



# PROJETO CUSTOMIZAÇÕES

RotasBPM

Amanda Alvim

## 1. Objetivo

O objetivo desse documento é explicar todas as possibilidades de uso da lib RotasBPM para auxiliar nas execuções, aberturas, listagem de documentos, cancelamentos, login por fora da ferramenta via programação JAVA.

O jar se encontra no Projeto Base no nosso git (<http://git.lecom.com.br/PSP/Projeto-Base-BPM>) no diretório libs/RotasBBPM.jar.

## 2. Como usar

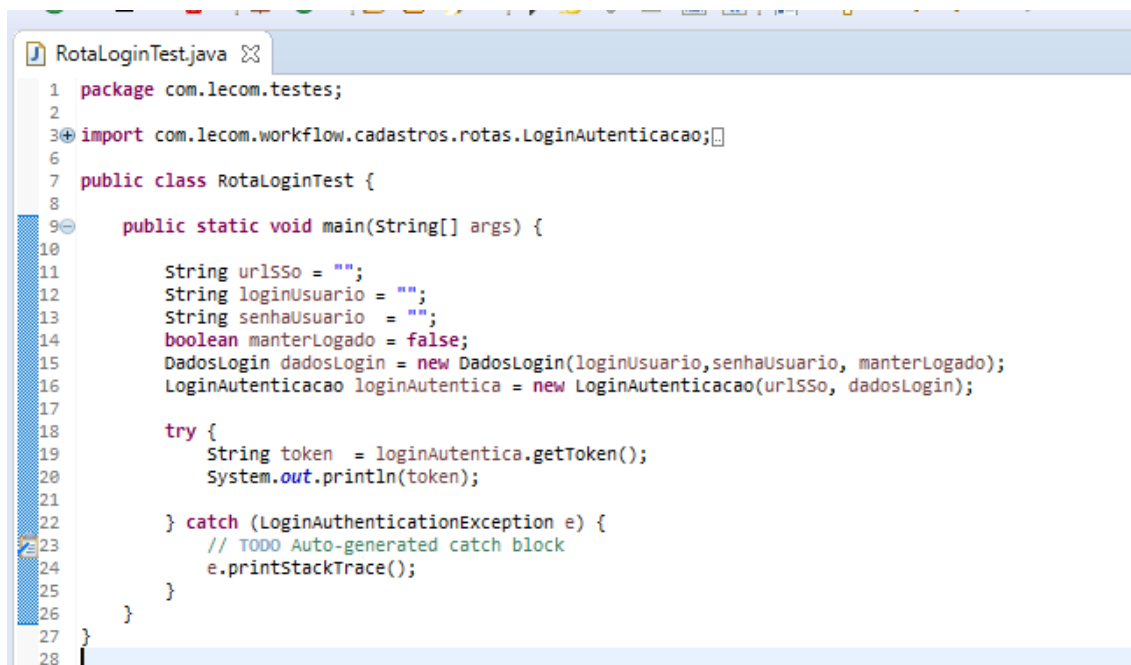
No momento que você for utilizar os recursos contidos nessa lib você irá copiar a lib para o seu projeto e adicionar no build path do projeto do eclipse, dessa forma nas suas classes já irá conseguir utilizar os recursos que iremos explicar nesse documento.

Para utilizar no ambiente de um cliente, você precisará colocar essa lib no servidor do cliente, ficando no caminho /opt/lecom/app/tomcat/webapps/bpm/WEB-INF/lib para ambientes anteriores a nova forma de atualização das customizações e na nova forma fica no caminho /opt/lecom/custom/libs, após colocar essa lib em um desses caminhos é necessário restart do ambiente para que seus fontes consigam encontrar os recursos que ela possui.

## 3. Rotas

### a. Login

O objetivo dessa rota é realizar o login por fora da ferramenta internamente está sendo usado uma chamada da api do produto, para encapsular e cada um não realizar a chamada de um jeito foi feito esse encapsulamento a partir das chamadas da classe como vamos ver abaixo:



```
1 package com.lecom.testes;
2
3 import com.lecom.workflow.cadastros.rotas.LoginAutenticacao;
4
5
6
7 public class RotaLoginTest {
8
9     public static void main(String[] args) {
10
11         String urlSSo = "";
12         String loginUsuario = "";
13         String senhaUsuario = "";
14         boolean manterLogado = false;
15         DadosLogin dadosLogin = new DadosLogin(loginUsuario, senhaUsuario, manterLogado);
16         LoginAutenticacao loginAutentica = new LoginAutenticacao(urlSSo, dadosLogin);
17
18         try {
19             String token = loginAutentica.getToken();
20             System.out.println(token);
21
22         } catch (LoginAuthenticationException e) {
23             // TODO Auto-generated catch block
24             e.printStackTrace();
25         }
26     }
27 }
28
```

Esse arquivo **RotaLoginTest.java** está no Projeto Base mencionado no começo do documento no pacote de com.lecom.testes onde nesse pacote estará arquivos que podem ser rodados como Java Application e assim ver localmente o resultado da sua chamada testando os parâmetros que você usará no seu robô, integração, controller por exemplo.

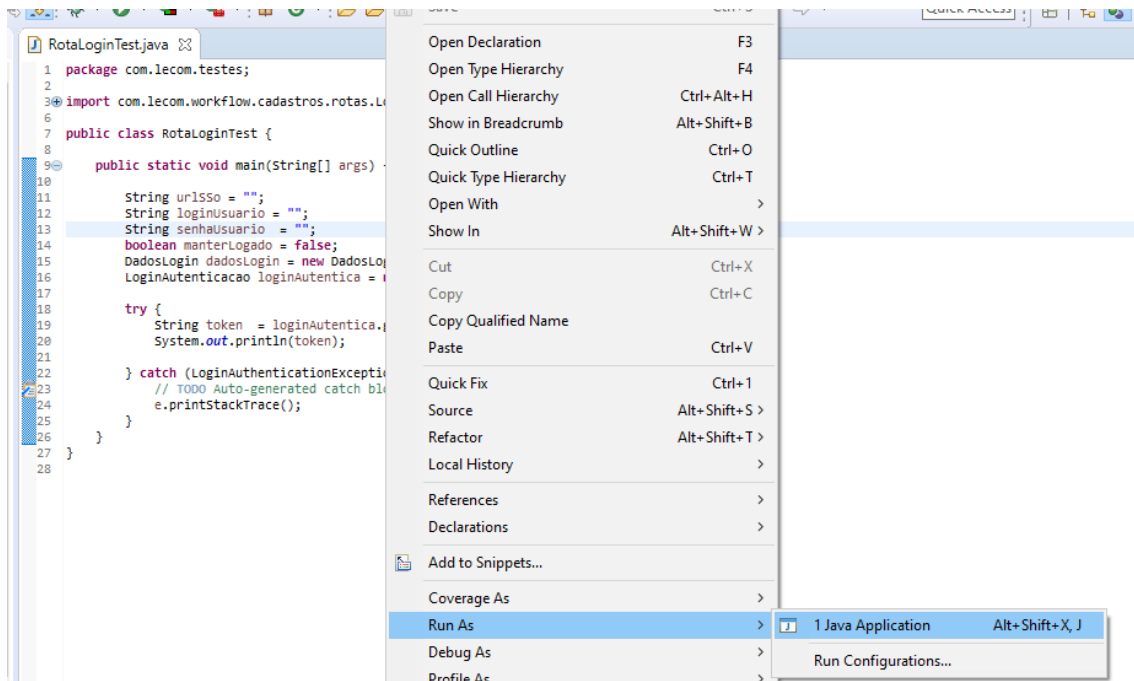
Para realizar o login precisamos instanciar duas classes e passar os parâmetros corretamente, primeira a classe a ser instanciada é a **DadosLogin**, para ela precisamos passar os seguintes parâmetros no construtor da classe:

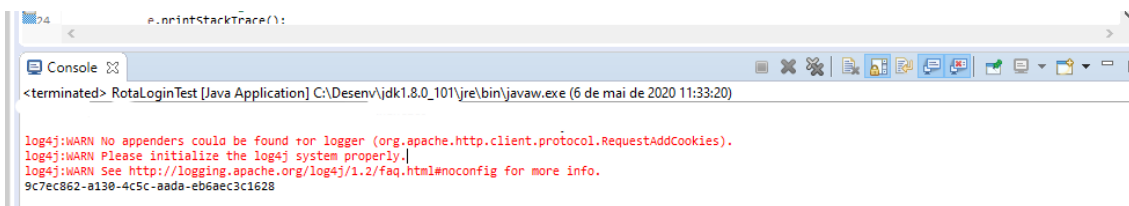
- **loginUsuario**: login do usuário que você deseja fazer por fora, geralmente utilizamos usuário automático para isso.
- **senhaUsuario**: senha do usuário que você deseja fazer por fora, geralmente utilizamos senha do usuário automático para isso, pois nós que criamos ele e temos essa senha para usar.
- **manterLogado**: esse parâmetro se refere como se fosse a tela inicial do Lecom BPM onde você escolher se quer ficar logado, geralmente utilizamos o false mesmo para não ter problema de ficar com um token muito tempo e inválido depois.

Precisaremos também instanciar a classe **LoginAutenticacao** passando parâmetros no construtor dela e após isso chamar o método de getToken ele que nos retornará o token a ser utilizado nas outras chamadas, os parâmetros a ser passado no construtor são:

- **urlSSO**: url do ambiente do cliente que você utilizará realizar o login por fora, a url precisa ser com a barra no final escrito SSO.
- **dadosLogin**: o objeto instanciado e preenchido acima precisa ser passado nesse construtor.

Após isso utilizamos um try/catch para tratar a exceção da realização do login, por isso usamos a classe exception de **LoginAutenticacaoException** ela irá retornar uma mensagem caso a chamada do método **getToken** executar com falhas, isso pode ocorrer caso passe a url errada, login errado, senha errada, caso não tenha falhas você usará o retorno que é uma String com o token de acesso desse usuário logado.





```
e.printStackTrace();

<terminated> RotaLoginTest [Java Application] C:\Desenv\jdk1.8.0_101\jre\bin\javaw.exe (6 de mai de 2020 11:33:20)

log4j:WARN No appenders could be found for logger (org.apache.http.client.protocol.RequestAddCookies).
log4j:WARN Please initialize the log4j system properly.
log4j:WARN See http://logging.apache.org/log4j/1.2/faq.html#noconfig for more info.
9c7ec862-a138-4c5c-aada-eb6aec3c1628
```

Na imagem acima temos primeiro como realiza a execução local, você precisa clicar dentro da classe com botão direito do mouse ir no menu Run as e escolher a opção Java Application assim ele executará abrindo essa aba de console abaixo e retornando no caso por conta dessa classe ter um `system.out.println` do token ele printa na tela o token retornado.

## b. Abertura Processo

O objetivo dessa rota é realizar abertura da instancia do processo por fora da ferramenta, isso pode ser utilizado em robôs que podem ficar monitorando alguma ação externa do cliente e realizar a abertura daquela instância, podem ser usado em uma integração que assim conseguiria abrir uma instancia de um processo filho por exemplo ficando assim um subprocesso. Vamos ver abaixo como que realizamos essa abertura utilizando a lib do Rotas BPM, lembrando que internamente está sendo feita uma chamada para api do produto sendo feito um encapsulamento para que não houvesse consumos diferentes dessa api em cada desenvolvimento.



```
RotaAberturaProcesso.java
1 package com.lecom.testes;
2
3 import com.lecom.workflow.cadastros.rotas.AbreProcesso;
4
5
6
7
8
9
10
11 public class RotaAberturaProcesso {
12
13     public static void main(String[] args) {
14         try {
15             String urlSso = "";
16             String loginUsuario = "";
17             String senhaUsuario = "";
18             boolean manterLogado = false;
19             DadosLogin dadosLogin = new DadosLogin(loginUsuario, senhaUsuario, manterLogado);
20             LoginAutenticacao loginAutentica = new LoginAutenticacao(urlSso, dadosLogin);
21
22             String token = loginAutentica.getToken();
23             System.out.println("token = "+token);
24
25             String urlBPM = "";
26             String codigoFormulario = "";
27             String codigoversao = "";
28             String modoTeste = "";
29             String codigousuarioIniciador = "";
30             DadosProcesso dadosProcesso = null;
31
32             AbreProcesso abreProcesso = new AbreProcesso(urlBPM, token, codigoFormulario, codigoversao, modoTeste, codigousuarioIniciador, dadosProcesso);
33             DadosProcessoAbertura aberturaUtil = abreProcesso.getAbreProcesso();
34
35             System.out.println(aberturaUtil.getProcessInstanceId());
36             System.out.println(aberturaUtil.getCurrentActivityInstanceId());
37             System.out.println(aberturaUtil.getCurrentCycle());
38
39         } catch (LoginAuthenticationException e) {
40             e.printStackTrace();
41         } catch (AbreProcessoException e) {
42             e.printStackTrace();
43         }
44     }
45 }
46
47
48
```

Na imagem acima temos o print do arquivo **RotaAberturaProcesso.java** ele está no Projeto Base mencionado no começo do documento no pacote de `com.lecom.testes` onde nesse pacote estará arquivos que podem ser rodados como Java Application e assim ver localmente o resultado da sua chamada testando os parâmetros configurados.

Para realizar a abertura do processo precisará realizar o login do usuário primeiro conforme visto no tópico acima e a partir do token gerado conseguiremos instanciar a classe **AbreProcesso** passando os seguintes parâmetros no construtor dela:

- urlBPM: url do ambiente do cliente que deseja utilizar para realizar a abertura do processo, passando a url com o final bpm
- token: o token retornado da rota do login

- codigoFormulario: o código do formulário que deseja abrir por fora, informação que você consegue pegar na listagem de modelos no menu STUDIO
- codigoVersao: o código da versão do formulário que deseja abrir por fora, informação que você consegue pegar na listagem de modelos no menu STUDIO
- modoTeste: precisa informar se esse formulário será aberto em modo teste, passando true, ou se será aberto em modo normal, passando false
- codigoUsuarioIniciador: caso seja aberto em modo teste é necessário informar o código do usuário que será utilizado para iniciar a instancia desse processo, geralmente utilizamos o mesmo código do usuário logado, caso seja aberto em modo normal somente passar null nesse parâmetro
- dadosProcesso: instancia de um outro objeto que veremos afundo no próximo tópico a utilização e preenchimento desse objeto, nesse exemplo passamos null que significa que abriremos uma instancia do processo configurado sem preenchimento automático de nenhum campo por fora.

Após isso é utilizado a chamada do método **getAbreProcesso**, e ele nos retornará um objeto da classe DadosProcessoAbertura, e ela nos retornará o código da instancia aberta, o código da atividade inicial e o código do ciclo caso dê sucesso se não irá cair na exceção específica AbreProcessoException e assim na tela aparecerá a mensagem que a api do bpm nos retorna, nesse caso erro poder ser url, código do formulário, versão errada, pode ser que o modelo não está publicado no Studio, o usuário que você logou não tem permissão para abrir e outros.



```
<terminated> RotaAberturaProcesso [Java Application] C:\Desenv\jdk1.8.0_101\jre\bin\javaw.exe (6 de mai de 2020 12:08:09)

log4j:WARN No appenders could be found for logger (org.apache.http.client.protocol.RequestAddCookies).
log4j:WARN Please initialize the log4j system properly.
log4j:WARN See http://logging.apache.org/log4j/1.2/faq.html#noconfig for more info.
token = 6afe3156-6ed9-4a80-b3ff-4aeeaac2755a
4107
1
1
```

Após clicar na classe com botão direito do mouse, e escolher o menu Run -> Java Application, o retorno na aba console será conforme imagem acima, mostrando as informações retornadas dos system.out.println que tem no print anterior da classe como um todo, então ele printa na tela o token, código da instancia, código da atividade e ciclo aberta.

### c. Aprova Processo

O objetivo dessa rota é realizar as aprovações automáticas por fora da ferramenta, como se estivéssemos clicando no botão de aprovar para seguir o roteamento conforme configurado no processo, lembrando que internamente está sendo feita uma chamada para api do produto sendo feito um encapsulamento para que não houvesse consumos diferentes dessa api em cada desenvolvimento.

```

RotaAprovaProcesso.java
8 import com.lecom.workflow.cadastros.rotas.util.DadosProcesso;
9 import com.lecom.workflow.cadastros.rotas.util.DadosProcessoAbertura;
10
11 public class RotaAprovaProcesso {
12
13     public static void main(String[] args) {
14         try {
15             String urlSSO = "";
16             String loginUsuario = "";
17             String senhaUsuario = "";
18             boolean manterLogado = false;
19             DadosLogin dadosLogin = new DadosLogin(loginUsuario, senhaUsuario, manterLogado);
20             LoginAutenticacao loginAutentica = new LoginAutenticacao(urlSSO, dadosLogin);
21
22             String token = loginAutentica.getToken();
23             System.out.println("token = " + token);
24
25             String urlBPM = "";
26
27             DadosProcesso dadosProcesso = new DadosProcesso("P");
28
29             String codProcesso = "";
30             String codEtapa = "";
31             String codCiclo = "";
32             String modoTeste = "";
33             String codUsuario = "";
34             DadosProcessoAbertura dadosProcessoAbertura = new DadosProcessoAbertura();
35             dadosProcessoAbertura.setProcessInstanceId(codProcesso);
36             dadosProcessoAbertura.setCurrentActivityInstanceId(codEtapa);
37             dadosProcessoAbertura.setCurrentCycle(codCiclo);
38
39             AprovaProcesso aprovaProcesso = new AprovaProcesso(urlBPM, token, dadosProcessoAbertura, dadosProcesso, modoTeste, codUsuario);
40             String retorno = aprovaProcesso.aprovaProcesso();
41             System.out.println(retorno);
42
43         } catch (LoginAuthenticationException e) {
44             e.printStackTrace();
45         } catch (AprovaProcessoException e) {
46             e.printStackTrace();
47         }
48     }
49 }
50
51 }

```

Na imagem acima temos o print do arquivo **RotaAprovaProcesso.java** ele está no Projeto Base mencionado no começo do documento no pacote de `com.lecom.testes` onde nesse pacote estará arquivos que podem ser rodados como Java Application e assim ver localmente o resultado da sua chamada testando os parâmetros configurados.

Para realizar a aprovação do processo precisará realizar o login do usuário primeiro conforme visto no tópico acima e a partir do token gerado conseguiremos instanciar a classe **AprovaProcesso** passando os seguintes parâmetros no construtor dela:

- urlBPM: url do ambiente do cliente que deseja utilizar para realizar a abertura do processo, passando a url com o final bpm
- token: o token retornado da rota do login
- dadosProcessoAbertura: é o objeto retornado na chamada da rota de AbreProcesso, porem para esse exemplo nós iremos instanciar e preencher o objeto para apenas aprovar a instancia, esse objeto precisa ser preenchido as informações de código do processo, da atividade e do ciclo conforme as linhas 34 a 37
- dadosProcesso: instancia de um outro objeto que veremos mais afundo em outro exemplo que acoplará tudo que estamos vendo, nele nós iremos preencher os dados de campos a serem preenchidos automaticamente no momento da aprovação e nesse objeto é onde falamos qual a ação que será feita no construtor da classe (linha 27) passaremos P , se quisermos que seja aprovado o processo e R se quisermos que seja rejeitado o processo.
- modoTeste: precisa informar se esse formulário será aberto em modo teste, passando true, ou se será aberto em modo normal, passando false
- codigoUsuario: caso seja aberto em modo teste é necessário informar o código do usuário que será utilizado para iniciar a instancia desse processo, geralmente utilizamos o mesmo código do usuário logado, caso seja aberto em modo normal somente passar null nesse parâmetro

Após isso iremos chamar o método **aprovaProcesso**, retornando uma String que irá vir o valor OK caso tenha sucesso na chamada se não cairá na exceção específica para aprovação que é a **AprovaProcessoException**, onde mostrará em tela o erro retornado pela api, no caso pode ser algum campo obrigatório que não foi preenchido, usuário que logou não tem permissão ao processo, atividade e ciclo passado e outros.

```

Console
<terminated> RotaAprovaProcesso [Java Application] C:\Desenv\jdk1.8.0_101\jre\bin\javaw.exe (6 de mai de 2020 12:46:09)

log4j:WARN No appenders could be found for logger (org.apache.http.client.protocol.RequestAddCookies).
log4j:WARN Please initialize the log4j system properly.
log4j:WARN See http://logging.apache.org/log4j/1.2/faq.html#noconfig for more info.
token = 005af8a6-6d40-4fae-b2bd-74a305ecef2
url = process-instances/4061/activity-instances/2/cycles/2
key = ticket-sso value=005af8a6-6d40-4fae-b2bd-74a305ecef2
key = test-mode value=false
JsonFinalPost = {"action":"P","values":[],"datagridCommands":[]}
OK

```

Após clicar na classe com botão direito do mouse, e escolher o menu Run -> Java Application, o retorno na aba console será conforme imagem acima, mostrando as informações retornadas dos system.out.println que tem no print anterior da classe como um todo, então ele printa na tela o token, a url chamada do bpm, apenas no print deixei aparecendo código do processo, atividade e ciclo para preservar ambiente utilizado de teste, mostra o que foi enviado para conseguir realizar a aprovação, no caso ticket sso e ticket mode (false, não está no modo teste) e os valores de campos a serem preenchidos nesse caso nenhum e apenas ação, embaixo mostra OK, que significa a atividade foi aprovada corretamente.

#### d. Salvar Processo

O objetivo dessa rota é realizar um salvamento automático do processo, atividade, ciclo informado, como se tivesse na tela do processo e clicasse no disquete para salvar aquelas informações, lembrando que internamente está sendo feita uma chamada para api do produto sendo feito um encapsulamento para que não houvesse consumos diferentes dessa api em cada desenvolvimento.

```

RotaSalvarProcesso.java RotaLoginTest.java RotaAprovaProcesso.java
import com.lecom.workflow.cadastros.rotas.util.DadosProcessoAbertura;
import com.lecom.workflow.cadastros.rotas.util.DadosProcesso;

10
11 public class RotaSalvarProcesso {
12
13     public static void main(String[] args){
14         try {
15
16             String urlSso = "";
17             String loginUsuario = "";
18             String senhaUsuario = "";
19             boolean manterLogado = false;
20             DadosLogin dadosLogin = new DadosLogin(loginUsuario, senhaUsuario, manterLogado);
21             LoginAutenticacao loginAutentica = new LoginAutenticacao(urlSso, dadosLogin);
22             String token = loginAutentica.getToken();
23
24             String urlBpm = "";
25             String codProcesso = "";
26             String codEtapa = "";
27             String codCiclo = "";
28             String modoTeste = "";
29             String codUsuario = null;
30
31             DadosProcessoAbertura dadosProcessoAbertura = new DadosProcessoAbertura();
32             dadosProcessoAbertura.setProcessInstanceId(codProcesso);
33             dadosProcessoAbertura.setCurrentActivityInstanceId(codEtapa);
34             dadosProcessoAbertura.setCurrentCycle(codCiclo);
35
36             DadosProcesso dadosProcesso = new DadosProcesso("S");
37
38             SalvaProcesso salva = new SalvaProcesso(urlBpm, token, dadosProcessoAbertura, dadosProcesso, modoTeste, codUsuario);
39             String retorno = salva.salvaProcesso();
40             System.out.println(retorno);
41
42         } catch (LoginAuthenticationException e) {
43             // TODO Auto-generated catch block
44             e.printStackTrace();
45         } catch (SalvaProcessoException e) {
46             // TODO Auto-generated catch block
47             e.printStackTrace();
48         }
49     }
50 }
51

```

Na imagem acima temos o print do arquivo **RotaSalvarProcesso.java** ele está no Projeto Base mencionado no começo do documento no pacote de `com.lecom.testes` onde nesse pacote estará arquivos que podem ser rodados como Java Application e assim ver localmente o resultado da sua chamada testando os parâmetros configurados.

Para realizar o salvamento do processo precisará realizar o login do usuário primeiro conforme visto no tópico acima e a partir do token gerado conseguiremos instanciar a classe **SalvarProcesso** passando os seguintes parâmetros no construtor dela:

- **urlBPM**: url do ambiente do cliente que deseja utilizar para realizar a abertura do processo, passando a url com o final bpm
- **token**: o token retornado da rota do login
- **dadosProcessoAbertura**: é o objeto retornado na chamada da rota de AbreProcesso, porem para esse exemplo nós iremos instanciar e preencher o objeto para apenas aprovar a instancia, esse objeto precisa ser preenchido as informações de código do processo, da atividade e do ciclo conforme as linhas 31 a 34
- **dadosProcesso**: instancia de um outro objeto que veremos mais afundo em outro exemplo que acoplará tudo que estamos vendo, nele nós iremos preencher os dados de campos a serem preenchidos automaticamente no momento da aprovação e nesse objeto é onde falamos qual a ação que será feita no construtor da classe (linha 36) passaremos S para que seja feito o salvamento dos dados.
- **modoTeste**: precisa informar se esse formulário será aberto em modo teste, passando true, ou se será aberto em modo normal, passando false
- **codUsuario**: caso seja aberto em modo teste é necessário informar o código do usuário que será utilizado para iniciar a instancia desse processo, geralmente utilizamos o mesmo código do usuário logado, caso seja aberto em modo normal somente passar null nesse parâmetro



Após isso iremos chamar o método **salvarProcesso**, retornando uma String que irá vir o valor OK caso tenha sucesso na chamada se não cairá na exceção específica para salvamento que é a **SalvarProcessoException**, onde mostrará em tela o erro retornado pela api, podendo ser algum parâmetro errado que foi passado acima.

```

<terminated> RotaSalvarProcesso [Java Application] C:\Desenv\jdk1.8.0_101\jre\bin\javaw.exe (6 de mai de 2020 14:18:12)

log4j:WARN No appenders could be found for logger (org.apache.http.client.protocol.RequestAddCookies).
log4j:WARN Please initialize the log4j system properly.
log4j:WARN See http://logging.apache.org/log4j/1.2/faq.html#noconfig for more info.
url = ticket-sso value=7e157dde-eb8f-43f1-855b-116c4bdf6856 process-instances/3765/activity-instances/1/cycles/1,
key = test-mode value=false
JsonFinalPost = {"action":"S","values":[],"datagridCommands":[]}
OK

```

Após clicar na classe com botão direito do mouse, e escolher o menu Run -> Java Application, o retorno na aba console será conforme imagem acima, mostrando as informações retornadas dos system.out.println que tem no print anterior da classe como um todo, a url chamada do bpm, apenas no print deixei aparecendo código do processo, atividade e ciclo para preservar ambiente utilizado de teste, mostra o que foi enviado para conseguir realizar o salvamento, no caso ticket sso e ticket mode (false, não está no modo teste o processo passado) e os valores de campos a serem preenchidos nesse caso nenhum e apenas ação, embaixo mostra OK, que significa a atividade foi salva corretamente.

## e. Listar Documentos a partir de um template

O objetivo dessa rota é trazer todos os documentos anexados para um determinado template.

```

RotaListarDocumento.java  RotaLoginTest.java  GeradorPDF.java
1 package com.lecom.testes;
2
3 import java.util.List;
4
11
12 public class RotaListarDocumento {
13     public static void main(String[] args) {
14         try {
15
16             String urlSso = "";
17             String loginUsuario = "";
18             String senhaUsuario = "";
19             boolean manterLogado = false;
20             DadosLogin dadosLogin = new DadosLogin(loginUsuario, senhaUsuario, manterLogado);
21             LoginAutenticacao loginAutentica = new LoginAutenticacao(urlSso, dadosLogin);
22
23             String urlECM = "";
24             String idTemplate = "";
25             String page = "0";
26             int qtdRegistro = 50;
27
28             DocumentosECM documentosECM = new DocumentosECM(urlECM, loginAutentica.getToken(), idTemplate, page, qtdRegistro);
29             List<DadosDocumento> dados = documentosECM.getDocumentosByTemplate();
30             dados.forEach(k -> System.out.println(k.getFilename()+"-"+k.getFileUniqueId()));
31             //DadosDocumento info = dados.stream().filter(x -> "247".equals(x.getDocumentId())).findAny().get();
32         } catch (LoginAuthenticationException e1) {
33             // TODO Auto-generated catch block
34             e1.printStackTrace();
35         } catch (DocumentosECMException e) {
36             // TODO Auto-generated catch block
37             e.printStackTrace();
38         }
39     }
40 }
41
42

```

Na imagem acima temos o print do arquivo **RotaListarDocumento.java** ele está no Projeto Base mencionado no começo do documento no pacote de com.lecom.testes onde nesse pacote estará arquivos que podem ser rodados como Java Application e assim ver localmente o resultado da sua chamada testando os parâmetros configurados.

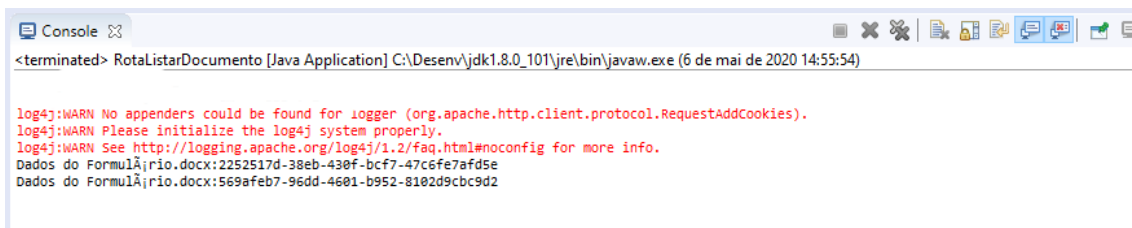
Para realizar a listagem de documentos a partir de um template precisará realizar o login do usuário primeiro conforme visto no tópico acima e a partir do token gerado

conseguiremos instanciar a classe **DocumentosECM** passando os seguintes parâmetros no construtor dela:

- urlECM: url do ambiente do cliente que deseja utilizar para realizar a abertura do processo, passando a url com o final ecmcore
- idTemplate: o id do template que deseja trazer os documentos anexados
- page: passamos o número da página que queremos que retorne, geralmente passamos 0
- qtdRegistro: passamos quantidade máxima que queremos de retorno da listagem de documento, podemos por exemplo passar 50

Após isso chamaremos o método **getDocumentosByTemplate** e caso dê alguma falha na chamada ele apresentara um erro na exceção específica para documento **DocumentosECMException**, caso dê sucesso retornará uma List<DadosDocumento> e assim você pode listar as informações que deseja esse objeto DadosDocumento tem os seguintes atributos:

- FileUniqueld: a informação de único id do ECM
- fileName: nome do arquivo importado
- documentId: id do documento
- author: usuário que realizou o import desse documento
- version: versão do documento



```

Console
<terminated> RotaListarDocumento [Java Application] C:\Desenv\jdk1.8.0_101\jre\bin\javaw.exe (6 de mai de 2020 14:55:54)

log4j:WARN No appenders could be found for logger (org.apache.http.client.protocol.RequestAddCookies).
log4j:WARN Please initialize the log4j system properly.
log4j:WARN See http://logging.apache.org/log4j/1.2/faq.html#noconfig for more info.
Dados do Formulário.docx:2252517d-38eb-430f-bcf7-47c6fe7afd5e
Dados do Formulário.docx:569afeb7-96dd-4601-b952-8102d9c9c9d2
  
```

Após clicar na classe com botão direito do mouse, e escolher o menu Run -> Java Application, o retorno na aba console será conforme imagem acima, mostrando as informações retornadas dos system.out.println que tem no momento em que está fazendo um foreach do retorno de documentos para aquele template e assim ele está listando nome do arquivo e unique id do ecm.

## f. Cancelar Processo

O objetivo dessa rota é realizar um cancelamento automático do processo, atividade, ciclo informado, como se tivesse na tela do processo e clicasse no disquete para salvar aquelas informações, lembrando que internamente está sendo feita uma chamada para api do produto sendo feito um encapsulamento para que não houvesse consumos diferentes dessa api em cada desenvolvimento.


```
RotaCancelarProcesso.java RotaLoginTest.java RotaSalvarProcesso.java
1 package com.lecom.testes;
2
3 import com.lecom.workflow.cadastros.rotas.CancelarProcesso;
4
5
6
7
8
9
10 public class RotaCancelarProcesso {
11     public static void main(String[] args) {
12         try {
13             String urlSSO = "";
14             String loginUsuario = "";
15             String senhaUsuario = "";
16             boolean manterLogado = false;
17             DadosLogin dadosLogin = new DadosLogin(loginUsuario, senhaUsuario, manterLogado);
18             LoginAutenticacao loginAutentica = new LoginAutenticacao(urlSSO, dadosLogin);
19             String token = loginAutentica.getToken();
20
21
22             String urlBPM = "";
23             String codProcesso = "";
24             String codEtapa = "";
25             String codCiclo = "";
26             String modoTeste = "";
27             String codUsuario = null;
28
29             DadosProcessoAbertura dadosProcessoAbertura = new DadosProcessoAbertura();
30             dadosProcessoAbertura.setProcessInstanceId(codProcesso);
31             dadosProcessoAbertura.setCurrentActivityInstanceId(codEtapa);
32             dadosProcessoAbertura.setCurrentCycle(codCiclo);
33
34             CancelarProcesso cancelar = new CancelarProcesso(urlBPM, token, dadosProcessoAbertura, modoTeste, codUsuario);
35
36             System.out.println(cancelar.cancelarProcesso());
37         } catch (LoginAuthenticationException e) {
38             // TODO Auto-generated catch block
39             e.printStackTrace();
40         } catch (CancelarProcessoException e1) {
41             // TODO Auto-generated catch block
42             e1.printStackTrace();
43             //logger.error("Erro" + Funcoes.exceptionPrinter(e1));
44         }
45     }
46 }
47
```

Na imagem acima temos o print do arquivo **RotaCancelarProcesso.java** ele está no Projeto Base mencionado no começo do documento no pacote de `com.lecom.testes` onde nesse pacote estará arquivos que podem ser rodados como Java Application e assim ver localmente o resultado da sua chamada testando os parâmetros configurados.

Para realizar o cancelamento do processo precisará realizar o login do usuário primeiro conforme visto no tópico acima e a partir do token gerado conseguiremos instanciar a classe **CancelarProcesso** passando os seguintes parâmetros no construtor dela:

- urlBPM: url do ambiente do cliente que deseja utilizar para realizar a abertura do processo, passando a url com o final bpm
- token: o token retornado da rota do login
- dadosProcessoAbertura: é o objeto retornado na chamada da rota de AbreProcesso, porem para esse exemplo nós iremos instanciar e preencher o objeto para apenas aprovar a instancia, esse objeto precisa ser preenchido as informações de código do processo, da atividade e do ciclo conforme as linhas 29 a 32
- modoTeste: precisa informar se esse formulário será aberto em modo teste, passando true, ou se será aberto em modo normal, passando false
- codUsuario: caso seja aberto em modo teste é necessário informar o código do usuário que será utilizado para iniciar a instancia desse processo, geralmente utilizamos o mesmo código do usuário logado, caso seja aberto em modo normal somente passar null nesse parâmetro

Após isso iremos chamar o método **cancelarProcesso**, retornando uma String que irá vir o valor OK caso tenha sucesso na chamada se não cairá na exceção específica para cancelamento que é a **CancelarProcessoException**, onde mostrará em tela o erro retornado pela api, podendo ser algum parâmetro errado que foi passado acima.



```
<terminated> RotaCancelarProcesso [Java Application] C:\Desenv\jdk1.8.0_101\jre\bin\javaw.exe (6 de mai de 2020 15:04:03)

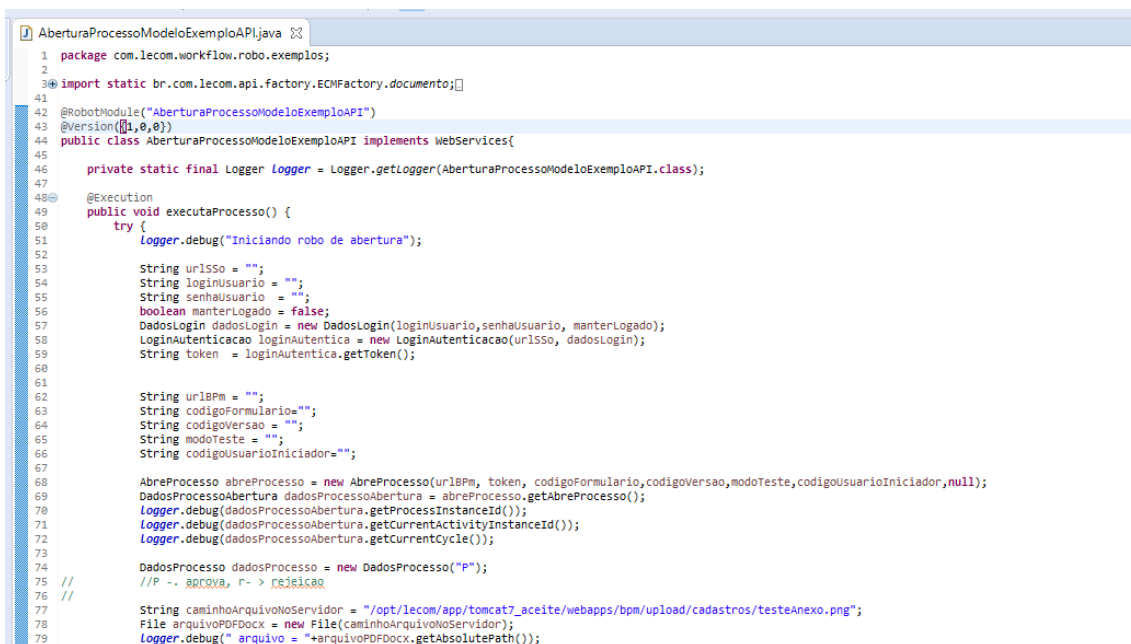
log4j:WARN No appenders could be found for logger (org.apache.http.client.protocol.RequestAddCookies).
log4j:WARN Please initialize the log4j system properly.
log4j:WARN See http://logging.apache.org/log4j/1.2/faq.html#noconfig for more info.
url = /process-Instances/3765/activity-Instances/1/cycles/1/
key = ticket-sso value=5bfdd3af-4a30-4538-9ef4-9652c22e9752
key = language value=pt_BR
key = test-mode value=false
key = Content-Type value=application/JSON

OK
```

Após clicar na classe com botão direito do mouse, e escolher o menu Run -> Java Application, o retorno na aba console será conforme imagem acima, mostrando as informações retornadas dos `system.out.println` que tem no print anterior da classe como um todo, a url chamada do bpm, apenas no print deixei aparecendo código do processo, atividade e ciclo para preservar ambiente utilizado de teste, no caso ticket sso e ticket mode (false, não está no modo teste o processo passado) e abaixo a mensagem OK que foi realizado o cancelamento com sucesso.

#### 4. Preenchimento de campos por fora

Nesse tópico vamos deslanchar sobre o objeto `DadosProcesso`, que utilizamos para realizar as ações de preenchimento de campos normais e de grid por fora da ferramenta utilizando mecanismo da própria linguagem java para isso, vamos ver um exemplo utilizando Rotas de Login, Abertura e Aprovação juntas para assim preencheremos campos normais e de grid.



```
AberturaProcessoModeloExemploAPI.java
1 package com.lecom.workflow.robos.exemplos;
2
3 import static br.com.lecom.api.factory.ECMFactory.documento;
4
5 @RobotModule("AberturaProcessoModeloExemploAPI")
6 @Version(1,0,0)
7 public class AberturaProcessoModeloExemploAPI implements Webservice {
8
9     private static final Logger logger = Logger.getLogger(AperturaProcessoModeloExemploAPI.class);
10
11     @Execution
12     public void executaProcesso() {
13         try {
14             logger.debug("Iniciando robo de abertura");
15
16             String urlSso = "";
17             String loginUsuario = "";
18             String senhaUsuario = "";
19             boolean manterLogado = false;
20             DadosLogin dadosLogin = new DadosLogin(loginUsuario, senhaUsuario, manterLogado);
21             LoginAutenticacao loginAutentica = new LoginAutenticacao(urlSso, dadosLogin);
22             String token = loginAutentica.getToken();
23
24             String urlBPM = "";
25             String codigoFormulario = "";
26             String codigoVersao = "";
27             String modoTeste = "";
28             String codigoUsuarioIniciador = "";
29
30             AbreProcesso abreProcesso = new AbreProcesso(urlBPM, token, codigoFormulario, codigoVersao, modoTeste, codigoUsuarioIniciador, null);
31             DadosProcessoAbertura dadosProcessoAbertura = abreProcesso.getAbreProcesso();
32             logger.debug(dadosProcessoAbertura.getProcessInstanceId());
33             logger.debug(dadosProcessoAbertura.getCurrentActivityInstanceId());
34             logger.debug(dadosProcessoAbertura.getCurrentCycle());
35
36             DadosProcesso dadosProcesso = new DadosProcesso("P");
37             //P -> 06361630
38
39             String caminhoArquivoNoServidor = "/opt/lecom/app/tomcat7_aceite/webapps/bpm/upload/cadastros/testeAnexo.png";
40             File arquivoPDFDocx = new File(caminhoArquivoNoServidor);
41             logger.debug("arquivo = " + arquivoPDFDocx.getAbsolutePath());
42         }
43     }
44 }
```

```
AberturaProcessoModeloExemploAPI.java
81 String identificadorTemplate = "TEMPLATE_0";
82 Document documento = documento().criarDocumentoComIdentificador(arquivoPDFDocx, identificadorTemplate).salvar();
83 Logger.debug(" documento = "+documento);
84 DocFile arquivoDoc = documento.getCurrentFile();
85
86 String nomeArquivo = arquivoDoc.getFileName();
87 String nomeCriptografado = arquivoDoc.getFileUniqueId().getValue();
88
89 Logger.debug(" nomeArquivo = "+nomeArquivo + " nomeCriptografado = "+nomeCriptografado);
90
91 Map<String,String> campos = new HashMap<>();
92 campos.put("NOME_USUARIO", "Nome do Usuário");
93 campos.put("EMAIL_USER", "email do usuário");
94 campos.put("TOTAL_DESPESAS", "20.00");
95 campos.put("ANEXO_SOLICITANTE", nomeArquivo+"-"+nomeCriptografado);
96
97 Logger.debug(" campos = "+campos);
98
99 dadosProcesso.geraPadroes(campos);
100
101 List<Map<String, Object>> valoresGrid = new ArrayList<Map<String, Object>>();
102
103 Instant timeZone = Instant.now();
104 LocalDate localDate = LocalDate.now();
105 timeZone = localDate.atStartOfDay(ZoneId.systemDefault()).toInstant();
106
107 Map<String, Object> linhaGrid1 = new HashMap<>();
108 linhaGrid1.put("TIPO_DESPESA", "Alimentação");
109 linhaGrid1.put("DATA_DESPESA", timeZone.toString());
110 linhaGrid1.put("QUANTIDADE", "1");
111 linhaGrid1.put("VALOR_DESPESA", "10");
112 linhaGrid1.put("TOTAL", "10");
113 valoresGrid.add(linhaGrid1);
114
115 Map<String, Object> linhaGrid2 = new HashMap<>();
116 linhaGrid2.put("TIPO_DESPESA", "Hospedagem");
117 linhaGrid2.put("DATA_DESPESA", timeZone.toString());
118 linhaGrid2.put("QUANTIDADE", "1");
119 linhaGrid2.put("VALOR_DESPESA", "10");
120 linhaGrid2.put("TOTAL", "10");
121 valoresGrid.add(linhaGrid2);
122
123 dadosProcesso.geraValoresGrid("DESPESAS", valoresGrid);
124
125
126
127
128
129 } catch (LoginAuthenticationException e) {
130 // TODO Auto-generated catch block
131 e.printStackTrace();
132 Logger.error("Erro ao login", e);
133 } catch (AbreProcessoException e) {
134 // TODO Auto-generated catch block
135 e.printStackTrace();
136 Logger.error("Erro ao abrir", e);
137 }
138 catch (AprovaProcessoException e) {
139 Logger.error("Erro ao aprovar", e);
140 e.printStackTrace();
141 }
142 catch (ConteudoDocumentoException e) {
143 // TODO Auto-generated catch block
144 e.printStackTrace();
145 } catch (DocumentoException e) {
146 // TODO Auto-generated catch block
147 e.printStackTrace();
148 } catch (ProfileException e) {
149 // TODO Auto-generated catch block
150 e.printStackTrace();
151 } catch (CriarDocumentoException e) {
152 // TODO Auto-generated catch block
153 e.printStackTrace();
154 } catch (ArquivoInvalidoException e) {
155 // TODO Auto-generated catch block
156 e.printStackTrace();
157 } catch (ArquivoNaoEncontradoException e) {
158 // TODO Auto-generated catch block
159 e.printStackTrace();
160 } catch (TemplateNaoEncontradoException e) {
161 // TODO Auto-generated catch block
162 e.printStackTrace();
163 } catch (TemplateException e) {
164 // TODO Auto-generated catch block
165 e.printStackTrace();
166 }
167 }
```

Na imagem acima temos o print do arquivo **AberturaProcessoModeloExemploAPI.java** ele está no Projeto Base mencionado no começo do documento no pacote de com.lecom.workflow.robo.exemplos onde nesse pacote estará alguns exemplos de robô.

No começo da classe das **linhas 53 a 59** está utilizando a Rota Login explicada em tópico anterior, como nas **linhas 68 a 72** está utilizando a Rota Abertura de Processo explicada em tópico anterior. Na **Linha 74** está sendo feito a inicialização do objeto DadosProcesso passando no construtor o valor de P, para aprovar o processo após a abertura.

Nas **linhas 77 a 79**, o que está sendo feito é inicializar um objeto File, passando o caminho completo de um arquivo que já está colocado diretamente no servidor do cliente, para assim usarmos esse arquivo físico e anexarmos em um campo do processo por fora, para isso

iremos utilizar uma api do ECM para criar um documento dentro de um template por fora e com o valor retornado iremos adicionar ao valor do campo do modelo.

Na **linha 81** estamos declarando o identificador do template, template esse que deve ser o mesmo atrelado ao campo que iremos preencher do modelo criado.

Na **linha 82** estamos iniciando a api do ecm, chamando **documento().criarDocumentoComIdentificador().salvar()**, essa api do ecm vem do import na classe (import static br.com.lecom.api.factory.ECMFactory.documento) e para o método criarDocumentoComIdentificador você irá passar a variável do objeto file instanciado na **linha 78** e o nome do identificador do template, esse método salvar retorna o objeto **Document**, e desse objeto conseguindo retornar um **DocFile** como na **linha 84**, e nas **linhas 86 e 87** você tem os valores que precisamos para preencher o campo do modelo que é o nome do arquivo e nomeCriptografado, que contém o unique id do ecm.

Nas **linhas 91 a 95** instanciamos um Map<String,String> com nome de variável campos, nele que vamos preencher cada valor de cada campo que queremos que seja preenchido no momento da aprovação do processo, por isso começamos a realizar o put desses map, colocando campos.put("NOME\_CAMPO"(aqui você tem que colocar o nome do seu campo igual criou no processo),"VALOR\_CAMPO"(aqui você colocará o valor que esse campo receberá)), assim como no exemplo estamos preenchendo o campo NOME\_USUARIO,EMAIL\_USUARIO,TOTAL\_DESPESAS,ANEXO\_SOLICITANTE, campos esses que são normais, dentro do campo ANEXO\_SOLICITANTE veja que colocamos o **nomeArquivo:nomeCriptografado** que é o Lecom BPM espera para o preenchimento de campos do tipo Template.

Na **linha 99** utilizamos aquele objeto inicializado na **linha 74** para chamar o método geraPadroes, passando para ele o map que instanciamos e explicamos acima, esse método criará um json dos valores de campos que vocês enviaram para assim passar para Rota de Aprovar e conseguir realizar as aprovações com preenchimento correto dos dados.

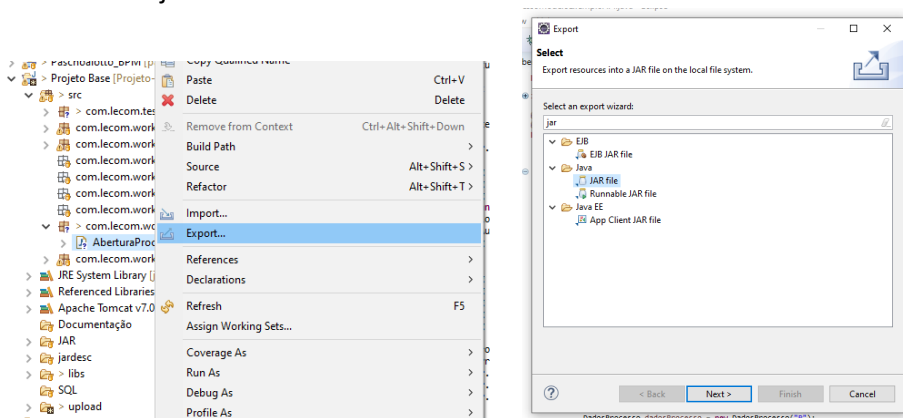
Nas **linhas 101 a 123** vamos ver como preenchemos campos do tipo grid por fora, vamos precisar tem uma List<Map<String,String>> instanciada como na **linha 101**, e iremos adicionar nela cada linha da grid, e cada linha da grid é um Map instanciado e preenchido com nome dos campos da grid e seus valores, conforme as **linhas 107 a 113 e 115 a 121** demonstram, nessas linhas esta sendo preenchido valores para os campos TIPO\_DESPESA,DATA\_DESPESA,QUANTIDADE,VALOR\_DESPESA,TOTAL e nas **linhas 113 e 121** são feitas as adições desses maps na lista principal instanciada na **linha 101**. Nesse mesmo bloco também mostramos uma forma que deve ser passada os campos datas, eles precisam ser passados para o Lecom BPM no formato ISO, por isso utilizamos nas **linhas 103 a 105** classes do pacote do JAVA 8 para inicializar a data de hoje e transformar o objeto do tipo LocalDate para um objeto do tipo Instant para o java esse objeto Instant é o formato ISO que precisamos, por isso nas **linhas 109 e 117** é colocado essa variável como valor dos campos data.

Na **linha 123** utilizamos aquele objeto inicializado na **linha 74** para chamar o método geraValoresGrid, para esse método você precisa passar o identificador da sua grid que irá preencher, no caso no exemplo chama DESPESAS, e a lista de grid instanciada na **linha 101** é preenchida nas **linhas 113 e 121**, esse método estará transformando esses valores em json para serem utilizados na aprovação do processo.

Nas **linhas 125 a 127** está utilizando a Rota de Aprovação explicada no tópico anterior.

Nesse exemplo como ele é um robô você precisaria realizar uns passos para você conseguir executar ele:

- Primeiro irá precisar pegar a lib do RotasBPM.jar que está na pasta libs do projeto e copiar para o servidor do cliente conforme tópico de como usar.
- Segundo irá precisar clicar com o botão direito do mouse em cima da classe para gerar um jar dele:



- Após o jar gerado subir ele no ambiente do cliente, no menu Studio -> Serviços -> Robo.
- Depois de colocar para rodar o robô você pode clicar em ver logs e avaliar como saiu os logs configurados no seu robô, irá ver que sai iniciando, código de processo, código de etapa, código de ciclo, caminho do arquivo colocado, objeto do documento, nome do arquivo , nome criptografado do arquivo, campos preenchidos, e o retorno ok da aprovação ou pode sair algum erro das exceptions configuradas.

Acima mostramos um robô que faz o login, abertura, preenche os campos e aprovação do processo, vamos ver agora como fazemos para editar dados de uma grid já existente utilizando um método do objeto DadosProcesso.



```
RotaAlterarGrid.java
17 public class RotaAlterarGrid {
18     public static void main(String[] args) {
19         try {
20             String urlSSO = "";
21             String loginUsuario = "";
22             String senhaUsuario = "";
23             boolean manterLogado = false;
24             DadosLogin dadosLogin = new DadosLogin(loginUsuario, senhaUsuario, manterLogado);
25             LoginAutenticacao loginAutentica = new LoginAutenticacao(urlSSO, dadosLogin);
26
27             String token = loginAutentica.getToken();
28             System.out.println("token = "+token);
29
30             String urlBPM = "";
31
32             DadosProcesso dadosProcesso = new DadosProcesso("P");
33
34             String identificadorLinha = "3";
35
36             Instant timeZone = Instant.now();
37             LocalDate localDate = LocalDate.now();
38             timeZone = localDate.atStartOfDay(ZoneId.systemDefault()).toInstant();
39
40             Map<String, Map<String, Object>> valoresGridExigEstadual = new HashMap<String, Map<String, Object>>();
41             Map<String, Object> linhaGrid1 = new HashMap<>();
42             linhaGrid1.put("TIPO_DESPESA", "Alimentação");
43             linhaGrid1.put("DATA_DESPESA", timeZone.toString());
44             linhaGrid1.put("QUANTIDADE", "1");
45             linhaGrid1.put("VALOR_DESPESA", "20");
46             linhaGrid1.put("TOTAL", "20");
47
48             valoresGridExigEstadual.put(identificadorLinha, linhaGrid1);
49             String identificadorGrid = "DESPESAS";
50
51             dadosProcesso.atualizarValoresGrid(identificadorGrid, valoresGridExigEstadual);
52
53             String codProcesso = "";
54             String codEtapa = "";
55             String codCiclo = "";
56             String modoTeste = "";
57             String codUsuario = null;
58             DadosProcessoAbertura dadosProcessoAbertura = new DadosProcessoAbertura();
59             dadosProcessoAbertura.setProcessInstanceId(codProcesso);
60             dadosProcessoAbertura.setCurrentActivityInstanceId(codEtapa);
61         }
62     }
63 }
```

Na imagem acima temos um print da classe **RotaAlterarGrid.java** ele está no Projeto Base mencionado no começo do documento no pacote de `com.lecom.testes` onde nesse pacote estará arquivos que podem ser rodados como Java Application e assim ver localmente o resultado da sua chamada testando os parâmetros configurados.

Utilizamos nesse exemplo acima mesmos nomes de campos e identificador da grid utilizado no exemplo anterior para abertura e aprovação de processo, a grande diferença desse exemplo é que você utilizara a Rota de Login e a Rota de Aprovação vistos no tópico anterior e iremos preencher os campos utilizando o objeto **DadosProcesso**, e nos maps que iremos montar vamos utilizar o identificador da linha.

Nas **linhas 40 a 51**, estamos fazendo a ação de alteração da grid, na **linha 40** estamos instanciando um `Map<String, Map<String, String>>` pois é nele que você preencherá uma ou mais linhas da grid que deseja que sofra alterações, a variável **identificadorLinha** é a chave da atualização da grid, essa informação você terá na tabela do processo, quando falamos em grid o nome da tabela criada é **g\_nometabelaProcessoidentificadorgrid** nessa tabela existe uma coluna chamada **identificador**, é esse valor que você utilizará para realizar as alterações por fora, passando essa informação na **linha 48** por exemplo, onde nela está sendo feito o put do map iniciado na **linha 40**, sendo primeiro parâmetro o identificador da linha e o segundo parâmetro o map da linha com os valores dos campos.

Na **linha 51** que passamos esse map completo, podendo conter várias linhas ou uma como no exemplo, para o método **atualizarValoresGrid** do objeto **DadosProcesso** nesse passamos o primeiro parâmetro sendo o identificador da grid e o segundo é o map que preenchemos anteriormente, assim internamente o método irá transformar isso em um json para ser feito a aprovação por fora corretamente.