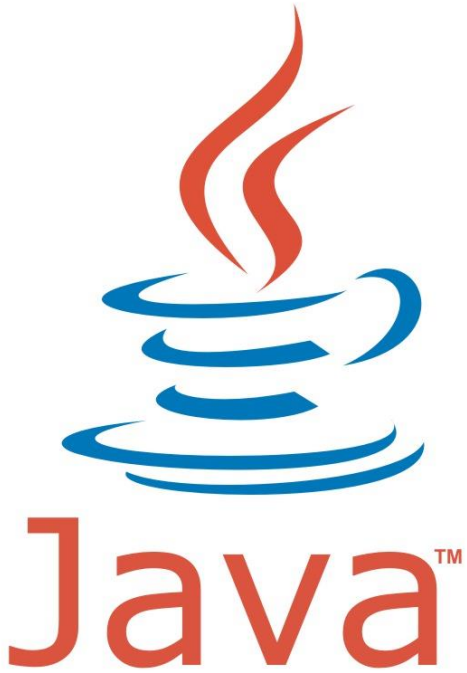
Java e Orientação a Objetos



Autor: Samuel Cruz

Threads

Thread

Muitas vezes precisamos “rodar duas coisas ao mesmo tempo” dentro de uma aplicação.

Por exemplo, imagine um programa que gera um relatório muito grande em .pdf. É um processo demorado e enquanto ele carrega as informações necessárias é interessante dar alguma satisfação ao usuário, como por exemplo uma barrinha de progresso.

Este tipo de problema pode ser facilmente resolvido usando as famosas **Threads**

Thread

Uma Thread nada mais é do que uma linha de execução do programa. Nós a utilizamos para fazer coisas em "paralelo".

Thread

Existem duas formas de se criar Threads em Java:

1. Estendendo da classe Thread;
2. Implementando a *Interface Runnable;*

Para ambos os casos, o programador deverá utilizar o método *run()*. Este método deverá possuir o código que rodará em "paralelo" com os outros códigos.

Thread

1. Estendendo da Classe Thread

```
public class GeraPDF extends Thread{

    @Override
    public void run() {
        // Código para gerar o .pdf
    }
}
```

```
public class BarraDeProgresso extends Thread{

    @Override
    public void run(){
        // Código para gerar a barra de progresso
    }
}
```

Thread

2. Implementando a *Interface* Runnable

```
public class GeraPDF implements Runnable{

    @Override
    public void run() {
        // Código para gerar o .pdf
    }
}
```

```
public class BarraDeProgresso implements Runnable{

    @Override
    public void run(){
        // Código para gerar a barra de progresso
    }
}
```

Thread

Mas e agora? Como executar estes métodos?

Esta resposta varia de acordo com a sua implementação

Thread

Caso tenha estendido da Classe Thread:

```
GeraPDF thread1 = new GeraPDF();
thread1.start();

BarraDeProgresso thread2 = new BarraDeProgresso();
thread2.start();
```

Thread

Caso tenha implementado a Interface Runnable:

```
GeraPDF o1 = new GeraPDF();
Thread t1 = new Thread(o1);
t1.start();

BarraDeProgresso o2 = new BarraDeProgresso();
Thread t2 = new Thread(o2);
t2.start();
```

Thread

O método *start()* é o método que inicia a nossa Thread. Ele irá executar o método *run()* e executar o seu conteúdo.

Cuidado ao trabalhar com Threads! Elas não tem uma ordem de execução padronizada, ou seja, não é recomendando trabalhar com Threads quando se deseja ações que exigem uma ordem determinada.

Thread

Para fazer uma Thread dormir por um tempo (a fim de executá-la mais devagar), utilizamos da função **Thread.sleep(n)**, onde **n** é o tempo que a Thread fica dormente (em milissegundos)

```
try {
    Thread.sleep(300);
} catch (InterruptedException ex) {
    System.out.println("Erro na Thread! " + ex.getMessage());
}
```

Obs.: Obrigatoriamente devemos fazer o tratamento de erros ao usar este método.

Exercício 9

Usando de Threads, crie uma aplicação que simule uma corrida de Atletas durante uma competição de 100 metros. O vencedor é o primeiro a ultrapassar a marca de 100 m.

Crie uma classe Atleta com os atributos: valorPercorrido = 0 e Nome. No construtor deve ser passado o Nome do atleta;

No método run, Crie um número aleatório através da Classe Random da seguinte forma: Random r = new Random();

Faça uma lógica que é gerado um número aleatório de 0 a 10 e que seja incrementado na variável *valorPercorrido.* Para isto use: r.nextInt(10);

Quando o *valorPercorrido* for maior ou igual a 100, imprima o nome do atleta que cruzou a linha de chegada e, se for o caso, interrompa a Thread.

Interface Gráfica com o Usuário (GUI)

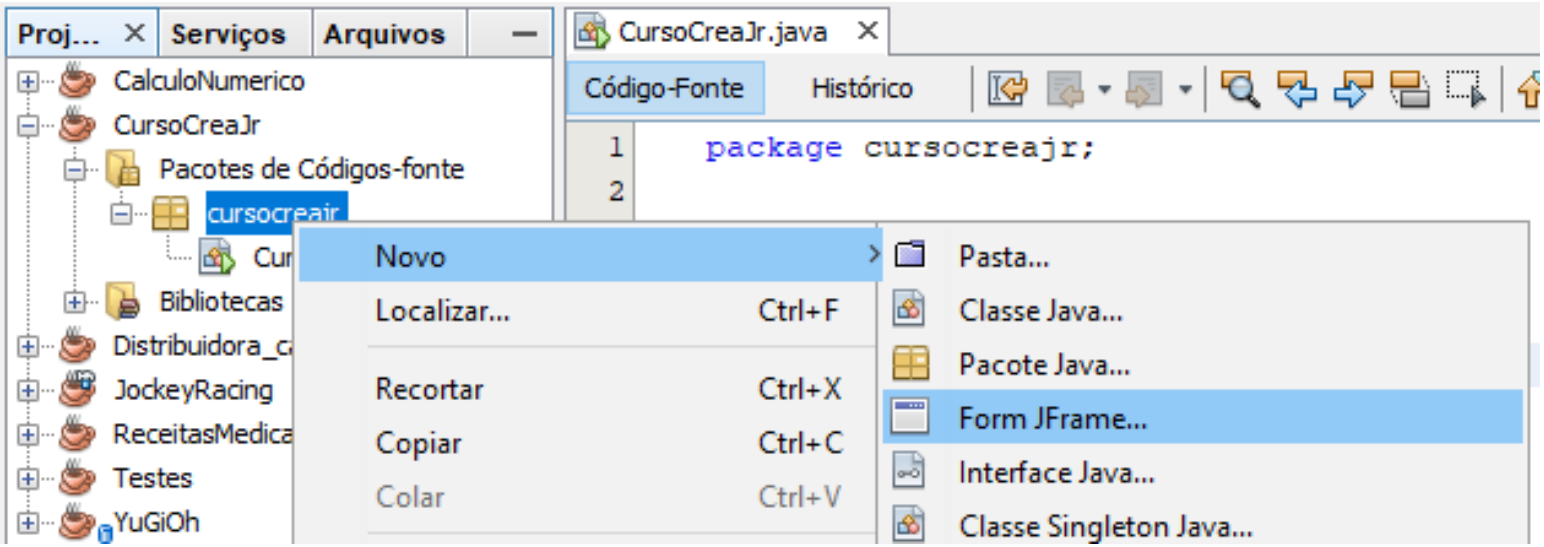
Interface Gráfica com o Usuário

Ementa

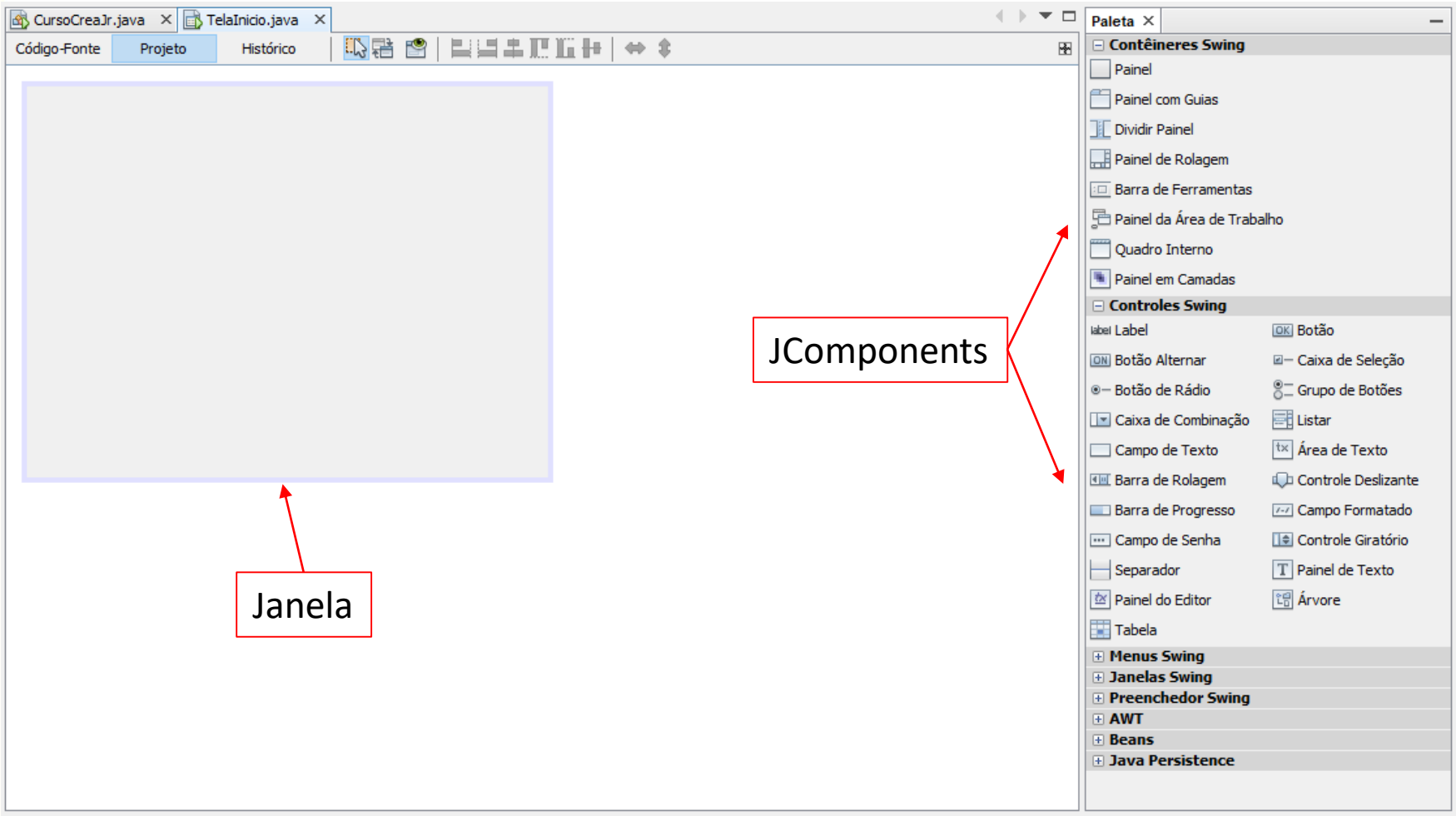
- JFrame
- JComponents
- Eventos

JFrame

Para começarmos a criar uma interface gráfica, primeiro é necessário criar uma classe do tipo *JFrame.*



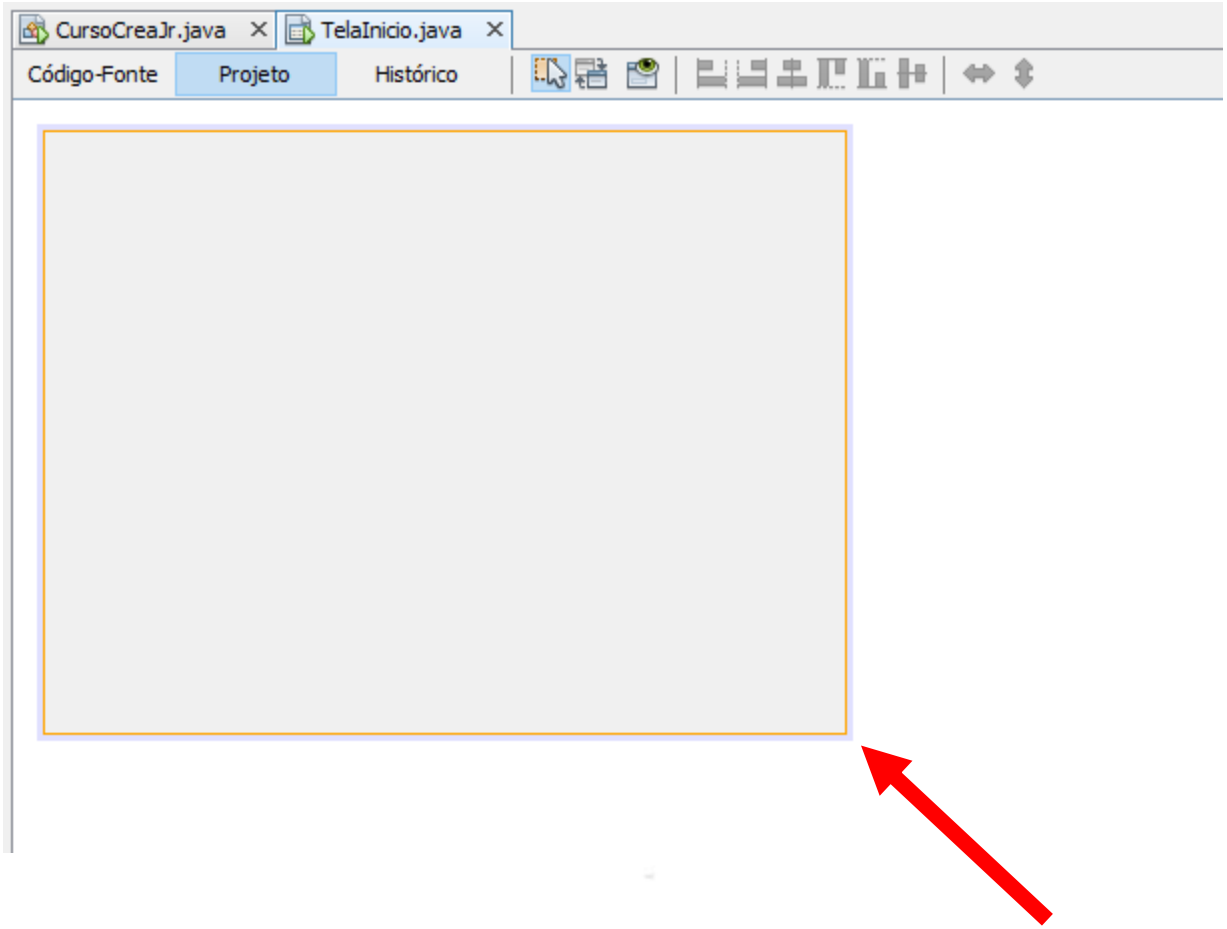
JFrame



JFrame

O quadrado cinza no meio é nossa janela, que por enquanto está vazia. Podemos mudar seu tamanho arrastando a ponta ou clicando duas vezes em cima da lateral.

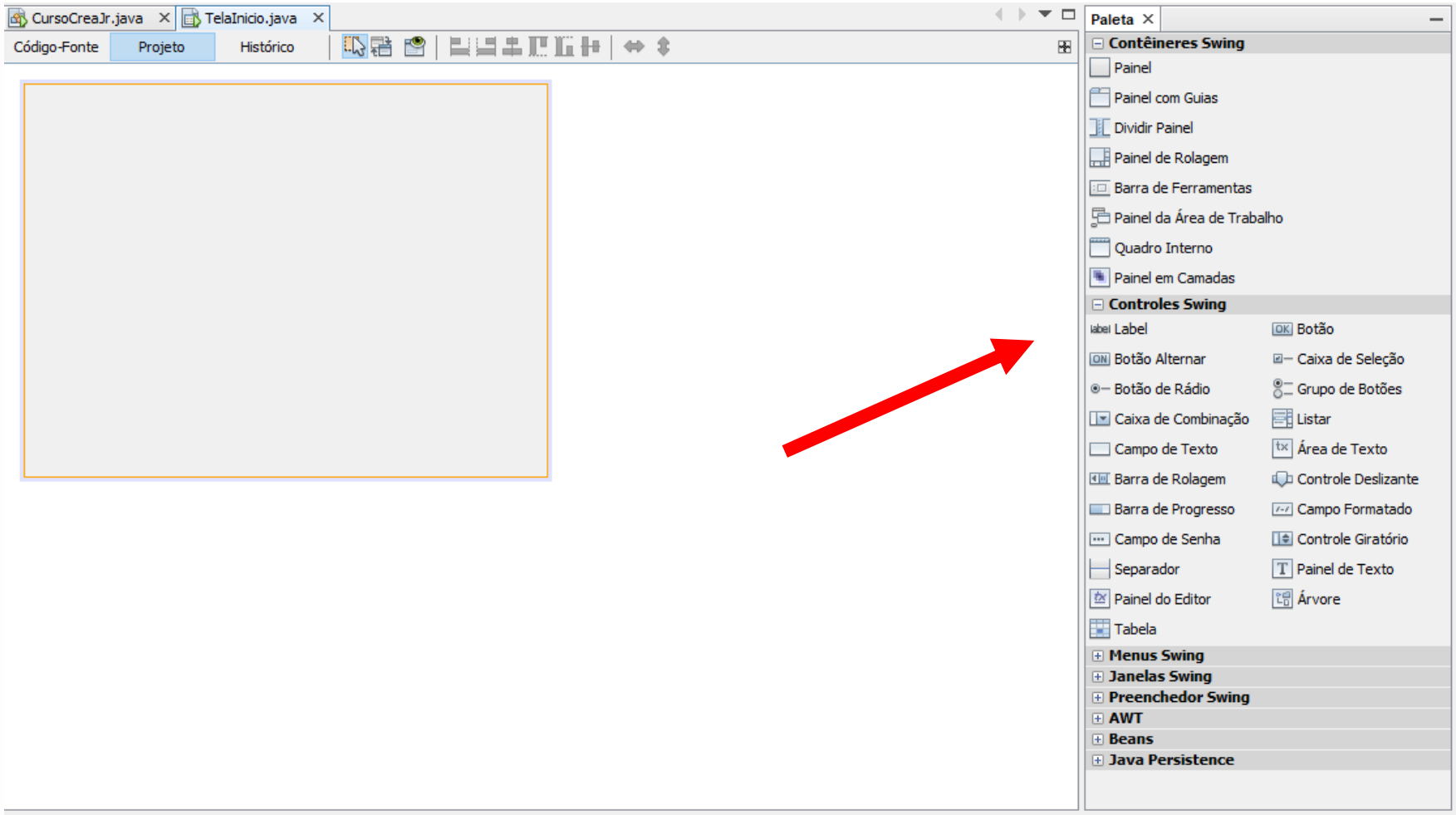
JFrame



JFrame

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

JFrame



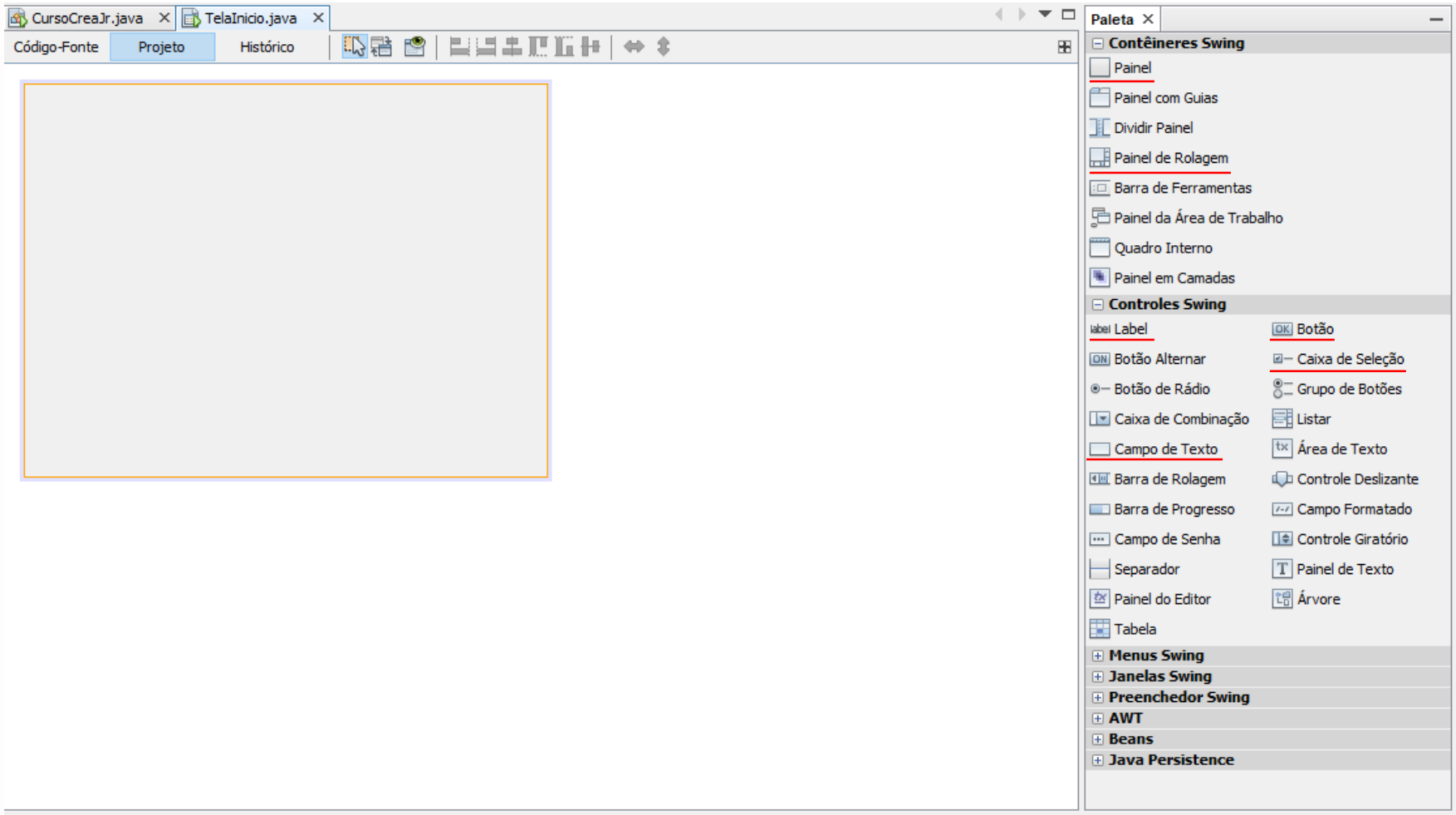
JComponents

Estes componentes são chamados de *JComponents.*

Alguns exemplos muito utilizados:

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

JComponents



Exemplo prático - Steam

Vamos utilizar de todos os conceitos vistos anteriormente juntamente com interface gráfica.

Exemplo prático - Steam

Vamos criar um sistema para gerenciar uma loja de jogos. O software deverá ter a opção criar um usuário e, este usuário poderá cadastrar vários jogos na sua conta. Também será possível visualizar uma Lista com os Jogos já cadastrados.

Exemplo prático - Steam

Baixe a imagem *logo.png* e o projeto *CursoJavaGUI.zip*, nele terá alguns códigos que iremos usar mais adiante.

Link:

Disponível no OneDrive

Exemplo prático - Steam

Primeiro, vamos pensar: quais são as Classes que a nossa aplicação deve ter? Quais objetos vamos precisar usar?

- ✓ Uma classe para armazenar os dados de um usuário
- ✓ Uma classe para armazenar os dados dos Jogos;
- ✓ Uma classe para salvar os dados em arquivo (consistência de dados);

Exemplo prático - Steam

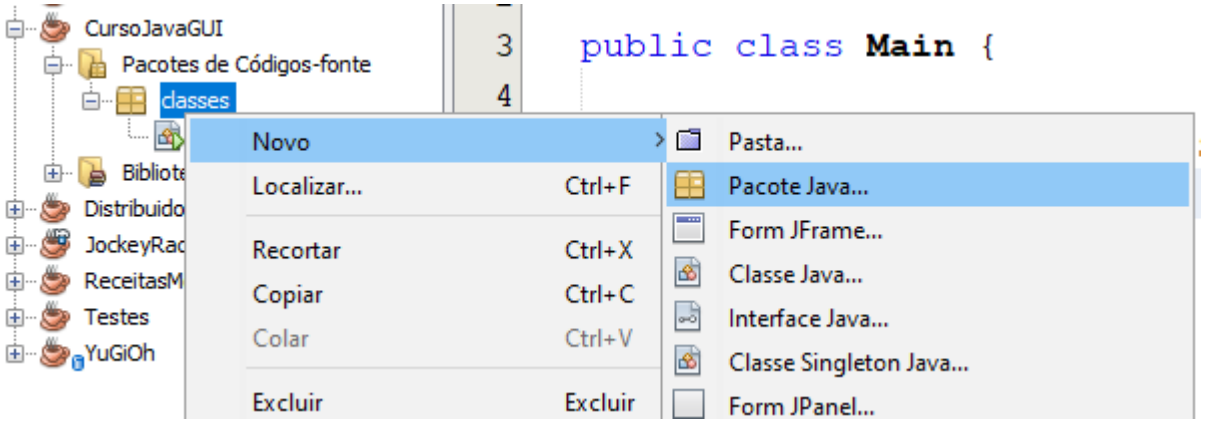
A classe para manipular arquivos já se encontra no projeto.

Ela está com um erro, porém basta criarmos a Classe com o nome “**Jogo**” que o erro desaparecerá.

Criando as Classes

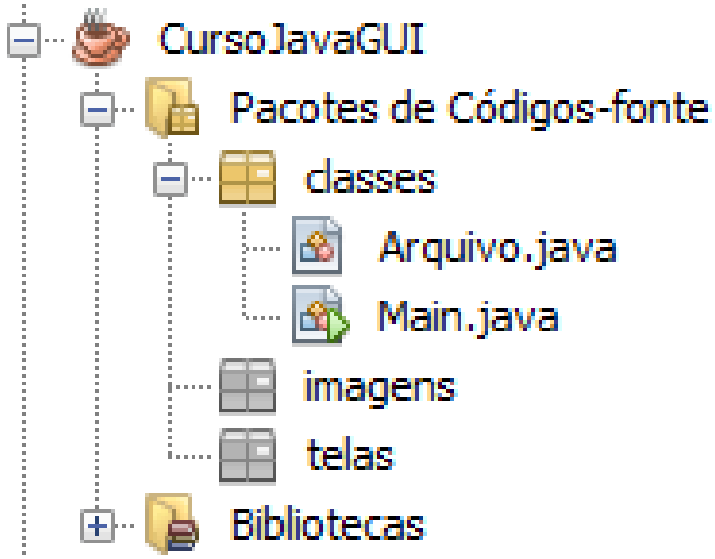
Vamos criar nossas classes!

Um *pacote* nada mais é que uma *pasta* para salvar nossas classes, imagens e afins. É uma boa prática "quebrar" nosso código em vários pacotes para ficar mais organizado nosso projeto.



Criando as Classes

Vamos criar pacotes para salvar as imagens que vamos utilizar no projeto, outro para as classes e um para a interface gráfica



Vamos criar nossas classes de Usuário e Jogo

```
public class Usuario {
    // Atributos
    private String login;
    private String senha;
    private String nomeUsuario;
```

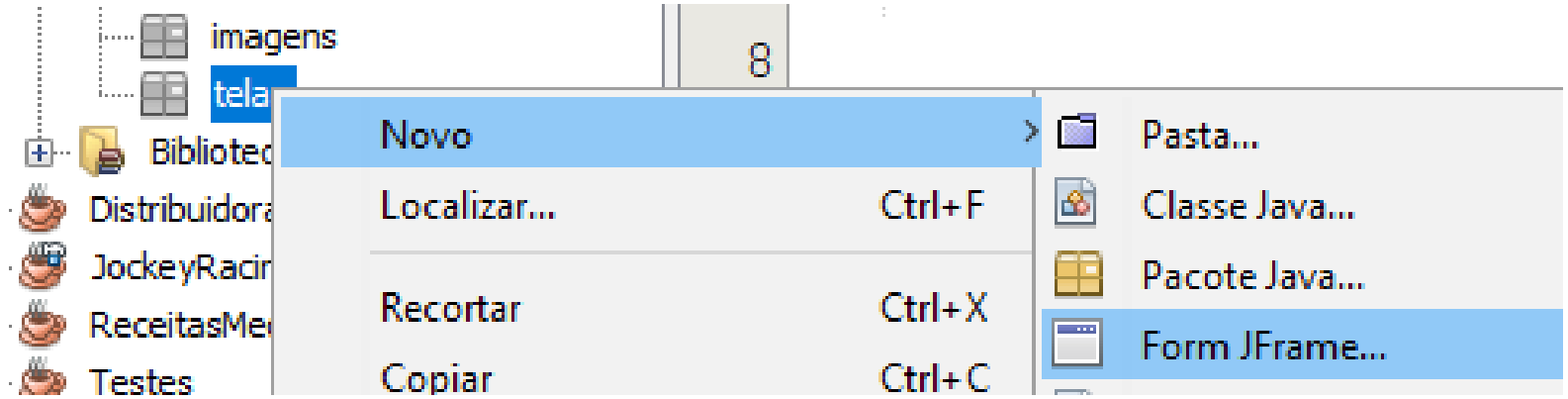
Isto será necessário para salvar os dados no arquivo!

```
public class Jogo implements Serializable{
    // Atributos
    private String nome;
    private String genero;
    private int qtd;
```

Criando as Telas

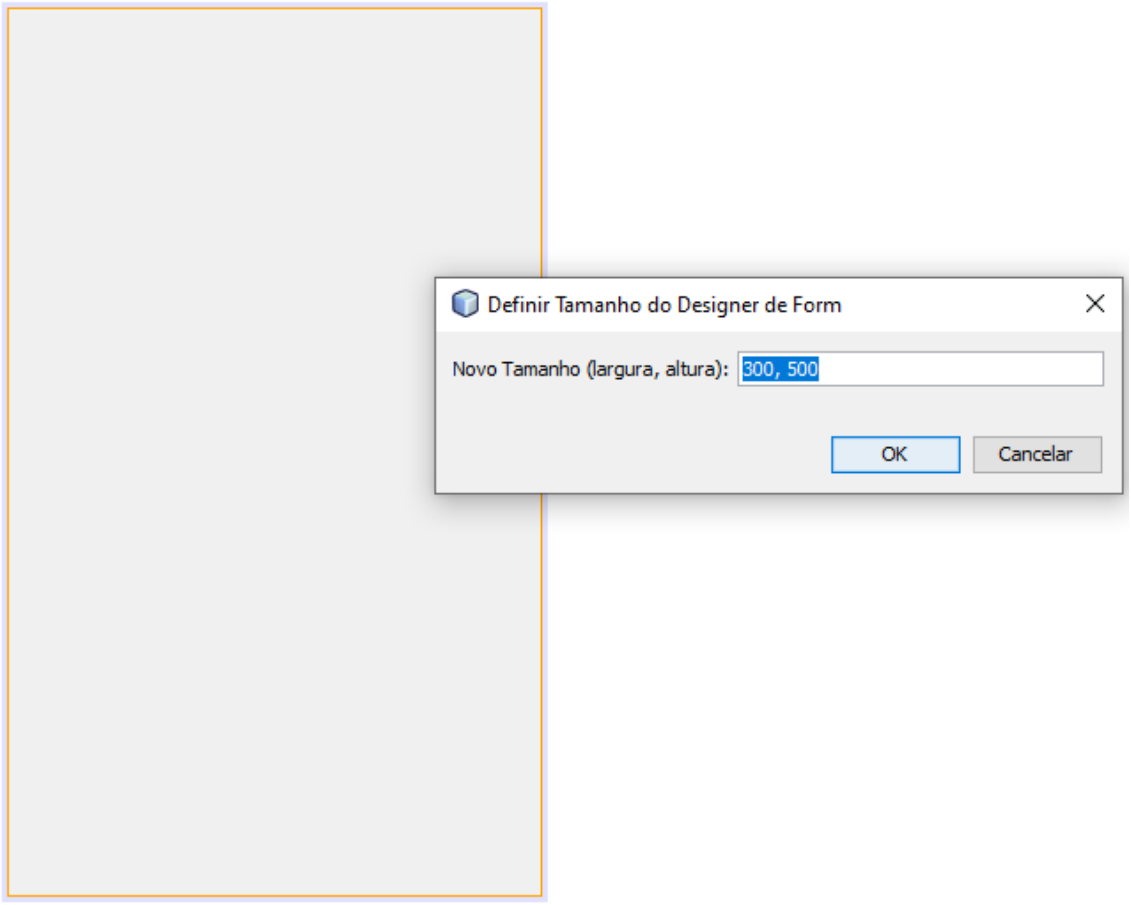
Agora, vamos criar as telas do nosso Software.

Primeiro a tela de *Login*. Vamos cria-la em um pacote somente para telas.



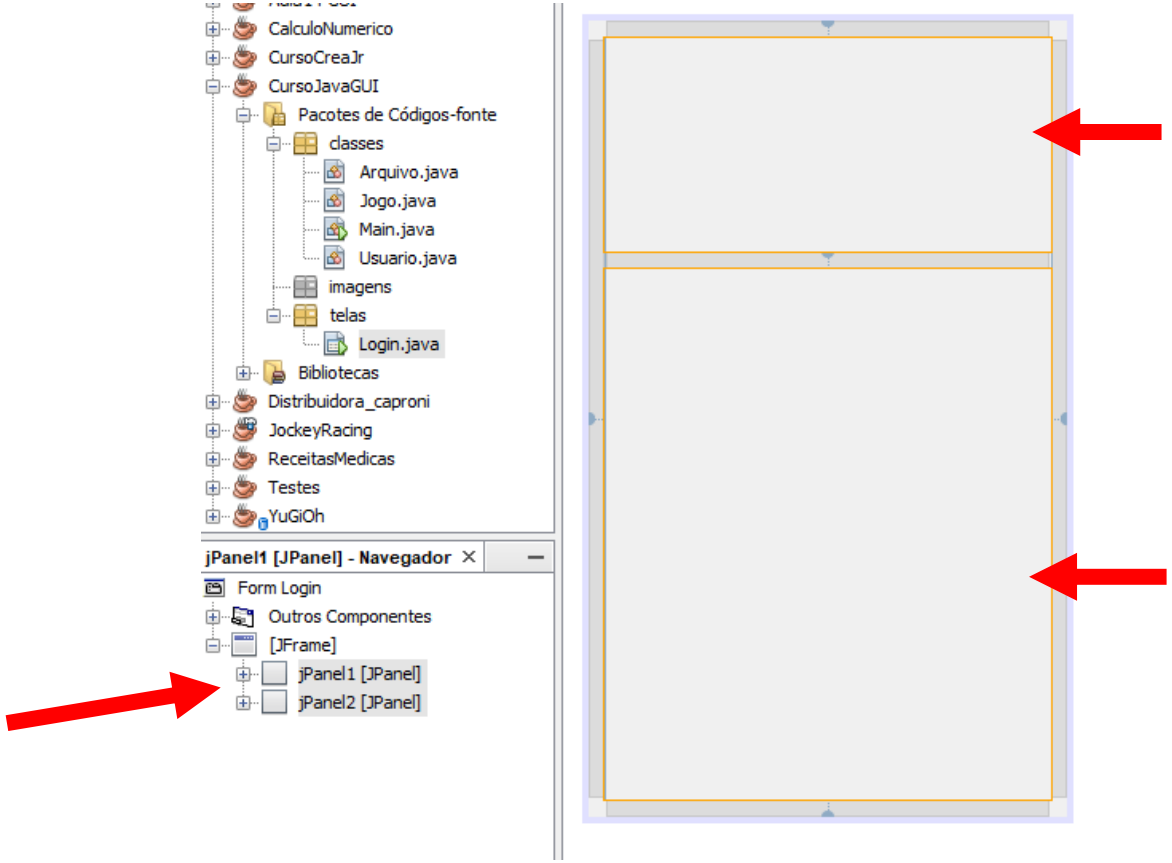
Criando as Telas

Coloque as dimensões de 300 x 500



Criando as Telas

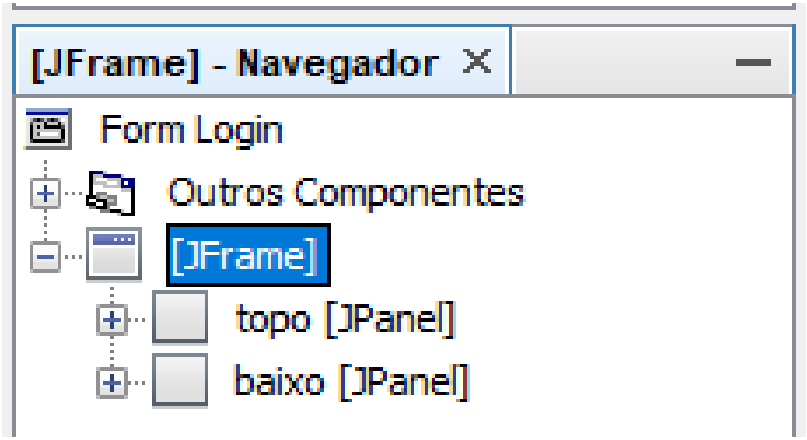
É uma boa prática dividir a nossa tela em *Painéis*. Então, vamos fazer isso!



Criando as Telas

Vamos renomear os Painéis, para que possa ficar mais fácil de se localizar.

Clique com o botão direito -> Alterar nome da variável



Adicionando os Componentes

Agora, vamos adicionar os nossos *JComponents* na tela.

Vamos usar:

JLabel: serve para mostrar algum texto ou imagem, não é possível que o usuário edite seu conteúdo.

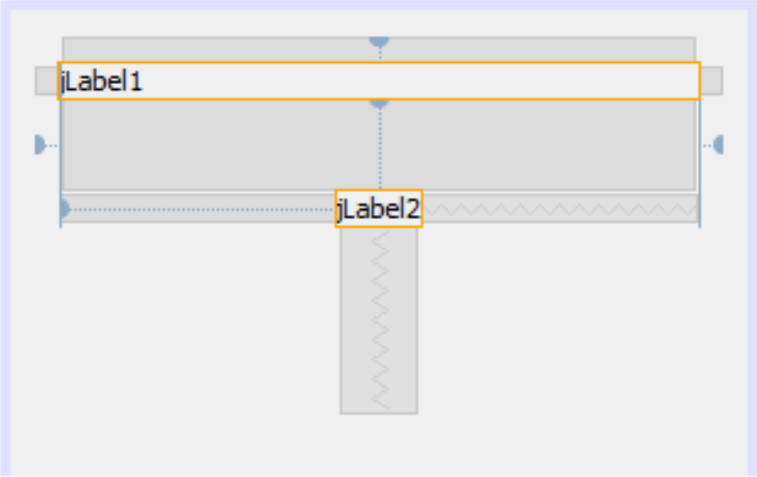
Campo de Texto (JText Field): Serve para entrada de dados, para que o usuário digite algum texto.

Campo de Senha: Tem o mesmo papel do Campo de Texto, porém usado para senhas.

Botão (JBotton): Para verificar os dados e navegar.

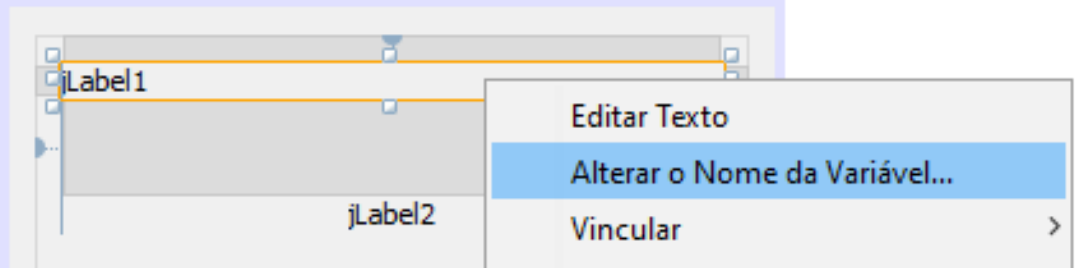
Adicionando os Componentes

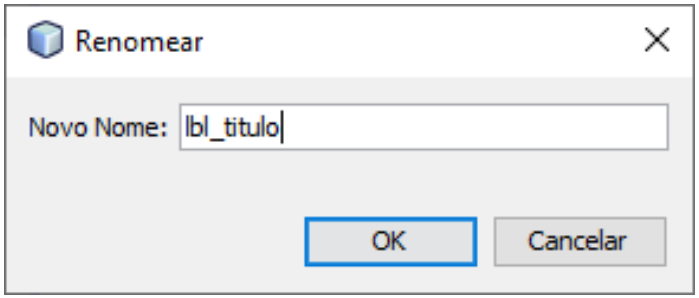
Vamos adicionar um título e um logo na nossa aplicação. Coloque a imagem disponível no pacote *imagens* para evitar erros.



Adicionando os Componentes

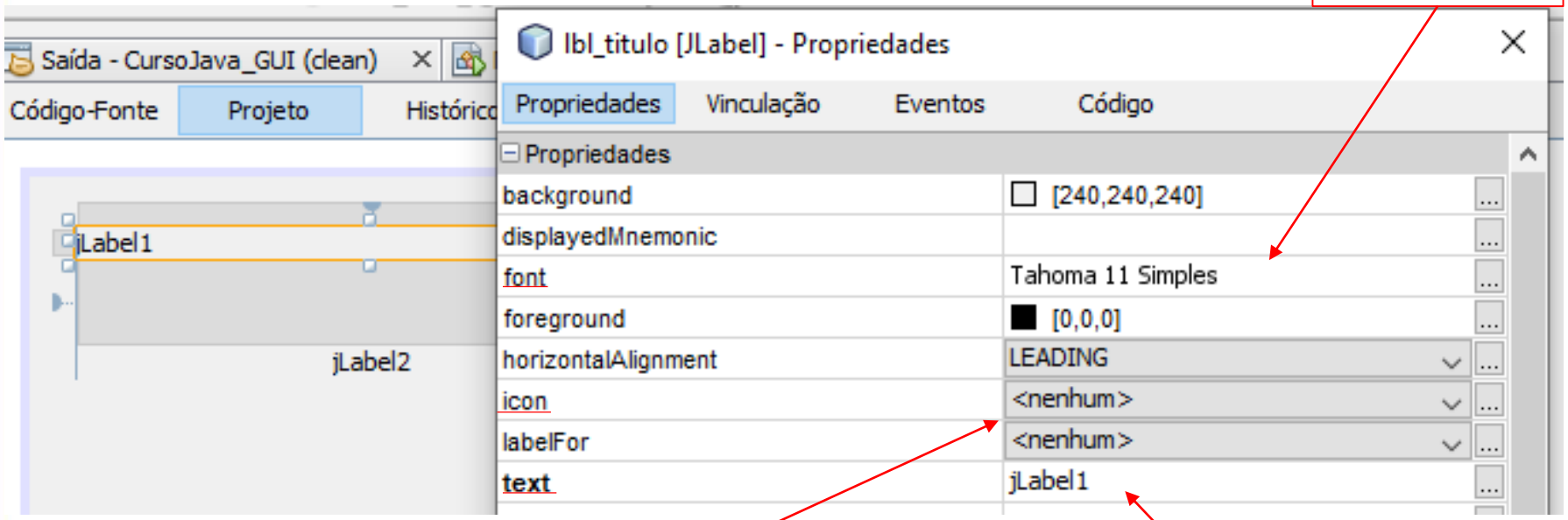
Vamos renomear estas variáveis para deixar nosso código mais intuitivo.





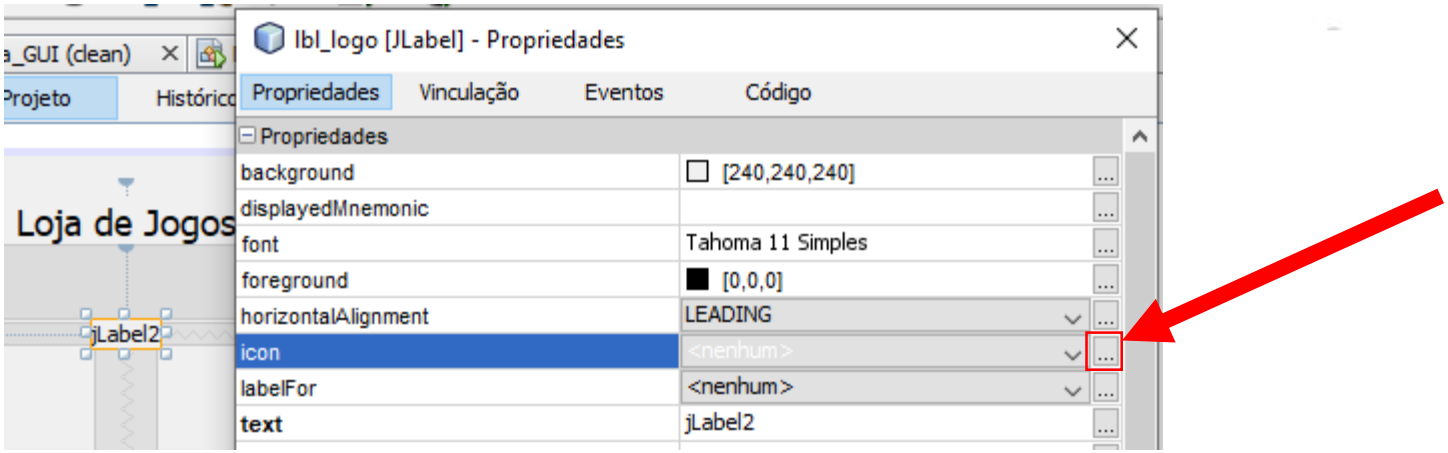
Adicionando os Componentes

Editando o JLabel
Clique com o botão direito -> Propriedades



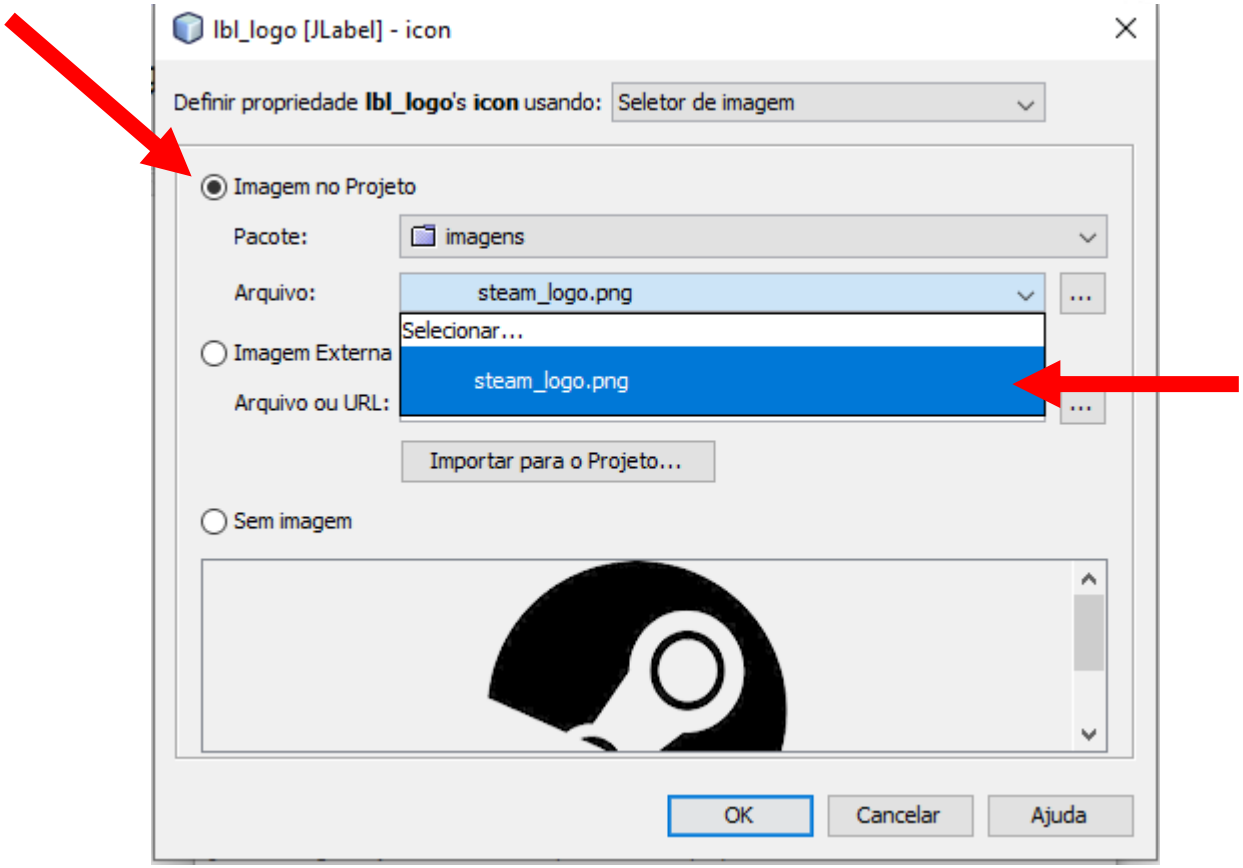
Adicionando os Componentes

Colocando o logo



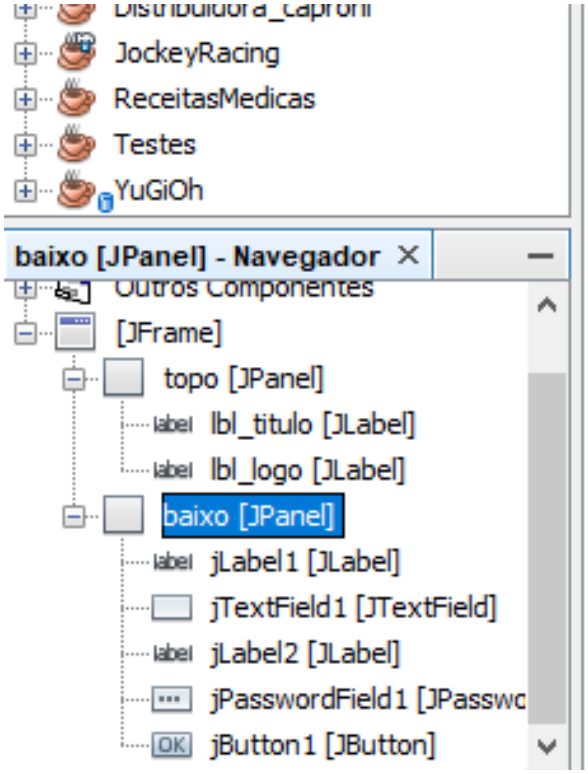
Adicionando os Componentes

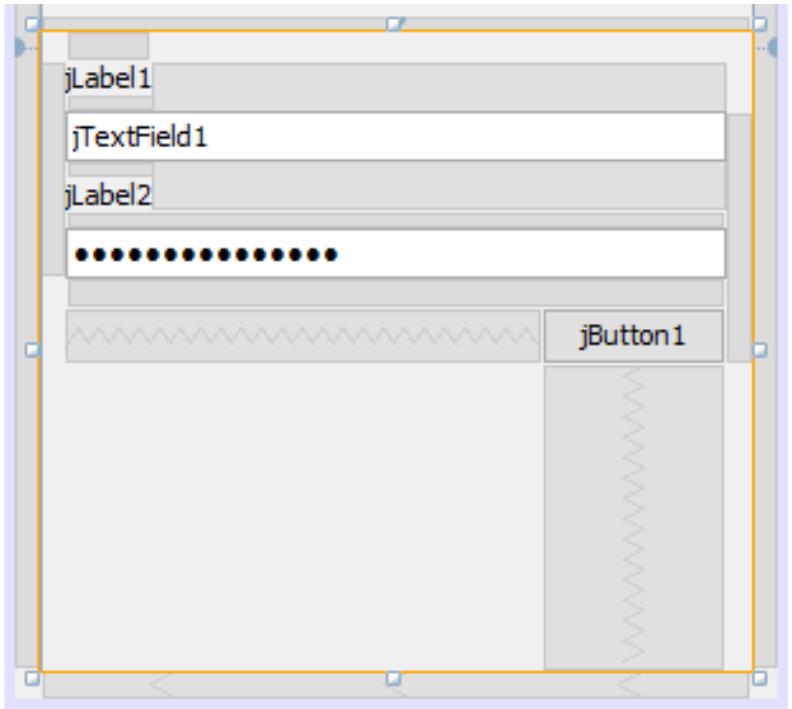
Colocando o logo



Adicionando os Componentes

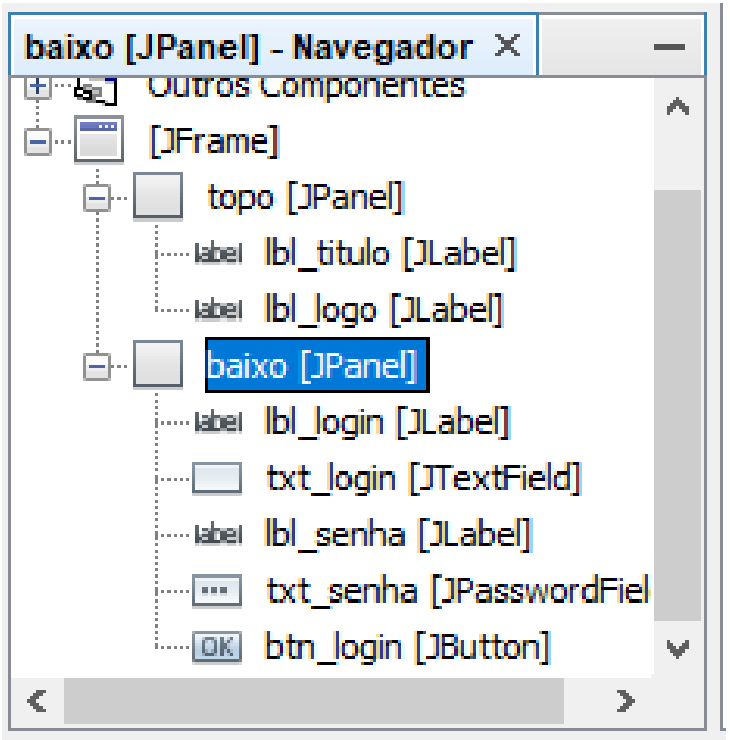
Vamos adicionar as opções para o *Login* do usuário

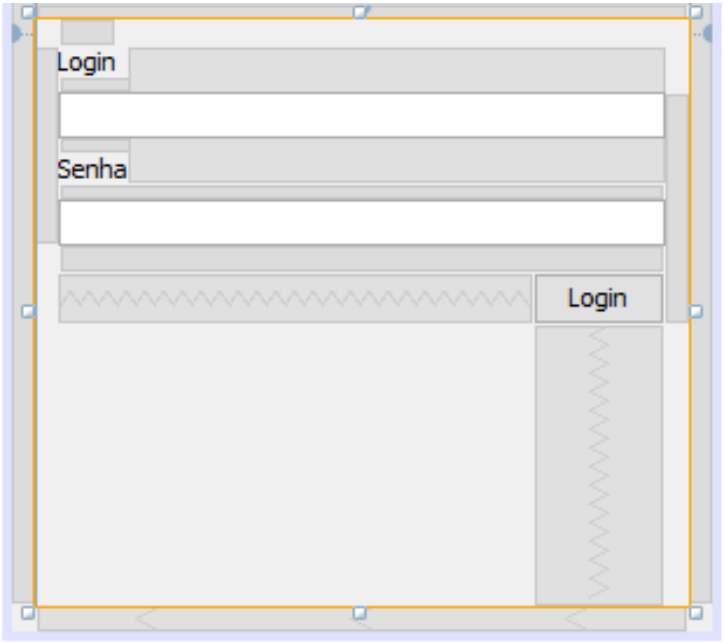




Adicionando os Componentes

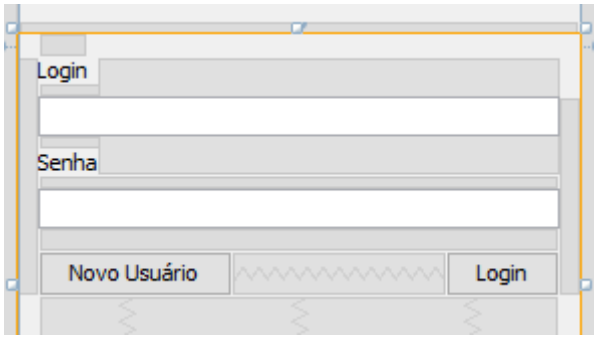
Mude o nome das variáveis para ficar mais intuitivo

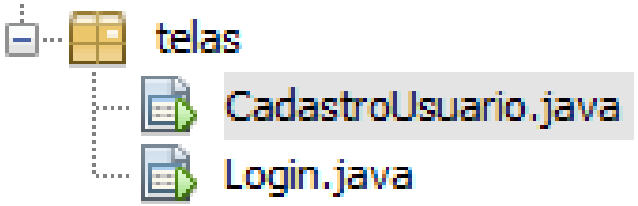




Cadastro de novo usuário

Vamos criar um botão para cadastrar um novo Usuário, que vai redirecionar para uma nova tela.

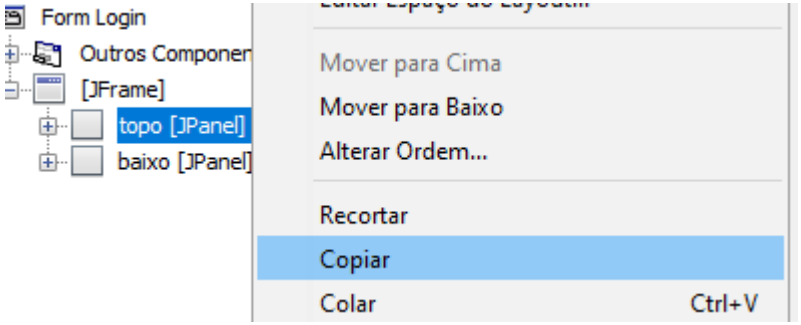


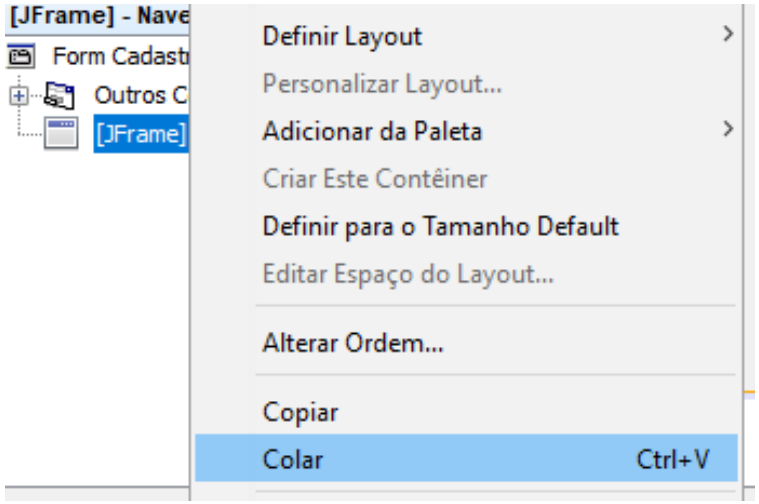


Cadastro de novo usuário

Vamos pedir um nome de usuário e senha somente.

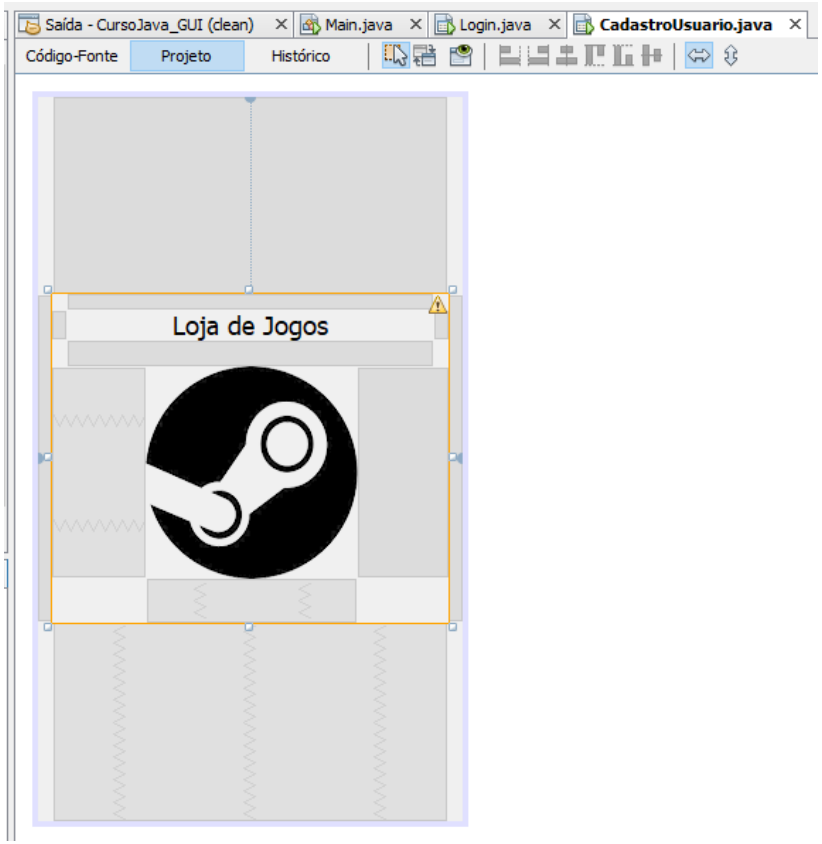
Dica: Podemos copiar nossos painéis e aproveitá-los !





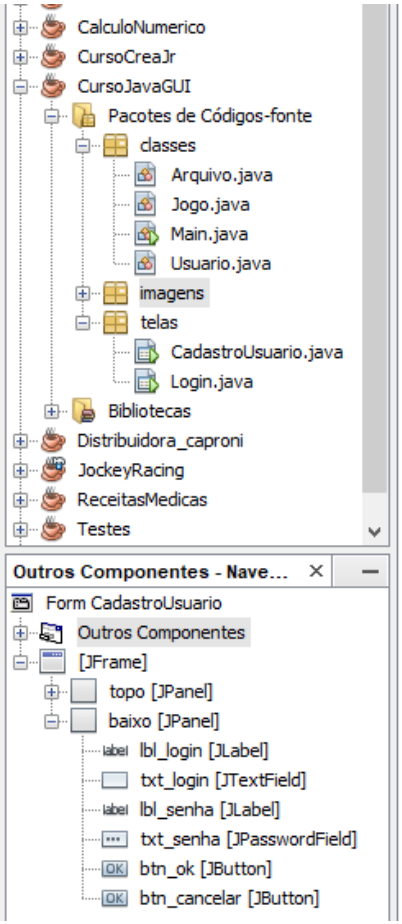
Cadastro de novo usuário

Basta posicioná-lo e editar conforme a nossa necessidade.



Cadastro de novo usuário

Irá ficar mais ou menos assim

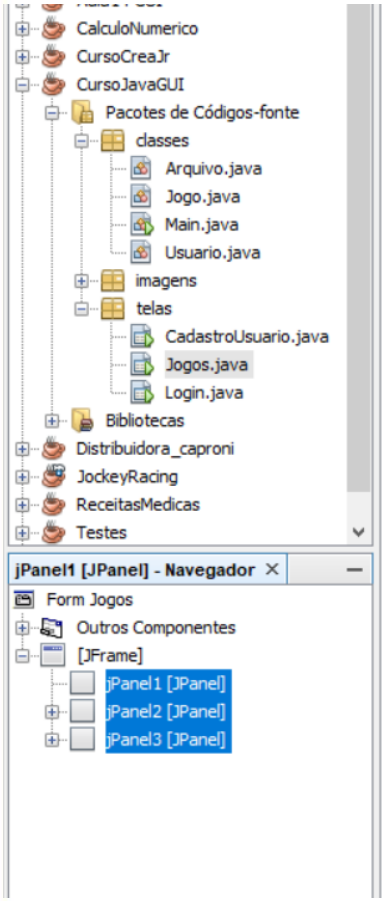


Tela Principal

Vamos criar a tela principal, onde o usuário poderá adicionar e ver os seus jogos salvos.
Coloque as dimensões 875 x 550 na tela.

Vamos separá-la em vários painéis, onde cada um será responsável por realizar uma ação.

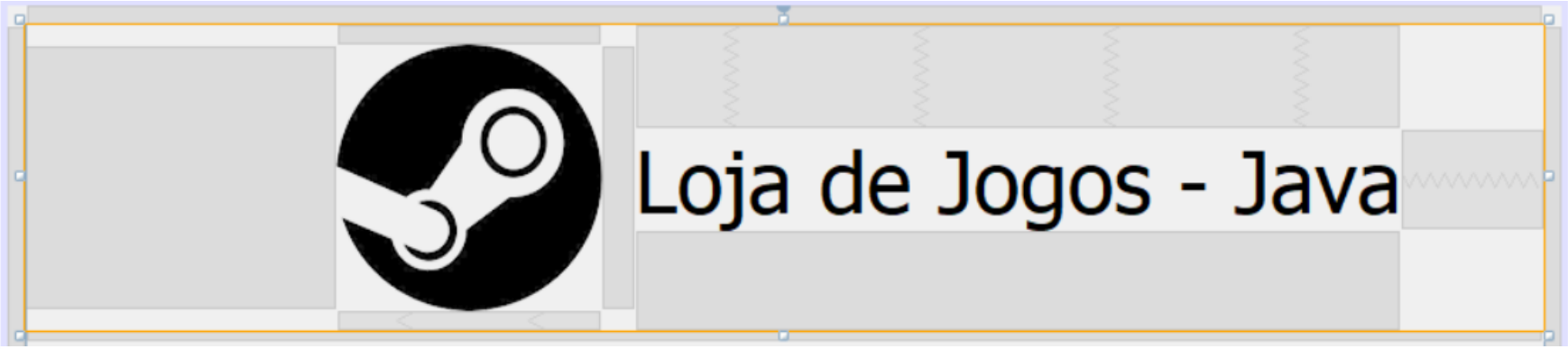
Tela Principal

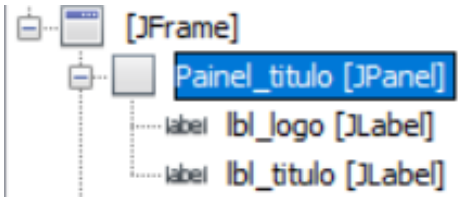


Tela Principal

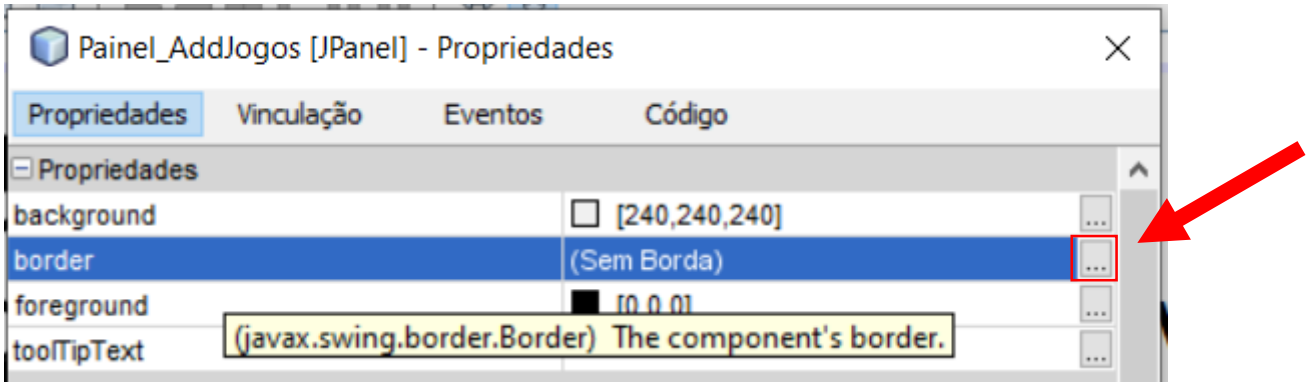
O Primeiro painel deverá ter as dimensões de 855 x 172 e nele irá conter o logo e um título.

O JLabel do Logo deverá ter as dimensões de 150 x 150 e o título com a fonte tamanho 48.

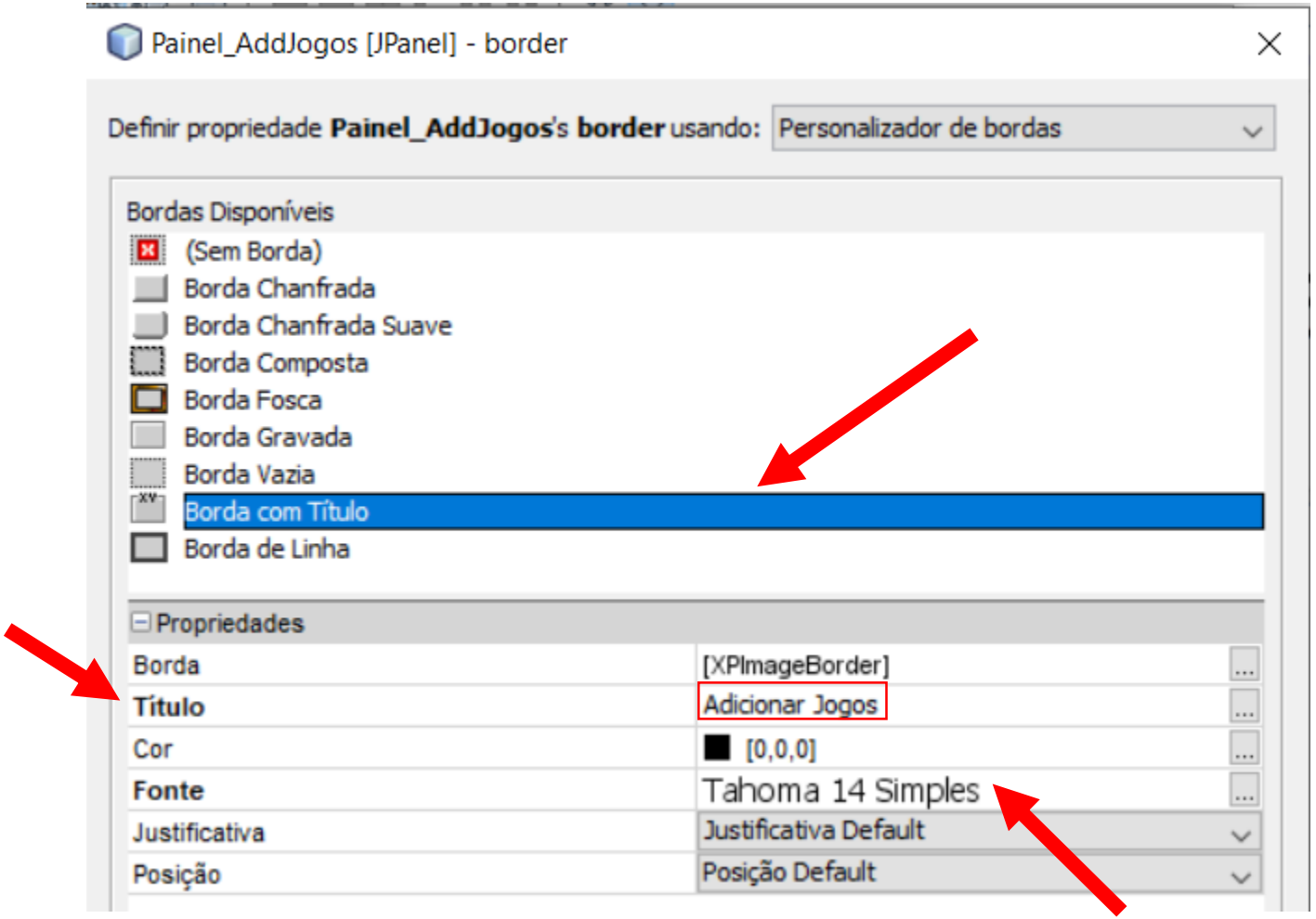




O Painel a esquerda será o nosso painel de cadastro de jogos. Vamos adicionar uma borda com título e os atributos dos jogos

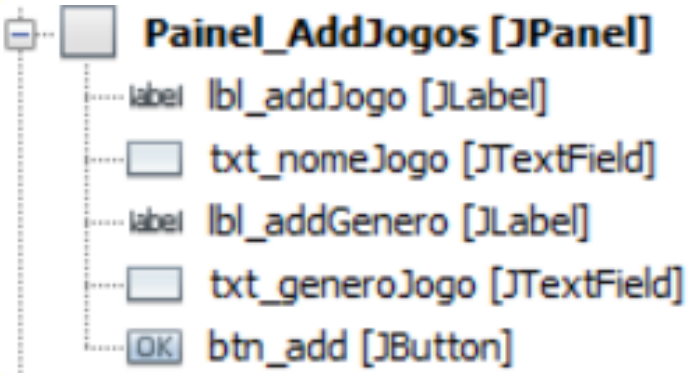


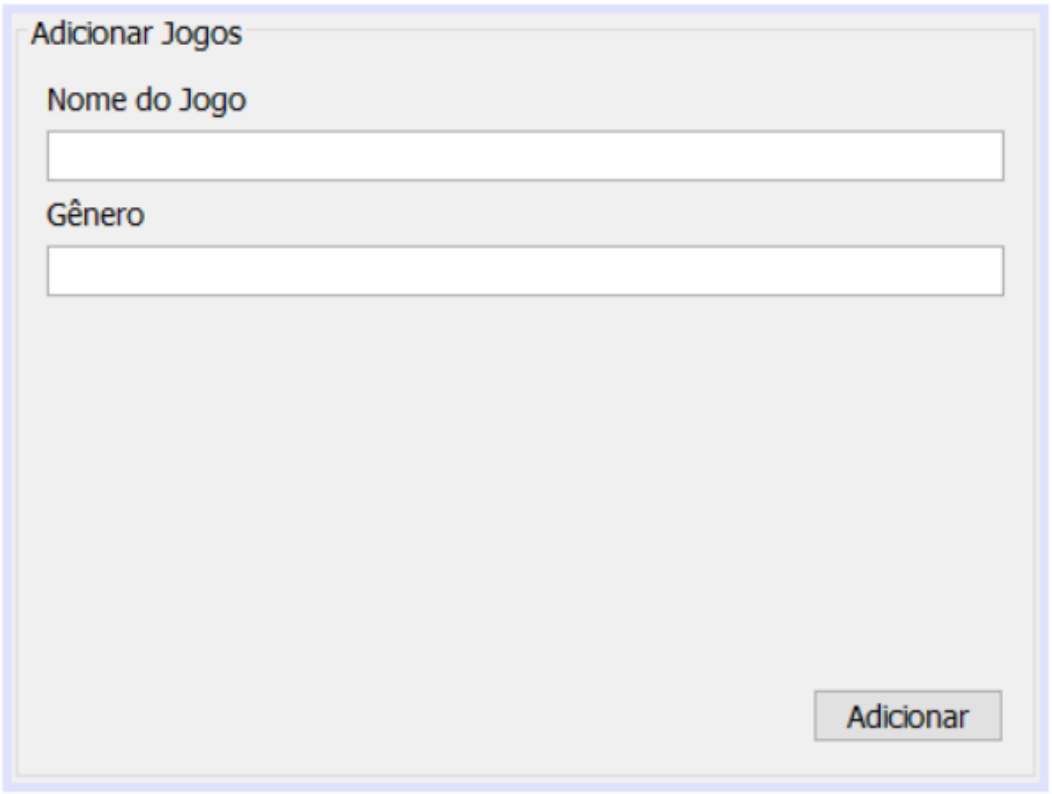
Tela Principal



Tela Principal

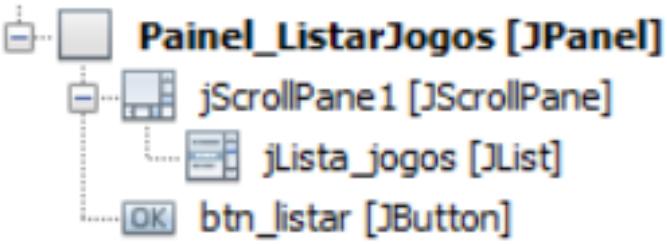
Vamos colocar os campos para o nosso usuário adicionar os jogos.





Tela Principal

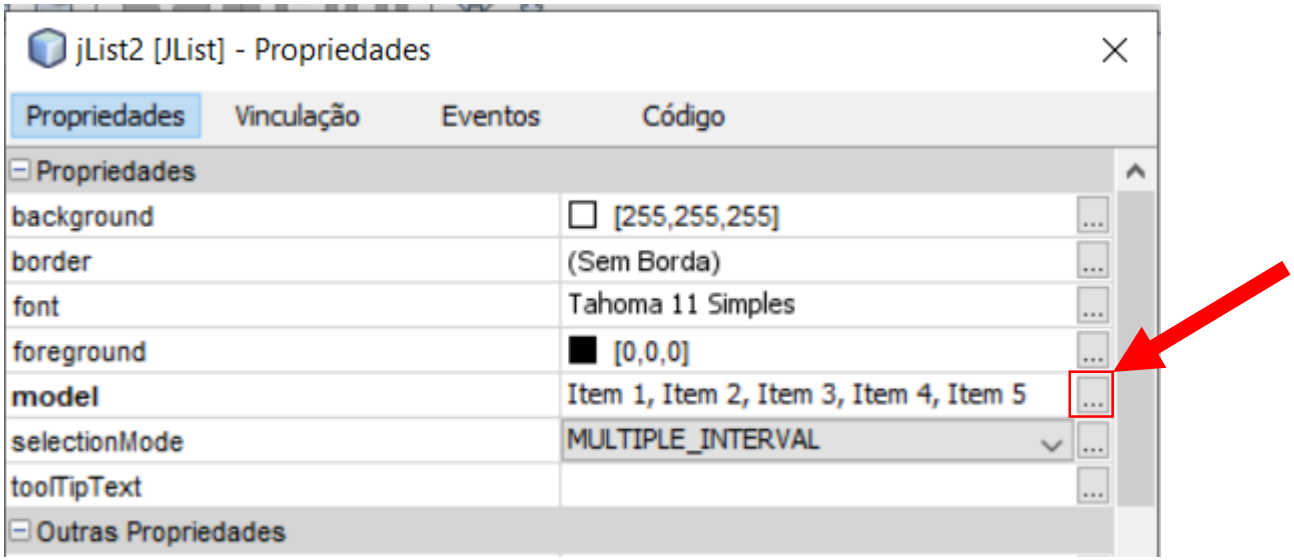
O painel a direita irá exibir uma lista dos Jogos. Vamos fazer algo parecido ao painel anterior. Vamos utilizar um *JList* e um botão.



Tela Principal

Observação: Remova antes os elementos da Lista que vem por padrão.

Clique com o botão direito -> Propriedades -> Model



Após clicar no botão, apague todos os Itens e clique em Ok

Tela Principal

Nossa tela de jogos irá ficar mais ou menos assim

Back-end

Quase tudo pronto!

Agora falta a comunicação entre as telas e a lógica do nosso Software.

Back-end

Primeiro vamos configurar os botões da tela de Login.

Dê um duplu clique em cima do botão para abrir o seu método de clique.

```
private void btn_loginActionPerformed(java.awt.event.ActionEvent evt) {
    // TODO add your handling code here:
}
```

Aqui iremos programar o que vai acontecer quando clicarmos no botão. É uma boa prática chamar somente métodos!

Back-end

O Código fica bem mais intuitivo e organizado, como podemos ver no exemplo abaixo

```
private void btn_loginActionPerformed(java.awt.event.ActionEvent evt) {
    // TODO add your handling code here:
    // Clique do botão de Login
    verificarDadosInseridos();
}
```

Back-end

Neste método iremos verificar se os dados estão inseridos corretamente

```
private void verificarDadosInseridos() {

    arquivo.lerArquivoUsuarios();

    String valorLogin = txt_login.getText();
    String valorSenha = txt_senha.getText();

    boolean aprovado = arquivo.verificarLogin(valorLogin, valorSenha);

    if (aprovado) {
        chamarTelaJogos();
    } else {
        mensagemErro();
    }
}
```

Back-end

Para chamar uma nova Tela criamos uma instância da tela e chamamos o método *setVisible* e passamos *verdadeiro* como parâmentro. Também devemos chamar o método *dispose* para "matar" a nossa tela atual.

```
private void chamarTelaJogos(){
    Jogos telaJogos = new Jogos();
    telaJogos.setVisible(true);
    dispose();
}
```

Back-end

Vamos configurar a tela de cadastro de usuários

```
private void btn_okActionPerformed(java.awt.event.ActionEvent evt) {
    // TODO add your handling code here:
    // Botão criar usuário
    criarNovoUsuario();
}

private void btn_cancelarActionPerformed(java.awt.event.ActionEvent evt) {
    // TODO add your handling code here:
    // Botão cancelar
    retornar();
}
```

Back-end

Vamos configurar a tela de cadastro de usuários

```
private void criarNovoUsuario() {
    String login = txt_login.getText();
    String senha = txt_senha.getText();

    arquivo.usuarioCriar(login, senha);

    JOptionPane.showMessageDialog(rootPane, "Sucesso!");
    chamarTelaLogin();
}
```

```
private void chamarTelaLogin(){
    new Login().setVisible(true);
    dispose();
}
```

```
private void retornar() {
    new Login().setVisible(true);
    dispose();
}
```

Back-end

Tudo certo!

Vamos desenvolver a lógica de cadastrar um novo jogo, salvá-lo e mostrar na lista.

Back-end

Editando o evento de clique do mouse no botão adicionar

```
private void btn_addActionPerformed(java.awt.event.ActionEvent evt) {
    // TODO add your handling code here:
    // Clicar no botão Adicionar
    salvarJogo();
}
```

Back-end

```
private void salvarJogo() {
    String nomeJogo = txt_nomeJogo.getText();
    String generoJogo = txt_generoJogo.getText();

    Jogo jogo = new Jogo();
    jogo.setNome(nomeJogo);
    jogo.setGenero(generoJogo);

    games.add(jogo);
    arquivo.jogoSalvar(games);

    limparCampos();
}
```

```
private void limparCampos() {
    txt_generoJogo.setText("");
    txt_nomeJogo.setText("");
}
```

Back-end

E por fim listar todos os Jogos já cadastrados

```
private void exibirJogos() {
    ArrayList<Jogo> leitura = arquivo.jogoLer();

    String[] meusJogos = new String[leitura.size()];

    for (int i = 0; i < meusJogos.length; i++) {
        meusJogos[i] = leitura.get(i).getNome();
    }

    jLista_jogos.setListData(meusJogos);
}
```

Back-end

Finalizamos a nossa Loja! Agora só falta mostrar as telas.

Para isso, no método *Main*, chame a Classe Login e pronto!

```
public class Main {

    public static void main(String[] args) {
        Login l = new Login();
        l.setVisible(true);
    }
}
```

Projeto Finalizado

Adicionando mais Recursos

Mas ficou faltando os atributos de Gênero do jogo e Quantidade.

Como implementar?

Desafios

Agora é com você !

Use da sua criatividade e lógica de programação para explorar ainda mais recursos desse nosso projeto.

Desafio: Através de uma Thread, mostre em um *JLabel* o gênero do jogo selecionado na lista.

Desafios

Agora use da sua criatividade!

Interface Gráfica você só aprende fuçando e se desafiando a fazer novas funções.

Desafio: Crie sua própria aplicação para cadastrar e listar alguma coisa. (*Não é necessário salvar em arquivo, mas caso queira, pode adaptar a Classe Arquivo do nosso Projeto*).

Obrigado!


samuel.souza@gec.inatel.br


https://github.com/Saam97