LINGUAGEM
GROOVY

INSTITUTO FEDERAL DE CIÊNCIAS
E TECNOLOGIA DE SÃO PAULO,
CÂMPUS CUBATÃO

LINGUAGEM

GROOVY

AUTORES

BEATRIZ FADEL MARTINS DA SILVA
FERNANDA DE CARVALHO ROCHA
FERNANDA PINTO DOS SANTOS
GABRIELE AMOROSO SANTOS
INGRID CAMPOS SILVA
JOÃO PEDRO CIVITA DO NASCIMENTO
YASMIN PERES DE OLIVEIRA SILVA

ORIENTADOR

MAURICIO NEVES ASENJO

GROOVY

ÍNDICE

1 CAPÍTULO

HISTÓRIA

A linguagem de programação Groovy originou-se do sonho e da necessidade de **James Strachan** em possuir uma linguagem dinâmica, compilada em Java e com a produtividade de Ruby e Python. Com essas ideias, Strachan, em 29 de agosto de 2003, publicou em seu blog suas primeiras ideias e concepções para a criação dessa nova linguagem. Após a publicação no blog, James Strachan se uniu ao Bob McWhirter e com o tempo formou um grupo de pessoas para o desenvolvimento da linguagem de programação Groovy.

No ano de 2004, ocorreu a fundação do GroovyOne e a entrada de mais outros desenvolvedores. Além disso, foi criada a Groovy Language Specification (Especificação de linguagem Groovy) GLS, e o kit para testes de compatibilidade (TCK).

Durante o ano de 2004 a 2006, diversas versões foram lançadas, entretanto, somente em janeiro de 2007, com processo de padronização do Java Community Process(JCP), realizou-se a adoção da versão "1.0". Em 7 de Dezembro de 2007, a versão 1.1, considerada a final, foi lançada. No entanto, no final do mesmo ano ela virou a versão 1.5, devido aos diversos erros que existiam na antiga versão.

James Strachan

CARACTERÍSTICAS

O Groovy é uma linguagem orientada a objeto, dinâmica, espelhada em Java e com aspectos semelhantes a grandes linguagens de programação como Python, Ruby, Perl e Smalltalk. Devido a isso, o Groovy permite rodar o código feito em Java e modificar o código da aplicação em tempo de execução, como na linguagem Ruby.

Atualmente, o Groovy se tornou uma linguagem que pode ser compilada pelo *groovyc* ou até mesmo interpretada como um *script*. Ela é caracterizada por ser uma linguagem ágil e dinâmica que roda sobre a JVM (Java Virtual Machine), classificando-a como uma linguagem multiplataforma, ou seja, pode ser executada em várias plataformas, como o Windows, Linux, Mac entre outras.

IDE'S

As IDES são softwares que auxiliam no desenvolvimento de aplicações, pois combinam ferramentas comuns de desenvolvimento em uma única interface gráfica do usuário. Existem diversas IDEs que podemos utilizar para programar em Groovy, as mais populares, entre elas, são:

TextMate: É um editor exclusivo para MacOS, que usa base UNIX para reduzir a sobrecarga e a necessidade de trabalho manual, além de vir com diversos recursos, incluindo destaque de sintaxe, blocos de código dobráveis, macros graváveis, etc.

NetBeans: É um ambiente de desenvolvimento integrado de código aberto, que oferece ótimas ferramentas de desenvolvimento. Pode ser executado em diversas plataformas como Linux, Windows, etc.

UltraEdit: É considerado um dos melhores editores de texto e existe há mais de 25 anos. Possui temas personalizáveis, além de ter recursos de localização, substituição e suporte para arquivos muito grandes.

IDE'S

Intellii IDEA: Uma IDE com diversos recursos que incluem auto complemento de código inteligente, injeção de linguagem, um ambiente centrado no editor e muitas ferramentas de construção úteis. Ele também oferece recursos específicos para o Groovy.

GroovyEclipse: O Groovy fornece suporte de ferramentas Eclipse para desenvolvedores. O código-fonte GroovyEclipse é um conjunto de projetos de plug-in do Eclipse. Cada projeto contribui com várias lógicas de ferramentas do Groovy para o Eclipse com diferentes pontos de extensão.

NETBEANS

INTELLI IDEA

TEXTMADE

ULTRAEDIT

GROOVY ECLIPSE

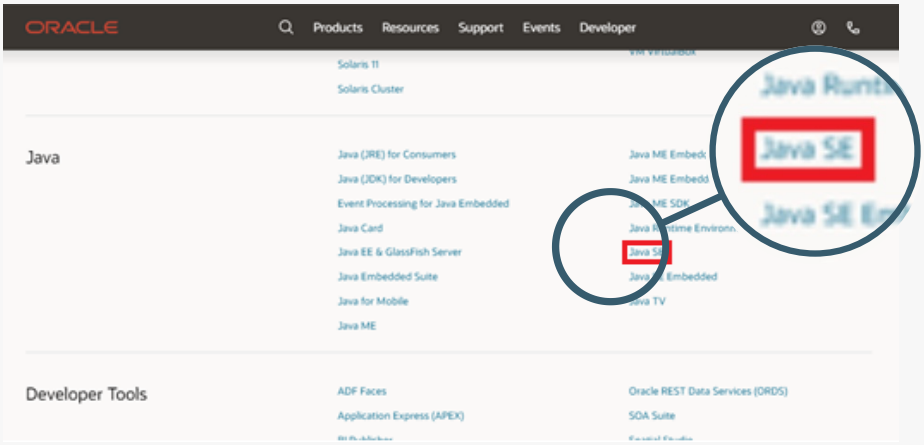
ECLIPSE E NETBEANS

A escolha das IDEs que serão usadas ao longo do trabalho é umas das decisões mais importantes, pois um dos principais objetivos das IDEs é facilitar o trabalho dos programadores. Após analisar todas as opções, o grupo escolheu para a realização do nosso projeto o Eclipse e o Netbeans; vários motivos foram levantados: um deles é que essas plataformas são consideradas uma das melhores IDEs existentes. Elas são IDEs completas e oferecem excelentes ferramentas e opções para o desenvolvimento da linguagem que escolhemos, Groovy, além disso, o grupo já possui afinidade com o Netbeans.

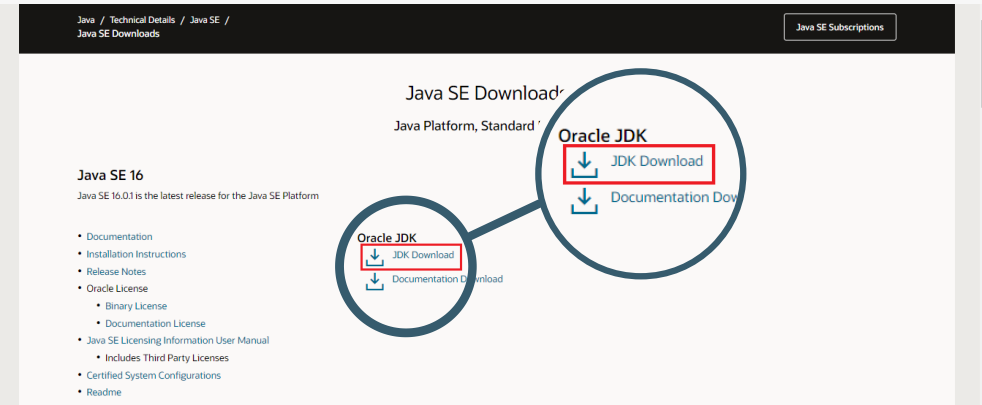
INSTALAÇÃO

Para realizar a instalação da linguagem Groovy e do seu respectivo ambiente de desenvolvimento, demanda-se a presença do **Java Development Kit (JDK)** na máquina onde se deseja trabalhar. Trata-se de um conjunto de ferramentas que possibilitam o desenvolvimento de aplicações na plataforma Java. Para realizar a instalação do JDK, deve-se:

1. Acessar a página de downloads da Oracle pelo endereço **'www.oracle.com/downloads/'** e procurar pela opção Java Standard Edition (Java SE);

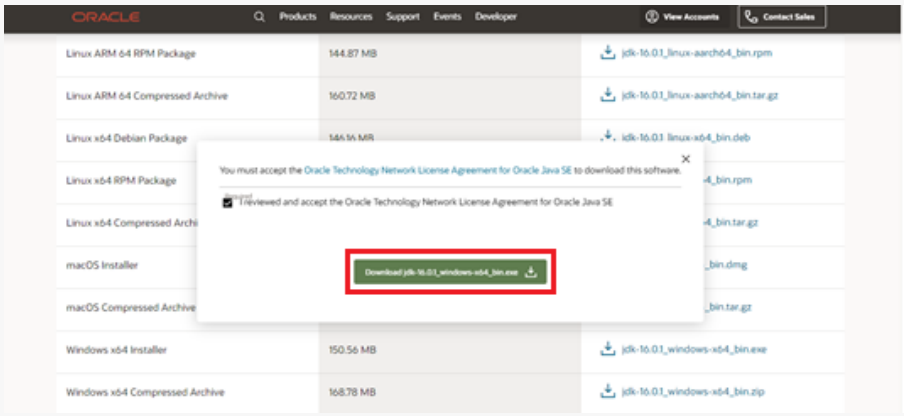


2. Após isso, esta tela abrirá, selecione a opção **"JDK Download"**



INSTALAÇÃO

3 Quando selecionado a opção Java SE, escolha o arquivo compatível com a sua máquina, e faça o download.

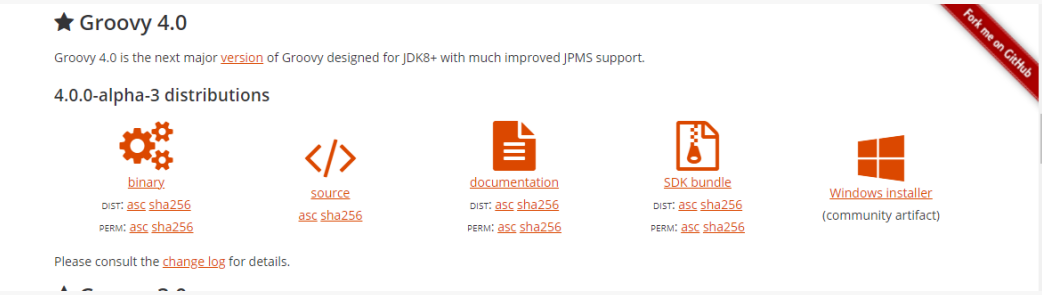


4 Após baixar o arquivo referente à arquitetura e ao sistema operacional da sua máquina, execute-o e siga os passos indicados pelo instalador;

Ao finalizar este processo, a instalação da linguagem Groovy poderá ser iniciada:

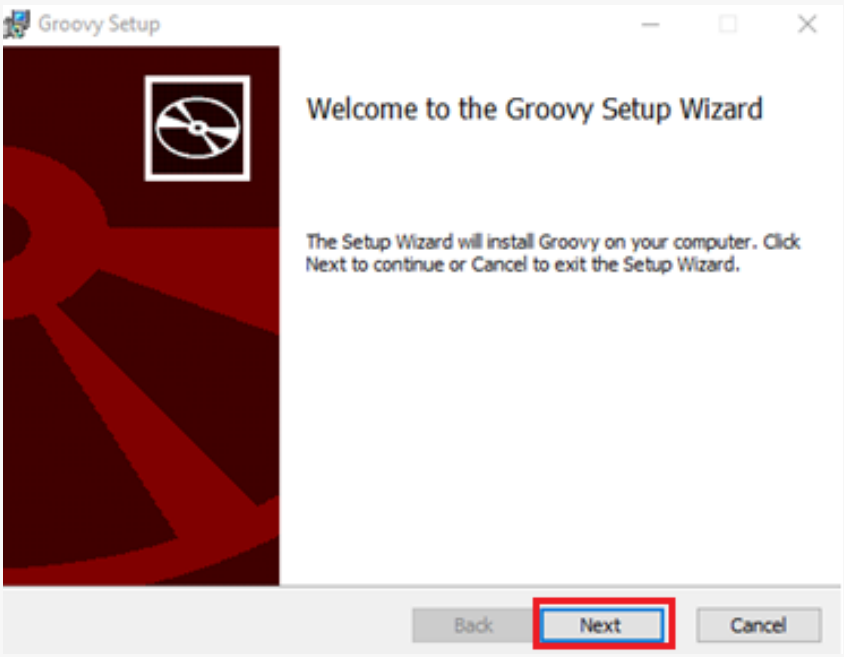
INSTALAÇÃO

5 Inicialmente, deve-se realizar o download do instalador pela página de download do Apache Groovy disponível no endereço‘**groovy.apache.org/download**’;



Convém ressaltar que se optou pela versão mais recente da linguagem - o Groovy 4.0. Ademais, neste tutorial utilizou-se o instalador do sistema operacional Microsoft Windows. Para os demais, a própria página do Apache Groovy indica os passos necessários para instalar a linguagem.

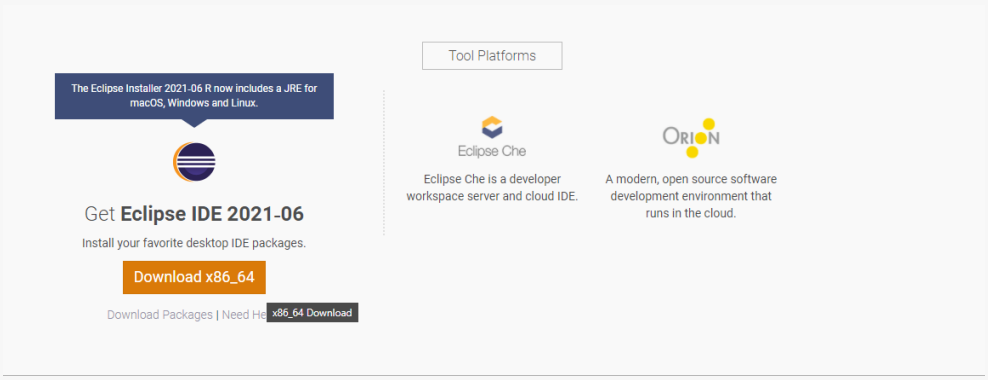
6 Com o download do instalador concluído, execute-o e siga os passos indicados. Atente-se ao diretório que for indicado por ele, haja vista que este será utilizado mais para frente;



Com a linguagem instalada, pode-se prosseguir para a obtenção da IDE que será utilizada:

INSTALAÇÃO

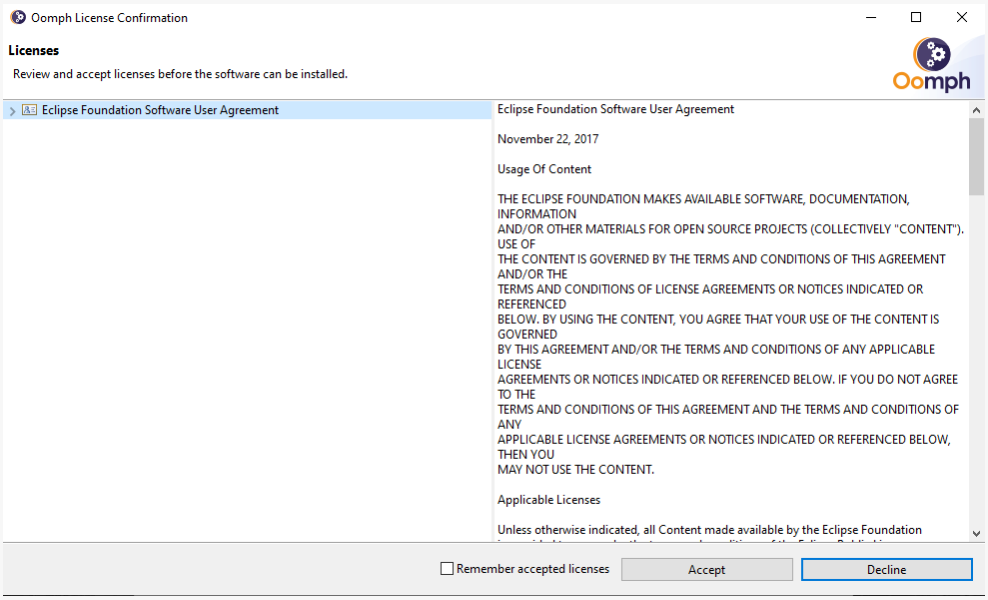
7 Inicialmente, deve-se realizar o download do instalador do ambiente de desenvolvimento Eclipse pela página **'www.eclipse.org/downloads/'**;



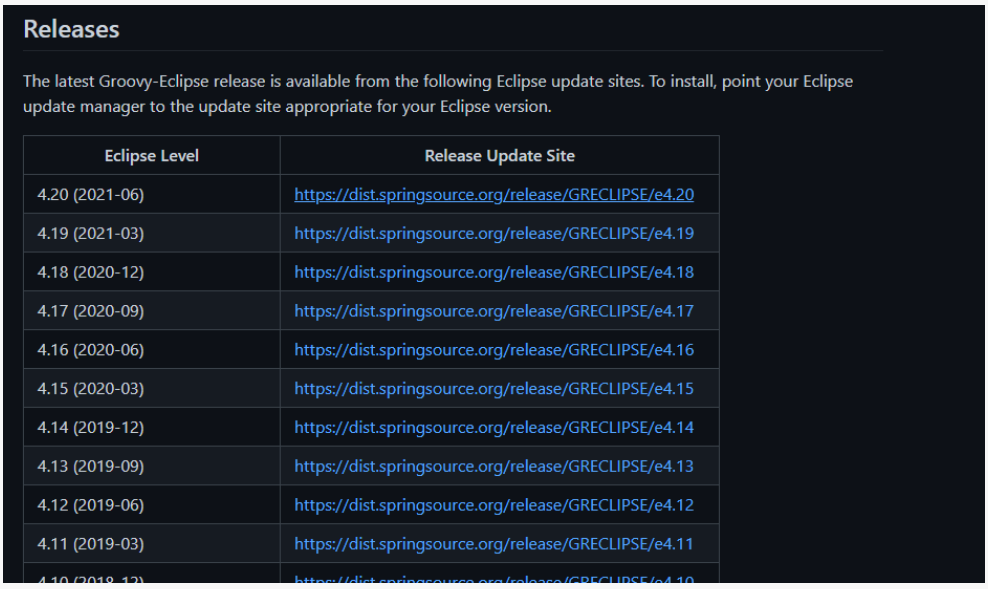
8 Assim que o download do instalador estiver concluído, execute-o selecione a primeira opção de instalação. Depois, pressione o botão de instalar e aguarde o processo ser finalizado. Aceite os termos e prossiga;

INSTALAÇÃO

9 Com o Eclipse instalado, cabe agora o download do plugin que permitirá a execução dos programas em Groovy.



10 Acesse o endereço **‘github.com/groovy/groovy-eclipse/wiki/’** e procure pela área de Releases na página. Copie o endereço da versão mais atual;



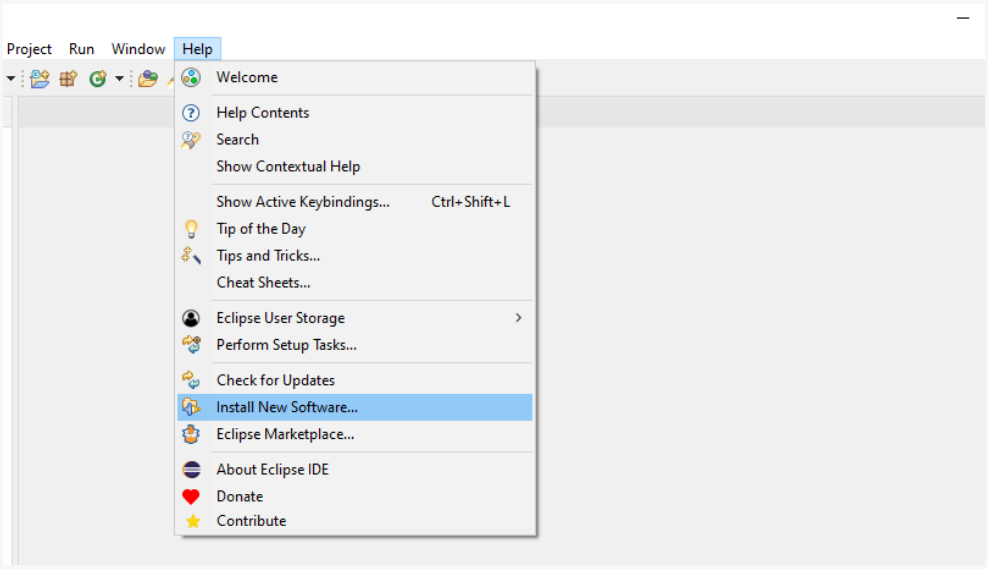
Releases

The latest Groovy-Eclipse release is available from the following Eclipse update sites. To install, point your Eclipse update manager to the update site appropriate for your Eclipse version.

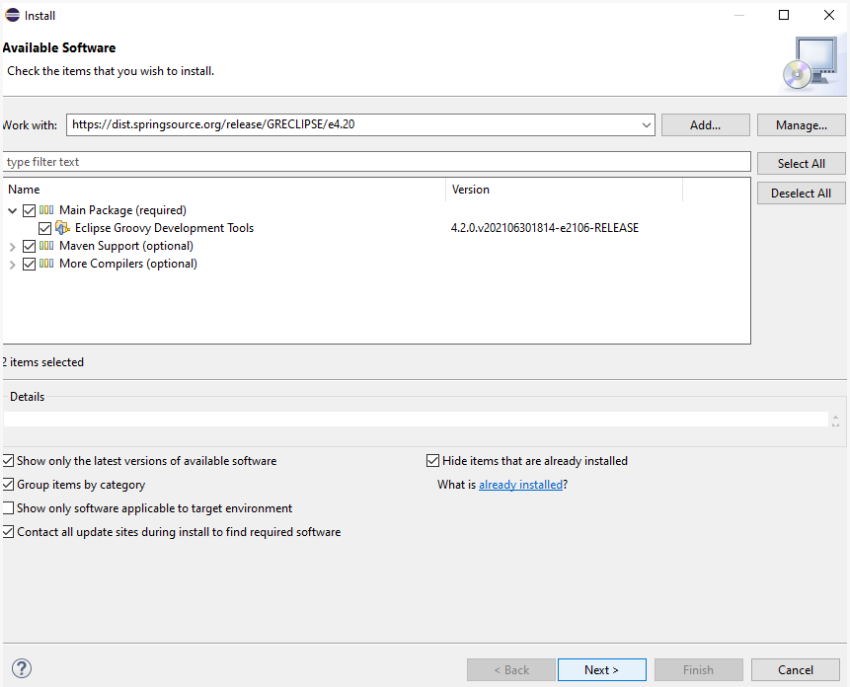
Eclipse Level	Release Update Site
4.20 (2021-06)	https://dist.springsource.org/release/GRECLIPSE/e4.20
4.19 (2021-03)	https://dist.springsource.org/release/GRECLIPSE/e4.19
4.18 (2020-12)	https://dist.springsource.org/release/GRECLIPSE/e4.18
4.17 (2020-09)	https://dist.springsource.org/release/GRECLIPSE/e4.17
4.16 (2020-06)	https://dist.springsource.org/release/GRECLIPSE/e4.16
4.15 (2020-03)	https://dist.springsource.org/release/GRECLIPSE/e4.15
4.14 (2019-12)	https://dist.springsource.org/release/GRECLIPSE/e4.14
4.13 (2019-09)	https://dist.springsource.org/release/GRECLIPSE/e4.13
4.12 (2019-06)	https://dist.springsource.org/release/GRECLIPSE/e4.12
4.11 (2019-03)	https://dist.springsource.org/release/GRECLIPSE/e4.11

INSTALAÇÃO

11 Com o endereço do plugin copiado, abra o Eclipse e procure pela aba de ajuda. Selecione a opção de instalação de software;

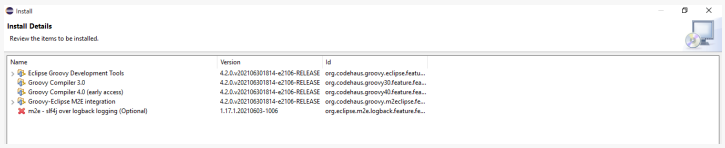


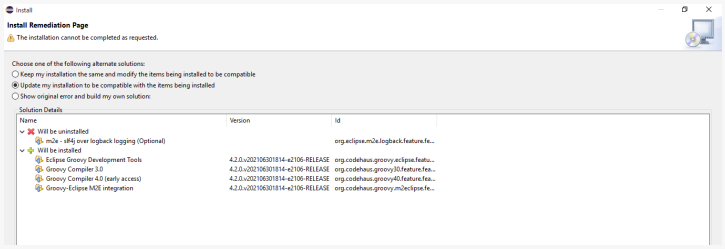
12 Na janela que abrir, procure pelo campo em que você colará o endereço copiado, selecione todas as caixas de seleção que serão mostradas e avance;

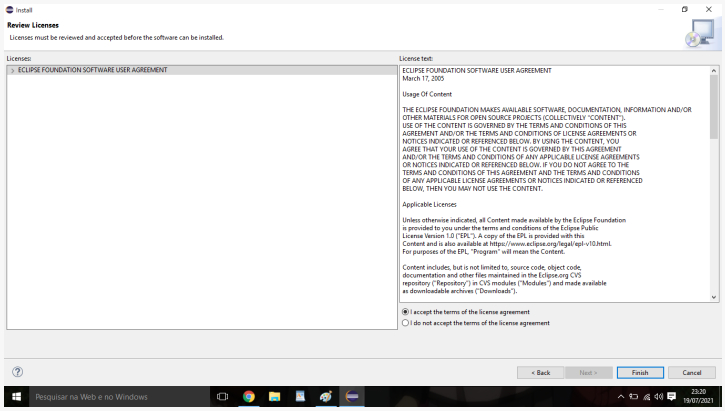


INSTALAÇÃO

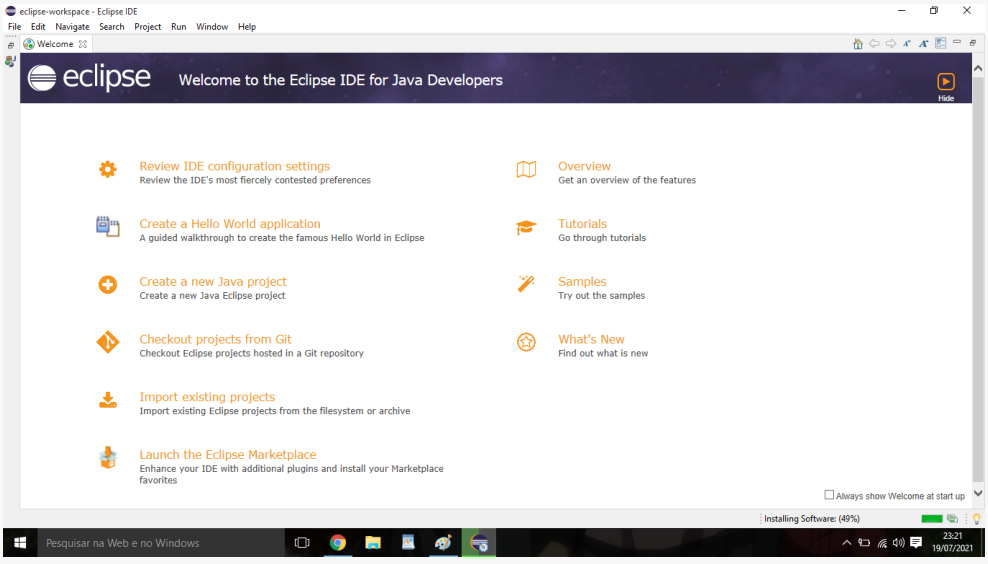
13 Caso apareça a seguinte mensagem de erro, selecione a segunda opção, avance e concorde com os termos.





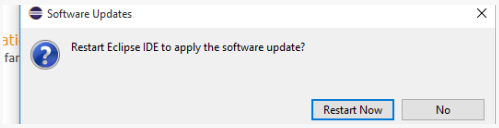


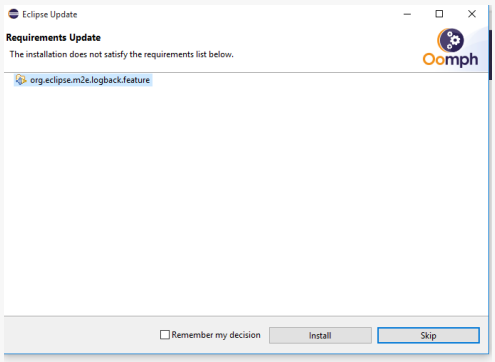
14 É importante atentar-se ao andamento da instalação, a fim de procurá-la somente após sua conclusão. Para isso, podemos observar a referente porcentagem no canto inferior direito da tela;



INSTALAÇÃO

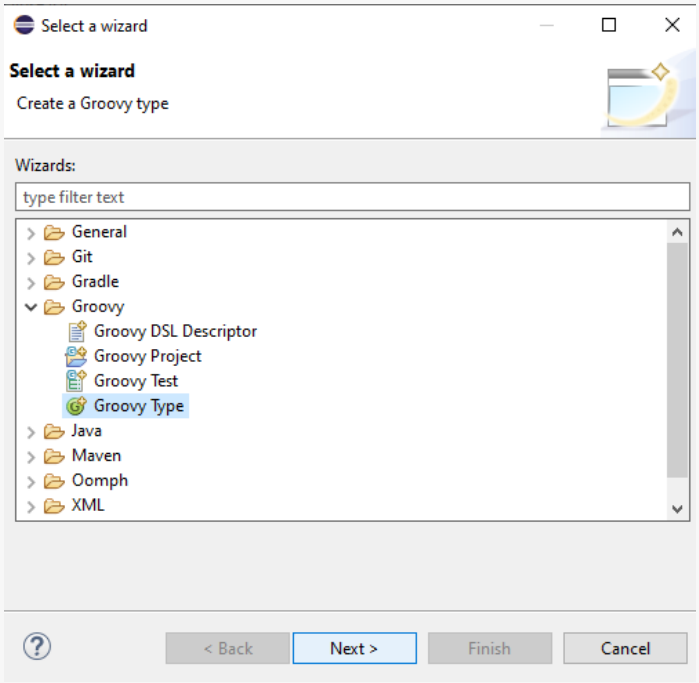
15 Ao fim da instalação será feita uma reinicialização da IDE, aceite e aguarde. Se aparecer o requerimento abaixo, apenas pule clicando na opção 'Skip'.





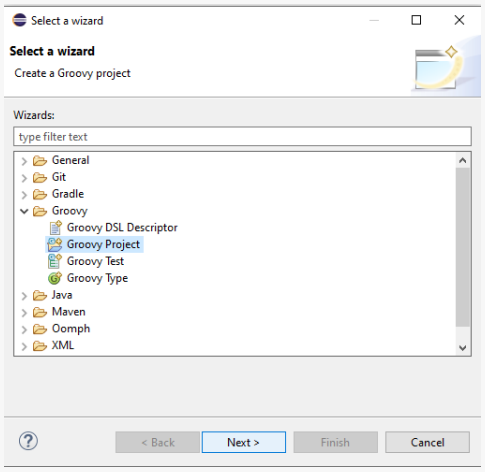
Com o plugin ativado, a criação do primeiro projeto em Groovy poderá ser realizada.

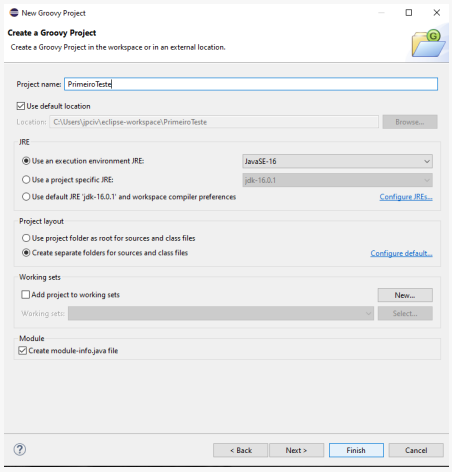
16 Selecione a opção de arquivos no Eclipse, depois, a opção de criar um novo projeto e, em seguida, clique em outras opções;



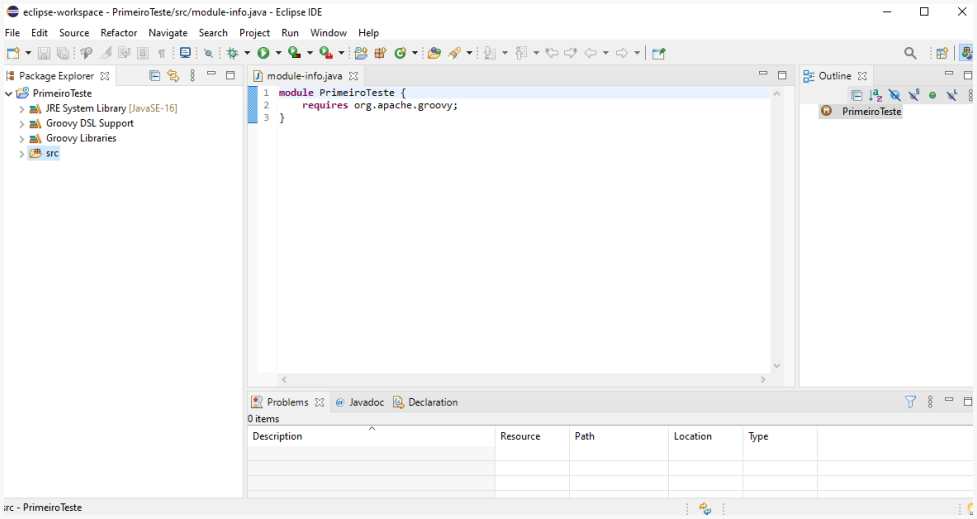
INSTALAÇÃO

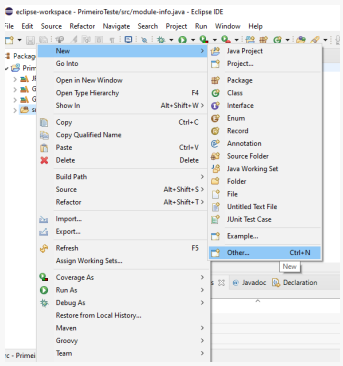
17 Selecione a categoria ‘Groovy Project’ e, em seguida, avance. Algumas janelas referentes a nomeação do projeto aparecerão;





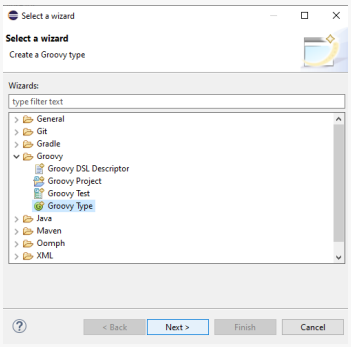
18 Procure pela pasta ‘src’ e pressione o botão direito do mouse. Pressione a opção da criação de um novo arquivo e vá em outros;

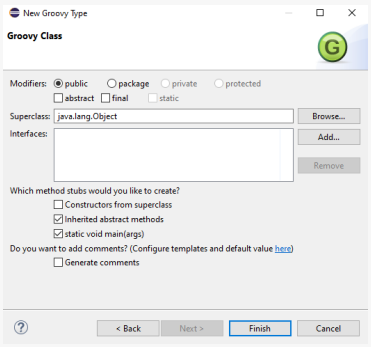


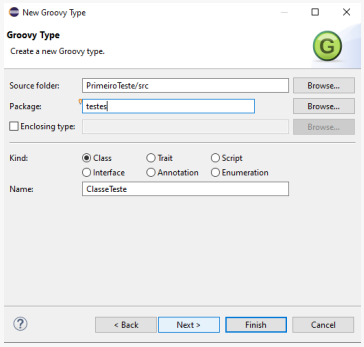


INSTALAÇÃO

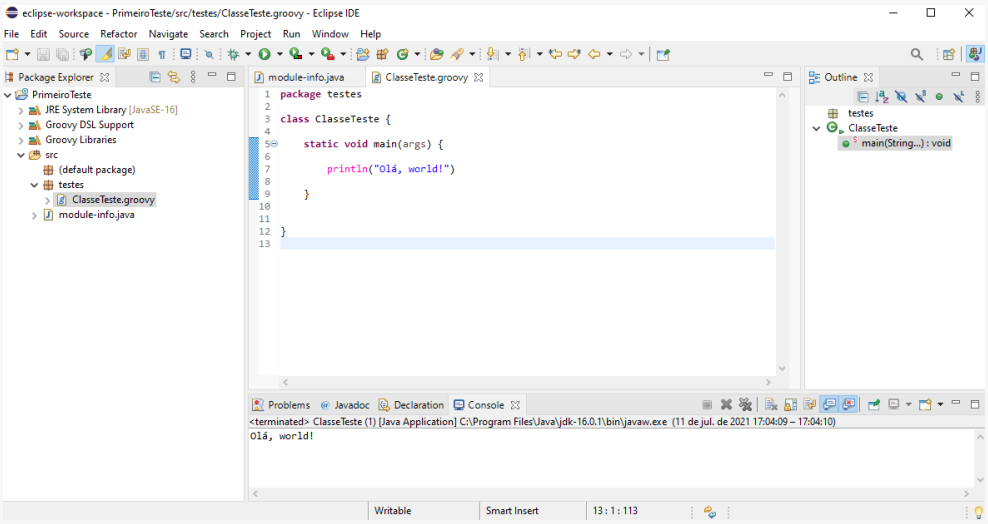
19 Selecione a opção ‘Groovy Type’ e, após isso, janelas para a nomeação dos arquivos aparecerão. Siga-as conforme mostram as imagens:







20 Uma tela será exibida onde você poderá digitar o seu código. Iniciaremos o teste com a sintaxe abaixo:



2 CAPÍTULO
COMANDOS BÁSICOS

ÁREA DO TRIÂNGULO

O exercício a seguir tem como propósito calcular a área de um triângulo através dos valores de sua base e altura inseridos pelo usuário. Primeiramente, foi realizada a declaração das variáveis.

```
Double a;
Double b;
Double h;
```

Após isso, o usuário entrará com um valor para a base do triângulo. Esse valor será armazenado na variável *b*. O *System.in.withReader* é utilizado para que permita a ação do usuário.

```
System.in.withReader{
println("Digite um valor para a base do triângulo: ");
def b = System.console().readLine().toDouble();
```

Agora, o valor da altura do triângulo será atribuído e armazenado na variável h.

```
println("Digite um valor para a altura do triângulo: ");
def h = System.console().readLine().toDouble();
```

Por fim, a área do triângulo será exibida.

```
 println( (b * h)/2 );

 }
```

SENO E COSSENO

Neste exercício, iremos calcular o seno e o cosseno com base no ângulo fornecido pelo usuário. Criando a variável de armazenamento da digitação, temos:

Double angulo;

E então, solicitamos a entrada, armazenando a informação digitada:

```
System.in.withReader{
println("Digite o valor do ângulo: ");
angulo = it.readLine().toDouble();
```

Utilizando a função *Math* para calcular o seno e o cosseno do ângulo digitado e exibindo o resultado:

```
println(" O valor do seno é = " + (Math.sin(angulo));
println(" O valor do seno é = " + (Math.cos(angulo));
}
```

MÉDIA ARITMÉTICA

APLICANDO IF-ELSE

Neste exercício, o programa deverá calcular a média aritmética entre quatro notas e falar para o usuário se ele está aprovado, reprovado ou de exame.

Em primeiro lugar, foi declarado cinco variáveis; as quatro primeiras estão sendo utilizadas para armazenar os valores de cada nota, e a última será utilizada para o cálculo:

```
Double primeiran;
Double segundan;
Double terceiran;
Double quartan;
Double media;
```

Neste momento, deve haver o pedido de digitação das notas do usuário nas variáveis já criadas:

```
System.in.withReader{
println("Digite a primeira nota:");
primeiran = it.readLine().toDouble();
println("Digite a segunda nota:");
segundan = it.readLine().toDouble();
println("Digite a terceira nota:");
terceiran = it.readLine().toDouble();
println("Digite a quarta nota:");
quartan = it.readLine().toDouble();
```

Agora será feita a média aritmética, que está na variável media:

media = ((primeiran + segundan + terceiran + quartan)/4);

Neste momento, será feito um *if else* para mostrar para o usuário se ele está aprovado, reprovado ou de exame. Caso a média seja maior ou igual a seis, ele está aprovado; se a média for menor que três, ele está reprovado; caso a média fique menor que seis e maior que três, ele está de exame.

```
if(media >= 6){
println("Você está aprovado!");
}
else{
if (media < 3){
println("Você está reprovado!");
}
else{
println("Você está de exame!");
}
}
}
```

CALCULADORA

APLICANDO IF ELSE

Neste exercício, será solicitado ao usuário dois valores e um caractere. Mediante ao caractere digitado, o programa fará o respectivo cálculo e exibirá o resultado.

Inicialmente, definiu-se as variáveis juntamente com seus tipos. A entrada de dados foi realizada como pode ser analisado a seguir:

```
String caracter;
Double numero1;
Double numero2;
Double resultado=0;

System.in.withReader{
println ("Informe um número:")
numero1 = it.readLine().toDouble();

println ("Informe outro número:");
numero2 = it.readLine().toDouble();

println ("Informe um caractere (+, -, * ou /):")
caracter =it.readLine();
}
```

Utilizou-se *if* para verificar o caractere escolhido e realizar o cálculo. Para finalizar, foi inserido um *else*, que só será acessado se o caractere não corresponder a +, -, * ou /.

```
if (caracter == "+"){
```

A variável resultado está armazenando o valor do cálculo, por esse motivo, para apresentar ao usuário, manipula-se o comando:

```
println(“${resultado}”).
resultado = (numero1 + numero2)
println("Resultado: ${resultado}");
else{
if (caracter == "-"){
resultado = (numero1 - numero2)
println("Resultado: ${resultado}");
 }
else{
if (caracter == "*"){
resultado = (numero1 * numero2)
println("Resultado: ${resultado}");
}
else{
if (caracter == "/"){
resultado = (numero1 / numero2)
println("Resultado: ${resultado}");
}
else{
println("Digite um caracter válido!")
}
}
}
}
}
}
```

CALCULADORA

APLICANDO SWITCH-CASE

Igualmente ao exercício anterior, será solicitado ao usuário dois valores e um caractere. Mediante ao caractere digitado o programa fará o respectivo cálculo e exibirá o resultado.

A definição de variáveis juntamente aos seus tipos realizou-se, também, semelhante ao exercício anterior, como pode ser visto a seguir:

```
String caracter;
Double numero1;
Double numero2;
Double resultado=0;

System.in.withReader{
println ("Informe um número:")
numero1 = it.readLine().toDouble();

println ("Informe outro número:");
numero2 = it.readLine().toDouble();

println ("Informe um caractere (+, -, * ou /):")
caracter = it.readLine();
}
```

O que difere do outro exercício, portanto, é esta parte, na qual a estrutura de decisão é o switch-case.

```
switch(caracter){
case "+":
resultado = (numero1 + numero2)
println("Resultado: ${resultado}");
break
case "-":
resultado = (numero1 - numero2)
println("Resultado: ${resultado}");
break
case "*":
resultado = (numero1 * numero2)
println("Resultado: ${resultado}");
break
case "/":
resultado = (numero1 / numero2)
println("Resultado: ${resultado}");
break
default:
println("Digite um caracter válido!");
}
}
}
```

TABUADA

APLICANDO FOR

Neste exemplo, criaremos um programa que deve começar solicitando a digitação de um número inteiro e positivo; a partir de então, o programa deverá exibir na tela a tabuada desse número.

Primeiramente, começamos com a declaração das variáveis que utilizaremos no programa: *valor* para armazenar o número digitado pelo usuário e *resultado* para armazenar o resultado de cada multiplicação da tabuada.

```
int valor;
int resultado;
```

Abrimos um *System.in.withReader* e, através do *println,* pedimos o valor desejado ao usuário.

```
System.in.withReader{
println ("Digite o valor desejado:");
```

Assim, após isso, transformamos o valor armazenado em *int* para que seja possível realizar operações com esse.

```
valor = it.readLine().toInt();
```

Depois, criamos um *for* para rodar entre os índices presentes em uma tabuada; atribuímos o resultado da multiplicação à variável *resultado* para que, assim, possamos exibir os valores da tabuada um de cada vez.

```
for(int i=0;i<=10;i++)
{
resultado = i*valor;
println("${valor} x ${i} = ${resultado}")
}
}
```

SEQUÊNCIA DE NÚMEROS

APLICANDO FOR

Neste exercício, irá aparecer uma sequência de números, mas é o usuário que escolhe quantos números aparecerá na tela. Por exemplo: caso ele digite “2”, então irá aparecer o 2 e 5; caso ele digite “5”, então irá aparecer 2, 5, 10, 17 e 26; e assim por diante.

Em primeiro lugar, foram declaradas as variáveis; a primeira é para armazenar quantas vezes o usuário quer que um número apareça; a segunda é para armazenar o termo geral (x^2 +1); e a última é para o *x*, que é o número de vezes que o programa vai rodar dentro dele:

```
Double n;
int valor;
int x=0;
```

Agora, o usuário irá digitar a quantidade de vezes que ele quer, o valor será atribuído a variável *n*:

```
System.in.withReader{
println("Digite o número de vezes de sua preferência: ");
n = it.readLine().toDouble();
```

Nesse caso, foi declaro um *i* dentro do *for*; este pega o valor que foi digitado pelo usuário para saber quantos números da sequência será necessário para mostrar na tela:

```
for (int i = 0; i < n; i++){
x++;
valor = (x * x + 1);
}
}
```

SEQUÊNCIA DE FIBONACCI

APLICANDO WHILE

Nesse exercício, o programa deverá listar os termos da série Fibonacci menores que 1000. Começaremos declarando três variáveis, duas delas com valores definidos e uma que servirá de auxiliar na estrutura de repetição:

```
int num1 =0;
int num2 =1;
int aux;
```

É importante destacar que as variáveis podem ser declaradas individualmente, como mostrado, ou juntas:

```
int num1 =0, num2 =1, aux;
```

Para o início da sequência de Fibonacci, necessitaremos utilizar o *while* para que percorra-se os valores menores,ou iguais a 1000:

```
while (num2 <= 1000)
{
```

Os valores devem ser somados e atribuídos uns aos outros da forma a seguir:

```
aux=num1;
num1 = num2;
num2 = num1 + aux;
```

Para a finalização do programa, é imprescindível que sejam exibidos na tela, utilizando o *println*:

```
println(" ${num2}");
}
```

EXERCÍCIO DE CONDIÇÃO

APLICANDO IF-ELSE E WHILE

Neste exercício, o programa pedirá dois valores ao usuário. A condição é que o segundo valor seja maior que o primeiro e, caso contrário, o programa deverá entrar em *looping* até que a condição seja satisfeita. Para iniciarmos, é necessário fazer a criação de duas variáveis:

```
int valor1;
int valor2;
```

Agora, deve haver o pedido de digitação dos dados e o armazenamento da informação provida pelo usuário nas variáveis já criadas:

```
System.in.withReader{
println ("Digite um valor:");
valor1 = it.readLine();
println ("Digite outro valor:");
valor2 = it.readLine();
```

Para que o programa, verifique se a condição do maior valor foi satisfeita; é necessário a utilização de um *if*:

```
if (valor2 > valor1){
println ("Fim");
};
```

Na sintaxe acima, a condição foi satisfeita e o programa acabou. Caso esta não ocorra, crie um *else* e, dentro dele, coloque um *while* para que repita o processo até que o segundo seja maior que o primeiro:

```
else{
while (valor1 > valor2){
println ("Digite um valor:");
valor1 = it.readLine();
println ("Digite outro valor:");
valor2 = it.readLine();
}
}
```

CONDIÇÃO DO SEXO

APLICANDO IF-ELSE

Neste exercício, o usuário deverá digitar o sexo utilizando as iniciais F ou M. Caso a *string* digitada seja diferente, o programa entrará em um laço de repetição até que a condição seja satisfeita, assim como no exercício anterior; para isso criaremos uma *string*:

```
String caracter;
```

Agora, deve haver o pedido de digitação do dado e o armazenamento da informação provida pelo usuário na variável criada:

```
System.in.withReader{
println ("Informe o sexo:");
caracter = it.readLine();
```

Para saber se a informação digitada compreende a condição, criaremos um *if*:

```
if (caracter == "F" || caracter == "M"){
println("Fim");
}
```

Caso a *string* seja diferente, o programa entrará em um *looping* com o *while* até que o F ou o M seja informado pelo usuário:

```
else {
while (caracter != "F" || caracter != "M"){
println ("Informe o sexo:");
caracter = it.readLine();
}
}
}
```

FATORIAL

APLICANDO DO-WHILE

Esse exercício tem como propósito apresentar o fatorial de um número qualquer, inteiro e positivo. Primeiramente, foi realizada a criação do *System.in.Reader* a fim de invocar a função de leitura da entrada do usuário. Após isso, foi realizada a declaração das variáveis em número inteiro.

```
System.in.Reader{
 int i, fat, valor;
char resp;
```

A entrada do valor que se deseja calcular o fatorial é realizada através de uma estrutura *do_while,* na qual, caso o usuário digite um valor inválido, o programa solicitará a entrada de um valor até que este atenda aos requisitos básicos, ou seja, inteiro e positivo.

```
do{
println("Digite um valor: ");
valor = it.readLine();
} while (valor <= 0);
```

O laço *for* é uma estrutura de repetição utilizada para realizar o cálculo do fatorial do número desejado. Após o cálculo, o valor é exibido.

```
fat = 1;
for (i = 1; i <= valor; i++) {
fat = fat * i;
println " " + valor " = " + fat;
```

Por fim, o usuário poderá escolher entre calcular o fatorial de outro valor. Foi utilizada a repetição *do_while*, introduzida no início do programa e concluída no final deste. Portanto, o programa realizará todas as etapas novamente enquanto o usuário digitar *S*. Caso contrário, se ele digitar *N,* o programa chegará ao fim.

```
do {
println("Deseja tentar novamente? ")
resp = it.readLine().toUpperCase();
} while (resp != ‘S’ && resp != ‘N’);
} while(resp == 'S');
}
```

ORDEM INVERSA

APLICANDO ARRAYS

Neste exemplo, criaremos um programa que recebe dez valores digitados pelo usuário e, em seguida, exibe-os na ordem inversa. Utilizamos, primeiramente, a criação de um vetor vazio.

```
double valores=[]
```

Utilizamos o *for* que percorre os valores no intervalo de um a dez e, paralelamente, pede ao o usuário que alimente a posição do vetor correspondente.

```
System.in.withReader{
for (indice in 1..10){
println("Digite um valor: ")
valores[indice] = it.readLine().toDouble();
}
```

Invocamos a função de inverter os valores do vetor e armazenamos o resultado em um novo. Em seguida, iniciamos a varredura dele utilizando um *for*.

```
def valoresReverso = valores.reverse()
for (indice in valoresReverso){
println(indice)
}
```

MÉDIA ARITMÉTICA

APLICANDO ARRAYS

Agora criaremos um programa que também recebe dez valores digitados pelo usuário, mas, em seguida, exibe a média aritmética deles. Após a criação do vetor vazio, criamos a variável que conterá a somatória dos valores.

```
double valores=[]
double soma = 0
```

Assim como feito anteriormente, criamos um *for* para receber os valores que o usuário optar.

```
System.in.withReader{
for (indice in 1..10){
println("Digite um valor: ")
valores[indice] = it.readLine().toDouble();
 }
```

Para realizar a somatória, utilizamos um outro *for* que acumulará os valores escolhidos dentro da variável anteriormente criada. Por fim, para exibir a média, dividimos o resultado pela quantidade de valores digitados.

```
 for (indice in valores){
 soma += indice
}
print (soma/10)
```

MENOR E MAIOR VALOR

APLICANDO ARRAYS

Para que possamos localizar o menor e maior valor digitado, utilizamos, incialmente, a mesma estrutura para capturar os valores do usuário dos exemplos anteriores.

```
double valores=[]
double soma = 0

System.in.withReader{
for (indice in 1..10){
println("Digite um valor: ")
valores[indice] = it.readLine().toDouble();
}
```

Em seguida, invocamos as funções *max* e *min* para encontrar o resultado almejado.

```
println(valores.max())
println(valores.min())
```

PESQUISA DE NOME

APLICANDO ARRAYS

Neste programa, o usuário irá entrar com dez nomes e suas respectivas idades. Após a entrada dos dados, o usuário poderá realizar uma pesquisa, na qual ele entrará com o nome e será exibida a idade dele. No final, será perguntado se ele deseja realizar outra pesquisa. Primeiramente, iremos declarar as variáveis e vetores que irão receber os nomes, idades e a resposta do usuário.

```
String nome [];
Int idade [];
String nome1;
String nome2;
String resposta;
```

Agora, iremos utilizar um for para receber os dados digitados.

```
System.in.withReader{
for (indice in 1..10){
println(“Digite o nome: “)
nome1[indice] = it.readLine();
println(“Digite a idade: “);
idade1[indice] = it.readLine().toint();
}
```

Após a digitação dos dez nomes e idades, iremos pedir para que o usuário coloque um nome para realizarmos a pesquisa.

```
Println (“Por favor, digite um nome para a pesquisa: “)
nome2 = it.readLine();
```

Agora, utilizaremos um *if else* para checar a informação; caso o nome seja correspondente aos informados no começo, a idade será exibida, do contrario, aparecerá uma mensagem dizendo que a pessoa não foi localizada.

```
if ( nome[nome1] == nome2 ){
println(“A idade dessa pessoa é: “ + idade[idade1]) ;
}
Else {
Println(“Pessoa não localizada.”)
}
```

Perguntaremos se o usuário deseja realizar uma nova pesquisa; caso sim, outra mensagem pedindo um nome será exibida.

```
println(“Deseja fazer uma nova reserva? (‘S’ ou ‘N’)”);
resposta = it.readLine();
while(resposta !=”S” || resposta !=”N”);{
Println (“Por favor, digite um nome para a pesquisa: “)
nome2 = it.readLine();
```

```
if ( nome[nome1] == nome2 ){
println(“A idade dessa pessoa é: “ + idade[idade1]);
}
Else {
Println(“Pessoa não localizada.”)
}
```

CINEMA

APLICANDO ARRAYS E ESTRUTURAS DE REPETIÇÃO

Por fim, desenvolveremos um programa tendo em mente a seguinte situação: determinado cinema tem vinte fileiras com quinze cadeiras cada uma; portanto, estamos falando de uma sala com trezentos assentos.

Primeiramente, devemos começar solicitando ao usuário a digitação de seu nome, o número da fileira e a cadeira em que deseja se sentar. Se o assento estiver vazio, ele será reservado, registrando no *array* seu nome, caso contrário, deverá se informar que o assento está ocupado. Após isso, o programa deverá nos questionar se desejamos nova reserva, validando nossa resposta e repetindo todo o processo.

Antes de tudo, devemos declarar as variáveis que vamos utilizar no programa; primeiramente, criamos a matriz que refere-se às cadeiras do cinema. Depois, as variáveis do nome do cliente, número da fileira, número da coluna e resposta do usuário quanto à fazer uma nova reserva.

```
String[][] cadeiras = new String [20][15];
String nomeCliente;
int fileira;
int coluna;
String resposta;
```

Agora, abrimos um *System.in.withReader* para realizar o *input* das informações do cliente. E também pedimos o nome do cliente e o armazenamos em sua respectiva variável.

```
System.in.withReader{
println("Digite seu nome:");
nomeCliente = it.readLine();
```

Perguntamos o número da fileira desejada e o armazenamos. Além disso, após atribuir o valor de tal variável, diminuímos uma unidade em seu valor original.

```
println("Qual é o número da fileira desejada? (1 até 20)");
fileira = it.readLine().toInt();
fileira = fileira-1;
```

Agora, perguntamos o número da cadeira desejada para que possamos saber qual é a coluna desejada, e também fazemos o mesmo que fizemos com as fileiras: após atribuir o valor da coluna, subtraímos uma unidade de seu valor.

```
println("OK! Agora digite o número da cadeira desejada (1 até 15):");
coluna = it.readLine().toInt();
coluna = coluna-1;
```

Depois do processo anterior, utilizamos os números digitados pelo usuário referentes à fileira e cadeira do cinema como seus respectivos índices de linha e coluna. A partir desse valor, checamos se essa posição da matriz está vazia. Caso a posição desejada esteja vazia, atribuímos o nome do cliente nesta e informamos. Porém, se a posição não estiver vazia, significa que outro cliente já a reservou; sendo assim, informamos o cliente deste fato.

```
if(cadeiras[fileira][coluna] == ""){
cadeiras[fileira][coluna] = nomeCliente;
println("Reserva finalizada com sucesso!");
}
else{
println("Ops! Este assento já está reservado...");
}
```

Após essa operação, perguntamos ao usuário se deseja realizar uma nova reserva, e validamos a resposta do usuário para que este possa responder somente com "S" ou "N".

```
println("Deseja fazer uma nova reserva? ('S' ou 'N')");
resposta =it.readLine();
while(resposta != "S" || resposta != "N"){
println("Deseja fazer uma nova reserva? ('S' ou 'N')");
resposta =it.readLine();
}
```

Assim, depois, enquanto a resposta do cliente for sim ("S"), repetimos todo o processo de reserva.

```
while(resposta == "S"){
println("Digite seu nome:");
nomeCliente = it.readLine();

println("Qual é o número da fileira desejada? (1 até 20)");
fileira = it.readLine().toInt();
fileira = fileira-1;
println("OK! Agora digite o número da cadeira desejada (1
até 15):");
coluna = it.readLine().toInt();
coluna = coluna-1;

if(cadeiras[fileira][coluna] == ""){
cadeiras[fileira][coluna] = nomeCliente;
println("Reserva finalizada com sucesso!");
}
else{
println("Ops! Este assento já está reservado...");
}

 println("Deseja fazer uma nova reserva? ('S' ou 'N')");
 resposta =it.readLine();

 while(resposta != "S" || resposta != "N"){
 println("Deseja fazer uma nova reserva? ('S' ou 'N')");
 resposta =it.readLine();
 }
 }
}
```

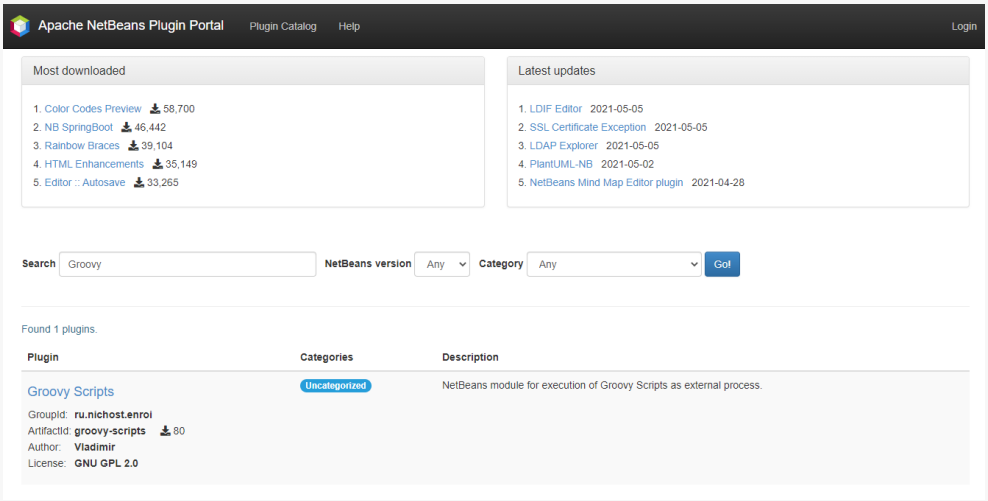
3 CAPÍTULO

WINDOWS FORMS APPLICATION

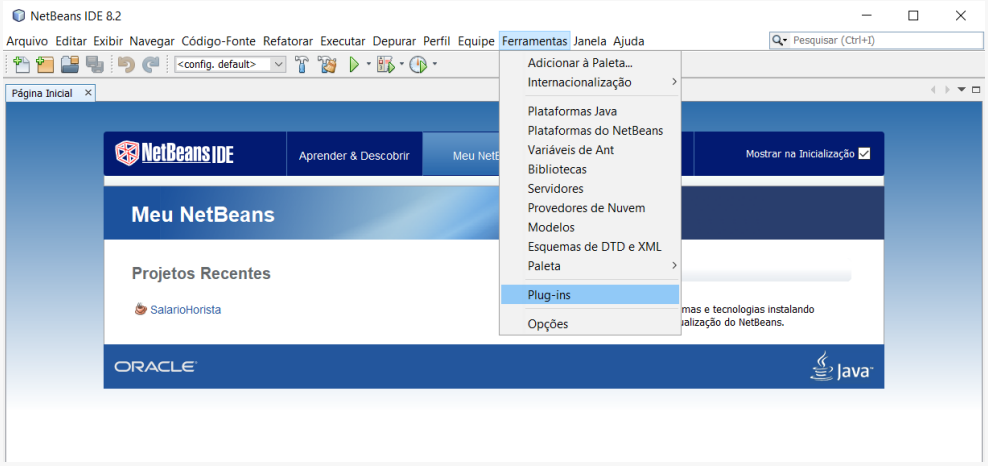
COMO CONECTAR UM FORMULÁRIO EM JAVA COM UMA CLASSE GROOVY

Para realizar a programação de formulários utilizando o Groovy, precisamos, primeiramente, utilizar uma ferramenta que facilite o processo de confecção dos próprios formulários. Para isso, utilizaremos a IDE do NetBeans 8.2.

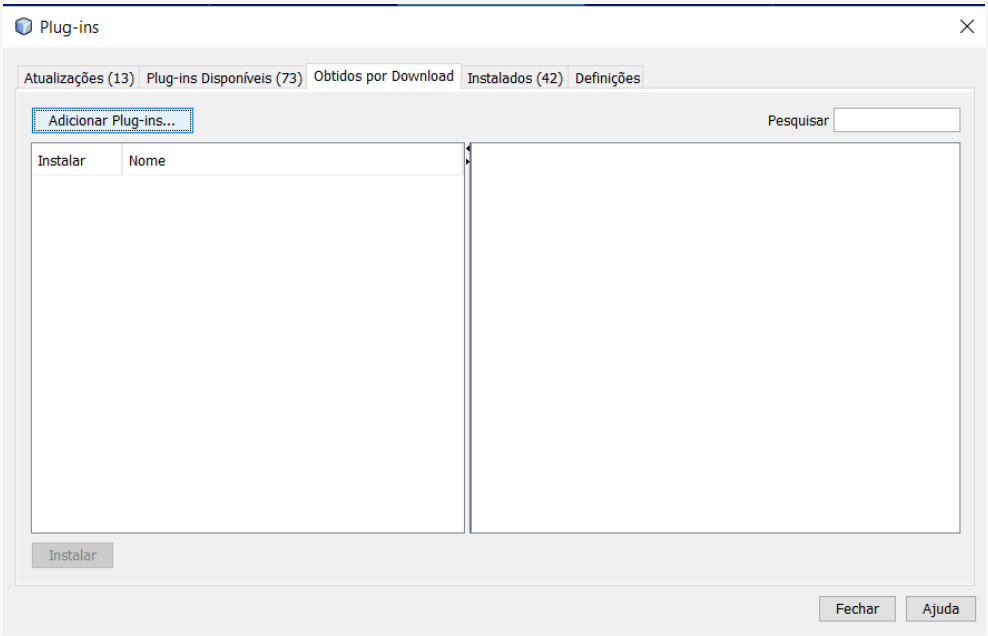
Inicialmente, deve-se realizar o download do plugin no endereço **‘plugins.netbeans.apache.org/’** e procurar pela opção Groovy no buscador da página;



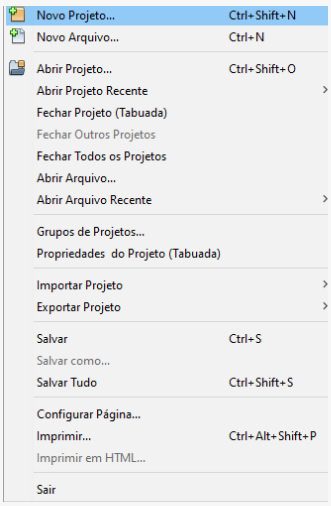
Com o arquivo baixado, abra o NetBeans e selecione o botão “ferramentas” que está na página inicial e clique em “plug-ins".

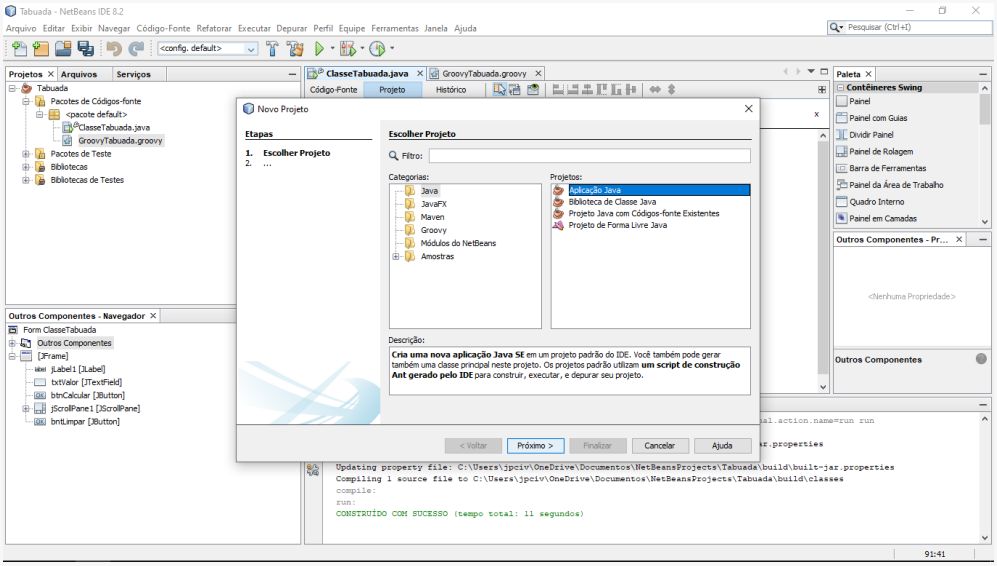


Na página que abrirá, escolha “obtidos por download” e aperte o botão “Adicionar Plug-ins...” e encontre em seu computador o arquivo Groovy que foi baixado na primeira etapa e clique no botão inferior esquerdo “instalar” e siga as orientações que aparecem na tela.

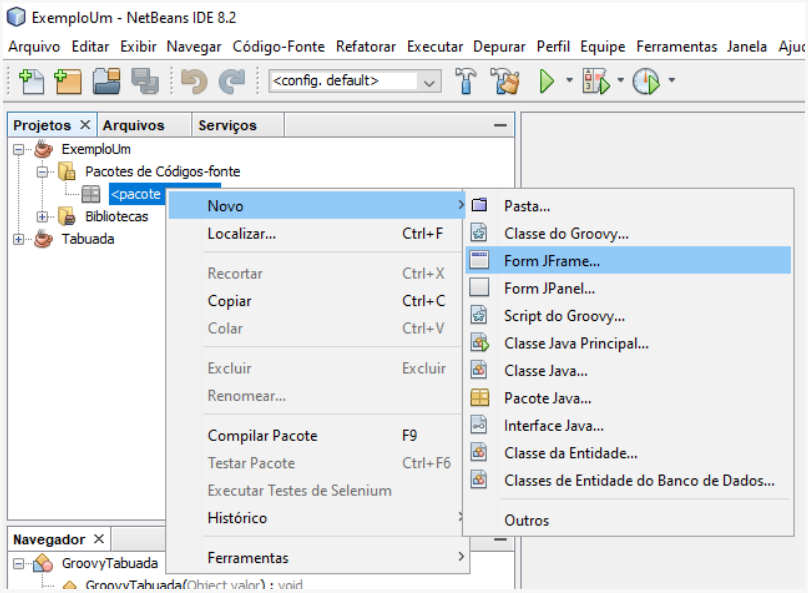


Crie um novo projeto e, em seguida, selecione a opção de aplicação em Java.

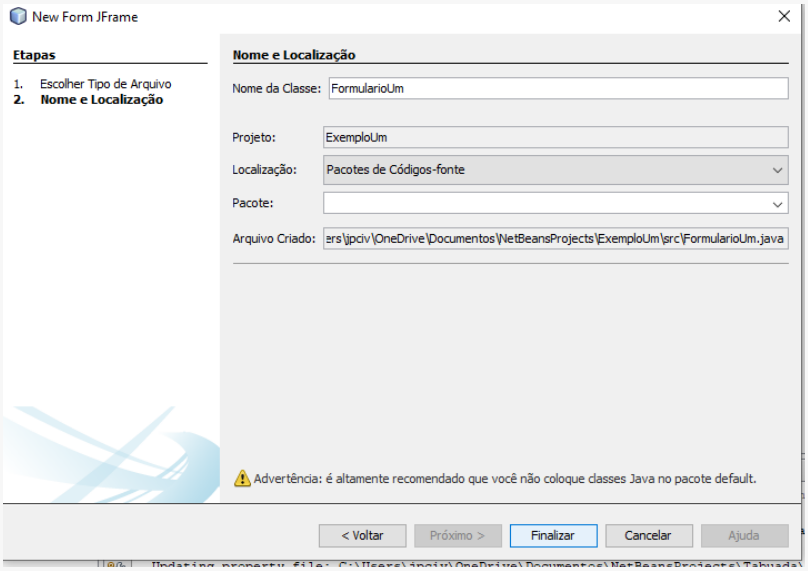




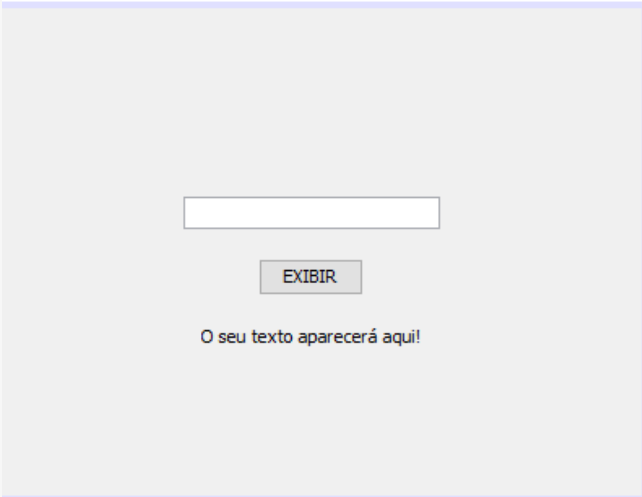
Nomeie a classe e desabilite a opção de criação de classe principal. Selecione a opção de pacote padrão e, com o botão esquerdou do mouse, crie um novo arquivo no formato JFrame.



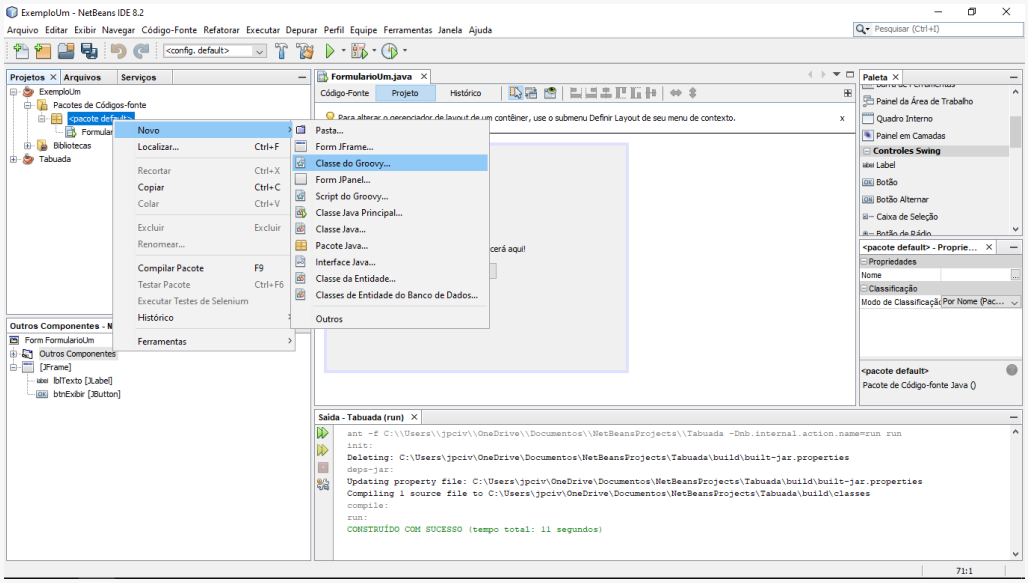
Nomeie o formulário e finalize.



A fins de exemplo, monte um formulário básico como o da imagem abaixo, lembrando sempre de configurar os nomes dos componentes.



Agora, criaremos a classe Groovy que estará conectada ao formulário. No pacote padrão, pressione o botão esquerdo do mouse e adicione uma nova classe em formato Groovy.



Nomeie a classe e finalize. No código abaixo, está um exemplo de classe que recebe um parâmetro do formulário Java para manipular a digitação conforme a vontade do usuário.

```
class ClasseGroovy {
String reposta;
ClasseGroovy(valor) {
this.reposta = "O seu valor digitado é: "+valor;
}}
```

No código do formulário Java, programaremos o botão para que ele possa se conectar à classe Groovy, enviar o parâmetro digitado pelo usuário e retornar a resposta para exibição.

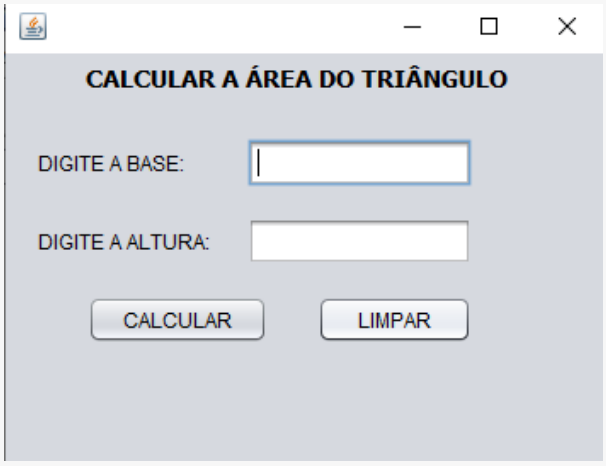
```
private void
btnExibirActionPerformed(java.awt.event.ActionEvent
evt) {
String texto = txtExibir.getText();
ClasseGroovy groovy = new ClasseGroovy(texto);
String exibir = groovy.getReposta().toString();
lblTexto.setText(exibir);}
```

Note que criamos um objeto chamado “groovy” da classe que fora criada anteriormente e enviamos, ao mesmo tempo, o parâmetro digitado pelo usuário no formulário. Em seguida, usou-se uma variável para retornar a resposta da classe Groovy, utilizando a sintaxe **“objeto.getVariavel().toString();”**. A reposta da execução deverá parecer desta maneira:

ÁREA DO TRIÂNGULO

EXEMPLO 01

Neste exercício, iremos calcular a área do triângulo com base nos dados de base e altura que serão fornecidos pelo usuário; o formulário utilizado está representado abaixo:



Para que o resultado seja exibido, iremos programa-lo na classe Groovy, da seguinte forma:

```
class ClasseArea {
String reposta;
ClasseArea(base, altura) {
this.reposta = "A área é: "+ (base * altura)/2;
}
}
```

Com a classe criada, o próximo passo é intregra-lo ao código do formulário em Java; para isso é necessário dar um “duplo clique” no botão “Exibir” do *form*, e inserir a seguinte sintaxe:

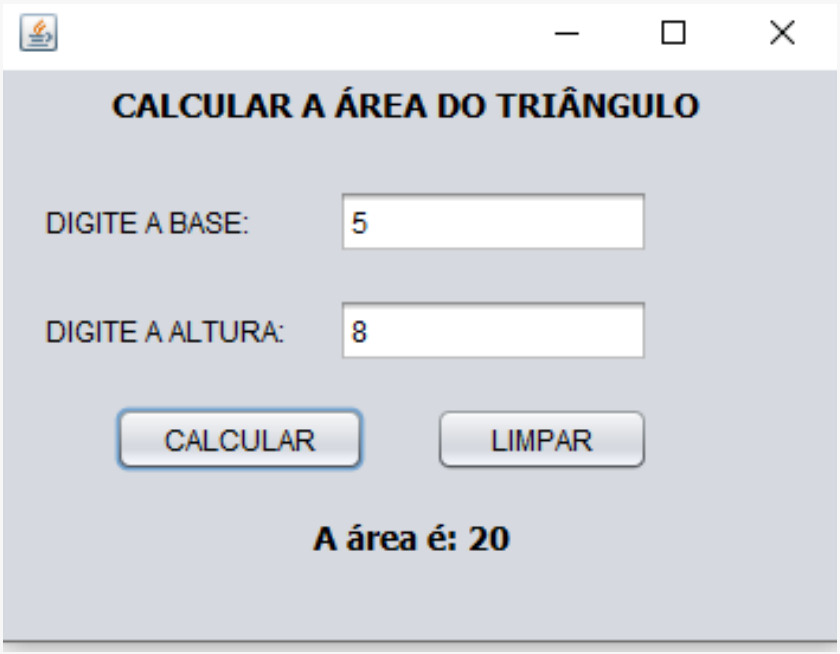
```
private void
btnCalcularActionPerformed(java.awt.event.ActionEvent
evt) {

 int base = Integer.parseInt(txtBase.getText());
 int altura = Integer.parseInt(txtAltura.getText());

 ClasseArea groovy = new ClasseArea(base, altura);
 String resposta = groovy.getReposta().toString();
 lblArea.setText(resposta);
 }
```

É imperativo pontuar que os valores devem estar atribuídos à variáveis numéricas para que o cálculo seja efetuado, caso contrário os valores serão apenas concatenados.

Para finalizar, o botão “Limpar” deve ser programado, para fazê-lo dê novamente um duplo clique e insira os seguintes comandos:

```
private void
btnLimparActionPerformed(java.awt.event.ActionEvent
evt) {

 txtBase.setText("");
 txtAltura.setText("");
 lblArea.setText("");

 }
```

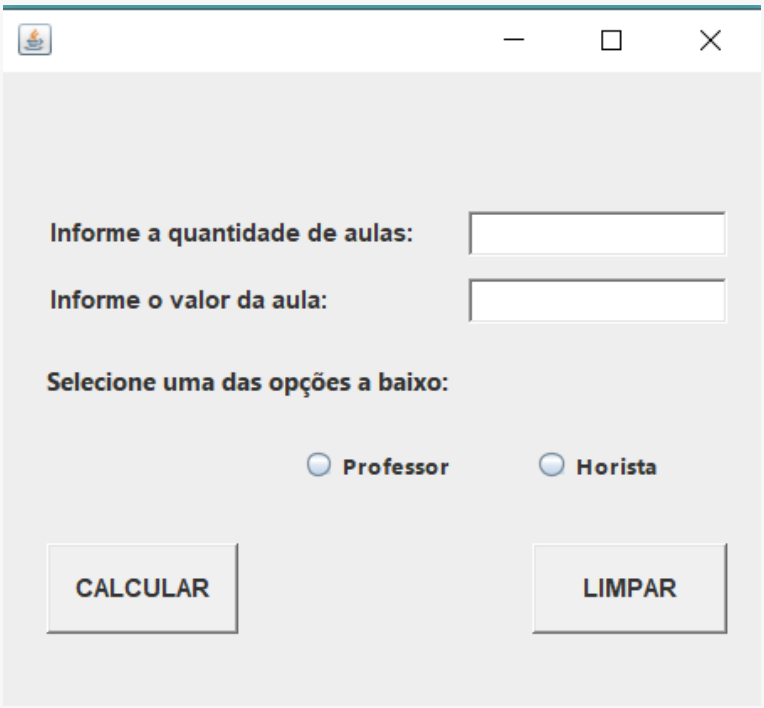
Assim, ao fim do programa quando houver a necessidade de limpar os campos apenas clique no botão. Com a programação concluída, o resultado deve apresentar-se dessa forma:



HORISTA

EXEMPLO 02

Para realizar a confecção desse exercício que possui o objetivo de calcular o salário de um professor e de um horista, criou-se um formulário que se encontra ilustrado a baixo:



Primeiramente, criou-se uma classe Groovy para que ela receba os valores que estão no formulário, como pode ser analisado a baixo:

```
class ClasseGroovy {
String resposta;
ClasseGroovy(salario){
this.resposta = "O salário é " +salario;
}
}
```

A seguir, desenvolveu-se a programação do botão calcular e limpar.

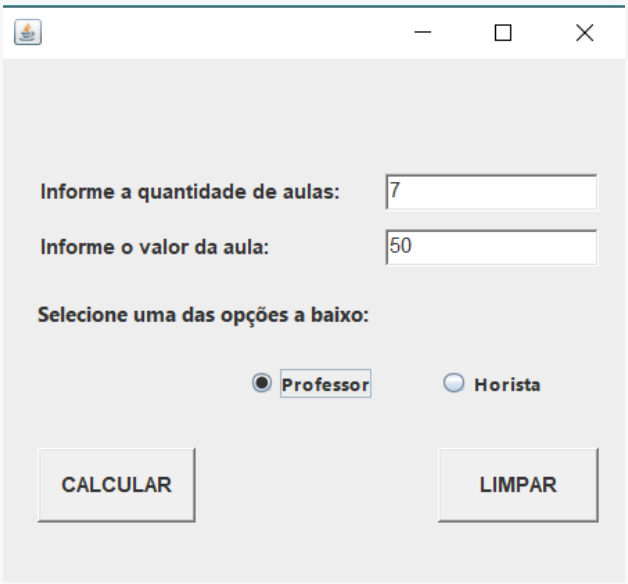
```
private void
btnCalcularActionPerformed(java.awt.event.ActionEvent
evt) {
float qtd = Float.parseFloat(txtQtd.getText());
float valor = Float.parseFloat(txtValor.getText());
float resultado;

if (rdbHorista.isSelected()) {
 resultado = qtd * valor;
} else {
resultado = (qtd * valor) * 1.25f;
}
ClasseGroovy groovy = new ClasseGroovy(resultado);
String salario = groovy.getResposta().toString();
lblSalario.setText(salario);
}
```

No botão “Limpar”, todos os campos foram zerados para uma nova consulta, ou seja, os campos voltaram a ser o que eram antes de serem utilizados.

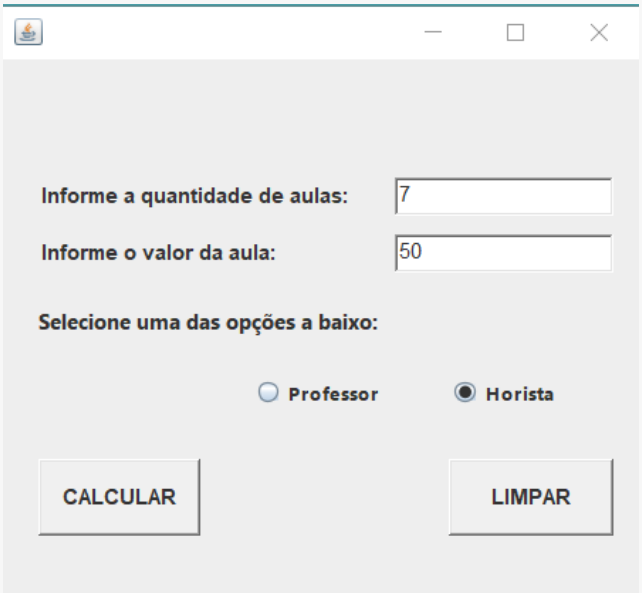
```
private void
btnLimparActionPerformed(java.awt.event.ActionEvent
evt) {
txtQtd.setText("");
txtValor.setText("");
txtQtd.requestFocus();
}
```

A seguir, está o resultado da depuração:

PROFESSOR



HORISTA



EXEMPLO 03

Neste exercício, iremos criar um formulário que, a partir da digitação de três valores do usuário, uma equação do segundo grau será calculada. Não haverá, por ora, alguma validação de digitação. Primeiramente, devemos construir a classe Groovy que fará os cálculos.

```
class Classeequacao {
 float valorA;
 float valorB;
 float valorC;

 Classeequacao (A, B, C){
 this.valorA=A;
 this.valorB=B;
 this.valorC=C;
 }

 public float calcularDelta() {
 return (valorB*valorB) - (4 * valorA * valorC)
 }
```

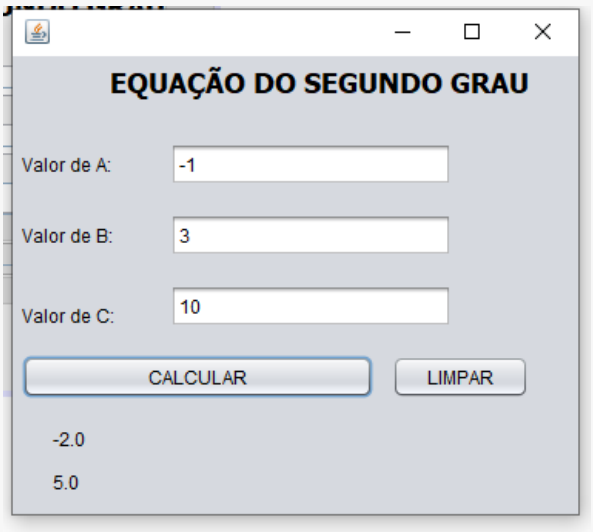
```
 public float calcularX1(){
 float delta = calcularDelta();
 return (-valorB + Math.sqrt(delta)) / (2 * valorA);
 }

 public float calcularX2(){
 float delta = calcularDelta();
 return (-valorB - Math.sqrt(delta)) / (2 * valorA);
 }
}
```

Com a classe construída, podemos partir para a construção do formulário, no qual programaremos botões para realizar o cálculo e para limpar os campos.

```
private void
jButton1ActionPerformed(java.awt.event.ActionEvent evt) {
float A = Float.parseFloat(txt_valorA.getText());
float B = Float.parseFloat(txt_valorB.getText());
float C = Float.parseFloat(txt_valorC.getText());

Classeequacao equacao = new Classeequacao(A, B, C);
float X1 = equacao.calcularX1();
loat X2 = equacao.calcularX2();
```

```
String f = String.valueOf(X1);
String g = String.valueOf(X2);

lbl_x1.setText(f);
lbl_X2.setText(g);
}

private void
jButton2ActionPerformed(java.awt.event.ActionEvent evt) {
lbl_x1.setText("");
lbl_X2.setText("");
txt_valorA.setText("");
txt_valorB.setText("");
txt_valorC.setText("");
}
```

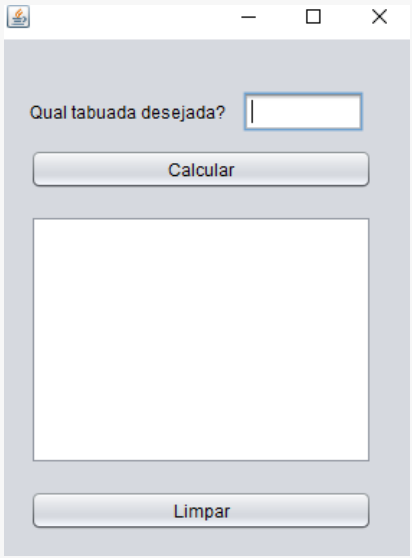
Com os botões criados, podemos executar o formulário. Veja-o abaixo:



TABUADA

EXEMPLO 04

Neste exercício, iremos construir um formulário que permita o cálculo da tabuada de um valor digitado pelo usuário. Primeiramente, deve-se construir o formulário utilizando elementos como botões e campos de textos, assim como realizado outrora nos exercícios anteriores. O resultado pode ser visualizado na imagem abaixo:



Para que a tabuada seja exibida, iremos programar um laço de repetição na classe Groovy nomeada “GroovyTabuada”. Escreva o código como o apresentado a seguir:

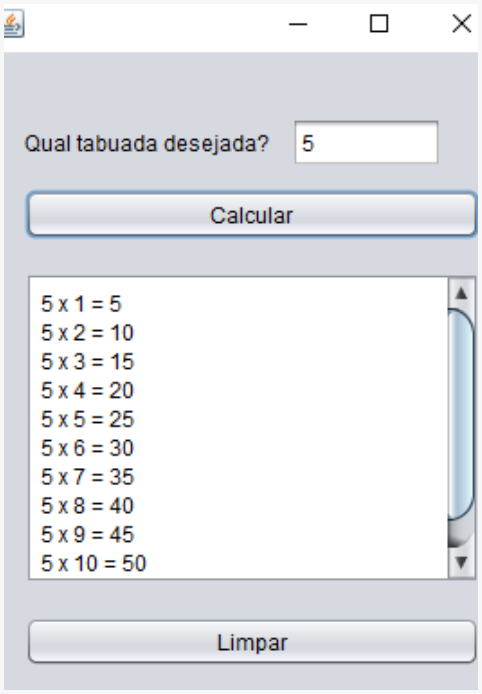
Para integrar a classe em Groovy com o código do formulário em Java, programaremos o botão “Calcular” com a seguinte sintaxe:

```
private void
btnCalcularActionPerformed(java.awt.event.ActionEvent
evt) {
int tabuada = Integer.parseInt(txtValor.getText());
GroovyTabuada groovy = new GroovyTabuada(tabuada);
String resultado = groovy.getResultado().toString();
txtTabuada.setText(resultado);
}
```

Para fins de melhor usabilidade, iremos criar um botão que limpará os campos de texto para que o usuário possa realizar a execução da aplicação com uma melhor praticidade. Programe esse botão dentro do próprio código do formulário em Java.

```
private void
bntLimparActionPerformed(java.awt.event.ActionEvent
evt) {
txtTabuada.setText("");
txtValor.setText("");
}
```

Com a programação concluída, execute o programa para testá-lo. Veja o exemplo de execução na captura de tela abaixo:

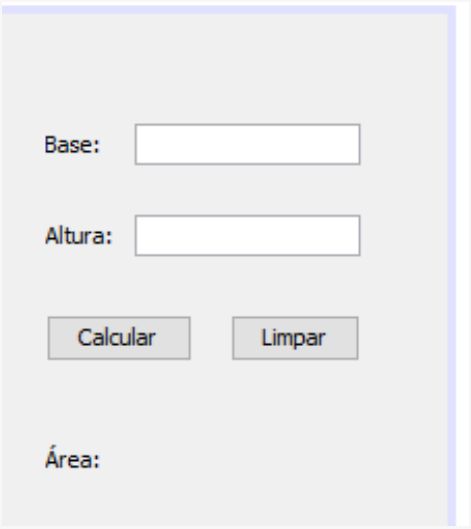


4 CAPÍTULO

PROGRAMAÇÃO ORIENTADA A OBJETO (POO)

ÁREA DO TRIÂNGULO

Neste exercício, iremos construir um programa que calcula a área de um triângulo a partir dos dados digitados pelo usuário. Para tanto, iremos confeccionar o formulário em Java que conterá os campos que interagirão com o usuário. Construa algo semelhante ao exibido abaixo:



Dentro dos botões criados, utilize a seguinte sintaxe que será responsável pela invocação dos métodos para o cálculo da área e para a limpeza dos campos.

```
private void
btnCalcularActionPerformed(java.awt.event.ActionEvent
evt) {
String valorBase = txtBase.getText();
String valorAltura = txtAltura.getText();
```

```
ClasseBLL validacao = new ClasseBLL();
String mensagem =
validacao.validaDados(valorBase, valorAltura);
if (mensagem == null){
ClasseTriangulo area = new
ClasseTriangulo(Float.parseFloat(valorBase),
Float.parseFloat(valorAltura));
lblArea.setText("Área: " +
Float.toString(area.calcularArea()));
 }
 else {
JOptionPane.showMessageDialog(null,mensagem);
 } }
 private void
btnLimparActionPerformed(java.awt.event.ActionEve
nt evt) {
txtBase.setText("");
txtAltura.setText("");
lblArea.setText("Área: ");
 }
```

Note que utilizamos a orientação a objetos a partir de uma classe que fora construída em Groovy. A seguinte classe deve ser programada da seguinte maneira:

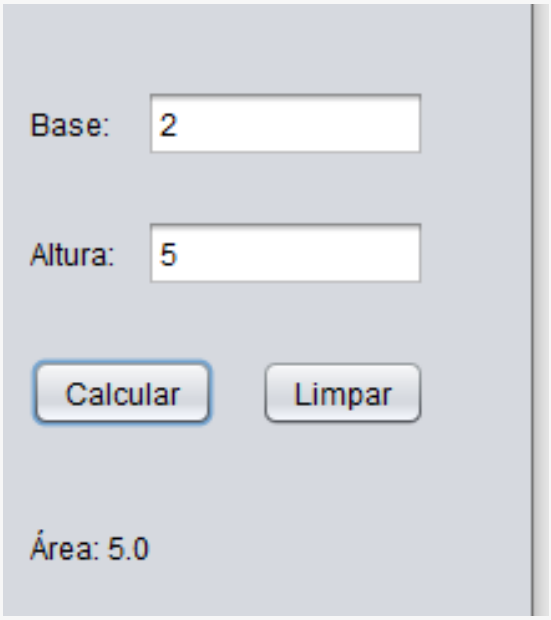
```
Class ClasseTriangulo {
 float base;
 float altura;
 ClasseTriangulo (B, A){
 this.base = B;
 this.altura = A;
 }
```

```
public float calcularArea(){
return(base * altura) / 2;
}
}
```

Em seguida, construiremos nossa classe de tratamento de erros.

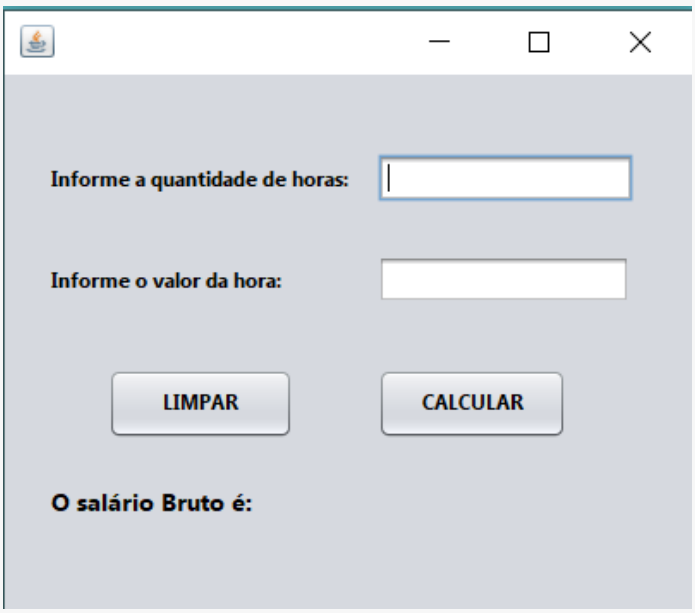
```
class ClasseBLL {
public String validaDados(String base, String altura)
{
if (base == "" || base.isNumber() == false || (base as
float) <= 0){
return "O prenchimento do valor da base está vazio,
não é numérico ou é menor que zero.";
}
if (altura == "" || altura.isNumber() == false || (altura
as float) <= 0){
return "O prenchimento do valor da altura está
vazio, não é numérico ou é menor que zero.";
}
}
}
```

Com todos esses processos finalizados, a execução do programa poderá ser realizada. Confira-a abaixo:


Base:
2
Altura:
5
Calcular
Limpar
Área: 5.0

HORISTA

Neste programa, será simulado o cálculo do salário de um horista. Ele, ao contrário do que foi apresentado no capítulo anterior, contém validação de dados.



O programa foi confeccionado separado em cinco partes. A primeira é o botão limpar e o botão calcular, como pode ser analisado a baixo:

```
private void
btnLimparActionPerformed(java.awt.event.ActionEv
ent evt) {
txtQtd.setText("");
txtValor.setText("");
lblSal.setText("");
txtQtd.requestFocus();
txtQtd.setEditable(true);
txtValor.setEditable(true);
}
```

No botão calcular, há diversas chamadas, elas são as responsáveis por chamar os códigos que não estão presentes nessa etapa de programação, como pode ser visto a seguir:

```
private void
btnCalcularActionPerformed(java.awt.event.ActionEve
nt evt) {
 String quantidade = txtQtd.getText();
 String valor = txtValor.getText();

HoristaBLL horista = new HoristaBLL();
String validar = horista.validaDados(quantidade, valor);
if (Erro.getErro()){
 JOptionPane.showMessageDialog(null,Erro.getMens())
;
 } else {
 if (validar == null) {
 Horista umhorista = new Horista();
 umhorista.setQtd(txtQtd.getText());
 umhorista.setValor(txtValor.getText());

ClasseGroovy groovy = new
ClasseGroovy(umhorista.getSalarioBruto());
 String salario = groovy.getResposta().toString();
 lblSal.setText(salario);
 }
 }
 }
```

Após isso, criou-se a classe groovy para receber os valores do formulário.

```
class ClasseGroovy {
 String resposta;
 ClasseGroovy(salario){
 this.resposta = "O salário bruto é " +salario;
 }
}
```

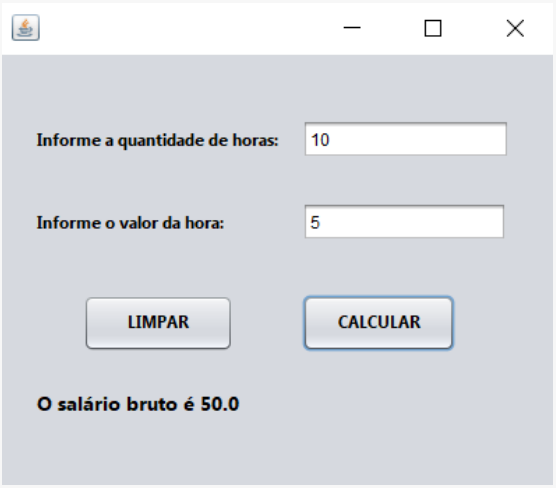
As outras TRÊS partes do projeto, que são chamadas no botão calcular, servem para melhorar o código. Devido a isso, foi criado “Erro.groovy”, “HoristaBLL.groovy” e “Horista.groovy”, pois cada uma possui uma função específica que contribui para o objetivo final do programa, como pode ser analisado a baixo:

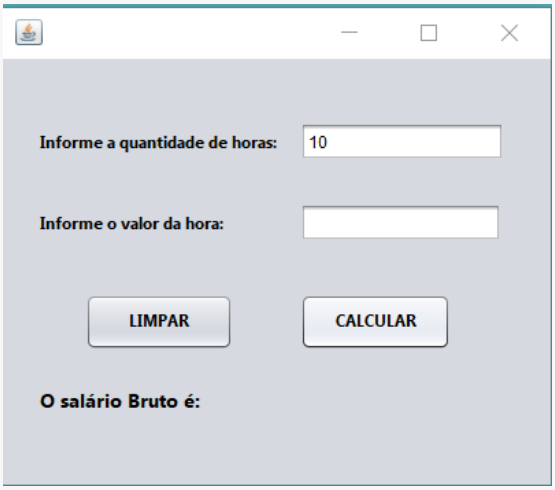
```
class Erro {
 private static boolean erro;
 private static String mens;
public static void setErro(boolean _erro) { erro = _erro; }
public static void setErro(String _mens) { erro = true; mens = _mens; }
 public static boolean getErro() { return erro; }
 public static String getMens() { return mens; }
 }
```

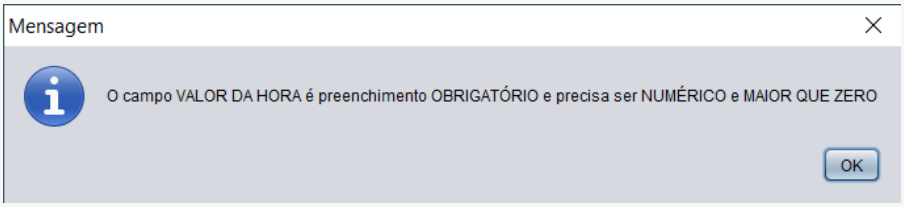
```
Class Horista {
 private String qtd;
 private String valor;

 public void setQtd(String _qtd) { qtd = _qtd; }
 public String getQtd() { return qtd; }
 public void setValor(String _valor) { valor = _valor; }
 public String getValor() { return valor; }
 public String getSalarioBruto()
 {
 return
(Float.parseFloat(getQtd())*Float.parseFloat(getValor())
).toString();
 }
}

class HoristaBLL {
 public String validaDados(String quantidade, String
valor) {
 Erro.setErro(false);

if (valor == "" || valor.isNumber() == false || (valor as
float) <= 0){
Erro.setErro("O campo VALOR DA HORA é
preenchimento OBRIGATÓRIO e precisa ser
NUMÉRICO e MAIOR QUE ZERO ");
return;
}
if (quantidade == "" || quantidade.isNumber() == false ||
(quantidade as float) <= 0){
```

```
Erro.setErro("O campo QUANTIDADE DE HORAS é
preenchimento OBRIGATÓRIO e precisa ser
NUMÉRICO e MAIOR QUE ZERO ");
return Erro.setErro(true)
}}}
```

Como resultado final, há um programa capaz de realizar o cálculo do salário de um horista com validação de dados. Confira-o a seguir:



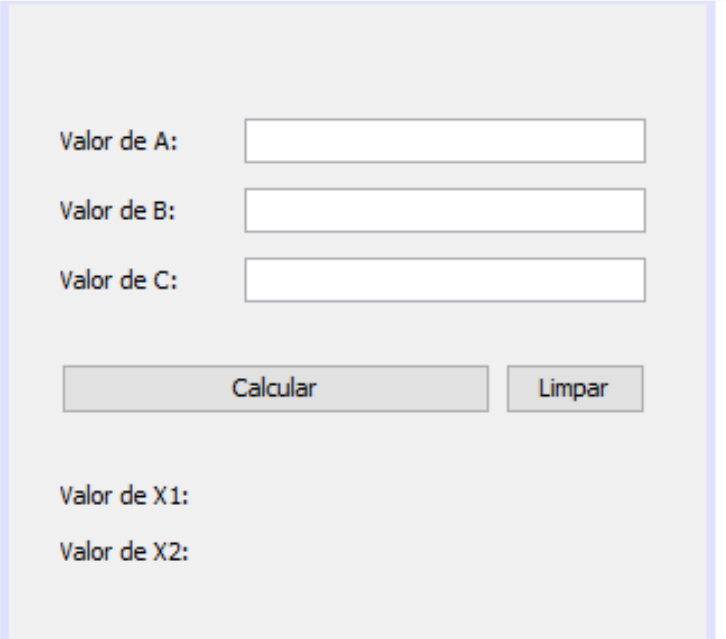




EQUAÇÃO DE 2° GRAU

Neste exercício, iremos criar um programa que, a partir da digitação de três variáveis pelo usuário, calculará as raízes da respectiva equação do segundo grau.

Primeiramente, iremos confeccionar o formulário em Java que conterá os campos. Veja-o na imagem abaixo:



Em seguida, devemos programar a classe que conterá os cálculos. Utilizando a programação orientada a objetos com Groovy, temos:

```
class ClasseEquacao {
float valorA;
float valorB;
float valorC;
```

```
ClasseEquacao (A, B, C){
this.valorA=A;
this.valorB=B;
this.valorC=C;
}
public float calcularDelta() {
return (valorB*valorB) - (4 * valorA * valorC)
}
public float calcularX1(){
float delta = calcularDelta();
return (-valorB + Math.sqrt(delta)) / (2 * valorA);
}
public float calcularX2(){
float delta = calcularDelta();
return (-valorB - Math.sqrt(delta)) / (2 * valorA);
}
}
```

Note que utilizamos um construtor, inicialmente, para retomar os parâmetros que o usuário digitará. Posteriormente, desenvolveu-se métodos que calculam o delta e as raízes da equação. Agora, utilizaremos a programação em Java para retornar esses valores e exibi-los ao usuário pelo formulário criado anteriormente.

```
class ClasseBLL {
public String validaDados(String valorA, String valorB, String valorC) {
 if (valorA == "" || valorA.isNumber() == false){
return "O prenchimento do valor de A está vazio ou não é numérico.";
 }
```

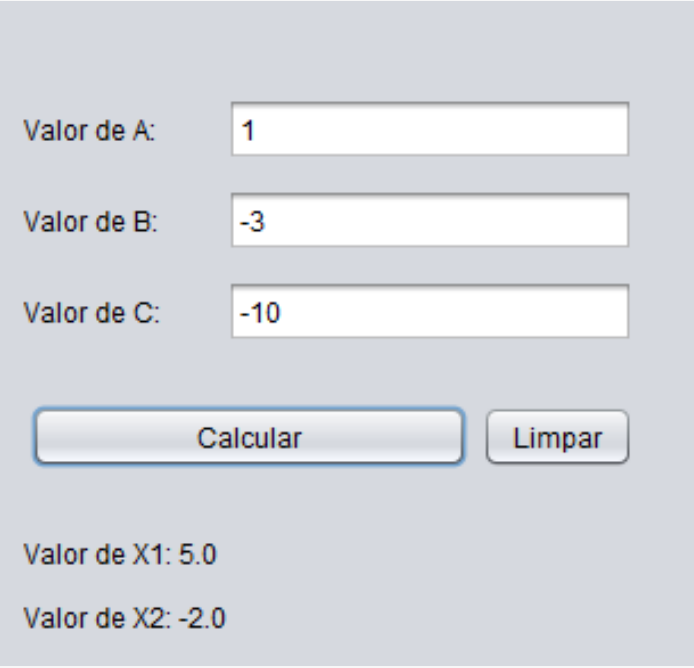
```
 if (valorB == "" || valorB.isNumber() == false){
  return "O prenchimento do valor de A está vazio ou não é numérico.";
 }
 if (valorC == "" || valorC.isNumber() == false){
  return "O prenchimento do valor de A está vazio ou não é numérico.";
 }
 }
}
```

Agora, utilizaremos a programação em Java para retornar esses valores e exibi-los ao usuário pelo formulário criado anteriormente. Programe o botão responsável pelos cálculos e pela limpeza dos campos da seguinte maneira:

```
private void btnCalcularActionPerformed(java.awt.event.ActionEvent evt) {
 String valorA = txtValorA.getText();
 String valorB = txtValorB.getText();
 String valorC = txtValorC.getText();

ClasseBLL validacao = new ClasseBLL();
String mensagem = validacao.validaDados(valorA, valorB, valorC);
 if (mensagem == null){
ClasseEquacao equacao = new ClasseEquacao(Float.parseFloat(valorA), Float.parseFloat(valorB), Float.parseFloat(valorC));
 float X1 = equacao.calcularX1();
 float X2 = equacao.calcularX2();
```

```
 lblX1.setText("Valor de X1: " + Float.toString(X1));
 lblX2.setText("Valor de X2: " +Float.toString(X2));
 }
 else {
JOptionPane.showMessageDialog(null,mensagem);
 }
 }
 private void
btnLimparActionPerformed(java.awt.event.ActionEven
t evt) {
 lblX1.setText("Valor de X1:");
 lblX2.setText("Valor de X2:");
 txtValorA.setText("");
 txtValorB.setText("");
 txtValorC.setText("");
 }
```

Com todos esses processos finalizados, o programa pode ser executado. Veja-o na seguinte imagem:



CADASTRAR LIVROS

Neste exercício, iremos criar um formulário que permita um usuário cadastrar informações de um livro e consulta-las após esse processo. Para isso, iremos construir a classe em Groovy que fará a construção das propriedades do livro.

```
class ClasseLivro {
 String codigo;
 String titulo;
 String autor;
 String editora;
 float ano;

 ClasseLivro (C, T, A, E, An){
this.codigo = C;
this.titulo = T;
this.autor = A;
 this.editora = E;
 this.ano = An;
 }
}
```

Após isso, iremos construir nossa classe que tratará dos equívocos de digitação pelo usuário.

```
class ClasseBLL {
 public String validaDados(String codigo, String titulo,
String autor, String editora, String ano) {
 if (codigo == ""){
 return "O prenchimento do código está vazio.";
 }

 if (titulo == ""){
 return "O prenchimento do título está vazio.";
 }

 if (autor == ""){
 return "O prenchimento do autor está vazio.";
 }

 if (editora == ""){
 return "O prenchimento da editora está vazio.";
 }

 if (ano == "" || ano.isNumber() == false || (ano as float)
<= 0){
  return "O prenchimento do ano está vazio, não é
numérico ou é menor que zero.";
 }
 }
}
```

Agora, podemos construir o código dos botões do formulário. Na função de arquivamento dos livros, iremos retomar o objeto criado para inicializar o construtor das propriedades dele.

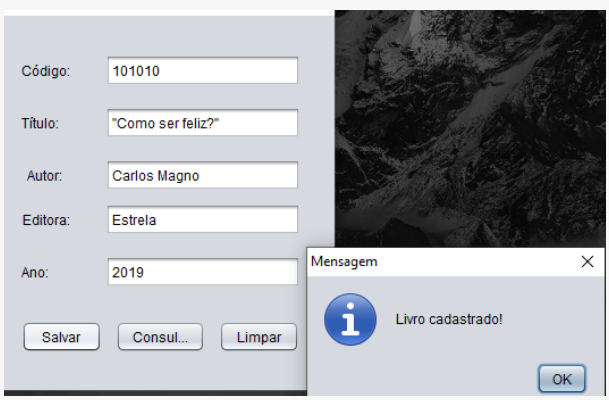
```
private void
bntSalvarActionPerformed(java.awt.event.ActionEvent
evt) {
 String codigo = txtCodigo.getText();
 String titulo = txtTitulo.getText();
 String autor = txtAutor.getText();
 String editora = txtEditora.getText();
 String ano = txtAno.getText();

ClasseBLL validacao = new ClasseBLL();
String mensagem = validacao.validaDados(codigo,
titulo, autor, editora, ano);
if (mensagem == null){
livro = new ClasseLivro(codigo, titulo, autor, editora,
Float.parseFloat(ano));
JOptionPane.showMessageDialog(this, "Livro
cadastrado!");
 }
 else {
JOptionPane.showMessageDialog(null,mensagem);
 }
 }

private void
btnConsultarActionPerformed(java.awt.event.ActionEv
ent evt) {
 String codigo = txtCodigo.getText();

 if (livro.getCodigo().equals(codigo)){
txtCodigo.setText(livro.getCodigo());
txtTitulo.setText(livro.getTitulo());
txtAutor.setText(livro.getAutor());
```

```
txtEditora.setText(livro.getEditora());
txtAno.setText(Float.toString(livro.getAno()));
 } else{
JOptionPane.showMessageDialog(null,"Livro não encontrado.");
 }
 }
```

Agora, podemos construir o código dos botões do formulário. Na função de arquivamento dos livros, iremos retomar o objeto criado para inicializar o construtor das propriedades dele.

```
 private void btnLimparActionPerformed(java.awt.event.ActionEvent evt) {
 txtCodigo.setText("");
 txtTitulo.setText("");
 txtAutor.setText("");
 txtEditora.setText("");
 txtAno.setText("");
 }
```

Com esses processos finalizados, o programa poderá ser executado. Veja-o abaixo:



5 CAPÍTULO

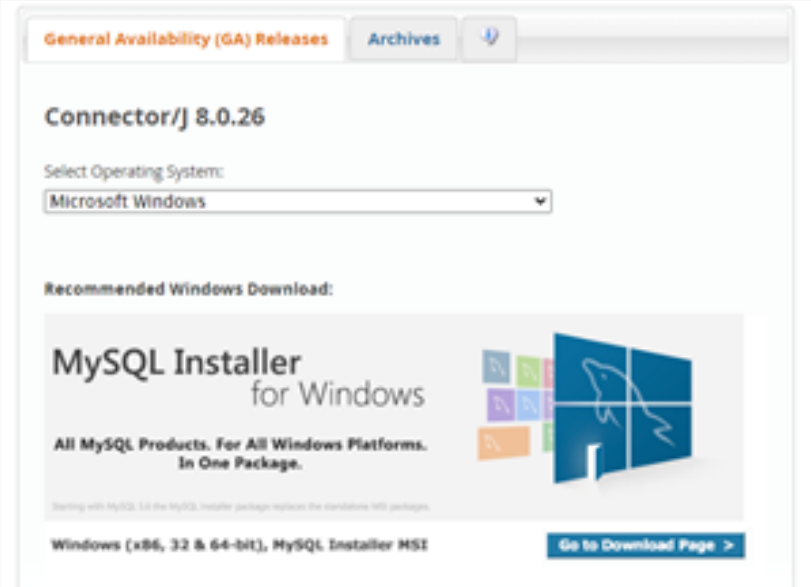
BANCO DE DADOS

CONFIGURAR O NETBEANS COM O MYSQL

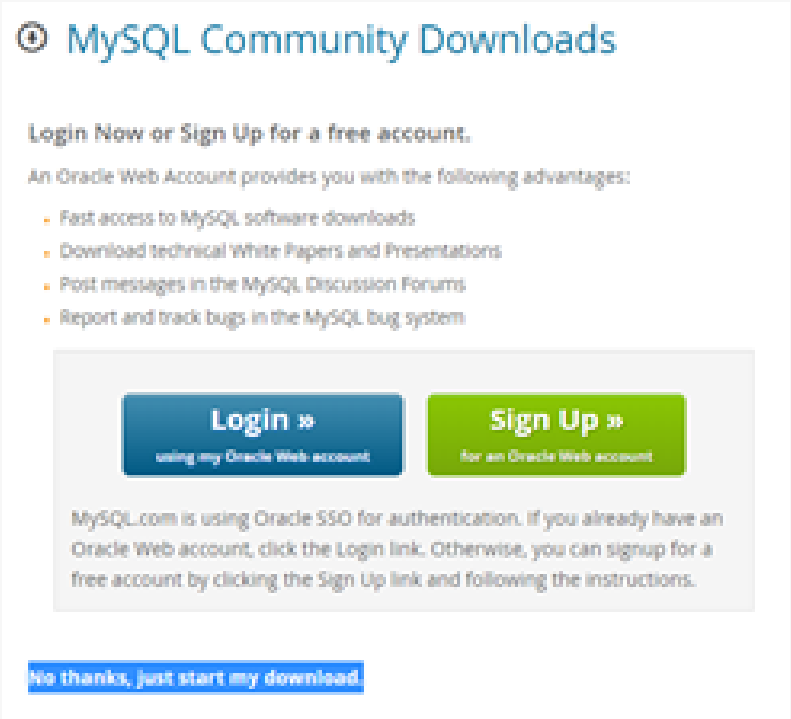
Para conectar o seu código Groovy com um banco de dados relacional, nós utilizaremos o MySQL como plataforma para a manipulação da base de dados. Para instalá-lo, siga as instruções disponíveis no endereço **<https://dicasdeprogramacao.com.br/como-instalar-o-mysql-no-windows/>**. Com a instalação completa, prosseguiremos para a configuração da IDE e do MySQL para que eles se integrem e possam funcionar em conjunto.

Inicialmente, baixe o instalador do conector do MySQL através do endereço **<https://dev.mysql.com/downloads/connector/j/>**.

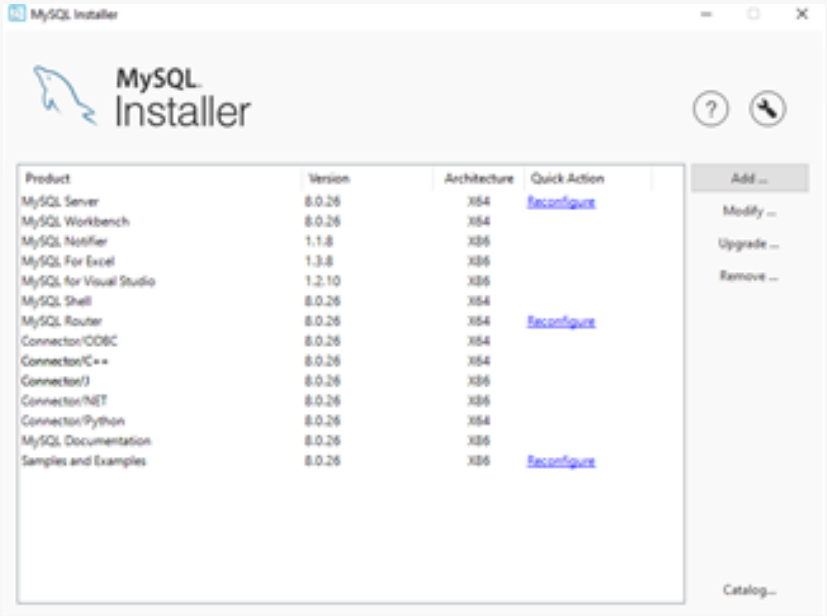
Selecione a opção do sistema Windows e pressione **“Go to Download Page”**.



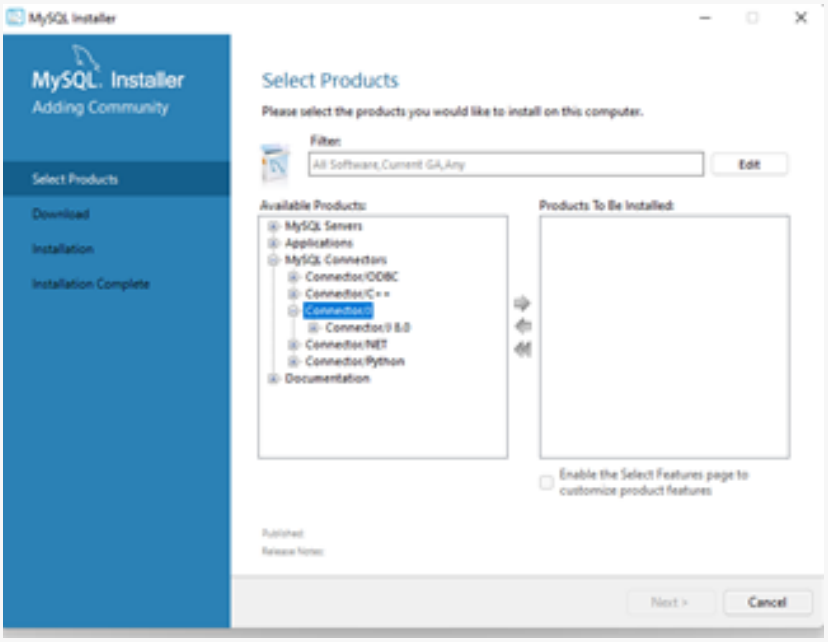
Selecione a versão do seu sistema e, quando aparecer a tela abaixo, opte por **“No thanks, just start my download”**.



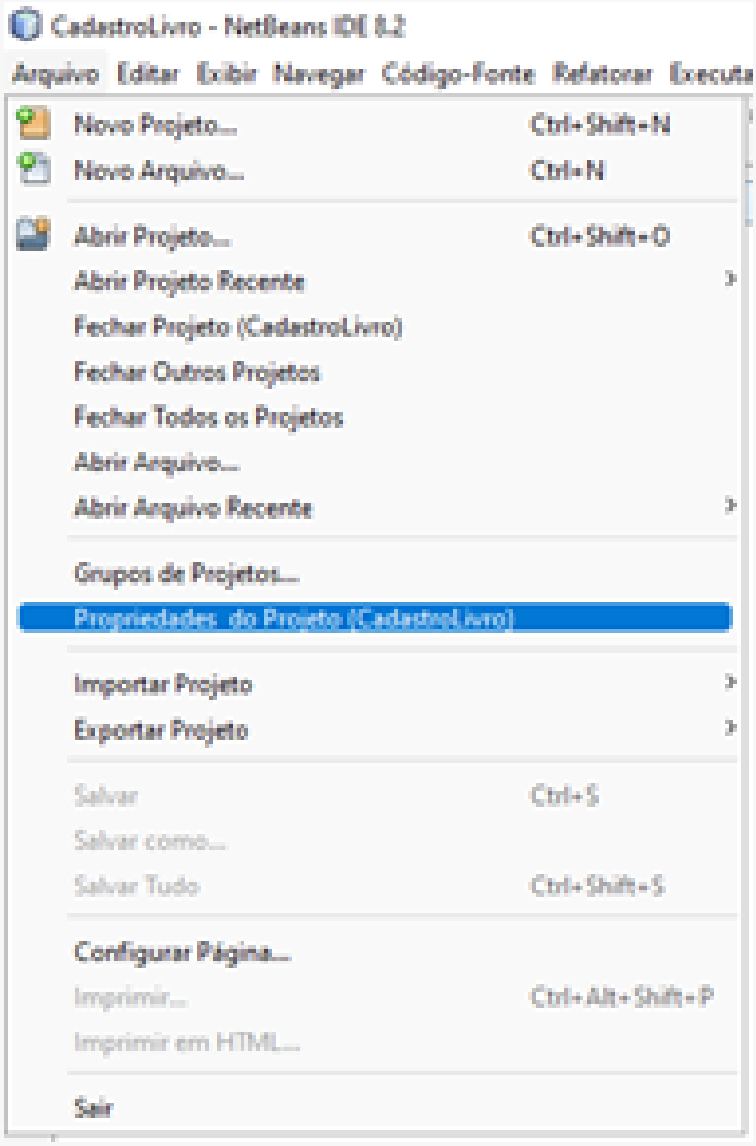
Abra o arquivo executável que fora baixado e, quando aparecer a tela abaixo, opte por **“Add”**.



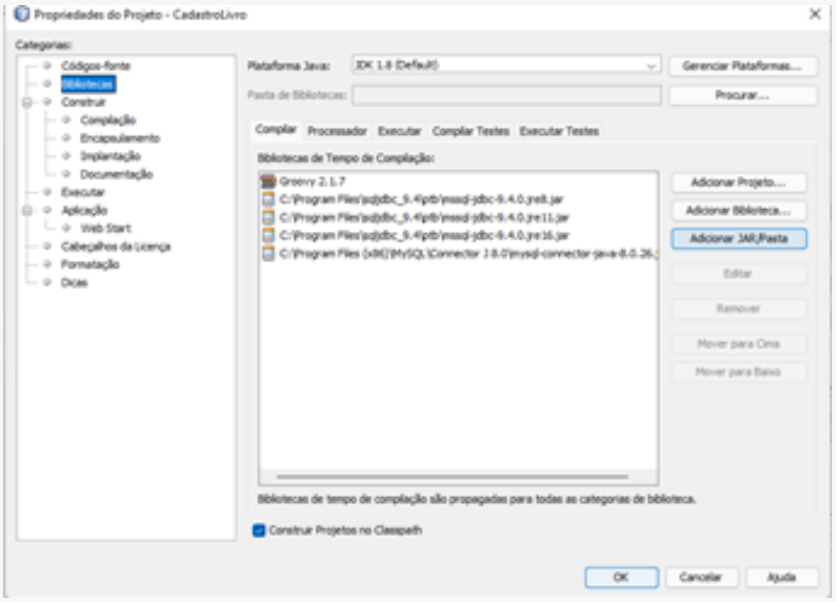
Expanda a opção **“MySQL Connectors”** e, em seguida, expanda **“Connector/J”**. Selecione a opção compatível com o seu sistema, pressione **“Next”** e siga as instruções do instalador.



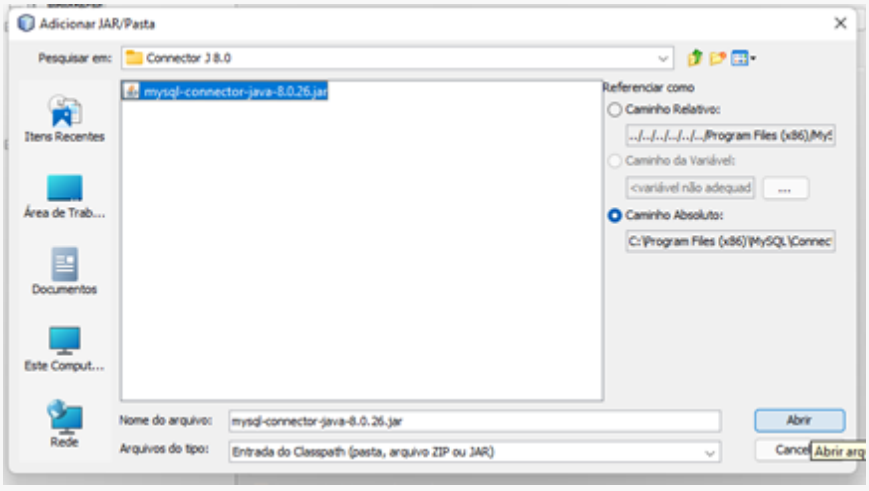
Crie o projeto no qual haverá a conexão com o banco de dados. Selecione o nome do projeto e, em seguida, pressione **“Arquivo”** e procure a opção **“Propriedades do Projeto”.**



No canto esquerdo, selecione **“Bibliotecas”** e, no canto direito, **“Adicionar JAR/Pasta”.**



Procure pelo arquivo de extensão “.jar” no seu computador. Neste tutorial, ele estava presente no diretório “C:\Program Files (x86)\MySQL\Connector J 8.0”. Selecione o conector e clique em “Abrir”.



Agora a sua IDE está apta para realizar a comunicação com uma base de dados do MySQL.

CADASTRO DE LIVROS

COM "CRUD"

Neste exercício, iremos criar uma aplicação que, por meio de um formulário, permita o usuário manipular uma base de dados através dos comandos CREATE, READ, UPDATE e DELETE (CRUD). Primeiramente, iremos abrir o MySQL para fazer a criação da base de dados e, posteriormente, analisar as mudanças feitas nela. Crie a tabela com a seguinte sintaxe:

```
CREATE DATABASE livro;
USE livro;

CREATE TABLE livros (
codigo VARCHAR(5) PRIMARY KEY,
titulo TEXT,
autor TEXT,
editora TEXT,
ano SMALLINT);

SELECT * FROM livros;
```

Utilizaremos o comando SELECT para ir consultando as alterações feitas em tempo de execução.

CADASTRO DE LIVROS

COM "CRUD"

Agora, criaremos o formulário em JAVA para o contato com o usuário, definindo e nomeando os componentes de modo que sejam exibidos da maneira abaixo:

Agora, iremos reaproveitar a classe BLL para verificação de erros utilizada no capítulo anterior. Crie uma classe Groovy chamada “ClasseBLL” e coloque a seguinte sintaxe:

```
class ClasseBLL {
      public String validaDados(String codigo, String
titulo, String autor, String editora, String ano) {
    if (codigo == ""){
      return "O prenchimento do código está vazio.";
    }
    if (titulo == ""){
      return "O prenchimento do título está vazio.";
    }
    if (autor == ""){
      return "O prenchimento do autor está vazio.";
    }
    if (editora == ""){
      return "O prenchimento da editora está vazio.";
  }
      if (ano == "" || ano.isNumber() == false || (ano as
float) <= 0){
       return "O prenchimento do ano está vazio, não é
numérico ou é menor que zero.";
    }
  }
}
```

Para realizar a comunicação com a base de dados, crie uma nova classe Groovy nomeada como “ClasseDLL”. Nela, colocaremos os procedimentos necessários para realizar os comandos CRUD. Convém ressaltar que, no procedimento “conectarBDD()”, você precisa trocar as áreas em vermelho pelos seus dados de logon definidos na instalação do MySQL.

```
import java.sql.*
import groovy.sql.Sql

class ClasseDLL {
   def sql
   def titulo
   def autor
   def editora
   def ano

 public void conectarBDD(){
sql=
Sql.newInstance("jdbc:mysql://localhost:3306/livro",
"USER", "PASSWORD", "com.mysql.jdbc.Driver")
   }
   public void desconectarBDD(){
sql.close()
   }
```

```
public void inserirLivro(String codigo, String titulo, String autor, String editora, Integer ano){
def comando = "INSERT INTO livros (codigo, titulo, autor, editora, ano) VALUES (${codigo}, ${titulo}, ${autor}, ${editora}, ${ano})"
 sql.execute(comando);
sql.commit()
 }

public void consultarLivro(String codigo){
def comando = "SELECT * FROM livros WHERE codigo = ${codigo}"
 sql.eachRow(comando){
 tp ->
 titulo = tp.titulo
 autor = tp.autor
 editora = tp.editora
 ano = tp.ano
 }
 }

 public void deletarLivro(String codigo){
 def comando = "DELETE FROM livros WHERE codigo = ${codigo}"
 sql.execute(comando);
sql.commit()
 }
```

```
public void alterarLivro(String codigo, String titulo, String autor, String editora, Integer ano){
 def comando = "UPDATE livros SET titulo=${titulo}, autor=${autor}, editora=${editora}, ano=${ano} WHERE codigo = ${codigo}"
 sql.execute(comando);
sql.commit()
 }
}
```

No código do formulário Java, crie um objeto da classe anteriormente criada abaixo da função de inicialização dos componentes.

```
public FormularioLivro() {
initComponents();
  }
  ClasseDLL livro = new ClasseDLL();
```

Configure os botões criados com os seguintes códigos. Para o botão de salvar os dados:

```
private void
bntSalvarActionPerformed(java.awt.event.ActionEvent
evt) {
 String codigo = txtCodigo.getText();
 String titulo = txtTitulo.getText();
 String autor = txtAutor.getText();
 String editora = txtEditora.getText();
 String ano = txtAno.getText();

 ClasseBLL validacao = new ClasseBLL();
  String mensagem = validacao.validaDados(codigo,
titulo, autor, editora, ano);
 if (mensagem == null){
 livro.conectarBDD();
   livro.inserirLivro(codigo, titulo, autor, editora,
Integer.parseInt(ano));
      JOptionPane.showMessageDialog(this, "Livro
cadastrado!");
 livro.desconectarBDD();
 }
 else {
JOptionPane.showMessageDialog(null,mensagem);
 }
 }
```

Para o botão de consultar os dados criados:

```
private void
btnConsultarActionPerformed(java.awt.event.ActionEv
ent evt) {
 String codigo = txtCodigo.getText();
 livro.conectarBDD();
 livro.consultarLivro(codigo);
txtTitulo.setText(livro.getTitulo().toString());
 txtAutor.setText(livro.getAutor().toString());
txtEditora.setText(livro.getEditora().toString());
txtAno.setText(livro.getAno().toString());

 }
```

Para o botão de limpar os campos do formulário:

```
private void
btnLimparActionPerformed(java.awt.event.ActionEven
t evt) {
txtCodigo.setText("");
txtTitulo.setText("");
txtAutor.setText("");
txtEditora.setText("");
      txtAno.setText("");
  }
```

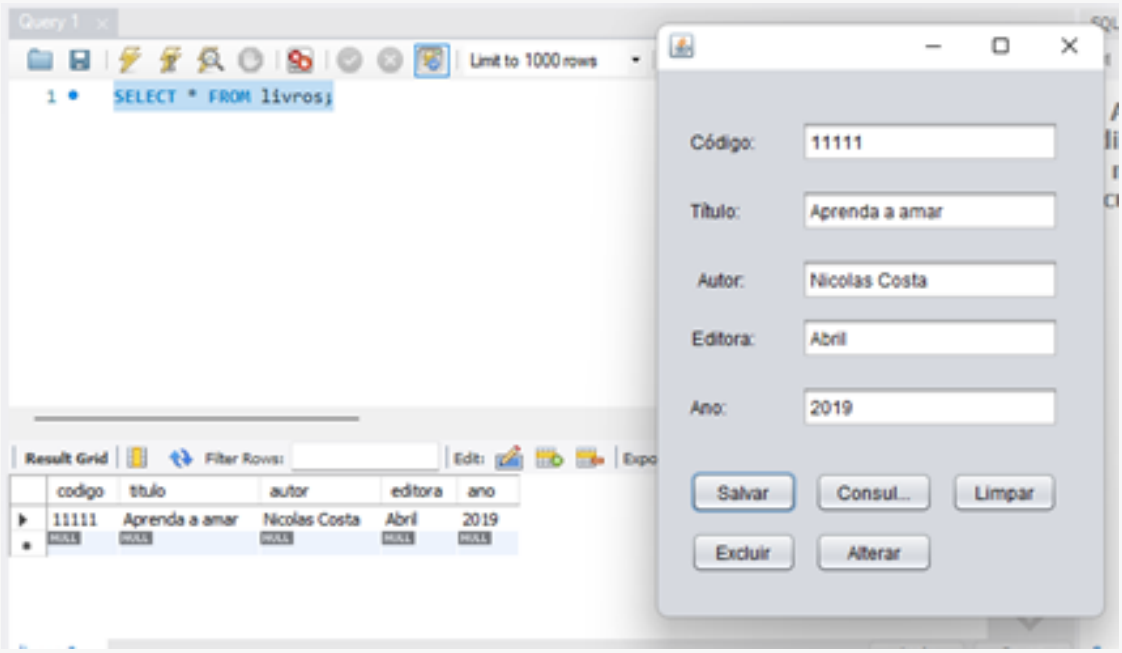
Para o botão de excluir alguma informação da base de dados:

```
private void
bntExcluirActionPerformed(java.awt.event.ActionEvent
evt) {
 String codigo = txtCodigo.getText();
 livro.conectarBDD();
 livro.deletarLivro(codigo);
 }
```

Para o botão de alterar alguma informação da base de dados:

```
private void
bntAlterarActionPerformed(java.awt.event.ActionEvent
evt) {
    String codigo = txtCodigo.getText();
    String titulo = txtTitulo.getText();
    String autor = txtAutor.getText();
    String editora = txtEditora.getText();
    String ano = txtAno.getText();
    livro.conectarBDD();
          livro.alterarLivro(codigo, titulo, autor, editora,
Integer.parseInt(ano));
  }
```

Com essas inserções finalizadas, partiremos para a execução. Posicionaremos o formulário ao lado do MySQL para que possamos acompanhar as alterações. Para o caso de salvar os dados, temos:



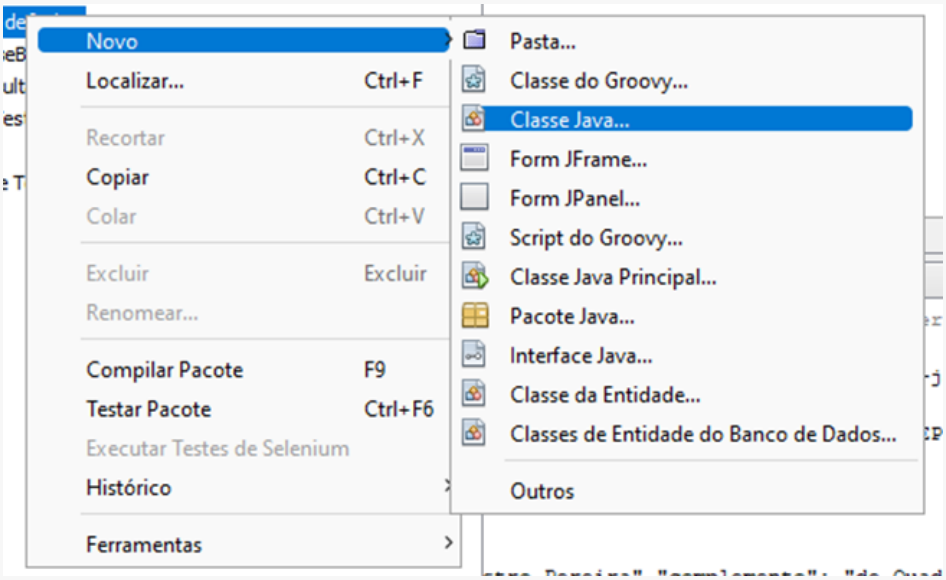
6 CAPÍTULO

API

VALIDANDO UMA API COM JAVA E GROOVY

Para trabalhar com uma API, iremos trabalhar com o Java e o Groovy em conjunto. Nesse tutorial, consultaremos as informações correspondentes a um Código de Endereço Postal (CEP) digitado pelo usuário.

Primeiramente, devemos criar uma classe em Java para poder realizar a conexão do código com os serviços da API. Dessa vez, optaremos por fazer a programação via CONSOLE. A criação do projeto é a mesma utilizada nas etapas anteriores, contudo, ao invés de acrescentarmos um formulário, inseriremos uma classe Java.



Para a validação dos dados digitados pelo usuário, iremos criar uma classe Groovy e inseriremos o seguinte código queconsiste, majoritariamente, na validação do tipo do dado digitado e o tamanho deste:

```
class ClasseBLL {
  public String validaDados(String CEP) {
    if (CEP == ""){
      return "O prenchimento do CEP está vazio.";
    }

    if (CEP.isNumber() == false){
      return "O prenchimento CEP não é numérico.";
    }

    if (CEP.length()!=8){
          return "O prenchimento do CEP está com a
quantidade de digitos incorreta.";
    }
  }
}
```

Agora, partiremos para a programação em Java. Faremos a conexão com a API de serviço online e, em seguida, criaremos um objeto da classe Groovy criada anteriormente para executar as suas funções:

```
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStream;
import java.io.InputStreamReader;
import java.net.URL;
import java.net.URLConnection;
import java.util.HashMap;
import java.util.Map;
import java.util.regex.Matcher;
import java.util.regex.Pattern;
import java.util.Scanner;

public class ConsultaCEP {
	public static String buscarCep(String cep) {
	String json;

	try {
		URL url = new URL("http://viacep.com.br/ws/"+
cep +"/json");
```

```
AURLConnection urlConnection = url.openConnection();
 InputStream is = urlConnection.getInputStream();
   BufferedReader br = new BufferedReader(new InputStreamReader(is));

 StringBuilder jsonSb = new StringBuilder();

 br.lines().forEach(l -> jsonSb.append(l.trim()));

 json = jsonSb.toString();

 } catch (Exception e) {
 throw new RuntimeException(e);
 }

 return json;
 }

 public static void main(String[] args) {
 Scanner Ler = new Scanner(System.in);

 System.out.println("Informe o CEP: ");
 String CEP = Ler.nextLine();

 ClasseBLL validar = new ClasseBLL();
 String validacao = validar.validaDados(CEP);
```

```
if (validacao == null){
 String CEPcasa = String.valueOf(CEP);
 String json = buscarCep(CEPcasa);
 System.out.println(json);

 Map<String,String> mapa = new HashMap<>();

  Matcher matcher = Pattern.compile("\"\\D.?\": \".?
\"").matcher(json);
 while (matcher.find()) {
 String[] group = matcher.group().split(":");
mapa.put(group[0].replaceAll("\"", "").trim(),
group[1].replaceAll("\"", "").trim());
 }

 System.out.println(mapa);
 }
 else{
 System.out.println(validacao);
 }
 }
}
```

Com esses processos finalizados, podemos realizar as execuções do programa. Os testes encontram-se ilustrados abaixo: