campo do modelo, as informações que precisa ser passado nesse parâmetro separadas por @ são:

- Código do Formulário
- Código da Etapa que será alterado
- Tabela do modelo
- Campo do modelo como base
- Horário limite

Ficando no final a configuração por exemplo: 5@8@f_teste@DATA_LIMITE_PARAM@18:00.

4. Fonte JAVA

RoboAlterarDataLimite.java

```
27
28  @RobotModule("RoboAlterarDataLimite")
29  @Version({1,0,0})
30  public class RoboAlterarDataLimite implements WebServices{
31
32      private static final Logger logger = Logger.getLogger(RoboAlterarDataLimite.class);
33      private String configPath = Funcoes.getWFRootDir() + "/upload/cadastros/config/";
34
35
36      @Execution
37      public void executaAlteracoesDataLimite(){
38
39          logger.debug("Inicio Robo executaAlteracoesDataLimite");
40          try {
41
42              RetornaInformacoesBPM retornaDadosBaseBPM = new RetornaInformacoesBPM();
43              Map<String,String> parametros = Funcoes.getParametrosIntegracao(configPath +"RoboAlterarDataLimite");
44              String alteracoesLimite = parametros.get("alteraDataLimiteGeral");
45              if(!alteracoesLimite.equals("")) {
46                  alteraDataLimiteDias(retornaDadosBaseBPM, alteracoesLimite);
47              }
48
49              String alteracoesLimiteCampo = parametros.get("alteracoesLimiteCampo");
50              if(!alteracoesLimiteCampo.equals("")) {
51                  alteraDataLimiteCampo(retornaDadosBaseBPM, alteracoesLimiteCampo);
52              }
53
54              logger.debug("Fim Robo executaAlteracoesDataLimite");
55
56          } catch (Exception e) {
57
58              logger.error("[ERRO] : ", e);
59              e.printStackTrace();
60
61          }
62      }
```

Acima temos a imagem do fonte com o seu método principal, onde nas **linhas 32 a 33** temos a inicialização da variável de log, e o caminho de onde vai pegar o arquivo de configuração, o properties explicado acima.

Na **linha 42** estamos inicializando a classe **RetornaInformacoesBPM**, ela que retornará uns dados importantes do banco de dados para conseguirmos fazer a alteração da data limite considerando os feriados, e horários de trabalho cadastrado no sistema do cliente.

Nas **linhas 44 a 47** estamos pegando o valor do parâmetro **alteraDataLimiteGeral** e chamamos o método específico que tratará cada informação do parâmetro e fará a ação de alteração da data limite.

Nas **linhas 49 a 52** estamos pegando o valor do parâmetro **alteracoesLimiteCampo** e chamamos o método específico que tratará cada informação do parâmetro e fará a ação de alteração da data limite.

RoboAlterarDataLimite.java

```
64  private void alteraDataLimiteCampo(RetornaInformacoesBPM retornaDadosBaseBPM, String alteracoesLimite)
65          throws Exception, SQLException {
66      logger.debug("alteraDataLimiteCampo= "+alteracoesLimite);
67      String[] alteracoes = alteracoesLimite.split(";");
68
69      for (String alteracao : alteracoes) {
70          logger.debug("alteracao= "+alteracao);
71          String[] camposAlteracaoLimite = alteracao.split("@");
72
73          String codForm = camposAlteracaoLimite[0];
74          String codEtapaLimite = camposAlteracaoLimite[1];
75          String tabelaModelo = camposAlteracaoLimite[2];
76          String campoLimiteDefinido = camposAlteracaoLimite[3]; //campo PRAZO_CONDICIONANTE
77          String horaLimiteDefinido = Funcoes.nulo(camposAlteracaoLimite[4],"");
78
79          Calendar dataGravacao = Calendar.getInstance();
80          Calendar dataLimite = Calendar.getInstance();
81          Map<String, Map<String, String>> turnoSemanaMap = null;
82          List<String> diasNaoTrabalhadosList = null;
83
84          try(Connection con = DBUtils.getConnection("workflow")){
85
86              StringBuilder sqlVerificaSLA = new StringBuilder();
87              sqlVerificaSLA.append(" SELECT ");
88              sqlVerificaSLA.append("     p.COD_PROCESSO, p.COD_ETAPA_ATUAL, p.COD_CICLO_ATUAL, pe.dat_gravacao, pe.DAT_LIMITE, pe.DAT_FINALIZACAO, pe.VLR_ATRASO1 ");
89              sqlVerificaSLA.append(" ,f."+campoLimiteDefinido);
90              sqlVerificaSLA.append(" from  processo p , processo_etapa pe, "+tabelaModelo+" f ");
91              sqlVerificaSLA.append(" where cod_form = ? ");
92              sqlVerificaSLA.append(" and f.COD_PROCESSO_F = p.COD_PROCESSO");
93              sqlVerificaSLA.append(" and f.COD_ETAPA_F = p.COD_ETAPA_ATUAL");
94              sqlVerificaSLA.append(" and f.COD_CICLO_F = p.COD_CICLO_ATUAL");
95              sqlVerificaSLA.append(" and p.COD_PROCESSO = pe.COD_PROCESSO");
96              sqlVerificaSLA.append(" and p.COD_ETAPA_ATUAL = pe.COD_ETAPA");
97              sqlVerificaSLA.append(" and p.COD_CICLO_ATUAL = pe.COD_CICLO");
98              sqlVerificaSLA.append(" and p.COD_ETAPA_ATUAL = ? ");
99              sqlVerificaSLA.append(" and pe.DAT_LIMITE is null ");
100             sqlVerificaSLA.append(" AND PE.IDE_STATUS = 'A' ");
```

RoboAlterarDataLimite.java

```
103                 turnoSemanaMap = new RetornaInformacoesBPM().getTurnoSemanaMapBanco(con);
104
105                 // Recupera os dias não trabalhados por ser feriado
106                 diasNaoTrabalhadosList = new RetornaInformacoesBPM().getDiasNaoTrabalhadosList(con);
107
108                 logger.debug("sqlVerificaSLA = "+sqlVerificaSLA.toString());
109                 try(PreparedStatement pst = con.prepareStatement(sqlVerificaSLA.toString())){
110
111                     pst.setString(1, codForm);
112                     pst.setString(2, codEtapaLimite);
113
114                     try(ResultSet rs = pst.executeQuery()){
115
116                         while(rs.next()){
117
118                             logger.debug("Inicio while resultset");
119                             String codProcesso = rs.getString("cod_processo");
120                             String codEtapa = rs.getString("cod_etapa_atual");
121                             String codCiclo = rs.getString("cod_ciclo_atual");
122                             Timestamp dataGravacaoEtapaAtual = rs.getTimestamp("dat_gravacao");
123
124                             Date dataLimiteDefinido = rs.getDate(campoLimiteDefinido.trim());
125                             if(dataLimiteDefinido != null){
126
127                                 logger.debug("dataLimiteDefinido = "+dataLimiteDefinido);
128                                 logger.debug("horaLimiteDefinido = "+horaLimiteDefinido);
129                                 Calendar dataParam = Calendar.getInstance();
130                                 dataParam.setTime(dataLimiteDefinido);
131
132                                 if(!horaLimiteDefinido.equals("")){
133                                     String[] hora = horaLimiteDefinido.split(":");
134                                     dataParam.set(Calendar.HOUR_OF_DAY,Integer.parseInt(hora[0]));
135                                     dataParam.set(Calendar.MINUTE,Integer.parseInt(hora[1]));
136                                 }
137
138                                 //dataLimite.setTimeInMillis(dataParam.getTime() + 28800000);
139                                 dataLimite = dataParam;
140                                 dataGravacao.setTimeInMillis(dataGravacaoEtapaAtual.getTime());
```

RoboAlterarDataLimite.java

```
142                             dataLimite = getDiaUtil(dataLimite, diasNaoTrabalhadosList);
143
144                             long  atraso1 = new CalculaTempoAtraso(dataLimite, dataGravacao, turnoSemanaMap, diasNaoTrabalhadosList).getTotalEmMilisegundos();
145
146                             logger.debug("codProcesso = "+codProcesso);
147                             logger.debug("codEtapa = "+codEtapa);
148                             logger.debug("codCiclo = "+codCiclo);
149                             logger.debug("campoLimiteDefinido = "+campoLimiteDefinido+" dataLimiteDefinido = "+dataLimiteDefinido);
150                             logger.debug("dataLimite = "+dataLimite.getTime());
151                             logger.debug("atraso1 = "+atraso1);
152
153                             // Update para configuração do SLA na etapa
154                             StringBuilder sqlConfiguraSLA = new StringBuilder();
155                             sqlConfiguraSLA.append(" UPDATE ");
156                             sqlConfiguraSLA.append("     PROCESSO_ETAPA ");
157                             sqlConfiguraSLA.append(" SET ");
158                             sqlConfiguraSLA.append("     VLR_ATRASO1 = ? ");
159                             sqlConfiguraSLA.append("     , DAT_LIMITE = ? ");
160                             sqlConfiguraSLA.append("     , DAT_FINALIZACAO = ? ");
161                             sqlConfiguraSLA.append(" WHERE ");
162                             sqlConfiguraSLA.append("     COD_PROCESSO = ? ");
163                             sqlConfiguraSLA.append("     AND COD_ETAPA = ? ");
164                             sqlConfiguraSLA.append("     AND COD_CICLO = ? ");
165
166                             try (PreparedStatement pstConfiguraSLA = con.prepareStatement(sqlConfiguraSLA.toString());) {
167
168                                 // Caso seja o primeiro ciclo
169                                 pstConfiguraSLA.setLong(1, atraso1);
170                                 pstConfiguraSLA.setTimestamp(2, new Timestamp(dataLimite.getTimeInMillis()));
171                                 pstConfiguraSLA.setTimestamp(3, new Timestamp(dataLimite.getTimeInMillis()));
172                                 pstConfiguraSLA.setString(4, codProcesso);
173                                 pstConfiguraSLA.setString(5, codEtapa);
174                                 pstConfiguraSLA.setString(6, codCiclo);
175                                 pstConfiguraSLA.executeUpdate();
176                             }
177
178                         } else {
179                             logger.debug("Campo definido para data limite está com valor nulo");
180                         }
```

Acima temos o método **alteraDataLimiteCampo** onde ele receberá o objeto da classe **RetornoInformacoesBPM** e o valor do atributo **alteracoesLimiteCampo**, assim começamos a realizar o split do ; (ponto e vírgula), e para cada registro dentro do for é feito o split do @, com isso nas **linhas 73 a 77** estão sendo atribuídos às variáveis respectivas.

Nas **linhas 86 a 100** estamos fazendo um select buscando todos os processos daquele formulário parados naquela atividade e que NÂO tenham data limite configurada, e nas colunas de retorno pegamos a do campo em questão passado nos parâmetros.

Nas **linhas 103 e 106** estamos retornando os valores do banco de dados relacionados com os turnos de trabalho desse ambiente, e os dias não trabalhados, os feriados tudo que tiver cadastrado no ambiente isso interfere diretamente na data limite.

Vamos começar a trabalhar em cima de cada processo/etapa/ciclo que foi retornado, primeiramente atribuímos as informações que vem no select para atributos pegando corretamente o que é String, Timestamp e Date.

Caso tenha um valor de data definido no campo passado de parâmetro (validação feita na **linha 125**).

Nas **linhas 129 e 130** estamos inicializando o objeto Calendar com o getInstance que pega o valor da data atual, porém abaixo nós estamos setando o valor desse calendar para a data limite que veio do campo configurado no properties.

Nas **linhas 132 a 136** validamos se existe um valor de hora configurado no properties e se tiver setamos essa informação no objeto calendar instanciado acima.

Na **linha 139** estamos atribuindo o valor desse objeto calendar configurado acima para a variável que instanciamos antes na **linha 80**.

Na **linha 140** estamos atribuindo para variável que foi instanciada na **linha 79,** o valor da data de gravação da atividade, ou seja, a data inicial dela campo esse que foi retornado do banco na **linha 122.**

Na **linha 142** estamos chamando o método **getDiaUtil** nele irá ser feito a validação se a data escolhida é uma data útil, ou seja, não cai nas situação de algum feriado cadastrado no sistema.

Na **linha 144** utilizamos uma outra classe útil **CalculaTempoAtraso,** que internamente usa as mesmas formas de cálculo do próprio produto, para ela vamos passar as informações da data limite , a que acabamos de retornar na **linha 142** , passamos a data de gravação, os turnos cadastrados no sistema e os dias não trabalhados cadastrados no sistema, dessa classe chamamos o método **getTotalEmMilisegundos,** que nos retornará a diferença entre a data inicial com a data limite configurada em milissegundos formato esse que precisamos para alterar a data no banco de dados.

Nas **linhas 154 a 176** está sendo feito o update na tabela do produto passando as informações de data limite, precisamos preencher os campos vlr_atraso1, dat_limite e dat_finalizacao, para o processo/etapa/ciclo e assim funciona o mecanismo de atraso que o Lecom BPM já tem. Em breve nas próximas versões teremos api do produto para isso então não será mais feito esse update no banco

RoboAlterarDataLimite.java

```
189
190    public Calendar getDiaUtil(Calendar data, List<String> diasNaoTrabalhadosList){
191
192        logger.debug(" getDiaUtil ");
193        Calendar retorno = Calendar.getInstance();
194
195        try {
196
197            while(validaData(data,diasNaoTrabalhadosList)){
198                logger.debug(" validando data = "+data.getTime());
199                data = retornaData(data,diasNaoTrabalhadosList);
200                logger.debug(" retornaData data = "+data.getTime());
201            }
202
203            retorno = data;
204
205        } catch (Exception e) {
206
207            logger.error("[ERRO] : ", e);
208            throw e;
209        }
210
211        return retorno;
212    }
```

Na imagem acima temos o método **getDiaUtil** onde nela chamaremos outros dois métodos para fazer a validação da data se ela faz parte dos dias uteis e não feriados desse ambiente, e assim vai somando dias até que encontra o próximo dia útil, por isso realiza as chamadas nos métodos **validaData** e **retornaData** que iremos ver abaixo o conteúdo deles.

RoboAlterarDataLimite.java

```
213
214    public boolean validaData(Calendar data,List<String> diasNaoTrabalhadosList){
215
216        logger.debug(" validaData = "+data.getTime());
217        boolean validacaoData = false;
218        int diaSemana = data.get(Calendar.DAY_OF_WEEK);
219
220        if(diaSemana==1){ //Domingo
221            validacaoData = true;
222
223        } else if(diaSemana==7){ //Sábado
224            validacaoData = true;
225
226        } else {
227            SimpleDateFormat dateFormat     = new SimpleDateFormat("dd/MM/yyyy");
228            String sDataAux = dateFormat.format(data.getTime());
229            if(diasNaoTrabalhadosList.contains(sDataAux)) {
230                validacaoData = true;
231            }
232        }
233
234        logger.debug(" validacaoData = "+validacaoData);
235        return validacaoData;
236    }
```

Na imagem acima temos o método **validaData**, ele está validando se a data é final de semana ou se ela contém dentro dos dias não trabalhados cadastrados no sistema, enquanto ele retornar true, o método do **getDiaUtil** continuará chamando o método **retornaData.**

RoboAlterarDataLimite.java

```
237
238    public Calendar retornaData(Calendar data,List<String> diasNaoTrabalhadosList){
239
240        logger.debug(" retornaData = "+data.getTime());
241        Calendar dataNova = Calendar.getInstance();
242        int diaSemana = data.get(Calendar.DAY_OF_WEEK);
243
244        if(diaSemana==1){ //Domingo
245            data.add(Calendar.DAY_OF_MONTH, 1);
246
247        } else if(diaSemana==7){ //Sábado
248            data.add(Calendar.DAY_OF_MONTH, 2);
249        }
250
251        SimpleDateFormat dateFormat     = new SimpleDateFormat("dd/MM/yyyy");
252        String sDataAux = dateFormat.format(data.getTime());
253        if(diasNaoTrabalhadosList.contains(sDataAux)) {
254            data.add(Calendar.DAY_OF_MONTH, 1);
255        }
256
257        dataNova = data;
258        logger.debug(" retornaData  dataNova = "+dataNova.getTime());
259        return dataNova;
260    }
```

Na imagem acima o método **retornaData** ele verifica se o dia da data é final de semana e se contem na listagem de dias não trabalhados ele adiciona um dia a essa data (**linha 254)**, e volta para o método **getDiaUtil** até que o método **validaData** retorne false assim saberemos que estamos com uma data em dia útil e usamos essa data como parâmetro para atualizar o processo/etapa/ciclo.

RoboAlterarDataLimite.java

```
261
262    private void alteraDataLimiteDias(RetornaInformacoesBPM retornaDadosBaseBPM, String alteracoesLimite)
263            throws Exception, SQLException {
264        logger.debug("alteraDataLimiteDias= "+alteracoesLimite);
265        String[] alteracoes = alteracoesLimite.split(";");
266
267        for (String alteracao : alteracoes) {
268            logger.debug("alteracao= "+alteracao);
269            String[] camposAlteracaoLimite = alteracao.split("@");
270
271            String codForm         = camposAlteracaoLimite[0]; //Código do formulário
272            String codEtapaAlterar = camposAlteracaoLimite[1]; //Etapa
273            String QtdeDiasHoras   = camposAlteracaoLimite[2]; //Quantidade de dias ou horas à adicionar
274            String identificadorDH = camposAlteracaoLimite[3]; //Identificador para somar(D = Dias, H = Horas)
275            String identificadorUC = camposAlteracaoLimite[4]; //Identificador dias (U = Uteis, C = Corridos)
276
277            Calendar dataGravacaoEtapa = Calendar.getInstance();
278            Calendar dataLimite = Calendar.getInstance();
279            Map<String, Map<String, String>> turnoSemanaMap = null;
280            List<String> diasNaoTrabalhadosList = null;
281            long atraso1 = 0;
282            try(Connection con = DBUtils.getConnection("workflow")){
283
284                StringBuilder sqlVerificaSLA = new StringBuilder();
285                sqlVerificaSLA.append("SELECT PE.COD_PROCESSO, PE.COD_ETAPA, PE.COD_CICLO, PE.DAT_GRAVACAO, ");
286                sqlVerificaSLA.append("PE.DAT_LIMITE, PE.DAT_FINALIZACAO, PE.VLR_ATRASO1  ");
287                sqlVerificaSLA.append("FROM  PROCESSO_ETAPA PE ");
288                sqlVerificaSLA.append("WHERE PE.COD_PROCESSO IN (SELECT P.COD_PROCESSO FROM PROCESSO P ");
289                sqlVerificaSLA.append("                                WHERE P.COD_PROCESSO = PE.COD_PROCESSO ");
290                sqlVerificaSLA.append("                                  AND P.COD_FORM = ?)   ");
291                sqlVerificaSLA.append("and PE.COD_ETAPA = ?               ");
292                sqlVerificaSLA.append("and PE.DAT_LIMITE IS NULL            ");
293                sqlVerificaSLA.append("AND PE.IDE_STATUS = 'A'");
294
295                // Recupera o turno semanal cadastrado no BPM
296                turnoSemanaMap = retornaDadosBaseBPM.getTurnoSemanaMapBanco(con);
297
298                // Recupera os dias não trabalhados por ser feriado
299                diasNaoTrabalhadosList = retornaDadosBaseBPM.getDiasNaoTrabalhadosList(con);
```

RoboAlterarDataLimite.java

```
301             logger.debug("sqlVerificaSLA = "+sqlVerificaSLA.toString());
302             try(PreparedStatement pst = con.prepareStatement(sqlVerificaSLA.toString())){
303
304                 pst.setString(1, codForm);
305                 pst.setString(2, codEtapaAlterar);
306
307                 try(ResultSet rs = pst.executeQuery()){
308
309                     while(rs.next()){
310
311                         logger.debug("Inicio while resultset");
312                         String codProcesso              = rs.getString("cod_processo");
313                         String codEtapa                 = rs.getString("cod_etapa");
314                         String codCiclo                 = rs.getString("cod_ciclo");
315                         Timestamp dataGravacaoEtapaAtual = rs.getTimestamp("dat_gravacao");
316
317                         if(identificadorUC.equals("U")){
318                             dataGravacaoEtapa.setTimeInMillis(dataGravacaoEtapaAtual.getTime());
319                             dataLimite = new CalculaTempoLimite(dataGravacaoEtapa, 01, turnoSemanaMap, diasNaoTrabalhadosList)
320                                     .getCalculaTempoAtrasoSegundos(Long.parseLong(QtdeDiasHoras), identificadorDH);
321
322                             atraso1 = new CalculaTempoAtraso(dataLimite, dataGravacaoEtapa, turnoSemanaMap, diasNaoTrabalhadosList).getTotalEmMilisegundos();
323                         }else{
324                             Calendar dataParam = Calendar.getInstance();
325                             dataParam.setTime(dataGravacaoEtapaAtual);
326
327                             if(!QtdeDiasHoras.equals("") && identificadorDH.equals("H")){
328                                 dataParam.add(Calendar.HOUR_OF_DAY,Integer.parseInt(QtdeDiasHoras));
329                             }else if (!QtdeDiasHoras.equals("") && identificadorDH.equals("D")){
330                                 dataParam.add(Calendar.DAY_OF_MONTH,Integer.parseInt(QtdeDiasHoras));
331                             }
332
333                             dataLimite = dataParam;
334                             dataGravacaoEtapa.setTimeInMillis(dataGravacaoEtapaAtual.getTime());
335                             atraso1 =  dataLimite.getTimeInMillis() - dataGravacaoEtapa.getTimeInMillis();
336                         }
337
```

RoboAlterarDataLimite.java

```
337
338                             logger.debug("codProcesso = "+codProcesso);
339                             logger.debug("codEtapa = "+codEtapa);
340                             logger.debug("codCiclo = "+codCiclo);
341                             logger.debug("dataGravacao apos tempoatraso = "+dataGravacaoEtapa.getTime());
342                             logger.debug("Dias ou Horas à somar = "+ QtdeDiasHoras);
343                             logger.debug("Somar dias ou horas = " + identificadorDH);
344                             logger.debug("Somar dias Uteis ou corridos = " + identificadorUC);
345                             logger.debug("dataLimite  apos tempoatraso = "+dataLimite.getTime());
346                             logger.debug("atraso1 = "+ atraso1);
347
348                             // Update para configuração do SLA na etapa
349                             StringBuilder sqlConfiguraSLA = new StringBuilder();
350                             sqlConfiguraSLA.append(" UPDATE ");
351                             sqlConfiguraSLA.append("    PROCESSO_ETAPA ");
352                             sqlConfiguraSLA.append(" SET ");
353                             sqlConfiguraSLA.append("     VLR_ATRASO1 = ? ");
354                             sqlConfiguraSLA.append("     , DAT_LIMITE = ? ");
355                             sqlConfiguraSLA.append("     , DAT_FINALIZACAO = ? ");
356                             sqlConfiguraSLA.append(" WHERE ");
357                             sqlConfiguraSLA.append("    COD_PROCESSO = ? ");
358                             sqlConfiguraSLA.append("    AND COD_ETAPA = ? ");
359                             sqlConfiguraSLA.append("    AND COD_CICLO = ? ");
360
361                             try (PreparedStatement pstConfiguraSLA = con.prepareStatement(sqlConfiguraSLA.toString());) {
362                                 // Caso seja o primeiro ciclo
363                                 pstConfiguraSLA.setLong(1, atraso1);
364                                 pstConfiguraSLA.setTimestamp(2, new Timestamp(dataLimite.getTimeInMillis()));
365                                 pstConfiguraSLA.setTimestamp(3, new Timestamp(dataLimite.getTimeInMillis()));
366                                 pstConfiguraSLA.setString(4, codProcesso);
367                                 pstConfiguraSLA.setString(5, codEtapa);
368                                 pstConfiguraSLA.setString(6, codCiclo);
369                                 pstConfiguraSLA.executeUpdate();
370                             }
371                         }
372                     }
373                 }
374             }
```

Nas imagens acima temos o método **alteraDataLimiteDias**, que receberá o objeto da classe RetornaInformacoesBPM e o valor do atributo alteraDataLimiteGeral do properties.

A partir desse valor é feito o split do ; e do @ para pegar cada parâmetro separadamente, e isso é feito nas **linhas 271 a 275**.

Nas **linhas 284 a 293** estamos fazendo um select que busca todos os processos parados naquela atividade para aquele formulário e que não tenha data limite configurada pelo produto.

Com isso para cada processo/etapa/ciclo retornado vamos começar a validar os parâmetros que temos configurado a partir da **linha 317.**

Na **linha 317** validamos se o último parâmetro é útil ou corrido, se for útil, começamos a setar o valor para o atributo instanciado na **linha 277** com o valor da data inicial dessa atividade, variável essa que veio como retorno do banco na **linha 315.**

Na **linha 319** estamos instanciando outra classe útil que usamos chamada **CalculaTempoLimite** passando para ela a data de gravação, a de início da atividade, o tempo

de atraso, os turnos cadastrados no sistema, os dias não trabalhados cadastros no sistema, e disso chamamos o método **getCalculaTempoAtrasoSegundos** passando quantidade de dias ou de horas que deve ser adicionada a data de gravação e o parâmetro se é hora ou dia. Esse método nos retornará a data limite já fazendo os devidos cálculos em cima das configurações do ambiente.

Na **linha 320** faremos a chamada da classe **CalculaTempoAtraso** passando a data limite, a data da gravação, os turnos cadastros no sistema, os dias não trabalhado e chamamos o método **getTotalEmMilisegundos**, que nos retornara em milissegundos a diferença entra a data inicial e data limite.

A partir da **linha 324** vamos fazer configuração para ter a data limite e o tempo em milissegundos a partir dos parâmetros que são corridos. Para isso na **linha 327** é validado se tem o valor para a quantidade de dias ou horas, e se é uma hora, se for na linha abaixo é chamado o método add do calendar para adicionar a quantidade de horas passada para a data inicial do processo, data essa que foi instanciada para serem feitos esses cálculos nas **linhas 324 e 325** , caso não foi hora usará o outro tipo para adicionar , adicionando os dias.

Após isso temos as duas informações a data limite, e a data da gravação da atividade, pegamos os milissegundos de cada data e pegamos a diferença entre eles. Com esses dados utilizaremos mais abaixo para realizar o update.

Nas **linhas 349 a 370** está sendo feito o update direto na tabela do processo para atualizar essa informação, lembrando que em versões futuras teremos a api para isso e esse método mudará para não fazer mais update direto no banco.