
Trabalhamos pra
você ir mais longe.
Lecom

Lecom

INTEGRAÇÕES E ROBÔS

INTEGRAÇÕES E ROBÔS

- ✓ O QUE SÃO INTEGRAÇÕES E ROBÔS?
- ✓ INTEGRAÇÕES
- ✓ INTEGRAÇÕES HTTP
- ✓ INTEGRAÇÕES JAVA
- ✓ ROBÔS JAVA
- ✓GRÁFICOS ESPECIFICOS
- ✓ LOGS

Lecom

O QUE SÃO INTEGRAÇÕES E ROBÔS?

- ✓ Integrações e robôs, para o **Lecom BPM**, são, basicamente, rotinas desenvolvidas de forma a manipular dados, sendo disparadas pela aplicação.

- ✓ A diferença entre integrações e robôs reside no fato de que a integração é executada a partir de uma ação do usuário, enquanto o robô é uma rotina que é executada automaticamente, de tempos em tempos, de acordo com um intervalo especificado pelo usuário.

Lecom

INTEGRAÇÕES E ROBÔS

- ✓ O QUE SÃO INTEGRAÇÕES E ROBÔS?
- ✓ INTEGRAÇÕES
- ✓ INTEGRAÇÕES HTTP
- ✓ INTEGRAÇÕES JAVA
- ✓ ROBÔS JAVA
- ✓GRÁFICOS ESPECIFICOS
- ✓ LOGS

Lecom

INTEGRAÇÕES

- ✓ As integrações, dentro do **Lecom BPM**, podem ser feitas de duas maneiras:

1) **Via HTTP:** Este tipo de integração pode ser desenvolvida em qualquer linguagem web, porém o **Lecom BPM** não aguarda o retorno da execução para prosseguir as suas funcionalidades.

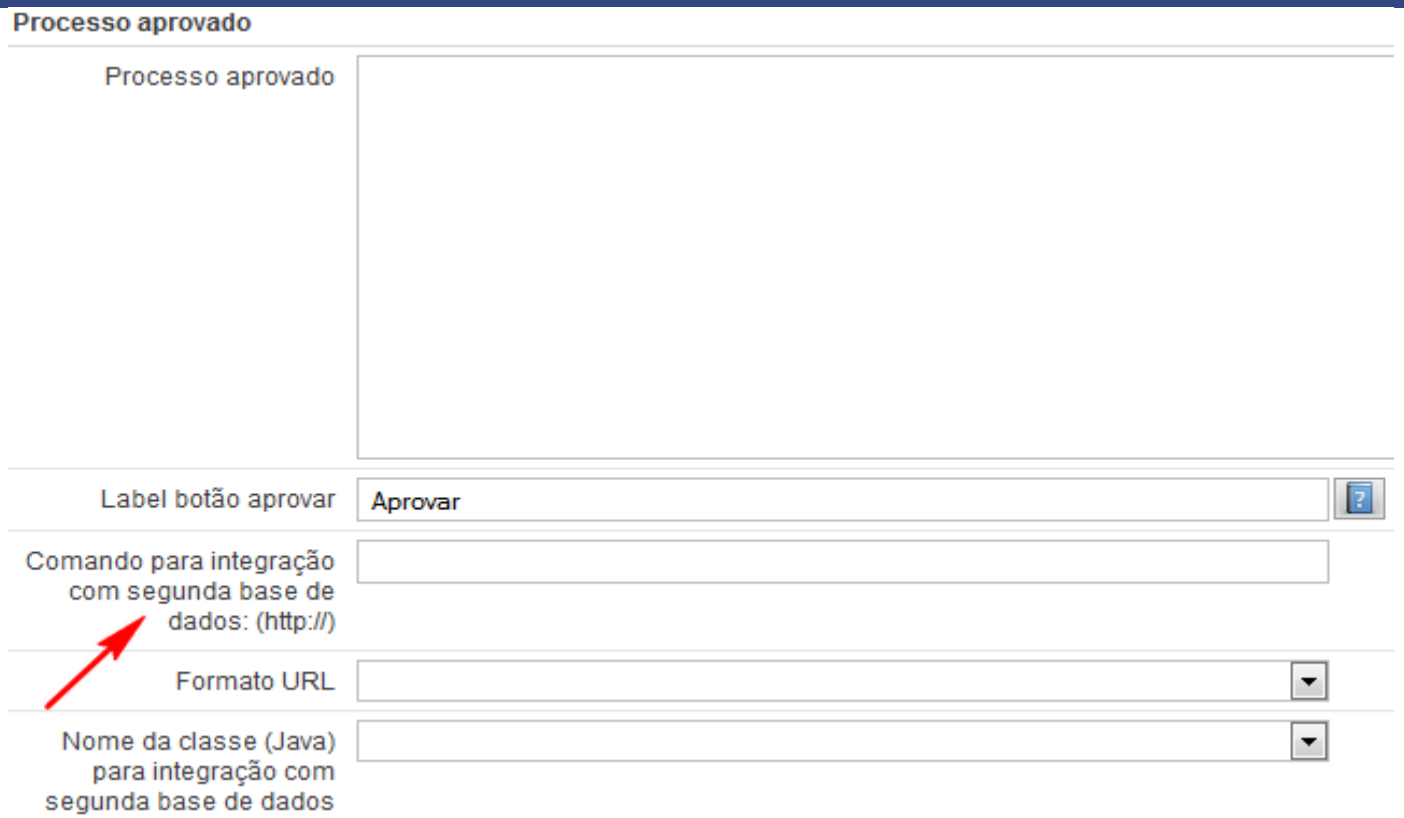
2) **Via Java:** Este tipo de integração deve ser desenvolvida na linguagem Java. Neste tipo de integração, o **Lecom BPM** aguarda o retorno da execução para só então, definir se prossegue ou não o andamento do processo.

INTEGRAÇÕES E ROBÔS

- ✓ O QUE SÃO INTEGRAÇÕES E ROBÔS?
- ✓ INTEGRAÇÕES
- ✓ INTEGRAÇÕES HTTP
- ✓ INTEGRAÇÕES JAVA
- ✓ ROBÔS JAVA
- ✓GRÁFICOS ESPECIFICOS
- ✓ LOGS

Lecom

INTEGRAÇÕES HTTP



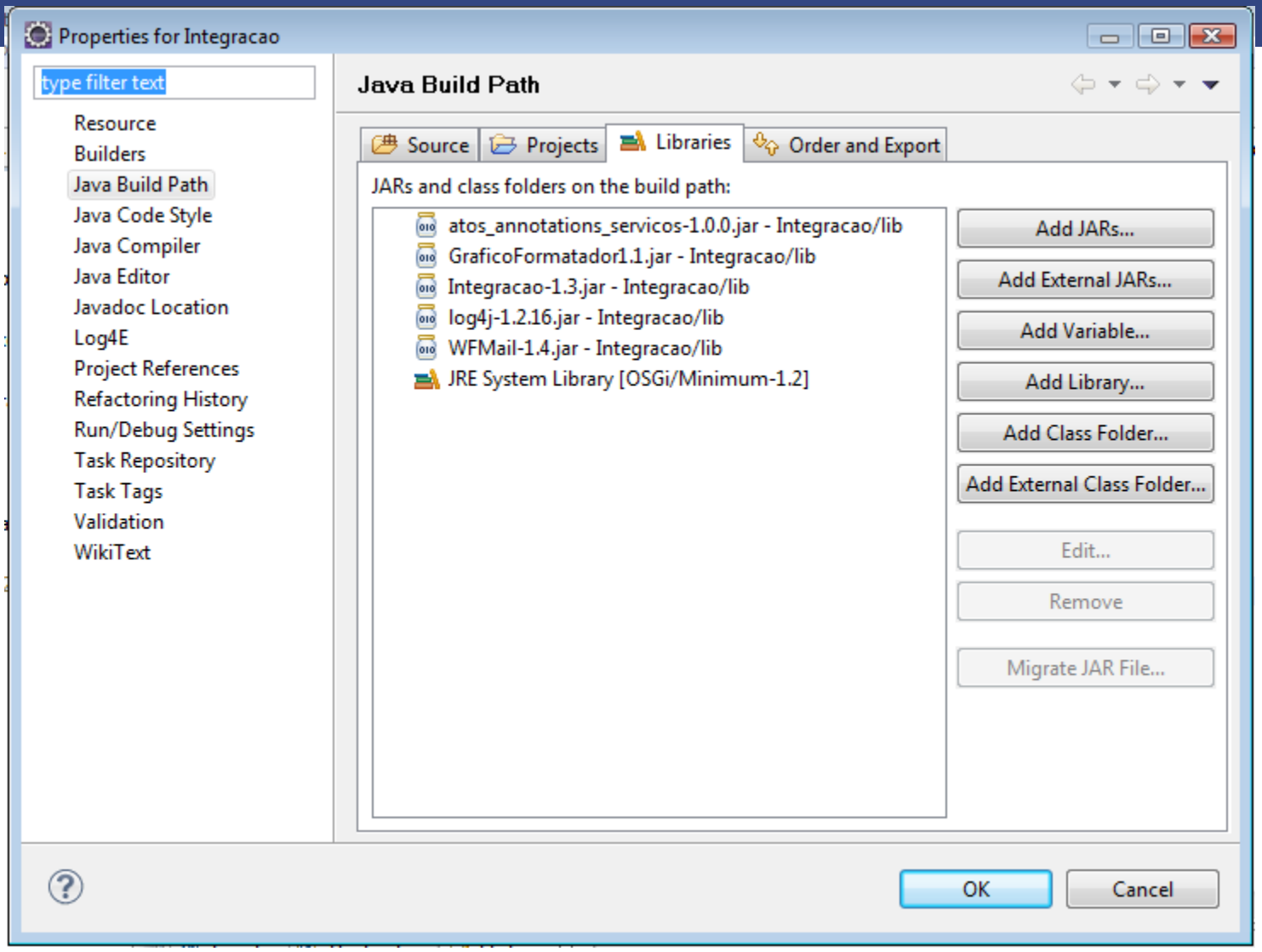
- ✓ Para configurar uma integração HTTP basta inserir a url completa da página web que deverá receber a chamada no campo "Comando para integração com segunda base de dados" do cadastro de etapa. Os parâmetros e dados dos campos das etapas é passado via request para a página definida.

INTEGRAÇÕES E ROBÔS

- ✓ O QUE SÃO INTEGRAÇÕES E ROBÔS?
- ✓ INTEGRAÇÕES
- ✓ INTEGRAÇÕES HTTP
- ✓ INTEGRAÇÕES JAVA
- ✓ ROBÔS JAVA
- ✓GRÁFICOS ESPECIFICOS
- ✓ LOGS

Lecom

INTEGRAÇÕES JAVA – PRÉ-REQUISITOS



INTEGRAÇÕES JAVA – PRÉ-REQUISITOS

- ✓ Para se desenvolver uma integração Java, utilizando os recursos oferecidos pelo **Lecom BPM**, é necessário possuir as seguintes bibliotecas:
  - ✓ **Integração-1.4.jar**: possui uma série de métodos destinados a ajudar o usuário no desenvolvimento.
  - ✓ **atos_annotations_servicos-1.0.0.jar**: para facilitar o desenvolvimento foram criadas algumas annotations pertencentes ao **Lecom BPM** e foram adicionadas a essa lib, em seguida veremos o seu uso.
  - ✓ **WFMail-1.4.jar:** envio de e-mail.
  - ✓ **log4j.jar**: desenvolvido pela *Apache Software Foundation*. Ele fornece uma API para que o desenvolvedor de *software* possa fazer LOG de dados.

Lecom

INTEGRAÇÕES JAVA – ESTRUTURA

```java
package com.lecom.workflow.integracao.exemplo;

import com.lecom.workflow.vo.IntegracaoVO;

import br.com.lecom.atos.servicos.annotation.Execution;
import br.com.lecom.atos.servicos.annotation.IntegrationModule;
import br.com.lecom.atos.servicos.annotation.Version;

@IntegrationModule(value="Valor campo")
@Version(value={1,0,0})

public class ValorCampo {

    @Execution
    public String valorCampo(IntegracaoVO integracao){

        return "0|Valor campo";

    }

}
```

- ✓ Como boas praticas de desenvolvimento a classe a ser desenvolvida deverá estar no package (pacote) que contenha o seguinte nome:

com.lecom.workflow.integracao.<nome>

- ✓ Toda classe de integração deverá conter a annotation *@IntegrationModule, @Version* e @Execution, sendo a annotation @Execution implementada no método que terá o inicio da execução que receberá como parâmetro um objeto do tipo IntegracaoVO e retorna uma String – que é o método chamado pelo **Lecom BPM** e que deverá conter o programa principal da integração.

INTEGRAÇÕES JAVA – IntegracaoVO

- ✓ Através do objeto da classe IntegracaoVO é possível recuperar informações que poderão ser utilizadas na integração.

```
integracaoVO.getCodProcesso(); //codigo do processo
integracaoVO.getCodEtapa(); //codigo da etapa
integracaoVO.getCodCiclo(); //ciclo da etapa
integracaoVO.getCodForm(); //codigo do modelo/formulario
integracaoVO.getNomeTabelaModelo(); //nome da tabela do modelo
integracaoVO.getCodUsuarioIniciador(); //codigo do usuario iniciador do processo
integracaoVO.getCodUsuarioEtapa(); //codigo do usuario da etapa
integracaoVO.getAcao(); //acao executada - 'P' (etapa aprovada), 'R' (etapa rejeitada)
integracaoVO.getStatus(); //'E' (em andamento), 'T' (atrasada)
integracaoVO.getDatData(); //data de abertura da etapa
integracaoVO.getDatFinalizacao(); //data limite da etapa
integracaoVO.getDesMailHost(); //mail host configurado no cadastro de parametros
integracaoVO.getDesFrom(); //mail from configurado no cadastro de parametros
integracaoVO.getDesReply(); //mail reply configurado no cadastro de parametros
integracaoVO.getMapCamposFormulario(); //'Map' com os valores dos campos do formulario na etapa
```

Lecom

INTEGRAÇÕES JAVA – IntegracaoVO

```
Map<String, String> paramForm = integracaoVO.getMapCamposFormulario(); //'Map' com os valores dos campos do formulario na etapa
String campo1 = paramForm.get("$CAMPO1");
String campo2 = paramForm.get("$CAMPO2");
```

- ✓ Acima podemos ver como são recuperados as informações dos campos do formulário.
- ✓ Abaixo, vemos como recuperar valores de campos que estão em Grid:

```
List<Map<String, Object>> dadosModeloGrid;
 try {
  dadosModeloGrid = integracaoVO.getDadosModeloGrid("ID_GRID");
  for (Map<String, Object> map : dadosModeloGrid) {

   for (String nomeCampo : map.keySet()) {
    System.out.println(nomeCampo + " = " + map.get(nomeCampo));
   }
  }
 } catch (Exception e) {
  e.printStackTrace();
 }
```

INTEGRAÇÕES JAVA – Conexão com base de dados

```
Connection connection = null;
try {
    integracaoVO.setConexao("workflow");
    connection = integracaoVO.getConexao();

} catch (Exception e) {
    e.printStackTrace();
} finally {
    try {
        connection.close();
    } catch (Exception e2) {
    }
}
```

- ✓ Ainda por meio deste objeto, podemos também abrir conexões com bases de dados, como ilustrado ao lado.
- ✓ No método setConexao() é passado como parâmetro o nome do arquivo ".properties" criado para o **Lecom BPM** fazer conexão com um database.

- ✓ Todas as conexões, resultSets e preparedStatements devem ser fechados após o seu uso.
- ✓ O **Lecom BPM** possui um pool para controle de conexões aberta e, por padrão, limita ao número mínimo de 01 conexão e máximo de 20 conexões abertas simultaneamente para cada arquivo configurado no módulo Administrativo. Portanto, se alguma conexão ficar aberta, após algumas execuções, o ambiente começará a apresentar problemas de lentidão e/ou travamento.

Lecom

INTEGRAÇÕES JAVA – Conexão com base de dados

- ✓ Abaixo segue um exemplo de fechamento de conexão, resultSet e preparedStatement:

```
Connection conn = null;
ResultSet rs = null;
PreparedStatement pst = null;

try {
    /* LÓGICA DE PROGRAMACAO */
} catch (SQLException e) {
    logger.error("Erro.");
    e.printStackTrace();
}finally{
    try {
        conn.close();
        rs.close();
        pst.close();
    } catch (SQLException e) {
        logger.error("Falha ao fechar as conexões.");
        e.printStackTrace();
    }finally{
        conn = null;
        rs = null;
        pst = null;
    }
}
```

Lecom

INTEGRAÇÕES JAVA – Conexão com base de dados

- *A execução de uma querie pode gerar uma exceção para o Java e é preciso fazer um tratamento da mesma utilizando o bloco Try/Catch. É importante que o fechamento das conexões, resultSet e preparedStatement fiquem dentro do bloco* ***Finally****, pois em caso de sucesso ou erro (que gera uma exceção) este bloco sempre é executado e assim garantimos o fechamento das conexões, resultSet e preparedStatement.*
- *Dependendo da lógica de programação utilizada pode ocorrer de conexões, resultSet e preparedStatement não serem populados, ficando o objeto nulo. Caso isso ocorra a chamada do método* ***.close()*** *para um objeto nulo irá gerar um erro que pode impedir a execução da integração, por isso é indicado que se houver a possibilidade do objeto estar nulo ele seja testado antes de ser fechado, conforme exemplo abaixo:*

```
if(conn != null){
      conn.close();
}

if(rs != null){
      rs.close();
}

if(pst!= null){
      pst.close();
}
```

Lecom

INTEGRAÇÕES JAVA – Envio de e-mail

- ✓ O método de envio de e-mail permite enviar e-mails para múltiplos usuários, e-mails com cópias, e-mail com cópias ocultas e arquivos em anexo.

- ✓ Para utilizar este método é bem simples, basta criar um objeto da classe EmailMessage e passar os parâmetros necessários para criar a mensagem.

1) **Subject:** [String] assunto do e-mail;

2) **Message :** [String] mensagem que irá no corpo do email;

3) **From:** [String] e-mail que será utilizado para enviar o e-mail (caso o ambiente necessite de autenticação do e-mail deve-se utilizar o mesmo e-mail do cadastro de parâmetros, pois a senha utilizada para autenticação será a que estiver cadastrada).

4) **To:** [List<String>] usuário(s) que receberá(ão) o e-mail.

5) **HTML:** [boolean] se a mensagem do e-mail terá ou não HTML.

6) **ReplyTo:** [List<String>] usuários que receberão resposta do email. (Opcional)

INTEGRAÇÕES JAVA – Envio de e-mail

```
String subject = "Teste de email";
String message = "Conteudo do email vai aqui";
String from    = "usuario1@lecom.com.br";

List<String> to = new ArrayList<String>();
to.add("usuario1@lecom.com.br");
to.add("usuario2@lecom.com.br");
to.add("usuario3@lecom.com.br");
```

①

```
List<String> replyTo = new ArrayList<String>();
replyTo.add("usuario4@lecom.com.br");
replyTo.add("usuario5@lecom.com.br");
```

②

```
EmailMessage emailMessage = new EmailMessage(subject, message, from, to, true, replyTo);

// Anexos
emailMessage.setAttached("C:\\teste.txt");
```

③

```
// Lista de Copia
List<String> cc = new ArrayList<String>();
cc.add("");
emailMessage.setListCc(cc);
```

④

```
// Lista de Copia Oculta
List<String> bcc = new ArrayList<String>();
bcc.add("");
emailMessage.setListCc(bcc);
```

⑤

```
integracaoVO.enviaEmailMessage(emailMessage);
```

⑥

Lecom

INTEGRAÇÕES JAVA – Envio de e-mail

- ✓ Para envio de e-mail a múltiplos usuários basta ir adicionando os outros endereços no list ①.
- ✓ Para usuários que receberão resposta do email, basta ir adicionando endereços no list ②.
- ✓ Para enviar um arquivo em anexo é necessário setar o caminho (é necessário que o arquivo esteja no servidor onde se encontra a instalação do **Lecom BPM**) após instanciar o objeto ③.
- ✓ Para enviar cópia e cópia oculta no e-mail basta setar os respectivos endereços eletrônicos ④ e ⑤.
- ✓ Para efetivar o envio, basta chamar o método enviaEmailMessage(), passando como parâmetro o EmailMessage configurado ⑥.

Lecom

INTEGRAÇÕES JAVA – Envio de e-mail autenticado

- ✓ Envio de email com autenticação normal (parâmetros do produto)

```
integracaoVO.enviaEmailMessage(emailMessage);
```

- ✓ Envio de email com outro usuário para autenticação

1) emailMessage – [emailMessage] estrutura do email
2) user – [String] login do usuário autenticado
3) pass – [String] senha não criptografada.

```
integracaoVO.enviaEmailMessage(emailMessage, "usuario", "senha_sem_critpgrafia");
```

Obs: Não é possível mudar o servidor e a porta, portanto o usuário autenticado deve ser do mesmo servidor de email cadastrado no workflow.

Lecom

INTEGRAÇÕES JAVA – Configuração

- ✓ Após o desenvolvimento da classe é preciso pegar os arquivos compilados e gerar um arquivo .jar

- ✓ Com o .jar gerado, é necessário cadastrar a integração na Área Modelista, opção Modelador > Serviços > Integrações.

- ✓ Na tela de cadastro é necessário informar:
  1) Nome da integração que será utilizado dentro do **Lecom BPM**;
  2) Tipo: se vai ser executada na página e/ou campo ou se vai ser executada na mudança de etapa.
  3) Arquivo, onde será feito o upload do arquivo **jar**;
  4) Descrição do que essa integração realiza;
  5) Modelistas permitidos, que poderão alterar o cadastro da integração e utilizá-la;
  6) Proprietário, usuário que pode adicionar/remover modelistas permitidos;

- ✓ Após a configuração, clicar no botão Salvar.

INTEGRAÇÕES JAVA – Configuração

- ✓ Apesar das mudanças que a versão nova do **Lecom BPM** apresenta, os integradores desenvolvidos e configurados nas versões anteriores do produto, ainda terá o mesmo efeito de execução, com isso não será necessário que seja atualizado os integradores já desenvolvidos.

INTEGRAÇÕES JAVA – Modo de execução

- ✓ Os integradores em Java podem ser executadas na aprovação/ rejeição de uma etapa ou no carregamento da etapa.

- ✓ Em ambos os casos será aguardado o retorno da execução para o prosseguimento do funcionamento normal do **Lecom BPM**.

- ✓ O que difere os dois tipos é o local de configuração e o os dados disponíveis no momento.

Lecom

INTEGRAÇÕES JAVA – Aprovação/rejeição da etapa

Independente do modo de execução a classe Java sempre precisa retornar um valor, indicando se a execução foi bem sucedida.

No caso de sucesso valor retornado é 0.

Ex.:

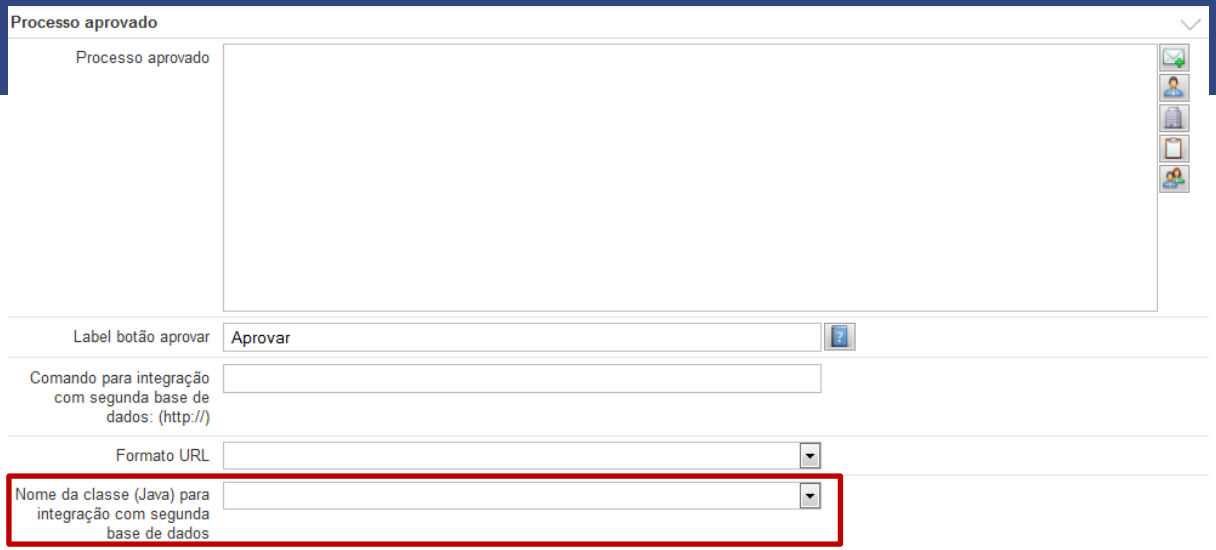
```
public String executa(IntegracaoVO integracaoVO) {
        return “0”;
}
```

Em caso de erro, deve ser retornado um valor acima de 6.

Ex.:

```
public String executa(IntegracaoVO integracaoVO) {
        return “99| [ERRO] Mensagem de erro. ”;
}
```

INTEGRAÇÕES JAVA – Aprovação/rejeição da etapa



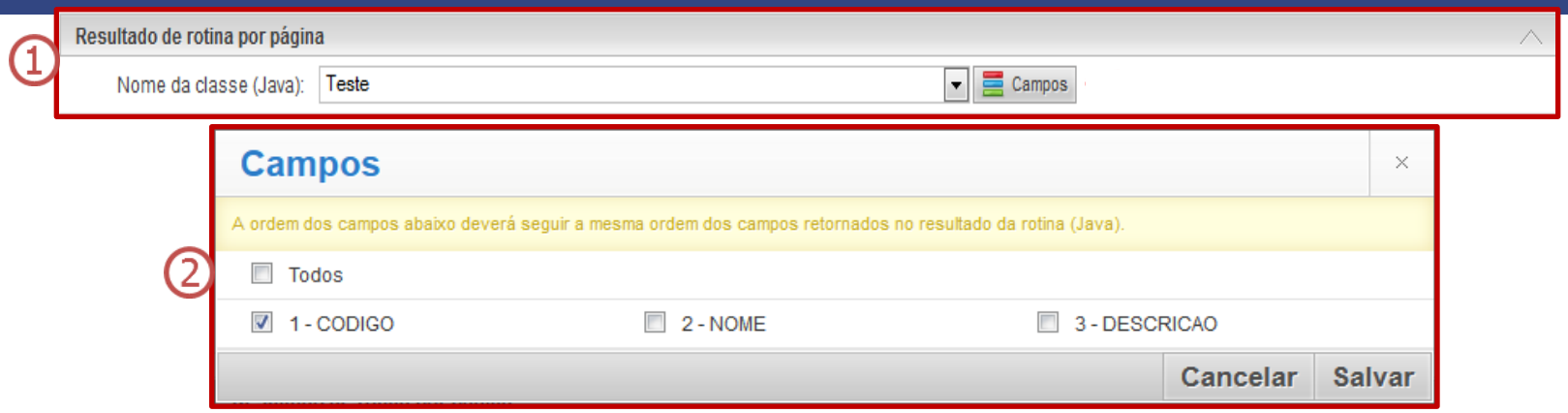
- ✓ É configurado no cadastro da etapa, podendo ser configurada na aprovação ou rejeição da etapa, e será executada no momento que o usuário executar a etapa. Basta colocar o nome da classe no campo "Nome da classe (Java) para integração com segunda base de dados", conforme mostra a imagem.

- ✓ Neste tipo de execução não é possível manipular informações pertinentes ao processo no banco de dados, porém tem a vantagem de ter os dados dos campos no objeto IntegracaoVO.

- ✓ Este tipo de integração é útil quando há a necessidade de realizar algum processamento com os dados do processo, sem precisar alterá-los.

Lecom

INTEGRAÇÕES JAVA – Carregamento da etapa



- ✓ É configurado no cadastro das propriedades dos campos e será executada antes dos dados de uma etapa serem carregados para o usuário.

- ✓ Para configurá-la basta colocar o nome da classe no campo "Nome da classe (Java)", na área de Resultado para rotina por página, conforme mostra a imagem ① .

- ✓ Utilizando este tipo de execução é possível manipular informações pertinentes ao processo no banco de dados, incluindo a possibilidade da classe já retornar os campos para etapa que será carregada preenchidos.

- ✓ Para que o resultado seja retornado nos campos é preciso selecionar os campos que receberão os valores. Após salvar o nome da classe clique no botão campos e será exibida a janela para a escolha dos campos, conforme a figura ② .

Lecom

INTEGRAÇÕES JAVA – Carregamento da etapa

- ✓ Neste tipo de execução o objeto IntegracaoVO ainda não possui os dados da etapa, pois eles não foram carregados ainda, tornando imprescindível a busca das informações no banco, caso seja necessário.

- ✓ Este tipo de integração é útil quando há a necessidade manipular as informações do processo.

INTEGRAÇÕES JAVA – Carregamento da etapa

- *A integração que preencherá campos da etapa precisa retornar os valores de acordo com a ordem (numeração) dos campos selecionados*
- *Independente do modo de execução a classe Java sempre precisa retornar um valor indicando se a execução foi bem sucedida, no caso de sucesso valor retornado é 0. Para as integrações que preenchem os campos é preciso retornar 0|<valores dos campos na ordem que foram selecionados>.*

Exemplo de retorno e formatações (todos os valores devem ser uma string):

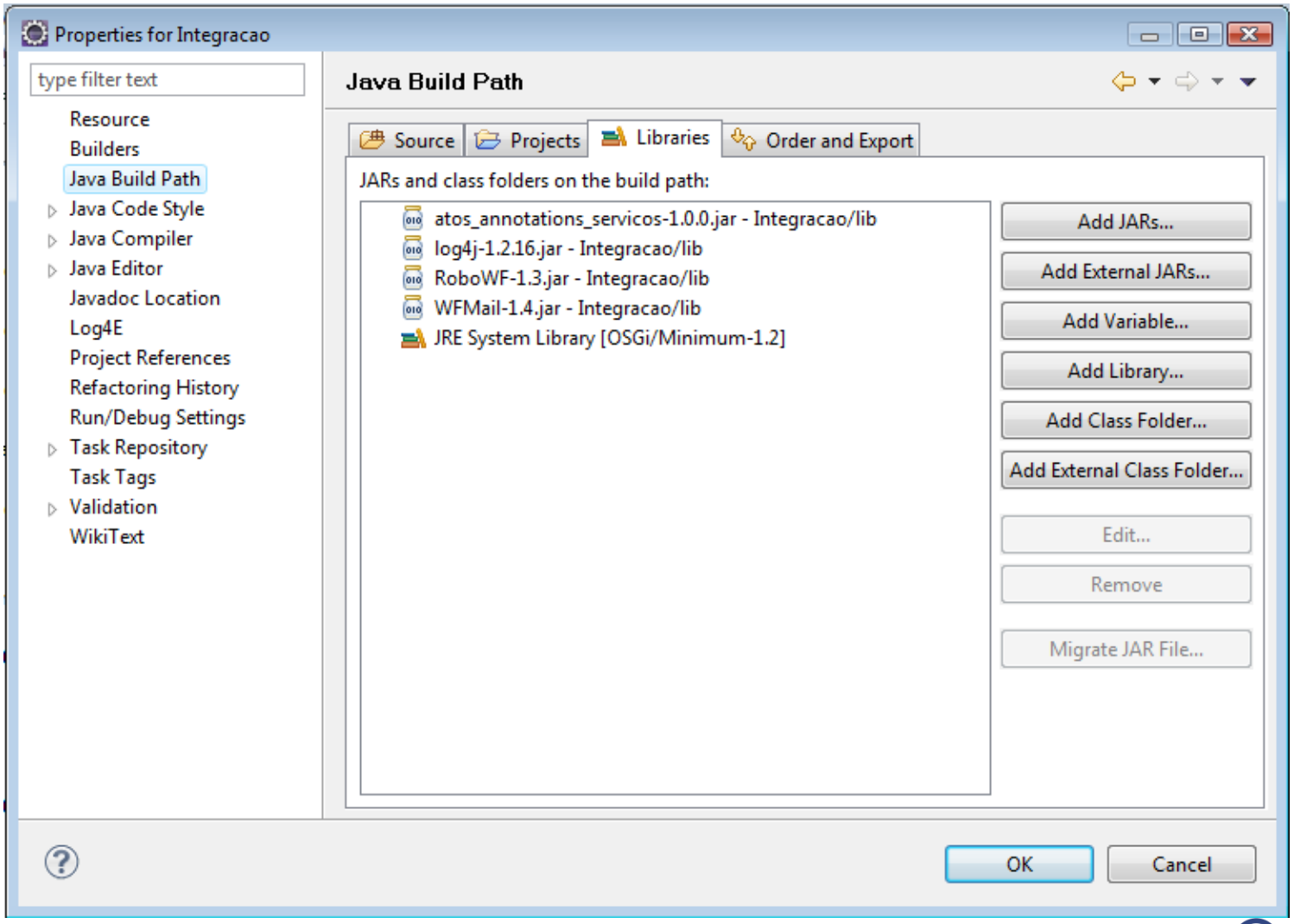
```
0|                          // 0 indica sucesso
Teste|                      // campo do tipo linha de texto
Teste |                     // campo do tipo caixa de texto
123|                        // campo do tipo inteiro
123.45|                     // campo do tipo monetário
1.0|                        // campo do tipo numérico com 1 casa decimal
01/01/2010|                 // campo do tipo data
;Opção 01;Opção 02;Opção 03 // campos do tipo lista com a 1ª opção em branco
```

INTEGRAÇÕES E ROBÔS

- ✓ O QUE SÃO INTEGRAÇÕES E ROBÔS?
- ✓ INTEGRAÇÕES
- ✓ INTEGRAÇÕES HTTP
- ✓ INTEGRAÇÕES JAVA
- ✓ ROBÔS JAVA
- ✓GRÁFICOS ESPECIFICOS
- ✓ LOGS

Lecom

ROBÔS JAVA – PRÉ-REQUISITOS



ROBÔS JAVA – PRÉ-REQUISITOS

- ✓ Para se desenvolver uma integração Java, utilizando os recursos oferecidos pelo **Lecom BPM**, é necessário possuir as seguintes bibliotecas:

  - ✓ **RoboWF-1.2.jar**: possui uma série de métodos destinados a ajudar o usuário no desenvolvimento, e também é necessario pelo fato de a interface a ser implementada [WebServices], requerida pelo **Lecom BPM**, estar dentro desta biblioteca.

  - ✓ **atos_annotations_servicos-1.0.0.jar**: para facilitar o desenvolvimento foram criadas algumas annotations pertencentes ao **Lecom BPM** e foram adicionadas a essa lib, em seguida veremos o seu uso.

  - ✓ **WFMail-1.4.jar:** envio de e-mail.

  - ✓ **log4j.jar**: desenvolvido pela *Apache Software Foundation*. Ele fornece uma API para que o desenvolvedor de *software* possa fazer LOG de dados.

Lecom

ROBÔS JAVA – ESTRUTURA

```java
package com.lecom.workflow.robo.enviamail;

import br.com.lecom.atos.servicos.annotation.Execution;
import br.com.lecom.atos.servicos.annotation.RobotModule;
import br.com.lecom.atos.servicos.annotation.Version;

@RobotModule(value="Robo de envio de email")
@Version(value={1,0,0})

public class RoboEnviaEmail {

    @Execution
    public void EnviaEmail(){

    }

}
```

- ✓ Como boas praticas de desenvolvimento a classe a ser desenvolvida deverá estar no package (pacote) que contenha o seguinte nome:

com.lecom.workflow.robo.<nome>

- ✓ Toda classe de robô deverá conter a annotation *@RobotModule, @Version* e @Execution, sendo a annotetion @Execution implementada no método que terá o inicio da execução.

Lecom

ROBÔS JAVA – Conexão com base de dados

```
Connection connection = null;
try {
    connection = DBUtils.getConnection("workflow");
} catch (Exception e) {
    e.printStackTrace();
} finally {
    try {
        connection.close();
    } catch (SQLException e) {
    }
}
```

- ✓ Na ilustração ao lado, podemos ver como é feita a conexão com o banco de dados.
- ✓ Uma conexão é retornada por uma chamada no método DBUtils.getConnection().
- ✓ Este método recebe como parâmetro o nome do arquivo ".properties" criado para o **Lecom BPM** fazer conexão com um database.
- ✓ Todas as conexões, resultSets e preparedStatements devem ser fechados após o seu uso.
- ✓ O **Lecom BPM** possui um pool para controle de conexões aberta e, por padrão, limita ao número mínimo de 01 conexão e máximo de 20 conexões abertas simultaneamente para cada arquivo configurado no módulo Administrativo. Portanto, se alguma conexão ficar aberta, após algumas execuções, o ambiente começará a apresentar problemas de lentidão e/ou travamento.

Lecom

ROBÔS JAVA – Conexão com base de dados

- ✓ Abaixo segue um exemplo de fechamento de conexão, resultSet e preparedStatement:

```
Connection conn = null;
ResultSet rs = null;
PreparedStatement pst = null;

try {

    /* LÓGICA DE PROGRAMACAO */

} catch (SQLException e) {
    logger.error("Erro.");
    e.printStackTrace();

}finally{

    try {

        conn.close();
        rs.close();
        pst.close();

    } catch (SQLException e) {
        logger.error("Falha ao fechar as conexões.");
        e.printStackTrace();

    }finally{
        conn = null;
        rs = null;
        pst = null;

    }
}
```

Lecom

ROBÔS JAVA – Conexão com base de dados

- *A execução de uma querie pode gerar uma exceção para o Java e é preciso fazer um tratamento da mesma utilizando o bloco Try/Catch. É importante que o fechamento das conexões, resultSet e preparedStatement fiquem dentro do bloco* ***Finally****, pois em caso de sucesso ou erro (que gera uma exceção) este bloco sempre é executado e assim garantimos o fechamento das conexões, resultSet e preparedStatement.*
- *Dependendo da lógica de programação utilizada pode ocorrer de conexões, resultSet e preparedStatement não serem populados, ficando o objeto nulo. Caso isso ocorra a chamada do método* ***.close()*** *para um objeto nulo irá gerar um erro que pode impedir a execução da integração, por isso é indicado que se houver a possibilidade do objeto estar nulo ele seja testado antes de ser fechado, conforme exemplo abaixo:*

```
if(conn != null){
      conn.close();
}

if(rs != null){
      rs.close();
}

if(pst!= null){
      pst.close();
}
```

Lecom

ROBÔS JAVA – Envio de e-mail

- ✓ No robô, para enviar e-mail, é necessária a utilização de uma outra biblioteca, WFMail-1.4.jar.
- ✓ O envio de e-mail permite enviar e-mails para múltiplos usuários, e-mails com cópias, e-mail com cópias ocultas e arquivos em anexo.
- ✓ Para utilizá-lo, basta criar um objeto da classe EmailMessage e passar os parâmetros necessários para criar a mensagem.

1) Subject – [String] assunto do e-mail;
2) Message – [String] mensagem que irá no corpo do email;
3) From – [String] e-mail que será utilizado para enviar o e-mail (caso o ambiente necessite de autenticação do e-mail deve-se utilizar o mesmo e-mail do cadastro de parâmetros, pois a senha utilizada para autenticação será a que estiver cadastrada).
4) To – [List<String>] usuário(s) que receberá(ão) o e-mail.
5) HTML – [boolean] se a mensagem do e-mail terá ou não HTML.
6) ReplyTo [List<String>] usuários que receberão resposta do email. (Opcional)

- ✓ No próximo slide temos um exemplo de como utilizar o novo envio de e-mails.

Lecom

ROBÔS JAVA – Envio de e-mail

```
String subject = "Teste envio email";
String message = "Conteudo do email.";
String from    = "usuario1@lecom.com.br";

List<String> to = new ArrayList<String>();
to.add("usuario2@lecom.com.br");
to.add("usuario3@lecom.com.br");                                          (1)

List<String> replyTo = new ArrayList<String>();
replyTo.add("usuario5@lecom.com.br");
replyTo.add("usuario6@lecom.com.br");                                     (2)

EmailMessage emailMessage = new EmailMessage(subject, message, from, to, true, replyTo);

// Anexos
emailMessage.setAttached("C:\\anexo.txt");                                (3)

// Lista de Copia
List<String> cc = new ArrayList<String>();
cc.add("usuario5@lecom.com.br");
emailMessage.setListCc(cc);                                               (4)

// Lista de Copia
List<String> bcc = new ArrayList<String>();
bcc.add("usuario7@lecom.com.br");
emailMessage.setListCc(bcc);                                              (5)

// Envia o e-mail
WFMail wfMail = new WFMail();
wfMail.enviaEmailMessage(emailMessage);                                   (6)
```

Lecom

ROBÔS JAVA – Envio de e-mail

- ✓ Para envio de e-mail a múltiplos usuários basta ir adicionando os outros endereços no list ①.
- ✓ Para usuários que receberão resposta do email, basta ir adicionando endereços no list ②.
- ✓ Para enviar um arquivo em anexo é necessário setar o caminho (é necessário que o arquivo esteja no servidor onde se encontra a instalação do **Lecom BPM**) após instanciar o objeto ③.
- ✓ Para enviar cópia e cópia oculta no e-mail basta setar os respectivos endereços eletrônicos ④ e ⑤.
- ✓ Para efetivar o envio, é necessário instanciar a classe WFMail e fazer a chamada do método enviaEmailMessage(), passando como parâmetro o EmailMessage configurado ⑥.

Lecom

ROBÔS JAVA – Envio de e-mail autenticado

- ✓ Envio de email com usuário autenticado
  - **1)** user – [String] login do usuário autenticado
  - **2)** pass – [String] senha não criptografada.

```
WFMail wfMail = new WFMail("usuario", "senha_sem_critpgrafia");
wfMail.enviaEmailMessage(emailMessage);
```

OU

```
WFMail wfMail = new WFMail();
wfMail.setUser("usuario");
wfMail.setPass("senha_sem_critpgrafia");
wfMail.enviaEmailMessage(emailMessage);
```

Obs: Não é possível mudar o servidor e a porta, portanto o usuário autenticado deve ser do mesmo servidor de email cadastrado no workflow.

Lecom

ROBÔS JAVA – Configuração

- ✓ Após o desenvolvimento da(s) classe(s) é preciso pegar os arquivos compilados e gerar um arquivo .jar

- ✓ Com o .jar gerado, é necessário cadastrar o robô na Área Modelista, opção Modelador > Serviços > Robôs.

- ✓ Na tela de cadastro é necessário informar:

  1) Nome do robô que será utilizado dentro do **Lecom BPM**;
  2) Arquivo, onde será feito o upload do arquivo **jar**;
  3) Descrição do que essa integração realiza;
  4) Tempo de Execução: intervalo de execução (de quanto em quanto tempo o robô será executado);
  5) Modelistas permitidos, que poderão alterar o cadastro da integração e utilizá-la;
  6) Proprietário, usuário que pode adicionar/remover modelistas permitidos;

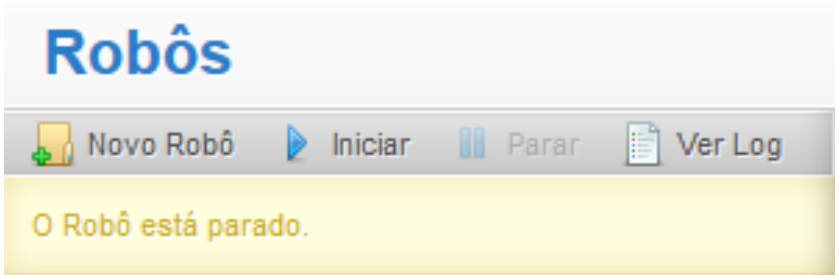
- ✓ Após a configuração, clicar no botão Salvar.

Lecom

ROBÔS JAVA – Configuração

- ✓ Apesar das mudanças que a versão nova do **Lecom BPM** apresenta, os integradores desenvolvidos e configurados nas versões anteriores do produto, ainda terá o mesmo efeito de execução, com isso não será necessário que seja atualizado os integradores já desenvolvidos.

ROBÔS JAVA – Configuração



- ✓ Para iniciar a execução do robô, clique em [Iniciar].
- ✓ Para pará-lo, clique em [Parar]
- ✓ Para visualizar o log gerado, clique em [Ver Log].

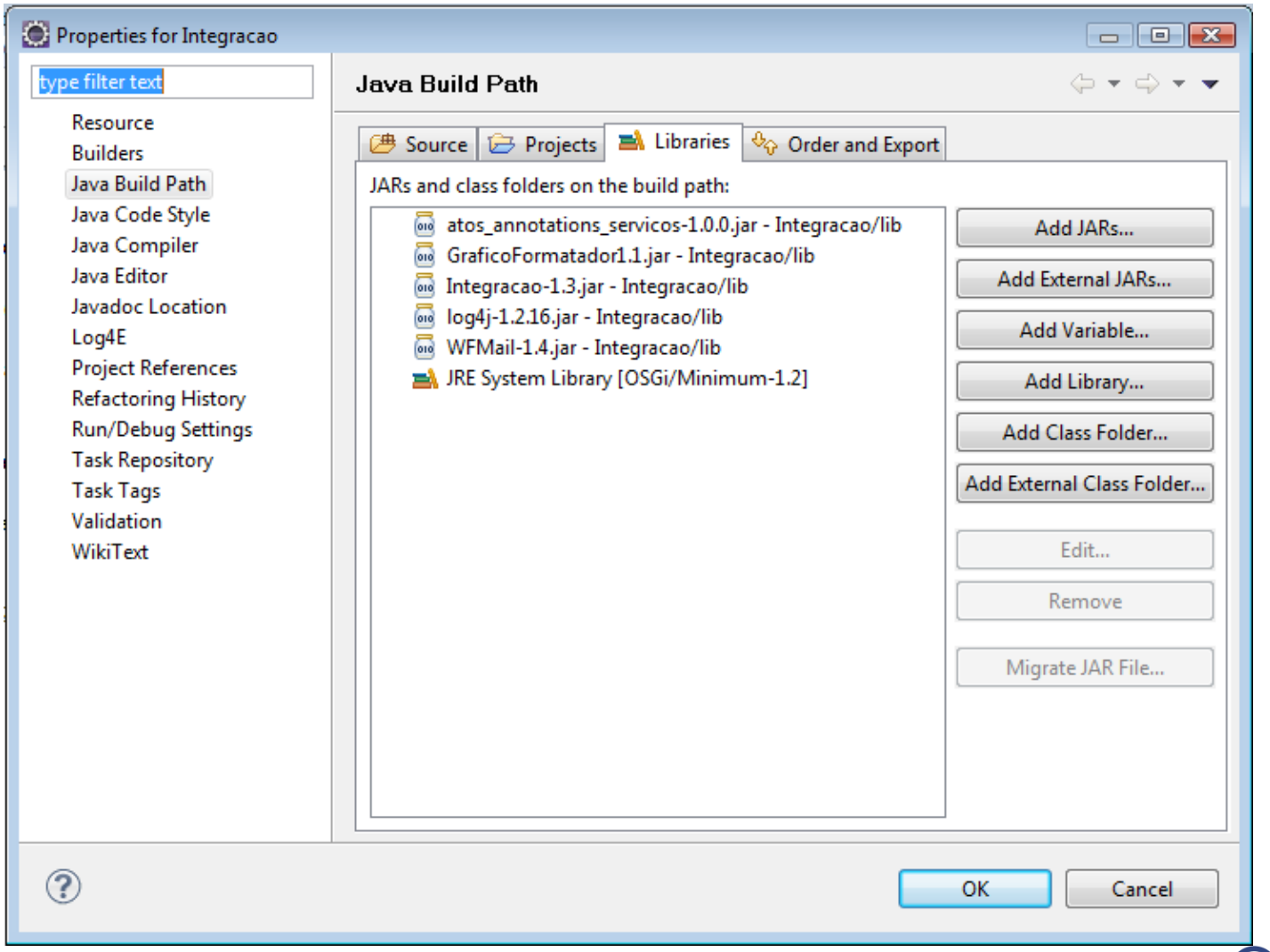
Lecom

INTEGRAÇÕES E ROBÔS

- ✓ O QUE SÃO INTEGRAÇÕES E ROBÔS?
- ✓ INTEGRAÇÕES
- ✓ INTEGRAÇÕES HTTP
- ✓ INTEGRAÇÕES JAVA
- ✓ ROBÔS JAVA
- ✓GRÁFICOS ESPECIFICOS
- ✓ LOGS

Lecom

GRÁFICOS ESPECIFICOS – PRÉ-REQUISITOS



GRÁFICOS ESPECIFICOS – PRÉ-REQUISITOS

- ✓ Para se desenvolver um gráfico especifico em Java, utilizando os recursos oferecidos pelo **Lecom BPM**, é necessário possuir as seguintes bibliotecas:

  - ✓ **Integração-1.3.jar e GraficoFormatador1.1.jar**: possui uma série de métodos destinados a ajudar o usuário no desenvolvimento.

  - ✓ **atos_annotations_servicos-1.0.0.jar**: para facilitar o desenvolvimento foram criadas algumas annotations pertencentes ao **Lecom BPM** e foram adicionadas a essa lib, em seguida veremos o seu uso.

  - ✓ **WFMail-1.4.jar:** envio de e-mail.

  - ✓ **log4j.jar**: desenvolvido pela *Apache Software Foundation*. Ele fornece uma API para que o desenvolvedor de *software* possa fazer LOG de dados.

Lecom

GRÁFICOS ESPECIFICOS – ESTRUTURA

```
1  package com.lecom.workflow.integracao.grafico;
2  import br.com.lecom.atos.servicos.annotation.ChartModule;
5
6
7  @ChartModule(value="Gráfico Usuarios Inativos")
8  @Version(value={1,5,0})
9  public class GraficoUsuariosInativos {
10
11     @Execution
12     public String gerarGrafico(){
13         return "";
14     }
15
16 }
17
```

- ✓ Como boas praticas de desenvolvimento a classe a ser desenvolvida deverá estar no package (pacote) que contenha o seguinte nome:

com.lecom.workflow.integracao.grafico

- ✓ Toda classe de gráficos deverá conter a annotation *@ChartModule, @Version* e @Execution, sendo a annotetion @Execution implementada no método que terá o inicio da execução que receberá como parâmetro um objeto do tipo IntegracaoVO e retornará uma String – que é o método chamado pelo **Lecom BPM** e que deverá conter o programa principal do gráfico.

GRÁFICOS ESPECIFICOS – IntegracaoVO, Conexão com base de dados.

- ✓ Através do objeto da classe IntegracaoVO é possível recuperar informações que poderão ser utilizadas na integração (Como é feito atualmente em uma integração).

- ✓ As conexões com a base de dados podem ser realizadas da mesma forma em que é feito atualmente com uma integração , através do método setConexao() do objeto IntegracaoVO , passando como parâmetro o nome do arquivo ".properties".

GRÁFICOS ESPECIFICOS – FORMATADORES

- ✓ Existem 2 tipos de formatadores xml para gerar o gráfico
- GraficoXMLFormatador.formataXMLEixo
- GraficoXMLFormatador.formataXMLPizza

- ✓ Sendo que o "GraficoXMLFormatador.*formataXMLEixo" é utilizado para os gráficos do tipo* ***barra,coluna e linha****.*

- ✓ *E o* GraficoXMLFormatador.formataXMLPizza *é utilizado somente para gráficos do tipo* ***pizza***

Lecom

GRÁFICOS ESPECIFICOS – TIPOS DE LIST

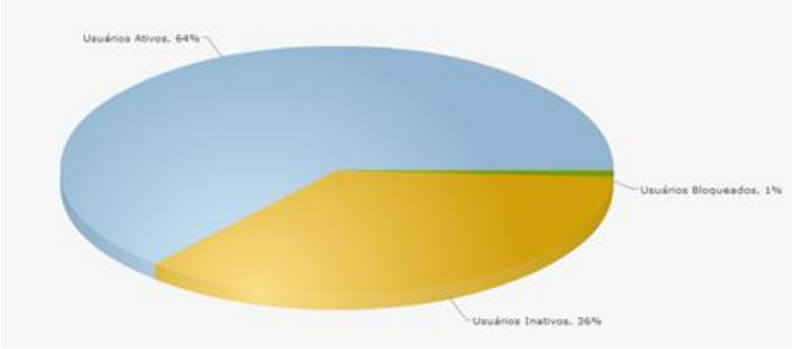
- ✓ Para se gerar um gráfico do tipo pizza é necessário que os dados estejam em uma lista do tipo "GraficoPizza" como o ex:

```
List<GraficoPizza> dados = new ArrayList<GraficoPizza>();

GraficoPizza p1 = new GraficoPizza();
p1.setDescricao("Usuários Ativos");
p1.setValor(10);

GraficoPizza p2 = new GraficoPizza();
p2.setDescricao("Usuários Bloqueados");
p2.setValor(90);

dados.add(p1);
dados.add(p2);
```



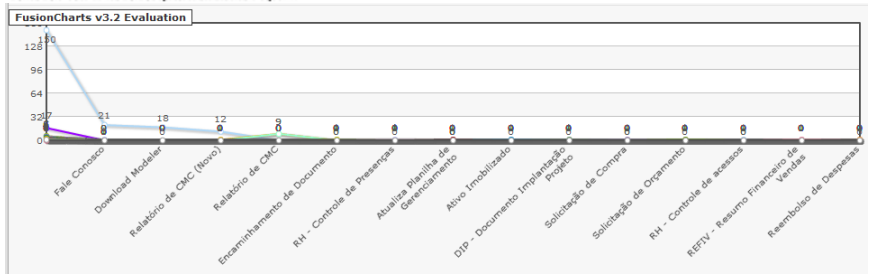
- ✓ Para os demais tipos de gráficos (coluna , barra e linha) deve se utilizar uma lista do tipo "GraficoEixo" como o ex:

```
List<GraficoEixo> dados = new ArrayList<GraficoEixo>();

GraficoEixo linha = new GraficoEixo();
linha.setColunaDescricaoX("Descricao");
linha.setLinhaLegendaY("Linha Legenda Y1");
linha.setValorLegendaY(50);

GraficoEixo linha2 = new GraficoEixo();
linha2.setColunaDescricaoX("Descricao");
linha2.setLinhaLegendaY("Linha Legenda Y2");
linha2.setValorLegendaY(200);

dados.add(linha);
dados.add(linha2);
```



Lecom

GRÁFICOS ESPECIFICOS – PARAMETROS CONSTRUTOR

- ✓ Para cada formatador temos métodos de implementação onde os parâmetros podem ser:

dadosFiltrados = ArrayList com os dados que serão mostrados
tipoFormatacao = Tipo do resultado, se será valor absoluto ou %
subCaption = Título do gráfico , este campo pode ser nulo
impressao *= Boolean que caso verdadeiro não mostra a cor de fundo padrão.*
cor *= String com valor da cor em hexa, este campo pode ser nulo*
expressaoFiltroRank *= String para exibir Top maiores ou menores*

- ✓ O parâmetro expressaoFiltroRank segue o padrão "qtde|tipo|ordem", onde:

qtde = número de itens a exibir
tipo = "maiores" ou "menores"
ordem = "ASC" ou "DESC"

Exemplo: "10|maiores|ASC"

Lecom

GRÁFICOS ESPECIFICOS – RETURN

Independente da área onde a classe Java será executada sempre precisa retornar GraficoXMLFormatador, contendo o xml com os dados do gráfico gerado.

Ex.:

```
public String executa(IntegracaoVO integracaoVO) {

.....
        return GraficoXMLFormatador.formataXMLEixo(dados,
TipoFormatacao.VALOR, subCaption, impressao, cor, expressaoFiltroRank);
}
```

✓ Veja na próxima tela um exemplo do código pronto que é utilizado para gerar um gráfico.

```java
package com.lecom.workflow.grafico;

import java.util.ArrayList;

public class MeuGraficoEspecificoLinha implements IntegracaoFace {

    public String executa(IntegracaoVO arg0) {
        String subCaption = "Titulo";
        String dataSet="";
        String hexaCor="D9ECFF";
        try{

            List<GraficoEixo> dados = new ArrayList<GraficoEixo>();

            GraficoEixo linha = new GraficoEixo();
            linha.setColunaDescricaoX("Descricao");
            linha.setLinhaLegendaY("Linha Legenda Y1");
            linha.setValorLegendaY(50);

            GraficoEixo linha2 = new GraficoEixo();
            linha2.setColunaDescricaoX("Descricao");
            linha2.setLinhaLegendaY("Linha Legenda Y2");
            linha2.setValorLegendaY(200);

            dados.add(linha);
            dados.add(linha2);

            dataSet = GraficoXMLFormatador.formataXMLEixo(dados, TipoFormatacao.VALOR, hexaCor,subCaption);

        } catch(Exception e){
            e.printStackTrace();
        }
        return dataSet;
    }
```

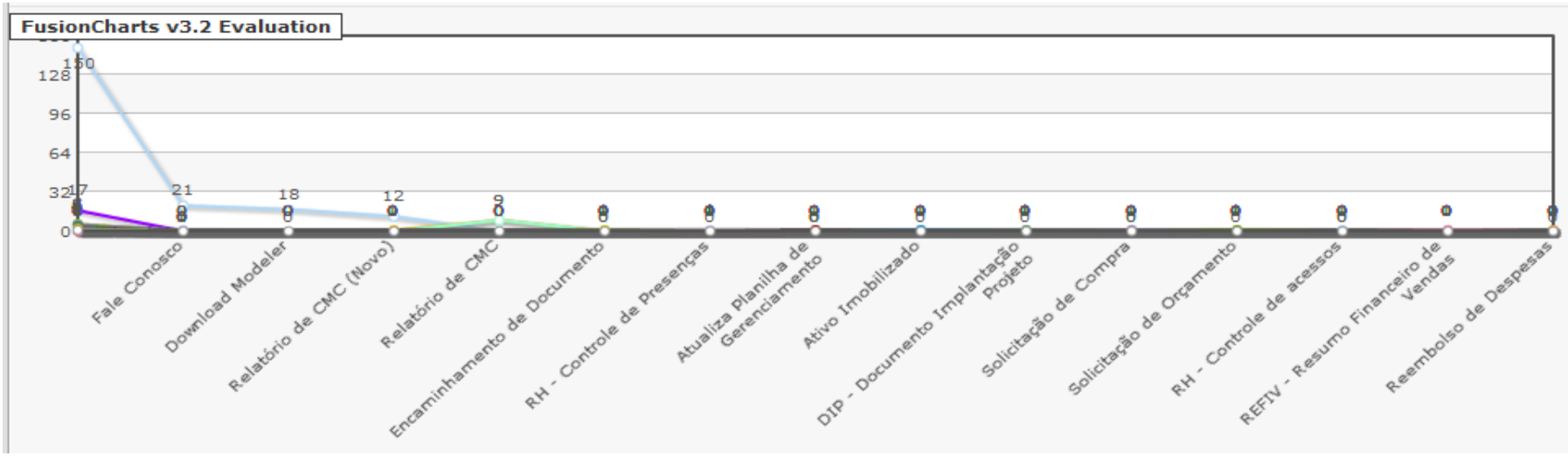
- ✓ Gráfico e data set gerado por esta classe no próximo slide.

Lecom

GRÁFICOS ESPECIFICOS – DATASET E GRÁFICO GERADO

- ✓ Exemplo do código xml gerado e da imagem que é mostrada ao usuário .

```
<graph showValues='0' numDivLines='10' formatNumberScale='0' anchorSides='10' anchorRadius='3'
    anchorBorderColor='009900' rotateNames='1' subCaption='Titulo'
    bgColor='D9ECFF'><categories><category name='Linha Legenda Y1' hoverText='Linha Legenda
    Y1'/><category name='Linha Legenda Y2' hoverText='Linha Legenda Y2'/></categories><dataset
    seriesName='Descricao' color='AFD8F8' showValues='1'><set value='50' /><set value='200'
    /></dataset><styles><definition>   <style name='myCaptionFont' type='font'
    align='right'/></definition><application>   <apply toObject='Caption' styles='myCaptionFont' />
    <apply toObject='SubCaption' styles='myCaptionFont' /></application></styles></graph>
```



GRÁFICOS ESPECIFICOS – Configuração

- ✓ Após o desenvolvimento da classe é preciso pegar os arquivos compilados e adicioná-los no diretório abaixo ou gerar um arquivo **jar** e fazer o upload através do cadastro

<workflow>/WEB-INF/classes/com/lecom/workflow/grafico

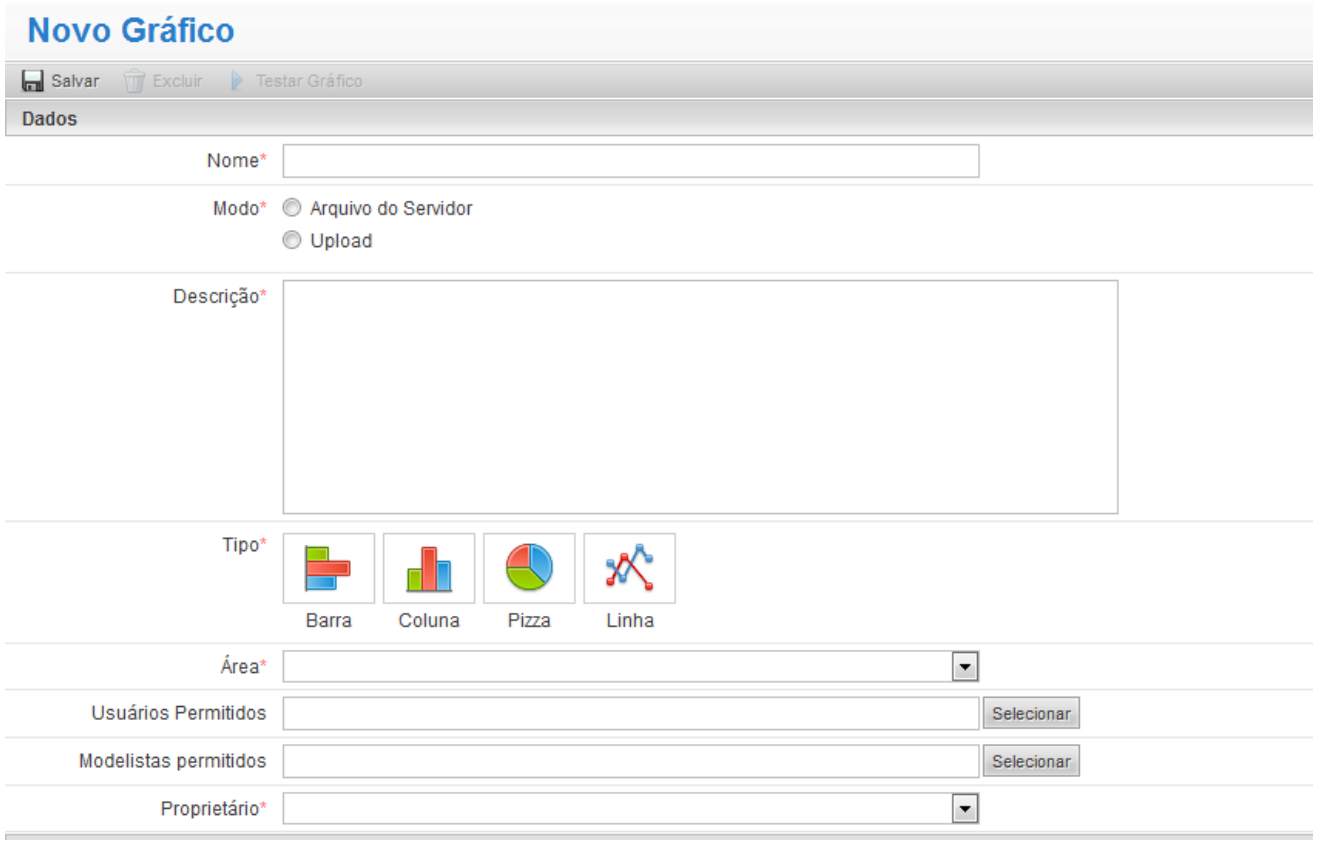
- ✓ No caso de haverem outras bibliotecas, utilizadas para o desenvolvimento de outras funcionalidades, elas devem ser colocadas no diretório

<workflow>/WEB-INF/lib

Lecom

GRÁFICOS ESPECIFICOS – CADASTRO

- ✓ Na Área Modelista, acesse o menu [Serviços] >> [Gráficos], clique no botão [Novo Gráfico] e será exibida a tela abaixo.



GRÁFICOS ESPECIFICOS – CADASTRO

✓ Para cadastrar um gráfico é necessário preencher os seguintes campos:

1) **Nome do gráfico:** Titulo do gráfico.

2) **Arquivo do Servidor:** Nome do arquivo que está na pasta Upload.
   Ex: Jan2013.xls, lembrando que esse arquivo deve ficar na pasta upload.

3) **Upload:** Opção onde o arquivo será enviado ao servidor.

4) **Descrição:** Descrição do gráfico.

5) **Tipo:** Tipo do gráfico (Barra, Coluna, Pizza, Linha).

Lecom

GRÁFICOS ESPECIFICOS – CADASTRO

✓ Para cadastrar um gráfico é necessário preencher os seguintes campos:

6) **Área:** Seleciona em quais áreas o gráfico poderá ser visualizado.

7) **Usuários Permitidos:** Usuários que poderão visualizar esse gráfico.

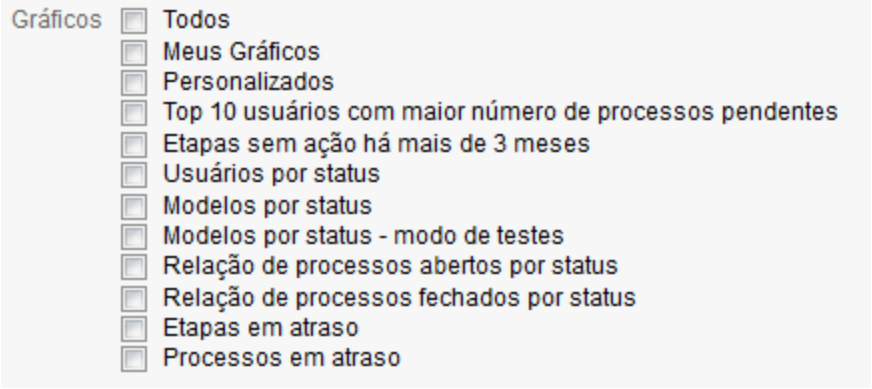
8) **Modelistas Permitidos:** Usuários que poderão utilizar esse gráfico.

9) **Proprietário:** Usuário que poderá adicionar/remover modelistas permitidos.

GRÁFICOS ESPECIFICOS – Modo de execução

- ✓ As classes Java podem ser executadas na etapa de um processo ou na área de gráficos , ou até mesmo em ambas as áreas.

- ✓ Após cadastrar o gráfico para poder visualizá-lo é necessário liberar o acesso do usuário ao gráfico , no cadastro do usuário ou apenas selecionar os usuários que terão esse privilégio na hora do cadastro do gráfico.



INTEGRAÇÕES E ROBÔS

- ✓ O QUE SÃO INTEGRAÇÕES E ROBÔS?
- ✓ INTEGRAÇÕES
- ✓ INTEGRAÇÕES HTTP
- ✓ INTEGRAÇÕES JAVA
- ✓ ROBÔS JAVA
- ✓GRÁFICOS ESPECIFICOS
- ✓ LOGS

Lecom

LOGS

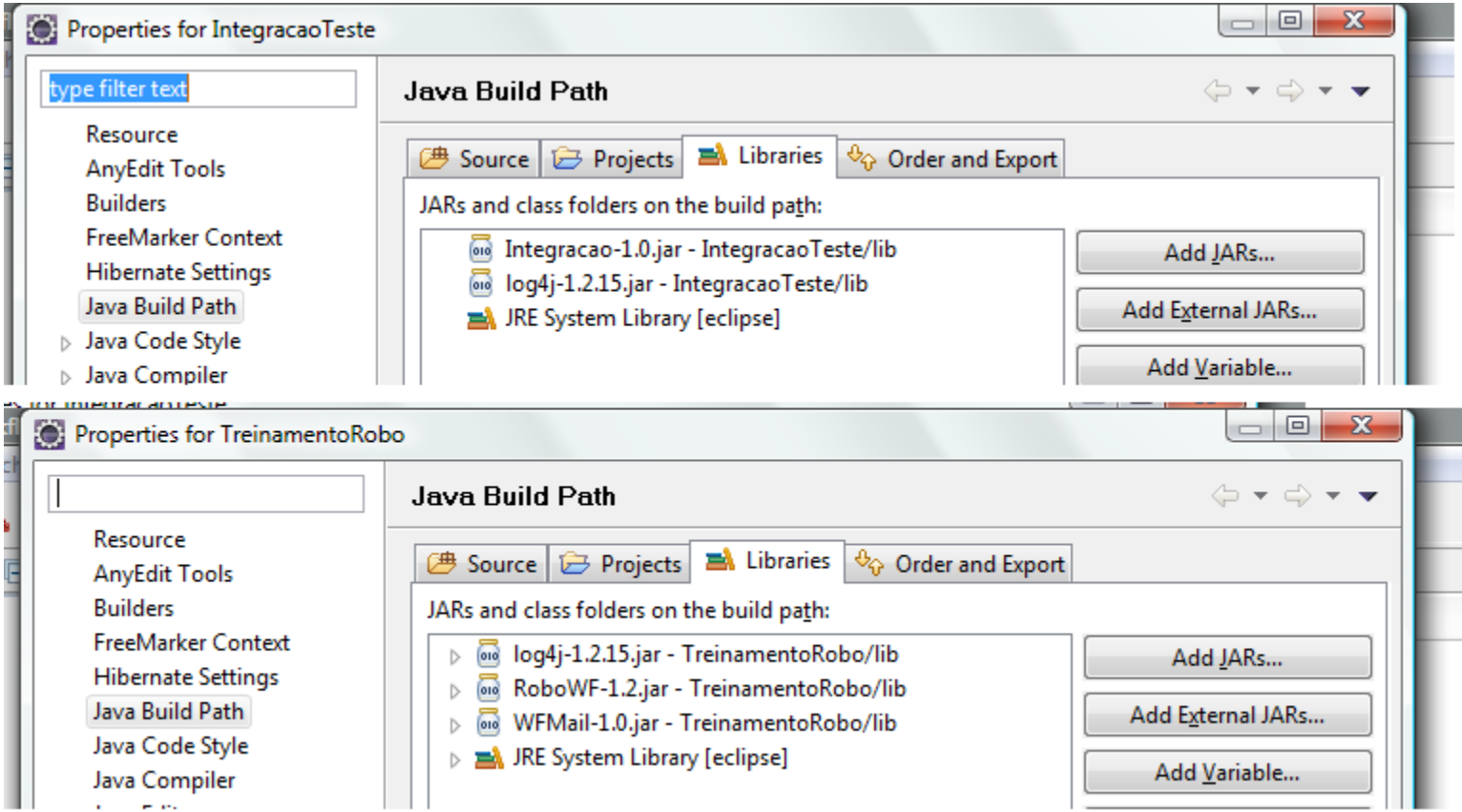
- ✓ O **Lecom BPM** está configurado para utilizar o log4j para gravar os seus logs em arquivos de log.

- ✓ As classes utilizadas como integrações possuem um arquivo dedicado aos seus logs. Eles são salvos no arquivo INTEGRACAO_WORKFLOW.log que fica no diretório logs do tomcat.

- ✓ As classes utilizadas como robôs possuem um arquivo dedicado aos seus logs. Eles são salvos no arquivo ROBO_WORKFLOW.log que fica no diretório logs do tomcat.

- ✓ Para configurar o seu robô/integração para usar o log é preciso adicionar a biblioteca log4j-1.2.x.jar no seu projeto e colocar uma linha de declaração, ambos mostrados nas ilustrações a seguir.

LOGS


Properties for IntegracaoTeste
type filter text
Resource
AnyEdit Tools
Builders
FreeMarker Context
Hibernate Settings
Java Build Path
Java Code Style
Java Compiler
Java Build Path
Source
Projects
Libraries
Order and Export
JARs and class folders on the build path:
Integracao-1.0.jar - IntegracaoTeste/lib
log4j-1.2.15.jar - IntegracaoTeste/lib
JRE System Library [eclipse]
Add JARs...
Add External JARs...
Add Variable...
Properties for TreinamentoRobo
Resource
AnyEdit Tools
Builders
FreeMarker Context
Hibernate Settings
Java Build Path
Java Code Style
Java Compiler
Java Build Path
Source
Projects
Libraries
Order and Export
JARs and class folders on the build path:
log4j-1.2.15.jar - TreinamentoRobo/lib
RoboWF-1.2.jar - TreinamentoRobo/lib
WFMail-1.0.jar - TreinamentoRobo/lib
JRE System Library [eclipse]
Add JARs...
Add External JARs...
Add Variable...

Lecom

LOGS

```
public class Aprovar{

    private static final Logger logger = Logger.getLogger(Aprovar.class);
```

- ✓ Acima estão as linhas, declarando um logger para as classes de Integração e Robô, respectivamente.

LOGS

- ✓ O Logger possui cinco níveis de depuração:

1) **FATAL:** apresenta mensagens de erros graves, que irá abortar a execução do aplicativo.

2) **ERROR:** apresenta mensagens de erros não tão graves, que provavelmente permitirão a aplicação continuar executando.

3) **WARN:** apresenta mensagens de advertências.

4) **INFO:** apresenta mensagens informativas sobre a execução do aplicativo.

5) **DEBUG:** apresenta mensagens de depuração.

Os níveis de log são hierárquicos, ou seja, todos os níveis mostram as mensagens correspondentes a ele e dos níveis acima (em relação à descrição acima), por exemplo, DEBUG mostrará todos os níveis.

LOGS

- ✓ Para utilizá-lo, é preciso definir um nível de depuração e informar a mensagem que será apresentada, conforme mostra o código abaixo:

```
logger.error("Falha ao fechar as conexões.");
```

É importante sempre utilizar o nível de log adequado, para que mensagens desnecessárias não sejam gravadas nos arquivos de log e também facilitamos a visualização do mesmo quando há a necessidade de analisá-lo.

CONHEÇA OUTROS CLIENTES

DEDINI
INDÚSTRIAS DE BASE

- Solicitação Reembolso de Despesas

cootravale®

- Gestão processos administrativos – certificação ISO

Melitta®

- Aprovação de Despesas

Lecom

CONHEÇA OUTROS CLIENTES

MAGAL

TOTVS

CEAGESP

AM
Amsted MAXION

M MAUSA

GROUPE ROULLIER

BETTANIN

Rudnick
Móveis de Qualidade

RICLAN

Unimed
Londrina

HERGEN
PAPER MACHINERY

SYNTHES®

Lecom

CONHEÇA OUTROS CLIENTES

Obrigado!