

Curso Java



Autor: Lucas Abbade

Interface Gráfica de Usuário (GUI)

Interface Gráfica de Usuário

- JFrame
- JComponents
- Eventos

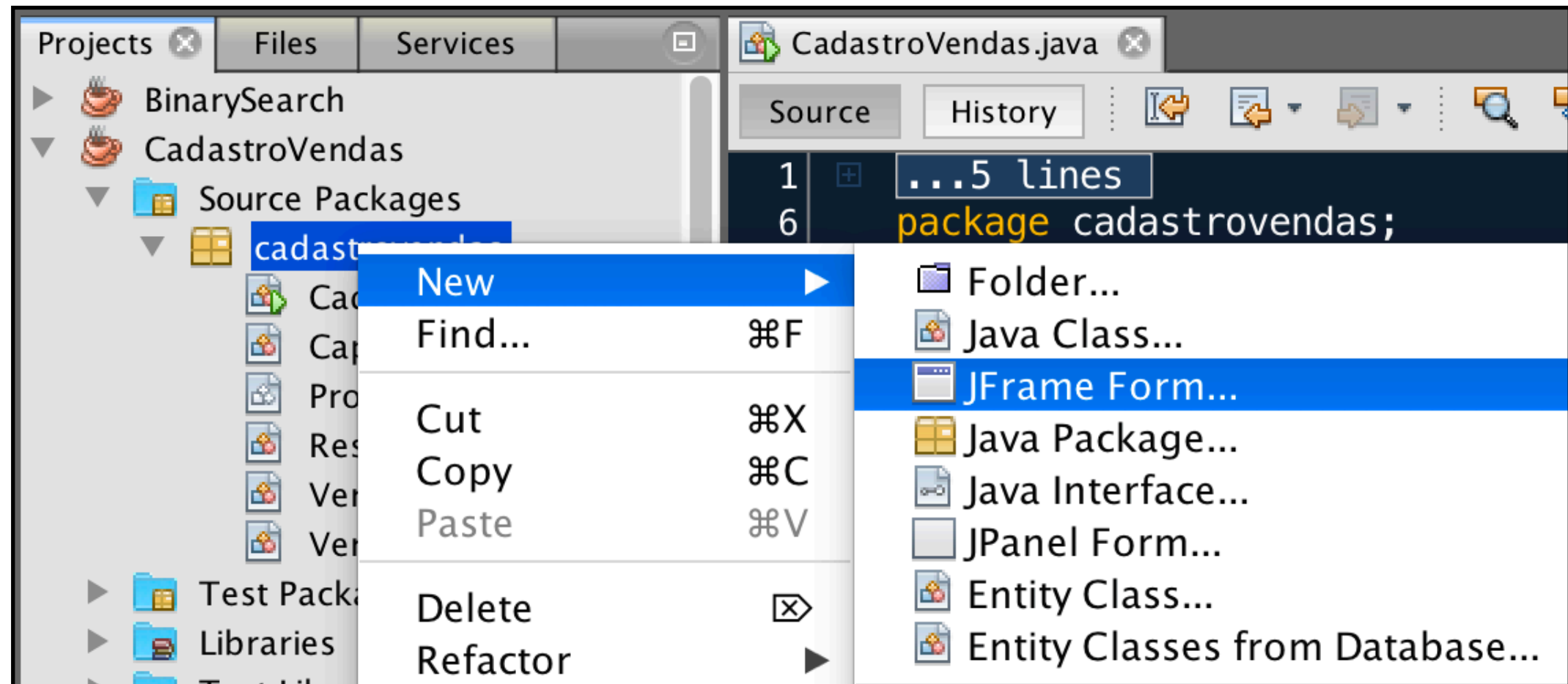
Interface Gráfica de Usuário

Vamos criar uma interface gráfica simples para tornar nosso programa Cadastro de Vendas muito mais simples.

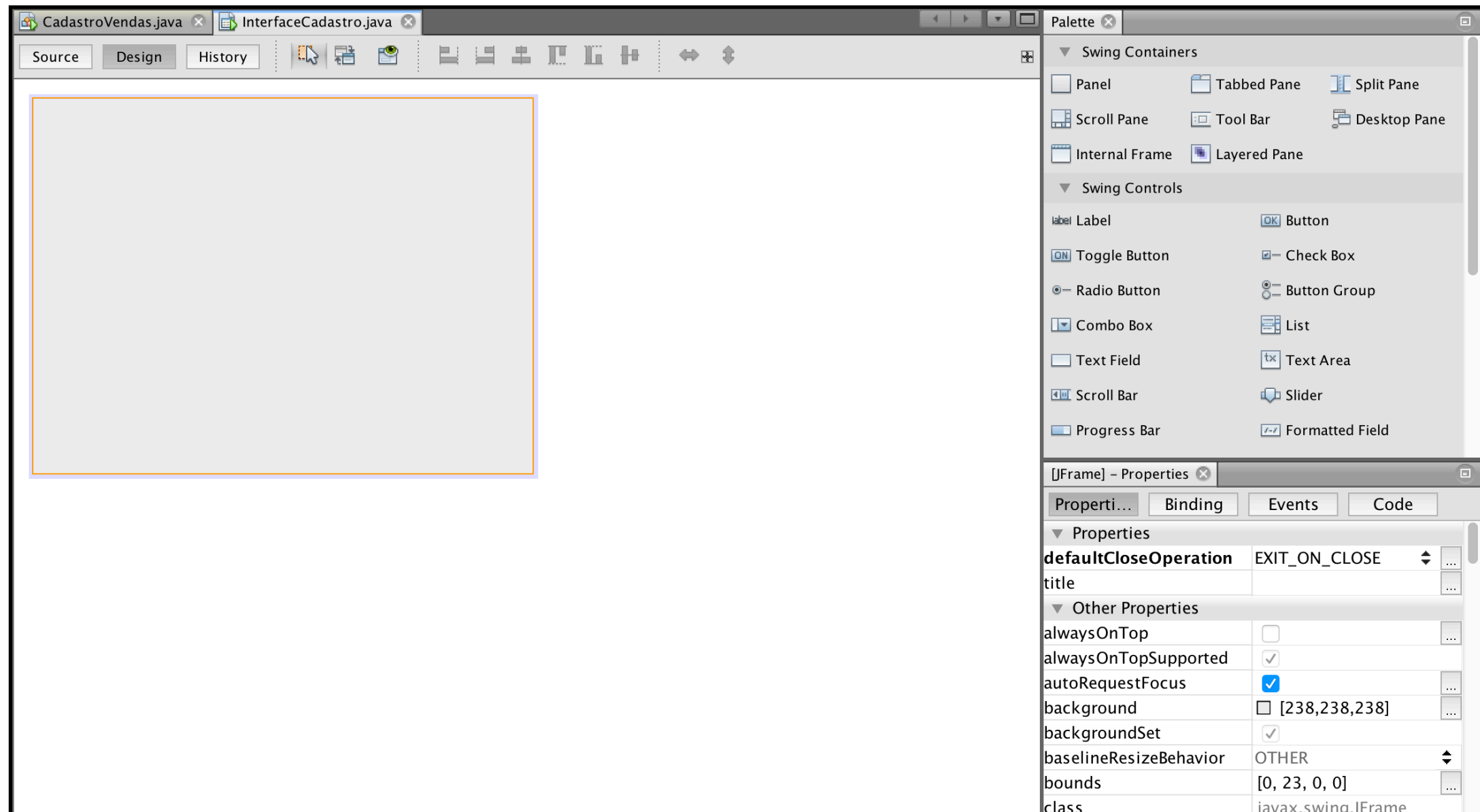
JFrame

Para começar nossa interface gráfica, primeiro é necessário criar uma classe do tipo *JFrame*.

JFrame



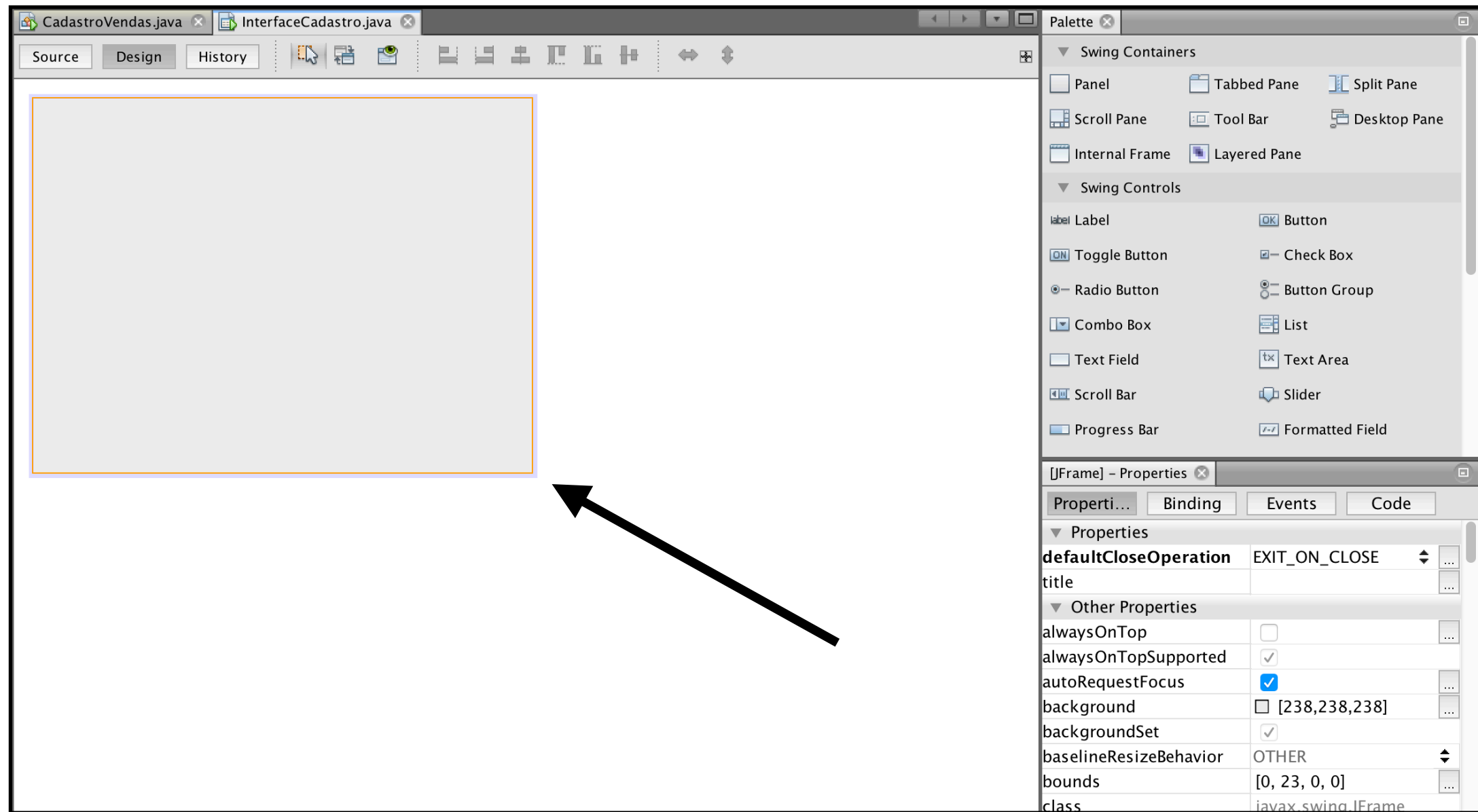
JFrame



JFrame

O quadrado cinza no meio, é nossa janela, que por enquanto está vazia. Podemos mudar seu tamanho arrastando a ponta.

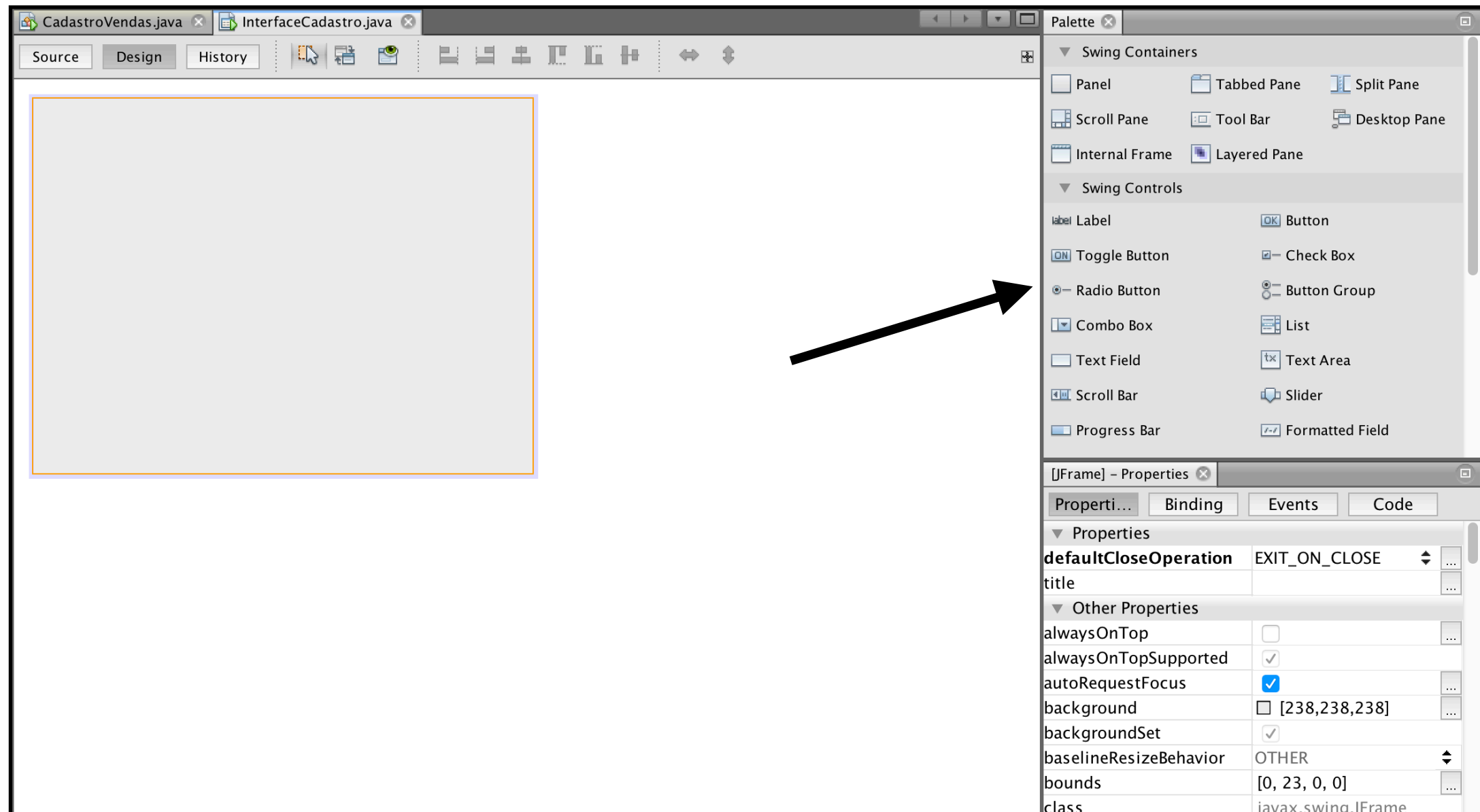
JFrame



JFrame

Do lado direito superior, estão disponíveis os diversos componentes que podemos usar em nossa interface.

JFrame



JComponents

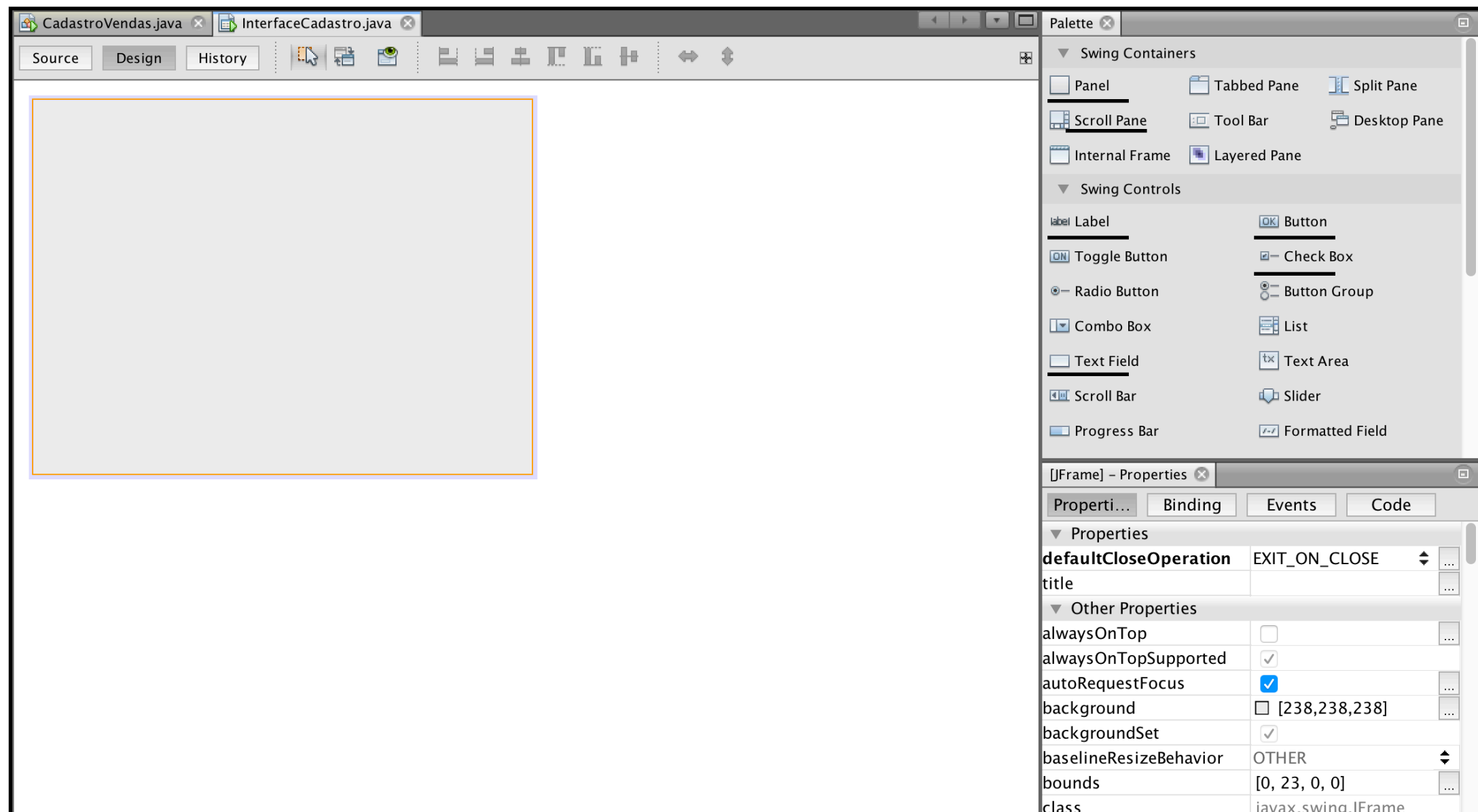
Estes componentes são chamados de
JComponents.

JComponents

Alguns exemplos muito utilizados:

- Panel
- Scroll Pane
- Label
- Button
- TextField
- Check Box

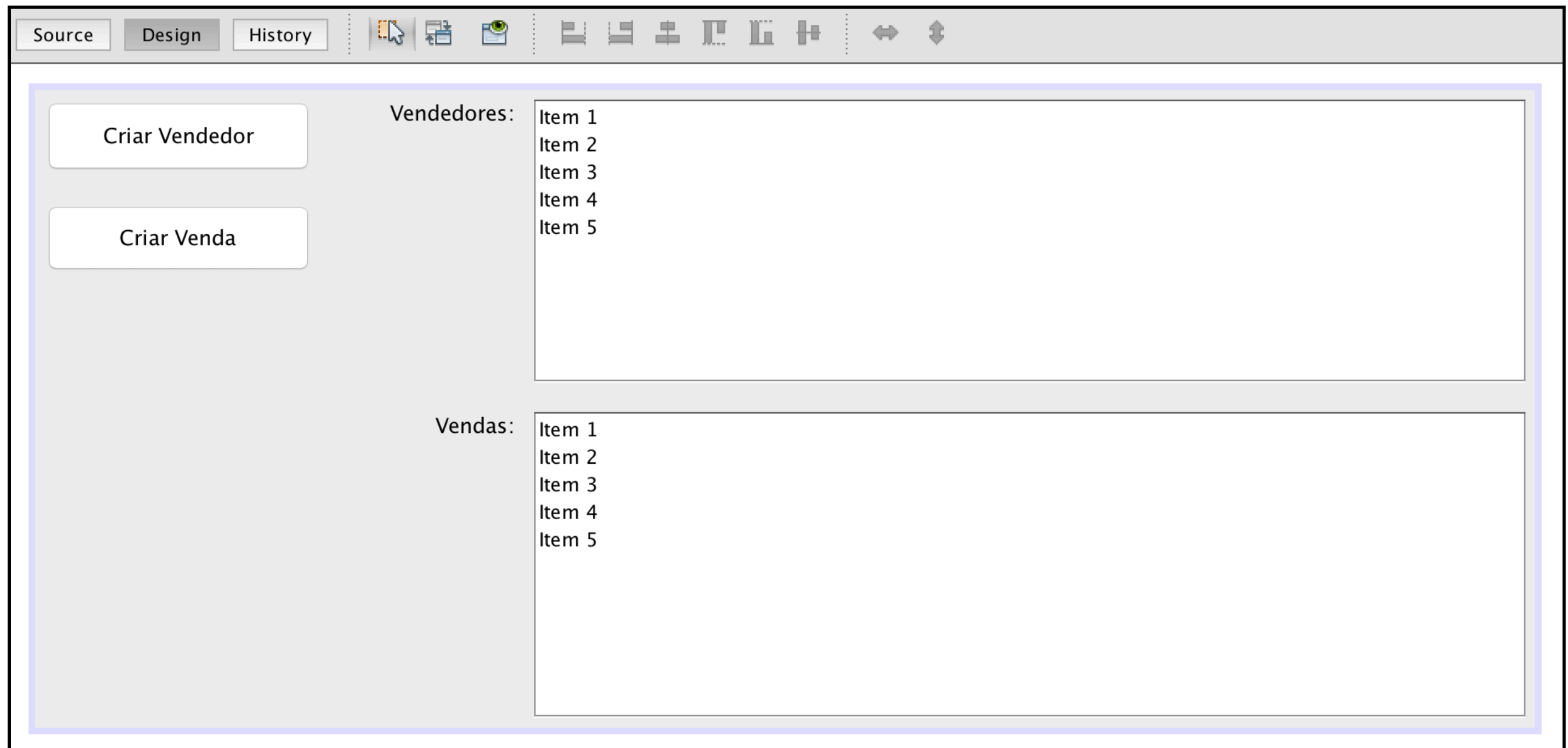
JComponents



JComponents

Criando a interface do programa:

JComponents



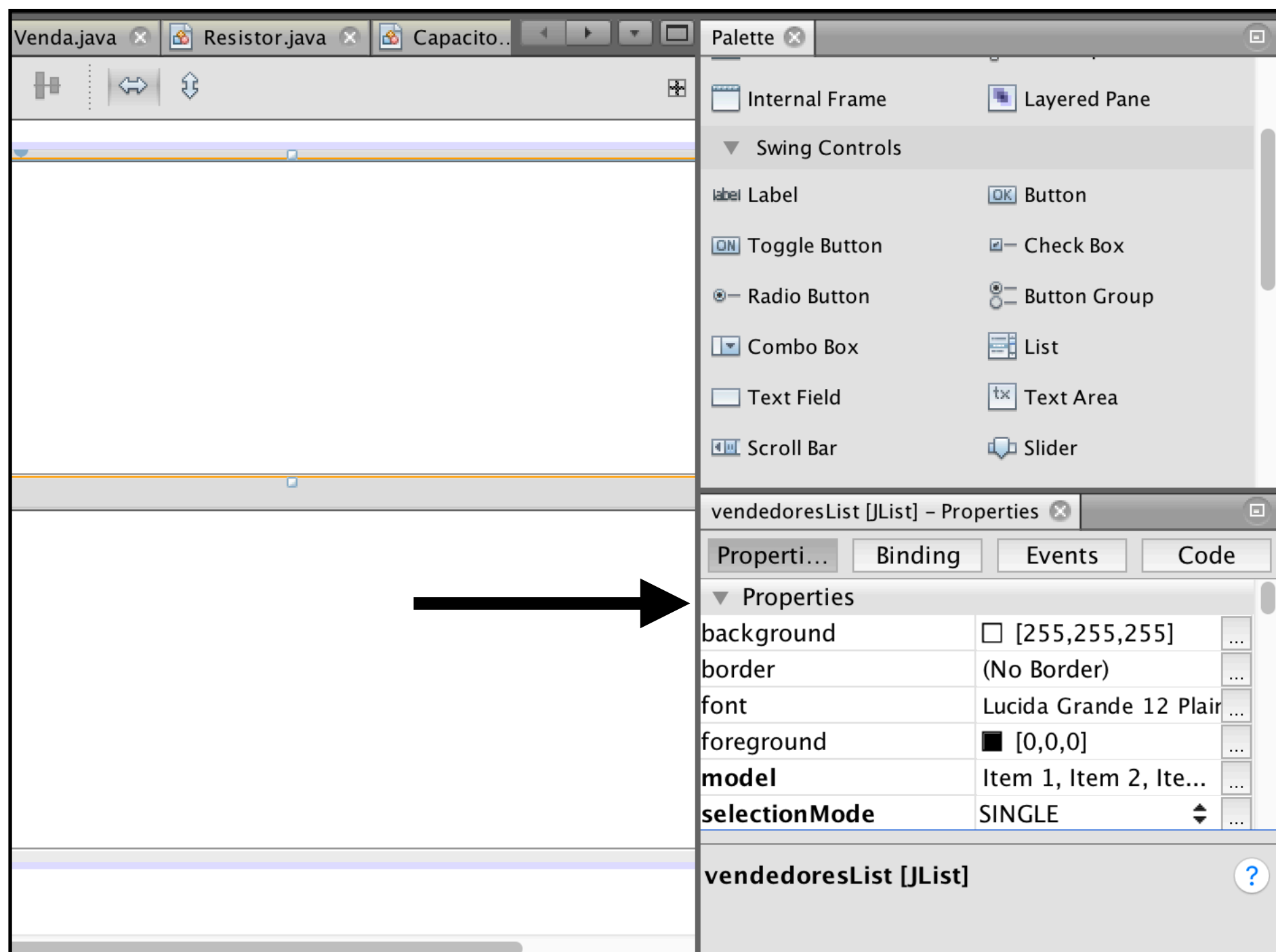
JComponents

Para esta interface, utilizamos 2 *Buttons*, 2 *Lists* e 2 *Labels*.

JComponents

Podemos alterar diversas propriedades de qualquer *JComponent* pelo painel “*properties*”, também do lado direito.

JComponents



JComponents

Cada *JComponent* usado na tela, é definido dentro da classe *JFrame* como um atributo.

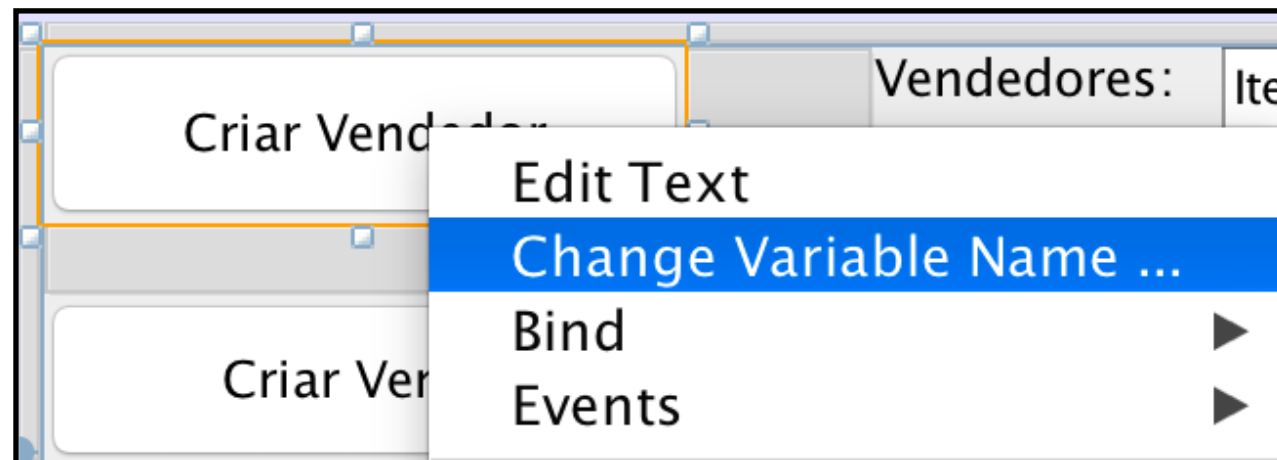
É possível mudar o nome dos componentes, para dar nomes mais significativos.

JComponents

Para isto, clique com o botão direito no componente que deseja mudar o nome, e selecione “mudar o nome da variável”.

Digite o nome desejado no campo que aparece.

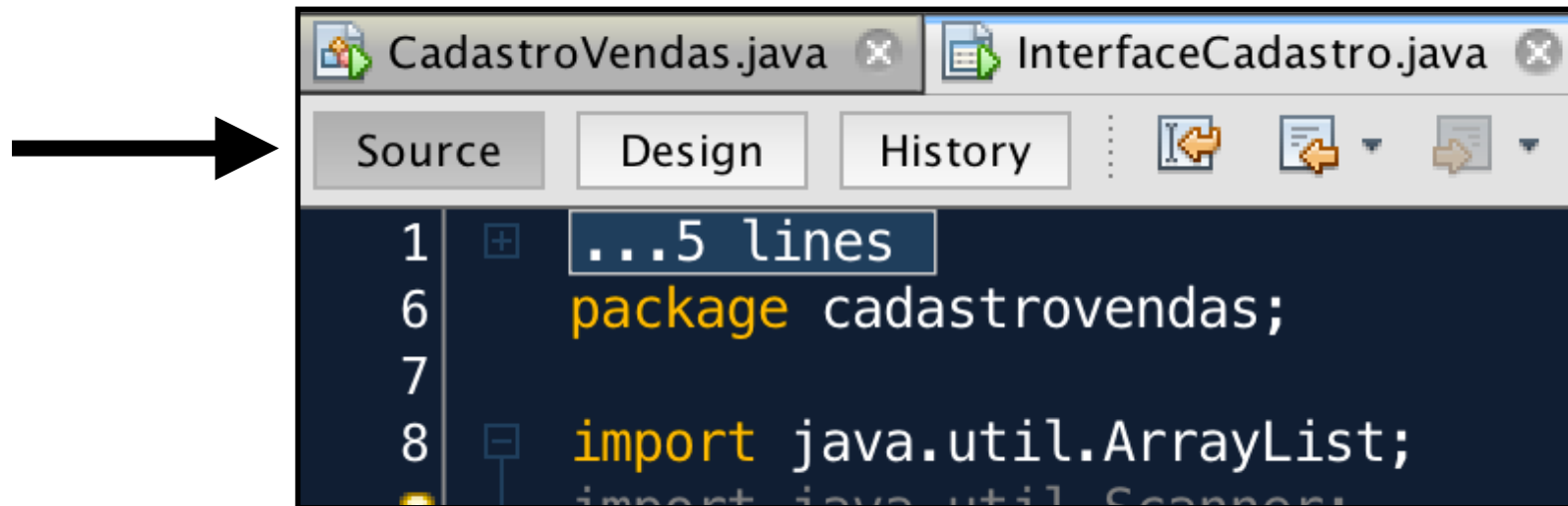
JComponents



JComponents

Se clicarmos no botão *source*, em cima do lado esquerdo, é aberto o editor novamente, com o código fonte de *JFrame*, que iremos alterar.

JComponents



JComponents

Podemos visualizar os componentes criados no final da classe *JFrame*.

JComponents

```
// Variables declaration – do not modify  
private javax.swing.JButton criarVendaButton;  
private javax.swing.JButton criarVendedorButton;  
private javax.swing.JLabel jLabel1;  
private javax.swing.JLabel jLabel2;  
private javax.swing.JPanel jPanel1;  
private javax.swing.JScrollPane jScrollPane1;  
private javax.swing.JScrollPane jScrollPane2;  
private javax.swing.JList<String> vendasList;  
private javax.swing.JList<String> vendedoresList;  
// End of variables declaration
```

JComponents

Note que não é possível alterar as variáveis diretamente no código.

JFrame - Source

Vamos agora criar as estruturas que guardarão as informações dos vendedores e vendas.

JFrame - Source

Iremos manter os *ArrayLists* vendedores e vendas, onde estarão todas as informações.

Além disso, agora temos 2 objetos do tipo “*DefaultListModel*”. Estes objetos são listas de *Strings* que serão utilizados pelas *JLists*, para montar as listas da interface.

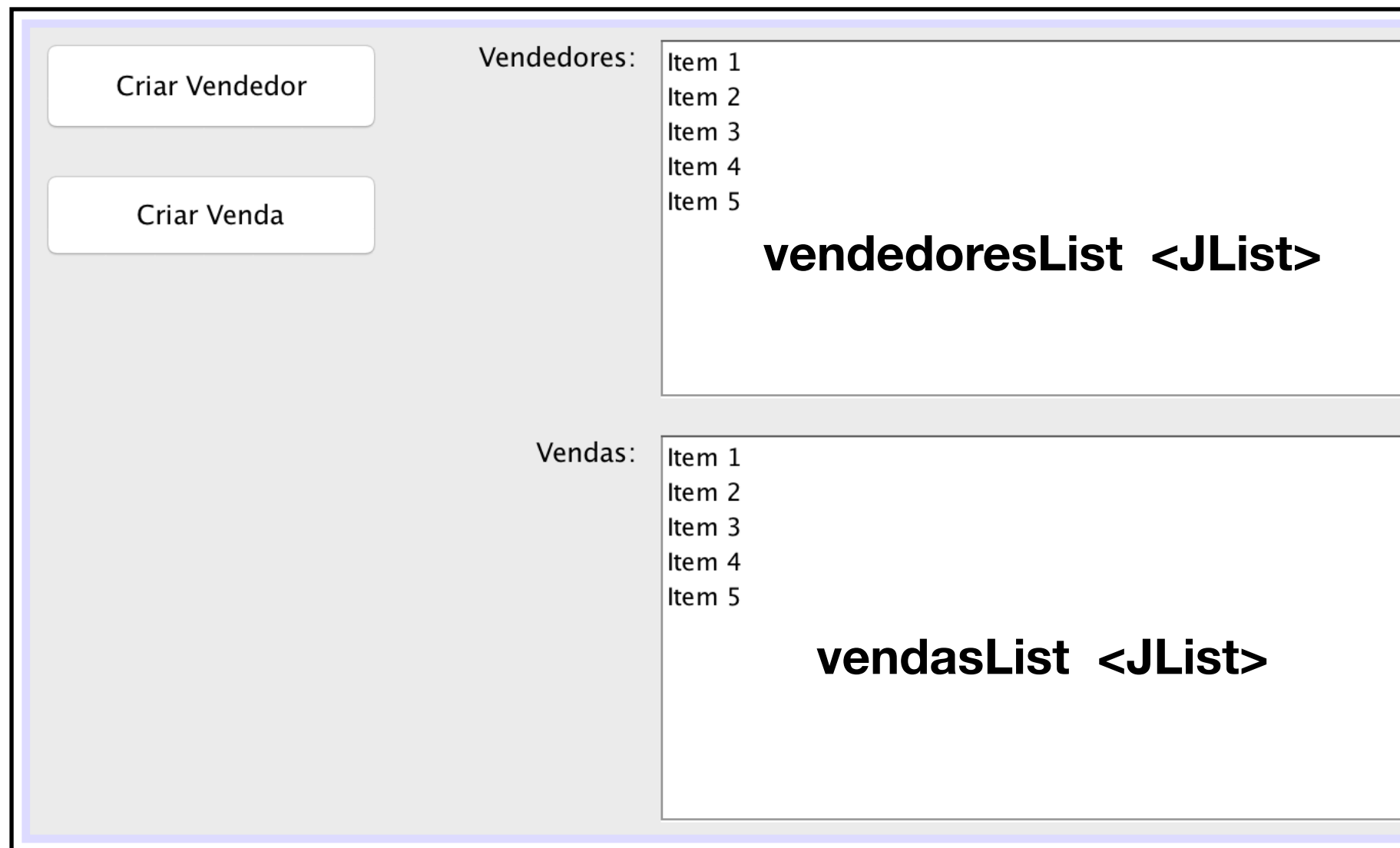
JFrame - Source

```
public class InterfaceCadastro extends javax.swing.JFrame
{
    ArrayList<Object> vendedores = new ArrayList<>();
    ArrayList<Object> vendas = new ArrayList<>();

    DefaultListModel<String> vendedoresModel = new DefaultListModel<>();
    DefaultListModel<String> vendasModel = new DefaultListModel<>();

    /** Creates new form InterfaceCadastro ...3 lines */
    public InterfaceCadastro()
    {
        initComponents();
        vendedoresModel.removeAllElements();
        vendasModel.removeAllElements();
        vendedoresList.setModel(vendedoresModel);
        vendasList.setModel(vendasModel);
    }
}
```

JFrame - Source



JFrame - Source

As *JLists* recebem os dados que irão mostrar dos objetos *DefaultListModel*, quando estes são alterados, a *JList* atualiza seu conteúdo automaticamente.

JFrame - Source

Para associar o *JList* a um *DefaultListModel*, utilizamos o método “*.setModel()*” da *JList*.

JFrame - Source

Agora precisamos ser capazes de criar vendedores e vendas, pra isso vamos usar os botões em conjunto com uma classe muito util do Java, chamada *JOptionPane*.

Eventos

Primeiro vamos criar os eventos dos botões.

Eventos

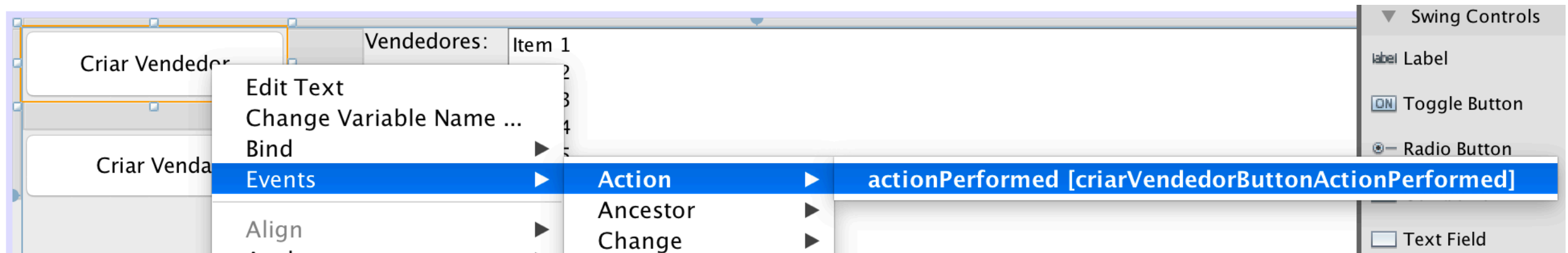
Eventos são funções que são chamadas quando alguma coisa acontece. Nesse caso, o nosso evento será o usuário clicar no botão.

Eventos

Para criar o evento, clique com o botão direito no componente que deseja criar o evento, então em “eventos” e selecione o evento desejado.

No caso do botão, usaremos o evento “ação” -> “ação realizada”

Eventos



Eventos

Isso cria uma função no *source* do *JFrame*, a ser executada quando o evento “disparar”.

Eventos

```
private void criarVendedorButtonActionPerformed(java.awt.event.ActionEvent evt)
{
    vendedores.add(criarVendedor());
    refresh();
}

private void criarVendaButtonActionPerformed(java.awt.event.ActionEvent evt)
{
    vendas.add(criarVenda());
    refresh();
}
```


Eventos

Assim como nas declarações dos *JComponents*, também não somos capazes de alterar a estrutura do evento diretamente.

Eventos

Iremos então, chamar uma função que criará o vendedor ou a venda, incluiremos no *ArrayList* correto, e atualizaremos os *JLists*, para exibir o novo valor adicionado.

Eventos

Começão pelos métodos de criação:

Eventos

```
public Vendedor criarVendedor() {
    String nome = JOptionPane.showInputDialog("Digite o nome do vendedor");
    return new Vendedor(nome);
}

public Venda criarVenda() {
    int id = Integer.parseInt(JOptionPane.showInputDialog("Qual o ID do vendedor? "));
    int produtoVendido = Integer.parseInt(JOptionPane.showInputDialog("Qual produto foi vendido (1 ou 2)?"));
    int quantidade = Integer.parseInt(JOptionPane.showInputDialog("Qual a quantidade vendida?"));
    Produto p;

    if (produtoVendido == 1) {
        p = new Resistor(2.50, "1K", 10); // so pra exemplo
    } else {
        p = new Capacitor(0.10, 1000, 0.1);
    }

    return new Venda(id, p, quantidade);
}
```

JOptionPane

Como estamos agora trabalhando com uma interface gráfica, não podemos requisitar dados ao usuário por meio da linha de comando.

JOptionPane

Uma das opções mais simples de requisitar dados e exibir mensagens por interface, é o *JOptionPane*.

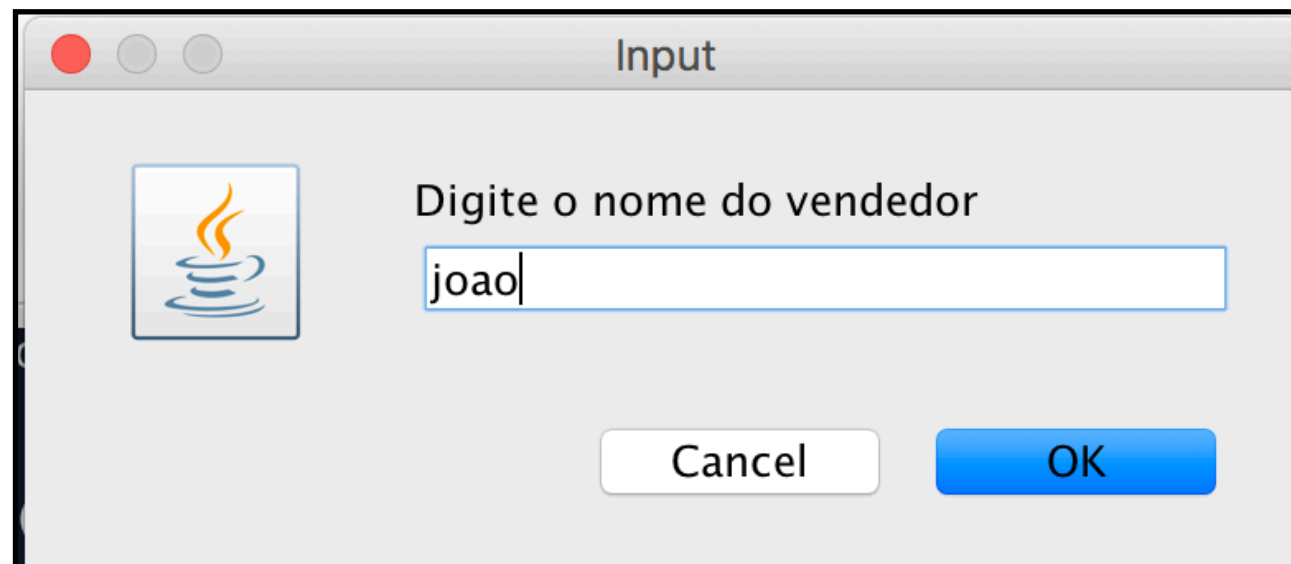
JOptionPane

O *JOptionPane* é uma janela que se cria rapidamente, capaz de exibir mensagens, requisitar uma entrada, pedir confirmação, exibir erros, et cetera.

JOptionPane

O retorno do *JOptionPane* é sempre uma *String*, portanto precisamos convertê-lo para *int* quando necessário.

JOptionPane



JFrame - Source

Uma vez criado o objeto desejado e adicionado na sua respectiva lista, chamamos o método “*refresh*” para atualizar a lista.

JFrame - Source

```
private void refresh() {  
    repopulateListModel(vendedoresModel, vendedores);  
    repopulateListModel(vendasModel, vendas);  
}  
  
private void repopulateListModel(DefaultListModel<String> model, ArrayList<Object> source){  
    model.clear();  
  
    for (int i = 0; i < source.size(); i++){  
        model.addElement(source.get(i).toString());  
    }  
}
```

JFrame - Source

Para atualizar os *JLists*, precisamos apenas atualizar o *DefaultListModel* à que eles estão associados.

JFrame - Source

O *DefaultListModel* deve conter, obrigatoriamente, apenas *Strings*, por isso não podemos salvar os objetos diretamente neles.

JFrame - Source

Note a abstração do tipo de *ArrayList* por meio do tipo *Object*.

JFrame - Source

Todo objeto no Java, herda por padrão da classe *Object*, é dela que sai a função *.toString()*, por exemplo, por isso podemos abstrair qualquer classe, mesmo que não tenham relação aparente, por meio da classe *Object*.

JFrame

Nossa interface para o programa do cadastro de vendas está concluída, porém você já deve ter notado que toda vez que reiniciamos o programa, perdemos todos os dados, como manter os nossos dados então?

Banco de Dados

Banco de Dados

- Criação do banco
- DAO
- Conexão
- CRUD

Banco de Dados

Bancos de dados são aplicações muito robustas com o propósito de armazenar dados, de forma mais simples e rápida que com um arquivo.

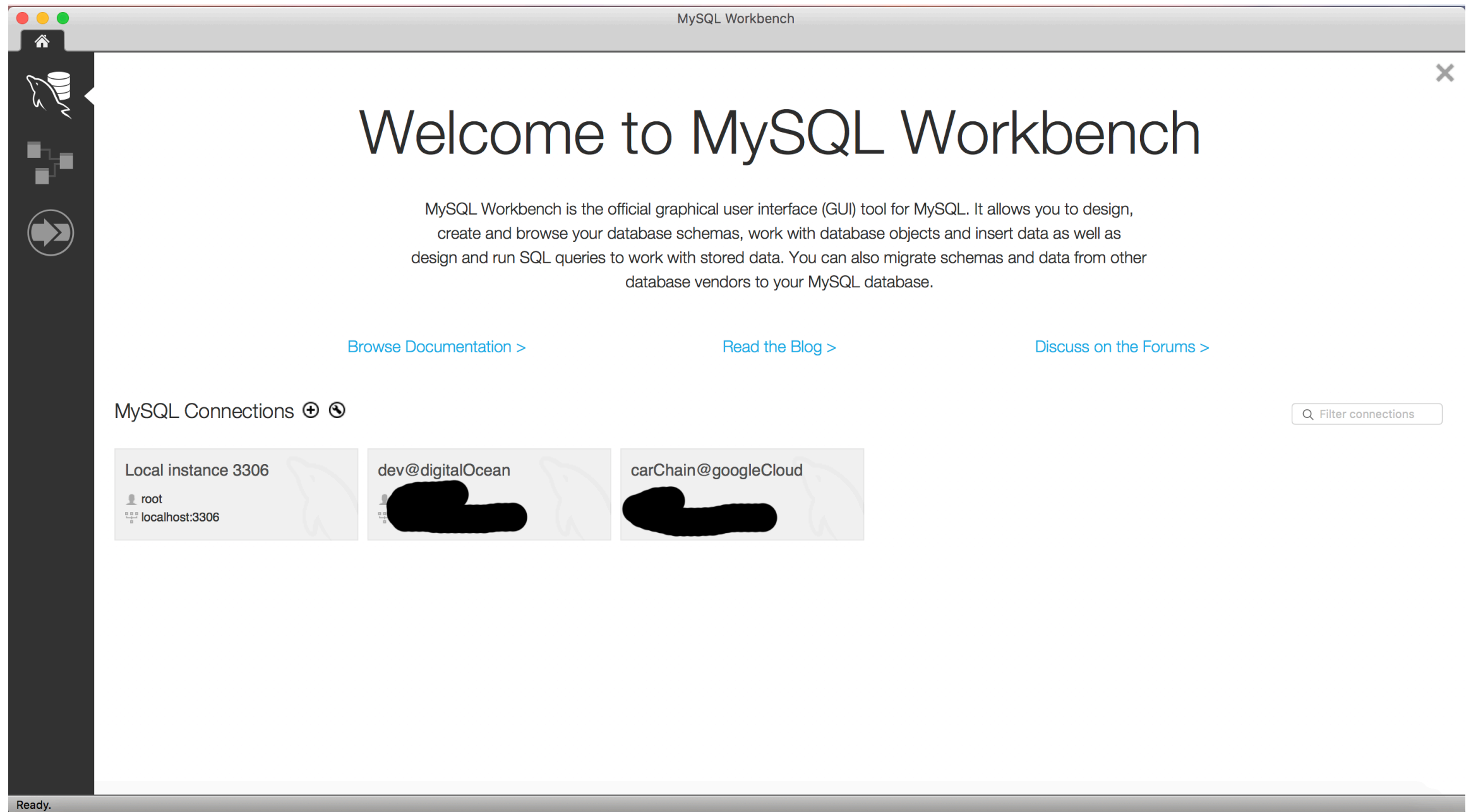
Banco de Dados

Existem diversos tipos de Bancos de dados, os mais comuns são os relacionais. Para este projeto utilizaremos o MySQL, um banco de dados relacional *open-source*.

Banco de Dados

Primeiramente, vamos abrir o MySQL Workbench, para podermos criar o nosso banco de dados.

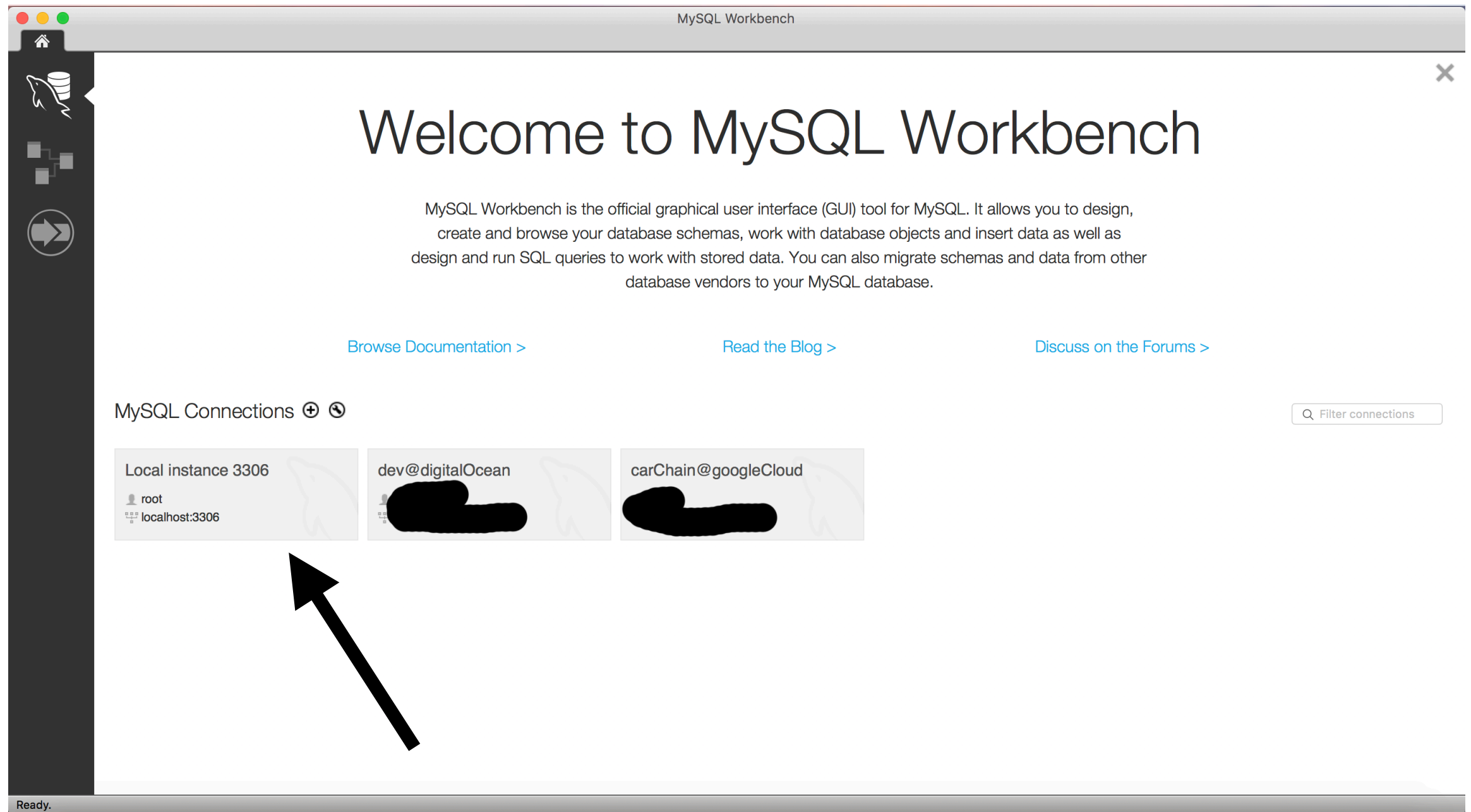
Banco de Dados



Banco de Dados

Clique em “*Local Instance*” para acessar o banco local da máquina.

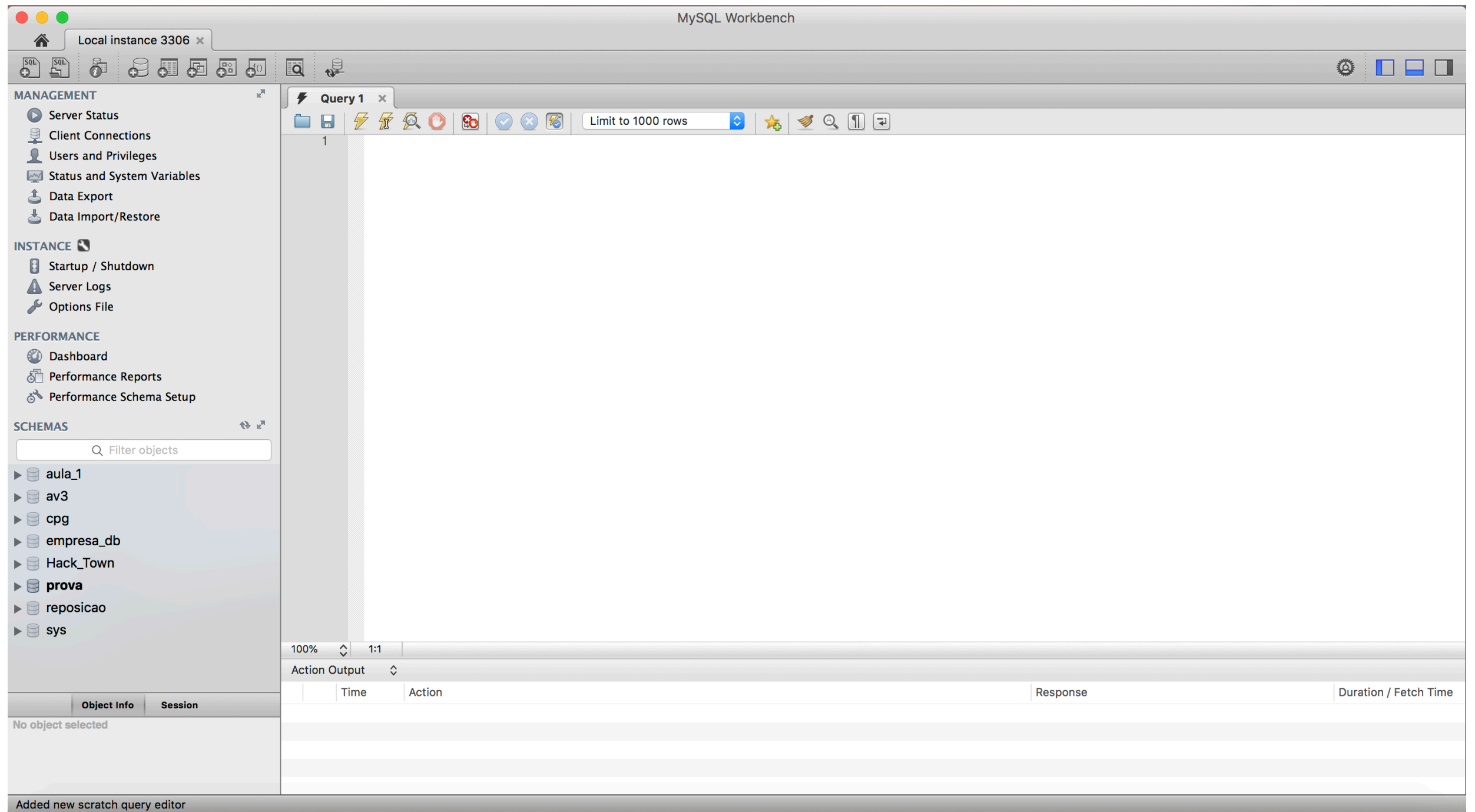
Banco de Dados



Banco de Dados

Se abrirá uma tela como a seguinte:

Banco de Dados



Banco de Dados

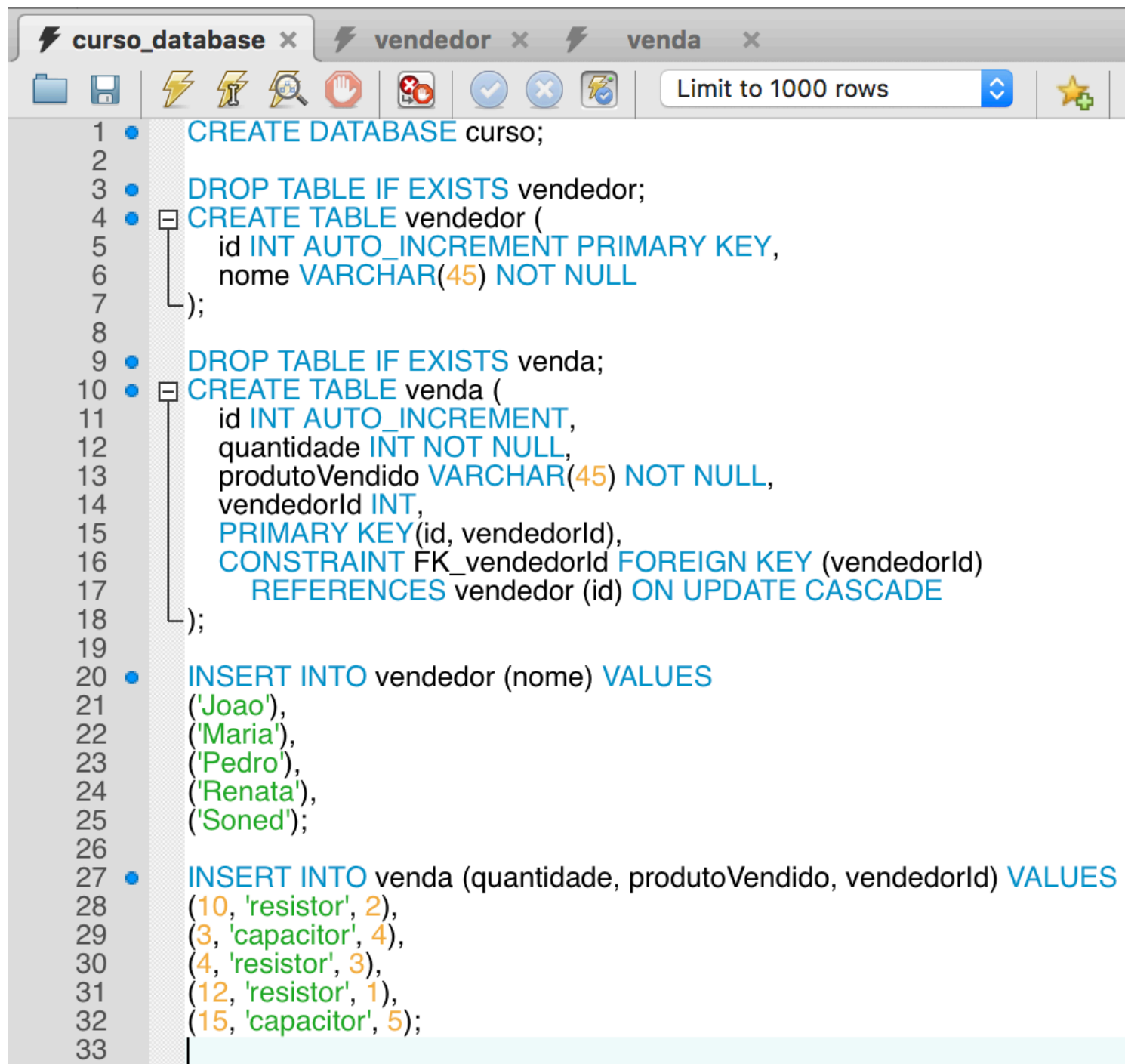
Aqui podemos digitar comandos na linguagem *SQL*, porém como este não é o foco do curso, o código para criar o banco a ser utilizado está disponível em:

<http://www.github.com/LRAbbade/Curso-Java/>

Banco de Dados

Baixe o arquivo *.sql* e execute-o dentro do *MySQL Workbench*. Seu banco de dados será criado.

Banco de Dados



The screenshot shows a database management tool interface with three tabs: 'curso_database', 'vendedor', and 'venda'. The 'curso_database' tab is active, displaying a list of SQL statements. The interface includes a toolbar with icons for file operations, execution, and a 'Limit to 1000 rows' dropdown. The SQL code is as follows:

```
1 CREATE DATABASE curso;
2
3 DROP TABLE IF EXISTS vendedor;
4 CREATE TABLE vendedor (
5     id INT AUTO_INCREMENT PRIMARY KEY,
6     nome VARCHAR(45) NOT NULL
7 );
8
9 DROP TABLE IF EXISTS venda;
10 CREATE TABLE venda (
11     id INT AUTO_INCREMENT,
12     quantidade INT NOT NULL,
13     produtoVendido VARCHAR(45) NOT NULL,
14     vendedorId INT,
15     PRIMARY KEY(id, vendedorId),
16     CONSTRAINT FK_vendedorId FOREIGN KEY (vendedorId)
17     REFERENCES vendedor (id) ON UPDATE CASCADE
18 );
19
20 INSERT INTO vendedor (nome) VALUES
21 ('Joao'),
22 ('Maria'),
23 ('Pedro'),
24 ('Renata'),
25 ('Soned');
26
27 INSERT INTO venda (quantidade, produtoVendido, vendedorId) VALUES
28 (10, 'resistor', 2),
29 (3, 'capacitor', 4),
30 (4, 'resistor', 3),
31 (12, 'resistor', 1),
32 (15, 'capacitor', 5);
33
```

Banco de Dados

Agora que o banco está criado, precisamos criar uma classe Java para nos conectar ao banco de dados.

Banco de Dados

Esta classe é comumente chamada de DAO
(Data Access Object).

Banco de Dados

O objetivo do DAO é abstrair toda a parte de conexão com o banco de dados para uma classe responsável apenas por isso.

Banco de Dados

A classe DAO implementa a conexão com o banco de dados e os métodos CRUD (*Create, Read, Update Delete*).

Banco de Dados

Leia a classe DAO do projeto com calma para tentar entender ao certo como ela funciona.

<https://github.com/LRAbbade/Curso-Java/blob/master/CadastroVendasComInterface> e [BD/src/cadastrovendas/DAO.java](#)

Banco de Dados

Para que a classe DAO funcione corretamente, será necessário importar a biblioteca do MySQL.

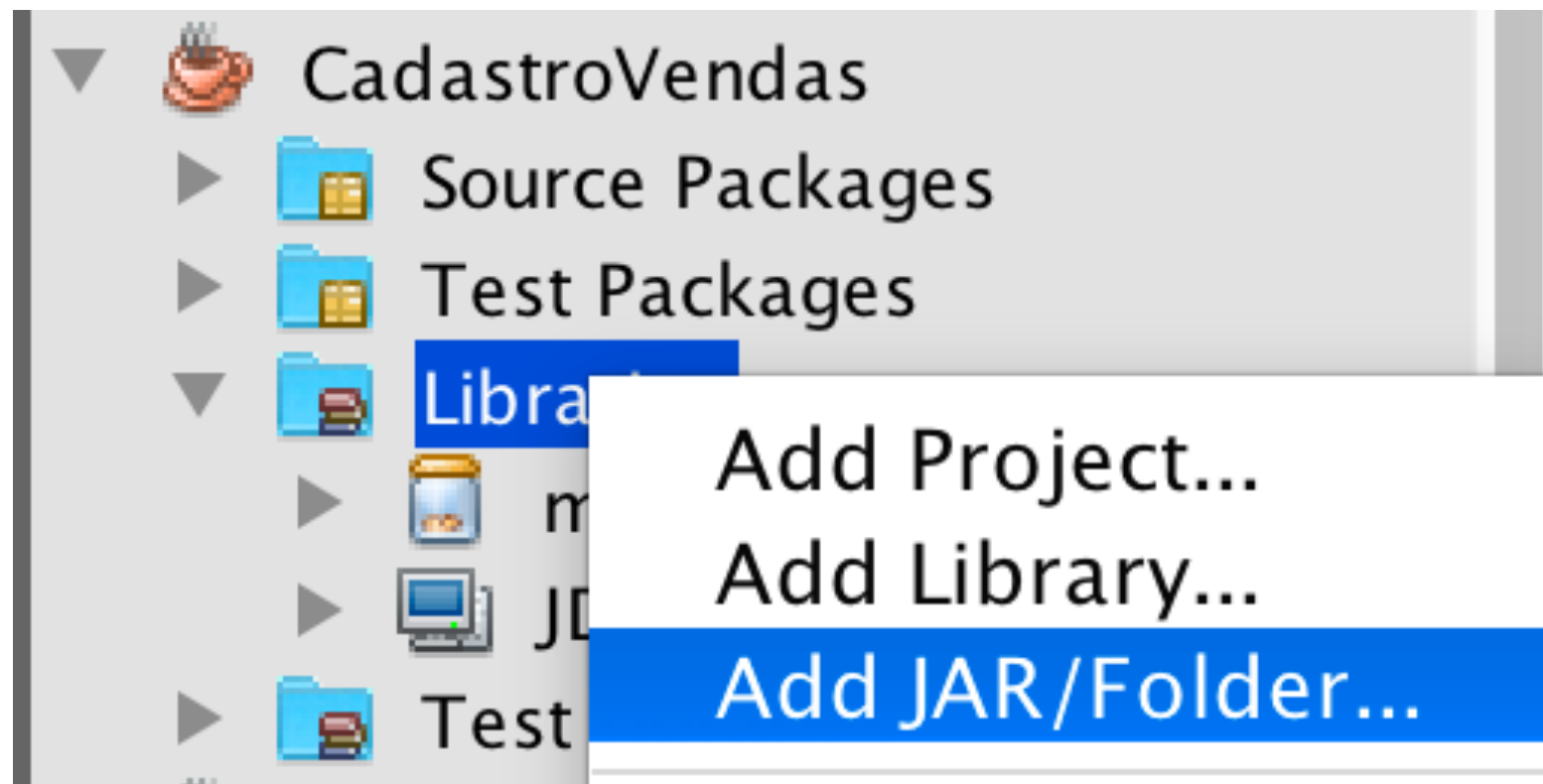
A mesma encontra-se disponível em:

<https://github.com/LRAbbade/Curso-Java/tree/master/Libs>

Banco de Dados

Após baixá-la, vá no explorador de projetos do NetBeans (canto esquerdo), clique com o botão direito na pasta “Bibliotecas”, e selecione “Adicionar .jar”.

Banco de Dados



Banco de Dados

Selecione então o arquivo .jar do MySQL baixado no seu computador, e pronto.

RESTful APIs

RESTful APIs

- HTTP
- GET
- Status Codes
- JSON

RESTful APIs

RESTful APIs (*Application Programming Interface*) são interfaces que se pode usar para ter acesso à algum dado público.

São códigos disponíveis para que um programador faça o pedido de alguma coisa pela internet.

HTTP

Para fazer o pedido, é necessário utilizar o protocolo HTTP (*Hyper Text Transfer Protocol*), que permite a comunicação entre *softwares online*.

HTTP

O protocolo HTTP possui diversos métodos, porém só iremos explorar o mais comum, o GET.

GET

O método GET, é utilizado quando deseja-se requisitar algum dado de uma API.

GET

Para fazer um *GET request* no Java, utilizaremos a classe *Requests*, que já está pronta e disponível no projeto “HTTPTest”, em:

github.com/LRAbbade/Curso-Java/

GET

Não é necessário entender o que a classe Requests está fazendo, o nosso objetivo é apenas usá-la.

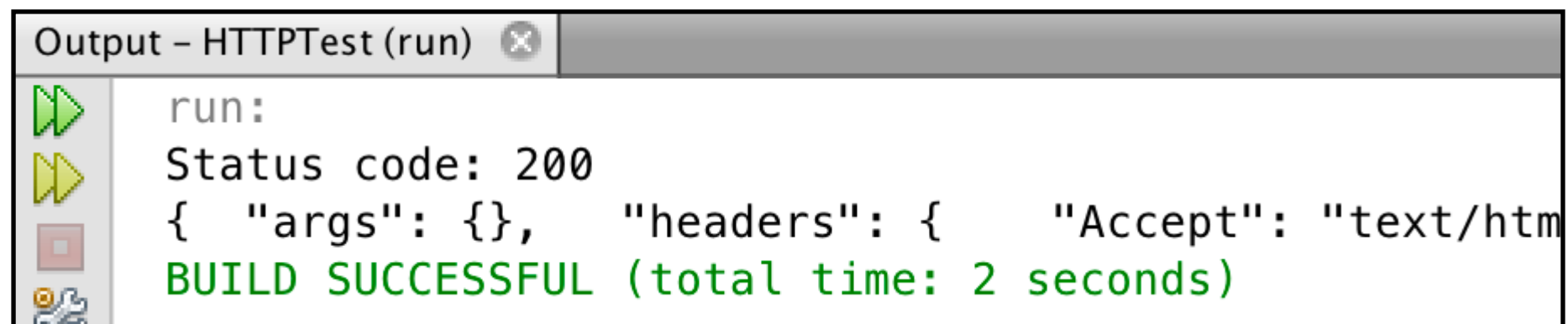
GET

Para usá-la, basta enviar uma URL como parâmetro da função `.get`, como exemplo, podemos fazer um request ao site de teste “httpbin”, que serve exclusivamente para testar requests.

GET

```
String testUrl = "https://httpbin.org/get";  
String r = Requests.get(testUrl);  
System.out.println(r);
```


GET



```
run:  
Status code: 200  
{  "args": {},  "headers": {    "Accept": "text/html"  
BUILD SUCCESSFUL (total time: 2 seconds)
```

GET

Repare que temos dois resultados, o primeiro, é o *status_code* da resposta, o segundo, uma longa string.

Status Code

Status Code é um código que nos diz se o nosso *request* teve sucesso ou não.

Existem diversos *status codes*, o mais importante é saber que 2xx significa sucesso, e 4xx significa falha.

JSON

A linha longa que recebemos como resposta, na verdade é um JSON (Javascript Object Notation). JSONs são amplamente utilizados por APIs por sua facilidade em representar diversos tipos de estruturas de dados.

JSON

Para conseguirmos manipular o JSON, vamos utilizar a biblioteca *org.json*, que deve ser incluída no projeto da mesma forma que o .jar do MySQL que fizemos no projeto anterior.

Esta também está disponível na pasta Lib, no repositório do curso no GitHub.

JSON

Após incluir a biblioteca no seu projeto, podemos tratar JSONs com maior facilidade e utilizar de APIs para diversas coisas, seja pegar notícias pela internet, informação de clima, bolsa de valores, redes sociais, entre outras.

APIs

Para tenha interesse em buscar por diferentes APIs, veja o site:

<https://www.programmableweb.com/>

Este compila milhares de APIs disponíveis na internet, suas rotas e descrições.

Fim

