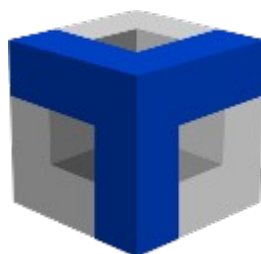


David de Almeida Bezerra Júnior



TETRIS IDE

DESENVOLVIMENTO VISUAL JAVA

Manual do Usuário

Orientação para desenvolvimento de aplicativos
Desktop

Versão 1.3

2020

CONTROLE DE VERSÃO

Data	Versão	Alterações
04/03/2016	1.0	Criação do documento.
01/03/2017	1.1	Adicionado título 14 – TRABALHANDO COM OUTROS BANCOS DE DADOS (FIREBIRD).
29/05/2020	1.3	<ul style="list-style-type: none">• Alterado título 3.6 - Área de trabalho e Visualizador de código-fonte gerado;• Modificada instrução quanto a instalação;• Inserção de descrição de Eventos OnResize, OnMove, OnHide, OnShow e OnPaint;e• Instrução quanto a utilização de Procedures com passagem de parâmetros e retorno.

Índice

1 INTRODUÇÃO.....	1
2 INSTALANDO O TETRISIDE.....	2
2.1 Requisitos mínimos de hardware.....	2
2.2 Requisitos mínimos de softwares.....	2
2.3 Instalação.....	2
2.4 Executando pela primeira vez.....	4
3 CONHECENDO A FERRAMENTA.....	5
3.1 A Janela Principal.....	5
3.2 Barra de Ferramentas.....	5
3.3 Explorador de Projetos.....	6
3.4 Explorador de Janelas.....	6
3.5 Gerenciador de Banco de Dados MySQL.....	7
3.6 Área de trabalho e Visualizador de código-fonte gerado.....	7
3.7 Paleta de Objetos.....	8
3.8 Inspetor de Objetos.....	8
3.9 Lista de Componentes.....	9
3.10 Configurações.....	9
3.11 Componentes Externos.....	10
3.12 Log.....	11
3.13 Exportar.....	11
4 MEU PRIMEIRO PROGRAMA.....	12
4.1 Criando um projeto.....	12
4.2 Modificando o Título da Janela.....	12
4.3 Adicionando Objetos.....	13
4.4 Adicionando Funções.....	14
5 JANELAS.....	16
5.1 Tipos de Janelas.....	16
5.2 Brincando com Janelas.....	16
5.3 Retornando Valores.....	20
5.4 Ícone na Janela.....	24
6 TRABALHANDO COM OBJETOS.....	25
6.1 Objetos no TetrisIDE.....	25
6.1.1 Label.....	26
6.1.2 TextField.....	26
6.1.3 ComboBox.....	26

6.1.4 List.....	27
6.1.5 TextArea.....	27
6.1.6 EditorPane.....	27
6.1.7 Panel.....	28
6.1.8 ToolBar.....	29
6.1.9 TabbedPane.....	29
6.1.10 Image.....	30
6.1.11 Button.....	30
6.1.12 Table.....	31
6.1.13 CheckBox.....	31
6.1.14 RadioButton.....	32
6.1.15 MenuBar.....	32
6.1.16 Timer.....	33
7 FUNÇÕES E EVENTOS.....	34
7.1 Eventos.....	34
7.2 Funções.....	36
7.3 Calculadora.....	39
8 VARIÁVEIS.....	43
8.1 Brincando com variáveis.....	43
9 PROCEDURES.....	46
9.1 Usando Procedures.....	46
10 COMANDOS JAVA.....	53
10.1 O Java por trás do TetrisIDE.....	53
10.2 Cálculo de IMC.....	55
11 OPERAÇÕES COM MYSQL.....	57
11.1 SQL no TetrisIDE.....	57
11.2 Criando o Banco de Dados.....	59
11.3 Inserindo registros.....	62
11.4 Visualizando e selecionando registros.....	65
11.5 Alterando registros.....	67
11.6 Excluir registros.....	69
11.7 Criando janelas para acesso a dados rapidamente.....	70
12 COMPONENTES EXTERNOS (JAR).....	73
12.1 Olhe e sintá!.....	73
13 RELATÓRIOS COM O TETRISREPORT.....	76
13.1 Imprimindo uma Relação de Registros.....	76
13.2 Variáveis no Relatório.....	80
13.3 Código Java dentro do TetrisReport.....	81
14 TRABALHANDO COM OUTROS BANCOS DE DADOS (FIREBIRD).....	84
14.1 Instalando o Firebird.....	84
14.2 Inserindo Dados.....	85

1 INTRODUÇÃO

O TetrisIDE é uma ferramenta RAD (Rapid Application Development) que proporciona um desenvolvimento de aplicações Java para desktop sem codificação manual, suportando completa integração com o Sistema Gerenciador de Banco de Dados MySQL. Com esta simples e elegante IDE, você pode ser 80% mais rápido do que em um desenvolvimento convencional (utilizando um editor com codificação manual), criando sua solução de software.

O TetrisIDE foi criado por David de Almeida Bezerra Júnior buscando reduzir tempo e custo no desenvolvimento de software. Esta tarefa foi feita através da redução da carga de codificação manual que, na programação Java, é mais lenta que em outras linguagens, como Delphi/Lazarus/Object Pascal.

Este manual objetiva-se em transmitir ao usuário as informações necessárias para o correto uso do aplicativo, tornando-o apto para a produção de soluções de softwares.

Embora não seja requisito para o desenvolvimento de alguns programas no TetrisIDE, recomenda-se que o desenvolvedor tenha experiência com desenvolvimento de software na linguagem Java.

2 INSTALANDO O TETRISIDE

Antes da utilização, veremos como efetuar a correta instalação da IDE (Integrated Development Environment). Segue abaixo os requisitos necessários:

2.1 Requisitos mínimos de hardware

- Processador Intel Celeron;
- 1 GB RAM;e
- 161 MB de espaço em disco (somente para o TetrisIDE).

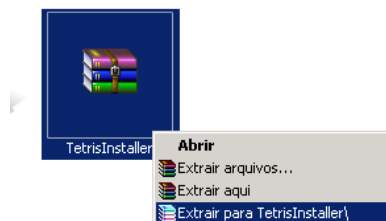
2.2 Requisitos mínimos de softwares

- TetrisInstaller (Instalador do TetrisIDE)
(<http://tetris.analisasoftware.com.br/br/download.php>);
- OpenJDK / Java Development Kit (JDK) 7 ou posterior.
(<http://www.oracle.com/technetwork/pt/java/javase/downloads/index.html> ou <https://jdk.java.net/java-se-ri/7>) (*Embutido no instalador para MS Windows*);e
- MySQL Community 5.1 ou posterior para operações em banco de dados.
(<https://dev.mysql.com/downloads/mysql/>).

2.3 Instalação

Caso use MS Windows, utilize o instalador automático **TetrisInstaller.exe**. Para efetuar a instalação através do instalador multiplataforma, instale o Java Development Kit (JDK) e o MySQL Community baixados e tenha certeza que ambos estão funcionando corretamente.

Extraia o **TetrisInstaller.zip** para o diretório do seu usuário.



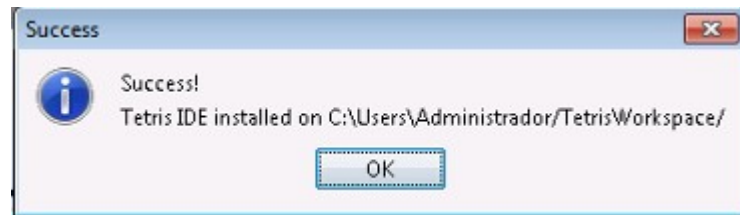
Abra o Arquivo Executável Jar **TetrisInstaller.jar**. Caso o seu JDK não seja instalado (arquivo compactado foi baixado e extraído), talvez você precise executar o instalador através da linha de comando: **(caminho do seu jdk)/bin/java -jar TetrisInstaller.jar**.



Escolha o seu idioma e clique no botão **Install**.



No Microsoft Windows, você pode acessar o aplicativo através do atalho criado na sua Área de trabalho. Você pode acessar também através do diretório TetrisWorkspace criado no diretório home do seu usuário.

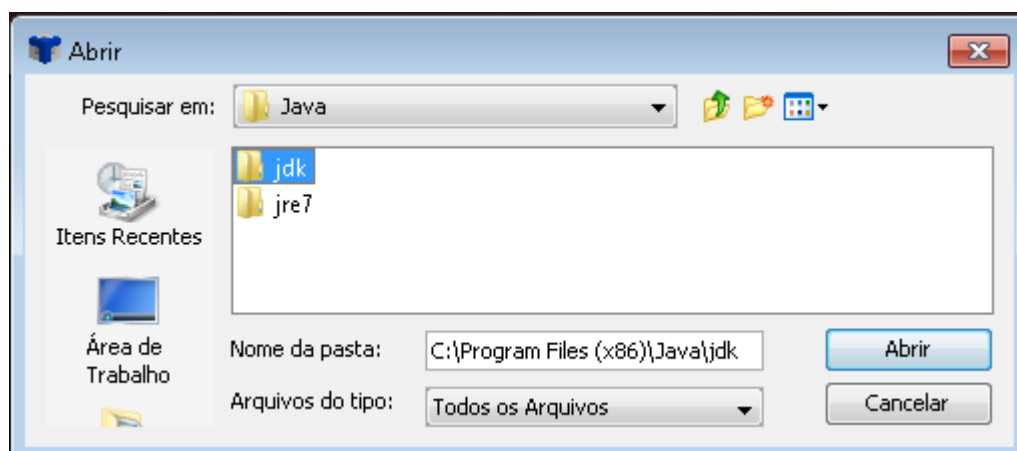


2.4 Executando pela primeira vez

Enquanto inicializa, o TetrisIDE tenta encontrar o JDK instalado em seu computador. Caso ele não encontre, exibirá a mensagem abaixo, perguntando se você deseja selecionar o local de instalação do Java Development Kit.



Clique em Yes e selecione o diretório de instalação do JDK (normalmente, fica em **C:\Arquivos de programas\Java\jdkxxxx**, sendo xxxx a versão do Java).



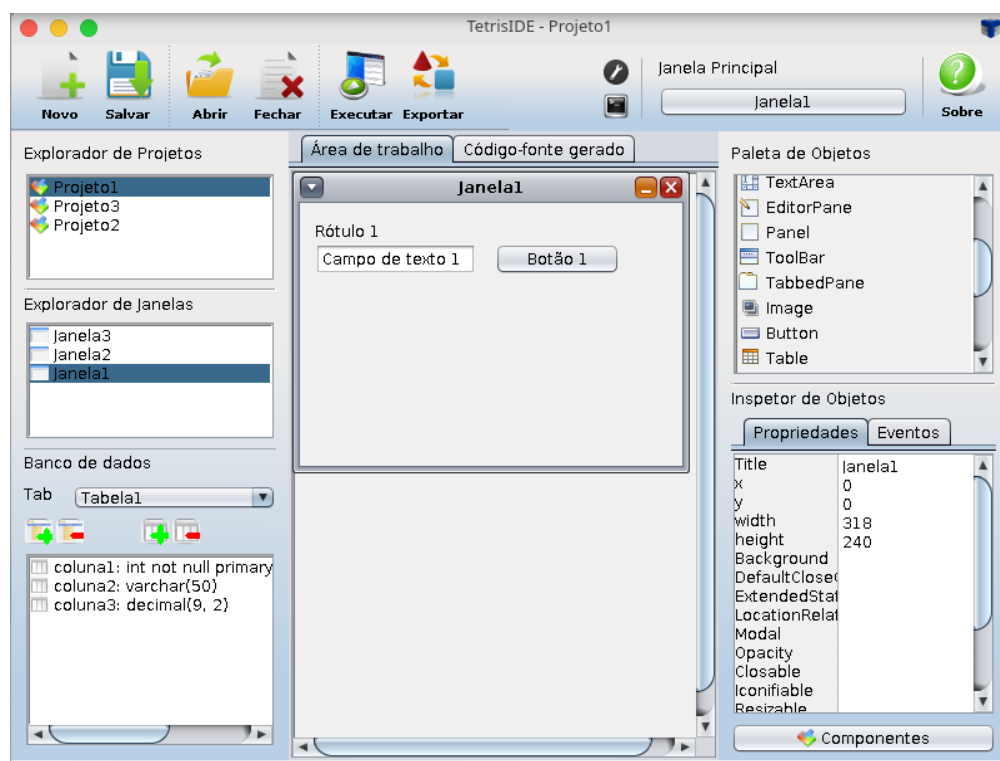
Após os preparativos da primeira inicialização, o TetrisIDE estará pronto para uso.

3 CONHECENDO A FERRAMENTA

O TetrísIDE é uma maravilhosa IDE (Integrated Development Environment) que nos traz um paradigma orientado a visão. Isso significa que você tem um desenvolvimento sem codificação manual.

3.1 A Janela Principal

O TetrísIDE trabalha com uma usabilidade simplista, trazendo uma ferramenta RAD superfácil.



3.2 Barra de Ferramentas

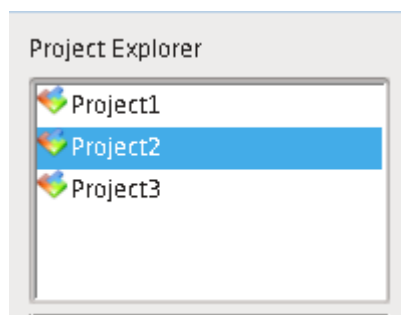
Essa barra contém as funções mais utilizadas da aplicação. Os botões Novo, Salvar, Abrir, Fechar, Executar e Exportar permitem ao usuário manipular projetos. O botão com uma chave inglesa abre a janela de Configurações,

e o botão com um terminal mostra os logs de compilação. Você pode escolher a janela principal do projeto através do botão **Janela Principal**.



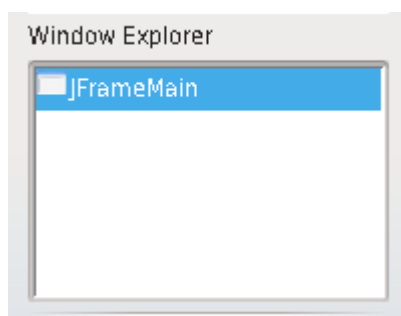
3.3 Explorador de Projetos

Localizado na barra lateral esquerda, o Explorador de Projetos proporciona uma visão e gerenciamento dos seus projetos, criando, renomeando, exportando, importando, abrindo e fechando. Tudo que você precisa fazer é efetuar um clique com o botão direito do mouse. Um menu popup aparecerá para você.



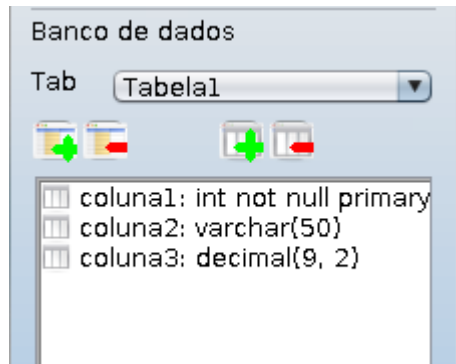
3.4 Explorador de Janelas

Abaixo do Explorador de Projetos, no Explorador de Janelas você pode manipular as janelas do projeto aberto. Pressione o botão direito do mouse e veja as opções.



3.5 Gerenciador de Banco de Dados MySQL

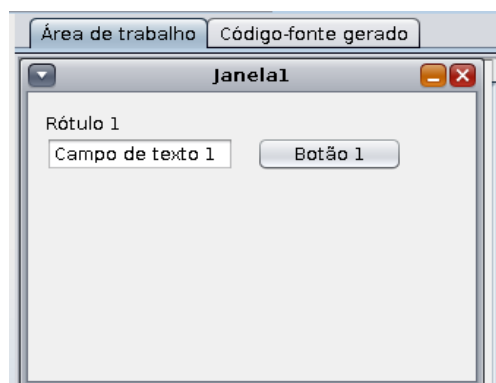
Última seção da barra lateral esquerda, o Gerenciador de Banco de Dados MySQL é um simples controlador para banco de dados MySQL. Aqui, você pode criar e excluir tabelas e colunas. As alterações somente serão gravadas no banco após a execução do projeto.



O TetrísIDE tem uma fabulosa funcionalidade neste controlador, proporcionando ao usuário criar janelas a partir de colunas das tabelas, reduzindo tempo e custo em seu processo de desenvolvimento.

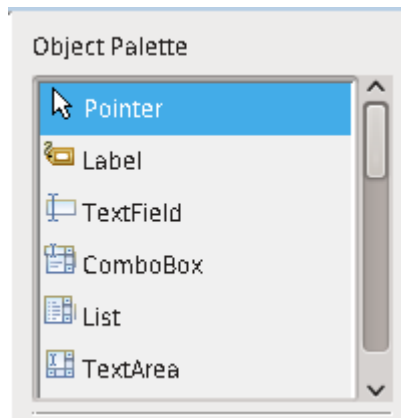
3.6 Área de trabalho e Visualizador de código-fonte gerado

Localizado no centro, a Área de trabalho permite que você monte e desenvolva as janelas. As funções mais populares de edição estão disponíveis aqui. A aba Código-fonte gerado exibe o código Java referente à janela em edição.



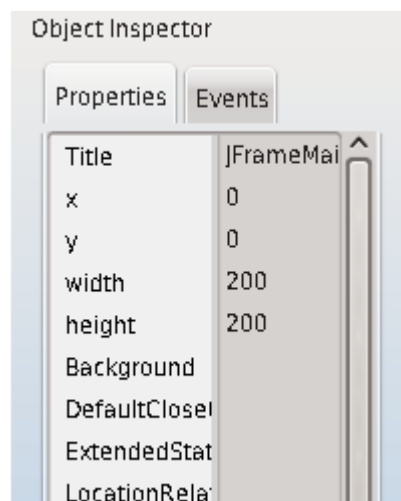
3.7 Paleta de Objetos

Primeira seção da barra lateral direita, a Paleta de Objetos traz objetos para adicionar à janela da Área de trabalho. Quando o **Pointer** está selecionado, você pode mover e redimensionar objetos já adicionados na janela.



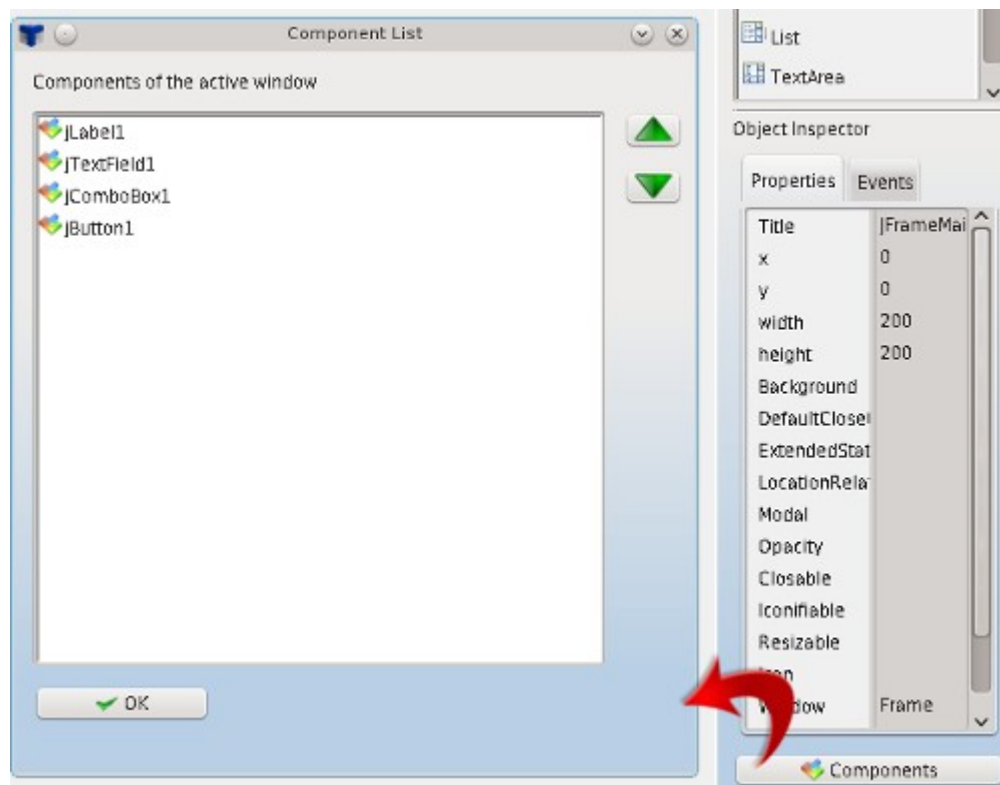
3.8 Inspetor de Objetos

Abaixo da Paleta de Objetos, o Inspetor de Objetos mostra e modifica propriedades do objeto selecionado. Você pode adicionar/remover funções aos eventos também.



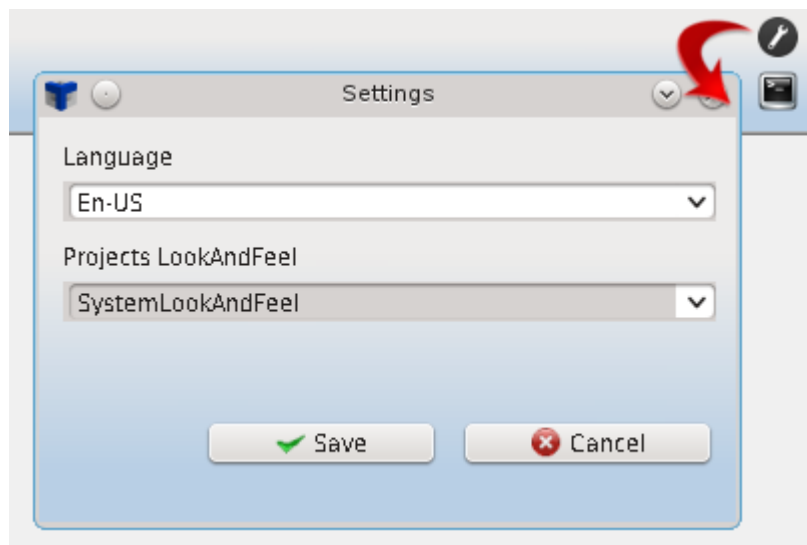
3.9 Lista de Componentes

Mostra uma lista com todos os componentes da janela em edição. Você pode modificar a ordem de criação e mudar o foco de edição. Pressione o botão Componentes na barra lateral direita ou tecle **CTRL + F**, para abrir a janela Lista de Componentes.



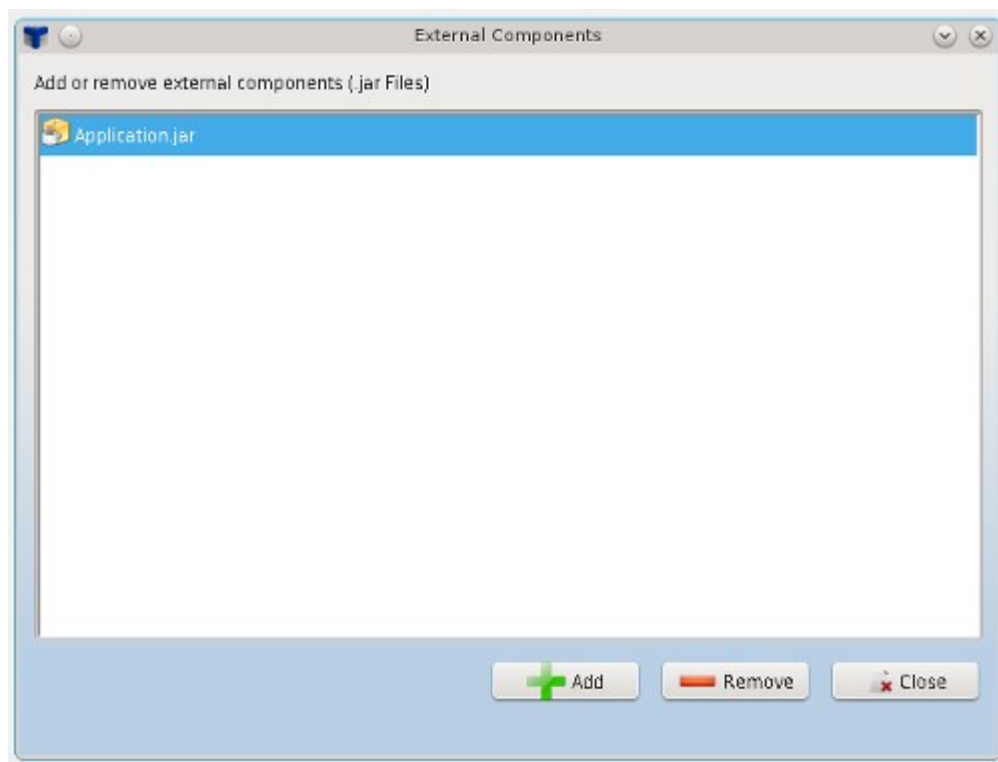
3.10 Configurações

Você pode mudar o idioma do sistema e o LookAndFeel do projeto. Vá no botão Configurações na Barra de ferramentas.



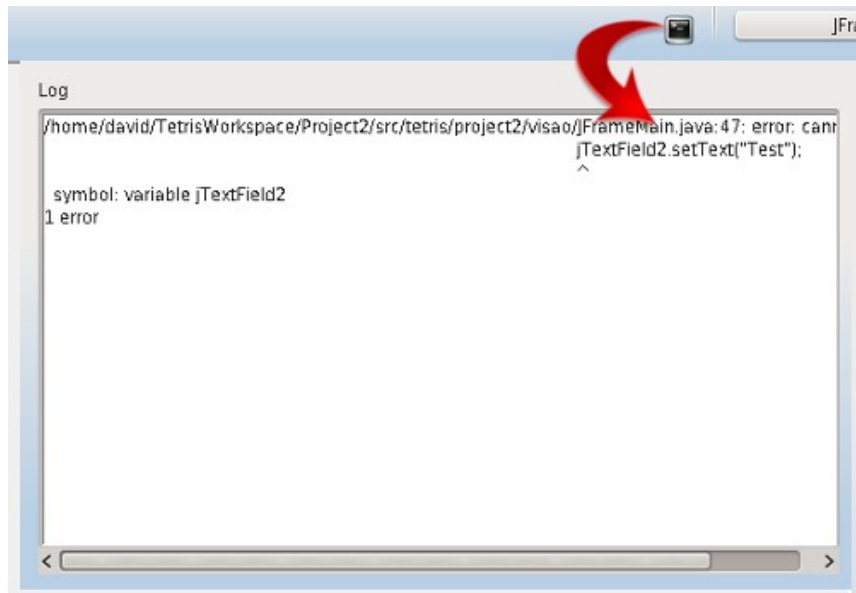
3.11 Componentes Externos

Botão direito do mouse no Explorador de Projetos. Selecione Componentes Externos. A janela que apareceu nos traz um gerenciador de arquivos JAR externos, proporcionando aos seus projetos a possibilidade de ter componentes externos.



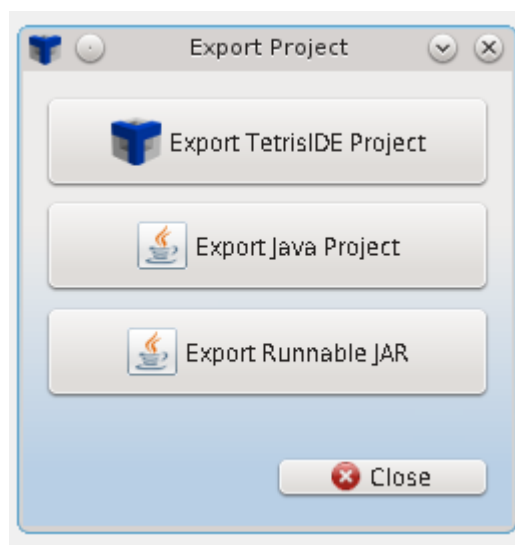
3.12 Log

O Painel de Log mostra os logs de compilação. A cada vez que o usuário executa o projeto, o conteúdo do painel muda.



3.13 Exportar

Você pode exportar seu Projeto TetrisIDE, código Java e Executável JAR. Basta clicar no botão Exportar na Barra de ferramentas ou no menu popup do Explorador de Projetos.

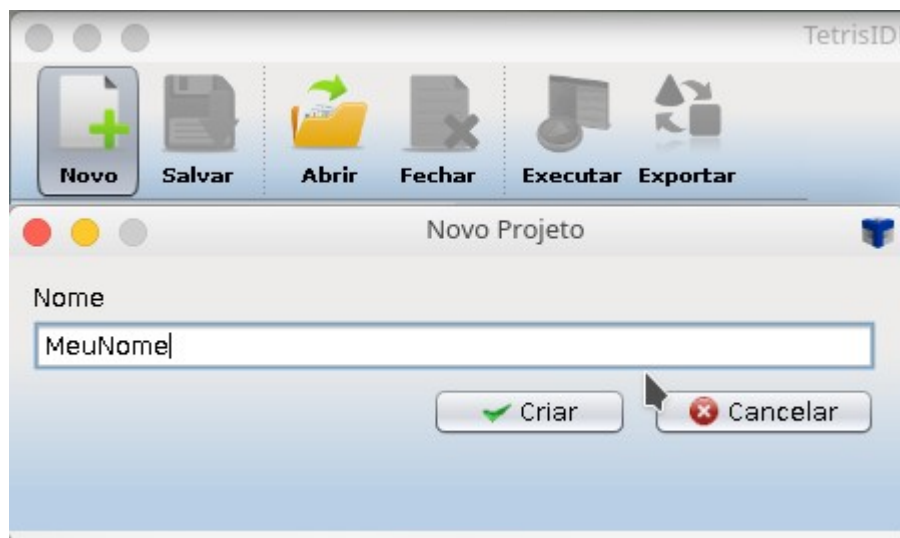


4 MEU PRIMEIRO PROGRAMA

Agora que conhecemos a ferramenta, construiremos nosso primeiro programa com o TetrisIDE. Nós iremos aprender que o desenvolvimento de software nesta ferramenta é muito fácil e rápido. Também veremos conceitos práticos sobre funções e o processo de construção da aplicação.

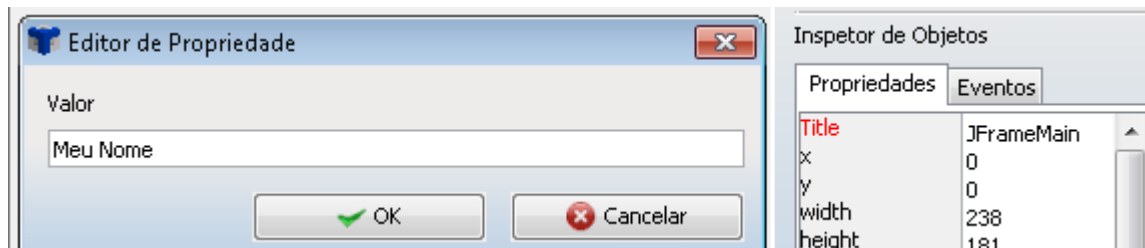
4.1 Criando um projeto

Pressione o botão Novo, na Barra de ferramentas. Crie o projeto com o nome **MeuNome**.



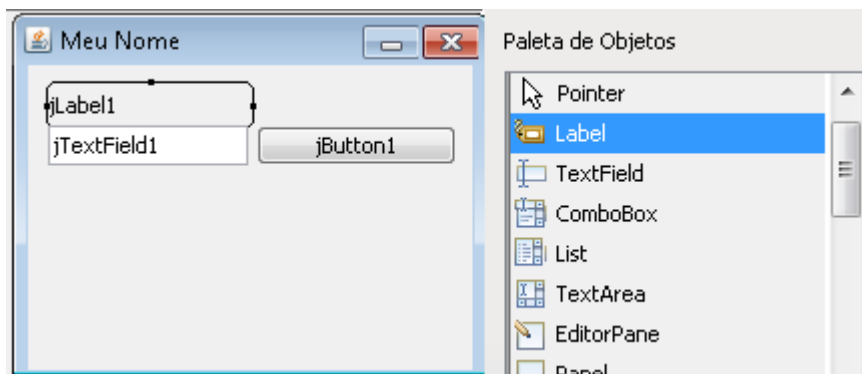
4.2 Modificando o Título da Janela

Efetue um duplo clique na propriedade **Title** (Inspetor de Objetos) e modifique para **Meu Nome**.

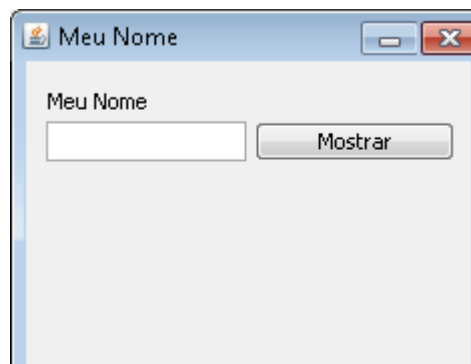


4.3 Adicionando Objetos

Selecione o **Label** na Paleta de Objetos e clique na Janela (Área de trabalho). Faça o mesmo com um **Button** e um **TextField**.

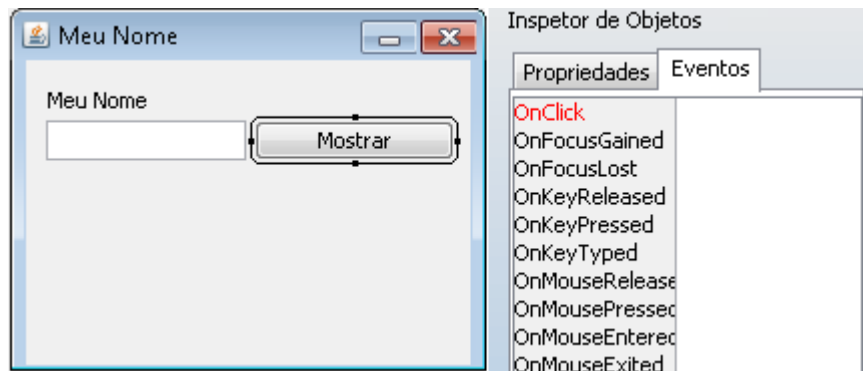


Modifique o nome do Label, TextField e Button para **jLabelMeuNome**, **jTextFieldMeuNome** e **jButtonMeuNome** respectivamente. Apague a propriedade **Text** do **jTextFieldMeuNome** e escreva **Meu Nome** na do **jLabelMeuNome**. No **jButtonMeuNome**, modifique a propriedade **Text** para **Mostrar**. Faça tudo isso no Inspetor de Objetos.



4.4 Adicionando Funções

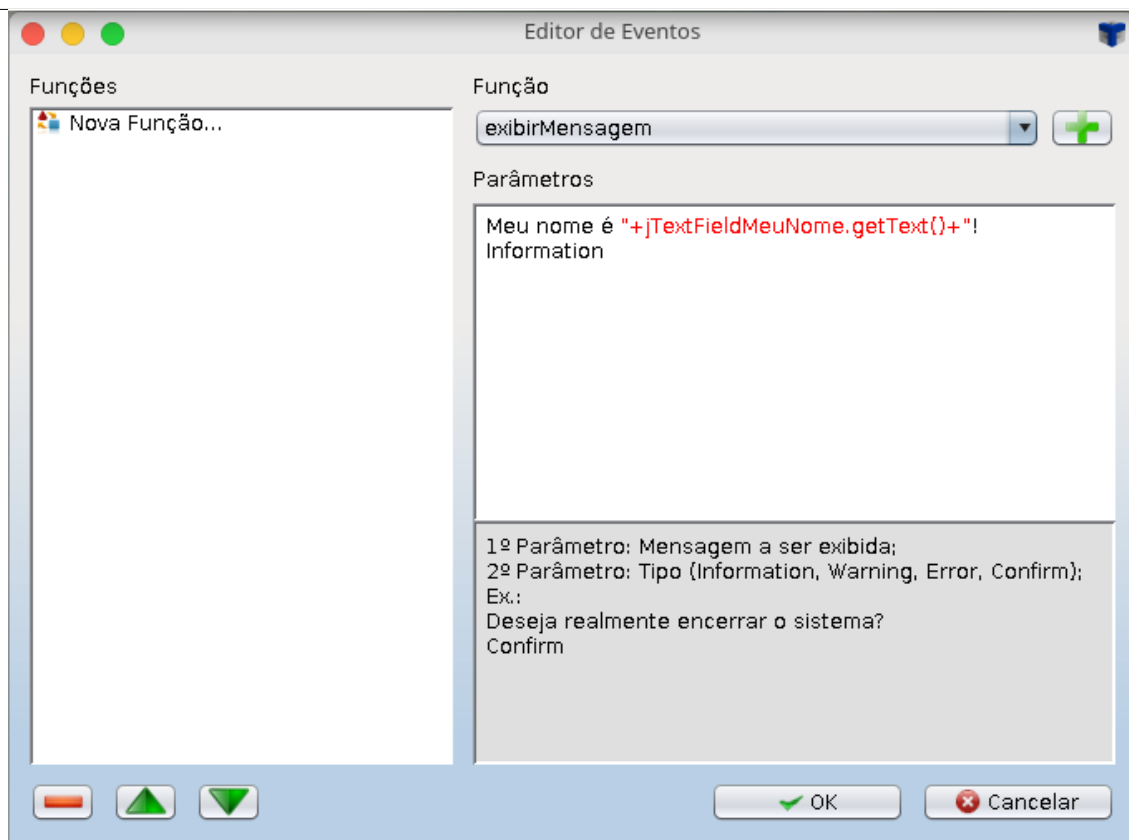
Clique no **jButtonMeuNome** e vá para a aba **Eventos** (Inspetor de Objetos). Efetue duplo clique no evento **OnClick**.



Selecione a função `exibirMensagem` (campo Função) e digite no campo Parâmetros:

Meu nome é `" + jTextFieldMeuNome.getText() + "`!

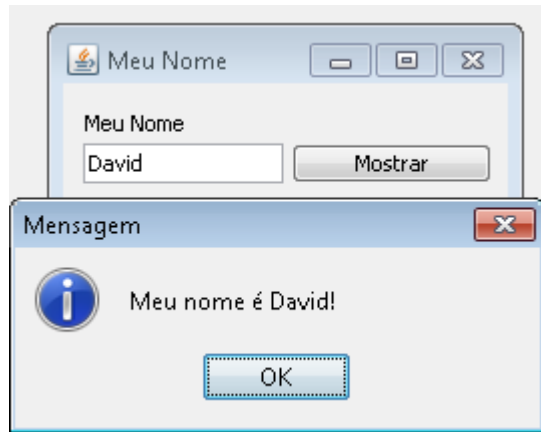
Information



Observe a dica abaixo do campo Parâmetros. Toda função selecionada terá uma dica.

Pressione o botão Adicionar (+) e o botão OK.

Salve e **Execute** sua aplicação, clicando nos respectivos botões na Barra de ferramentas ou pressionando **CTRL + S** e **F9**.



Perceba a forma como o valor contido na propriedade **Text** do **jTextFieldMeuNome** foi passado para a função **exibirMensagem**:

Meu nome é **"`+jTextFieldMeuNome.getText()+`"**

Toda vez que se deseje concatenar um valor a um parâmetro de natureza string (texto), deve-se colocar entre **"`+` e `+`"**.

O segundo parâmetro da função é uma das opções fornecidas pela dica abaixo do campo: **Information**, Warning, Error ou Confirm. São os tipos de diálogo de mensagem disponíveis na classe **JOptionPane**, do pacote **javax.swing** do Java.

5 JANELAS

Desde o surgimento dos primeiros Sistemas Operacionais com interface gráfica, as janelas são um dos principais elementos de interação. Sendo o TetrísIDE orientado a visão, precisamos saber como podemos trabalhar com janelas nesta ferramenta RAD.

5.1 Tipos de Janelas

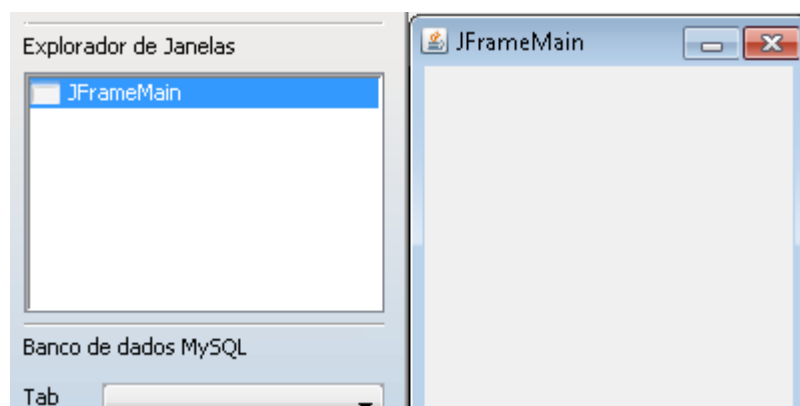
As janelas no TetrísIDE podem ser de três tipos:

- Frame (`javax.swing.JFrame`);
- Diálogo (`javax.swing.JDialog`);
- Frame Interno (`javax.swing.JInternalFrame`);

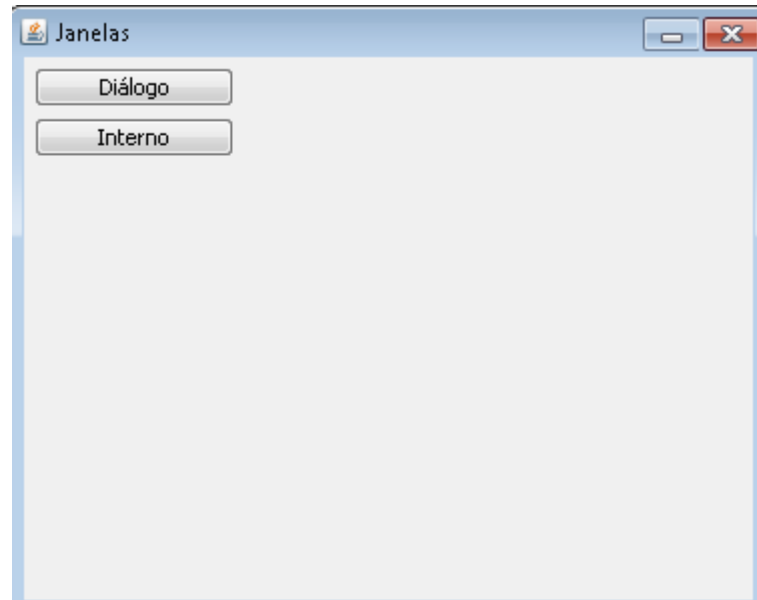
Os Frames são janelas independentes, utilizados quando queremos abrir janelas em nossa aplicação que não ficarão presas em outras janelas. Diálogos são janelas abertas, normalmente, para capturar uma resposta do usuário, ou informar algo. Já os Frames Internos são janelas que ficarão presas dentro da janela principal do sistema.

5.2 Brincando com Janelas

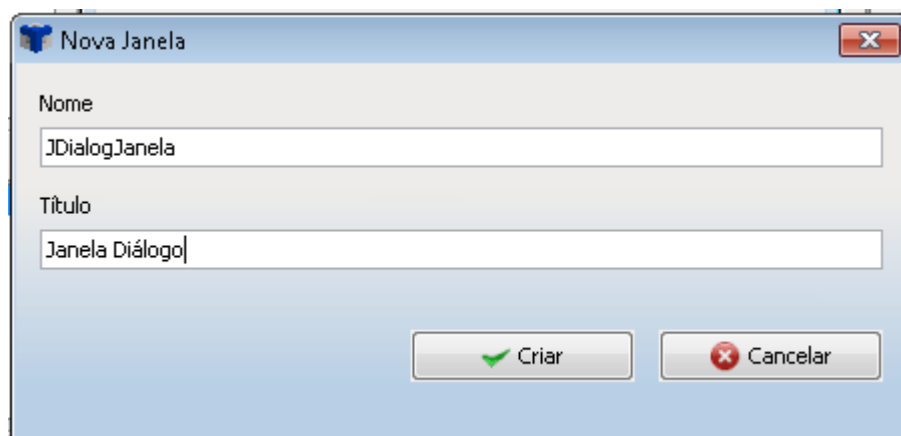
Crie um projeto chamado **Janelas**. Perceba que o TetrísIDE já cria uma janela principal chamada **JFrameMain**. Ela é do tipo Frame.



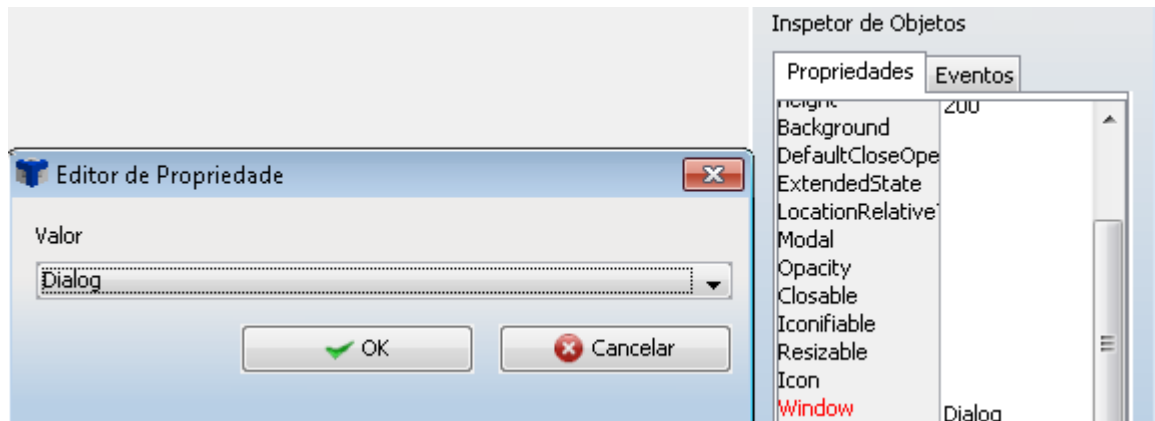
Adicione dois **Button** na **JFrameMain** e mude a propriedade **Text** de cada um para **Diálogo** e **Interno**. Mude também as propriedades **Title**, **width** e **height** da **JFrameMain** para **Janelas**, **400** e **300**.



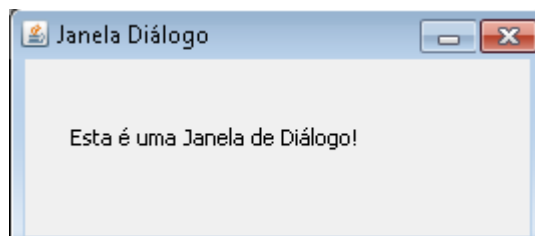
Clique com o botão direito do mouse no Explorador de Janelas e selecione a opção Nova. Na janela que apareceu, digite no campo Nome e Título os valores **JDialogJanela** e **Janela Diálogo**, respectivamente.



Na janela criada, mude a propriedade **Window** para **Dialog**.

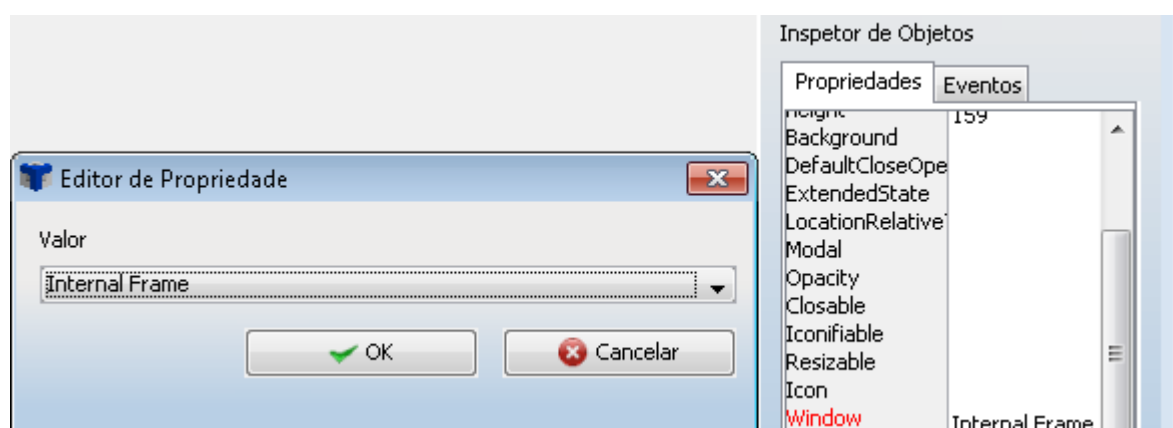


Adicione um **Label** e mude a propriedade **Text** para: **Esta é uma Janela de Diálogo.**

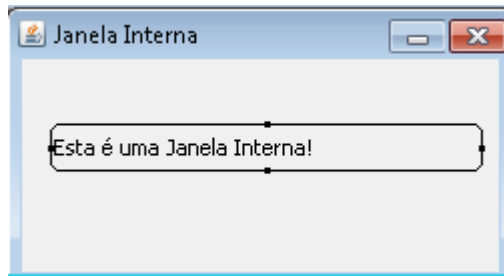


Agora, crie uma outra janela com Nome e Título **JInternalFrameJanela** e **Janela Interna**, respectivamente.

Mude a propriedade **Window** para **Internal Frame**.



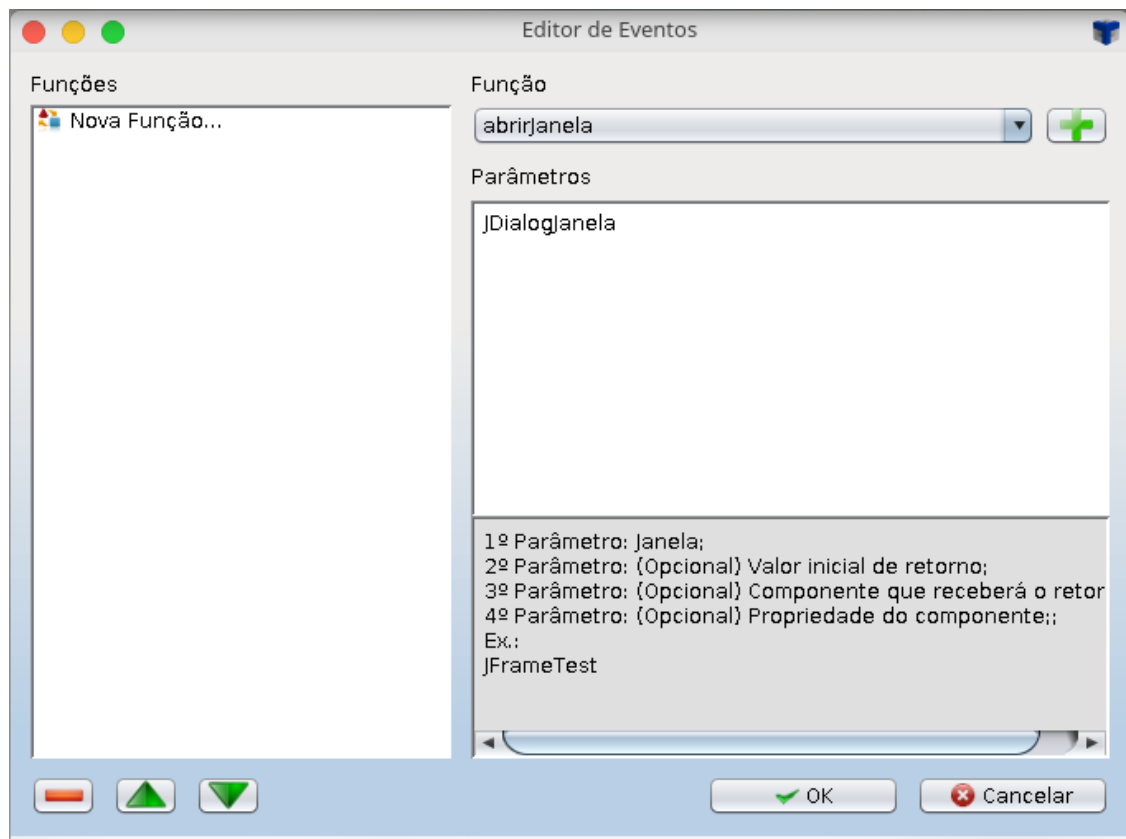
Adicione um **Label** e mude a propriedade **Text** para: **Esta é uma Janela Interna.**



Volte a **JFrameMain**, clique no botão Diálogo e efetue duplo clique no evento **OnClick**, na aba Eventos do Inspetor de Objetos.

Adicione uma função **abrirJanela** com os seguintes parâmetros:

JDialogJanela

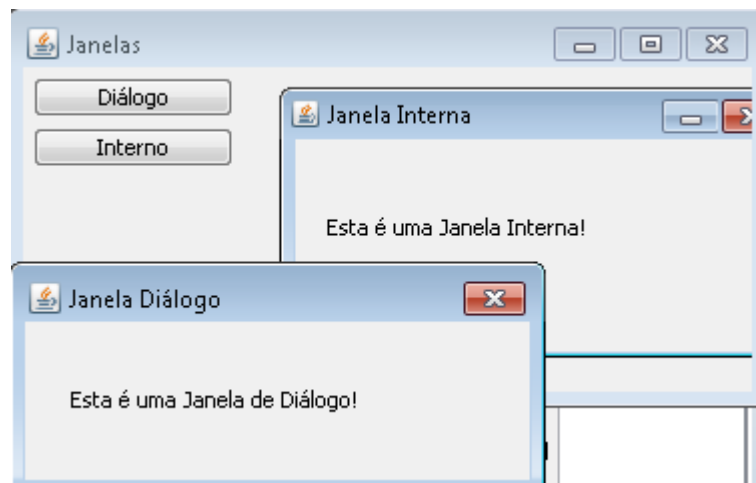


Como mostra a dica, a função **abrirJanela** pode receber 4 parâmetros, sendo o primeiro a própria janela que será aberta, e o segundo, terceiro e quarto opcionais, utilizados quando queremos trabalhar com janelas que retornam valores para a janela que a abriu (veremos mais adiante).

Faça o mesmo com o botão Interno, passando o seguinte parâmetro:

JInternalFrameJanela

Salve e **Execute** o projeto, pressionando os respectivos botões na Barra de ferramentas ou pressionando **CTRL + S** e **F9**.

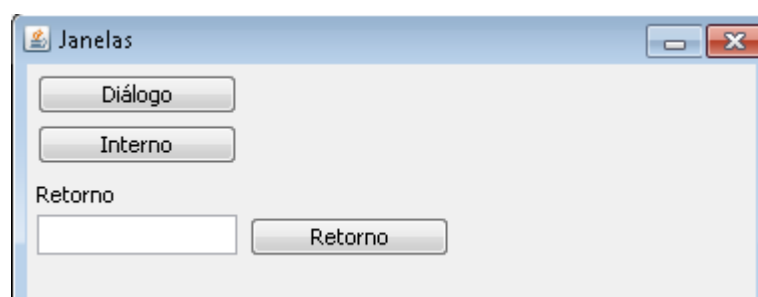


5.3 Retornando Valores

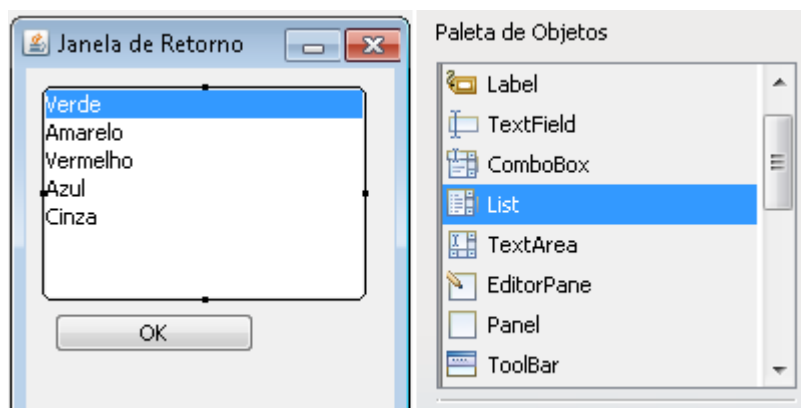
Muitas vezes, as aplicações necessitam solicitar ao usuário alguma informação enquanto um processo está em execução em uma janela. Uma opção bastante utilizada para esta tarefa são as janelas de diálogo modais.

Janelas modais são aquelas que, quando chamadas, colocam a tarefa que está sendo executada em modo de espera até que ela seja fechada. Normalmente, retornam algum valor para a janela que a abriu.

No projeto anterior, adicione um **Label**, um **TextField** e um **Button** à **JFrameMain** e renomeie para **jLabelRetorno**, **jTextFieldRetorno** e **jButtonRetorno**, respectivamente. Mude também a propriedade **Text** do **jLabelRetorno** e **jButtonRetorno** para **Retorno**. Apague o valor da propriedade **Text** do **jTextFieldRetorno**.



Crie uma janela com Nome **JDialogRetorno** e Título **Janela de Retorno**. Adicione um List à janela e mude o nome para **jListRetorno**. Adicione um botão e mude o nome e o Text para **jButtonOK** e **OK**.



Insira na propriedade **Items** do **jListRetorno**, no Inspetor de Objetos:

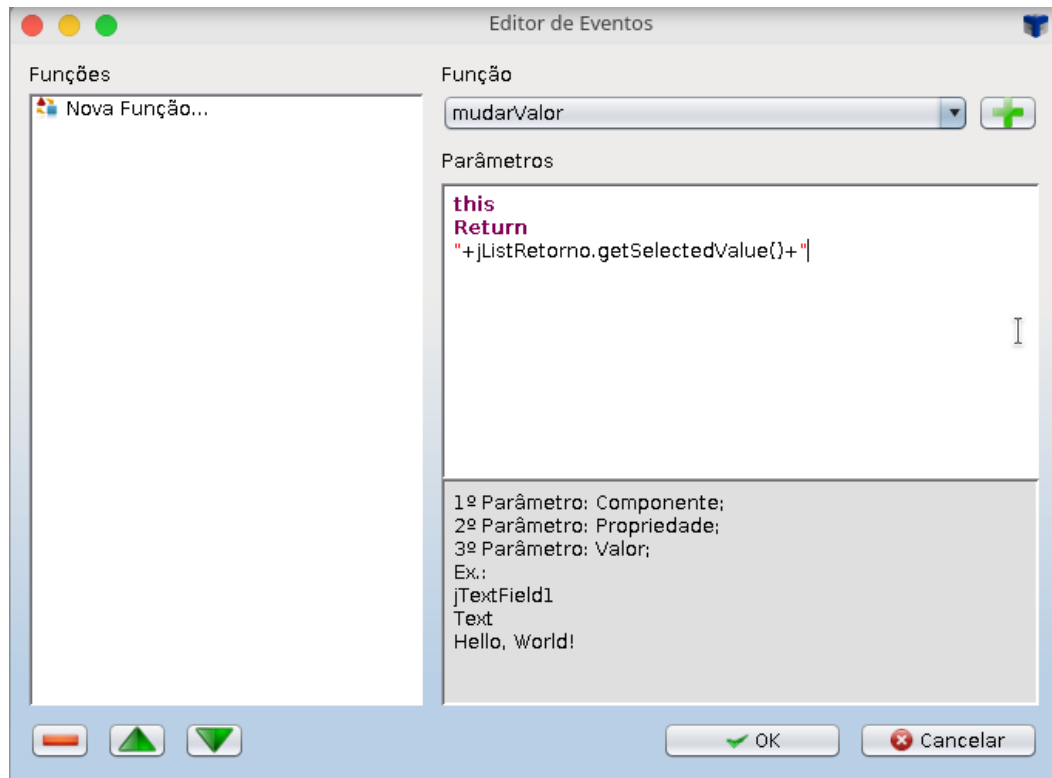
```
Verde  
Amarelo  
Vermelho  
Azul  
Cinza
```

Mude a propriedade **Window** da janela para **Dialog** e a propriedade **Modal** para **true**. Aqui, nós dizemos à janela para ela pausar a tarefa que a abriu enquanto ela estiver aberta. Mude também a propriedade **Resizable** para **false** e **LocationRelativeTo** para **Center**. Estas duas travam o redimensionamento da janela e colocam ela no centro da tela.

Clique no jButtonOK e adicione a função mudarValor com os seguintes parâmetros no evento OnClick, da aba Eventos do Inspetor de Objetos:

```
this  
Return  
"+jListRetorno.getSelectedValue()+"
```

Observe a dica para essa função. É possível mudar a propriedade de um componente passando 3 parâmetros: o componente, a propriedade e o novo valor.



Adicione uma função **fecharJanela** sem parâmetros e clique em OK.

No evento **OnShow** da janela, adicione uma função **mudarValor** com os seguintes parâmetros:

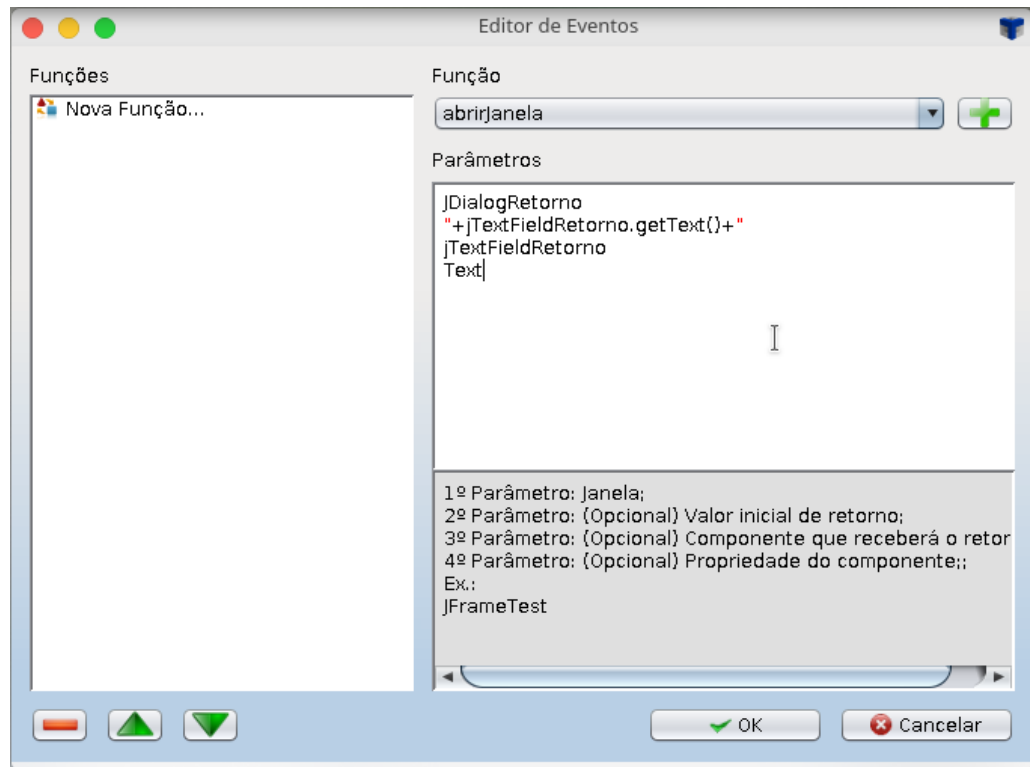
```
jListRetorno
SelectedValue
getReturn()
```

Nesta função, ao abrir a janela **JDialogRetorno**, estamos dizendo ao programa para selecionar o elemento com o valor igual ao do parâmetro passado como valor inicial do retorno ao abrir a janela.

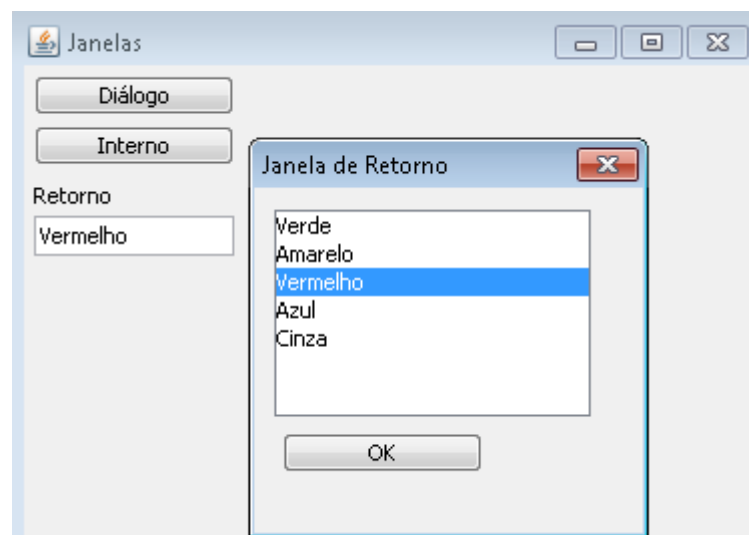
Volte a **JFrameMain** e, no **jButtonRetorno**, no evento **OnClick**, adicione uma função **abrirJanela** com os parâmetros:

```
JDialogRetorno
"+jTextFieldRetorno.getText()+"
jTextFieldRetorno
Text
```

No primeiro parâmetro, dizemos qual janela será aberta. No segundo, qual será o valor inicial do retorno. O terceiro e quarto diz à janela o componente e sua respectiva propriedade que receberá o retorno.



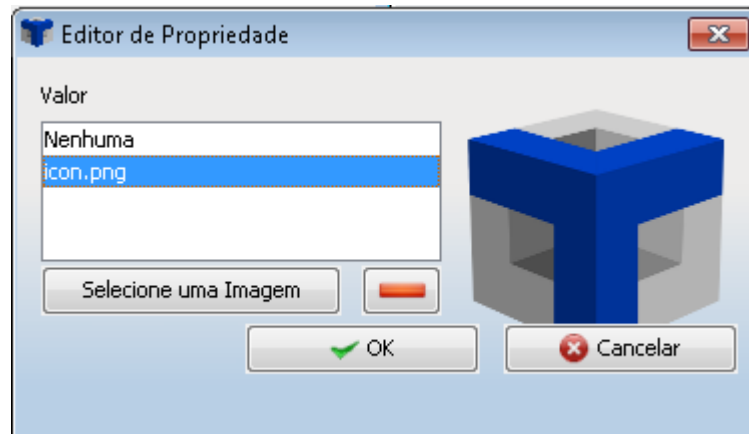
Salve e Execute o projeto, pressionando os respectivos botões na Barra de ferramentas ou pressionando **CTRL + S** e **F9**.



Perceba que, quando clicamos no botão Retorno, a janela de retorno é aberta e não deixa mexermos em mais nada até que ela seja fechada. Esse é o comportamento de uma janela modal.

5.4 Ícone na Janela

Ainda na JFrameMain, efetue duplo clique na propriedade **Icon**. Na janela que apareceu, pressione o botão **Selecione uma imagem** e escolha um ícone a seu gosto para a janela principal do sistema.



Salve e Execute o projeto, pressionando os respectivos botões na Barra de ferramentas ou pressionando **CTRL + S** e **F9**.

6 TRABALHANDO COM OBJETOS

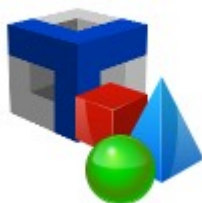
A linguagem Java é Orientada a Objetos, logo, nós acabamos trabalhando com objetos de qualquer forma. Abaixo veremos como trabalhar com objetos no TetrísIDE.

Se você não conhece o Paradigma Orientado a Objetos, dê uma olhada em

https://pt.wikipedia.org/wiki/Object-oriented_programming.

6.1 Objetos no TetrísIDE

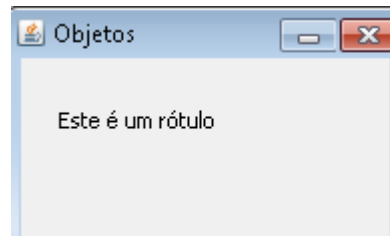
Objetos são representações do mundo real que são formados de atributos e métodos que propõe-lhes sentido. Todo objeto é uma instância de uma classe que, por sua vez, é o molde do objeto. Imagine que tenho uma classe chamada **Botao** que tem como argumentos **texto**, **altura**, **largura**, **posicaoHorizontal**, **posicaoVertical**. Vamos dizer que crio um objeto do tipo **Botao** com o nome **botaoOK**, **texto** OK, **altura** 20, **largura** 80, **posicaoHorizontal** 15 e **posicaoVertical** 15. O objeto **botaoOK** acaba ganhando propriedades definidas pela classe **Botao**.



Visualmente, você pode adicionar objetos na sua janela pela Paleta de Objetos, selecionando um item e clicando na janela da Área de trabalho.

6.1.1 Label

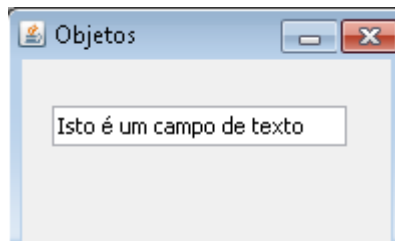
O Label nada mais é do que um rótulo. Define-se por um texto estático utilizado para informar ou indicar algo. A classe do Java que o define no TetrisIDE é a **javax.swing.JLabel**.



É possível ajustar diversas propriedades pelo Inspetor de Objetos, como a inserção de imagens no **Label**, através da propriedade **Icon**.

6.1.2 TextField

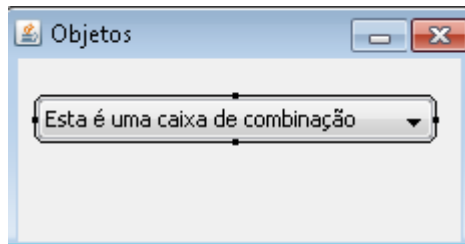
Um campo de texto linear que proporciona a digitação do usuário. A classe que o define, no caso do TetrisIDE, é a **javax.swing.JFormattedTextField**.



O **TextField** possui a propriedade **Mask**, sendo possível definir uma máscara para o campo de texto. Exemplo: CPF – 999.999.999-99.

6.1.3 ComboBox

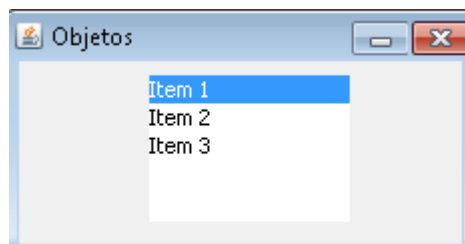
Como o próprio nome sugere, o ComboBox é uma caixa de combinação, podendo armazenar uma lista de valores selecionáveis. Definido pela classe **javax.swing.JComboBox**.



Os itens são adicionados na propriedade **Items** do Inspetor de Objetos.

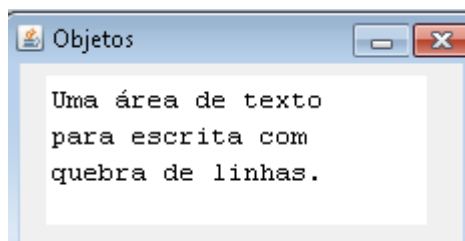
6.1.4 List

A lista funciona como um ComboBox, porém, mostrando mais de um item. Sua classe é a **javax.swing.JList**.



6.1.5 TextArea

Uma área de texto que funciona como um TextField com quebra de linhas, possibilitando ao usuário escrever textos. Definido por **javax.swing.JTextArea**.

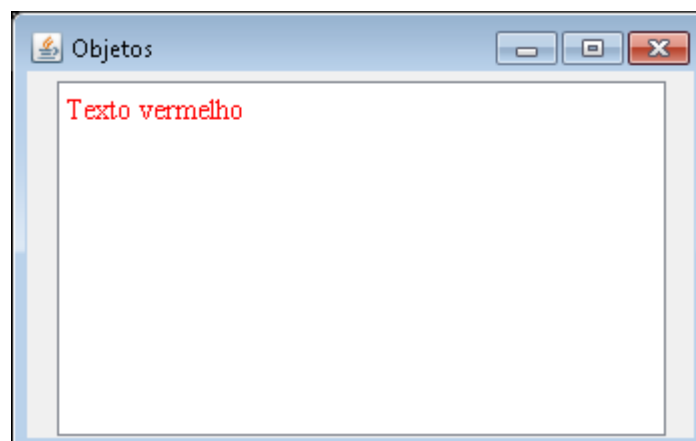


6.1.6 EditorPane

Este painel de edição mostra o texto com certa estilização, dependendo do valor presente na propriedade **ContentType**. Tomando como exemplo, se o usuário mudar a propriedade **ContentType** para **text/html**, e a propriedade **Text** para:

```
<!DOCTYPE html>
<html>
  <head>
  </head>
  <body>
    <font color='red'>Texto vermelho</font>
  </body>
</html>
```

Terá como resultado um **Texto vermelho** exibido no **EditorPane**, quando em execução.



Através da propriedade **Page**, é possível também exibir o conteúdo de uma página da web. A classe que o define é a **javax.swing.JEditorPane**.

6.1.7 Panel

Sendo um objeto que pode conter outros objetos graficamente, o Panel nada mais é do que um painel, onde pode-se estilizar diversos aspectos, como borda, na propriedade **Border**, cor de fundo, propriedade **Background** e imagem de fundo, na propriedade **Icon**. A sua classe é a **javax.swing.JPanel**.

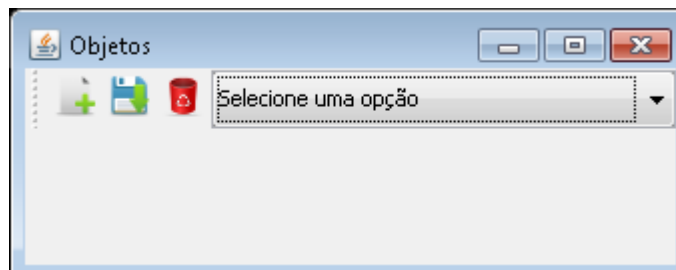
Se definirmos um valor para a propriedade **Text**, o **Panel** ganha uma

borda com título, o que torna muito útil quando queremos agrupar objetos rotulados em uma janela.



6.1.8 ToolBar

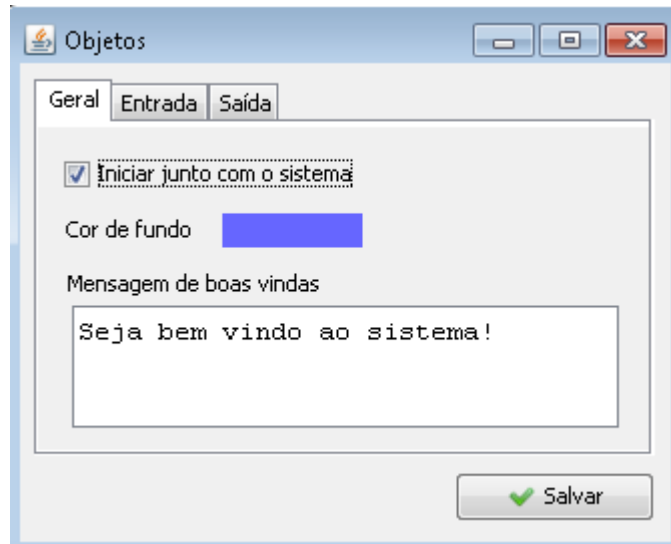
A barra de ferramentas é muito útil para agruparmos objetos, como botões e caixas de combinação. Como o seu nome sugere, ao adicionarmos uma `ToolBar` na janela, criamos uma barra para agruparmos ferramentas. Definida por `javax.swing.JToolBar`.



6.1.9 TabbedPane

O **TabbedPane** é um painel com abas, podendo agrupar muitas informações categorizadas em um espaço limitado. Imagine que você tem que montar uma tela de checkup onde o usuário precisará marcar 1000 campos de checagem. Com certeza, estes mil objetos não caberiam na tela de forma utilizável. Para isso, você criaria um **TabbedPane** com todos os mil campos dispostos de

forma categorizada em abas. A classe que rege o **TabbedPane** é a **javax.swing.JTabbedPane**. A propriedade utilizada para definir as abas no Inspetor de Objetos é a **Tab**.



6.1.10 Image

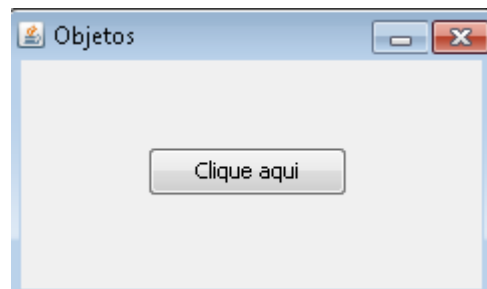
O **Image** comporta-se como um **Label**. Ao inserirmos na janela, o objeto pede para selecionarmos uma imagem em nosso computador. Após selecionado, o objeto assume as propriedades da imagem (altura e largura). Definida por **javax.swing.JLabel**.



6.1.11 Button

O **Button** é utilizado para executar tarefas ao clique do usuário. Sua

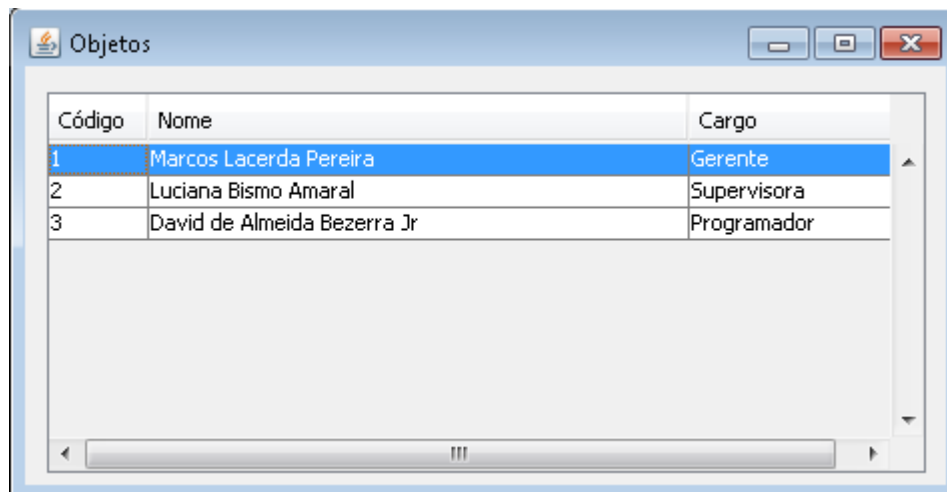
classe é a **javax.swing.JButton**.



6.1.12 Table

Talvez um dos objetos mais complicados de se trabalhar no Java, o **Table** no TetrISIDE acaba tornando muito fácil criar e popular tabelas com informações de banco de dados. Apoiada na classe **javax.swing.JTable**, traz funções específicas que torna sua composição simplificada.

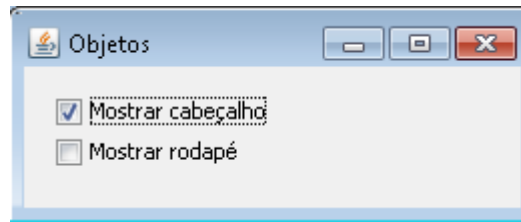
Posteriormente, neste manual, veremos como manipular **Tables** em operações com banco de dados.



6.1.13 CheckBox

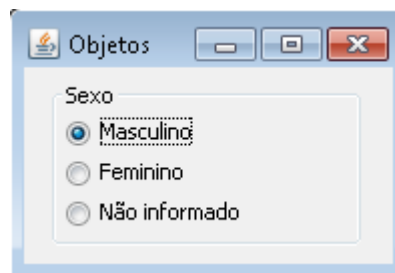
CheckBox são caixas de checagem provenientes da classe

`javax.swing.JCheckBox`.



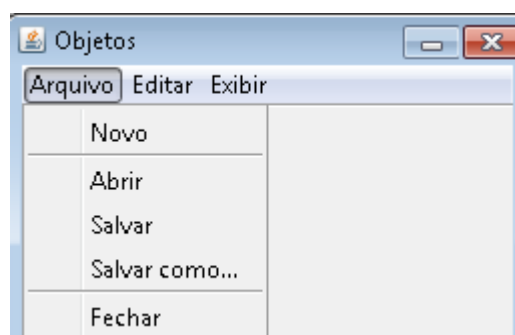
6.1.14 RadioButton

Diferente do **CheckBox**, este grupo de botões é utilizado quando queremos que o usuário marque somente uma das alternativas apresentadas. Sua classe é herdada da `javax.swing.JPanel`, e pode-se obter o seu `ButtonGroup` através do método `getButtonGroup()`. Também é possível obter um `ArrayList` com os `RadioButtons` através do método `getArrayListJRadioButtons()`. Efetue duplo clique na propriedade **RadioButtons** para adicionar os itens.



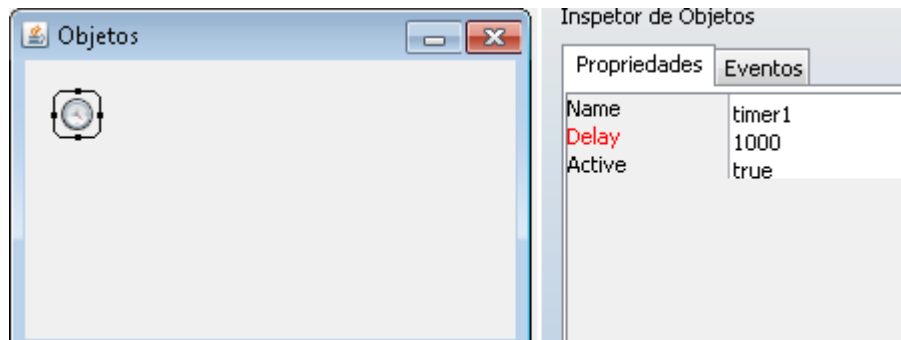
6.1.15 MenuBar

Cria uma barra de menu na janela, podendo o usuário inserir **Menus**, **MenuItems** e **Separators**. Sua classe é a `javax.swing.JMenuBar`.



6.1.16 Timer

O **Timer** é um objeto não gráfico (não aparece na janela quando em execução) que executa funções definidas no evento OnTimer, no Inspetor de Objetos, de acordo com o intervalo definido na propriedade **Delay** do Inspetor de Objetos. O **Delay** é definido em milissegundos. A classe do **Timer** é a **javax.swing.Timer**.

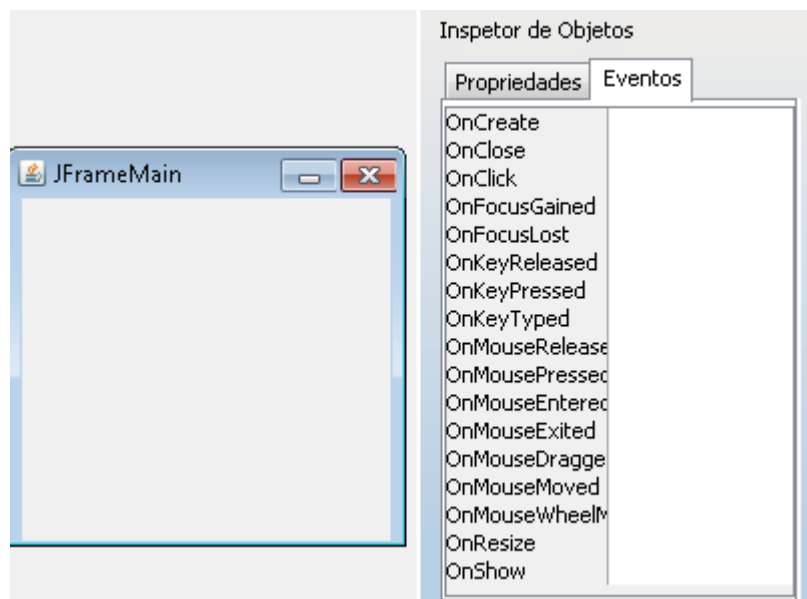


7 FUNÇÕES E EVENTOS

Os eventos no Java são contemplados por listeners (escutadores) e seus métodos. O TetrisIDE, buscando facilidade e agilidade, transformou estes fatores em eventos e funções, sendo que, cada evento pode conter mais de uma função, que são executadas de forma procedural e estruturada (não se esquivando do conceito orientado a objetos).

7.1 Eventos

Observe os eventos de uma janela:



Um evento é um acontecimento no sistema. Um botão sendo clicado gera um evento **OnClick**, que por sua vez, tendo funções definidas, executa-as.

Segue abaixo o que representa cada evento da janela:

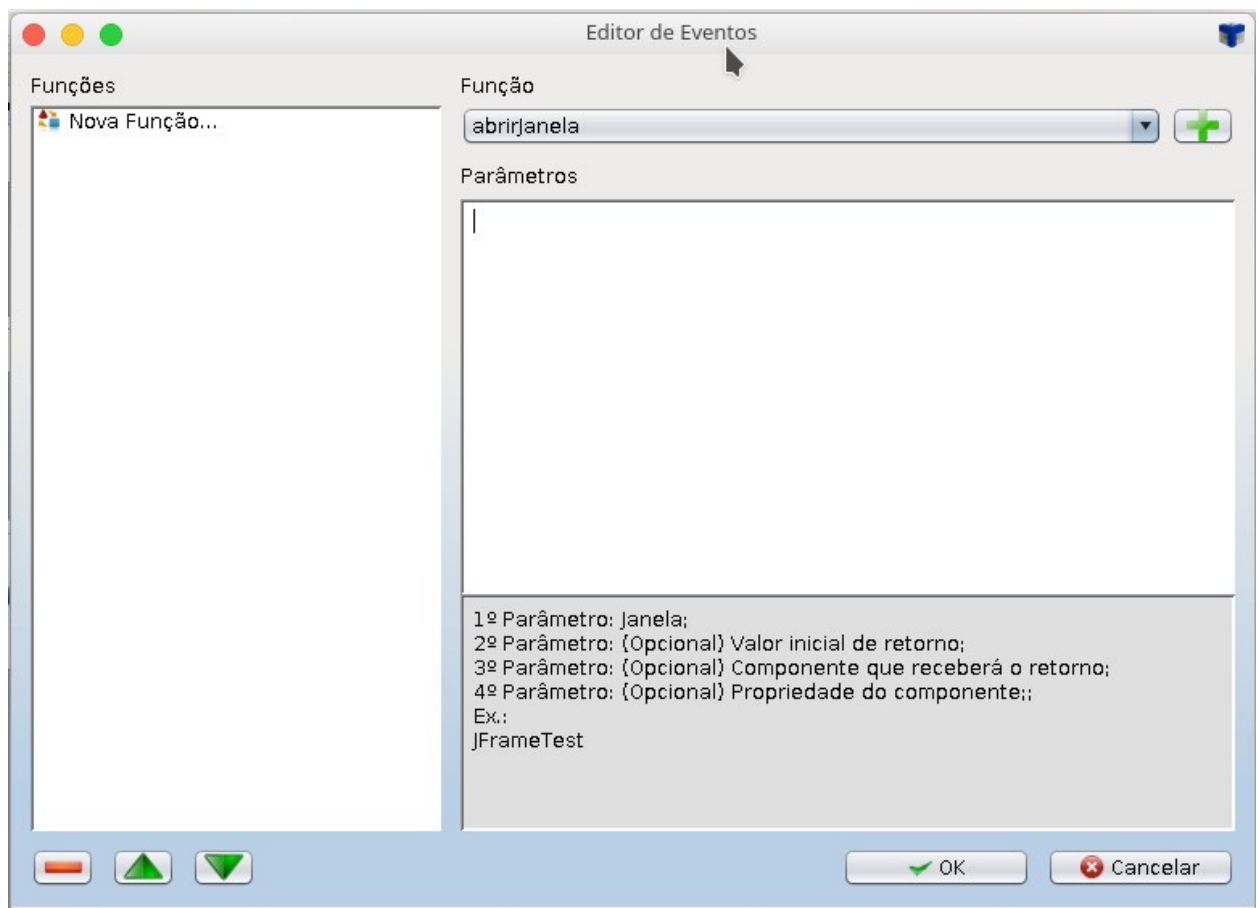
Evento	Descrição
OnCreate	No ato da criação da janela na memória em tempo de execução.
OnClose	Quando a janela é fechada.
OnClick	Clique do mouse.

OnFocusGained	Quando ganha foco.
OnFocusLost	Quando perde o foco.
OnKeyReleased	Quando o pressionamento da tecla é realizado.
OnKeyPressed	Quando a tecla é pressionada.
OnKeyTyped	Quando uma tecla é teclada.
OnMouseReleased	Quando um clique do mouse é realizado.
OnMousePressed	Quando um botão do mouse é pressionado.
OnMouseEntered	Quando o ponteiro do mouse entra na janela.
OnMouseExited	Quando o ponteiro do mouse sai da janela.
OnMouseDragged	Quando pressiona-se o botão do mouse e arrasta a janela (interno).
OnMouseMoved	Ao movimentar do ponteiro do mouse sobre a janela.
OnMouseWheelMoved	Ao girar a roda central do mouse.
OnResize	Ao redimensionar o componente.
OnShow	Quando a janela fica visível.
OnPaint	<p>Ao desenhar o componente na tela. Método contém uma variável java.awt.Graphics chamada de g. Através da função comandoJava, é possível desenhar no componente com código Java. Exemplo de parâmetro em função comandoJava em método OnPaint:</p> <pre><i>g.drawString("Texto desenhado!", 100, 100);</i></pre>
OnMove	Ao mover o componente na tela.
OnHide	Ao tornar o componente invisível.

Cada objeto tem a sua coleção de eventos, dependendo do seu tipo e individualidade.

7.2 Funções

Funções são ações predefinidas no TetrisIDE que facilitam a execução de alguma tarefa. Todas as funções podem ser encontradas no Editor de Eventos, janela que aparece quando efetuamos duplo clique em algum evento. Experimente efetuar um duplo clique no evento **OnShow** de uma janela.



Segue abaixo a lista de funções presentes no TetrisIDE:

Função	Parâmetros/Dica
abrirJanela	1º Parâmetro: Janela; 2º Parâmetro: (Opcional) Valor inicial de retorno;

	3º Parâmetro: (Opcional) Componente que receberá o retorno; 4º Parâmetro: (Opcional) Propriedade do componente; Ex.: JFrameTest
fecharJanela	1º Parâmetro: Janela (Opcional); Ex.: JFrameTest
habilitarCampos	Liste os campos a serem habilitados Ex.: jTextField1 jTextField2 jTextField3
desabilitarCampos	Liste os campos a serem desabilitados Ex.: jTextField1 jTextField2 jTextField3
mudarValor	1º Parâmetro: Componente; 2º Parâmetro: Propriedade; 3º Parâmetro: Valor; Ex.: jTextField1 Text Hello, World!
mudarFoco	1º Parâmetro: Componente; Ex.: jTextField1
selecionarRegistro	1º Parâmetro: Tabela do banco de dados; 2º Parâmetro: Campos selecionados; 3º Parâmetro: Condição de seleção; Ex.: client id, name where id='2'
gravarRegistro	1º Parâmetro: Tabela do banco de dados; 2º Parâmetro: Campos selecionados; 3º Parâmetro: Valores; Ex.: client id, name '2', 'David'
alterarRegistro	1º Parâmetro: Tabela do banco de dados; 2º Parâmetro: Campos selecionados; 3º Parâmetro: Condição de seleção; Ex.:

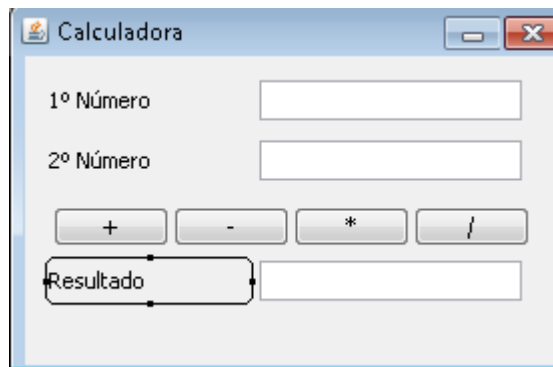
	client id='3', name='David' where id='2'
preencherTabela	1º Parâmetro: jTable; 2º Parâmetro: Tabela do banco de dados; 3º Parâmetro: Campos selecionados; 4º Parâmetro: Condição de seleção; Ex.: jTable1 client id, name where id='2'
excluirRegistro	1º Parâmetro: Tabela do banco de dados; 2º Parâmetro: Condição de seleção; Ex.: client where id='2'
verificarRegistro	1º Parâmetro: Tabela do banco de dados; 2º Parâmetro: Campos selecionados; 3º Parâmetro: Condição de seleção; 4º Parâmetro: (Opcional) Mensagem; 5º Parâmetro - Padrão (==): (Opcional) Comparador (!=, ==); Ex.: client id where id='2' Não há registro! ==
operacaoMatematica	1º Parâmetro: Primeiro número; 2º Parâmetro: Segundo número; 3º Parâmetro: Operação (+, -, *, /); 4º Parâmetro: Componente para receber o resultado; 5º Parâmetro: Propriedade; Ex.: 5 3 + jTextField1 Text
exibirMensagem	1º Parâmetro: Mensagem a ser exibida; 2º Parâmetro: Tipo (Information, Warning, Error, Confirm); Ex.: Deseja realmente encerrar o sistema? Confirm
verificarValor	1º Parâmetro: Componente;

	2º Parâmetro: Propriedade; 3º Parâmetro: Comparador (==, !=, >, <, >=, <=); 4º Parâmetro: Valor para comparar; 5º Parâmetro: (Opcional) Mensagem; Ex.: jTextField1 Text == Test O valor é 'Test'!
executarProcedure	1º Parâmetro: Procedure; 2º Parâmetro: (Opcional) Parâmetro; 3º Parâmetro: (Opcional) Parâmetro; Ex.: procedureSum 1 2
visualizarRelatorio	1º Parâmetro: Relatório; 2º Parâmetro: Tabela do banco de dados; 3º Parâmetro: Campos selecionados; 4º Parâmetro: Condição de seleção; Ex.: report1 client id, name where id=2
imprimirRelatorio	1º Parâmetro: Relatório; 2º Parâmetro: Tabela do banco de dados; 3º Parâmetro: Campos selecionados; 4º Parâmetro: Condição de seleção; Ex.: report1 client id, name where id=2
comandoJava	Escreva qualquer código Java.

7.3 Calculadora

Para vermos como o conceito de evento e função funciona no TetrisIDE, vamos construir uma calculadora simples, que efetua operações de soma, subtração, multiplicação e divisão entre dois números.

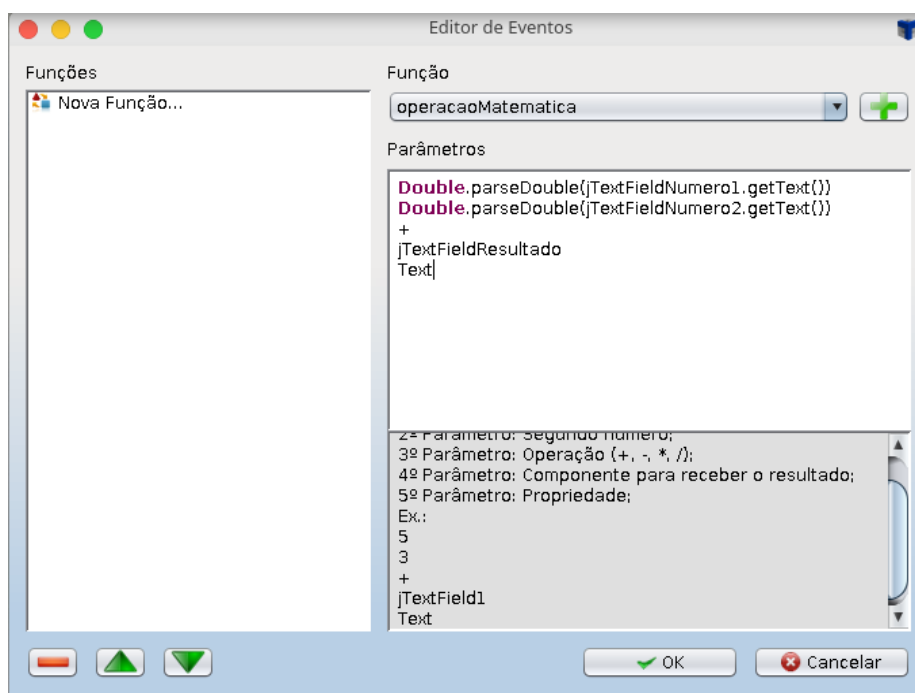
Primeiramente, crie o projeto com o nome **Calculadora**. Adicione três **Labels**, três **TextFields** e quatro **Buttons**, de acordo com a figura abaixo.



Mude o **Title** da **JFrameMain** para **Calculadora**, a propriedade **Text** dos **Labels** para **1º Número**, **2º Número** e **Resultado**. Apague o conteúdo do **TextFields** e nomeie cada um: **jTextFieldNumero1**, **jTextFieldNumero2** e **jTextFieldResultado**. Modifique a propriedade **Text** de cada botão para **+**, **-**, ***** e **/**. Em todos os **TextFields**, mude a propriedade **Mask** para **Decimal**.

No botão '+', efetue um duplo clique no evento **OnClick** no Inspetor de Objetos e adicione uma função **operacaoMatematica** com os seguintes parâmetros:

```
Double.parseDouble(jTextFieldNumero1.getText())
Double.parseDouble(jTextFieldNumero2.getText())
+
jTextFieldResultado
Text
```



Utilizamos aqui a classe **Double** do Java, para convertermos o conteúdo dos **TextFields** para número decimal, para podermos realizar a operação matemática. A dica abaixo do campo **Parâmetros** é autoexplicativa.

No botão '-', efetue um duplo clique no evento **OnClick** no Inspetor de Objetos e adicione uma função **operacaoMatematica** com os seguintes parâmetros:

```
Double.parseDouble(jTextFieldNumero1.getText())  
Double.parseDouble(jTextFieldNumero2.getText())  
-  
jTextFieldResultado  
Text
```

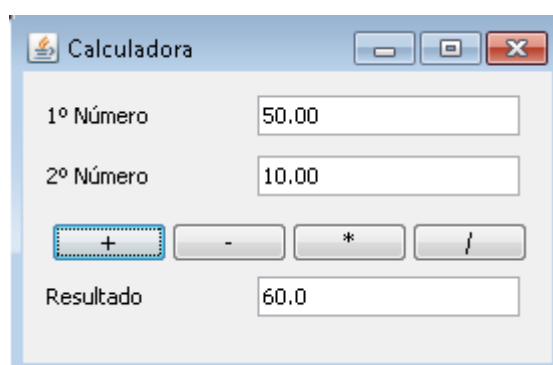
No botão '*', efetue um duplo clique no evento **OnClick** no Inspetor de Objetos e adicione uma função **operacaoMatematica** com os seguintes parâmetros:

```
Double.parseDouble(jTextFieldNumero1.getText())  
Double.parseDouble(jTextFieldNumero2.getText())  
*  
jTextFieldResultado  
Text
```

No botão '/', efetue um duplo clique no evento **OnClick** no Inspetor de Objetos e adicione uma função **operacaoMatematica** com os seguintes parâmetros:

```
Double.parseDouble(jTextFieldNumero1.getText())  
Double.parseDouble(jTextFieldNumero2.getText())  
/  
jTextFieldResultado  
Text
```

Salve e Execute o projeto, pressionando os respectivos botões na Barra de ferramentas ou pressionando **CTRL + S** e **F9**.



Desafio

Na calculadora construída neste capítulo, tente:

- Procurar nas funções do TetrisIDE uma forma de verificar se o 1º Número e o 2º Número estão preenchidos antes de cada operação;
- Travar a edição do campo do Resultado;
- Através do Inspetor de Objetos, busque como alinhar o conteúdo dos TextFields à direita;

8 VARIÁVEIS

Uma variável é um elemento na memória do computador que guarda determinado tipo de dado, podendo ser modificado ao decorrer da aplicação. Segue abaixo um simples exemplo.

```
Variável nome = "David";  
mostraTexto("Meu nome é "+nome); // O resultado é: Meu nome é David  
nome="Junior";  
mostraTexto("Agora, meu nome é "+nome); // O resultado é: Agora, meu nome é Junior
```

No exemplo conceitual acima, o algoritmo cria uma variável chamada **nome** na memória do computador e define **"David"** como o seu valor. Na linha seguinte, mostra-se o texto na tela **"Meu nome é " + o valor da variável nome**. Seguindo, o valor da variável **nome** é trocado para **"Junior"**. Posteriormente, mostra-se na tela o texto **"Agora, meu nome é " + o novo valor da variável nome**.

No TetrísIDE, podemos criar uma variável para a janela através do objeto **Variable**, presente na Paleta de Objetos. Automaticamente, quando adicionamos uma variável a uma janela, logo o TetrísIDE cria os métodos **getters** e **setters**, tornando a variável acessível a partir de outras janelas e atendendo aos conceitos de encapsulamento. Veja, de forma prática, sobre o conceito de encapsulamento em <http://blog.caelum.com.br/revisitando-a-orientacao-a-objetos-encapsulamento-no-java/>.

8.1 Brincando com variáveis

Para entendermos melhor como a ferramenta nos permite trabalhar com variáveis, criaremos um simples programa que, ao clicar em um botão, incrementa 1 unidade a uma variável do tipo número inteiro. Veja mais sobre tipos de dados em <http://www.devmedia.com.br/tipos-de-dados-por-valor-e-por-referencia-em-java/25293>.

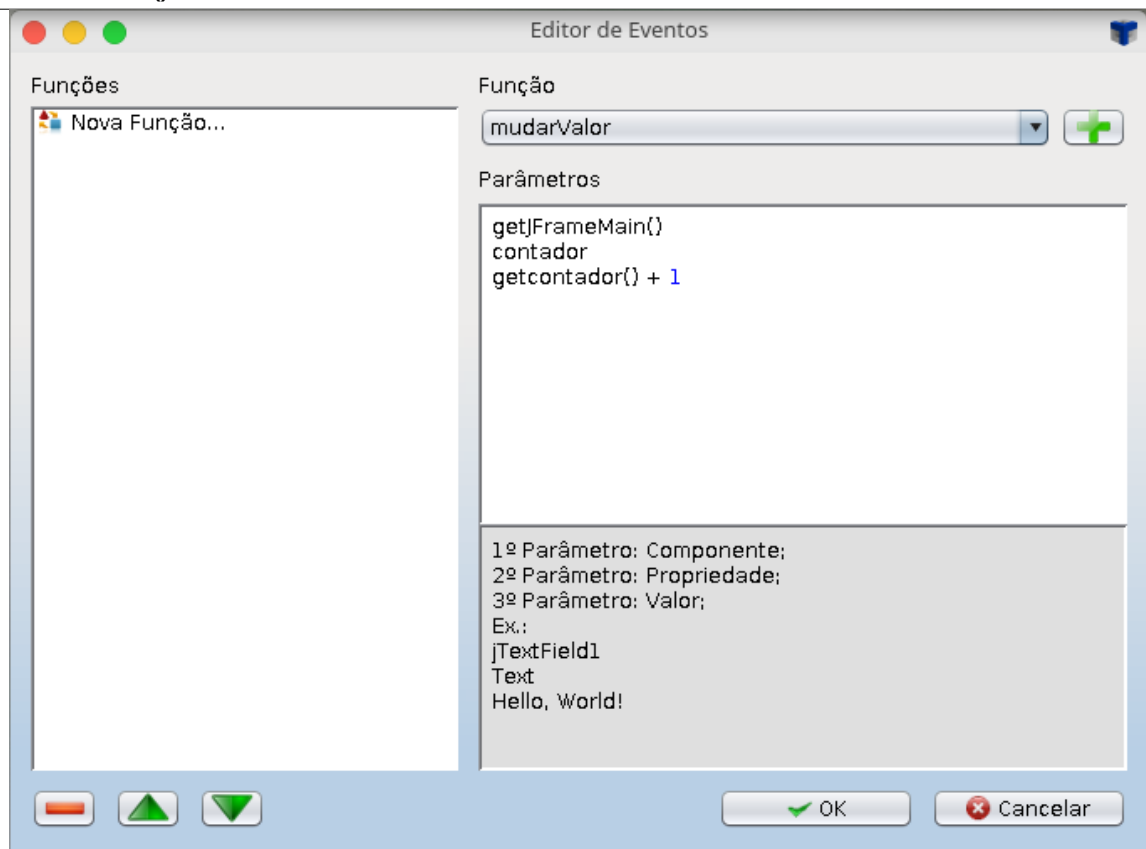
Crie um projeto chamado **Contador**. Mude o **Title** da **JFrameMain**

para **Contador**. Adicione um **Label**, um **Button** e uma **Variable**. Mude os nomes para **jLabelContador**, **jButtonContador** e **contador** respectivamente para o **Label**, o **Button** e a **Variable**. Modifique a propriedade **Text** do **jLabelContador** para **Contador: 0** e do **jButtonContador** para **Adicionar**. Modifique a propriedade **Type** da **Variable contador** para **int** e a propriedade **Value** para **0**.



Efetue duplo clique no evento **OnClick** do **jButtonContador** e adicione uma função **mudarValor** com os seguintes parâmetros:

```
getJFrameMain()  
contador  
getcontador() + 1
```



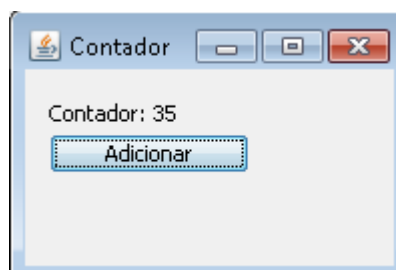
Uma **Variable** no TetrisIDE é um atributo da janela, então, quando queremos modificar o valor de uma variável através da função **mudarValor**, chamamos, no primeiro parâmetro, um método get que retorna a própria janela. A chamada do método é composta por **get + nome_da_janela + ()**. No nosso projeto, o nome da janela é **JFrameMain**, logo, o método que retorna a janela será **getJFrameMain()**. O segundo parâmetro é a propriedade/variável que desejamos modificar. No nosso caso, **contador**. O terceiro e último é o valor que a propriedade/variável receberá. No nosso caso, através do método **get** da variável **contador** (**getcontador()**), capturamos o valor de **contador** e somamos mais **1**.

Agora, no mesmo evento do botão, adicione um outro método **mudarValor** com os parâmetros:

```
jLabelContador  
Text  
Contador: "+getcontador()+"
```

Agora não estamos modificando diretamente uma variável da janela, e sim uma propriedade da **jLabelContador**. Logo, o primeiro parâmetro é o possessor da propriedade (**jLabelContador**), o segundo é a propriedade que será modificada (**Text**) e o terceiro, o novo valor. Toda propriedade do tipo **String** (**Text**, **Title**, **etc**) dispensa aspas para valores definidos no método **mudarValor**. Porém, quando queremos concatenar algum valor à string, devemos colocar entre **" + e + "**, como fizemos com o valor acima (**Contador: "+getcontador()+"**).

Salve e Execute o projeto, pressionando os respectivos botões na Barra de ferramentas ou pressionando **CTRL + S** e **F9**.



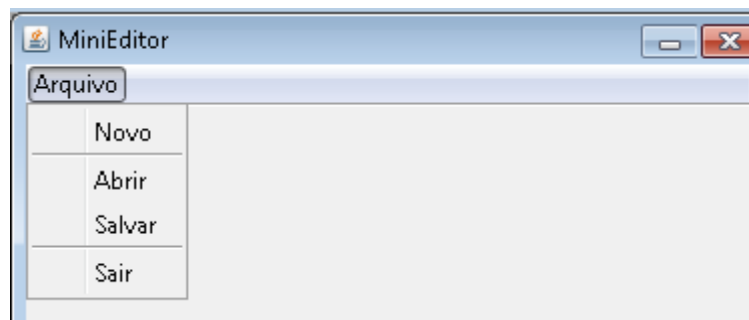
9 PROCEDURES

Procedures, no TetrisIDE, são métodos do Java adicionados à janela. Métodos são trechos de código que são escritos uma vez e podem ser chamados diversas vezes ao decorrer do programa. Imagine que pra implementar uma determinada tarefa o programador tenha que escrever 1000 linhas de código. Imagine ainda que essa tarefa será chamada 100 vezes dentro de todo o programa. Seria muito dispendioso escrever 100.000 linhas de código só por causa de uma tarefa. Logo, aquelas 1000 linhas referentes a função são definidas dentro de um método que, por sua vez, é chamado 100 vezes em toda a aplicação. Assim funciona um **Procedure** no TetrisIDE também. Veja mais sobre métodos em https://pt.wikipedia.org/wiki/M%C3%A9todo_%28programa%C3%A7%C3%A3o%29.

9.1 Usando Procedures

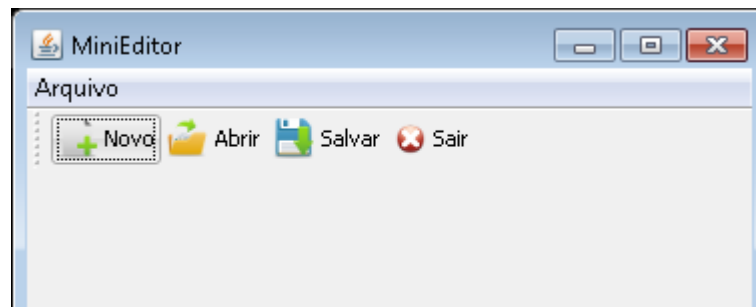
Neste exemplo, construiremos um pequeno editor de textos, misturando conceitos do TetrisIDE e do Java. Veremos a seguir como importar classes que estão em outros pacotes em nossas janelas, bem como trabalhar com código Java em nossas aplicações. Para começarmos, crie um projeto chamado **MiniEditor**.

Modifique o **Title** da **JFrameMain** para **MiniEditor** e adicione uma **MenuBar**, uma **ToolBar** e um **TextArea**. Na propriedade **Items** da **MenuBar**, adicione um **Menu** com o Text **Arquivo**. Efetue duplo clique no menu Arquivo e adicione quatro **Menu Items**: **Novo**, **Abrir**, **Salvar** e **Sair**.

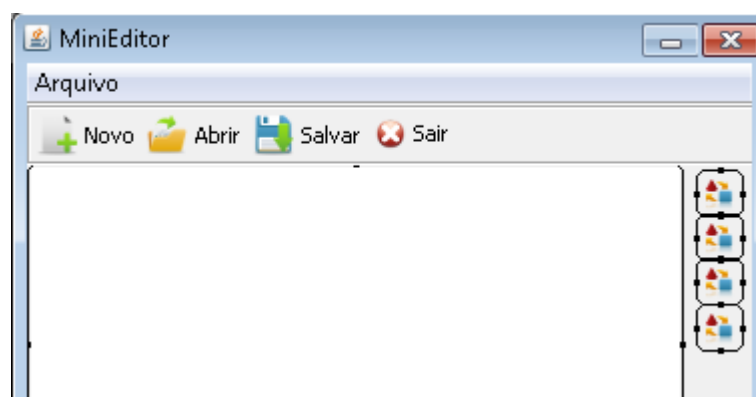


Na propriedade **Shortcut** dos MenuItem's **Novo**, **Abrir**, **Salvar** e **Sair** insira **CTRL+N**, **CTRL+A**, **CTRL+S** e **ALT+F4**, respectivamente. Caso utilize **JDK versão >= 9**, utilize **CTRL_DOWN+N**, **CTRL_DOWN+A**, **CTRL_DOWN+S** e **ALT_DOWN+F4**. Estes são os atalhos de teclado usados para cada menu item.

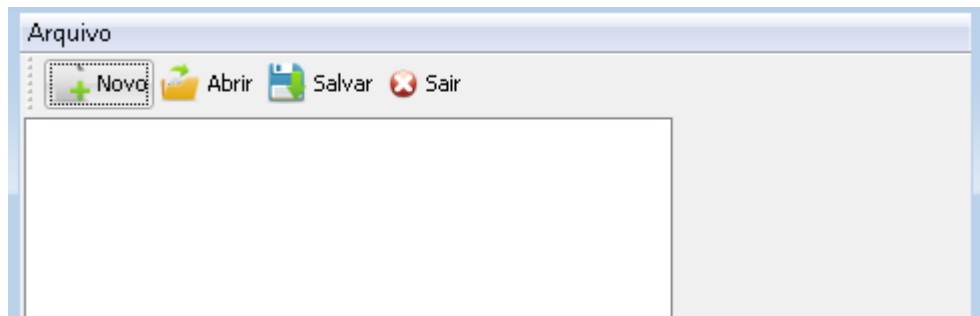
Na **ToolBar**, adicione quatro botões com os **Text**: **Novo**, **Abrir**, **Salvar** e **Sair**. Selecione a imagem que desejar na propriedade **Icon** de cada botão.



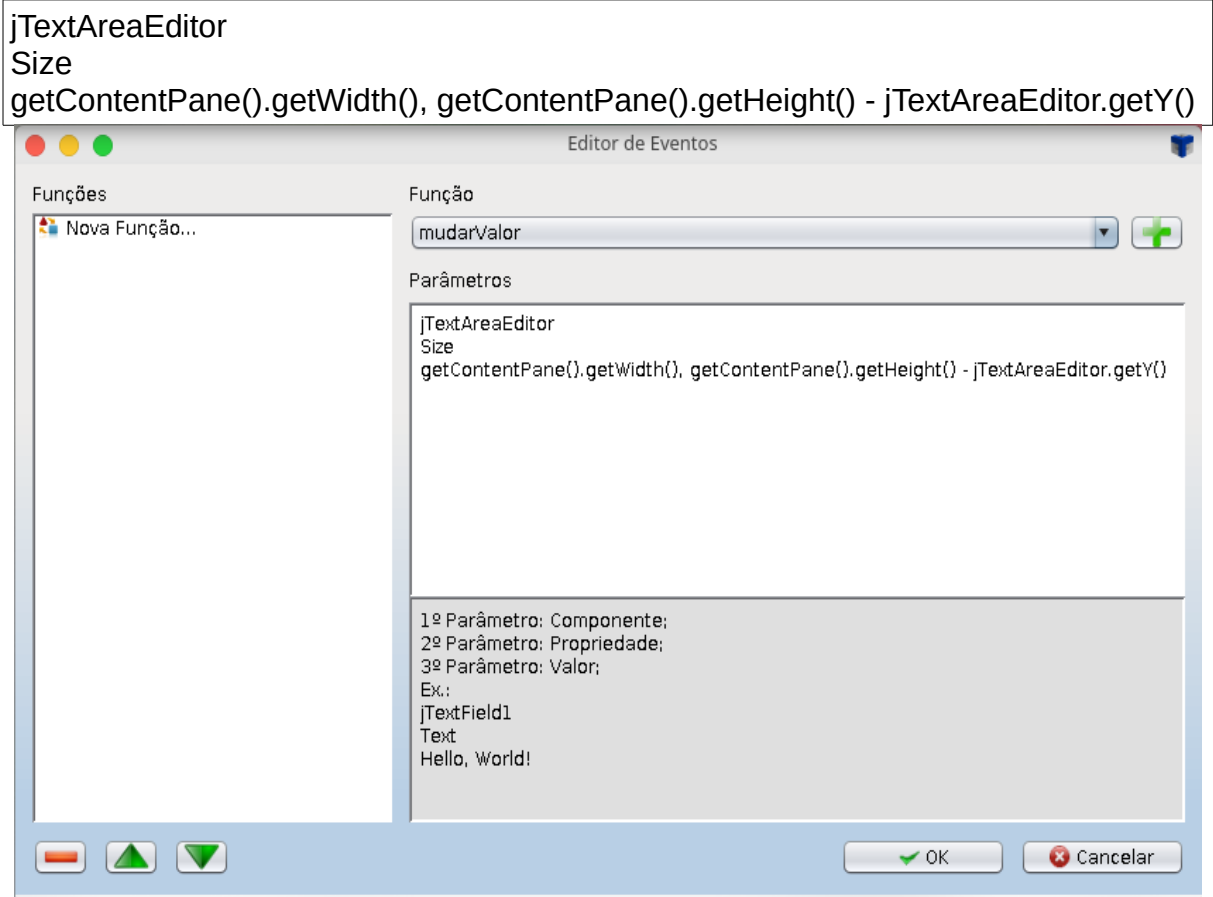
Renomeie o **TextArea** para **jTextAreaEditor** e adicione quatro **Procedures**, pela Paleta de Objetos, chamados: **novo**, **abrir**, **salvar** e **sair**. No Procedure **salvar**, altere as propriedades **Parameters**, **Type** e **Return** para **String content**, **boolean** e **false**, respectivamente. Acabamos de configurar nossa Procedure (como um método em Java) que recebe um parâmetro, do tipo **String**, chamado **content** e retorna um valor **boolean** (**true** ou **false**), sendo o valor padrão do retorno **false**.



Mude a propriedade **ExtendedState** da **JFrameMain** para **Maximizado**, para que a janela inicialize já maximizada. Salve e execute o projeto para visualizar a alteração.



Perceba que temos um problema: nosso `TextArea` não acompanha o tamanho da janela. Para resolver este inconveniente, vamos adicionar uma função **mudarValor** no evento **OnResize** da **JFrameMain** com os seguintes parâmetros:



Nesta função, a propriedade **Size** (que, na verdade, no Java é um método que preenche as propriedades **width** e **height**) recebe dois parâmetros: a largura do container da janela (**getContentPane().getWidth()**) e a altura menos a posição vertical do **TextArea** (**getContentPane().getHeight()**) -

jTextAreaEditor.getY()). Neste ponto do projeto já estamos misturando funções do TetrisIDE com código Java.

Para conseguirmos executar as próximas etapas, necessitaremos de classes Java que estão no pacote **java.io**, que contém métodos que executam as operações de IO (entrada e saída), inclusive de gravação e leitura de arquivos. Estas classes precisam ser importadas para a janela que irá utilizá-las. Para isso, efetue um duplo clique na propriedade **Import** da **JFrameMain** e insira o seguinte:

```
java.io.File  
java.io.FileReader  
java.io.BufferedReader  
java.io.FileWriter  
java.io.BufferedWriter  
javax.swing.JFileChooser
```

A primeira classe (**java.io.File**) importamos para trabalharmos com arquivos. A segunda e terceira (**java.io.FileReader** e **java.io.BufferedReader**) para lermos arquivos. A quarta e quinta (**java.io.FileWriter** e **java.io.BufferedWriter**) para gravar arquivos. A última classe (**javax.swing.JFileChooser**), utilizamos para abrir uma janela de diálogo para escolhermos o local de abertura e gravação do arquivo.

Clique no Procedure **novo** e adicione uma função **mudarValor** no evento **OnExecute** com os seguintes parâmetros:

```
jTextAreaEditor  
Text
```

Aqui, quando o Procedure **novo** for executado, dizemos ao programa para limpar o conteúdo da propriedade **Text** do **jTextAreaEditor**.

No Procedure **abrir**, adicione uma função **comandoJava** no evento **OnExecute** com os seguintes parâmetros:

```
try {  
    JFileChooser jFileChooser = new JFileChooser();  
  
    if(jFileChooser.showOpenDialog(getJFrameMain())==JFileChooser.APPROVE_O  
PTION){  
        String caminhoArquivo = jFileChooser.getSelectedFile().getAbsolutePath();
```

```

FileReader fileReader = new FileReader(caminhoArquivo);
BufferedReader bufferedReader= new BufferedReader(fileReader);

jTextAreaEditor.setText("");
String linha="";

while ((linha = bufferedReader.readLine()) != null) {
    jTextAreaEditor.setText(jTextAreaEditor.getText()+linha+"\n");
}

bufferedReader.close();
fileReader.close();
}
} catch (Exception exc) {
    exc.printStackTrace();
}
}

```

Para ver mais sobre leitura e gravação de arquivos no Java, visite <http://www.devmedia.com.br/lendo-dados-de-txt-com-java/23221>.

Agora, no Procedure **salvar**, adicione uma função **comandoJava** no evento **OnExecute** com os seguintes parâmetros:

```

try {
    JFileChooser jFileChooser = new JFileChooser();

    if(jFileChooser.showSaveDialog(getJFrameMain())==JFileChooser.APPROVE_OPTION){
        String caminhoArquivo = jFileChooser.getSelectedFile().getAbsolutePath();
        File file = new File(caminhoArquivo);
        BufferedWriter bufferedWriter = new BufferedWriter(new FileWriter(file));

        bufferedWriter.write(content); /*Escrevemos no arquivo em disco o conteúdo
do parâmetro content, do tipo String, configurado na propriedade Parameters do
Procedure salvar*/
        bufferedWriter.flush();

        bufferedWriter.close();

        retorno = true; /*Setamos o retorno do Procedure salvar true, já que
configuramos a propriedade Type como boolean*/
    }
} catch (Exception exc) {
    exc.printStackTrace();
}
}

```

No Procedure **sair**, adicione uma função **fecharJanela** sem parâmetros.

Para finalizarmos nosso programa, basta chamar as funções nos menus e nos botões. No menu **Novo** e no botão **Novo**, adicione uma função **executarProcedure** no evento **OnClick** com o parâmetro:

novo

Aqui estamos chamando o Procedure **novo** com a função **executarProcedure**.

No menu **Abrir** e no botão **Abrir**, adicione uma função **executarProcedure** no evento **OnClick** com o parâmetro:

abrir

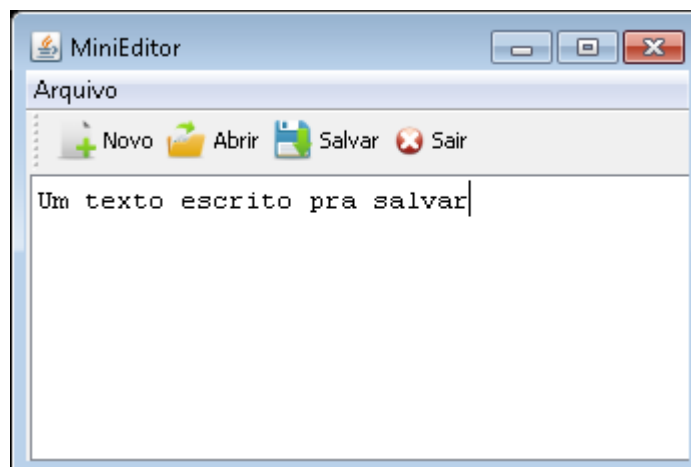
No menu **Salvar** e no botão **Salvar**, adicione uma função **executarProcedure** no evento **OnClick** com o parâmetro:

salvar
jTextAreaEditor.getText() /*Passamos o texto contido no jTextAreaEditor como valor do parâmetro content, definido na propriedade Parameters do Procedure salvar*/

No menu **Sair** e no botão **Sair**, adicione uma função **executarProcedure** no evento **OnClick** com o parâmetro:

sair

Salve e Execute o projeto, pressionando os respectivos botões na Barra de ferramentas ou pressionando **CTRL + S** e **F9**.



Neste capítulo, vimos que, apesar de o TetrisIDE ser construído para o desenvolvimento de aplicações Java desktop sem codificação manual, a funcionalidade de inserção de código é inteiramente suportado.

10 COMANDOS JAVA

O TetrisIDE foi desenvolvido para que programadores conseguissem desenvolver aplicações sem codificar uma linha se quer manualmente. Mas, isso não quer dizer que você não possa digitar seus códigos dentro da ferramenta.

Na sequência, veremos um pouco do Java por trás dessa poderosa IDE e como utilizá-lo.

10.1 O Java por trás do TetrisIDE

Tudo no TetrisIDE parte do Java (afinal, ele é feito em Java e para Java). Isso implica que tudo que ele fizer por você vem de alguma classe Java. Abaixo temos uma janela com um botão. Repare em seu código-fonte gerado pela ferramenta.



Código-fonte:

```
package tetris.javacommands.visao;  
  
import javax.swing.JFrame;  
import componentes.visao.*;  
  
public class JFrameMain extends JFrame{  
    private JTetrisButton jButton1;
```

```

private String retorno;
public JFrameMain getJFrameMain(){
    return this;
}

public JFrameMain(){
    setDefaultCloseOperation(JFrame.DO_NOTHING_ON_CLOSE);

    addWindowListener( new java.awt.event.WindowAdapter(){
        public void windowClosing(java.awt.event.WindowEvent arg0){
            fecharJanela();
        }
    });
    setContentPane(new JTetrisPanel(null));
    setTitle("Java Commands");
    setResizable(true);
    setBounds(0, 0, 200, 200);

    jButton1 = new JTetrisButton("");
    jButton1.setText("Java");
    jButton1.setBounds(40, 62, 100, 25);

    getContentPane().add(jButton1);
}

public void init(){
    setVisible(true);
}

public String init(String retorno){
    setRetorno(retorno);
    this.init();
    return getReturn();
}

public void fecharJanela(){
    System.exit(0);
}

public String getReturn(){
    return retorno;
}

public void setReturn(String retorno){
    this.retorno=retorno;
}
}

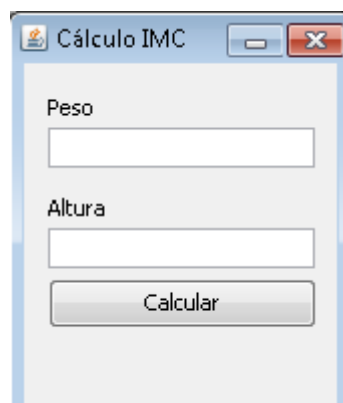
```

Perceba que toda a janela é uma classe Java herdada de um JFrame, JDialog ou JInternalFrame, como já falamos.

10.2 Cálculo de IMC

Desenvolveremos um programa que calcula o índice de massa corporal utilizando códigos Java no TetrísIDE. Para tanto, crie um projeto chamado **CalculoIMC**. Mude seu **Title** para **Cálculo IMC** e adicione dois **Labels**, dois **TextFields** e um **Button**.

Altere o **Text** dos **Labels** para **Peso** e **Altura** e apague o dos **TextFields**. Modifique o nome dos **TextFields** para **jTextFieldPeso** e **jTextFieldAltura**. Troque a propriedade **Text** do **Button** para **Calcular**. Na propriedade **Mask** de cada **TextField**, selecione **Decimal**.



Efetue um duplo clique no evento **OnClick**, no Inspetor de Objetos, do **Button** e adicione uma função **comandoJava** com os seguintes parâmetros:

```
double peso = Double.parseDouble(jTextFieldPeso.getText());  
double altura = Double.parseDouble(jTextFieldAltura.getText());  
  
double imc = peso / (altura * altura);  
  
JOptionPane.showMessageDialog(getJFrameMain(), "O IMC é: "+imc);
```

Este é um código simples, onde capturamos os valores digitados e os transformamos em números decimais. Posteriormente, efetuamos o cálculo do

IMC e exibimos uma mensagem através da classe **JOptionPane**.

Para finalizar, na propriedade **Import** do **JFrameMain**, insira **javax.swing.JOptionPane**.

Salve e Execute o projeto.

Você pode exportar código Java, para abrir em outra IDE, através do botão **Exportar**, localizado na Barra de ferramentas.

11 OPERAÇÕES COM MYSQL

O Sistema Gerenciador de Banco de Dados MySQL é um dos mais utilizados no mundo, estando presente numa grande parcela das aplicações da web. Mantido e fornecido pela Oracle, o MySQL possui diversas versões, inclusive uma vertente mantida pela comunidade, sendo gratuito e bem respaldado: o MySQL Community.

MySQL é o SGDB padrão do TetrísIDE, trazendo uma simples e rápida integração para sua aplicação. O desenvolvimento Java se tornou fácil com essa incrível solução RAD.



Como a grande maioria dos bancos de dados relacionais da atualidade, o MySQL, como seu próprio nome sugere, utiliza os conceitos e sintaxe da linguagem SQL (Structured Query Language). Caso não esteja familiarizado com a linguagem, dê uma olhada em <https://pt.wikipedia.org/wiki/SQL>.

11.1 SQL no TetrísIDE

Normalmente, a operação **select** no SQL é formada pela cláusula **select**, *colunas_para_selecionar*, cláusula **from**, *tabela_para_selecionar*, cláusula **where**, *condição*, cláusula **order by** e *colunas_para_ordenar*. Exemplo:

```
select id, nome from cliente where nome='David' order by id;
```

O TetrísIDE nos traz funções que recebem parâmetros para execuções de operações em banco de dados. Você não tem que efetuar conexão, ou carregar um resultset e executar um statement. A ferramenta faz isso por você. Abaixo segue as funções que realizam operações em banco de dados:

Função	Parâmetros/Dica
selecionarRegistro	1º Parâmetro: Tabela do banco de dados; 2º Parâmetro: Campos selecionados; 3º Parâmetro: Condição de seleção; Ex.: client id, name where id='2'
gravarRegistro	1º Parâmetro: Tabela do banco de dados; 2º Parâmetro: Campos selecionados; 3º Parâmetro: Valores; Ex.: client id, name '2', 'David'
alterarRegistro	1º Parâmetro: Tabela do banco de dados; 2º Parâmetro: Campos selecionados; 3º Parâmetro: Condição de seleção; Ex.: client id='3', name='David' where id='2'
preencherTabela	1º Parâmetro: jTable; 2º Parâmetro: Tabela do banco de dados; 3º Parâmetro: Campos selecionados; 4º Parâmetro: Condição de seleção; Ex.: jTable1 client id, name where id='2'
excluirRegistro	1º Parâmetro: Tabela do banco de dados; 2º Parâmetro: Condição de seleção; Ex.: client

	where id='2'
verificarRegistro	1º Parâmetro: Tabela do banco de dados; 2º Parâmetro: Campos selecionados; 3º Parâmetro: Condição de seleção; 4º Parâmetro: (Opcional) Mensagem; 5º Parâmetro - Padrão (==): (Opcional) Comparador (!=, ==); Ex.: client id where id='2' Não há registro! ==
visualizarRelatorio	1º Parâmetro: Relatório; 2º Parâmetro: Tabela do banco de dados; 3º Parâmetro: Campos selecionados; 4º Parâmetro: Condição de seleção; Ex.: report1 client id, name where id=2
imprimirRelatorio	1º Parâmetro: Relatório; 2º Parâmetro: Tabela do banco de dados; 3º Parâmetro: Campos selecionados; 4º Parâmetro: Condição de seleção; Ex.: report1 client id, name where id=2

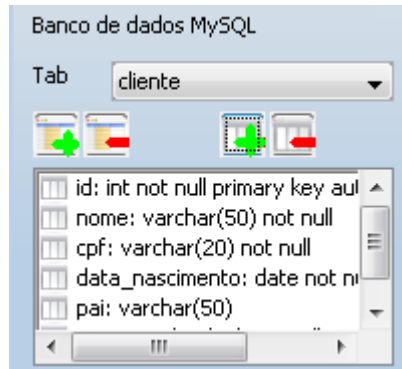
As duas últimas funções (visualizarRelatorio e imprimirRelatorio) veremos posteriormente, no capítulo **13 Relatórios com o TetrísReport**.

Crie um projeto chamado **OperacoesMySQL** e mude o **Title** da **JFrameMain** para **Operações MySQL**.


11.2 Criando o Banco de Dados

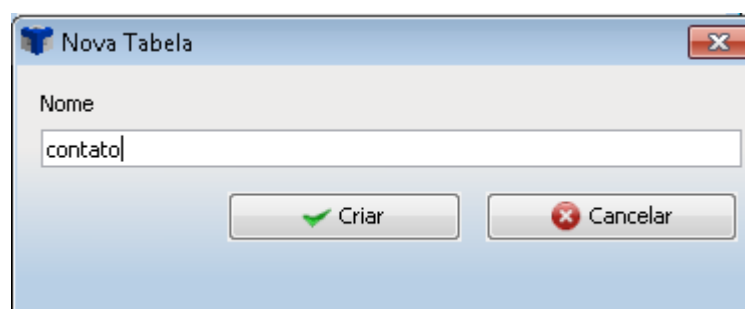
A IDE possui uma ferramenta para a composição do banco de dados da aplicação com extrema simplicidade, tornando fácil e rápido o processo. O Gerenciador de Banco de Dados MySQL pode ser encontrado na barra lateral


esquerda, abaixo do Explorador de Janelas. Nele, há uma caixa de combinação que lista todas as tabelas do banco de dados do projeto e, abaixo, suas colunas. Há também botões de operações, para inserir e remover tabelas e colunas.



Programas que utilizam o banco de dados MySQL feitos com o TetrísIDE criam o banco de dados automaticamente, caso ele não o encontre no servidor de banco de dados, facilitando bastante o trabalho de implantação do software no cliente. Toda a alteração feita no Gerenciador de Banco de Dados do TetrísIDE só será aplicado ao banco de dados quando o projeto for executado.

Em nossa aplicação, criaremos somente uma tabela para estudo. Clique no botão Adicionar tabela () e adicione uma tabela chamada **contato**.



Perceba que, ao adicionar uma tabela, um campo **id** é criado como chave primária. Vamos mantê-lo. Criaremos agora mais dois campos, clicando no botão Adicionar coluna ().

Crie a coluna **nome** do tipo **varchar(50)** e desmarque a opção **nulo**.



Nova Coluna

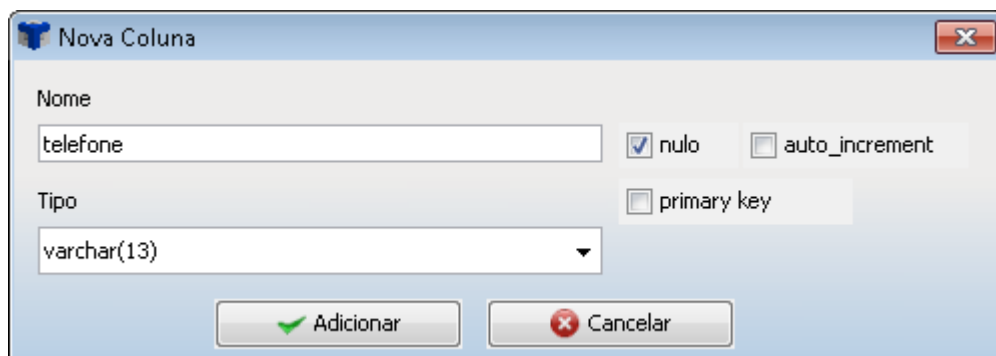
Nome
nome

Tipo
varchar(50)

☐ nulo ☐ auto_increment
☐ primary key

Adicionar Cancelar

Crie a coluna **telefone** do tipo **varchar(13)**.



Nova Coluna

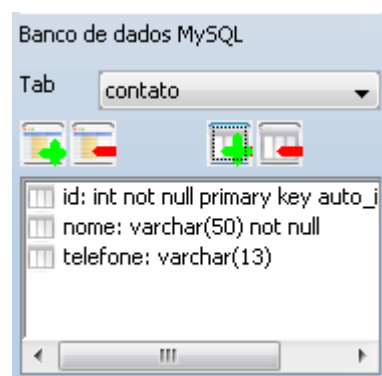
Nome
telefone

Tipo
varchar(13)

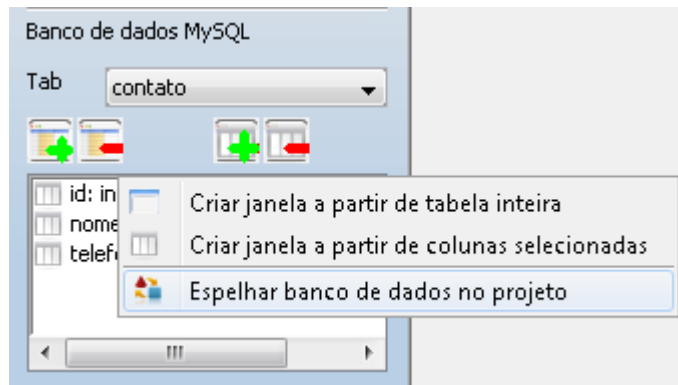
☒ nulo ☐ auto_increment
☐ primary key

Adicionar Cancelar

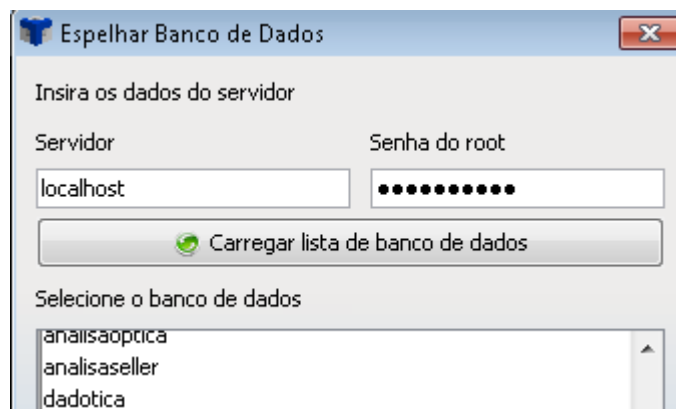
Seu Gerenciador de Banco de Dados deve estar assim:



Caso você prefira montar fora do TetrISIDE, é possível espelhar a estrutura de um banco de dados do servidor clicando com botão direito na lista de colunas e escolhendo a opção **Espelhar banco de dados no projeto**.

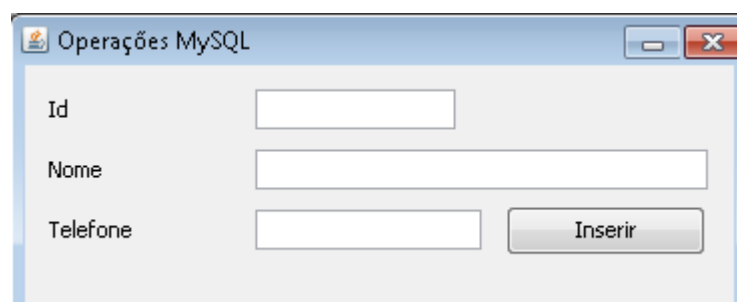


Na janela seguinte, basta inserir os dados de conexão do servidor, conectar e selecionar o banco de dados desejado.



11.3 Inserindo registros

Adicione três **Labels**, três **TextFields** e um **Button** e nomeie os TextFields: **jTextFieldId**, **jTextFieldNome** e **jTextFieldTelefone**. Apague o conteúdo da propriedade **Text** dos **TextFields**, e modifique a dos **Labels** para **Id**, **Nome** e **Telefone** e a do **Button** para **Inserir**.



No evento **OnClick** do **Button**, adicione uma função **verificarRegistro** com os parâmetros:

```
contato
id
where id="'+jTextFieldId.getText()+'"
Este Id já está sendo utilizado!
!=
```

Esta função serve para verificar a existência de registros em determinada tabela com determinada condição. No nosso caso, estamos verificando se existe algum registro com o Id inserido, pois a coluna **id** da tabela **contato** é um campo único. O primeiro parâmetro é a tabela a ser verificada. O segundo, são campos de seleção, normalmente, a chave primária é utilizada. O terceiro, é a condição da consulta que, no nosso caso, estamos verificando registros que tem o id igual ao conteúdo da propriedade **Text** do **jTextFieldId** (**where id="'+jTextFieldId.getText()+'**'). O quarto é a mensagem que será apresentada caso satisfaça a condição imposta no quinto parâmetro, este, sendo possível receber um dos dois valores: **==** ou **!=**. Quando o quinto parâmetro é preenchido com **==**, quando o número de registros retornados pela consulta for igual a zero, exibe-se a mensagem do quarto parâmetro e para a execução do evento. Caso o quinto parâmetro seja **!=**, quando o número de registros retornados pela consulta for diferente de zero, exibe-se a mensagem do quarto parâmetro e para a execução do evento, da mesma forma.

Adicione agora uma função **gravarRegistro** com os parâmetros:

```
contato
id, nome, telefone
'+jTextFieldId.getText()+'', '"+jTextFieldNome.getText()+'', '"+jTextFieldTelefone.getText()+''
```

Esta função insere um registro no banco de dados, tendo somente três parâmetros: tabela do banco de dados, colunas e os valores das colunas, separados por vírgulas. Caso deseje concatenar valores aos parâmetros, coloca-se entre **“+ e +”**, pois todos eles são do tipo string (texto).

Para finalizar a inserção, adicione três funções **mudarValor** com os parâmetros abaixo:

jTextFieldId Text
jTextFieldNome Text
jTextFieldTelefone Text

Salve e Execute o projeto. Você vai ter uma janela de configuração de conexão com o banco de dados ao executar o programa pela primeira vez. Preencha os dados de acordo com as informações abaixo (Respeite letras maiúsculas e minúsculas).

SERVER: localhost	Esse é o endereço do servidor.
DATABASE: OperacoesMySQL	Nome do bando de dados. Normalmente, o nome do projeto.
USER: root	Usuário do banco de dados.
PASSWORD: password do root	Troque esse campo pela senha do root da instalação do seu MySQL.

Insert the data to connect to the database:

SERVER
localhost

DATABASE USER PASSWORD
OperacoesMySQL root

OK Exit

Pressione o botão OK e inicie novamente a aplicação. Faça um teste e insira registros no banco de dados.

Operações MySQL

Id 1

Nome David de Almeida Bezerra Jr

Telefone (00)1111-2222 Inserir

11.4 Visualizando e selecionando registros

No TetrísIDE, podemos relacionar dados de uma consulta ao banco de dados facilmente em uma **Table**. Para tanto, inclua uma **Table** na **JFrameMain** e modifique seu nome para **jTableContato**.

Na **Table**, existem duas propriedades utilizadas pela IDE para o preenchimento de informações: **Titles** e **Columns**. A primeira é o título das colunas que aparecerão para o usuário, enquanto a segunda é o campo retornado da consulta ao banco de dados que preencherá a célula da tabela.

Na propriedade **Titles** da **jTableContato** insira:

Id Nome Telefone

Já na propriedade **Columns**, preencha:

id nome telefone

Vamos fazer agora o preenchimento da **jTableContato** através da função **preencherTabela**. Como vamos utilizá-la em mais de um evento, crie uma **Procedure** e chame-a de **atualizarTabela**. No evento **OnExecute** da **Procedure**, adicione uma função **preencherTabela** com os parâmetros:

jTableContato contato id, nome, telefone order by id

Adicione uma função **executarProcedure** ao evento **OnShow** da **JFrameMain** e ao evento **OnClick** do botão **Inserir** com o parâmetro:

atualizarTabela

Desta forma, assim que abrir a janela a **jTableContato** será preenchida. Da mesma forma acontecerá quando se inserir um registro na tabela **contato**.



Precisamos que, para fins de visualização, ao clicar em um registro da tabela, seus dados sejam visualizados nos **TextFields** acima, possibilitando uma posterior edição de seus valores. O TetrisiDE trata isso de forma bastante simples através da função **selecionarRegistro**. Para que um componente seja preenchido com um valor de um campo de uma consulta ao banco de dados, basta preencher na propriedade **Column** do Inspetor de Objetos o nome do campo que será retornado. Vamos ver na prática!

Para os **jTextFieldId**, **jTextFieldNome** e **jTextFieldTelefone**, modifique a propriedade **Column** para **id**, **nome** e **telefone**. Nos eventos **OnClick** e **OnKeyReleased** da **jTableContato** adicione uma função **selecionarRegistro** com os parâmetros:

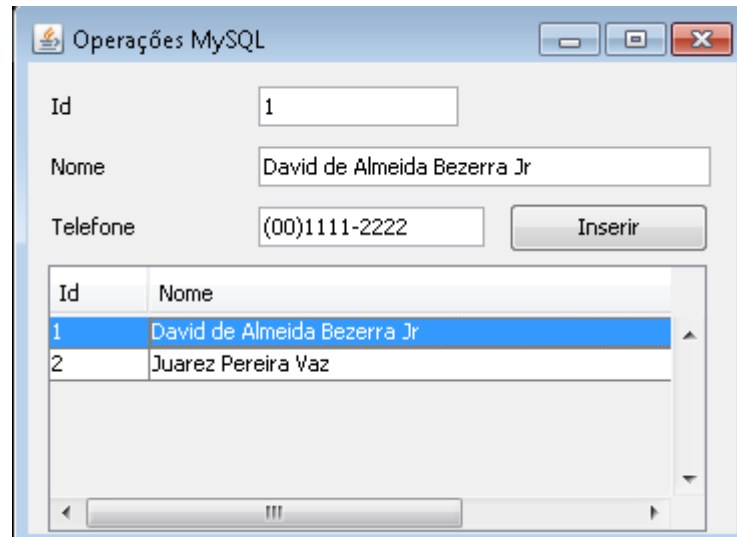
```
contato
id, nome, telefone
where id='"+jTableContato.getSelectedValue("id")+"'
```

Desta forma, toda vez que o usuário clicar na tabela ou apertar alguma tecla com o foco na tabela, o registro selecionado será visualizado nos **TextFields**.

Analisando esta função, o primeiro parâmetro é a tabela selecionada, o segundo, os campos que serão retornados na consulta, e o terceiro, a condição da consulta. Perceba que conseguimos pegar o valor selecionado da coluna **id** da **jTableContato** através do método **getSelectedValue("id")**.

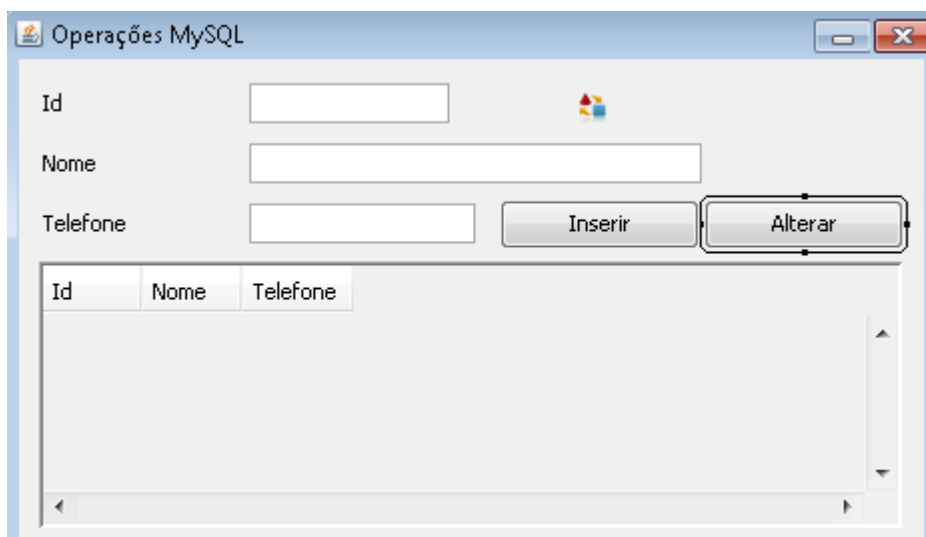
Desenvolvedores Java que já trabalharam com jTableS concordam que este método facilitou bastante o desenvolvimento.

Salve e execute o projeto. Experimente clicar em um dos registros da tabela.



11.5 Alterando registros

Da mesma forma que para inserir registros o TetrISIDE facilita a vida do desenvolvedor, alterar registros também não foge do padrão. A função utilizada para isso é a **alterarRegistro**, que segue o padrão das outras funções. Adicione um Button à janela e mude a propriedade **Text** para **Alterar**.



No evento **OnClick**, do botão **Alterar**, adicione uma função **verificarValor** com os parâmetros:

```
jTableContato  
SelectedRow  
<  
0  
Não há registro selecionado!
```

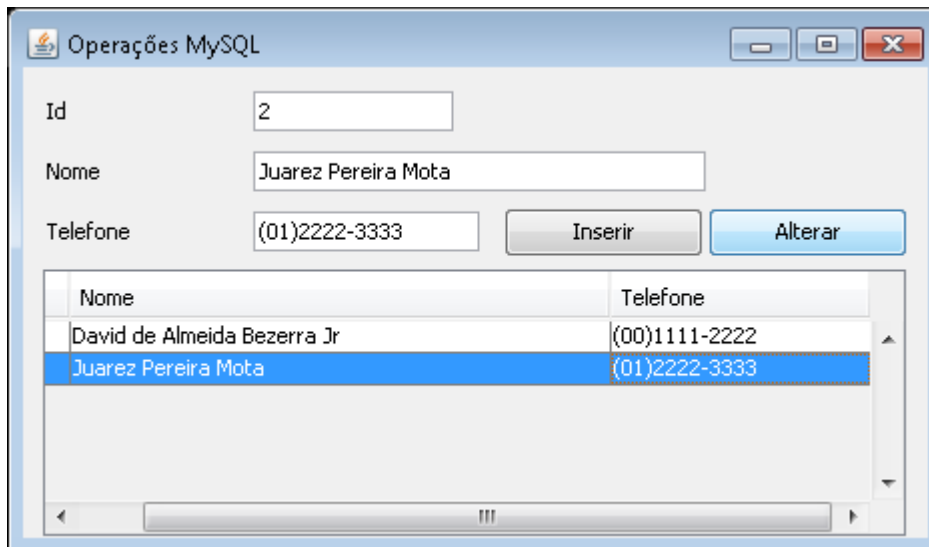
Esta função verifica se um valor atende a um requisito. Caso sim, exibe-se uma mensagem. O primeiro parâmetro é o componente que contém a propriedade (**jTableContato**). O segundo é a propriedade que será verificada (**SelectedRow**). Esta propriedade, na verdade, é um método da **jTableContato** que retorna o número da linha selecionada. Se fôssemos chamar em um código Java, seria **jTableContato.getSelectedRow()**. O terceiro parâmetro é o comparador (<, >, ==, !=, <=, >=), enquanto o quarto parâmetro é o valor a comparar. O quinto é a mensagem que será exibida caso a condição se contemple, parando a execução do evento.

Adicione agora uma função **alterarRegistro** com os parâmetros:

```
contato  
id="'+jTextFieldId.getText()+'", nome="'+jTextFieldNome.getText()+'", telefone="'+jTextFieldTelefone.getText()+'"  
where id="'+jTableContato.getSelectedValue("id")+"'
```

Como primeiro parâmetro, passamos a tabela a ter o registro alterado (contato). O segundo são as colunas juntamente com seus novos valores (id="'+jTextFieldId.getText()+'", nome="'+jTextFieldNome.getText()+'", telefone="'+jTextFieldTelefone.getText()+'"). Perceba a estrutura: **coluna='valor'**. O terceiro é a condição de alteração. Todos os registros que atenderem àquela condição (where id="'+jTableContato.getSelectedValue("id")+"'") serão alterados (nesse caso, somente um registro, já que o id é único). Para finalizar, adicione uma função **executarProcedure** chamando o Procedure **atualizarTabela**.

Salve e Execute a aplicação. Selecione um registro e modifique um de seus campos.



11.6 Excluir registros

Para efetuarmos a exclusão de registros no banco de dados, utilizamos a função **excluirRegistro**. Adicione um **Button** e modifique seu **Text** para **Excluir**.

Adicione uma função **verificarValor** ao evento **OnClick** do botão **Excluir**, para certificar-se que o usuário selecionou um registro na tabela, com os parâmetros:

```
jTableContato
SelectedRow
<
0
Não há registro selecionado!
```

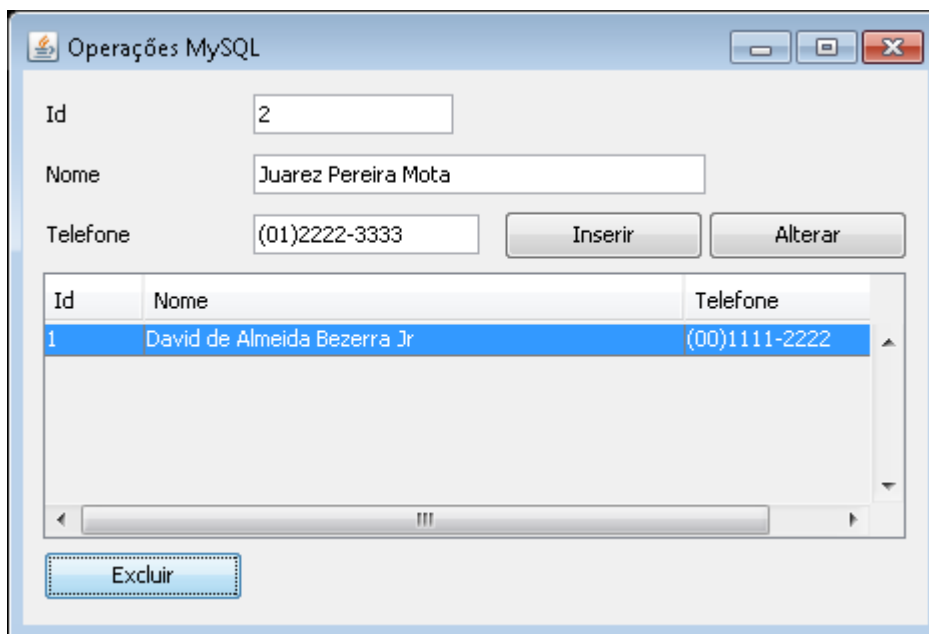
Para excluir o registro selecionado na tabela, adicione, também, uma função **excluirRegistro** com os parâmetros:

```
contato
where id='"+jTableContato.getSelectedValue("id")+"'
```

O primeiro parâmetro é a tabela que contém o registro a ser eliminado, enquanto o segundo é a condição para exclusão.

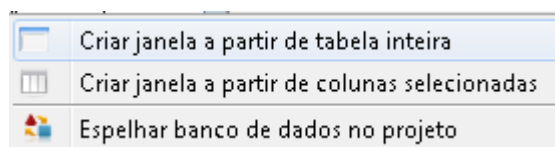
Para finalizar, adicione uma função **executarProcedure** chamando a

Procedure **atualizarTabela**, para atualizar a **jTableContato**.

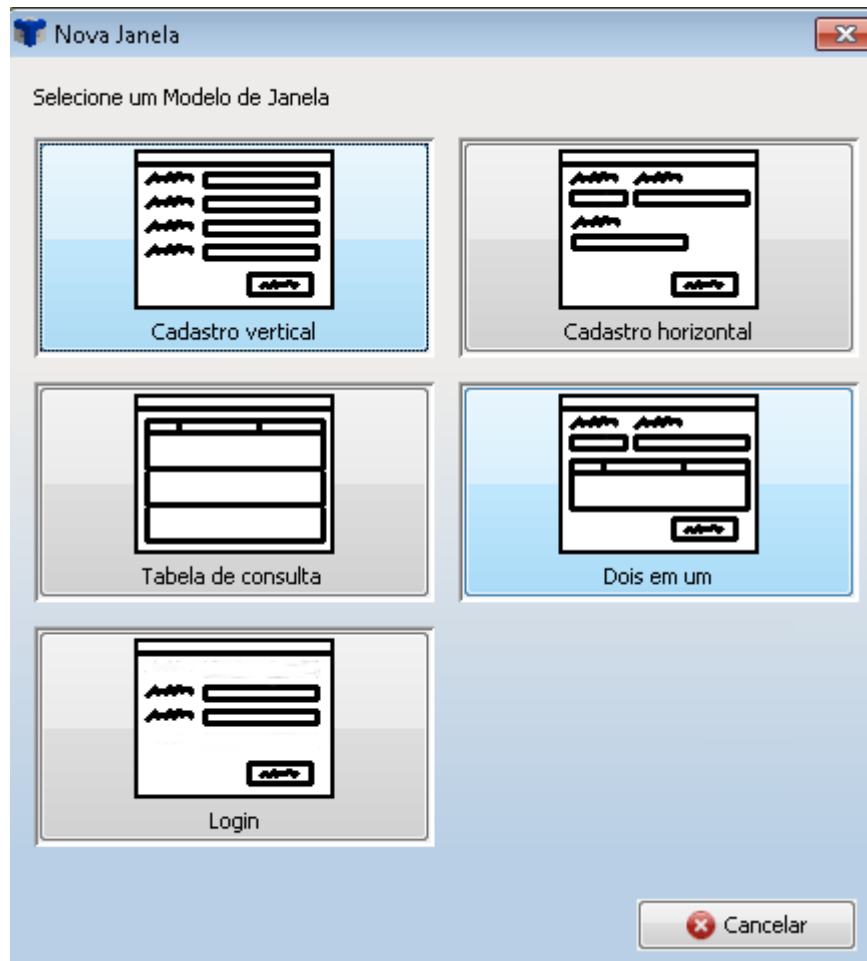


11.7 Criando janelas para acesso a dados rapidamente

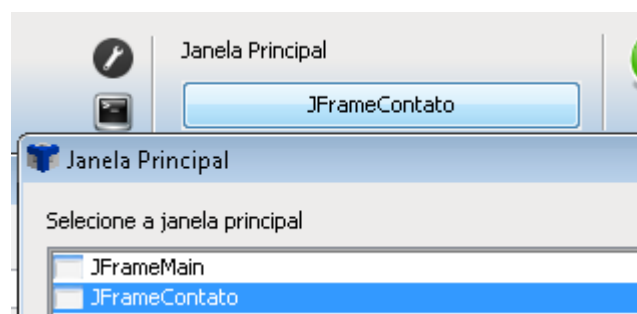
O TetrísIDE consegue ser ainda mais rápido na criação de janelas para operações com banco de dados. A partir de uma tabela, a IDE consegue montar uma janela completamente operacional para inserção, alteração, exclusão e visualização de dados. Vamos gerar uma janela a partir da tabela contato de nosso projeto. Para isso, clique com o botão direito do mouse na lista de colunas do Gerenciador de Banco de Dados e selecione a opção **Criar janela a partir de tabela inteira**.



É possível criar diversos tipos de janelas, para diversos tipos de tarefas. Em nosso exemplo, criaremos um formulário **Dois em um**, que contém operações de inclusão, alteração e exclusão em banco de dados, bem como uma tabela de consulta.



Clique no botão **Dois em um** e dê o nome **JFrameContato** e o título **Contato** para a nova janela. Para a janela ser executada primeiro quando iniciar o programa, na **Barra de ferramentas**, selecione-a como **Janela Principal**.



Salve e Execute a aplicação. Experimente todas as possibilidades que a nova janela lhe oferece.

The screenshot shows a Java Swing window titled "Contato". It contains three input fields at the top: "Id*" with the value "1", "Nome*" with the value "David de Almeida Bezerra Jr", and "Telefone" with the value "(01)1111-0000". Below these fields is a table with three columns: "id", "nome", and "telefone". The table contains one row with the values "1", "David de Almeida Bezerra Jr", and "(01)1111-0000". At the bottom of the window are five buttons: "Novo", "Alterar", "Gravar", "Cancelar", and "Excluir". The "Excluir" button is highlighted with a blue border.

id	nome	telefone
1	David de Almeida Bezerra Jr	(01)1111-0000

Neste capítulo, vimos as facilidades que o TetrísIDE nos proporciona no desenvolvimento de aplicações que efetuam operações em banco de dados. Esta tarefa, em um desenvolvimento convencional (utilizando uma IDE de codificação manual), requer muito mais esforço, principalmente, quando estamos trabalhando com a linguagem Java.

12 COMPONENTES EXTERNOS (JAR)

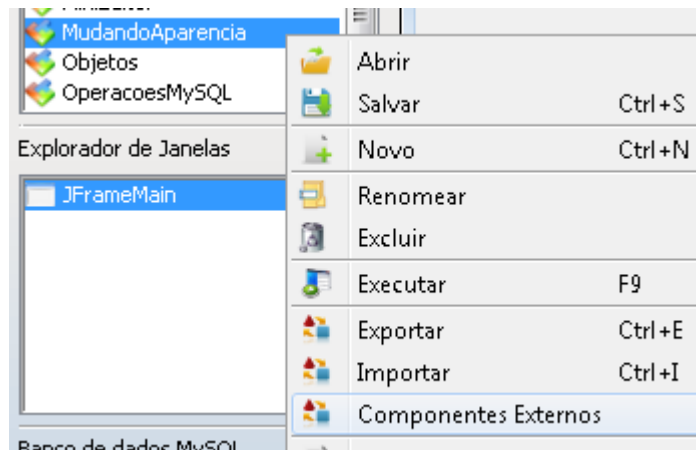
O Java presente no TetrisIDE permite que o desenvolvedor utilize de conceitos e componentes presentes no Java e fora dele. Assim, classes e componentes presentes em arquivos JAR externos podem ser importados e utilizados pela IDE.

12.1 Olhe e sintá!

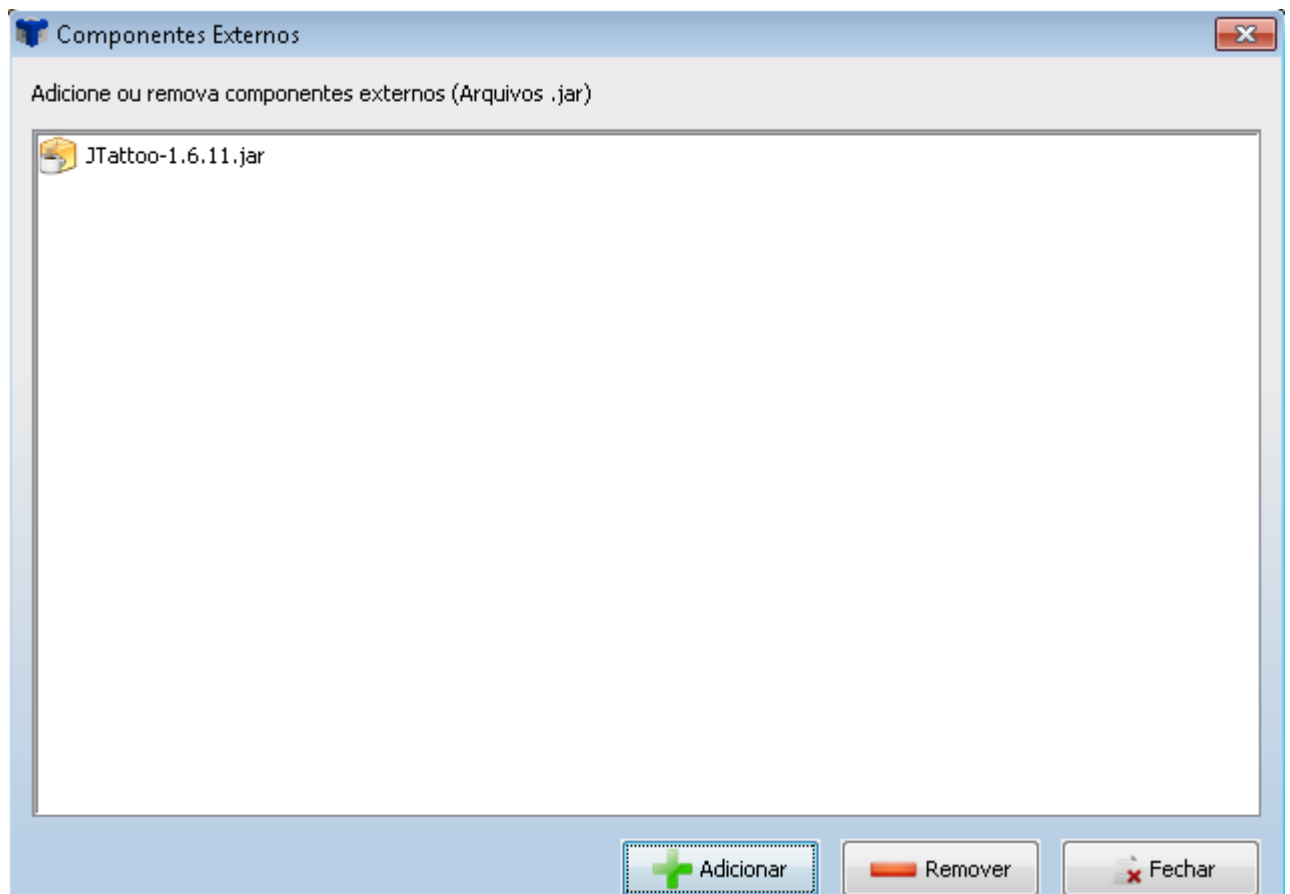
Em nosso exemplo, importaremos um componente externo que contém um pacote de **LookAndFeel** (aparência do sistema). Na sequência, alteraremos a aparência do nosso sistema. Crie um projeto e dê o nome de **MudandoAparencia**.

Baixe o arquivo JAR do projeto **JTattoo** em seu site (<http://www.jtattoo.net/Download.html>).

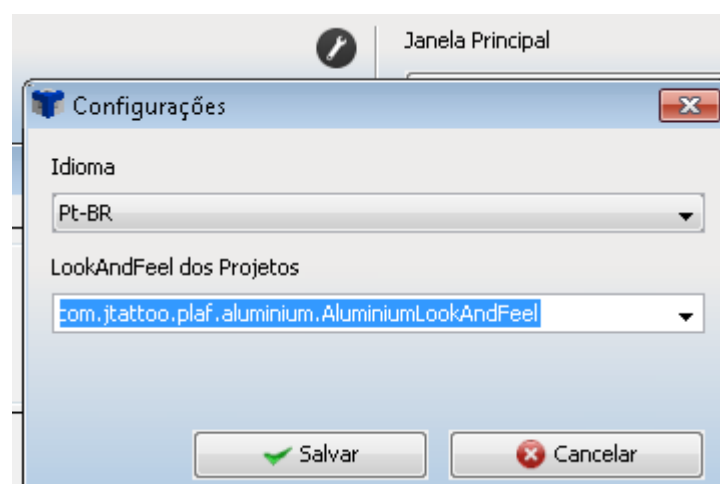
Clique com o botão direito no **Explorador de Projetos** e selecione a opção **Componentes Externos**.



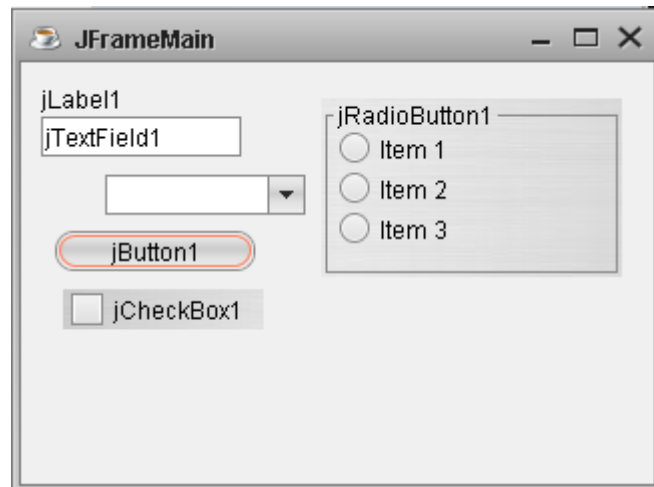
Na janela **Componentes Externos**, clique no botão **Adicionar** e selecione o arquivo JAR baixado. No meu caso, foi o **jTattoo-1.6.11.jar**. Aguarde a importação e feche a janela.



Na sequência, clique no botão Configuração (🔧), na Barra de ferramentas e mude o LookAndFeel dos Projetos para **com.jtattoo.plaf.aluminium.AluminiumLookAndFeel**. Pressione o botão **Salvar** e adicione alguns objetos à sua **JFrameMain**.



Salve e Execute o seu projeto para ver o resultado.



A configuração do **LookAndFeel** do TetrisIDE é geral para os projetos. Logo, quando você mudar de projeto, atente-se para o **LookAndFeel** salvo nas **Configurações**.

13 RELATÓRIOS COM O TETRISREPORT

Toda solução de software feita para o ambiente empresarial, com o intuito de oferecer apoio à gestão através de processamento de dados, deve considerar a geração de relatórios para impressão ou visualização. Para o Java, há um excelente e famoso projeto, que, inclusive, é utilizável no TetrisIDE, chamado **JasperReports** (<http://community.jaspersoft.com/project/jasperreports-library>).

Se estivermos buscando um suporte nativo à geração de relatórios na nossa ferramenta, podemos contar com o **TetrisReport**, que é um módulo integrado à IDE que nos permite confeccionar documentos através de código HTML. Os relatórios são automaticamente gerados em PDF, facilitando a portabilidade e comodidade para as soluções desenvolvidas com a IDE.

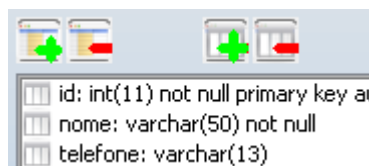
Caso não conheça a linguagem de marcação de texto HTML, dê uma olhada em <https://pt.wikipedia.org/wiki/HTML>.

13.1 Imprimindo uma Relação de Registros

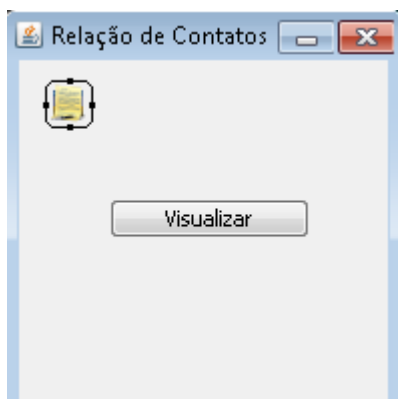
Para entendermos o funcionamento do TetrisReport, criaremos uma aplicação que, a partir de uma tabela populada de registros, imprimirá uma relação de seus itens. Para começarmos, crie um projeto com o nome **RelacaoContato** e mude o **Title** da **JFrameMain** para **Relação de Contatos**.

Crie uma tabela **contato** com as seguintes colunas:

Coluna	Tipo
id	int not null primary_key
nome	varchar(50) not null
telefone	varchar(13)



Adicione um Button e mude seu **Text** para **Visualizar**. Adicione um **Report** e modifique seu nome para **reportContato**. Efetue um duplo clique na propriedade **Report** do **reportContato**.



O TetrisReport trabalha com uma linguagem HTML modificada, sendo possível compor um relatório a partir de uma linguagem de marcação popular.

Há três identificadores no editor do relatório: #Header, #Detail e #Summary, referentes ao Cabeçalho, Detalhe e Sumário do relatório, respectivamente.

Quando queremos exibir o conteúdo de um campo de uma consulta SQL, digitamos o seu nome logo após um \$. Vamos compor o relatório deste exemplo para vermos como funciona na prática.

No editor de relatório aberto, insira o seguinte conteúdo:

```
#Header
<h3>
    Relação de Contatos
</h3>
<table border='1' width='100%'>
    <tr>
        <th width='40'>
            Id
        </th>
        <th width='200'>
            Nome
        </th>
        <th width='60'>
            Telefone
        </th>
    </tr>
#Detail
    <tr>
```

```

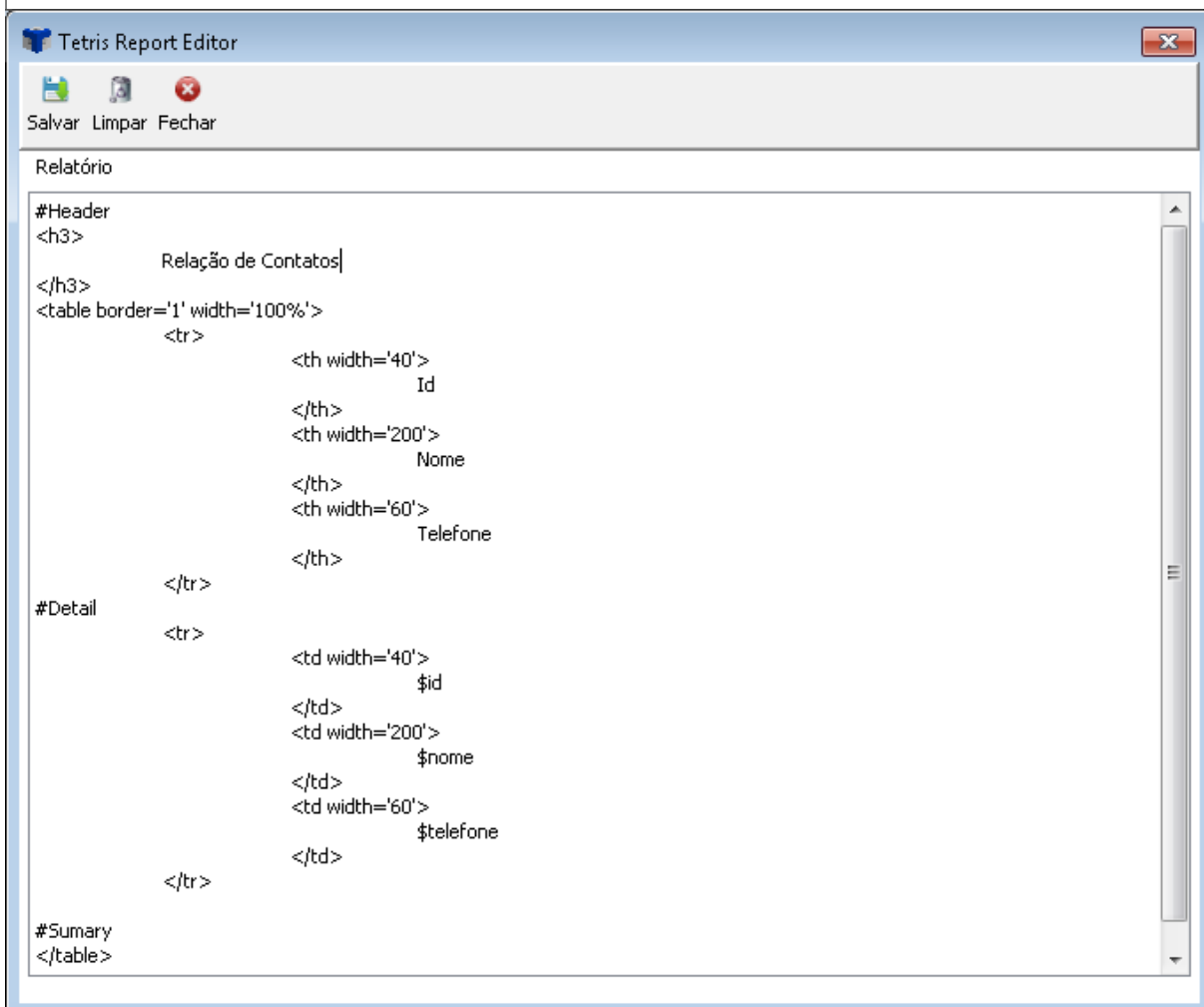
        <td width='40'>
            $id
        </td>
        <td width='200'>
            $nome
        </td>
        <td width='60'>
            $telefone
        </td>
    </tr>

```

```

#Summary
</table>

```

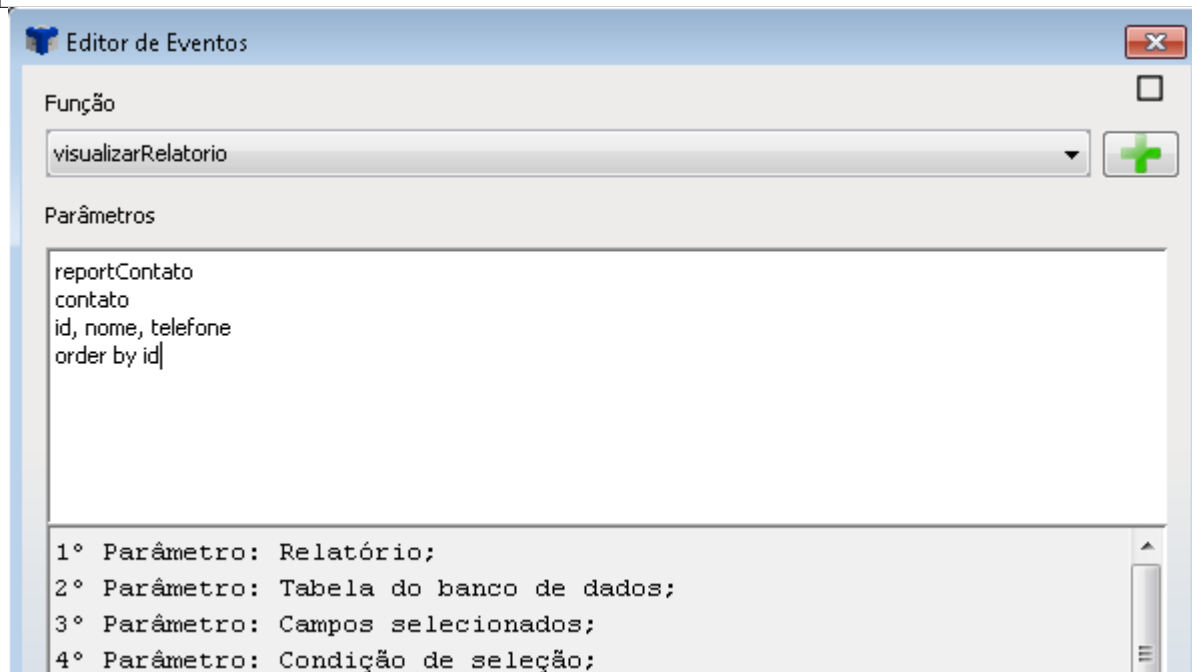


Neste exemplo, definimos um título com as tags **<h3></h3>** e uma tabela com 1 de borda e 100% de largura. Como foi dito, os campos da consulta SQL foram colocados após o identificador **#Detail**.

Salve o relatório e efetue um duplo clique no evento **OnClick** do

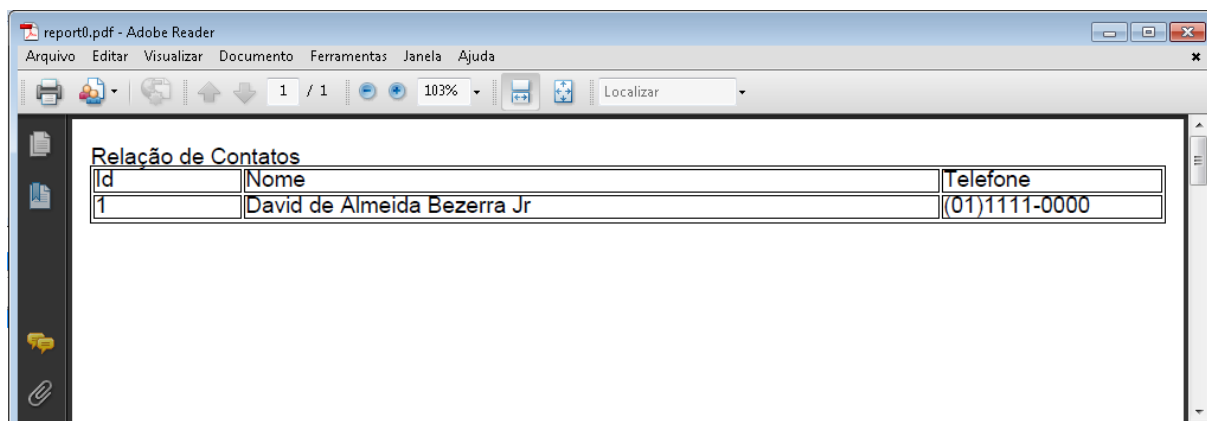
botão **Visualizar**. Adicione uma função **visualizarRelatorio** com os parâmetros:

```
reportContato  
contato  
id, nome, telefone  
order by id
```



A dica abaixo dos parâmetros explica-os muito bem. O primeiro parâmetro é o componente Report na janela, o segundo, a tabela do banco de dados. O terceiro são os campos da consulta SQL e o quarto, a condição e/ou a ordem da exibição da consulta.

Salve e Execute sua aplicação. Popule a tabela com dados e veja o resultado.



13.2 Variáveis no Relatório

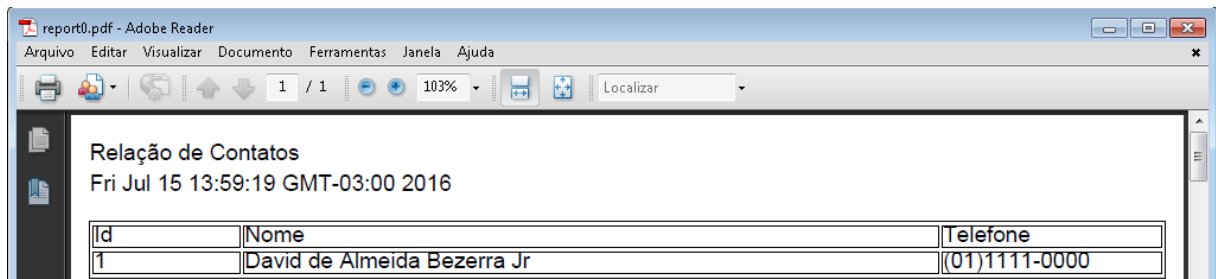
É possível inserirmos no relatório qualquer variável ou valor presente em um componente ou propriedade da janela, bastando concatenar o valor entre `$\"+` e `+$\"`. Vejamos no exemplo abaixo, onde criamos uma variável na janela que contém a data atual e inserimos no **#Header** do **reportContato**.

Vamos utilizar a classe **Date**, do pacote **java.util**. Para isso, modifique a propriedade **Import** da **JFrameMain** para **java.util.Date**. Adicione uma **Variable** à janela e mude seu nome para **data**. Modifique, também, o seu **Type** para **Date** e o seu **Value** para **new Date()**. Efetue duplo clique na propriedade **Report** do **reportContato** e adicione `$\"+getdata()+$\"` logo após o `</h3>`, ficando assim:

```
#Header
<h3>
    Relação de Contatos
</h3>
$"+getdata()+$"
<table border='1' width='100%'>
    <tr>
        <th width='40'>
            Id
        </th>
        <th width='200'>
            Nome
        </th>
        <th width='60'>
            Telefone
        </th>
    </tr>
#Detail
    <tr>
        <td width='40'>
            $id
        </td>
        <td width='200'>
            $nome
        </td>
        <td width='60'>
            $telefone
        </td>
    </tr>
#Summary
```

</table>

Salve, Execute e veja o resultado.



13.3 Código Java dentro do TetrisReport

Além da praticidade oferecida pelo TetrisReport, é também possível inserir código Java na composição do relatório, podendo, assim, trabalhar com sub detalhes, grupos e composições diferentes do padrão. Ainda no exemplo dos contatos, vamos mudar um requisito: o contato poderá ter mais de um telefone.

Crie uma tabela chamada **telefone** com as colunas:

Coluna	Tipo
id	int not null primary_key
contato_id	int not null
telefone	varchar(13) not null

Insira alguns telefones para cada contato no banco de dados e faça a seguinte alteração na propriedade **Report** do **reportContato**:

```
#Header
<h3>
    Relação de Contatos
</h3>
$\"+getdata()+$\"
<table border='1' width='100%'>
    <tr>
        <th width='40%'>
            Id
        </th>
        <th width='200%'>
```

```

        Nome
    </th>
    <th width='60'>
        Telefone
    </th>
</tr>
#Detail
<tr>
    <td width='40'>
        $id
    </td>
    <td width='200'>
        $nome
    </td>
    <td width='60'>
        $telefone
    </td>
</tr>
</table>

    $");
    Connection conn = getDB().getConnection();
    ResultSet resultSetTelefone = getDB().select(conn, $"telefone$",
    $"telefone$", $"where contato_id=$"+resultSet.getString($"id$")+ $"$");
    resultSetTelefone.first();

    if(resultSetTelefone.getRow()>0){
        do{
            reportContato.setDetail(reportContato.getDetail()+
    $"<br/>$"+resultSetTelefone.getString($"telefone$"));
        }while(resultSetTelefone.next());
    }
    resultSetTelefone.close();
    conn.close();
    reportContato.setDetail(reportContato.getDetail()+$"
    </td>
</tr>

#Summary
</table>

```

Primeiramente, fechamos a composição do #Detail com \$");. Na próxima linha, efetuamos a conexão com o banco de dados (**Connection conn = getDB().getConnection();**), para, em seguida, realizar a consulta SQL (**ResultSet resultSetTelefone = getDB().select(conn, \$"telefone\$", \$"telefone\$", \$"where contato_id=\$"+resultSet.getString(\$"id\$")+ \$"\$");**). Colocamos a resultSetTelefone na primeira posição (**resultSetTelefone.first();**) e verificamos se há registros (**if(resultSetTelefone.getRow())>0**). Se houver, adicionamos à

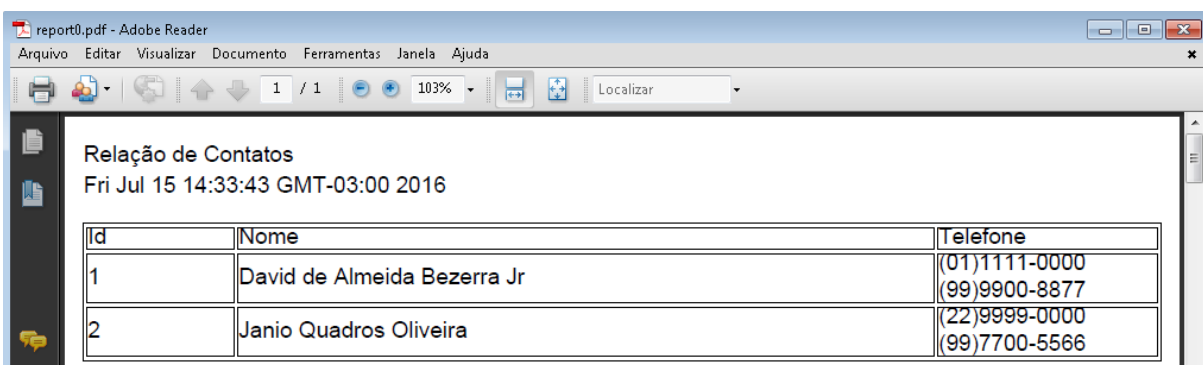
composição do **#Detail** do relatório (`reportContato.setDetail(reportContato.getDetail() + $!
$! + resultSetTelefone.getString($!telefone$!));`), fecha-se o `resultSetTelefone` (`resultSetTelefone.close();`) e a conexão (`conn.close();`) e continua-se a composição do **#Detail** (`reportContato.setDetail(reportContato.getDetail()+$!)`).

Dentro do identificador **#Detail**, é possível retornar o valor de um campo da consulta do relatório através do método `resultSet.getString($!campo$!)`, como fizemos com o campo `id` na consulta dos telefones (`resultSet.getString($!id$!)`).

Para finalizar, abra a propriedade **Import** da janela e adicione:

```
java.sql.Connection  
java.sql.ResultSet
```

Salve, Execute e veja o resultado.



The screenshot shows a PDF viewer window titled 'report0.pdf - Adobe Reader'. The report content is as follows:

Id	Nome	Telefone
1	David de Almeida Bezerra Jr	(01)1111-0000 (99)9900-8877
2	Janio Quadros Oliveira	(22)9999-0000 (99)7700-5566

14 TRABALHANDO COM OUTROS BANCOS DE DADOS (FIREBIRD)

A partir da versão 1.1, é possível desenvolver aplicativos no TetrísIDE integrando com outros SGDB's (Sistemas Gerenciadores de Bancos de Dados). Um ótimo exemplo, veremos com um banco de dados Firebird 2.5.

14.1 Instalando o Firebird

Primeiramente, efetue o download do instalador do Firebird em <https://www.firebirdsql.org/en/server-packages/>. A versão utilizada em nosso exemplo foi a 2.5.



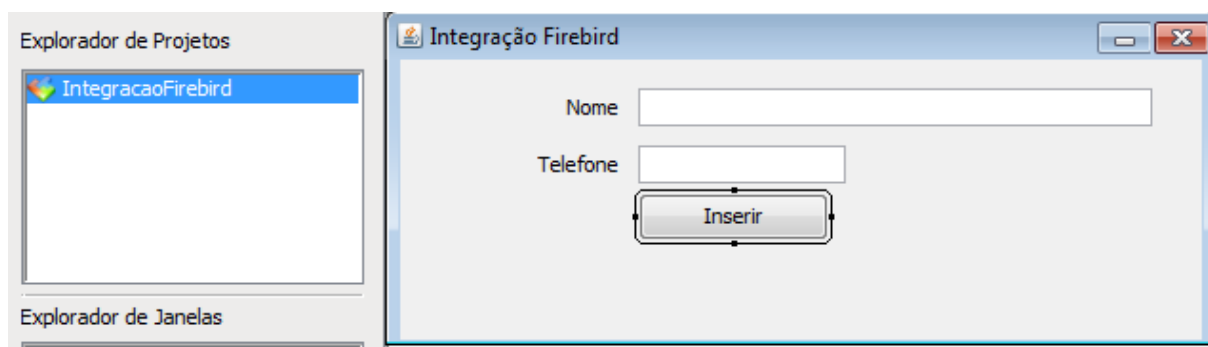
Siga todos os passos e finalize a instalação.

14.2 Inserindo Dados

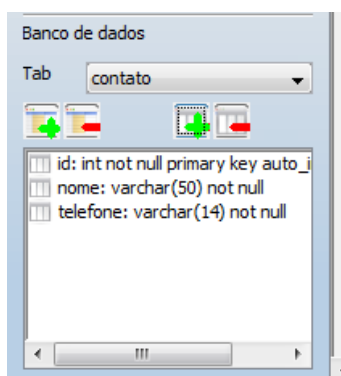
Crie um banco de dados, utilizando uma ferramenta como o IBExpert ou através da linha de comando, contendo a seguinte tabela e colunas:

Tabela: contato	
Coluna	Tipo
nome	varchar(50) not null
telefone	varchar(14) not null

Crie um projeto no TetrisIDE chamado **IntegracaoFirebird** e adicione dois **Labels**, dois **TextFields** e um **Button**. Modifique os nomes dos TextFields para **jTextFieldNome** e **jTextFieldTelefone**. Mude, também, a propriedade **Text** dos **Labels** para “**Nome**” e “**Telefone**”. Apague o conteúdo da propriedade **Text** dos TextFields e modifique a do Button para “**Inserir**”.



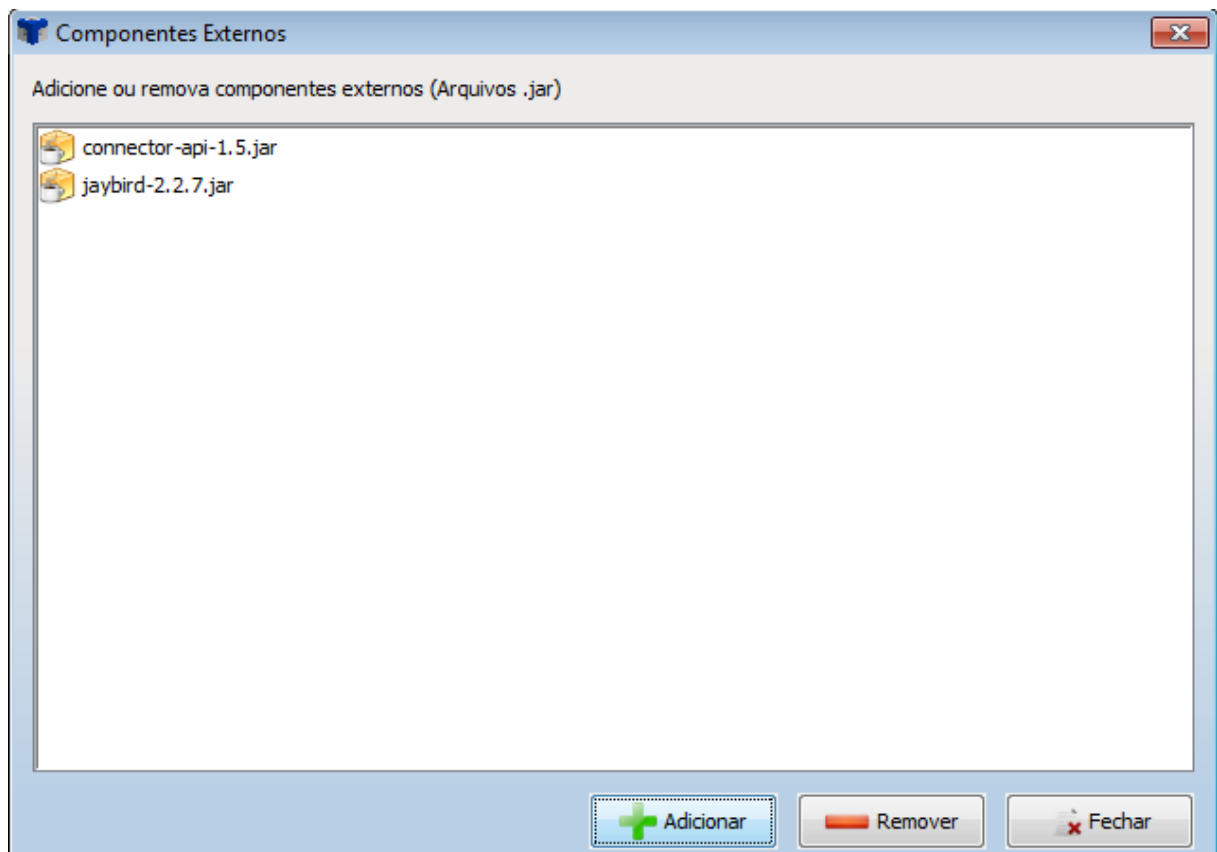
Insira uma tabela com o nome **contato** no Gerenciador de Banco de Dados com as mesmas colunas definidas em seu banco de dados Firebird.



Para podermos integrar o nosso projeto ao Firebird, precisamos incluir o driver de conexão em nosso sistema. Faça o download do Jaybird (biblioteca utilizada para integração Firebird-Java) em https://sourceforge.net/projects/firebird/files/firebird-jca-jdbc-driver/2.2.7-release/Jaybird-2.2.7-JDK_1.5.zip/download.

Descompacte e copie a DLL **jaybird22.dll** para o diretório **C:\Windows\system32**, caso seu sistema seja de 32 bits, ou a DLL **jaybird22_x64.dll** para **C:\Windows\SysWOW64**, caso seja 64 bits.

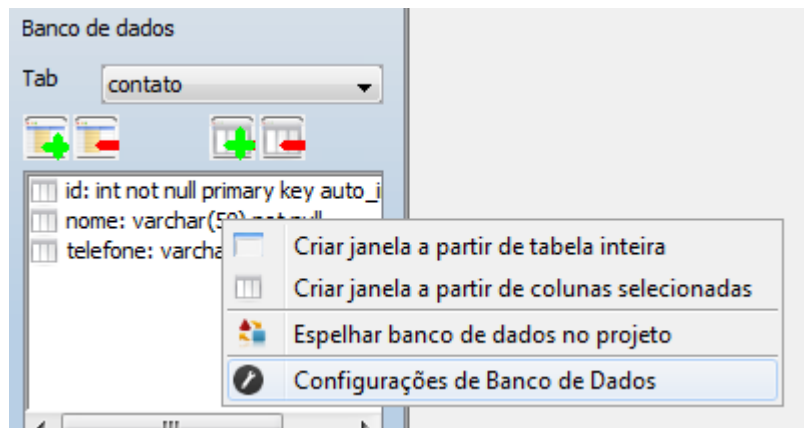
Adicione aos Componentes Externos do seu projeto os JAR's **jaybird-2.2.7.jar** e **lib/connector-api-1.5.jar**. Classes presentes nestes arquivos serão responsáveis pela conexão com o banco de dados.



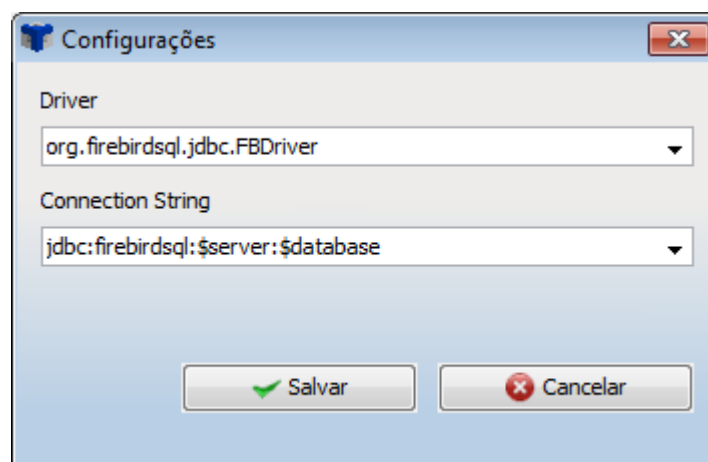
Com os componentes adicionados, precisamos agora configurar o nosso projeto para trabalhar com eles, modificando, nas configurações de banco de dados, o **Driver** que será carregado e a **Connection String**.

Clique com o botão direito do mouse no Gerenciador de Banco de

Dados e selecione a opção **Configurações de Banco de Dados**.



Na tela seguinte, modifique o **Driver** para **org.firebirdsql.jdbc.FBDriver** e a **Connection String** para **jdbc:firebirdsql:\$server:\$database**.

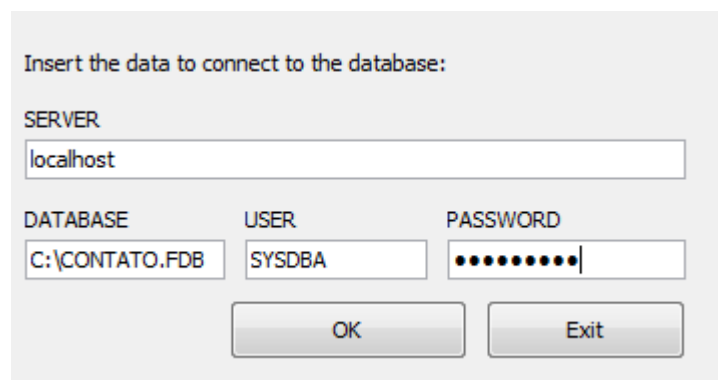


No momento da execução, as palavras-chaves **\$server** e **\$database** serão substituídas pelo **endereço do servidor** e **nome/caminho da base de dados**, respectivamente.

Salvas as configurações, nosso sistema está pronto para receber suas funções. Adicione uma função `gravarRegistro` no evento `OnClick` do botão `Inserir` com os seguintes parâmetros:

```
contato  
nome, telefone  
""+jTextFieldNome.getText()+", ""+jTextFieldTelefone.getText()+""
```

Salve, Execute e preencha os dados de conexão com o banco de dados. O **Server** é o endereço do servidor de banco de dados (se instalado em sua própria máquina, insira **localhost**). O **Database**, no caso do nosso exemplo, será o caminho do arquivo do banco de dados Firebird (exemplo: **C:\CONTATO.FDB**). Para o **User**, normalmente utiliza-se o **“SYSDBA”** para este SGDB, acompanhado do **Password “masterkey”**.



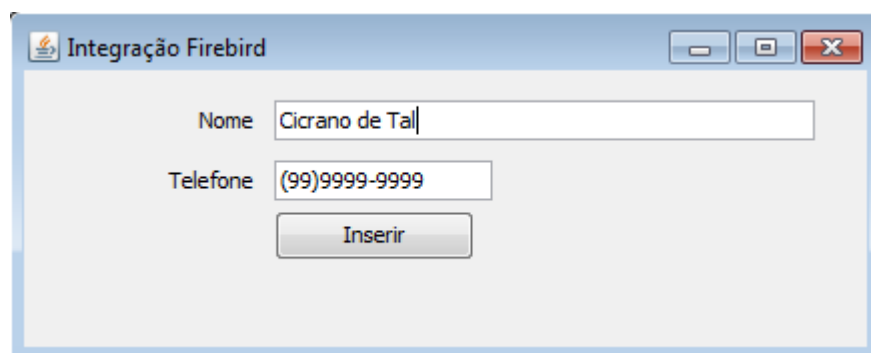
Insert the data to connect to the database:

SERVER
localhost

DATABASE USER PASSWORD
C:\CONTATO.FDB SYSDBA

OK Exit

Após o preenchimento, pressione o botão OK e execute novamente a aplicação.



Integração Firebird

Nome Cicrano de Tal

Telefone (99)9999-9999

Inserir

Insira alguns dados e visualize o resultado no banco de dados.

NOME	TELEFONE
► Fulano de Tal	(99)9999-9999
Cicrano de Tal	(99)9999-9999